Predict H1N1 and Seasonal Flu Vaccines

Overview

predict whether people got H1N1 and seasonal flu using information they shared about their backgrounds, opinions and behaviours

Problem statement

The goal of this study is to predict how likely individuals are to recieve their H1N1 and seasonal flu vaccines,whether individuals received the H1N1 and seasonal vaccines based on information about their backgrounds,opinions and health behaviours. Understanding the factors influencing vaccination decisions can inform strategies to improve vaccine uptake in future public initiaves. The study involves analyzing this survey data to uncover relationships between individual characteristics and vaccination patterns altimately develop predictive models to classify

###

Objectives

1. Predict vaccination status - Develop models to predict whether individuals recieved the H1N1 and seasonal flu vaccines based on their demogrpahic,social and behavioral data
2. Identify key factors: Determine the most influential factors affecting vaccination decisions such as socioecnomic status, health behaviours or preceptions of vaccine effectiveness
3. Enhance public health insights - Provide actionable insights to help public health officials design targeted campaigns to improve vaccine uptake.

Python Data cleaning and Analysis

1a) Importing libraries

In [1968]:

```
#importing Libraries we need
#common libraries
#import the pandas library
import pandas as pd
#import numpy library
import numpy as np
#import the seaborn library
import seaborn as sns
#import matplotlib library
import matplotlib.pyplot as plt

#scaling
from sklearn.preprocessing import MinMaxScaler

#one hot encoding libraries
from sklearn.preprocessing import OneHotEncoder

#Feature selection
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

#modelling
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

#Evaluation metrics
from sklearn.metrics import roc_curve,auc,mean_squared_error,r2_score


```

b)Reading datasets from our csv files

In [1969]:

```
#Read data from the csv file and create dataframes we need
training_data = pd.read_csv('training_set_features.csv')
test_data = pd.read_csv('test_set_features.csv')
training_label =pd.read_csv('training_set_labels.csv')
```

c)Previewing our datasets

In [1970]:
```
#training dataset
training_data.head()
```

Out[1970]:

	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	3	1.0	1.0	0.0	1.
4	4	2.0	1.0	0.0	1.

5 rows × 36 columns

In [1971]:
```
training_label.head()
```

Out[1971]:

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

In [1972]:
```
test_data.head()
```

Out[1972]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidanc
0	26707	2.0	2.0	0.0	1.
1	26708	1.0	1.0	0.0	0.
2	26709	2.0	2.0	0.0	0.
3	26710	1.0	1.0	0.0	0.
4	26711	3.0	1.0	1.0	1.

5 rows × 36 columns

d)Combine dataset

In [1973]:
```
#combine data with labels data that has the target features
data = pd.merge(training_data,training_label,how='inner',on='respondent_
data

```

Out[1973]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoi
0	0	1.0	0.0	0.0	
1	1	3.0	2.0	0.0	
2	2	1.0	1.0	0.0	
3	3	1.0	1.0	0.0	
4	4	2.0	1.0	0.0	
...	...	...	...	...	
26702	26702	2.0	0.0	0.0	
26703	26703	1.0	2.0	0.0	
26704	26704	2.0	2.0	0.0	
26705	26705	1.0	1.0	0.0	
26706	26706	0.0	0.0	0.0	

26707 rows × 38 columns

d)Accessing information about our dataset

In [1974]:
```
#getting to know about our dataset by accessing its information
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26707 entries, 0 to 26706
Data columns (total 38 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   respondent_id                26707 non-null  int64  
 1   h1n1_concern                 26615 non-null  float64
 2   h1n1_knowledge               26591 non-null  float64
 3   behavioral_antiviral_meds    26636 non-null  float64
 4   behavioral_avoidance         26499 non-null  float64
 5   behavioral_face_mask         26688 non-null  float64
 6   behavioral_wash_hands        26665 non-null  float64
 7   behavioral_large_gatherings  26620 non-null  float64
 8   behavioral_outside_home      26625 non-null  float64
 9   behavioral_touch_face        26579 non-null  float64
 10  doctor_recc_h1n1             24547 non-null  float64
 11  doctor_recc_seasonal         24547 non-null  float64
 12  chronic_med_condition        25736 non-null  float64
 13  child_under_6_months         25887 non-null  float64
 14  health_worker                25903 non-null  float64
 15  health_insurance             14433 non-null  float64
 16  opinion_h1n1_vacc_effective  26316 non-null  float64
 17  opinion_h1n1_risk            26319 non-null  float64
 18  opinion_h1n1_sick_from_vacc  26312 non-null  float64
 19  opinion_seas_vacc_effective  26245 non-null  float64
 20  opinion_seas_risk            26193 non-null  float64
 21  opinion_seas_sick_from_vacc  26170 non-null  float64
 22  age_group                    26707 non-null  object 
 23  education                    25300 non-null  object 
 24  race                         26707 non-null  object 
 25  sex                          26707 non-null  object 
 26  income_poverty               22284 non-null  object 
 27  marital_status               25299 non-null  object 
 28  rent_or_own                  24665 non-null  object 
 29  employment_status            25244 non-null  object 
 30  hhs_geo_region               26707 non-null  object 
 31  census_msa                   26707 non-null  object 
 32  household_adults             26458 non-null  float64
 33  household_children           26458 non-null  float64
 34  employment_industry          13377 non-null  object 
 35  employment_occupation        13237 non-null  object 
 36  h1n1_vaccine                 26707 non-null  int64  
 37  seasonal_vaccine             26707 non-null  int64  
dtypes: float64(23), int64(3), object(12)
memory usage: 7.9+ MB
```

In [1975]:
```
1 training_label.info()
```

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

e)Accessing Summary statistics about our data

In [1976]:
```
1 # statistics for int and float objects
2 data.describe()
```

Out[1976]:

	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 × 26 columns

In [1977]:
```
1 #statistics for float objects
2 data.describe(include='O')
```

Out[1977]:

	age_group	education	race	sex	income_poverty	marital_status	rent_or_own	emp
count	26707	25300	26707	26707	22284	25299	24665	
unique	5	4	4	2	3	2	2	
top	65+ Years	College Graduate	White	Female	<= $75,000, Above Poverty	Married	Own	
freq	6843	10097	21222	15858	12777	13555	18736	

2) Cleaning our Dataset

Perfoming data cleaning procedures below providing a documentation for our action and reaons.Will perform as many data cleaning procedures as we think suitable for the various dimensions of data

In [1978]:
```
#lets do a copy of our dataset first
training = data.copy()
#preview first five rows
training.head()
```

Out[1978]:

	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	3	1.0	1.0	0.0	1.
4	4	2.0	1.0	0.0	1.

5 rows × 38 columns

a)Check for missing values

In [1979]:
```
#Check the sum of missing values per column
training.isna().sum()
```

Out[1979]:
```
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
h1n1_vaccine                     0
seasonal_vaccine                 0
dtype: int64
```

In [1980]:
```
#check missing by percentage
((training.isna().sum()/training.shape[0]) * 100).sort_values(ascending=
```

Out[1980]:
```
employment_occupation          50.436215
employment_industry            49.912008
health_insurance               45.957989
income_poverty                 16.561201
doctor_recc_h1n1                8.087767
doctor_recc_seasonal            8.087767
rent_or_own                     7.645936
employment_status               5.477965
marital_status                  5.272026
education                       5.268282
chronic_med_condition           3.635751
child_under_6_months            3.070356
health_worker                   3.010447
opinion_seas_sick_from_vacc     2.010709
opinion_seas_risk               1.924589
opinion_seas_vacc_effective     1.729884
opinion_h1n1_sick_from_vacc     1.479013
opinion_h1n1_vacc_effective     1.464036
opinion_h1n1_risk               1.452803
household_adults                0.932340
household_children              0.932340
behavioral_avoidance            0.778822
behavioral_touch_face           0.479275
h1n1_knowledge                  0.434343
h1n1_concern                    0.344479
behavioral_large_gatherings     0.325757
behavioral_outside_home         0.307036
behavioral_antiviral_meds       0.265848
behavioral_wash_hands           0.157262
behavioral_face_mask            0.071142
h1n1_vaccine                    0.000000
respondent_id                   0.000000
census_msa                      0.000000
hhs_geo_region                  0.000000
sex                             0.000000
race                            0.000000
age_group                       0.000000
seasonal_vaccine                0.000000
dtype: float64
```

Drop columns with 50 and above missing values

In [1981]:
```
#calculate the percentage of null values per column
null_percentage = (training.isna().sum()/training.shape[0]) * 100
#identify columns with 50 + null values
columns_to_drop = null_percentage[null_percentage>49.9].index
print(f'columns_to_drop: {columns_to_drop}')
#Drop those columns from the dataframe
training.drop(columns=columns_to_drop,inplace=True)
#check missing values again
((training.isna().sum()/training.shape[0]) * 100).sort_values(ascending=
```

```
columns_to_drop: Index(['employment_industry', 'employment_occupation'], dtyp
e='object')
```

Out[1981]:
```
health_insurance             45.957989
income_poverty               16.561201
doctor_recc_h1n1              8.087767
doctor_recc_seasonal          8.087767
rent_or_own                   7.645936
employment_status             5.477965
marital_status                5.272026
education                     5.268282
chronic_med_condition         3.635751
child_under_6_months          3.070356
health_worker                 3.010447
opinion_seas_sick_from_vacc   2.010709
opinion_seas_risk             1.924589
opinion_seas_vacc_effective   1.729884
opinion_h1n1_sick_from_vacc   1.479013
opinion_h1n1_vacc_effective   1.464036
opinion_h1n1_risk             1.452803
household_adults              0.932340
household_children            0.932340
behavioral_avoidance          0.778822
behavioral_touch_face         0.479275
h1n1_knowledge                0.434343
h1n1_concern                  0.344479
behavioral_large_gatherings   0.325757
behavioral_outside_home       0.307036
behavioral_antiviral_meds     0.265848
behavioral_wash_hands         0.157262
behavioral_face_mask          0.071142
h1n1_vaccine                  0.000000
census_msa                    0.000000
respondent_id                 0.000000
hhs_geo_region                0.000000
sex                           0.000000
race                          0.000000
age_group                     0.000000
seasonal_vaccine              0.000000
dtype: float64
```

Deciding how to fill missing values

In [1982]:
```
#fill numerics with median and categorical with mode
for column in training.columns:
    if training[column].dtype=='object':
        training[column].fillna(training[column].mode()[0],inplace=True)
    elif training[column].dtype =='float64':
        training[column].fillna(training[column].median(),inplace=True)
```

In [1983]:
```
#techeck misssing
training.isna().sum()
```

Out[1983]:
```
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
h1n1_vaccine                 0
seasonal_vaccine             0
dtype: int64
```

b)Check for duplicates

In [1984]:
```
1 training.duplicated().sum() #They are no duplicates in our dataset
```

Out[1984]:
```
0
```

c)Remove unnecessary columns

In [1985]:
```
1 # hhs_geo_region,household_children,household_adults does not give much
2 training.drop(columns=['hhs_geo_region','household_children','household_
3
```

c)employment_status column

In [1986]:
```
1 ### combine not in labour with unemployed in employment_status column
2 training.replace('Not in Labor Force','Unemployed',inplace=True)
```

