# 1. Introduction

In this project, machine learning modelling such as classification models, data understanding and data preparation, visualization and model evaluation was used to find the probabilty of the H1N1 Vaccine and seasonal flu vaccine.

# 2. Business Understanding

## 2.1 Business Problem

In the awaking of 2020, COVID-19 became a huge dilemma that took the world by storm. Millions ended up dying from the virus. Governments across the globe have started and have already distributed the COVID-19 vaccines.

This project will revisit the public health response to a different recent major respiratory disease pandemic and try predict the probability of people taking the COVID-19 vaccine.

In early 2009, Swine Flu (H1N1 virus) became a pandemic and it swept the world and around 150,000 - 500,000 deaths were recorded globally.

In October 2009, a vaccine against the H1N1 flu virus became widely available. The National 2009 H1N1 Flu Survey was conducted in the United States in late 2009 and early 2010. This phone survey asked respondents if they had received the H1N1 and seasonal flu vaccines, as

well as personal questions. These additional questions focused on their social, economic, and demographic backgrounds, perspectives on illness risks and vaccine effectiveness, and behaviors aimed at preventing transmission. A better understanding of how these traits are related to personal vaccination patterns can help guide future public health efforts.

## 2.2 Aim

The health sector has had a lot of pressure in dealing with the pandemics and also the governments have struggled with the distribution of the vaccined created by the scientists. The Aim of the project is to find the probabilities of people to get vaccinated with the COVID-19 vaccine through the analysis of the Swine Flu vaccination data.

## 2.3 Main Objective

The main objective of the project is to forecast how likely people are to receive H1N1 and seasonal flu vaccines. You will specifically predict two probabilities: one for h1n1 vaccine and one for seasonal vaccine. Also how social, economic, and demographic backgrounds, perspectives on illness risks and vaccine effectiveness, and behaviors affect the intake of the vaccines.

## 2.4 Metrics of Success

The success of this project will be measured using the accuracy score, recall score and precision score.
Also in the competition once the submission is done its accuracy is calculated.

# 3. Data Understanding

The data used in this project was gotten from a competition in Driven Data (https://www.drivendata.org/competitions/66/flu-shot-learning/page/211/) website.

## 3.1 Labels

There are two target variables for this competition:

**Observation:**

h1n1 vaccine - Indicates whether the respondent received an H1N1 flu vaccine.

seasonal vaccine - Indicates whether the respondent received the seasonal flu vaccine.

Both are binary variables, with 0 indicating no and 1 indicating yes.
Some respondents did not receive either vaccine, while others received only one or both.
This is written as a multilabel (rather than a multiclass) problem.

## 3.2 Features

You've been given a dataset with 36 columns.
Respondent id is a unique and random identifier in the first column.
The remaining 35 features are discussed further below.

For all binary variables, **0 equals No and 1 equals Yes.**

- h1n1_concern - Level of concern about the H1N1 flu.

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

- h1n1_knowledge - Level of knowledge about H1N1 flu.

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

- behavioral_antiviral_meds - Has taken antiviral medications. (binary)
- behavioral_avoidance - Has avoided close contact with others with flu-like symptoms. (binary))
- behavioral_face_mask - Has bought a face mask. (binary))
- behavioral_wash_hands - Has frequently washed hands or used hand sanitizer. (binary))
- behavioral_large_gatherings - Has reduced time at large gatherings. (binary))
- behavioral_outside_home - Has reduced contact with people outside of own household. (binary))
- behavioral_touch_face - Has avoided touching eyes, nose, or mouth. (binary))
- doctor_recc_h1n1 - H1N1 flu vaccine was recommended by doctor. (binary))
- doctor_recc_seasonal - Seasonal flu vaccine was recommended by doctor. (binary))
- chronic_med_condition - Has any of the following chronic medical conditions: asthma or an other lung condition, diabetes, a heart condition, a kidney condition, sickle cell anemia or other anemia, a neurological or neuromuscular condition, a liver condition,

or a weakened immune system caused by a chronic illness or by medicines taken for a chronic illness. (binary)
- child_under_6_months - Has regular close contact with a child under the age of six months. (binary)
- health_worker - Is a healthcare worker. (binary)

- health_insurance - Has health insurance. (binary)

- opinion_h1n1_vacc_effective - Respondent's opinion about H1N1 vaccine effectiveness.

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

- opinion_h1n1_risk - Respondent's opinion about risk of getting sick with H1N1 flu without vaccine.
  *1 = Very Low; 2 = Somewhat low; 3 = Don't know; 4 = Somewhat high; 5 = Very high.*

- opinion_h1n1_sick_from_vacc - Respondent's worry of getting sick from taking H1N1 vaccine.
  *1 = Not at all worried; 2 = Not very worried; 3 = Don't know; 4 = Somewhat worried; 5 = Very worried.*

- opinion_seas_vacc_effective - Respondent's opinion about seasonal flu vaccine effectiveness.
  *1 = Not at all effective; 2 = Not very effective; 3 = Don't know; 4 = Somewhat effective; 5 = Very effective.*

- opinion_seas_risk - Respondent's opinion about risk of getting sick with seasonal flu without vaccine.
  *1 = Very Low; 2 = Somewhat low; 3 = Don't know; 4 = Somewhat high; 5 = Very high.._*

- opinion_seas_sick_from_vacc - Respondent's worry of getting sick from taking seasonal flu vaccine.
  *1 = Not at all worried; 2 = Not very worried; 3 = Don't know; 4 = Somewhat worried; 5 = Very worried.._*

- age_group - Age group of respondent.
- education - Self-reported education level.
- race - Race of respondent.
- sex - Sex of respondent.
- income_poverty - Household annual income of respondent with respect to 2008 Census poverty thresholds.
- marital_status - Marital status of respondent.
- rent_or_own - Housing situation of respondent.
- employment_status - Employment status of respondent.
- hhs_geo_region - Respondent's residence using a 10-region geographic classification

defined by the U.S. Dept. of Health and Human Services. Values are represented as short random character strings.
- census_msa - Respondent's residence within metropolitan statistical areas (MSA) as defined by the U.S. Census.
- household_adults - Number of other adults in household, top-coded to 3.
- household_children - Number of children in household, top-coded to 3.
- employment_industry - Type of industry respondent is employed in. Values are represented as short random character strings.
- employment_occupation - Type of occupation of respondent. Values are represented as short random character strings.

# 4. Requirements

1. Data Preparation

**Observation:**

- Loading Libraries
- Loading data
- Descriptive Exploration
- Data Cleaning
- Exploratory Descriptive Analysis (EDA)
- Pre-processing Data.

**Observation:**

2. Modelling

- Train test split
- Baseline Model
- Random Forest
- Decision Tree

3. Conclusion
4. Recommendation

# 5. Loading the Data

## 5.1 Loading the Relevant Libraries

In [104]:
```
#Importing libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter("ignore")
import numpy as np
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,
from sklearn.metrics import log_loss, accuracy_score, roc_curve, auc
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier, Ra
from sklearn.model_selection import cross_val_score, RandomizedSearchCV, t
from sklearn.metrics import accuracy_score, precision_score, f1_score, rec
```

# 5.2 Loading the Dataset

## 1. Training Features

In [27]:
```
#importing and parsing the training_set_features  dataset
training_features = pd.read_csv('training_set_features.csv')
training_features.head(3)
```

Out[27]:

| | respondent_id | h1n1_concern | h1n1_knowledge | behavioral_antiviral_meds | behavioral_avoidanc |
|---|---|---|---|---|---|
| **0** | 0 | 1.0 | 0.0 | 0.0 | 0 |
| **1** | 1 | 3.0 | 2.0 | 0.0 | 1 |
| **2** | 2 | 1.0 | 1.0 | 0.0 | 1 |

3 rows × 36 columns

In [28]:
```
#Looking at the shape
training_features.shape
```

Out[28]: (26707, 36)

In [29]:
```
1 #describing the dataset
2 training_features.describe()
```

Out[29]:

| | respondent_id | h1n1_concern | h1n1_knowledge | behavioral_antiviral_meds | behavioral_avoi |
|---|---|---|---|---|---|
| **count** | 26707.000000 | 26615.000000 | 26591.000000 | 26636.000000 | 26499.0 |
| **mean** | 13353.000000 | 1.618486 | 1.262532 | 0.048844 | 0.7 |
| **std** | 7709.791156 | 0.910311 | 0.618149 | 0.215545 | 0.4 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **25%** | 6676.500000 | 1.000000 | 1.000000 | 0.000000 | 0.0 |
| **50%** | 13353.000000 | 2.000000 | 1.000000 | 0.000000 | 1.0 |
| **75%** | 20029.500000 | 2.000000 | 2.000000 | 0.000000 | 1.0 |
| **max** | 26706.000000 | 3.000000 | 2.000000 | 1.000000 | 1.0 |

8 rows × 24 columns

In [30]:
```
1 #looking at the missing values
2 training_features.isna().sum()
```

Out[30]:
```
respondent_id                    0
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
dtype: int64
```

In [31]:
```
1 #checking for duplicated values
2 training_features.duplicated().sum()
```

Out[31]:
```
0
```

## Dealing with the missing values

In [32]:

```
# Dealing with the numerical columns
binary_1 = ['h1n1_concern',
            'h1n1_knowledge',
            'behavioral_antiviral_meds',
            'behavioral_avoidance',
            'behavioral_face_mask',
            'behavioral_wash_hands',
            'behavioral_large_gatherings',
            'behavioral_outside_home',
            'behavioral_touch_face',
            'doctor_recc_h1n1',
            'doctor_recc_seasonal',
            'chronic_med_condition',
            'child_under_6_months',
            'health_worker',
            'health_insurance',
            'household_adults',
            'household_children',
            'opinion_h1n1_vacc_effective',
            'opinion_h1n1_risk',
            'opinion_h1n1_sick_from_vacc',
            'opinion_seas_vacc_effective',
            'opinion_seas_risk', 'opinion_seas_sick_from_vacc']

imputer = SimpleImputer(strategy ='mean')

imputer = imputer.fit(training_features[binary_1])
training_features[binary_1] = imputer.transform(training_features[binary_1
```

In [33]:

```
#dealing with missing strings
household = [
    'income_poverty',
    'marital_status',
    'rent_or_own',
    'employment_status',
    'hhs_geo_region',
    'education'
]
employment = [
    'employment_industry',
    'employment_occupation',
    'income_poverty'
]

training_features[household] = training_features[household].fillna('N/A')
training_features[employment] = training_features[employment].fillna('N/A'
```

In [34]:
```
1 #checking for missing values
2 training_features.isna().sum()
```

Out[34]:
```
respondent_id                0
h1n1_concern                 0
h1n1_knowledge               0
behavioral_antiviral_meds    0
behavioral_avoidance         0
behavioral_face_mask         0
behavioral_wash_hands        0
behavioral_large_gatherings  0
behavioral_outside_home      0
behavioral_touch_face        0
doctor_recc_h1n1             0
doctor_recc_seasonal         0
chronic_med_condition        0
child_under_6_months         0
health_worker                0
health_insurance             0
opinion_h1n1_vacc_effective  0
opinion_h1n1_risk            0
opinion_h1n1_sick_from_vacc  0
opinion_seas_vacc_effective  0
opinion_seas_risk            0
opinion_seas_sick_from_vacc  0
age_group                    0
education                    0
race                         0
sex                          0
income_poverty               0
marital_status               0
rent_or_own                  0
employment_status            0
hhs_geo_region               0
census_msa                   0
household_adults             0
household_children           0
employment_industry          0
employment_occupation        0
dtype: int64
```

## 2. training labels

In [35]:
```
#importing and parsing the training_set_labels  dataset
training_labels = pd.read_csv('training_set_labels.csv')
training_labels
```

Out[35]:

| | respondent_id | h1n1_vaccine | seasonal_vaccine |
|---|---|---|---|
| **0** | 0 | 0 | 0 |
| **1** | 1 | 0 | 1 |
| **2** | 2 | 0 | 0 |
| **3** | 3 | 0 | 1 |
| **4** | 4 | 0 | 0 |
| **...** | ... | ... | ... |
| **26702** | 26702 | 0 | 0 |
| **26703** | 26703 | 0 | 0 |
| **26704** | 26704 | 0 | 1 |
| **26705** | 26705 | 0 | 0 |
| **26706** | 26706 | 0 | 0 |

26707 rows × 3 columns

In [36]:
```
#Looking at the shape
training_labels.shape
```

Out[36]: (26707, 3)

In [37]:
```
#describing the dataset
training_labels.describe()
```

Out[37]:

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

In [38]:
```
#checking for duplicated values
training_labels.duplicated().sum()
```

Out[38]: 0

In [39]:
```
#checking for missing values
training_labels.isna().sum()
```

Out[39]:
```
respondent_id       0
h1n1_vaccine        0
seasonal_vaccine    0
dtype: int64
```

## 3. Test features

In [40]:
```
#importing and parsing the test_set_features  dataset
test_features = pd.read_csv('test_set_features.csv')
test_features
```

Out[40]:

| | respondent_id | h1n1_concern | h1n1_knowledge | behavioral_antiviral_meds | behavioral_avoi |
|---|---|---|---|---|---|
| **0** | 26707 | 2.0 | 2.0 | 0.0 | |
| **1** | 26708 | 1.0 | 1.0 | 0.0 | |
| **2** | 26709 | 2.0 | 2.0 | 0.0 | |
| **3** | 26710 | 1.0 | 1.0 | 0.0 | |
| **4** | 26711 | 3.0 | 1.0 | 1.0 | |
| **...** | ... | ... | ... | ... | |
| **26703** | 53410 | 1.0 | 1.0 | 0.0 | |
| **26704** | 53411 | 3.0 | 1.0 | 0.0 | |
| **26705** | 53412 | 0.0 | 1.0 | 0.0 | |
| **26706** | 53413 | 3.0 | 1.0 | 0.0 | |
| **26707** | 53414 | 2.0 | 1.0 | 0.0 | |

26708 rows × 36 columns

In [41]:
```
#Looking at the shape
test_features.shape
```

Out[41]:
```
(26708, 36)
```

In [42]:
```
#describing the dataset
test_features.describe()
```

Out[42]:

| | respondent_id | h1n1_concern | h1n1_knowledge | behavioral_antiviral_meds | behavioral_avoi |
|---|---|---|---|---|---|
| **count** | 26708.000000 | 26623.000000 | 26586.000000 | 26629.000000 | 26495.0 |
| **mean** | 40060.500000 | 1.623145 | 1.266042 | 0.049645 | 0.7 |
| **std** | 7710.079831 | 0.902755 | 0.615617 | 0.217215 | 0.4 |
| **min** | 26707.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **25%** | 33383.750000 | 1.000000 | 1.000000 | 0.000000 | 0.0 |
| **50%** | 40060.500000 | 2.000000 | 1.000000 | 0.000000 | 1.0 |
| **75%** | 46737.250000 | 2.000000 | 2.000000 | 0.000000 | 1.0 |
| **max** | 53414.000000 | 3.000000 | 2.000000 | 1.000000 | 1.0 |

8 rows × 24 columns

In [43]:
```
#checking for duplicated values
test_features.duplicated().sum()
```

Out[43]:
```
0
```

In [44]:
```
#checking for missing values
test_features.isna().sum()
```

Out[44]:
```
respondent_id                    0
h1n1_concern                    85
h1n1_knowledge                 122
behavioral_antiviral_meds       79
behavioral_avoidance           213
behavioral_face_mask            19
behavioral_wash_hands           40
behavioral_large_gatherings     72
behavioral_outside_home         82
behavioral_touch_face          128
doctor_recc_h1n1              2160
doctor_recc_seasonal          2160
chronic_med_condition          932
child_under_6_months           813
health_worker                  789
health_insurance             12228
opinion_h1n1_vacc_effective    398
opinion_h1n1_risk              380
opinion_h1n1_sick_from_vacc    375
opinion_seas_vacc_effective    452
opinion_seas_risk              499
opinion_seas_sick_from_vacc    521
age_group                        0
education                     1407
race                             0
sex                              0
income_poverty                4497
marital_status                1442
rent_or_own                   2036
employment_status             1471
hhs_geo_region                   0
census_msa                       0
household_adults               225
household_children             225
employment_industry          13275
employment_occupation        13426
dtype: int64
```

## Dealing with the missing values

In [47]:
```
#Dealing with the numeric columns
imputer1 = SimpleImputer(strategy ='mean')

imputer1 = imputer1.fit(test_features[binary_1])
test_features[binary_1] = imputer1.transform(test_features[binary_1])
```

In [48]:
```
#dealing with missing strings
test_features[household] = test_features[household].fillna('N/A')
test_features[employment] = test_features[employment].fillna('N/A')
```

In [49]:
```
#checking missing values
test_features.isna().sum()
```

Out[49]:
```
respondent_id                0
h1n1_concern                 0
h1n1_knowledge               0
behavioral_antiviral_meds    0
behavioral_avoidance         0
behavioral_face_mask         0
behavioral_wash_hands        0
behavioral_large_gatherings  0
behavioral_outside_home      0
behavioral_touch_face        0
doctor_recc_h1n1             0
doctor_recc_seasonal         0
chronic_med_condition        0
child_under_6_months         0
health_worker                0
health_insurance             0
opinion_h1n1_vacc_effective  0
opinion_h1n1_risk            0
opinion_h1n1_sick_from_vacc  0
opinion_seas_vacc_effective  0
opinion_seas_risk            0
opinion_seas_sick_from_vacc  0
age_group                    0
education                    0
race                         0
sex                          0
income_poverty               0
marital_status               0
rent_or_own                  0
employment_status            0
hhs_geo_region               0
census_msa                   0
household_adults             0
household_children           0
employment_industry          0
employment_occupation        0
dtype: int64
```

# 6. Explanatory Data Analysis, EDA

In trying to understand our data better, let's check for the distribution of the individual features in our dataset.