3)EDA and Analysis

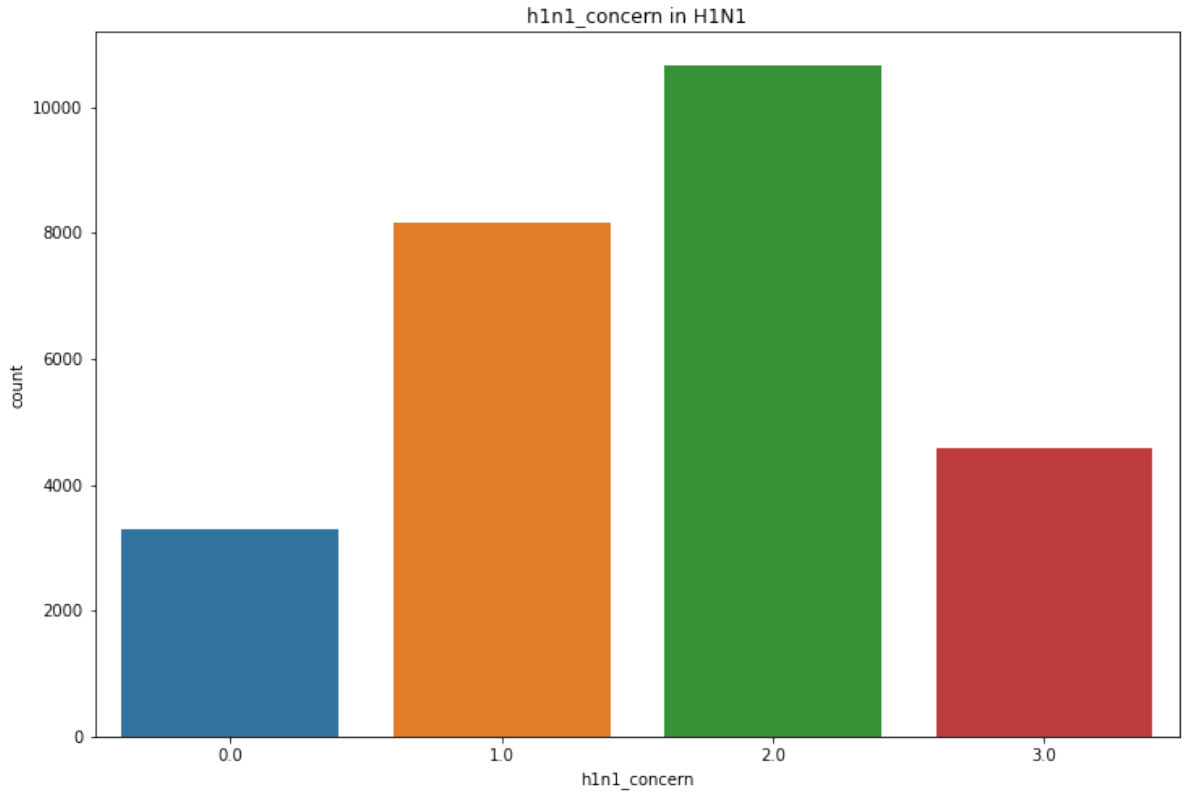
In [1987]:
```
1 training.columns
```

Out[1987]:
```
Index(['respondent_id', 'h1n1_concern', 'h1n1_knowledge',
       'behavioral_antiviral_meds', 'behavioral_avoidance',
       'behavioral_face_mask', 'behavioral_wash_hands',
       'behavioral_large_gatherings', 'behavioral_outside_home',
       'behavioral_touch_face', 'doctor_recc_h1n1', 'doctor_recc_seasonal',
       'chronic_med_condition', 'child_under_6_months', 'health_worker',
       'health_insurance', 'opinion_h1n1_vacc_effective', 'opinion_h1n1_ris
k',
       'opinion_h1n1_sick_from_vacc', 'opinion_seas_vacc_effective',
       'opinion_seas_risk', 'opinion_seas_sick_from_vacc', 'age_group',
       'education', 'race', 'sex', 'income_poverty', 'marital_status',
       'rent_or_own', 'employment_status', 'census_msa', 'h1n1_vaccine',
       'seasonal_vaccine'],
      dtype='object')
```

In [1988]:
```
# create functionsfor countplot and value counts to prevent repetition

#countplot
def countplot(x,y):
    fig,ax = plt.subplots(figsize=(12,8))
    sns.countplot(data=training,x=x)
    if y=='H1N1':
        ax.set(xlabel=x,title=f'{x} in H1N1')
    else:
        ax.set(xlabel=x,title=f'{x} in Seasonal flu')

#value_counts
def value_counts(x):
    return training[x].value_counts(normalize=True)
```

1)h1n1_concern

In [1989]:
```
#create a function for countplot to prevent repetition to draw countplot

#h1n1_concern
countplot('h1n1_concern','H1N1')
```



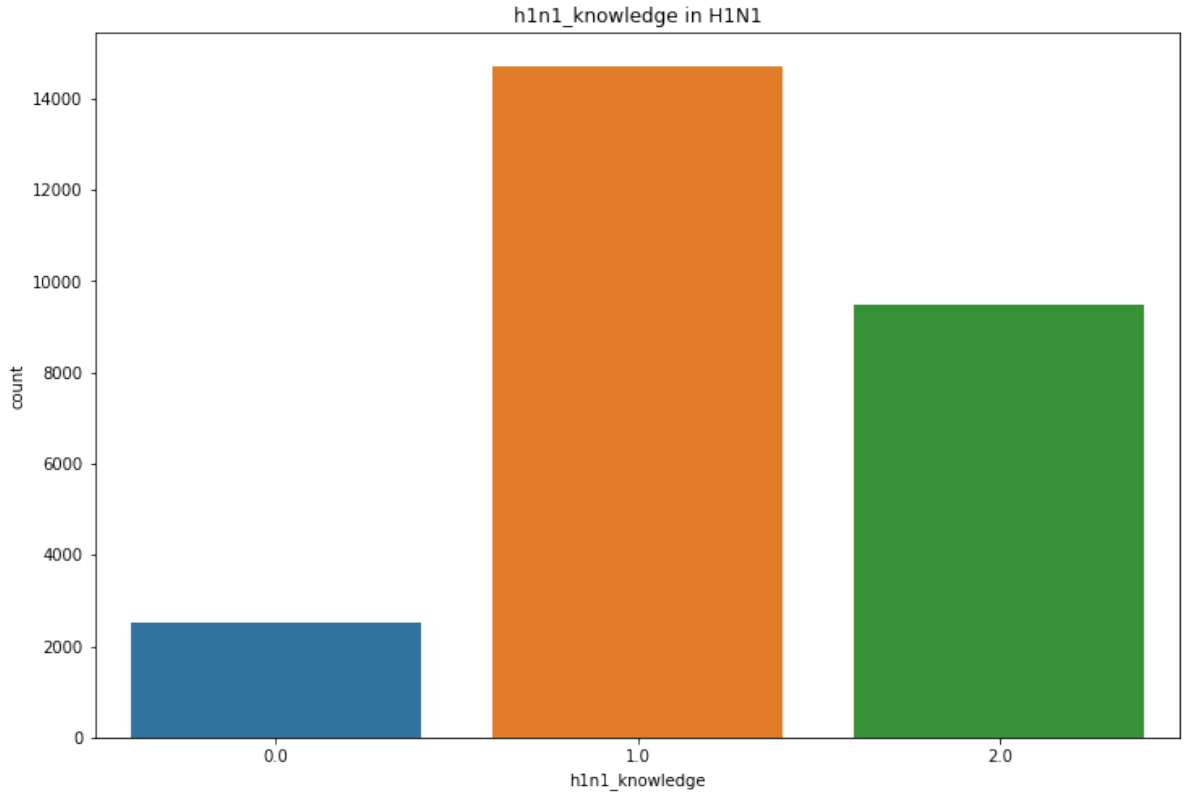
In [1990]:
```
value_counts('h1n1_concern')
```

Out[1990]:
```
2.0    0.399408
1.0    0.305276
3.0    0.171902
0.0    0.123413
Name: h1n1_concern, dtype: float64
```

most people are somewhat concerned with HINI vaccine

2)h1n1_knowledge

In [1991]:
```
1 countplot('h1n1_knowledge','H1N1')
```



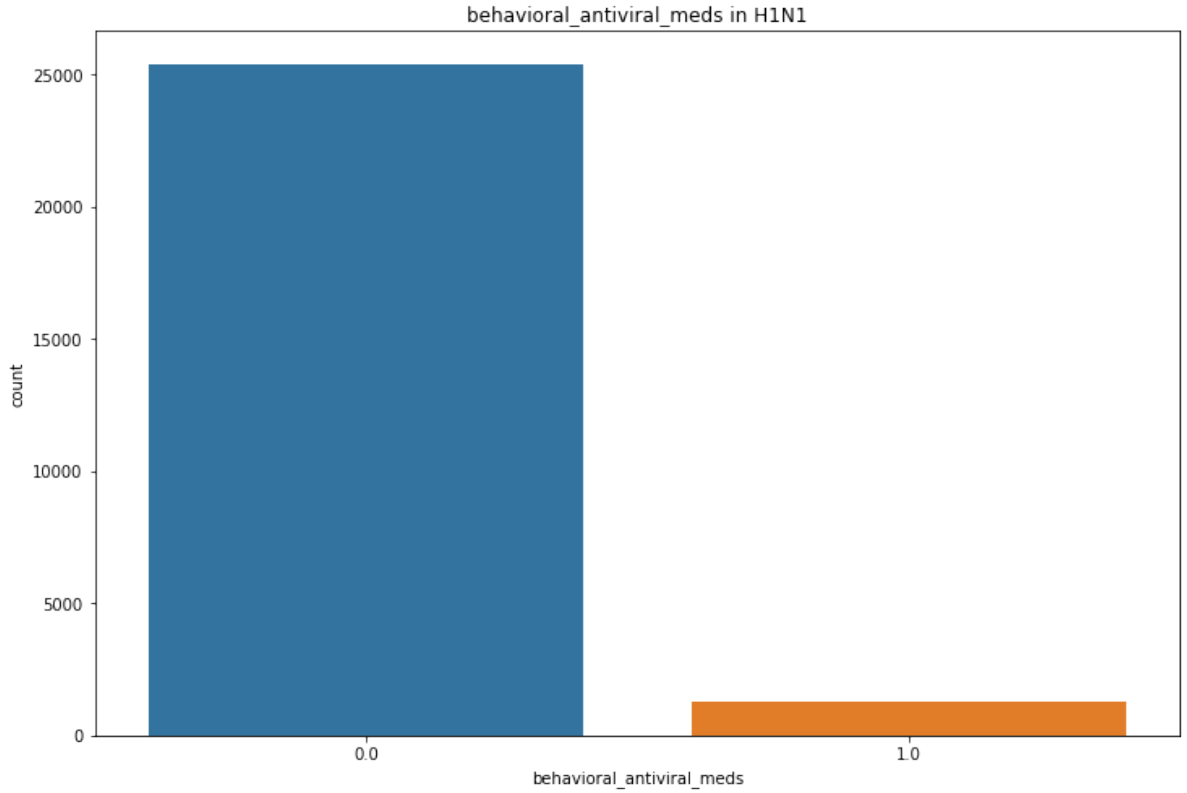
In [1992]:
```
1 value_counts('h1n1_knowledge')
```

Out[1992]:
```
1.0    0.550942
2.0    0.355225
0.0    0.093833
Name: h1n1_knowledge, dtype: float64
```

most people have little knowledge on the HINI vaccine

3) behavioral_antiviral_meds

In [1993]:
```
1 countplot('behavioral_antiviral_meds','H1N1')
```



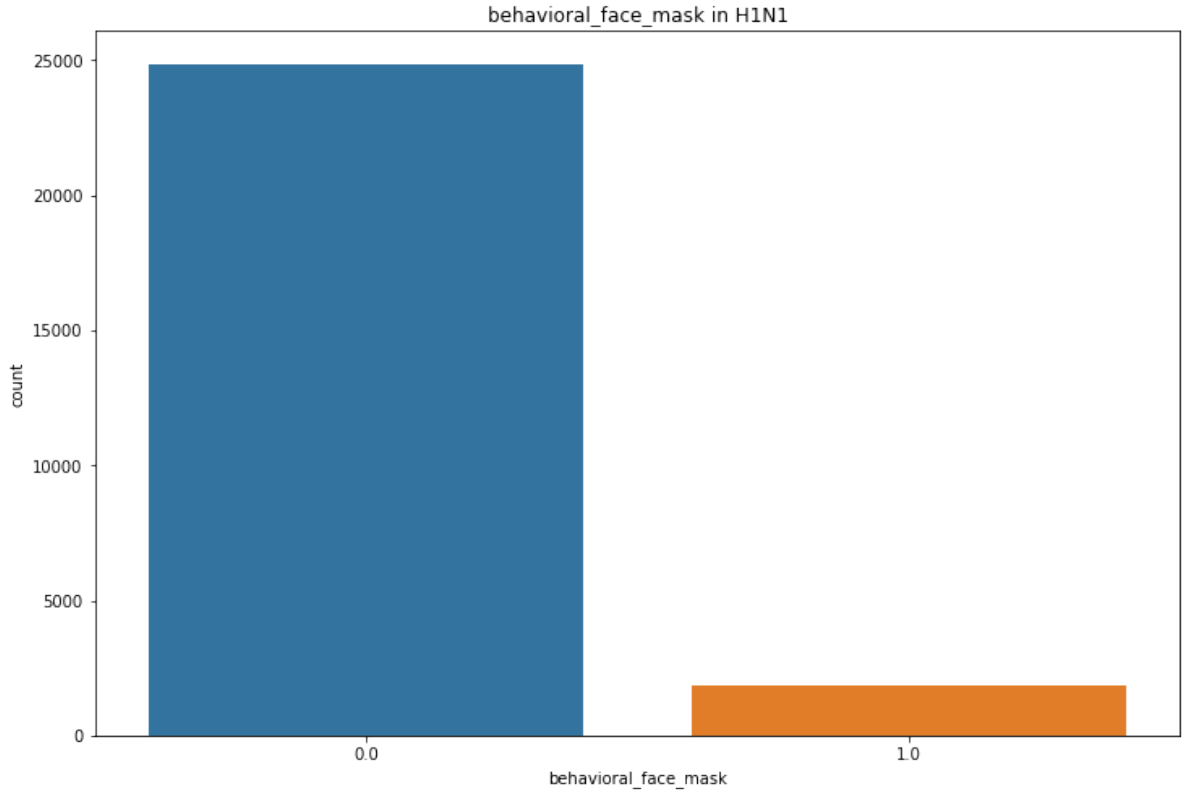
In [1994]:
```
1 value_counts('behavioral_antiviral_meds')
```

Out[1994]:
```
0.0    0.951286
1.0    0.048714
Name: behavioral_antiviral_meds, dtype: float64
```

More that 95% of people have not taken antiviral meds

4)behavioral_face_mask

In [1995]:
```
1 countplot('behavioral_face_mask','H1N1')
```



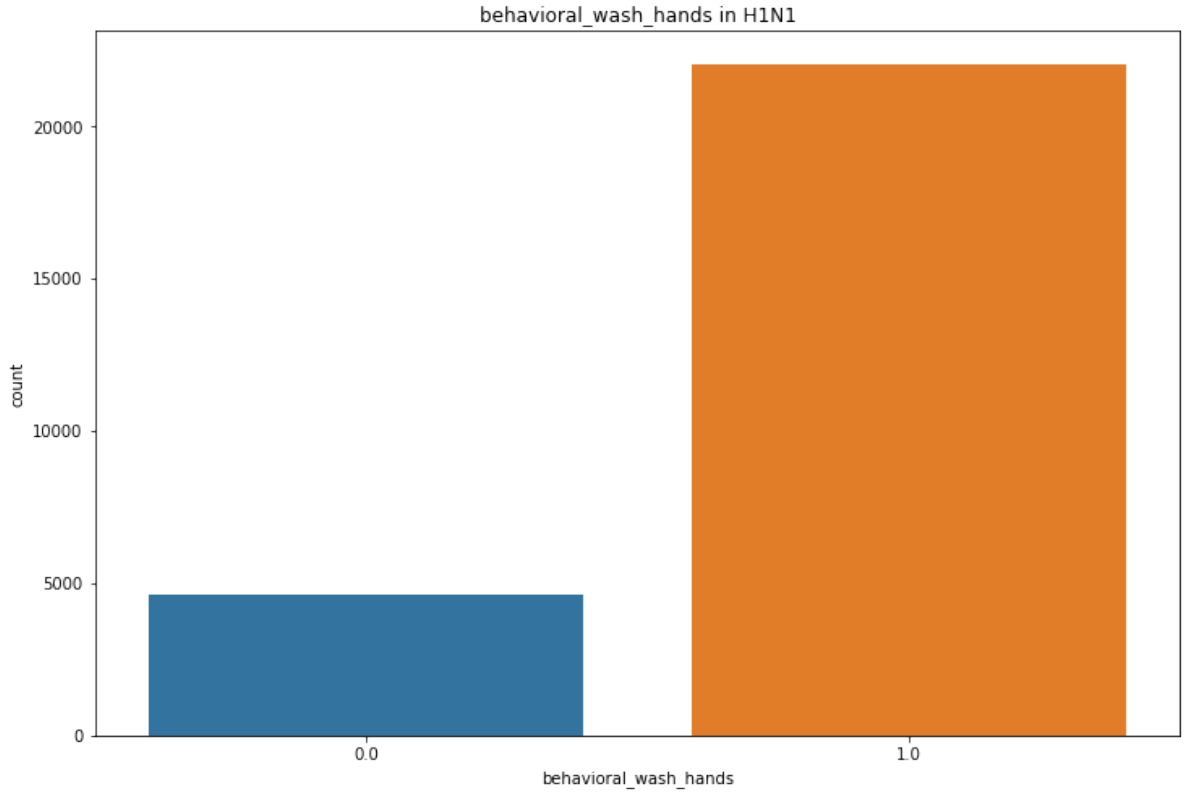
In [1996]:
```
1 value_counts('behavioral_face_mask')
```

Out[1996]:
```
0.0    0.931067
1.0    0.068933
Name: behavioral_face_mask, dtype: float64
```

More than 93 % of people do not wear face mask

5)behavioral_wash_hands

In [1997]:
```
1 countplot('behavioral_wash_hands','H1N1')
```



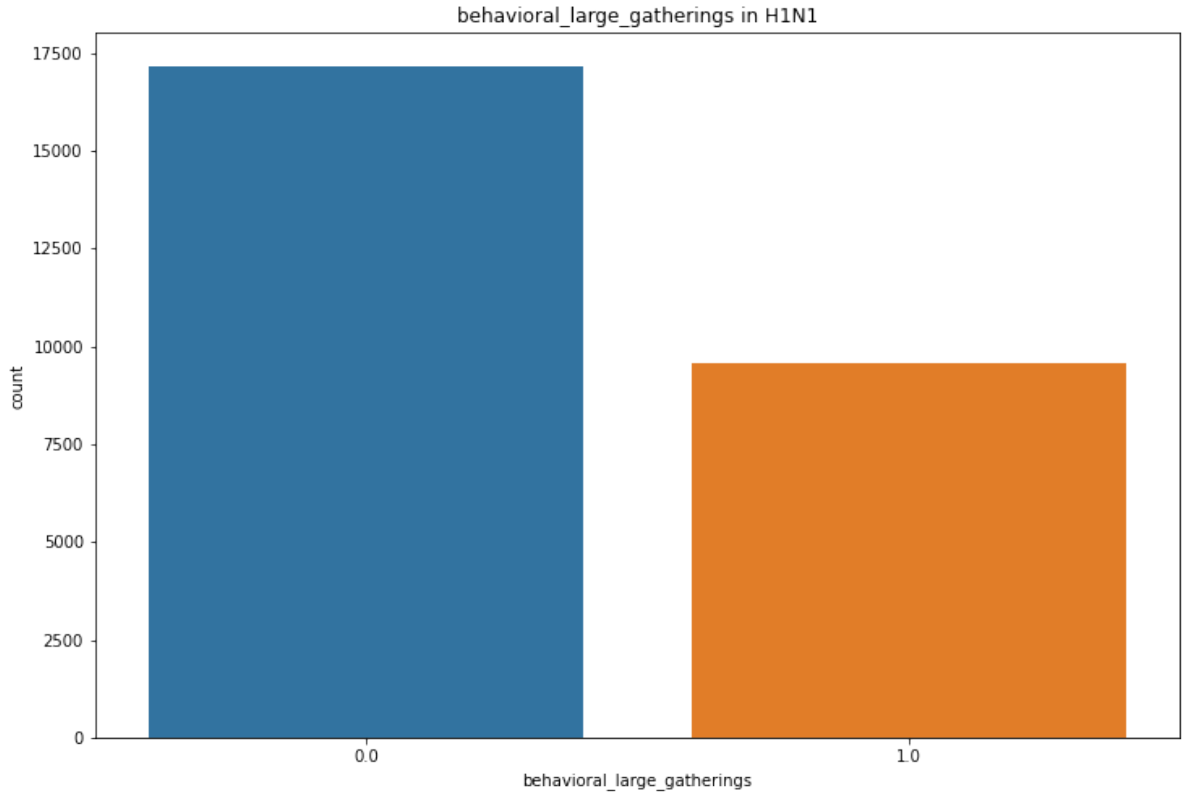
In [1998]:
```
1 value_counts('behavioral_wash_hands')
```

Out[1998]:
```
1.0    0.825888
0.0    0.174112
Name: behavioral_wash_hands, dtype: float64
```

most people(83%) wash hands

6)behavioral_large_gatherings

In [1999]:
```
1 countplot('behavioral_large_gatherings','H1N1')
```



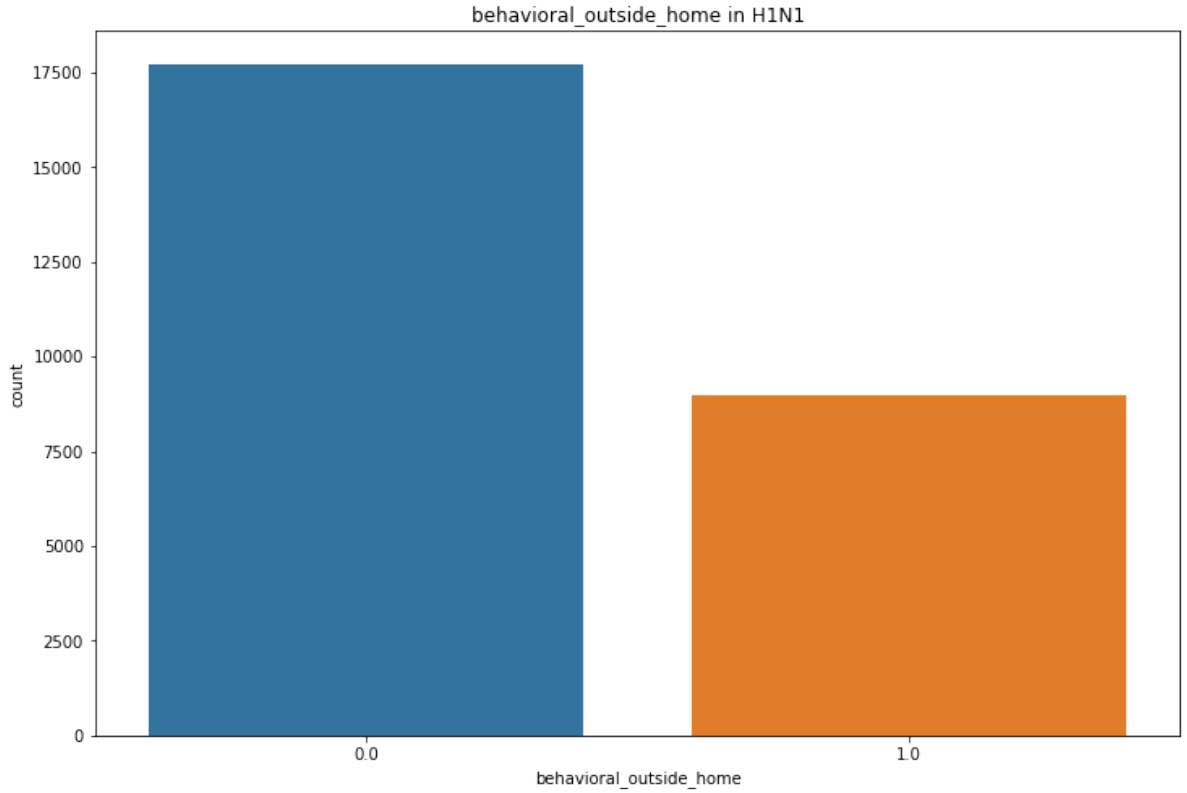
In [2000]:
```
1 value_counts('behavioral_large_gatherings')
```