Answer a few questions:

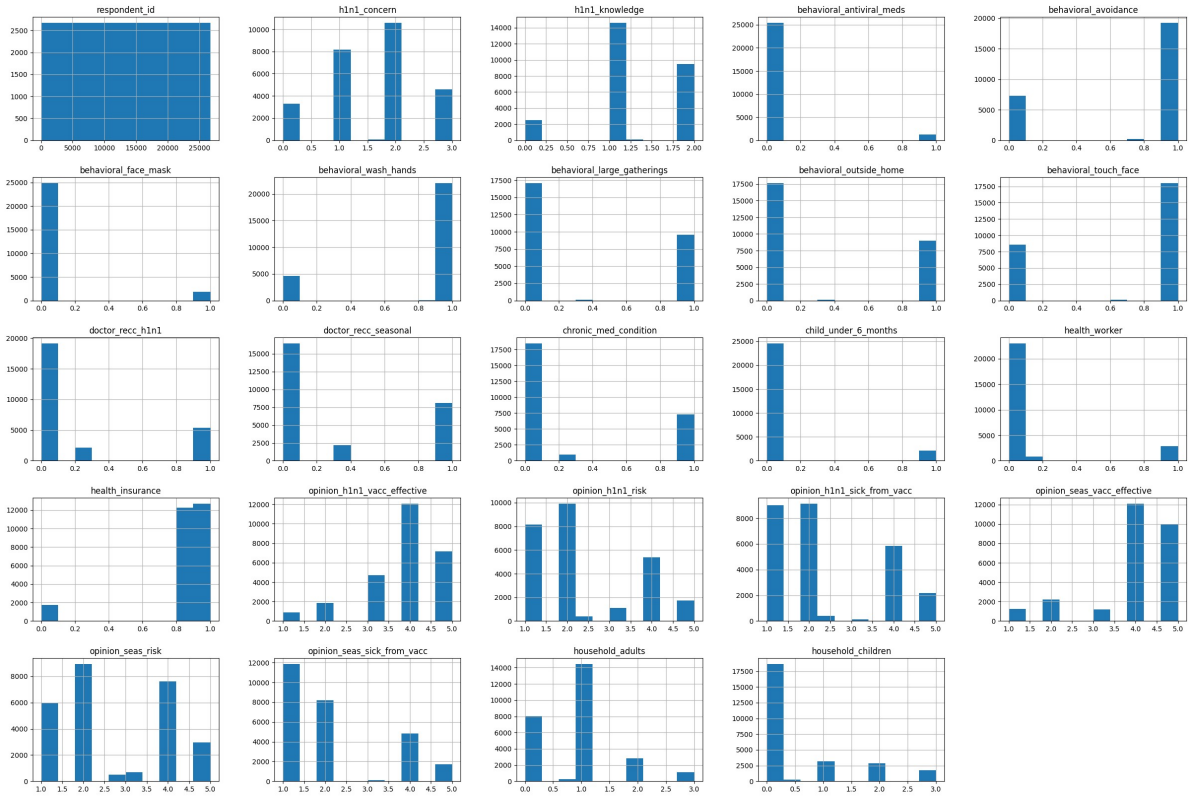
**Observation:**

- Distribution of data.
- Does sex has affected the distribution of the vaccine?

- Does race affect the distribution of the vaccine?
- Does Education level affect the distribution of the vaccine?
- Does Location affect the distribution of the vaccine?
- Does Homeownership affect the distribution of the vaccine
- How the chronic illnesses has affected the distribution of the vaccine?
- Does Age group affect the distribution of the vaccine?
- Correlation of the data.

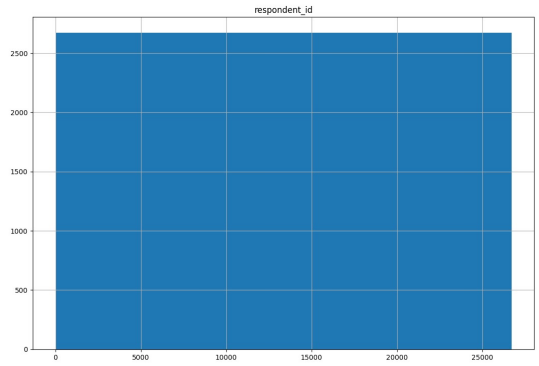
# 6.1 Distribution of Data.

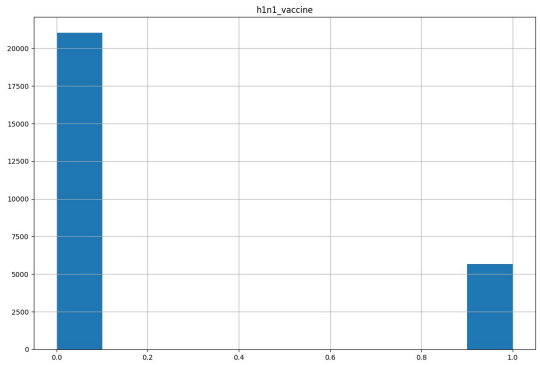
In [50]:
```
#Checking for Distribution of Data in the training features dataset
training_features.hist(figsize=(30,20))
plt.show();
```

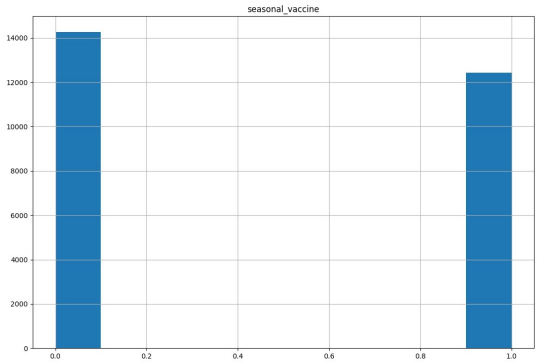



From the histoplot above, it would be clear to say that most data are skwed.

In [51]:
```
#Distribution_of data in the test features dataset
#Checking for Distribution of Data
training_labels.hist(figsize=(30,20))
plt.show();
```



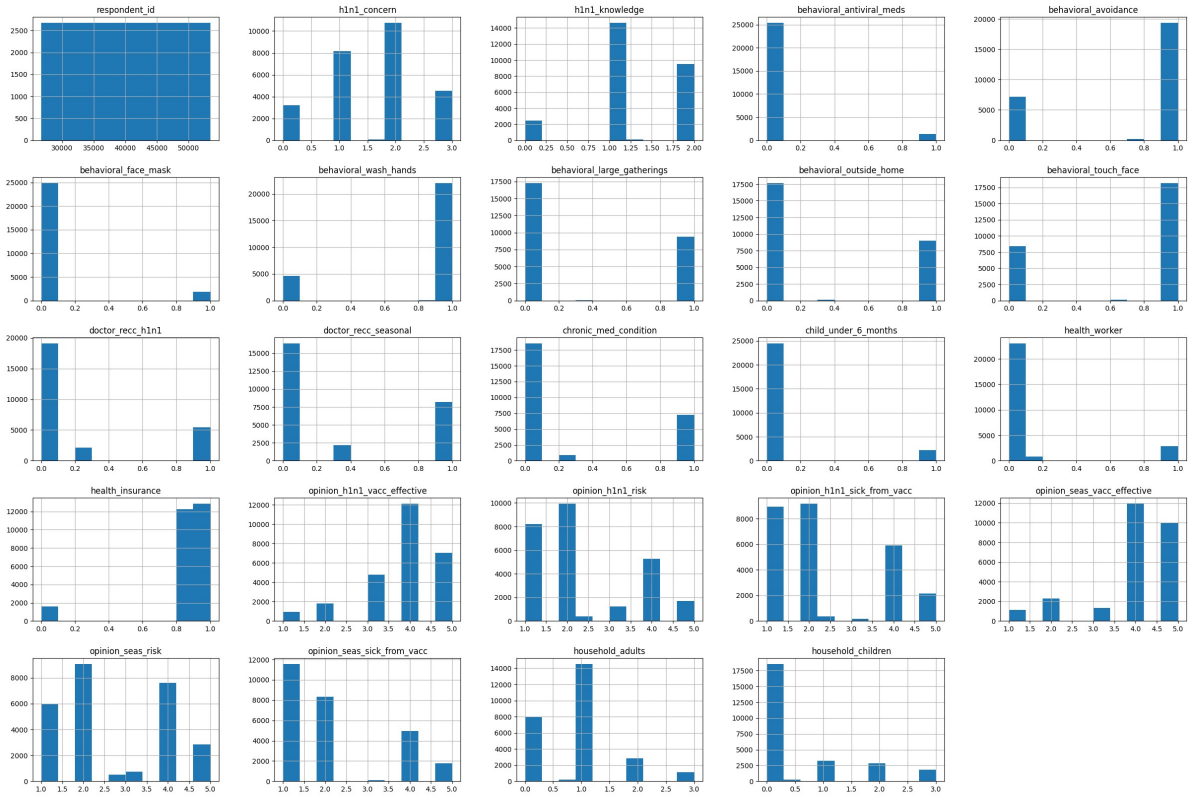




**Observation:**

In this graph, both the h1n1_vaccine and seasonal_vaccine column have more people not taking the vaccines.

In [52]:
```
#Distribution_of data in the test fea
#Checking for Distribution of Data
test_features.hist(figsize=(30,20))
plt.show();
```




**Observation:**

From the histoplot above, it would be clear to say that most data are skwed.

## Merging of the datasets for more plots

In [53]:

```
#Merging the training labes dataset and the training features dataset

#setting new index
training_labels.set_index('respondent_id', inplace=True, drop=False)
training_features.set_index('respondent_id', inplace=True, drop=False)

df = pd.merge(training_labels, training_features, left_index=True, right_i
df.head(3)
```

Out[53]:

| respondent_id | respondent_id_x | h1n1_vaccine | seasonal_vaccine | respondent_id_y | h1n1_concern |
|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 1.0 |
| **1** | 1 | 0 | 1 | 1 | 3.0 |
| **2** | 2 | 0 | 0 | 2 | 1.0 |

3 rows × 39 columns

In [54]:
```
1 #checking for missing values
2 df.isna().sum()
3
```

Out[54]:
```
respondent_id_x              0
h1n1_vaccine                 0
seasonal_vaccine             0
respondent_id_y              0
h1n1_concern                 0
h1n1_knowledge               0
behavioral_antiviral_meds    0
behavioral_avoidance         0
behavioral_face_mask         0
behavioral_wash_hands        0
behavioral_large_gatherings  0
behavioral_outside_home      0
behavioral_touch_face        0
doctor_recc_h1n1             0
doctor_recc_seasonal         0
chronic_med_condition        0
child_under_6_months         0
health_worker                0
health_insurance             0
opinion_h1n1_vacc_effective  0
opinion_h1n1_risk            0
opinion_h1n1_sick_from_vacc  0
opinion_seas_vacc_effective  0
opinion_seas_risk            0
opinion_seas_sick_from_vacc  0
age_group                    0
education                    0
race                         0
sex                          0
income_poverty               0
marital_status               0
rent_or_own                  0
employment_status            0
hhs_geo_region               0
census_msa                   0
household_adults             0
household_children           0
employment_industry          0
employment_occupation        0
dtype: int64
```

**Observation:**

There arent any missing values after joinging/merging the datasets.

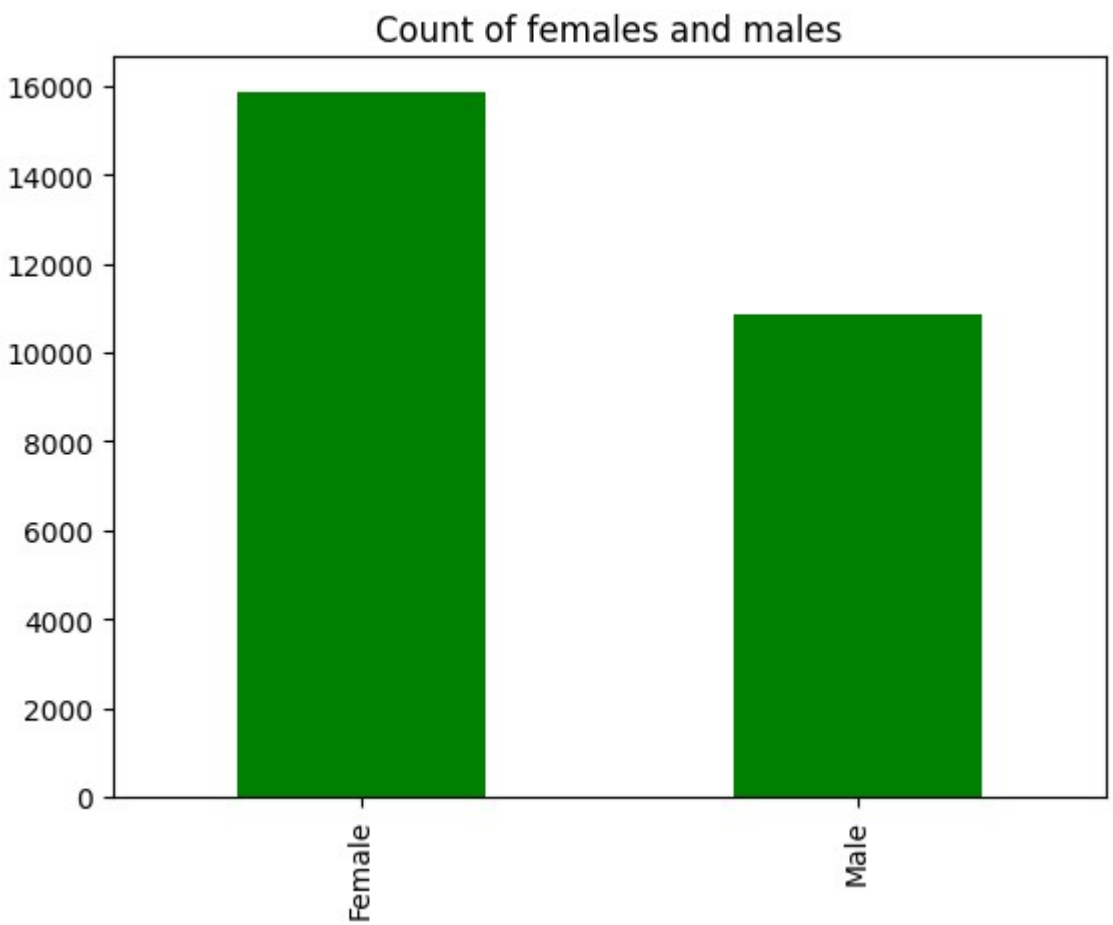
**Grouping with the joined table**

In [56]:
```
#grouping into different tables

# grouping by sex
sex = df.groupby('sex')

#grouping by race
race = df.groupby('race')

#grouping by education
educ = df.groupby('education')

#grouping by geo location
loc_ = df.groupby('hhs_geo_region')

#grouping by home ownership
home = df.groupby('rent_or_own')
```

## 6.2 Does sex has affected the distribution of the vaccine?

In [62]:
```
sex_plot = df['sex'].value_counts().plot(kind='bar', color = "green", titl
sex_plot
```

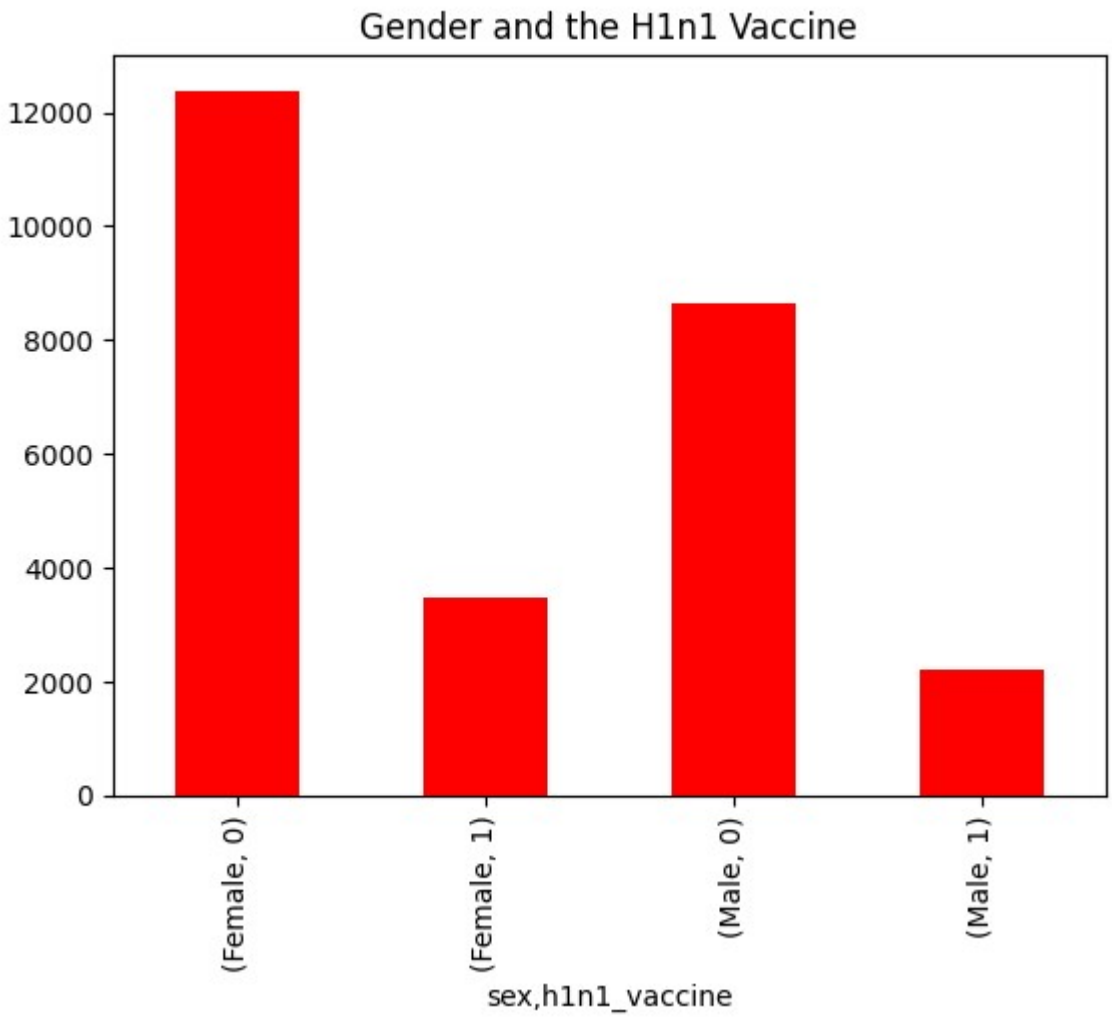
Out[62]: <AxesSubplot: title={'center': 'Count of females and males'}>



**Observation:**
The Number of females who took the survey were more than the number of males.

In [63]:
```
1 sex_vaccine = sex['h1n1_vaccine'].value_counts().plot(kind='bar', color =
2 sex_vaccine
```