Out[2000]:
```
0.0    0.642528
1.0    0.357472
Name: behavioral_large_gatherings, dtype: float64
```

most people have not reduced time at large gatherings

7)behavioral_outside_home

In [2001]:
```
1 countplot('behavioral_outside_home','H1N1')
```



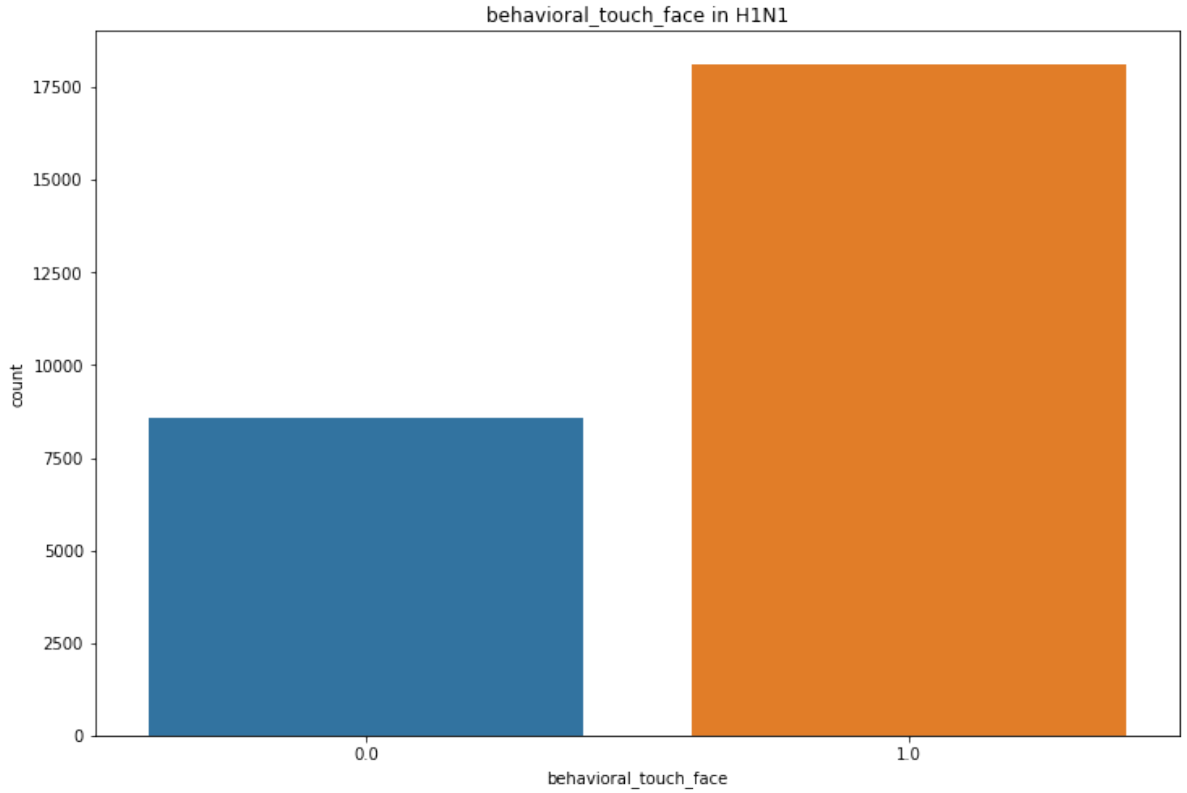
In [2002]:
```
1 value_counts('behavioral_outside_home')
```

Out[2002]:
```
0.0    0.663721
1.0    0.336279
Name: behavioral_outside_home, dtype: float64
```

most people have not reduced contact with people outside their own household

8)behavioral_touch_face

In [2003]:
```
1 countplot('behavioral_touch_face','H1N1')
```



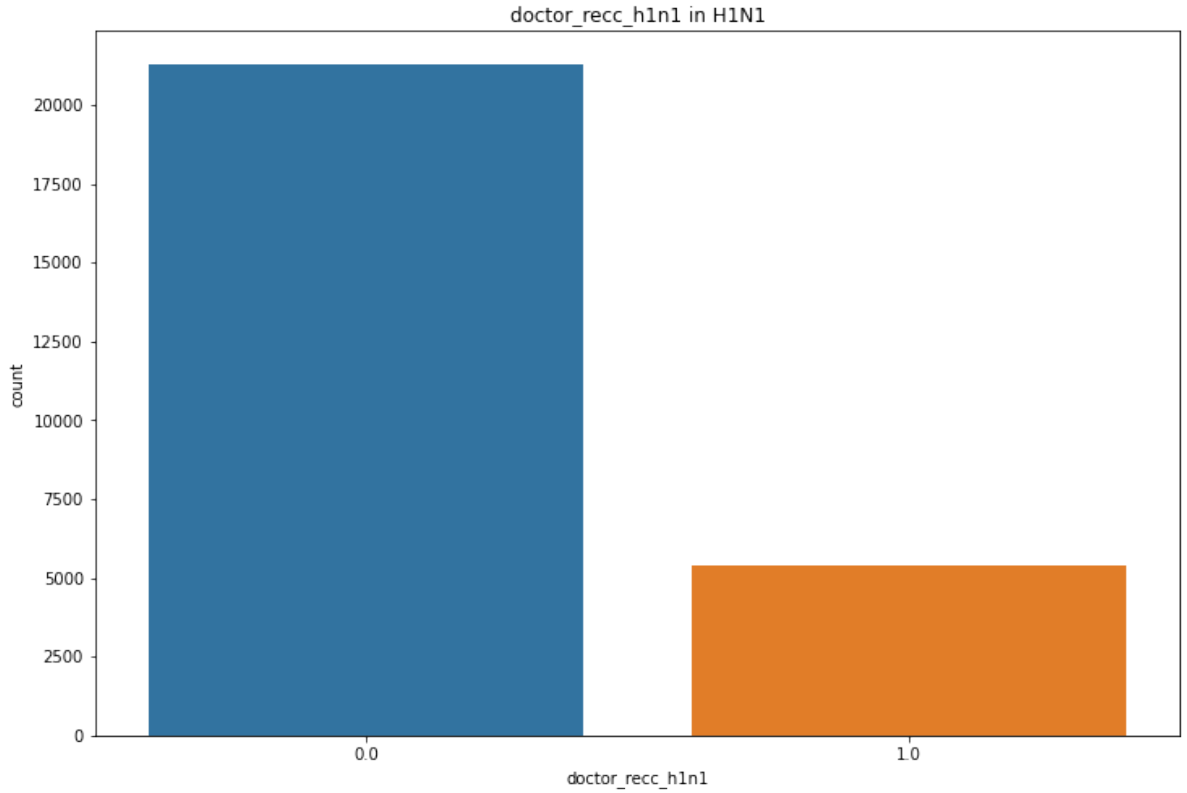
In [2004]:
```
1 value_counts('behavioral_touch_face')
```

Out[2004]:
```
1.0    0.678811
0.0    0.321189
Name: behavioral_touch_face, dtype: float64
```

most people have avoided touching eyes,nose and mouth

9) doctor_recc_h1n1

In [2005]:
```
1 countplot('doctor_recc_h1n1','H1N1')
```



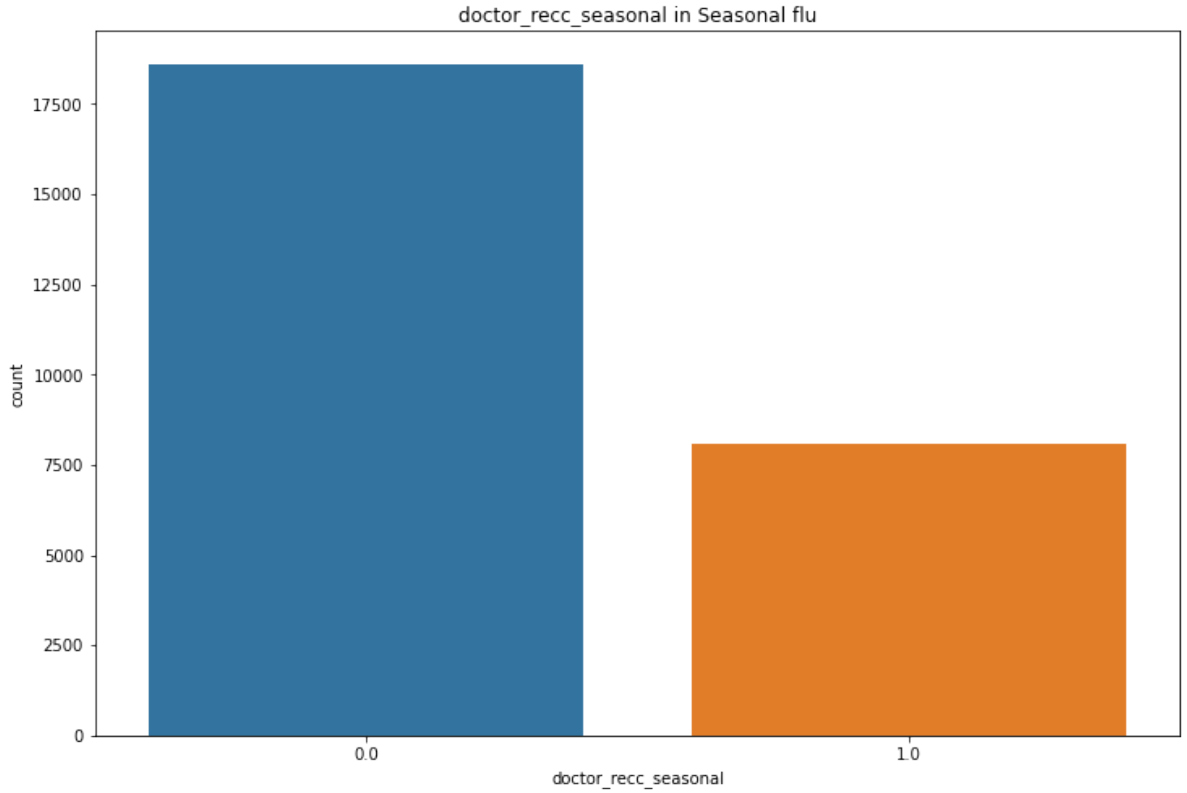
In [2006]:
```
1 value_counts('doctor_recc_h1n1')
```

Out[2006]:
```
0.0    0.797506
1.0    0.202494
Name: doctor_recc_h1n1, dtype: float64
```

Most people had not been recommended the H1N1 vaccine by a doctor

10)doctor_recc_seasonal

In [2007]:
```
1 countplot('doctor_recc_seasonal','seasonal')
```



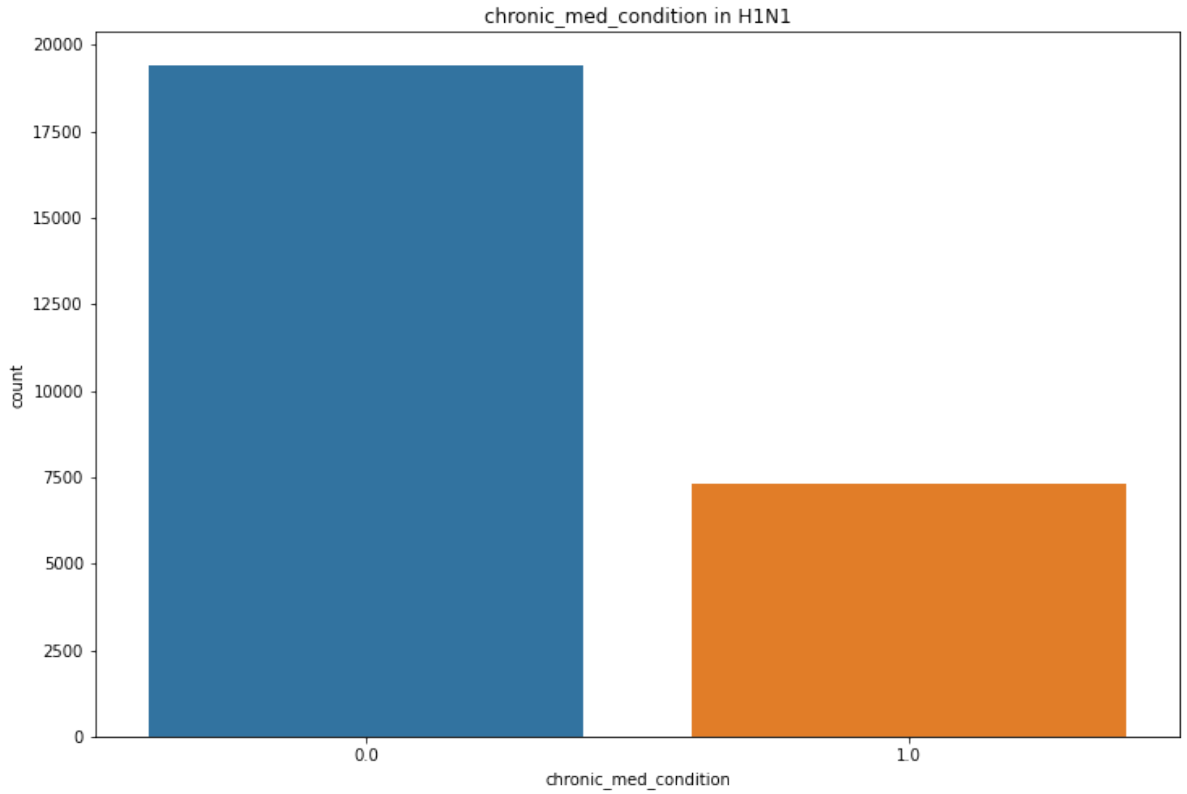
In [2008]:
```
1 value_counts('doctor_recc_seasonal')
```

Out[2008]:
```
0.0    0.696933
1.0    0.303067
Name: doctor_recc_seasonal, dtype: float64
```

most people had not been recommended seasonal flu vaccine by a doctor

11 chronic_med_condition

In [2009]:
```
1
2 countplot('chronic_med_condition','H1N1')
```



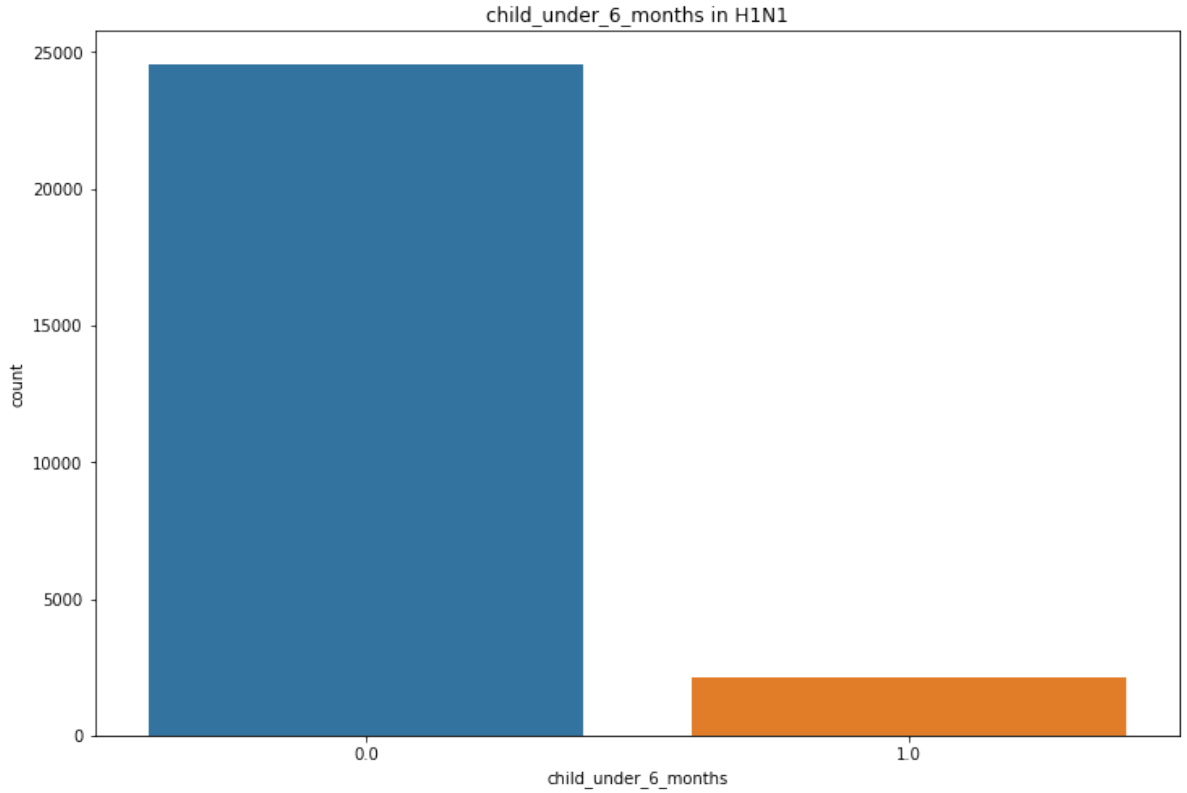
In [2010]:
```
1 value_counts('chronic_med_condition')
```

Out[2010]:
```
0.0    0.727038
1.0    0.272962
Name: chronic_med_condition, dtype: float64
```

most people do not have chronic illness

12. child_under_6_months

In [2011]:
```
1 countplot('child_under_6_months','H1N1')
```



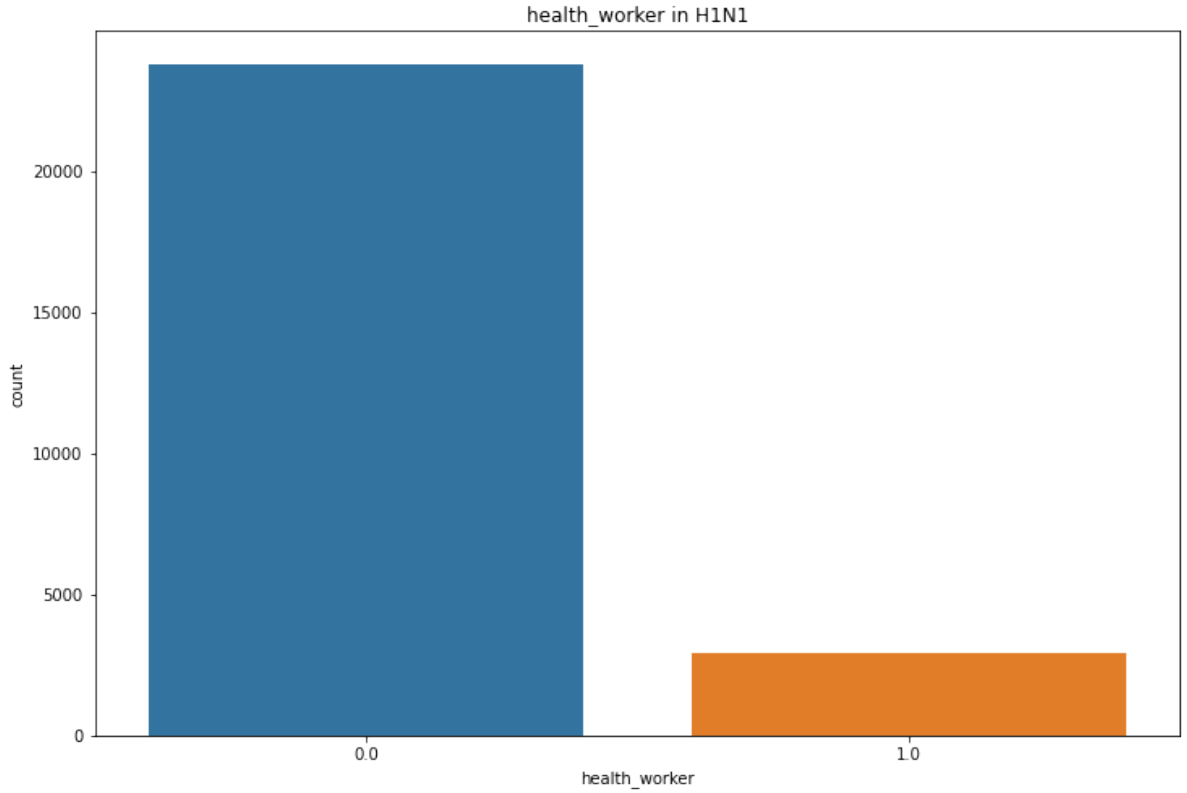
In [2012]:
```
1 value_counts('child_under_6_months')
```

Out[2012]:
```
0.0    0.919946
1.0    0.080054
Name: child_under_6_months, dtype: float64
```

most peopple (91%)do not have regular contact with children under 6 months only 9% have contact

13. health_worker

In [2013]:
```
1 countplot('health_worker','H1N1')
```



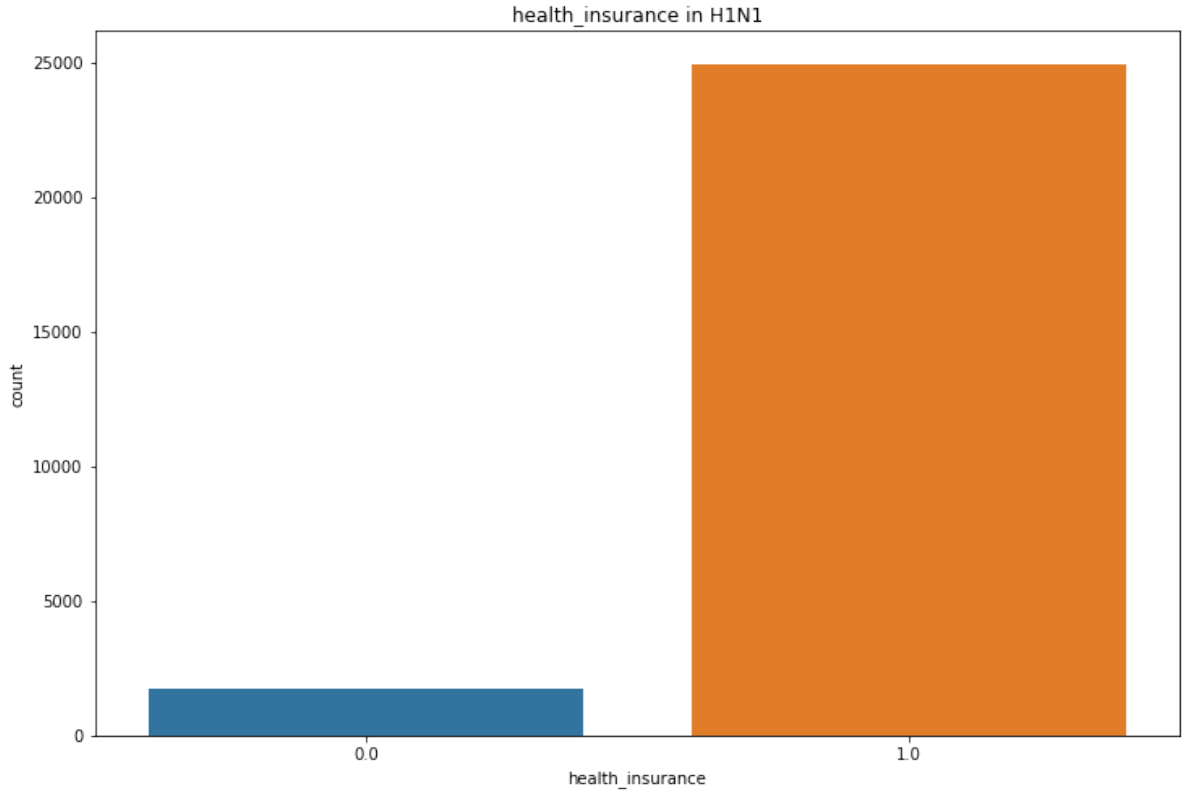
In [2014]:
```
1 value_counts('health_worker')
```