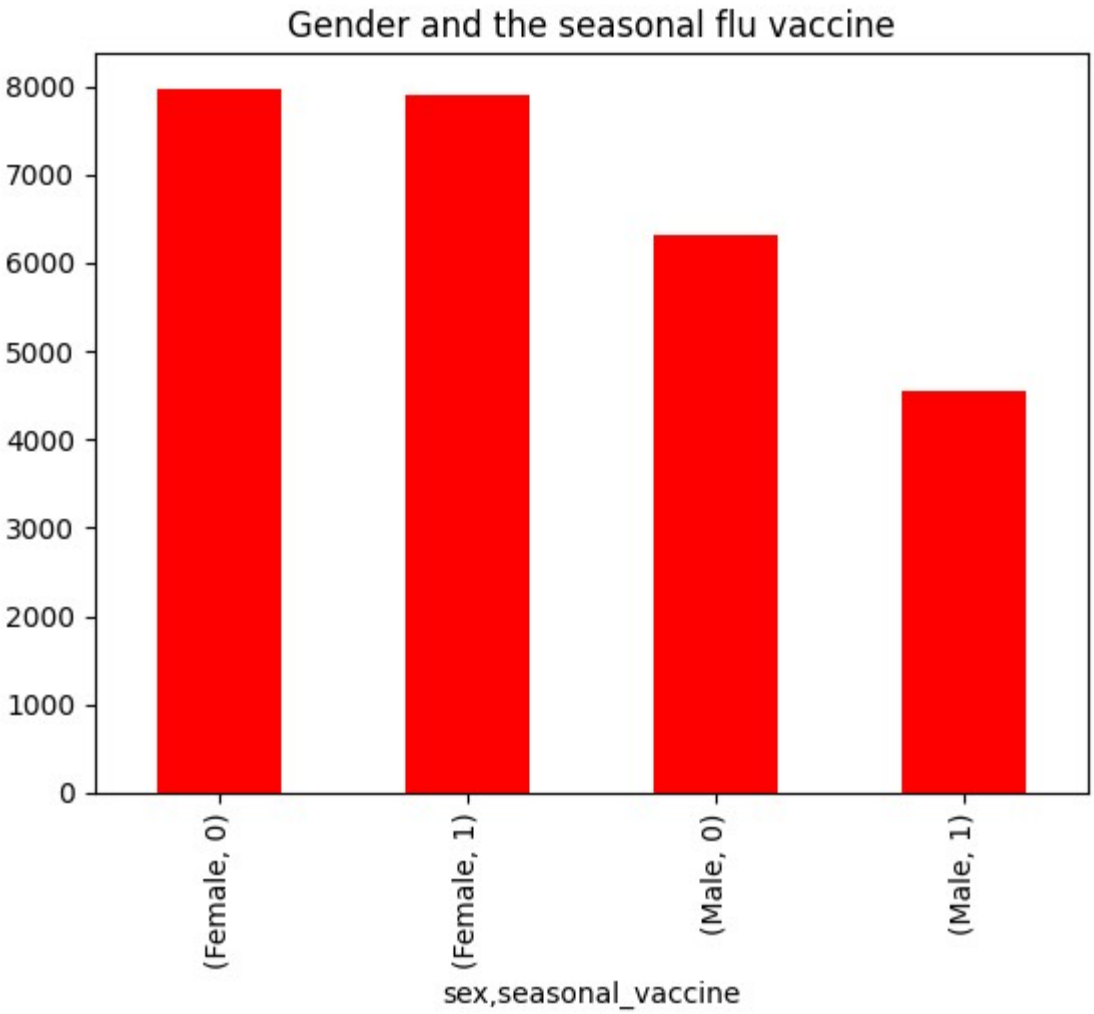
Out[63]:
```
<AxesSubplot: title={'center': 'Gender and the H1n1 Vaccine'}, xlabel='sex,h1
n1_vaccine'>
```




**Observation:**
The number of females and males that didnt receive the H1N1 vaccine are higher than those that received the vaccine.
Also the number of females that received the vaccine is higher than that of the male.

In [64]:
```
1 sex['seasonal_vaccine'].value_counts().plot(kind='bar', color = "red", tit
```

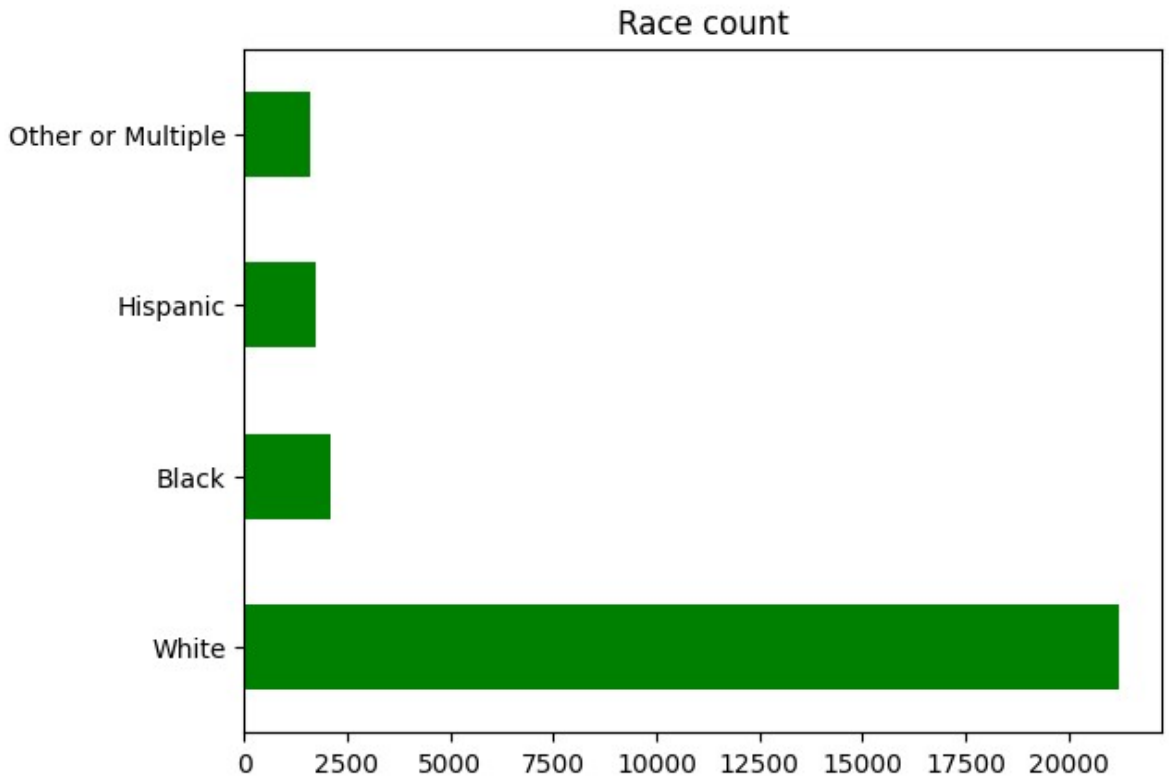
Out[64]:
```
<AxesSubplot: title={'center': 'Gender and the seasonal flu vaccine'}, xlabel
='sex,seasonal_vaccine'>
```




**Observation:**
The number of females who receive the seasonal flu vaccine and those who didn't receive are the flu vaccine are almost even.
The male who took the seasonal flu vaccine is more than those who took the H1N1 vaccine and also than those who didnt receive the same flu vaccine.

## 6.3 Does race affect the distribution of the vaccine?

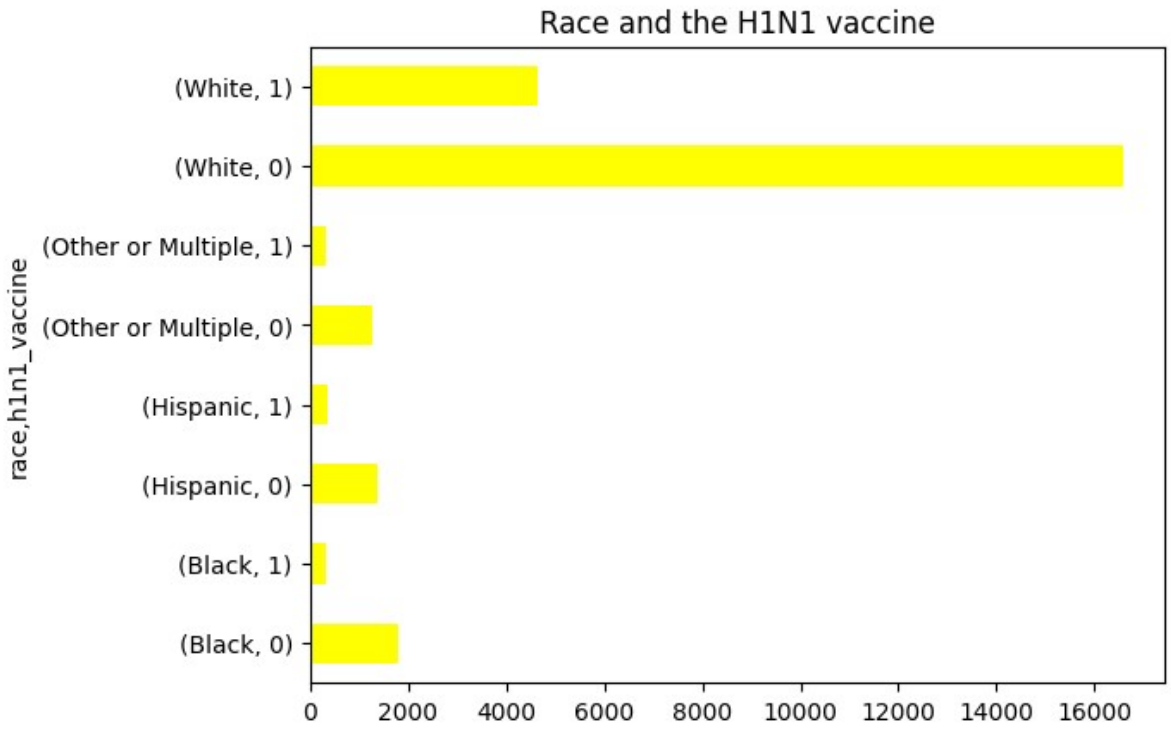
In [65]:
```
1 race_plot = df['race'].value_counts().plot(kind='barh', color = 'green', t
```




**Observation:**
The majority of those who took the survey were the Whites followed by Blacks, other or multiple have the least number of people doing the survey.

In [66]:
```
1 race['h1n1_vaccine'].value_counts().plot(kind='barh', color = "yellow", ti
```

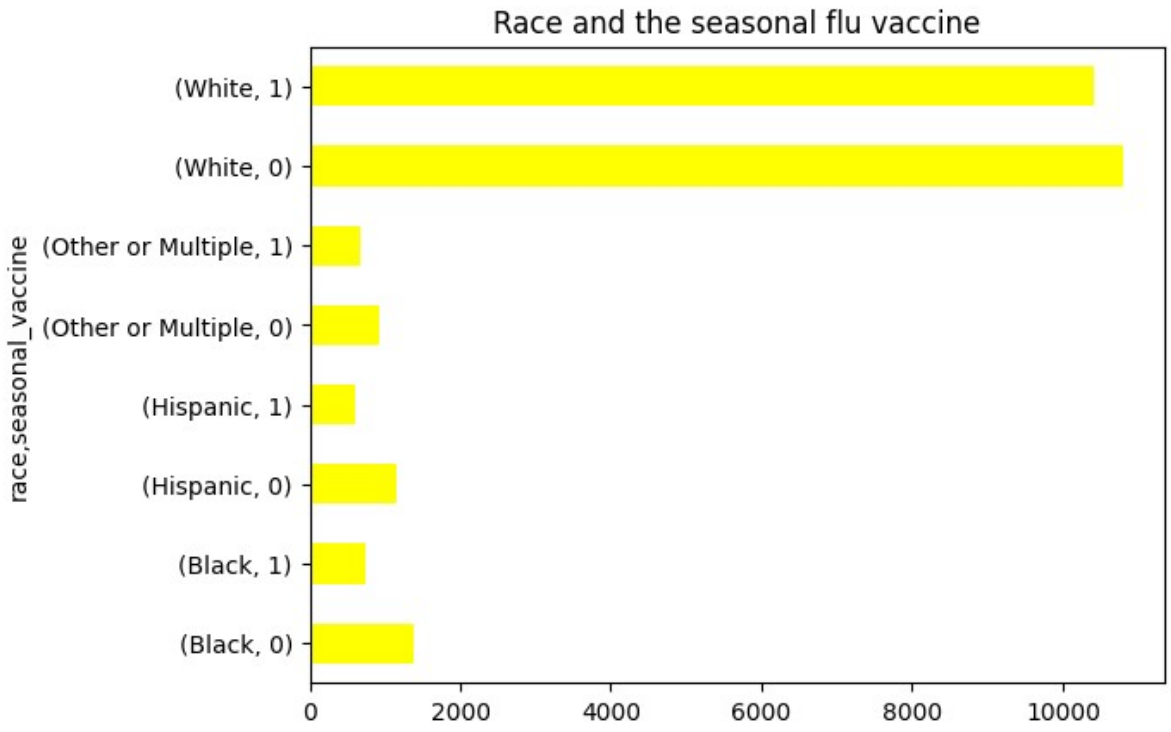
Out[66]:
```
<AxesSubplot: title={'center': 'Race and the H1N1 vaccine'}, ylabel='race,h1n
1_vaccine'>
```




**Observation:**
In all races the number of people didn't receive H1N1 vaccine is higher than those who received.

In [67]:
```
1 race['seasonal_vaccine'].value_counts().plot(kind='barh', color = "yellow"
```

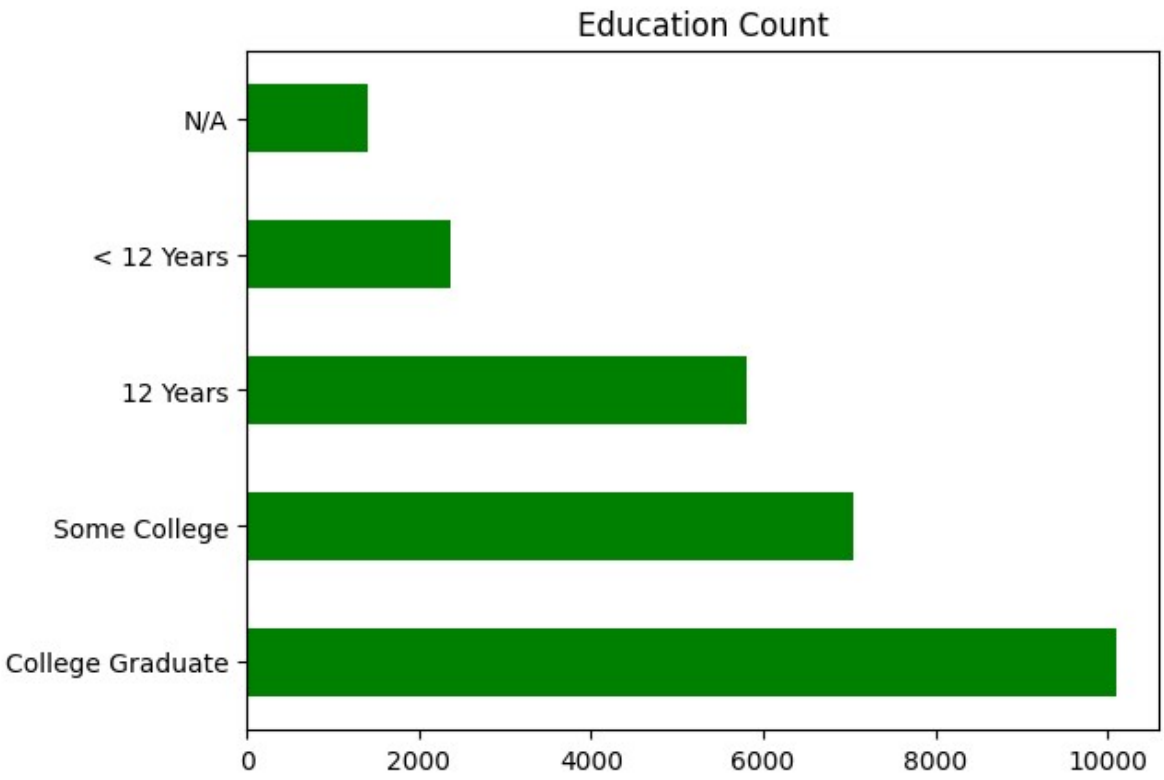
Out[67]:
```
<AxesSubplot: title={'center': 'Race and the seasonal flu vaccine'}, ylabel='
race,seasonal_vaccine'>
```




**Observation:**
In all races the number of people didn't receive Seasonal Flu vaccine is higher than those who received.

## 6.4 Does Education level affect the distribution of the vaccine?

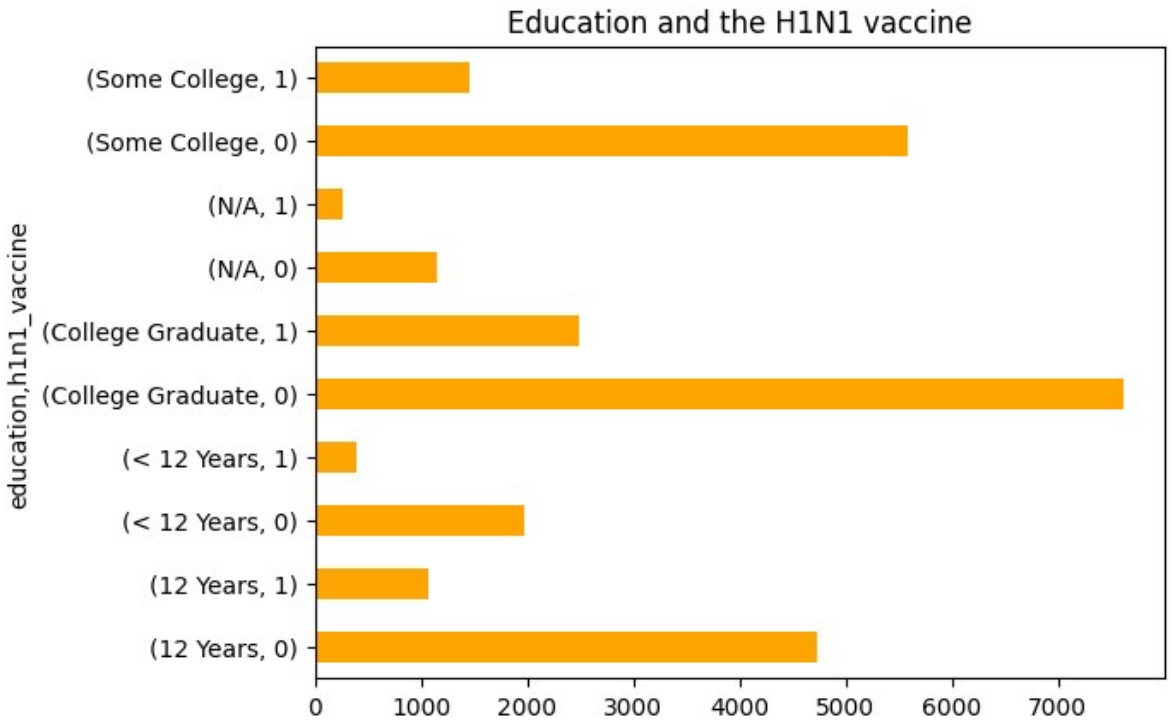
In [68]:
```
1 educ_plot = df['education'].value_counts().plot(kind='barh', color = 'gree
```




**Observation:**
The college graduates and those who went to some college lead in the did the survey. Those below 12 year old and 12 year olds who did the survey and were the least.

In [69]:
```
1 educ['h1n1_vaccine'].value_counts().plot(kind='barh', color = "orange", ti
```

Out[69]:
```
<AxesSubplot: title={'center': 'Education and the H1N1 vaccine'}, ylabel='edu
cation,h1n1_vaccine'>
```



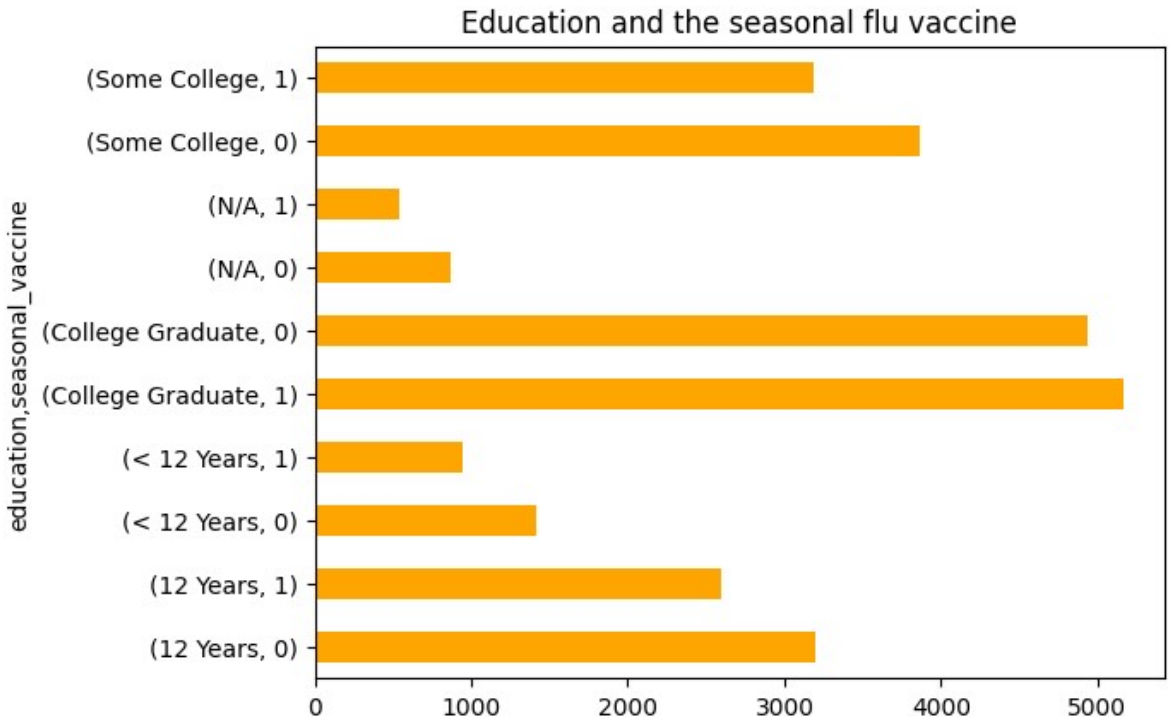

**Observation:**

The college graduate persons received the H1N1 vaccine most and also are the ones who failed to receive the vaccine most.

Those under 12 years old are the least to take the vaccine but it would be because few of them knew how to answer the questions asked during the survey.

In [70]:
```
1 educ['seasonal_vaccine'].value_counts().plot(kind='barh', color = "orange"
```

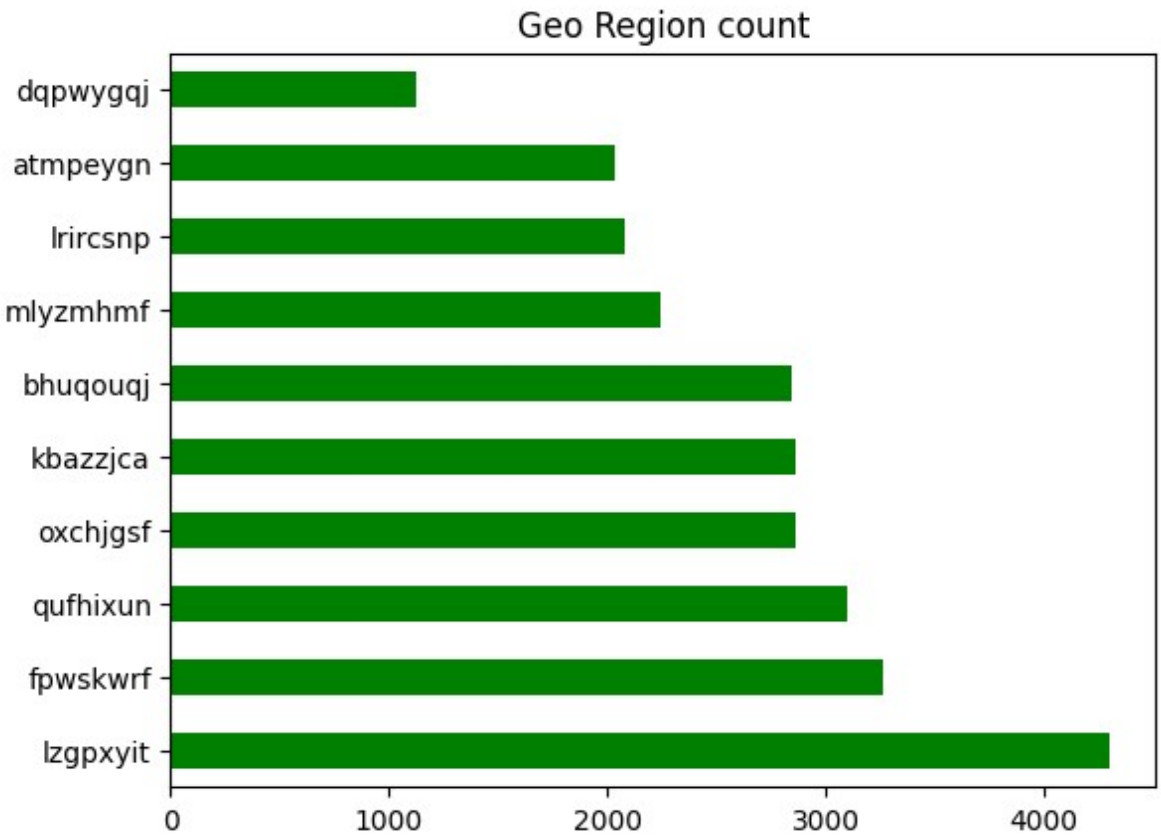
Out[70]:
```
<AxesSubplot: title={'center': 'Education and the seasonal flu vaccine'}, yla
bel='education,seasonal_vaccine'>
```




**Observation:**
The amount of those who took the seasonal vaccine are higher than those took the H1N1 vaccine.
In all education levels the difference between those who took the vaccine and those who didnt is significanly small.

## 6.5 Location affect the distribution of the vaccine?

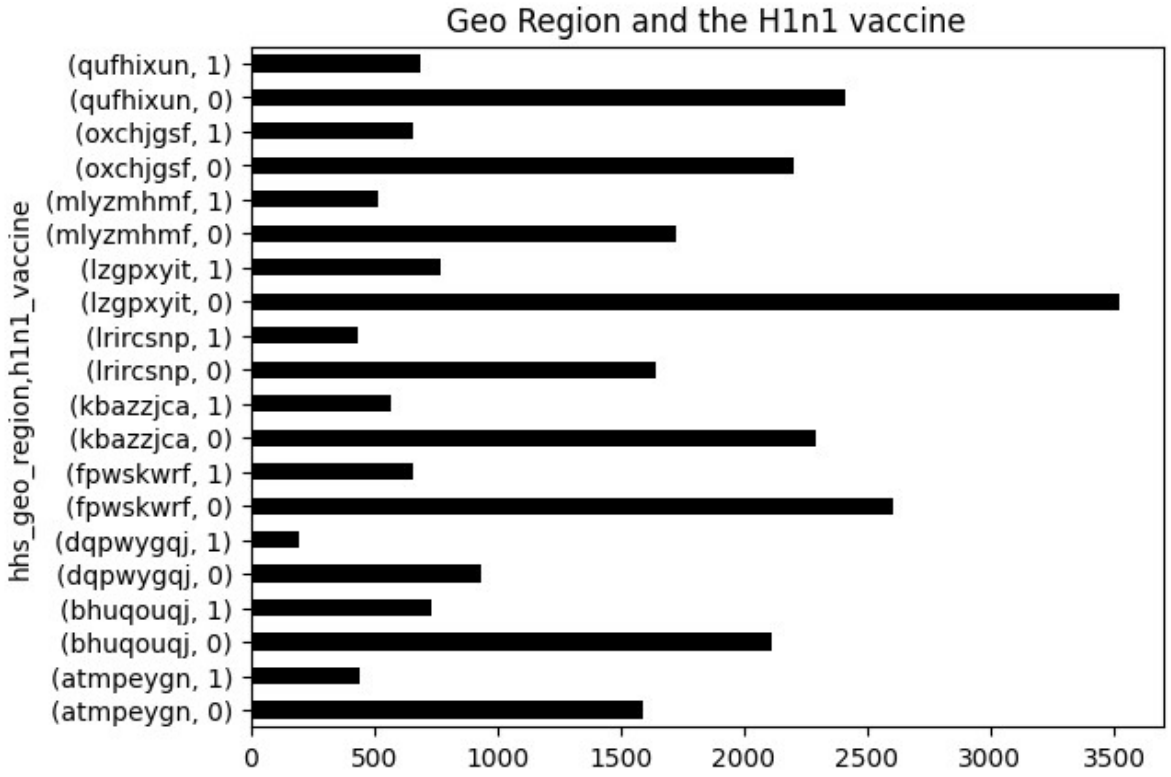
In [71]:
```
1 loc_plot = df['hhs_geo_region'].value_counts().plot(kind='barh', color = '
```




**Observation:**
The graph above shows the distribution of surveys taken by the geo location, with the highest been the lzgpxyit and the lowest been Dqpwyqj.

In [72]:
```
1 loc_['h1n1_vaccine'].value_counts().plot(kind='barh', color = "black", tit
```

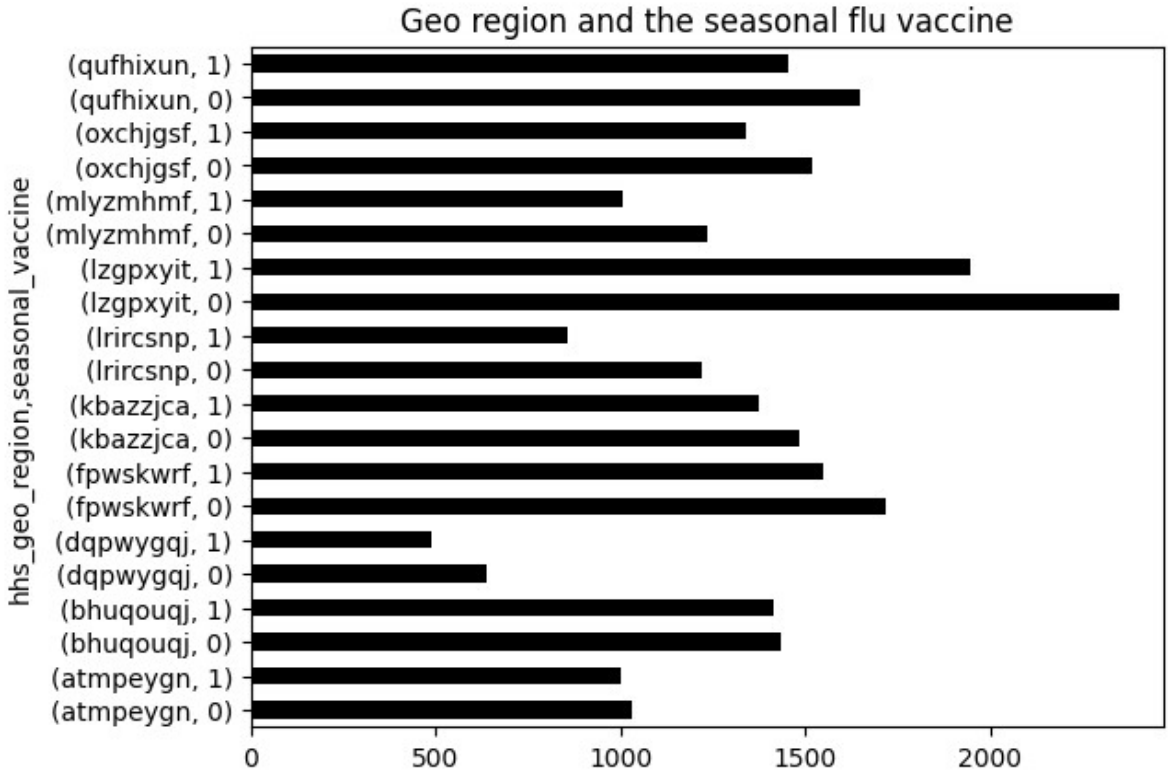
Out[72]:
```
<AxesSubplot: title={'center': 'Geo Region and the H1n1 vaccine'}, ylabel='hh
s_geo_region,h1n1_vaccine'>
```




**Observation:**
The number if people in all respective areas that didnt receive the vaccine is higher than those who received the vaccine.

In [45]:
```
1 loc_['seasonal_vaccine'].value_counts().plot(kind='barh', color = "black",
```

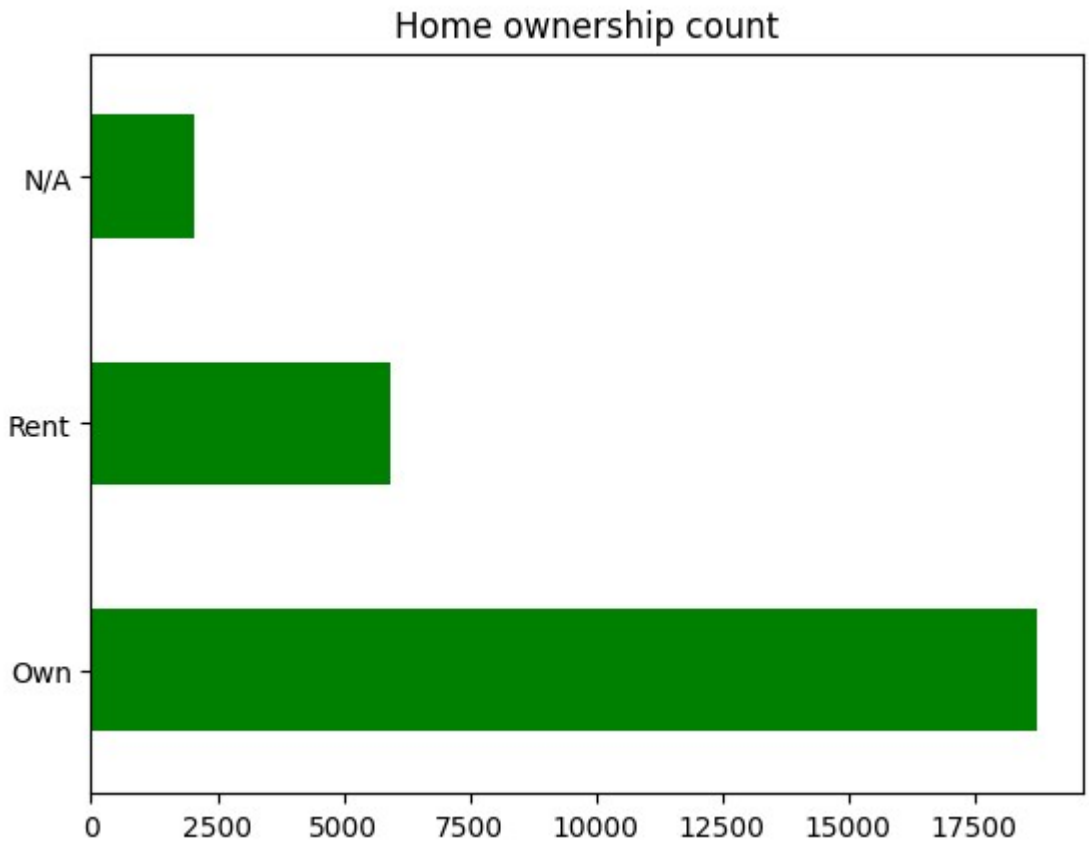
Out[45]:
```
<AxesSubplot: title={'center': 'Geo region and the seasonal flu vaccine'}, yl
abel='hhs_geo_region,seasonal_vaccine'>
```




**Observation:**
The amount of those who took the seasonal vaccine are higher than those took the H1N1 vaccine per geo location.
In all regions the difference between those who took the vaccine and those who didnt is significanly small.