Out[2014]:
```
0.0    0.891452
1.0    0.108548
Name: health_worker, dtype: float64
```

most people are not health care workers

14. health_insurance

In [2015]:
```
1 countplot('health_insurance','H1N1')
```



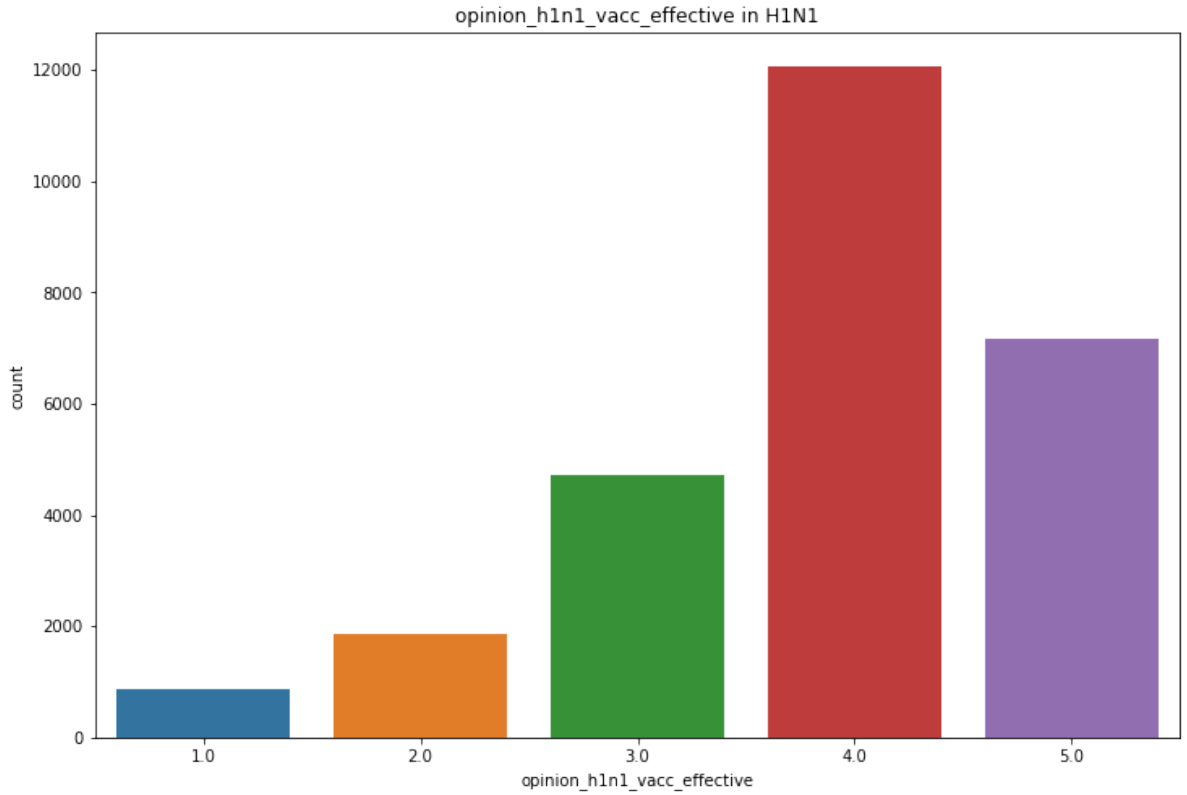
In [2016]:
```
1 value_counts('health_insurance')
```

Out[2016]:
```
1.0    0.934998
0.0    0.065002
Name: health_insurance, dtype: float64
```

most people(93%) have health insurance

15. opinion_h1n1_vacc_effective

In [2017]:
```
1 countplot('opinion_h1n1_vacc_effective','H1N1')
```



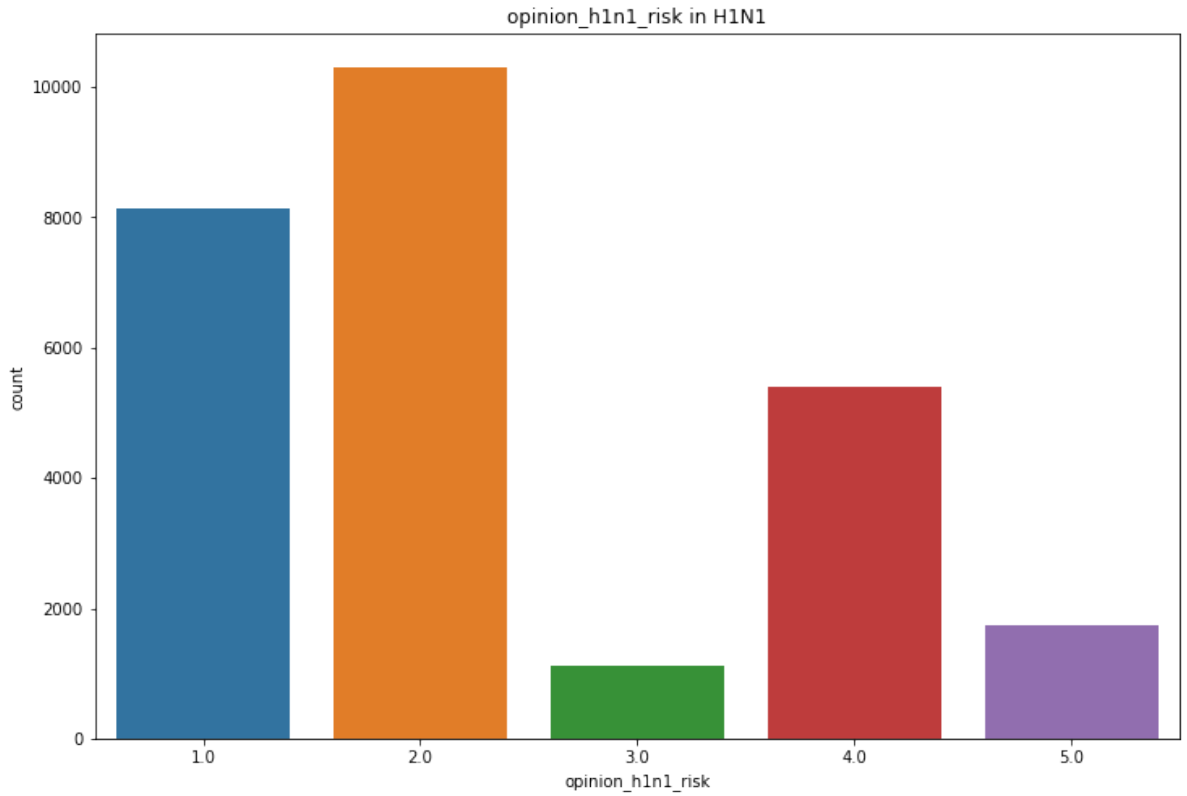
In [2018]:
```
1 value_counts('opinion_h1n1_vacc_effective')
```

Out[2018]:
```
4.0    0.452091
5.0    0.268319
3.0    0.176845
2.0    0.069570
1.0    0.033175
Name: opinion_h1n1_vacc_effective, dtype: float64
```

most people believe that h1n1 vaccine is somewhat effective

16. opinion_h1n1_risk

In [2019]:
```
1 countplot('opinion_h1n1_risk','H1N1')
```



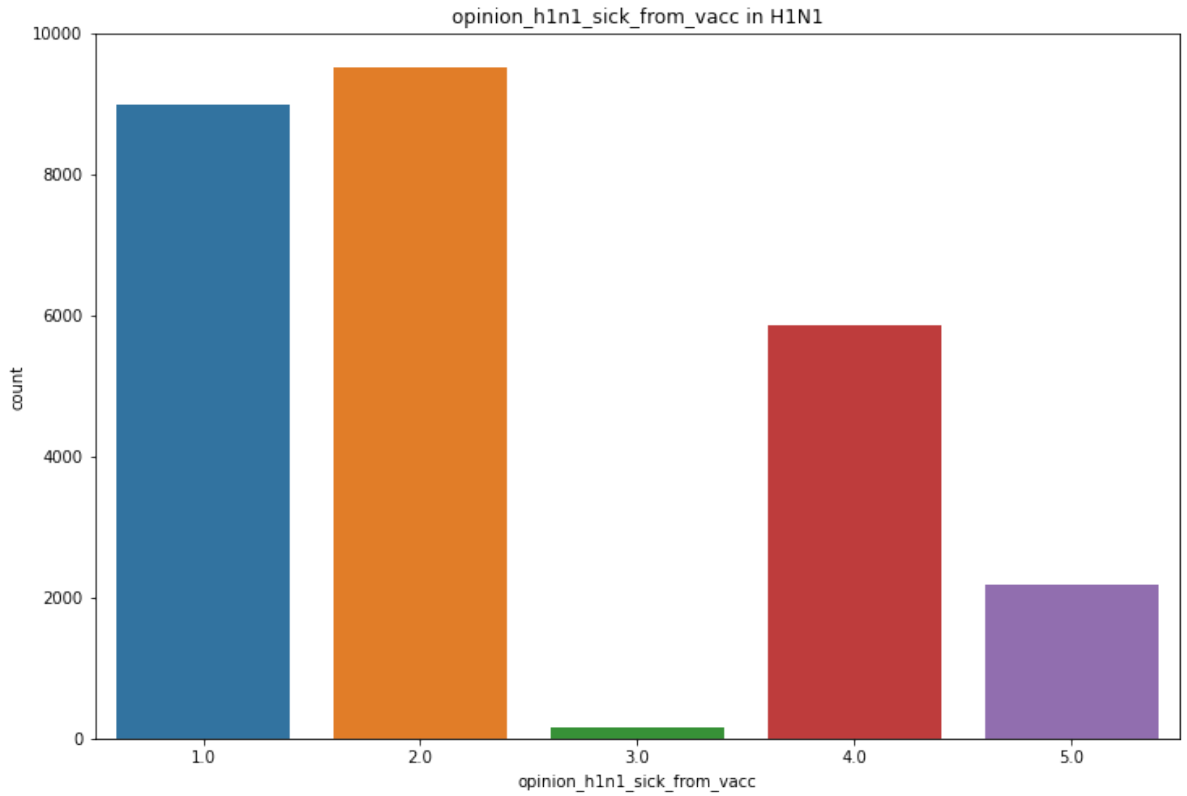
In [2020]:
```
1 value_counts('opinion_h1n1_risk')
```

Out[2020]:
```
2.0    0.385929
1.0    0.304752
4.0    0.201970
5.0    0.065526
3.0    0.041824
Name: opinion_h1n1_risk, dtype: float64
```

most people believe that the risk of getting sick with H1N1 flu without vaccine is low

17. opinion_h1n1_sick_from_vacc

In [2021]:
```
1 countplot('opinion_h1n1_sick_from_vacc','H1N1')
```



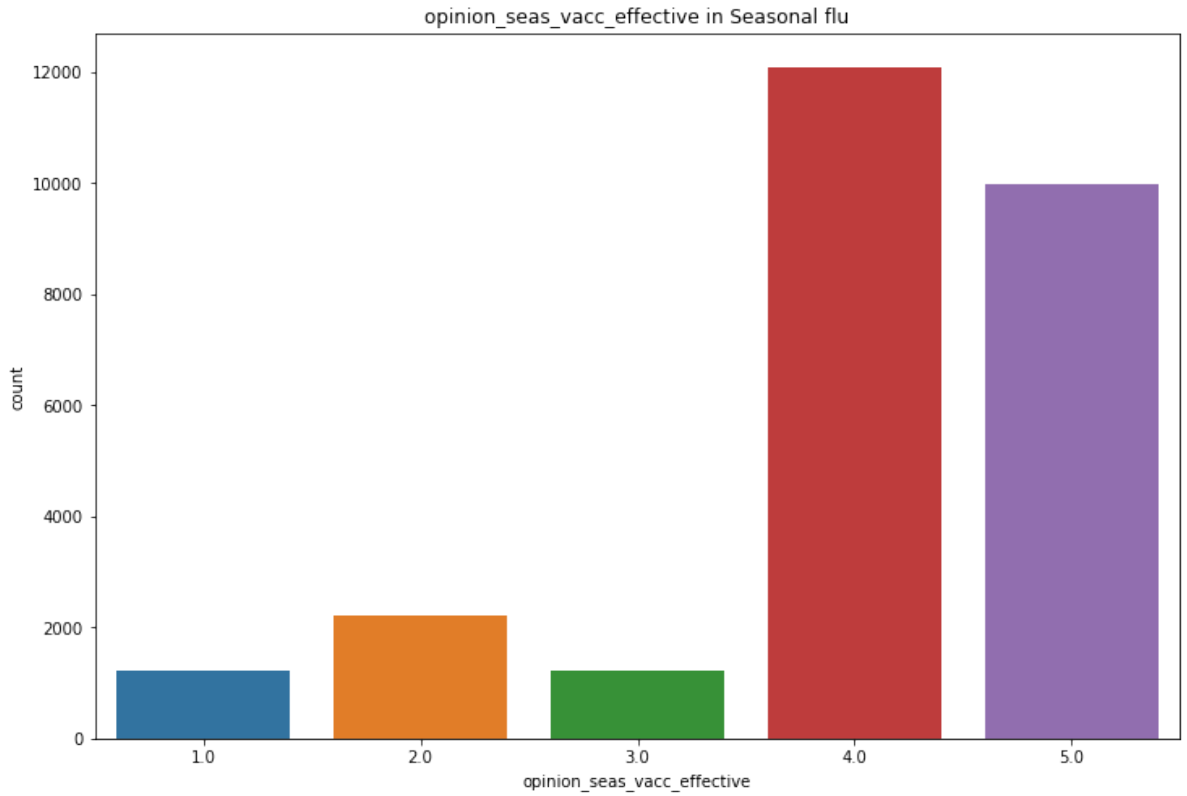
In [2022]:
```
1 value_counts('opinion_h1n1_sick_from_vacc')
```