## 6.6 Does Homeownership affect the distribution of the vaccine?

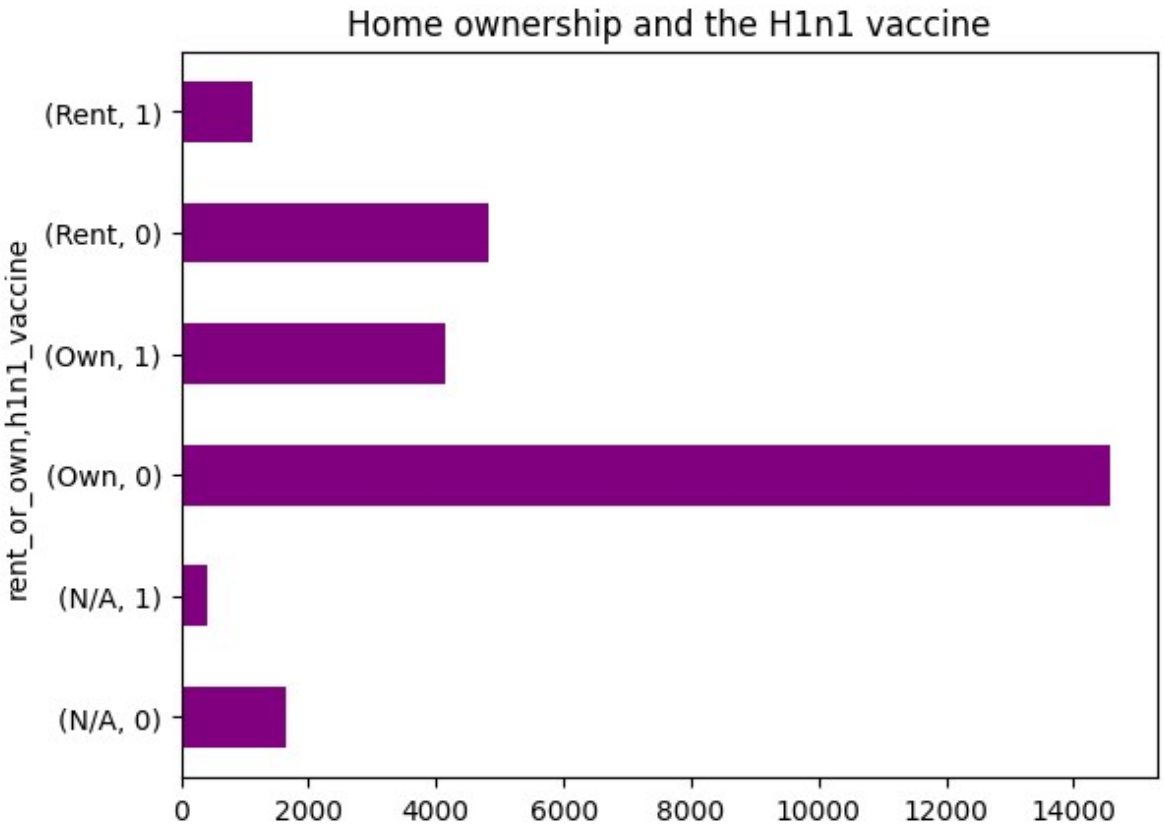
In [74]:
```
1 home_plot = df['rent_or_own'].value_counts().plot(kind='barh', color = 'gr
```




**Observation:**
Those who own their own houses took the survey the most.

In [75]:
```
1 home['h1n1_vaccine'].value_counts().plot(kind='barh', color = "purple", ti
```

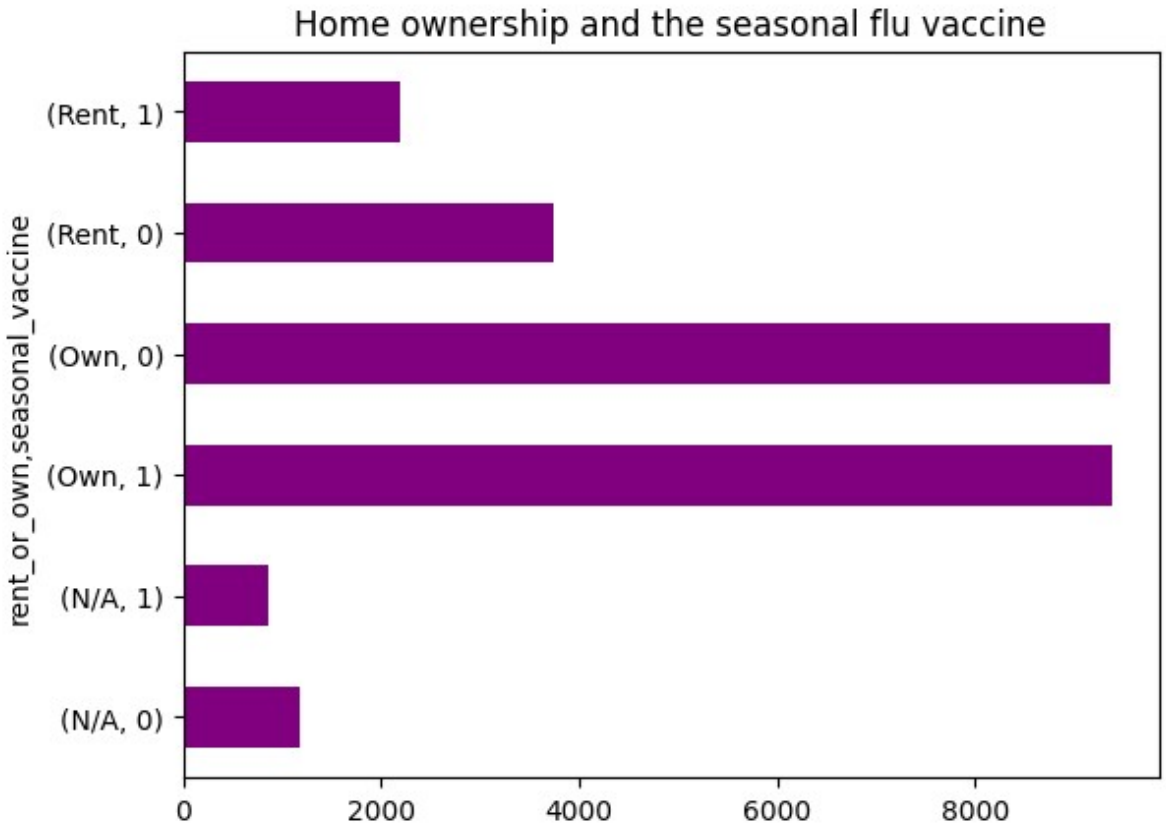
Out[75]:
```
<AxesSubplot: title={'center': 'Home ownership and the H1n1 vaccine'}, ylabel
='rent_or_own,h1n1_vaccine'>
```




**Observation:**
Those who own their own houses received more of the vaccine over whose who rented. This was also simillar to those who didn't receive the vaccine.

In [90]:
```
1 home['seasonal_vaccine'].value_counts().plot(kind='barh', color = "purple'
```

Out[90]:
```
<AxesSubplot: title={'center': 'Home ownership and the seasonal flu vaccine
'}, ylabel='rent_or_own,seasonal_vaccine'>
```



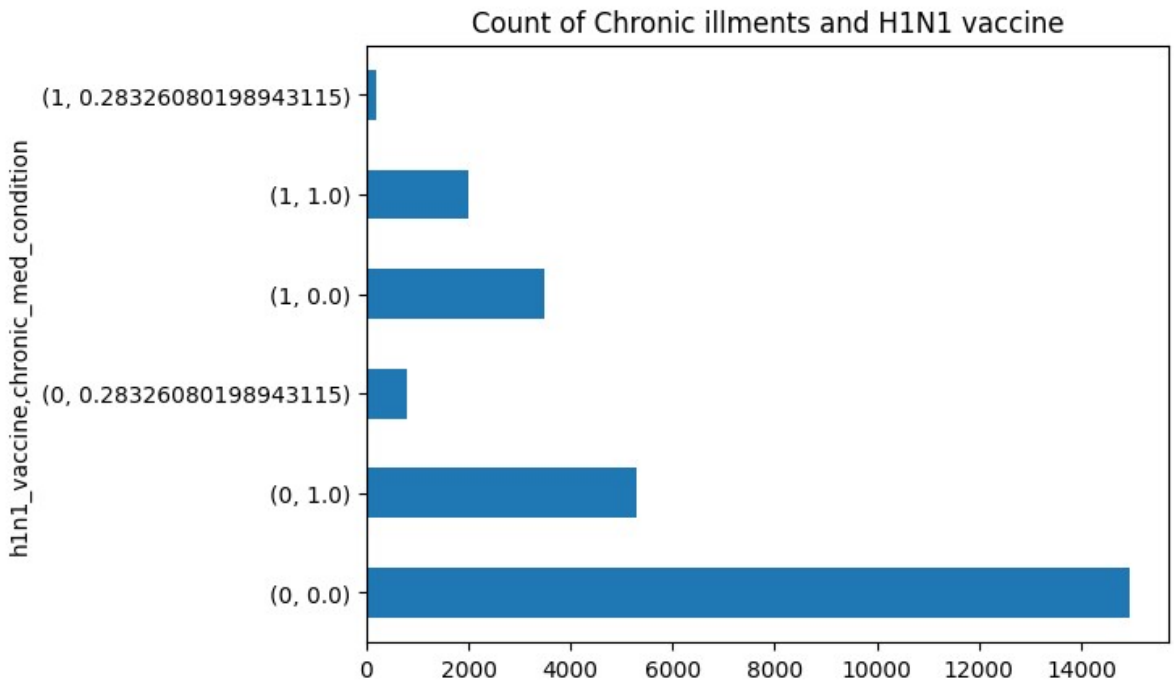

**Observation:**
In those who own their own houses and took the survey half received the vaccine and half didnt receive the vaccine.
while whose eho rent had a shorter number of those who took the vaccine.

## 6.7 How the chronic illnesses has affected the distribution of the vaccine?

In [96]:
```
1 h1n1_vaccine = df.groupby('h1n1_vaccine')
2 seasonal_vaccine = df.groupby('seasonal_vaccine')
```

In [93]:
```
1 h1n1_vaccine['chronic_med_condition'].value_counts().plot(kind='barh', tit
```

Out[93]:
```
<AxesSubplot: title={'center': 'Count of Chronic illments and H1N1 vaccine'},
ylabel='h1n1_vaccine,chronic_med_condition'>
```



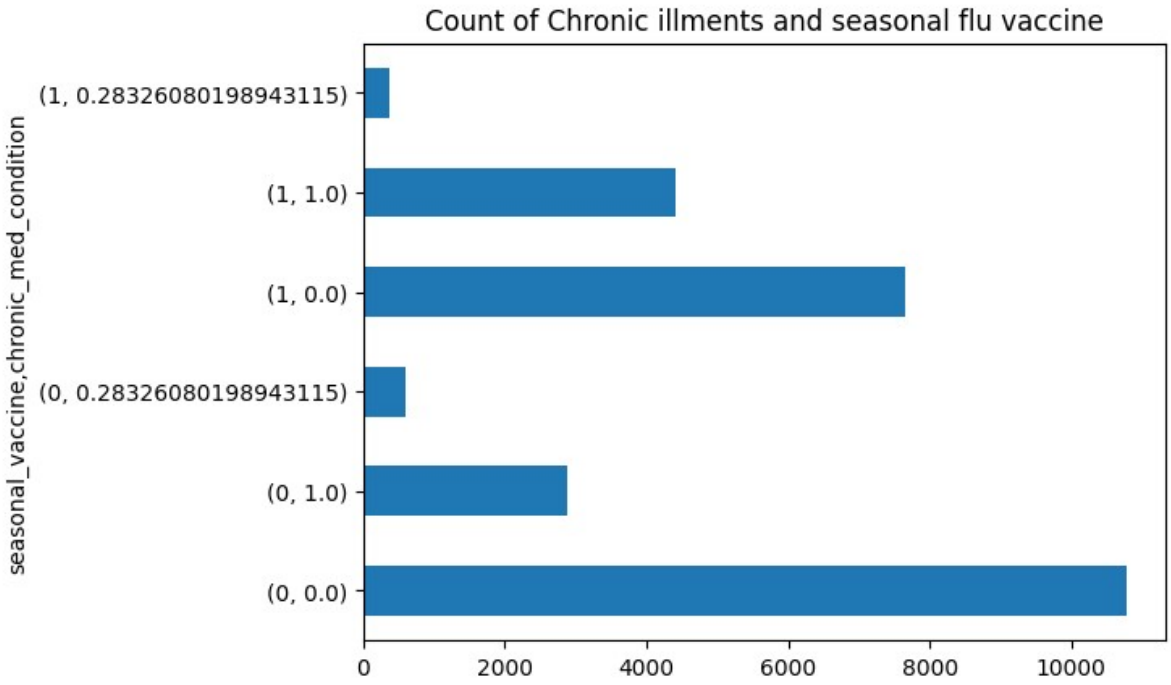

**Observation:**

For those who without chronic illnesses and didnt receive the virus take up a huge part of the data followed by those who didn't take the vaccine yrt have a chronic illness.

For those who received and had chronic illness summed up to the smallest percentage.

In [97]:
```
1 seasonal_vaccine['chronic_med_condition'].value_counts().plot(kind='barh',
```

Out[97]:
```
<AxesSubplot: title={'center': 'Count of Chronic illments and seasonal flu va
ccine'}, ylabel='seasonal_vaccine,chronic_med_condition'>
```

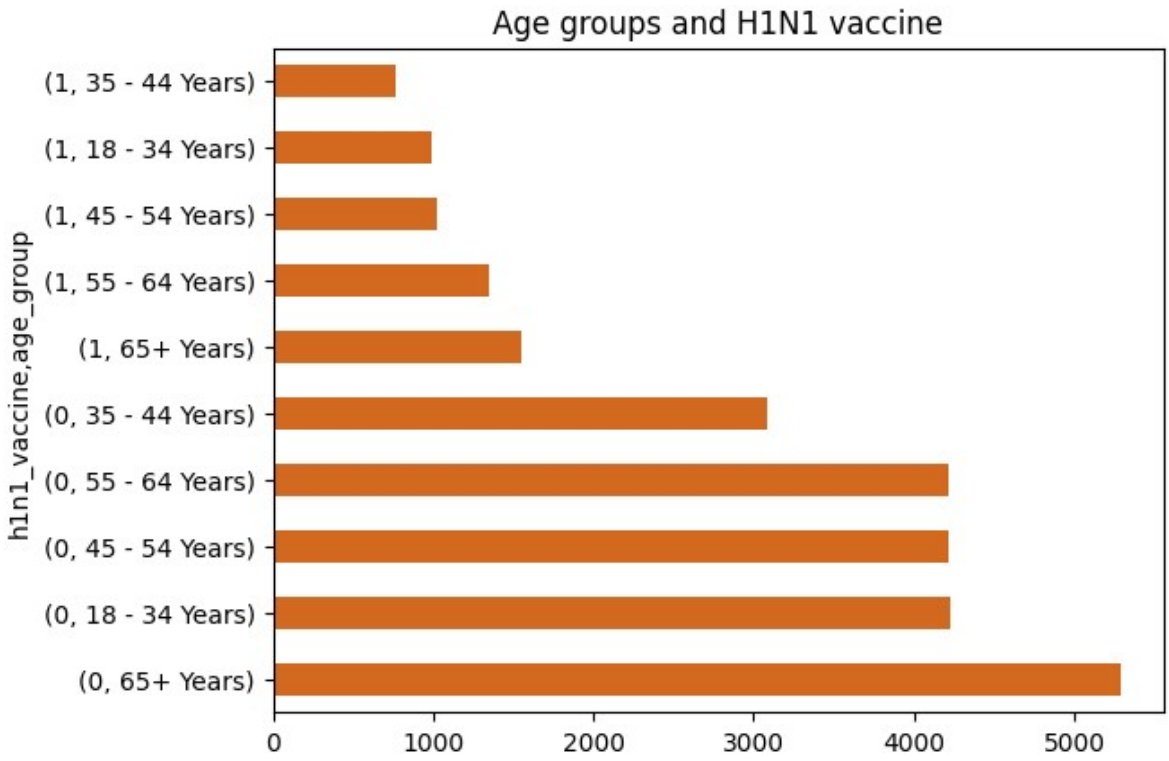



**Observation:**
In the seasonal flu vaccine, those who dont have any chronic illness and didn't receive the vaccine were the huge percentage, followed by those who received the vaccine and didn't have any chronic illnesses. The smallest percentage was held by those who didn't receive the vaccine but had a chronic illness.

## 6.8 Does Age group affect the distribution of the vaccine?

In [98]:
```
1 h1n1_vaccine['age_group'].value_counts().plot(kind='barh', color = 'chocol
```

Out[98]:
```
<AxesSubplot: title={'center': 'Age groups and H1N1 vaccine'}, ylabel='h1n1_v
accine,age_group'>
```



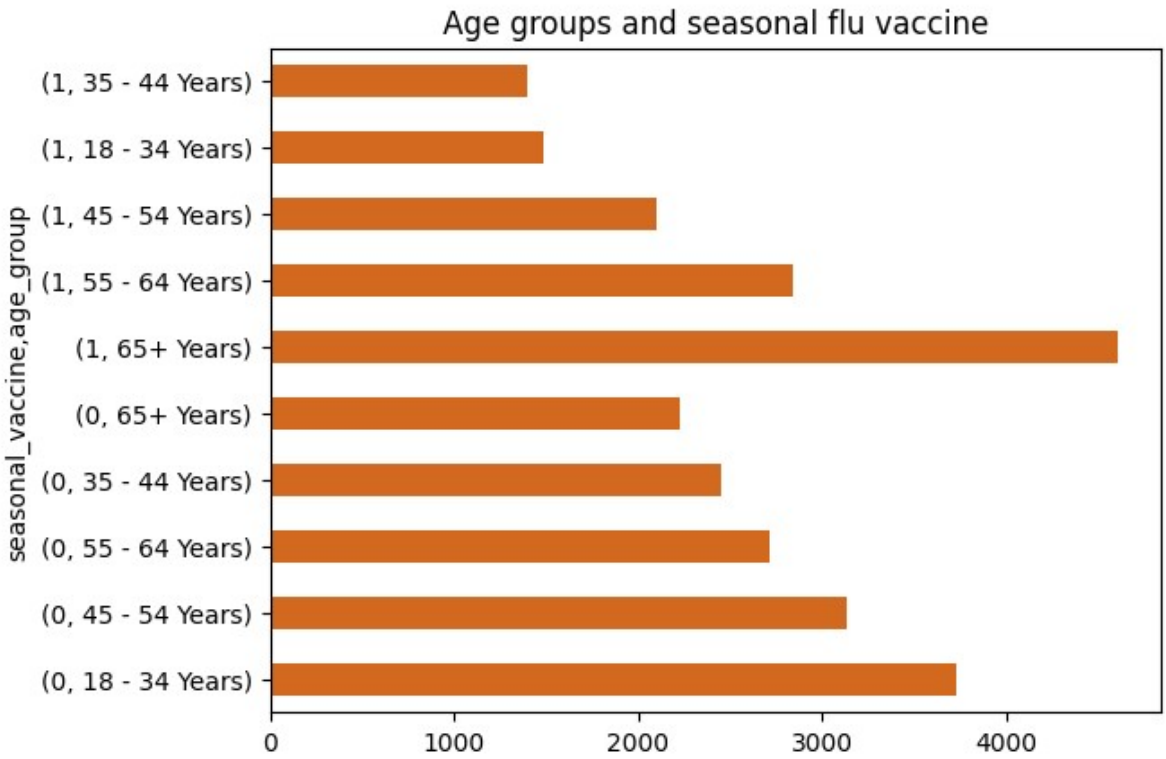

**Observation:**

The 65+ age group were the biggest percentrage of those who took the survey and a huge number of them didnt receive the H1N1 vaccine.

For those between 35 and 44 years, they were the least to take the survey and also least receivers of the vaccine.

In [99]:
```
1 seasonal_vaccine['age_group'].value_counts().plot(kind='barh', color = 'ch
```

Out[99]:
```
<AxesSubplot: title={'center': 'Age groups and seasonal flu vaccine'}, ylabel
='seasonal_vaccine,age_group'>
```




**Observation:**
Those in their 65+ years took the seasonal flu having the highest percentage.
As in the H1N1 vaccine, those between 35 and 44 years were the least to take the seasonal vaccine.
Most of the youths didnt receive the seasonal drug.

## 6.9 Correlation of the dataset

In [100]:
```
1 #correlation of h1n1 vaccine
2 corr_matrix = df.corr()
3 corr1 = pd.DataFrame(corr_matrix['h1n1_vaccine'].sort_values(ascending=Fal
4 corr1.head(5)
```

Out[100]:

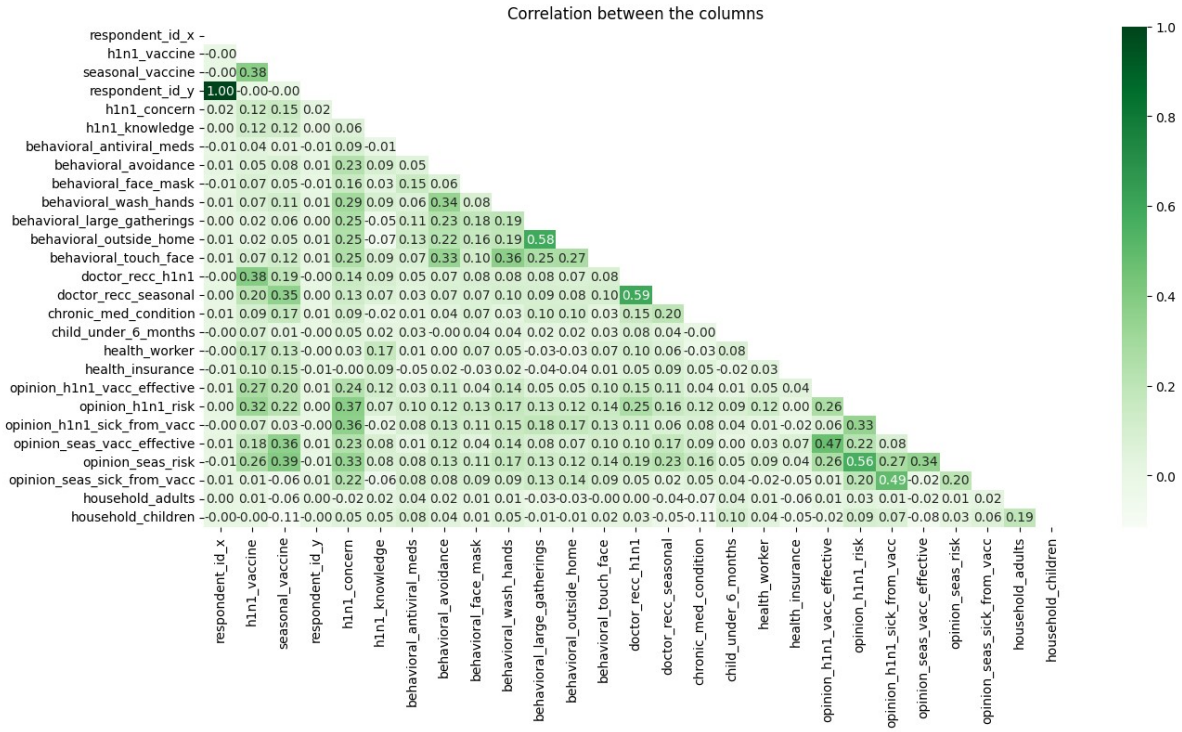
| | h1n1_vaccine |
|---|---|
| **h1n1_vaccine** | 1.000000 |
| **doctor_recc_h1n1** | 0.384662 |
| **seasonal_vaccine** | 0.377143 |
| **opinion_h1n1_risk** | 0.320833 |
| **opinion_h1n1_vacc_effective** | 0.267491 |

In [101]:
```
1 #correlation of the seasonal vaccine
2 corr_matrix = df.corr()
3 corr2 = pd.DataFrame(corr_matrix['seasonal_vaccine'].sort_values(ascending
4 corr2.head(5)
```

Out[101]:

| | seasonal_vaccine |
|---|---|
| **seasonal_vaccine** | 1.000000 |
| **opinion_seas_risk** | 0.386410 |
| **h1n1_vaccine** | 0.377143 |
| **opinion_seas_vacc_effective** | 0.358802 |
| **doctor_recc_seasonal** | 0.354363 |

In [102]:
```
#correlation heatmap
plt.figure(figsize=(15,7))
mask = np.triu(np.ones_like(df.corr(), dtype=bool))
sns.heatmap(df.corr(),annot=True,cmap="Greens",fmt=".2f", mask=mask);
plt.title('Correlation between the columns')
plt.show()
```




**Observation:**

The **doctor_recc_h1n1** column is highly correlated to the h1n1_vaccine column.

The **opinion_seas_risk** column is highly correlated to the seasonal_vaccine column.

# 7. Modelling

The metric I will be using for modellling is Accuracy

For modelling I will apply:

1. Perform an X_Train-y_Test Split
2. Build and Evaluate a Baseline Model
3. Find the Random Forest Model
4. Build Iterative Models to Find the Best Decision Tree Model
5. Build additional Logistic Regression Models
6. Choose and Evaluate a Final Model

## 7.1 Splitting of the data

In [103]:
```
#spliting of data
X_train, X_test, y_train, y_test = train_test_split(training_features, tra
```

In [106]:
```
#checking the data types of the training features dataset
training_features.dtypes
```

Out[106]:
```
respondent_id                  int64
h1n1_concern                 float64
h1n1_knowledge               float64
behavioral_antiviral_meds    float64
behavioral_avoidance         float64
behavioral_face_mask         float64
behavioral_wash_hands        float64
behavioral_large_gatherings  float64
behavioral_outside_home      float64
behavioral_touch_face        float64
doctor_recc_h1n1             float64
doctor_recc_seasonal         float64
chronic_med_condition        float64
child_under_6_months         float64
health_worker                float64
health_insurance             float64
opinion_h1n1_vacc_effective  float64
opinion_h1n1_risk            float64
opinion_h1n1_sick_from_vacc  float64
opinion_seas_vacc_effective  float64
opinion_seas_risk            float64
opinion_seas_sick_from_vacc  float64
age_group                     object
education                     object
race                          object
sex                           object
income_poverty                object
marital_status                object
rent_or_own                   object
employment_status             object
hhs_geo_region                object
census_msa                    object
household_adults             float64
household_children           float64
employment_industry           object
employment_occupation         object
dtype: object
```

In [105]:
```
#defining the numeric columns
numeric_cols = training_features.columns[training_features.dtypes != "obje
print(numeric_cols)
```

```
['respondent_id' 'h1n1_concern' 'h1n1_knowledge'
 'behavioral_antiviral_meds' 'behavioral_avoidance' 'behavioral_face_mask'
 'behavioral_wash_hands' 'behavioral_large_gatherings'
 'behavioral_outside_home' 'behavioral_touch_face' 'doctor_recc_h1n1'
 'doctor_recc_seasonal' 'chronic_med_condition' 'child_under_6_months'
 'health_worker' 'health_insurance' 'opinion_h1n1_vacc_effective'
 'opinion_h1n1_risk' 'opinion_h1n1_sick_from_vacc'
 'opinion_seas_vacc_effective' 'opinion_seas_risk'
 'opinion_seas_sick_from_vacc' 'household_adults' 'household_children']
```

In [107]:
```
#Defining the non numeric columns
non_numeric = training_features.columns[training_features.dtypes == 'objec
print(non_numeric)
```

```
['age_group' 'education' 'race' 'sex' 'income_poverty' 'marital_status'
 'rent_or_own' 'employment_status' 'hhs_geo_region' 'census_msa'
 'employment_industry' 'employment_occupation']
```

## 7.2 Scaling the Data

In [110]:
```
#scaling the numeric columns
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
numeric_col = scaler.fit_transform(X_train[numeric_cols])
pd.DataFrame(numeric_col).head(3)
```

Out[110]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | -1.041146 | 0.417121 | 1.194056 | -0.22696 | -1.625787 | -0.274046 | 0.464944 | -0.747082 | -0.716108 |
| **1** | -1.377771 | 0.417121 | -0.429874 | -0.22696 | 0.620043 | -0.274046 | 0.464944 | 1.343219 | -0.716108 |
| **2** | -0.875035 | 1.520757 | -0.429874 | -0.22696 | 0.620043 | -0.274046 | 0.464944 | -0.747082 | -0.716108 |

3 rows × 24 columns

In [109]:
```
#Encoding the non numeric data types
from sklearn.preprocessing import OrdinalEncoder

ordinal = OrdinalEncoder()
non_numeric_col = ordinal.fit_transform(X_train[non_numeric])
pd.DataFrame(non_numeric_col).head(3)
```

Out[109]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4.0 | 2.0 | 3.0 | 1.0 | 1.0 | 0.0 | 1.0 | 2.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **1** | 0.0 | 0.0 | 1.0 | 1.0 | 2.0 | 2.0 | 2.0 | 0.0 | 8.0 | 1.0 | 5.0 | 11.0 |
| **2** | 2.0 | 2.0 | 3.0 | 0.0 | 1.0 | 2.0 | 2.0 | 0.0 | 3.0 | 0.0 | 5.0 | 1.0 |

In [111]:
```
1 X_train[non_numeric] = non_numeric_col
2 X_train[numeric_cols] = numeric_col
3
4 X_train.head(3)
```

Out[111]:

| respondent_id | respondent_id | h1n1_concern | h1n1_knowledge | behavioral_antiviral_meds | behavi |
|---|---|---|---|---|---|
| **5303** | -1.041146 | 0.417121 | 1.194056 | -0.22696 | |
| **2703** | -1.377771 | 0.417121 | -0.429874 | -0.22696 | |
| **6586** | -0.875035 | 1.520757 | -0.429874 | -0.22696 | |

3 rows × 36 columns

In [112]:
```
1 numeric_col_test = scaler.fit_transform(X_test[numeric_cols])
2 numeric_col_test
```

Out[112]:
```
array([[ 0.3125444 ,  0.42614519, -0.41584226, ...,  0.0031866 ,
        -0.00238694,  0.00450462],
       [-0.51659653,  1.51931056, -2.03109356, ...,  0.0031866 ,
         0.14696183,  0.51060879],
       [ 0.40933179, -0.66702018, -0.41584226, ..., -0.84813565,
        -1.16887888,  0.51060879],
       ...,
       [ 1.62718545, -1.76018555, -2.03109356, ..., -0.84813565,
         1.46280254, -0.57681237],
       [-0.97682511,  1.51931056, -0.41584226, ..., -0.84813565,
         0.14696183, -0.57681237],
       [ 1.15445137,  1.51931056,  1.19940903, ..., -0.84813565,
         0.14696183, -0.57681237]])
```

In [113]:
```
1 non_numeric_col_test = ordinal.fit_transform(X_test[non_numeric])
2 non_numeric_col_test
```

Out[113]:
```
array([[ 0.,  3.,  3., ...,  1.,  0.,  0.],
       [ 1.,  3.,  3., ...,  1.,  0.,  0.],
       [ 2.,  2.,  3., ...,  0.,  9., 11.],
       ...,
       [ 0.,  1.,  3., ...,  1.,  0.,  0.],
       [ 3.,  2.,  0., ...,  1.,  0.,  0.],
       [ 3.,  4.,  1., ...,  1.,  0.,  0.]])
```

In [114]:
```
X_test[non_numeric] = non_numeric_col_test
X_test[numeric_cols] = numeric_col_test

X_test.head(3)
```

Out[114]:

| respondent_id | h1n1_concern | h1n1_knowledge | behavioral_antiviral_meds | behavi |
|---|---|---|---|---|
| **respondent_id** | | | | |
| **15772** | 0.312544 | 0.426145 | -0.415842 | -0.226798 |
| **9407** | -0.516597 | 1.519311 | -2.031094 | -0.226798 |
| **16515** | 0.409332 | -0.667020 | -0.415842 | -0.226798 |

3 rows × 36 columns

# 7.3.1. Logistical Regression

Documentation (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

In this section I will fit the logistic regression algorithm to the X_train and y_train.
Later predict our y with our model.

In [115]:
```
# Droping the respondent id and seasonal_vaccine column to remain with the
y_train_h1n1 = y_train.drop(columns=['respondent_id', 'seasonal_vaccine'])

```

In [116]:
```
# Droping the respondent id and seasonal_vaccine column to remain with the
y_test_h1n1 = y_test.drop(columns=['respondent_id', 'seasonal_vaccine'])
```

In [117]:
```
# Instantiate a LogisticRegression with random_state=42
baseline_model = LogisticRegression(random_state=42, C=1e12)

# Use cross_val_score with scoring="neg_log_loss" to evaluate the model
# on X_train and y_train
baseline_neg_log_loss_cv = cross_val_score(baseline_model, X_train, y_trai

baseline_log_loss = -(baseline_neg_log_loss_cv.mean())
baseline_log_loss
```

Out[117]:
```
0.38406664844463856
```

In [118]:
```
#Calculating the accuracy of the model
baseline_model.fit(X_train, y_train_h1n1)
y_pred = baseline_model.predict(X_test)

log_acc = accuracy_score(y_pred, y_test_h1n1)
log_acc
```

Out[118]:
```
0.8381380257082242
```

In [119]:
```
#calculating the precision

# Display the precision score

log_pre = precision_score(y_test_h1n1, y_pred)
log_pre
```

Out[119]: 0.6870876531573987

In [120]:
```
# Display the recall score

log_rec = recall_score(y_test_h1n1, y_pred)
log_rec
```

Out[120]: 0.43034238488783944

In [121]:
```
#modeling with the seasonal vaccine data
#droping the respondent id and h1n1_vaccine in both the training and test
y_train_seas = y_train.drop(columns=['respondent_id', 'h1n1_vaccine'])
y_test_seas = y_test.drop(columns=['respondent_id', 'h1n1_vaccine'])
```

In [122]:
```
# Use cross_val_score with scoring="neg_log_loss" to evaluate the model
# on X_train and y_train
baseline_neg_log_loss_cv2 = cross_val_score(baseline_model, X_train, y_tra


baseline_log_loss2 = -(baseline_neg_log_loss_cv2.mean())
baseline_log_loss2
```

Out[122]: 0.4893786174127728

In [123]:
```
#Calculating the accuracy of the model
baseline_model.fit(X_train, y_train_seas)
y_pred2 = baseline_model.predict(X_test)

log_acc2 = accuracy_score(y_pred2, y_test_seas)
log_acc2
```

Out[123]: 0.780980906027705

In [124]:
```
#calculating the precision

# Display the precision score
log_pre2 = precision_score(y_test_seas, y_pred2)
log_pre2
```

Out[124]: 0.7677473448854109

In [125]:
```
# Display the recall score

log_rec2 = recall_score(y_test_seas, y_pred2)
log_rec2
```

Out[125]: 0.7482974666303459

In [126]:
```
log_acc_final = (log_acc + log_acc2)/2
log_acc_final
```

Out[126]: 0.8095594658679646

In [127]:
```
log_r_2_ = r2_score(y_test_h1n1, y_pred)
# Calculate the root mean squared error between 'y_true' and 'y_predict'
log_mse = np.sqrt(mean_squared_error(y_test_h1n1, y_pred))
print(log_r_2_, log_mse)
```

```
0.02910255665294592 0.40232073559757753
```

In [128]:
```
log_r_2 = r2_score(y_test_seas, y_pred2)
# Calculate the root mean squared error between 'y_true' and 'y_predict'
log_mse_ = np.sqrt(mean_squared_error(y_test_seas, y_pred2))
print(log_r_2, log_mse_)
```

```
0.1177370130346771 0.4679947584880572
```

Our model got an accuracy of:

H1N1 vaccine Accuracy score: **83.81%**
Seasonal Flu vaccine Accuracy score: **78.09%**
Final Accuracy score: **80.95%**

# 7.3.2. K-Nearest Neighbours, (KNN)

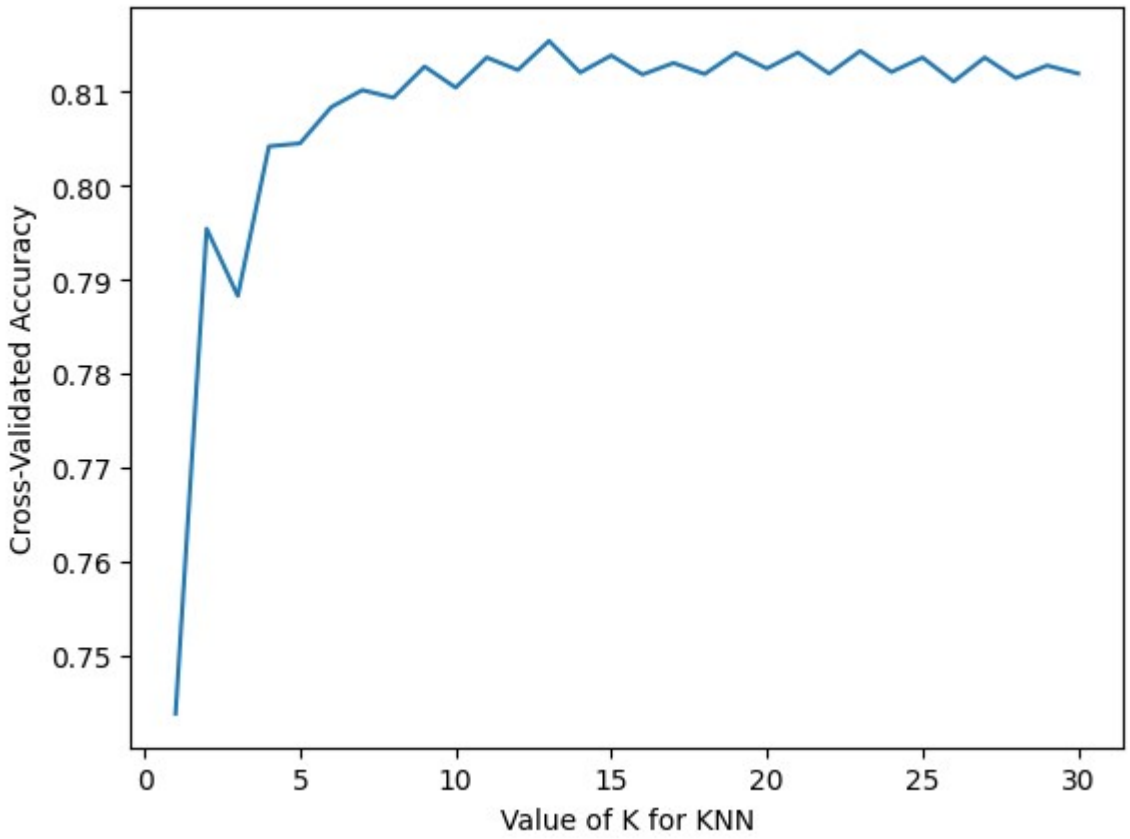
Documentation (https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html#sklearn.neighbors.KNeighborsRegressor)
With the KNN model, hyperparameter tuning will be used to find the best fitting model.