Out[2022]:
```
2.0    0.356611
1.0    0.336915
4.0    0.219044
5.0    0.081889
3.0    0.005542
Name: opinion_h1n1_sick_from_vacc, dtype: float64
```

most people are not very worried of getting sick from taking H1N1 vaccine

18. opinion_seas_vacc_effective

In [2023]:
```
1 countplot('opinion_seas_vacc_effective','seasonal')
```



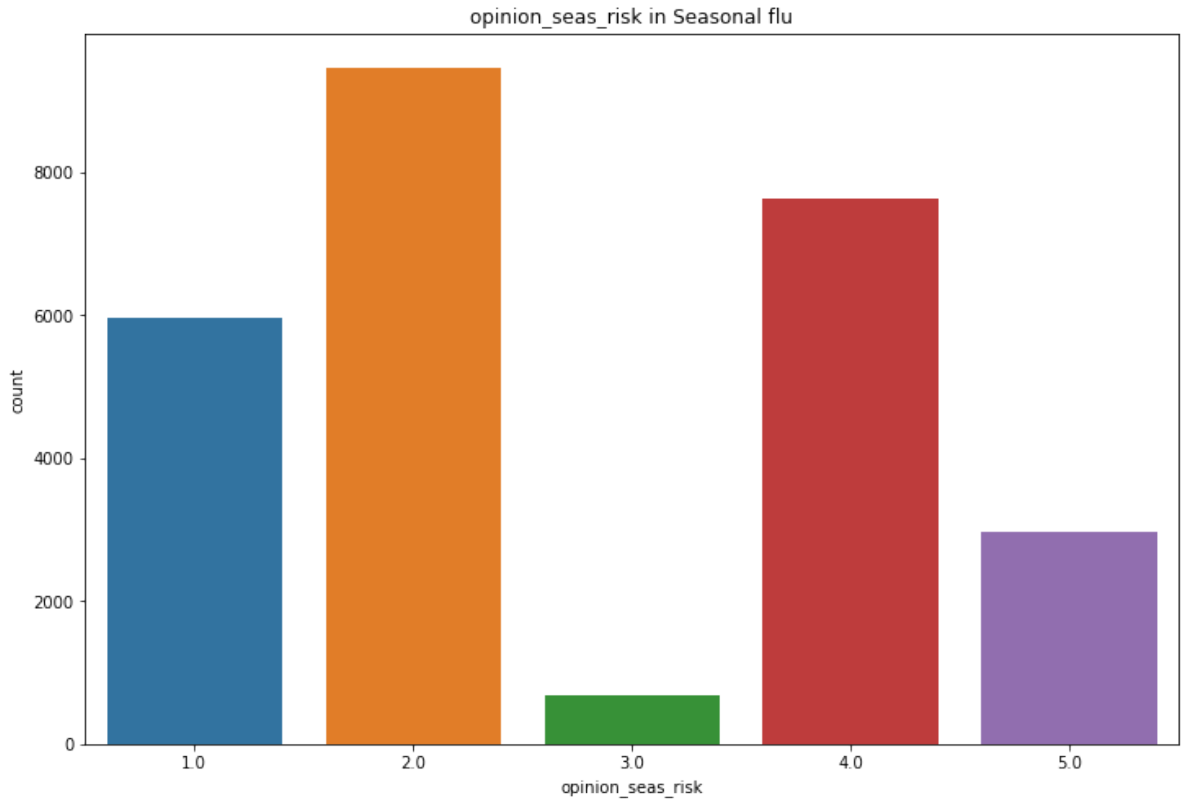
In [2024]:
```
1 value_counts('opinion_seas_vacc_effective')
```

Out[2024]:
```
4.0    0.452728
5.0    0.373423
2.0    0.082600
1.0    0.045718
3.0    0.045531
Name: opinion_seas_vacc_effective, dtype: float64
```

most people believe that Seasonal flu vaccine is somewhat effective

19. opinion_seas_risk

In [2025]:
```
1 countplot('opinion_seas_risk','seasonal')
```



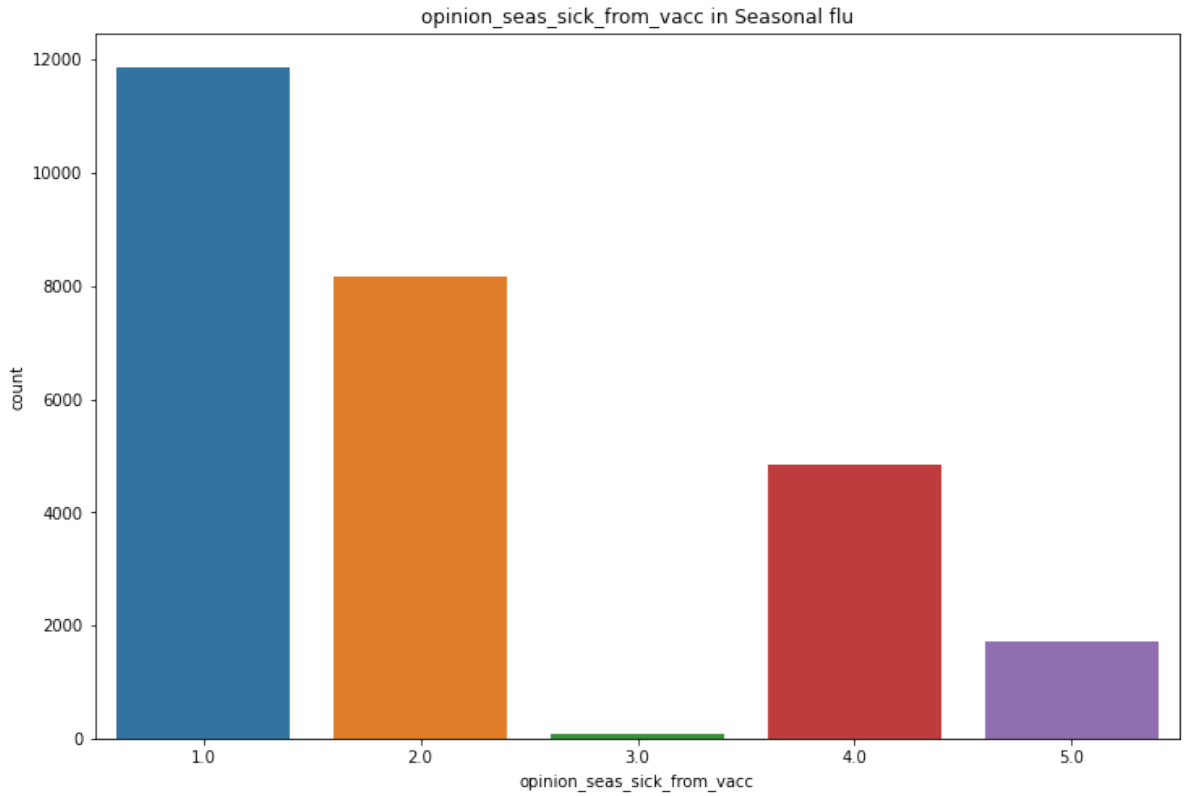
In [2026]:
```
1 value_counts('opinion_seas_risk')
```

Out[2026]:
```
2.0    0.354514
4.0    0.285693
1.0    0.223687
5.0    0.110757
3.0    0.025349
Name: opinion_seas_risk, dtype: float64
```

most people believe that the risk of getting sick with seasonal flu without vaccine is somewhat low

20. opinion_seas_sick_from_vacc

In [2027]:
```
1 countplot('opinion_seas_sick_from_vacc','seasonal')
```



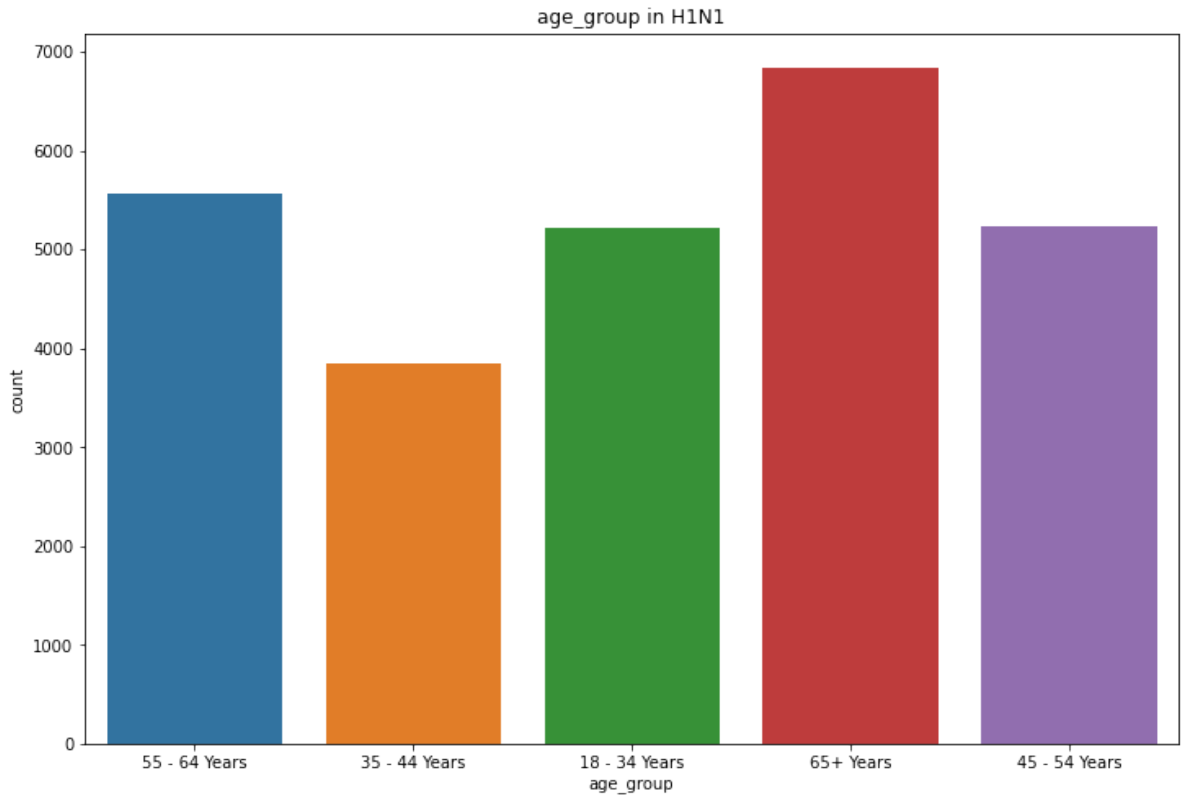
In [2028]:
```
1 value_counts('opinion_seas_sick_from_vacc')
```

Out[2028]:
```
1.0    0.444453
2.0    0.305912
4.0    0.181675
5.0    0.064440
3.0    0.003520
Name: opinion_seas_sick_from_vacc, dtype: float64
```

most people are not worried at all from getting sick on taking the seasonal flu vaccine