In [129]:
```
# Instantiatethe classifier model with n-neighbours of 3
knn = KNeighborsClassifier(n_neighbors=3)

# fitting the model
knn.fit(X_train, y_train_h1n1)

#performing the cross val score
knn_cross_val = cross_val_score(knn, X_train, y_train_h1n1, cv=3, scoring=
print(knn_cross_val)
```

```
[0.79155969 0.7960199  0.79008185]
```

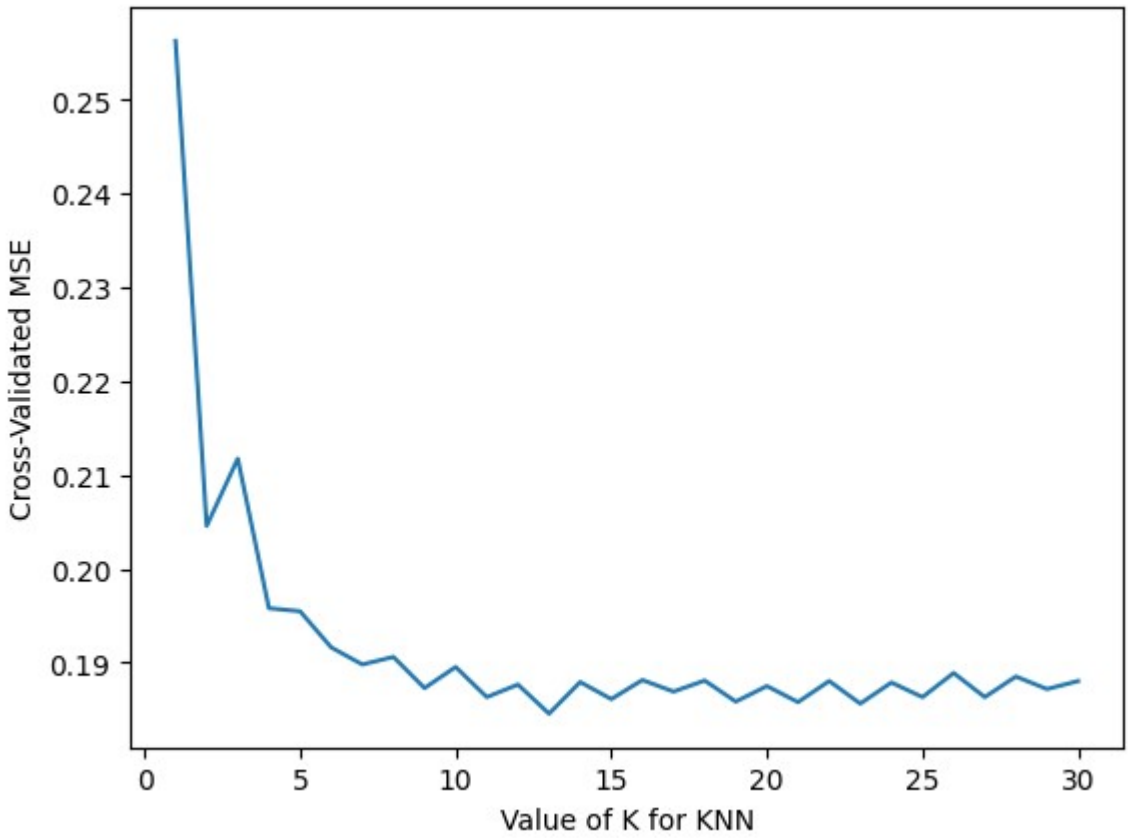
**Hyperparameter Tuning**

In [130]:
```
#Use cross val score for scoring(good for regression)
k_range = range(1, 31)
k_scores = []# use iteration to caclulator different k in models, then ret
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train_h1n1, cv=5, scoring='ac
    k_scores.append(scores.mean())# plot to see clearly
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.show()
```




The best value for the n-neighbours is about 9

In [131]:
```
#Use neg_mean_squared_error for scoring(good for regression)
k_range = range(1, 31)
k_scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    loss = abs(cross_val_score(knn, X_train, y_train_h1n1, cv=5, scoring='
    k_scores.append(loss.mean())
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated MSE')
plt.show()
```




Also by using the best fit for k is around **9**

In [134]:
```
#best model
from sklearn.neighbors import KNeighborsClassifier

knn1 = KNeighborsClassifier(n_neighbors=13)
knn1.fit(X_train, y_train_h1n1)

knn_cross_val1 = cross_val_score(knn, X_train, y_train_h1n1, cv=3, scoring
print(knn_cross_val1)
```

```
[0.81258023 0.81222918 0.8119082 ]
```

In [135]:
```
#Calculating the accuracy of the mode
knn1.fit(X_train, y_train_h1n1)
y_pred_knn = knn1.predict(X_test)

knn_acc = accuracy_score(y_pred_knn, y_test_h1n1)
knn_acc
```

Out[135]: 0.8133033820042431

In [136]:
```
#calculating the precision

# Display the precision score
knn_pre = precision_score(y_test_h1n1, y_pred_knn)
knn_pre
```

Out[136]: 0.6402266288951841

In [137]:
```
# Display the recall score

knn_rec = recall_score(y_test_h1n1, y_pred_knn)
knn_rec
```

Out[137]: 0.2668240850059032

In [138]:
```
#Calculating the accuracy of the model
knn1.fit(X_train, y_train_seas)
y_pred_knn2 = knn1.predict(X_test)

knn_acc2 = accuracy_score(y_pred_knn2, y_test_seas)
knn_acc2
```

Out[138]: 0.7273181080743791

In [139]:
```
#calculating the precision

# Display the precision score
knn_pre2 = precision_score(y_test_seas, y_pred_knn2)
knn_pre2
```

Out[139]: 0.7120433789954338

In [172]:
```
# Display the recall score

knn_rec2 = recall_score(y_test_seas, y_pred_knn2)
knn_rec2
```

Out[172]: 0.6796513211658949

In [140]:
```
acc_knn = (knn_acc + knn_acc2)/2
acc_knn
```

Out[140]: 0.7703107450393112

In [141]:
```
knn_r_2_ = r2_score(y_test_h1n1, y_pred_knn)
# Calculate the root mean squared error between 'y_true' and 'y_predict'
knn_mse = np.sqrt(mean_squared_error(y_test_h1n1, y_pred_knn))
print(knn_r_2_, knn_mse)
```

```
-0.11986320373723425 0.43208404043166987
```

In [142]:
```
knn_r_2 = r2_score(y_test_seas, y_pred_knn2)
# Calculate the root mean squared error between 'y_true' and 'y_predict'
knn_mse_ = np.sqrt(mean_squared_error(y_test_seas, y_pred_knn2))
print(knn_r_2, knn_mse_)
```

```
-0.09842998662064417 0.52218951724984
```

Our model got an accuracy of:

H1N1 vaccine Accuracy score: **81.33%**
Seasonal Flu vaccine Accuracy score: **72.73%**
Final Accuracy score: **77.03%**

# 7.3.3. Decision Tree

Documentation (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)
In this model i will carry out hyperparameters tuning to find the best fit for our model.

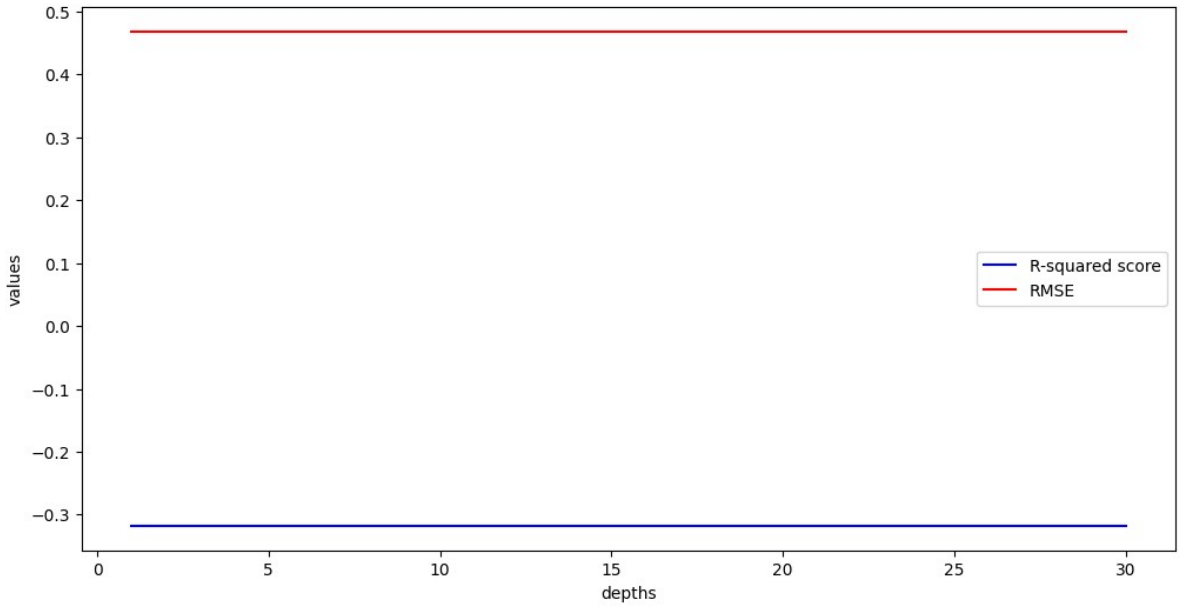
In [143]:
```
# Instantiate DecisionTreeRegressor
# Set random_state=45
clf = DecisionTreeClassifier(random_state=45)

# Fit the model to training data
clf.fit(X_train, y_train_h1n1)

# Make predictions on the test data
y_pred_tree = clf.predict(X_test)

r_2 = r2_score(y_test_h1n1, y_pred_tree)
mse_ = np.sqrt(mean_squared_error(y_test_h1n1, y_pred_tree))
print(r_2, mse_)
```

```
-0.317486122043805 0.4686609435348844
```

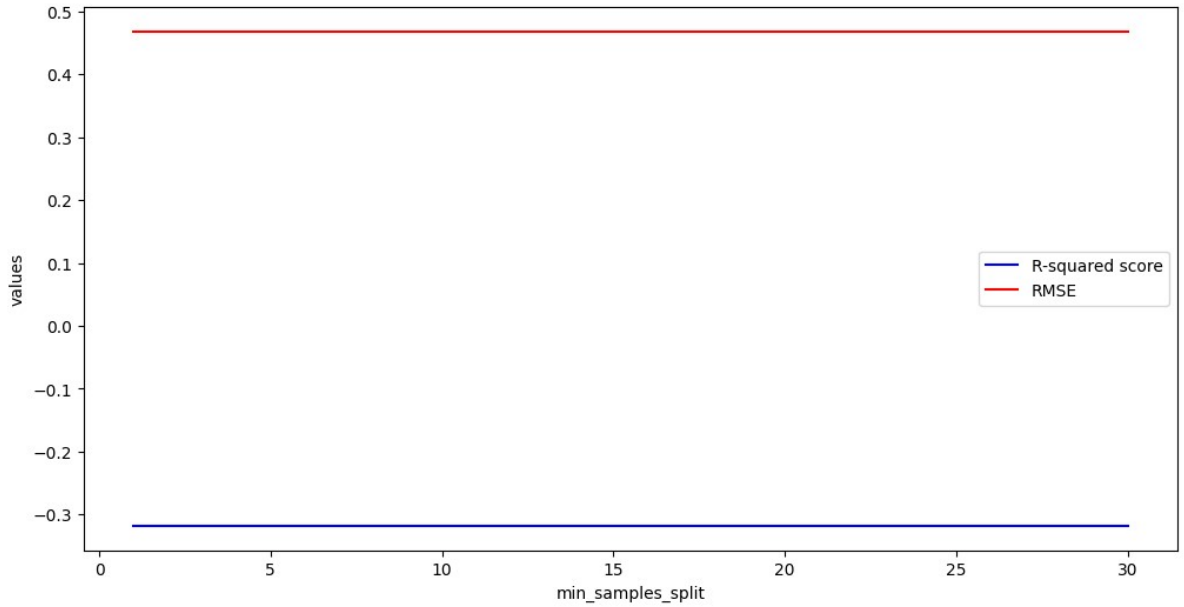
**Hyperparameter tuning**

In [144]:

```
#Find the best tree depth

depths = np.linspace(1, 30, 30)
r2_ = []
mse = []

for depth in depths:
    clf = DecisionTreeClassifier(random_state=45, max_depth=int(depth))
    clf.fit(X_train, y_train_h1n1)
    y_pred = clf.predict(X_test)
    r_2 = r2_score(y_test_h1n1, y_pred_tree)
    mse_ = np.sqrt(mean_squared_error(y_test_h1n1, y_pred_tree))
    r2_.append(r_2)
    mse.append(mse_)

plt.figure(figsize=(12,6))
plt.plot(depths, r2_, 'b', label='R-squared score')
plt.plot(depths, mse, 'r', label='RMSE')
plt.ylabel('values')
plt.xlabel('depths')
plt.legend()
plt.show()
```



In [145]:
```
#minimum sample splits
min_samples_split = np.linspace(0.1, 1.0, 30, endpoint=True)
r2_ = []
mse = []

for min in min_samples_split:
    clf = DecisionTreeClassifier(random_state=45, min_samples_split=min)
    clf.fit(X_train, y_train_h1n1)
    y_pred = clf.predict(X_test)
    r_2 = r2_score(y_test_h1n1, y_pred_tree)
    mse_ = np.sqrt(mean_squared_error(y_test_h1n1, y_pred_tree))
    r2_.append(r_2)
    mse.append(mse_)

plt.figure(figsize=(12,6))
plt.plot(depths, r2_, 'b', label='R-squared score')
plt.plot(depths, mse, 'r', label='RMSE')
plt.ylabel('values')
plt.xlabel('min_samples_split')
plt.legend()
plt.show()
```




Since our graphs' output are all straight lines, the best fit would be any number.
So we will just go with the **vanilla** model.

In [146]:
```
y_pred_tree = clf.predict(X_test)

# Calculate the r2 score between 'y_true' and 'y_predict'
r2_ = r2_score(y_test_h1n1, y_pred_tree)
# Calculate the root mean squared error between 'y_true' and 'y_predict'
mse = np.sqrt(mean_squared_error(y_test_h1n1, y_pred_tree))
# Return the score
print(r2_, mse)
```

```
-0.19022893980093758 0.4454520796031409
```

In [147]:
```
#print('Accuracy: ', accuracy_score(y_test_h1n1, y_pred_tree))
acc_tree1 = accuracy_score(y_test_h1n1, y_pred_tree)
print(acc_tree1)
```

```
0.801572444777237
```

In [148]:
```
# Calculate the r2 score between 'y_true' and 'y_predict'
r2_ = r2_score(y_test_seas, y_pred_tree)
# Calculate the root mean squared error between 'y_true' and 'y_predict'
mse = np.sqrt(mean_squared_error(y_test_seas, y_pred_tree))
# Return the score
print(r2_, mse)
```

```
-0.5860625207268342 0.6274832111952962
```

In [149]:
```
#print('Accuracy: ', accuracy_score(y_test_seas, y_pred_tree))
acc_tree2 = accuracy_score(y_test_seas, y_pred_tree)
print(acc_tree2)
```

```
0.6062648196680395
```

In [150]:
```
tree_acc = (acc_tree1 + acc_tree2)/2
tree_acc
```

Out[150]:
```
0.7039186322226383
```

Our model got an accuracy of:

H1N1 vaccine Accuracy score: **80.15%**
Seasonal Flu vaccine Accuracy score: **60.62%**
Final Accuracy score: **70.39%**

## Using entropy

In [151]:
```
#Instatiating the model with criterion='entropy' and max_depth=3
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train_h1n1)

#Predict the response for test dataset
y_pred_clf = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
entropy1 = accuracy_score(y_test_h1n1, y_pred_clf)
entropy1
```

Out[151]:
```
0.8251591164357919
```

In [152]:
```
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train_seas)

#Predict the response for test dataset
y_pred_clf2 = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
entropy2 = accuracy_score(y_test_seas, y_pred_clf2)
entropy2
```

Out[152]: 0.7349307375514789

In [153]:
```
tree_acc_final = (entropy1 + entropy2)/2
tree_acc_final
```

Out[153]: 0.7800449269936354

In [154]:
```
tree_r_2_ = r2_score(y_test_h1n1, y_pred_clf)
# Calculate the root mean squared error between 'y_true' and 'y_predict'
tree_mse = np.sqrt(mean_squared_error(y_test_h1n1, y_pred_clf))
print(tree_r_2_, tree_mse)
```

```
-0.04874889601327892 0.41813978950131997
```

In [155]:
```
tree_r_2 = r2_score(y_test_seas, y_pred_clf2)
# Calculate the root mean squared error between 'y_true' and 'y_predict'
tree_mse_ = np.sqrt(mean_squared_error(y_test_seas, y_pred_clf2))
print(tree_r_2, tree_mse_)
```

```
-0.06776443550674971 0.5148487762911758
```

In [156]:
```
tree_pre = precision_score(y_pred_clf, y_test_h1n1)
tree_pre
```

Out[156]: 0.33293978748524206

In [157]:
```
tree_rec = recall_score(y_pred_clf, y_test_h1n1)
tree_rec
```

Out[157]: 0.6754491017964072

In [158]:
```
tree_pre2 = precision_score(y_pred_clf2, y_test_seas)
tree_pre2
```

Out[158]: 0.6180877145192045

In [159]:
```
tree_rec2 = recall_score(y_pred_clf2, y_test_seas)
tree_rec2
```

Out[159]: 0.7586091608157807

Our entropy model got an accuracy of:

H1N1 vaccine Accuracy score: **82.52%**
Seasonal Flu vaccine Accuracy score: **73.49%**
Final Accuracy score: **78.00%**

# 7.3.4 Random Forest

Documentation (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)
In this model i will carry out hyperparameters tuning to find the best fit for our model.

In [161]:
```
#Instatiating the model
forest = RandomForestRegressor()

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 50, num = 1

# Number of features to consider at every split
max_features = ['auto', 'sqrt']

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)

# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]

# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]

# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [10, 14, 18, 23, 27, 32, 36, 41, 45, 50], 'max_features': ['
auto', 'sqrt'], 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, N
one], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootst
rap': [True, False]}
```

In [162]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune

forest= RandomForestRegressor()

# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
forest_random = RandomizedSearchCV(estimator = forest , param_distributior
                                   n_iter = 100, cv = 3, verbose=2, random_sta

# Fit the random search model
forest_random.fit(X_train, y_train_h1n1)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

Out[162]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [10, 14, 18, 23, 27,
32,
                                                         36, 41, 45, 50]},
                   random_state=42, verbose=2)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [163]:
```
forest_random.best_params_
```

Out[163]:
```
{'n_estimators': 50,
 'min_samples_split': 5,
 'min_samples_leaf': 4,
 'max_features': 'sqrt',
 'max_depth': None,
 'bootstrap': False}
```

In [164]:
```
# Instantiate and fit a RandomForestClassifier
forest1 = RandomForestClassifier(n_estimators = 50, min_samples_split = 5,
                                 min_samples_leaf = 4, max_features = 'sqr
forest1.fit(X_train, y_train_h1n1)
ytrain_pred = forest1.predict(X_test)

# Training accuracy score
print('Train:', forest1.score(X_train, y_train_h1n1))
print()
print('test', forest1.score(X_test, y_test_h1n1))

forest_acc = forest1.score(X_test, y_test_h1n1)
forest_acc
```

```
Train: 0.9545843586177383

test 0.8502433545488581
```

Out[164]:
```
0.8502433545488581
```

In [165]:
```
# Instantiate and fit a RandomForestClassifier
forest1 = RandomForestClassifier(n_estimators = 50, min_samples_split = 5,
                                 min_samples_leaf = 4, max_features = 'sqr
forest1.fit(X_train, y_train_seas)

ytrain_pred2 = forest1.predict(X_test)

# Training accuracy score
print('Train:', forest1.score(X_train, y_train_seas))
print()
print('test', forest1.score(X_test, y_test_seas))

forest_acc2 = forest1.score(X_test, y_test_seas)
forest_acc2
```

```
Train: 0.960468599550658

test 0.7847248221639835
```

Out[165]:
```
0.7847248221639835
```

In [166]:
```
final_forest = (forest_acc + forest_acc2)/2
final_forest
```

Out[166]:
```
0.8174840883564208
```

In [167]:
```
forest_pre = precision_score(ytrain_pred, y_test_h1n1)
forest_pre
```

Out[167]:
```
0.4510035419126328
```

In [168]:
```
forest_pre2 = precision_score(ytrain_pred2, y_test_h1n1)
forest_pre2
```

Out[168]:
```
0.71900826446281
```

In [169]:
```
forest_rec = recall_score(ytrain_pred, y_test_seas)
forest_rec
```

Out[169]: 0.8152804642166345

In [170]:
```
forest_rec2 = recall_score(ytrain_pred2, y_test_seas)
forest_rec2
```

Out[170]: 0.7648339684267828

Our Random Forest model got an accuracy of:

H1N1 vaccine Accuracy score: **85.02%**
Seasonal Flu vaccine Accuracy score: **78.47%**
Final Accuracy score: **81.74%**

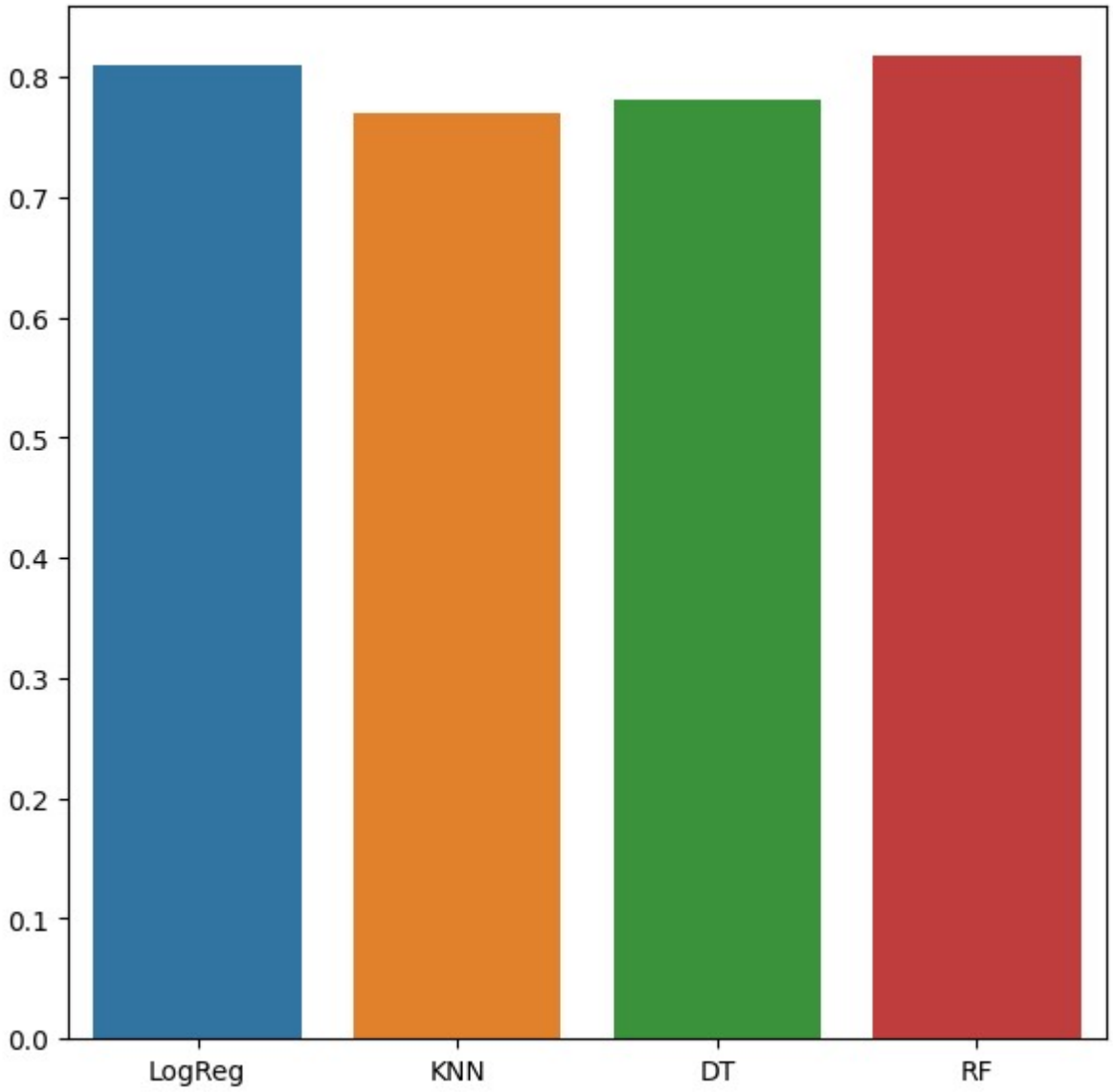
# 7.4 Models Output

In [175]:
```
#Creating a dataframe with the results from all the models
data = {'Model':["Logistic Regression", "KNN", "Decision Tree", 'Random Fo
        'Accuracy H1N1': [log_acc, knn_acc, entropy1, forest_acc],
        'Accuracy Seasonal':[log_acc2, knn_acc2, entropy2, forest_acc2],
        'Final Accuracy':[log_acc_final, acc_knn, tree_acc_final, final_fo
        'Precision H1N1':[log_pre, knn_pre, tree_pre, forest_pre],
        'precision Seasonal':[log_pre2, knn_pre2, tree_pre2, forest_pre2],
        'Recall H1N1':[log_rec, knn_rec, tree_rec, forest_rec],
        'Recall seasonal':[log_rec2, knn_rec2, tree_rec2, forest_rec2]
       }
output = pd.DataFrame(data)

output
```

Out[175]:

| | Model | Accuracy H1N1 | Accuracy Seasonal | Final Accuracy | Precision H1N1 | precision Seasonal | Recall H1N1 | Recall seasonal |
|---|---|---|---|---|---|---|---|---|
| **0** | Logistic Regression | 0.838138 | 0.780981 | 0.809559 | 0.687088 | 0.767747 | 0.430342 | 0.748297 |
| **1** | KNN | 0.813303 | 0.727318 | 0.770311 | 0.640227 | 0.712043 | 0.266824 | 0.679651 |
| **2** | Decision Tree | 0.825159 | 0.734931 | 0.780045 | 0.332940 | 0.618088 | 0.675449 | 0.758609 |
| **3** | Random Forest | 0.850243 | 0.784725 | 0.817484 | 0.451004 | 0.719008 | 0.815280 | 0.764834 |

In [174]:
```
# visualizing the test accuracy results
model_names = ["LogReg","KNN","DT", 'RF']
plt.figure(figsize = (7, 7))
sns.barplot(x = model_names, y = data['Final Accuracy'])
plt.show()
```




Logistic Regression Accuracy score: **80.95%**
KNN Accuracy score: **77.03%**
Decision Trees Accuracy score: **78.00%**
Random Forest Accuracy score: **81.74%**

The Best model is Random forest with an accuracy of **81.74**.

# 8. Evaluation

Choosing the model that has the best performance, I will instantiate a final model with these

best parameters.

## 8.1 Final model

The final model will be random forest because it did the best in the prediction accuracy.

In [176]:
```
#Loading our datasets
training1 = pd.read_csv('training_set_features.csv')
training2 = pd.read_csv('training_set_labels.csv')
test1 = pd.read_csv('test_set_features.csv')
```

In [177]:
```
#Looking the shape of the training features
training_features.shape
```

Out[177]: (26707, 36)

In [178]:
```
#Looking at the shape of the test features
test_features.shape
```

Out[178]: (26708, 36)

In [179]:
```
#Initializing the numeric colums
numeric_test = test_features.columns[training_features.dtypes != "object"]
print(numeric_test)
```

```
['respondent_id' 'h1n1_concern' 'h1n1_knowledge'
 'behavioral_antiviral_meds' 'behavioral_avoidance' 'behavioral_face_mask'
 'behavioral_wash_hands' 'behavioral_large_gatherings'
 'behavioral_outside_home' 'behavioral_touch_face' 'doctor_recc_h1n1'
 'doctor_recc_seasonal' 'chronic_med_condition' 'child_under_6_months'
 'health_worker' 'health_insurance' 'opinion_h1n1_vacc_effective'
 'opinion_h1n1_risk' 'opinion_h1n1_sick_from_vacc'
 'opinion_seas_vacc_effective' 'opinion_seas_risk'
 'opinion_seas_sick_from_vacc' 'household_adults' 'household_children']
```

In [180]:
```
#initializing the non numeric columns
non_numeric_test = test_features.columns[training_features.dtypes == 'obje
print(non_numeric_test)
```

```
['age_group' 'education' 'race' 'sex' 'income_poverty' 'marital_status'
 'rent_or_own' 'employment_status' 'hhs_geo_region' 'census_msa'
 'employment_industry' 'employment_occupation']
```

In [181]:
```
#Scaling of our numeric columns on the test teatures
test_features[numeric_test] = scaler.fit_transform(test_features[numeric_t
```

In [182]:
```
#Encoding the non numeric columns in the test features column
test_features[non_numeric_test] = ordinal.fit_transform(test_features[non_
test_features
```

Out[182]:

| | respondent_id | h1n1_concern | h1n1_knowledge | behavioral_antiviral_meds | behavioral_avoi |
|---|---|---|---|---|---|
| **0** | -1.731986 | 0.418123 | 1.194987 | -0.228896 | 0.6 |
| **1** | -1.731856 | -0.691385 | -0.433154 | -0.228896 | -1.6 |
| **2** | -1.731727 | 0.418123 | 1.194987 | -0.228896 | -1.6 |
| **3** | -1.731597 | -0.691385 | -0.433154 | -0.228896 | -1.6 |
| **4** | -1.731467 | 1.527632 | -0.433154 | 4.381753 | 0.6 |
| **...** | ... | ... | ... | ... | |
| **26703** | 1.731467 | -0.691385 | -0.433154 | -0.228896 | 0.6 |
| **26704** | 1.731597 | 1.527632 | -0.433154 | -0.228896 | 0.6 |
| **26705** | 1.731727 | -1.800894 | -0.433154 | -0.228896 | -1.6 |
| **26706** | 1.731856 | 1.527632 | -0.433154 | -0.228896 | 0.6 |
| **26707** | 1.731986 | 0.418123 | -0.433154 | -0.228896 | -1.6 |

26708 rows × 36 columns

In [183]:
```
#Using Random Forest
#Instatiating the model
final_rf = RandomForestClassifier(n_estimators = 50, min_samples_split = 5
                                  min_samples_leaf = 4, max_features = 'sqr
#Fitting the model.
final_rf.fit(X_train, y_train_h1n1)
```

Out[183]:
```
RandomForestClassifier(bootstrap=False, min_samples_leaf=4, min_samples_split
=5,
                       n_estimators=50)
```
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [184]:
```
#Probability prediction on the h1n1 vaccine
final_pred_h1n1 = final_rf.predict_proba(test_features)
final_pred_h1n1
```

Out[184]:
```
array([[0.82504762, 0.17495238],
       [0.95857143, 0.04142857],
       [0.69947941, 0.30052059],
       ...,
       [0.7517316 , 0.2482684 ],
       [0.87149567, 0.12850433],
       [0.45199206, 0.54800794]])
```

In [187]:
```
#creating a dataframe from the probabilities predictions
h1n1 = pd.DataFrame(
    {
        "h1n1_vaccine": final_pred_h1n1[:, 1],
    },
    index = test1.respondent_id
)
print("h1n1.shape:", h1n1.shape)
h1n1.head()
```

```
h1n1.shape: (26708, 1)
```

Out[187]:

| respondent_id | h1n1_vaccine |
|---|---|
| **26707** | 0.174952 |
| **26708** | 0.041429 |
| **26709** | 0.300521 |
| **26710** | 0.668857 |
| **26711** | 0.365556 |

In [188]:
```
#Probability prediction of the seasonal flu vaccine
final_rf.fit(X_train, y_train_seas)
final_pred_seas = final_rf.predict_proba(test_features)
```

In [189]:
```
#Dataframe creation for our probability predictions
seas = pd.DataFrame(
    {
        #"seasonal_vaccine": final_pred_h1n1[:, 1],
        "seasonal_vaccine": final_pred_seas[:, 1],
    },
    index = test1.respondent_id
)
print("seas.shape:", seas.shape)
seas.head()
```

```
seas.shape: (26708, 1)
```

Out[189]:

| respondent_id | seasonal_vaccine |
|---|---|
| **26707** | 0.323349 |
| **26708** | 0.069968 |
| **26709** | 0.735802 |
| **26710** | 0.914429 |
| **26711** | 0.519143 |

In [191]:
```
#Joining the two probability dataframes
sub = h1n1.join(seas, on='respondent_id')
sub['h1n1_vaccine'] = sub['h1n1_vaccine']
sub['seasonal_vaccine'] = sub['seasonal_vaccine']
#calling the dataframe
sub
```

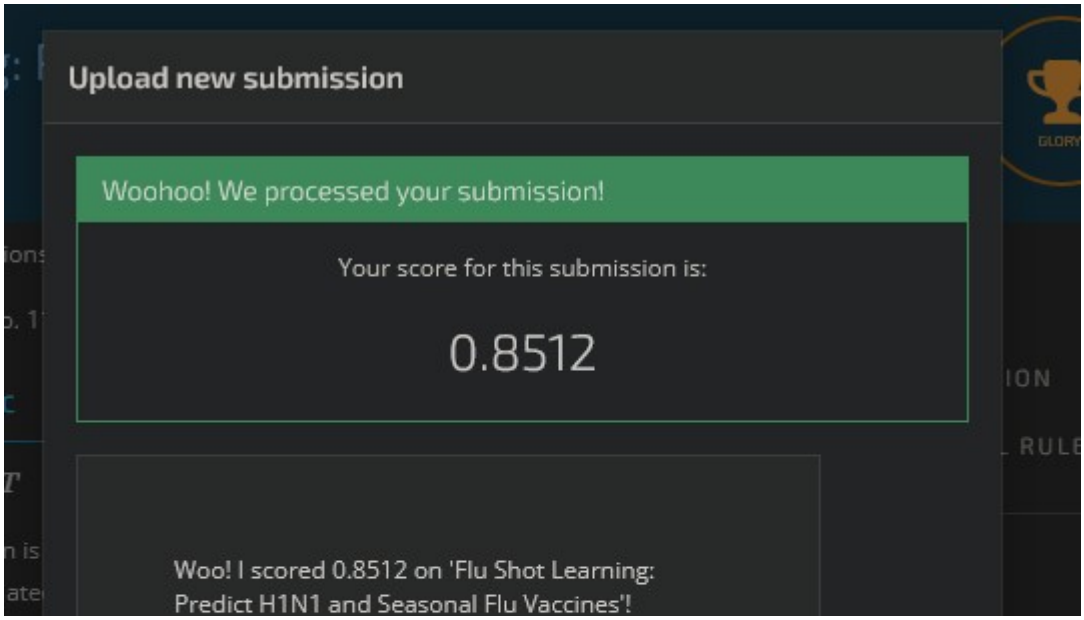
Out[191]:

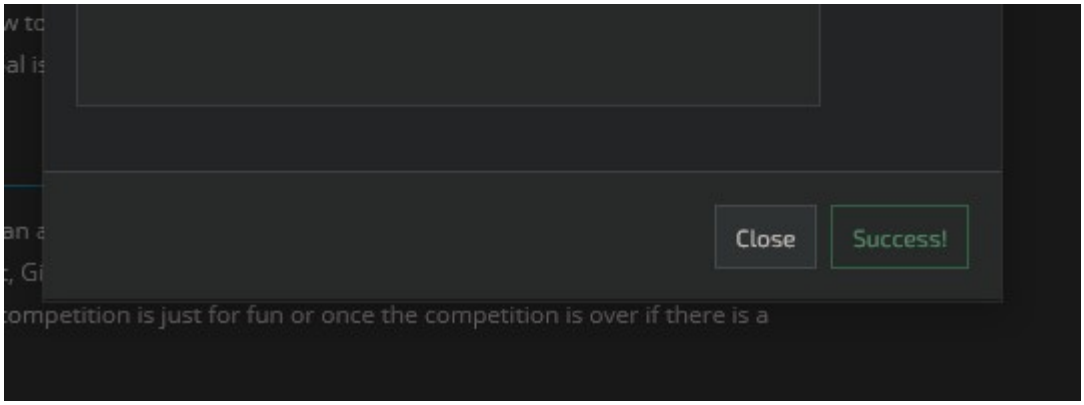
| respondent_id | h1n1_vaccine | seasonal_vaccine |
|---|---|---|
| **26707** | 0.174952 | 0.323349 |
| **26708** | 0.041429 | 0.069968 |
| **26709** | 0.300521 | 0.735802 |
| **26710** | 0.668857 | 0.914429 |
| **26711** | 0.365556 | 0.519143 |
| **...** | ... | ... |
| **53410** | 0.309095 | 0.543540 |
| **53411** | 0.157238 | 0.271190 |
| **53412** | 0.248268 | 0.296143 |
| **53413** | 0.128504 | 0.426405 |
| **53414** | 0.548008 | 0.612071 |

26708 rows × 2 columns

In [192]:
```
#Saving the dataframe as a .csv file
sub.to_csv('submission_format.csv')
```

With our model's accuracy of 81.74% our metrics of success we can say our model did quite well.
Also from the competitions submission process, the model got a 85.12%.






# 9. Conclusion

1. From this project it clear to conclude that:

   - The sex,
   - Age group,
   - Education level,
   - Location,
   - Residence,
   - Chronic illnesses
   - And Race

   All affect whether one would receive the vaccines.

2. Most people took the Seasonal Flu vaccine compared to the H1N1 vaccine.
3. Through the model we can clearly predict the intake of the vaccines and with more tuning it can predict the intake more perfectly.

# 10. Recommendation

- Through the model predictions we can recommend the governments to do more public education on the importance of the vaccines and mostly focus on the 34-44 years age group.
- The vaccines should be made more available and accessible to all people and most importantly those with chronic illnesses.