21. age_group

In [2029]:
```
1 countplot('age_group','H1N1')
```



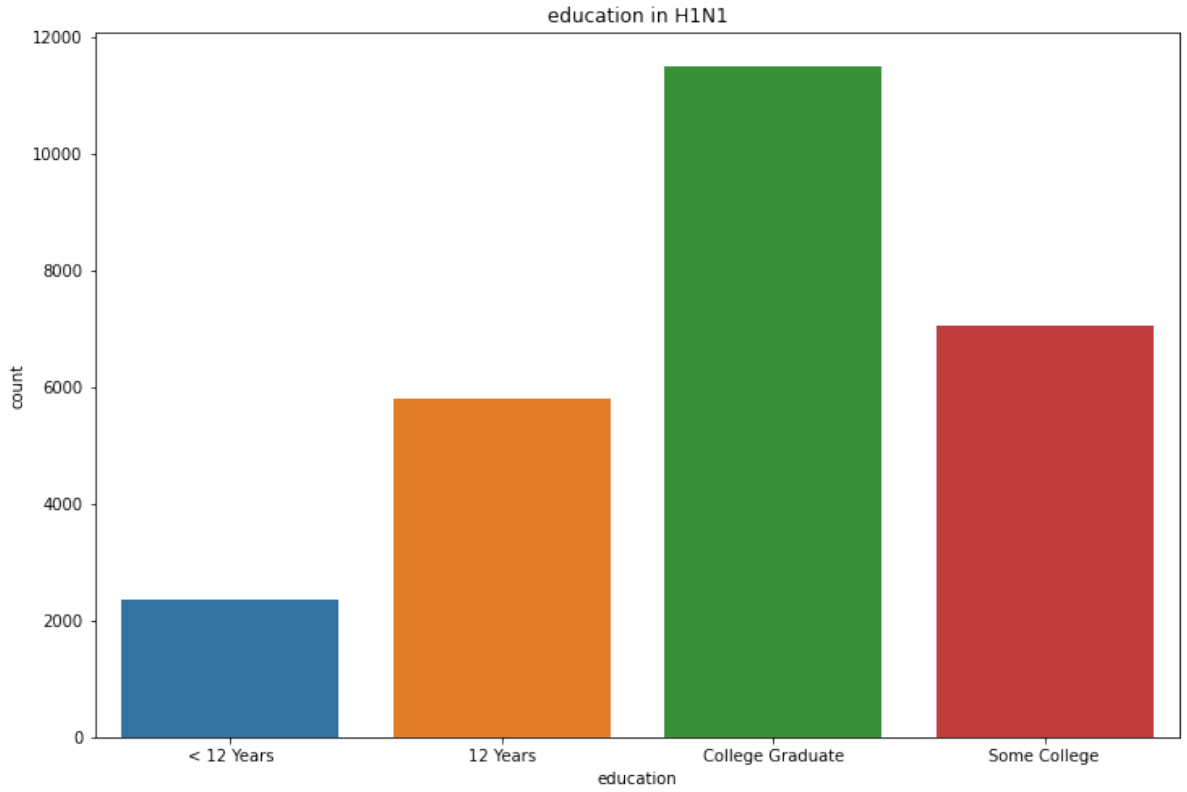
In [2030]:
```
1 value_counts('age_group')
```

Out[2030]:
```
65+ Years        0.256225
55 - 64 Years    0.208297
45 - 54 Years    0.196128
18 - 34 Years    0.195267
35 - 44 Years    0.144082
Name: age_group, dtype: float64
```

most Respodents who answered are 55 years and above

22.education

In [2031]:
```
1 countplot('education','H1N1')
```



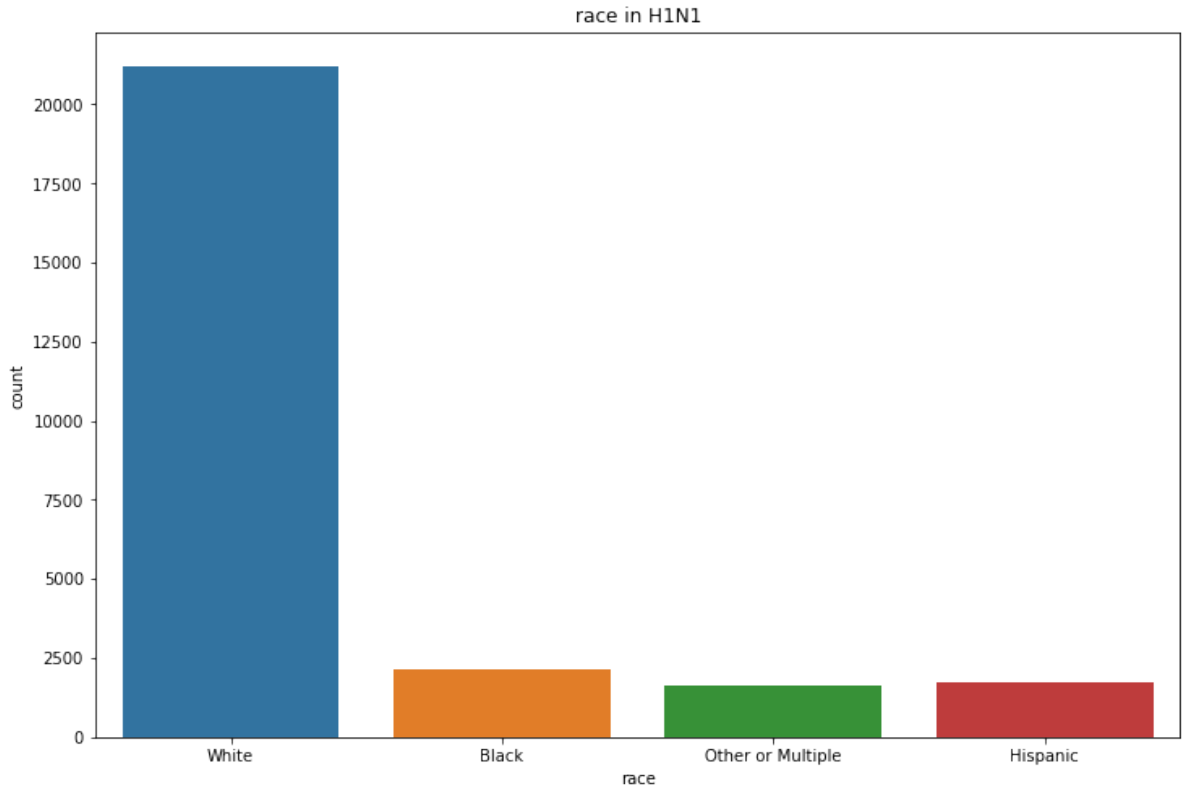
In [2032]:
```
1 value_counts('education')
```

Out[2032]:
```
College Graduate    0.430748
Some College        0.263714
12 Years            0.217059
< 12 Years          0.088479
Name: education, dtype: float64
```

Most respodents reported to be college graduates

23. race

In [2033]:
```
1 countplot('race','H1N1')
```



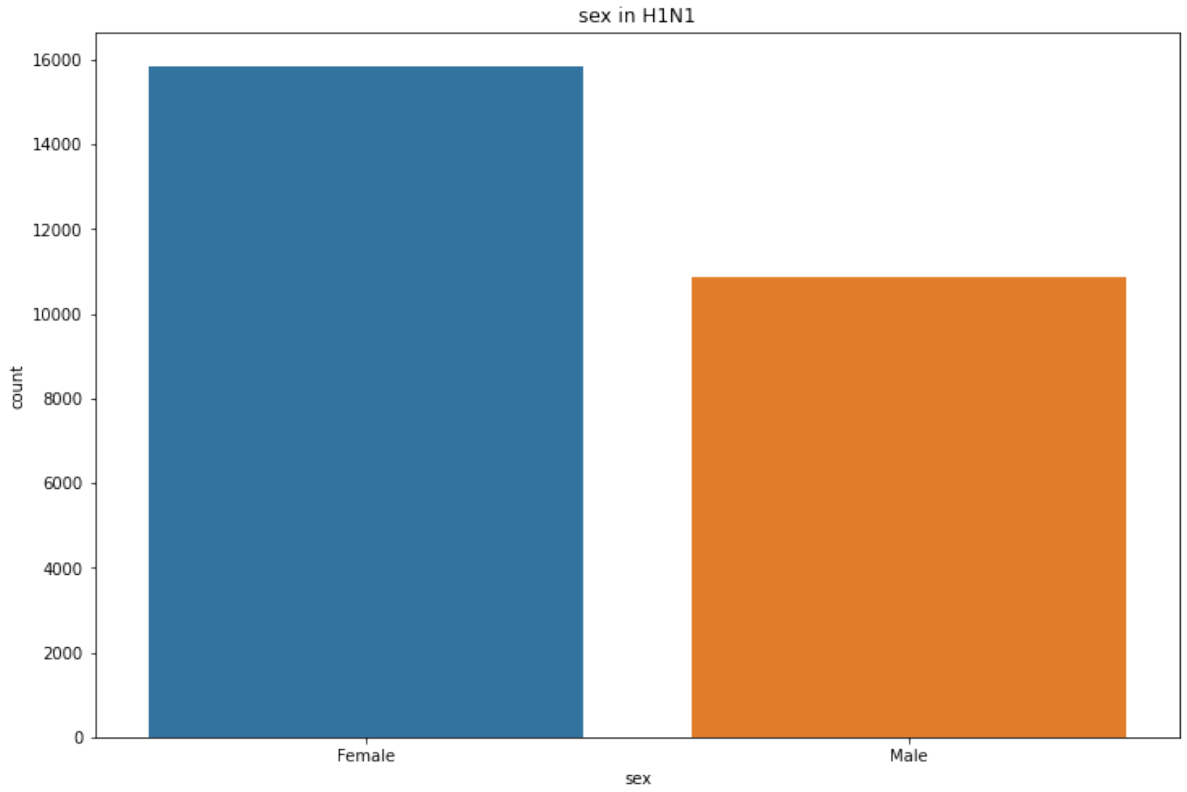
In [2034]:
```
1 value_counts('race')
```

Out[2034]:
```
White                0.794623
Black                0.079305
Hispanic             0.065713
Other or Multiple    0.060359
Name: race, dtype: float64
```

Most respodents are white

24. sex

In [2035]:
```
1 countplot('sex','H1N1')
```



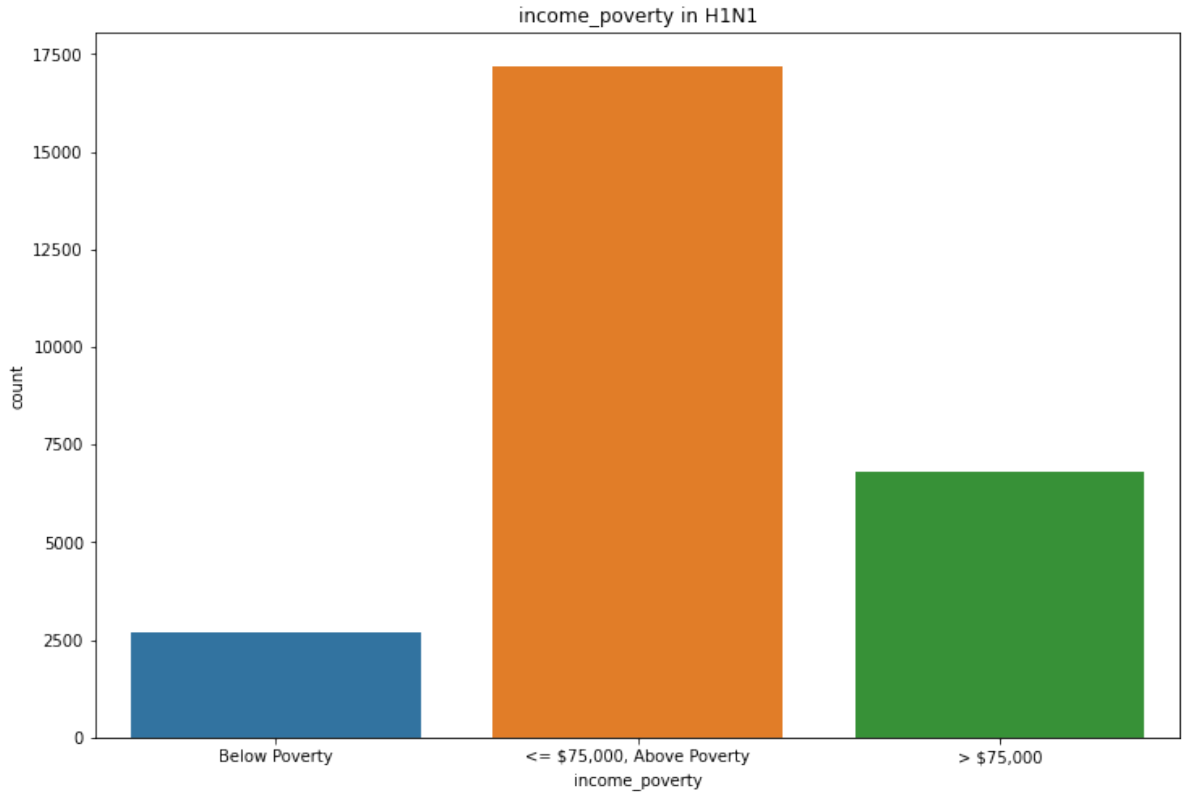
In [2036]:
```
1 value_counts('sex')
```

Out[2036]:
```
Female    0.593777
Male      0.406223
Name: sex, dtype: float64
```

Most respodents are females

25. income_poverty

In [2037]:
```
1 countplot('income_poverty','H1N1')
```



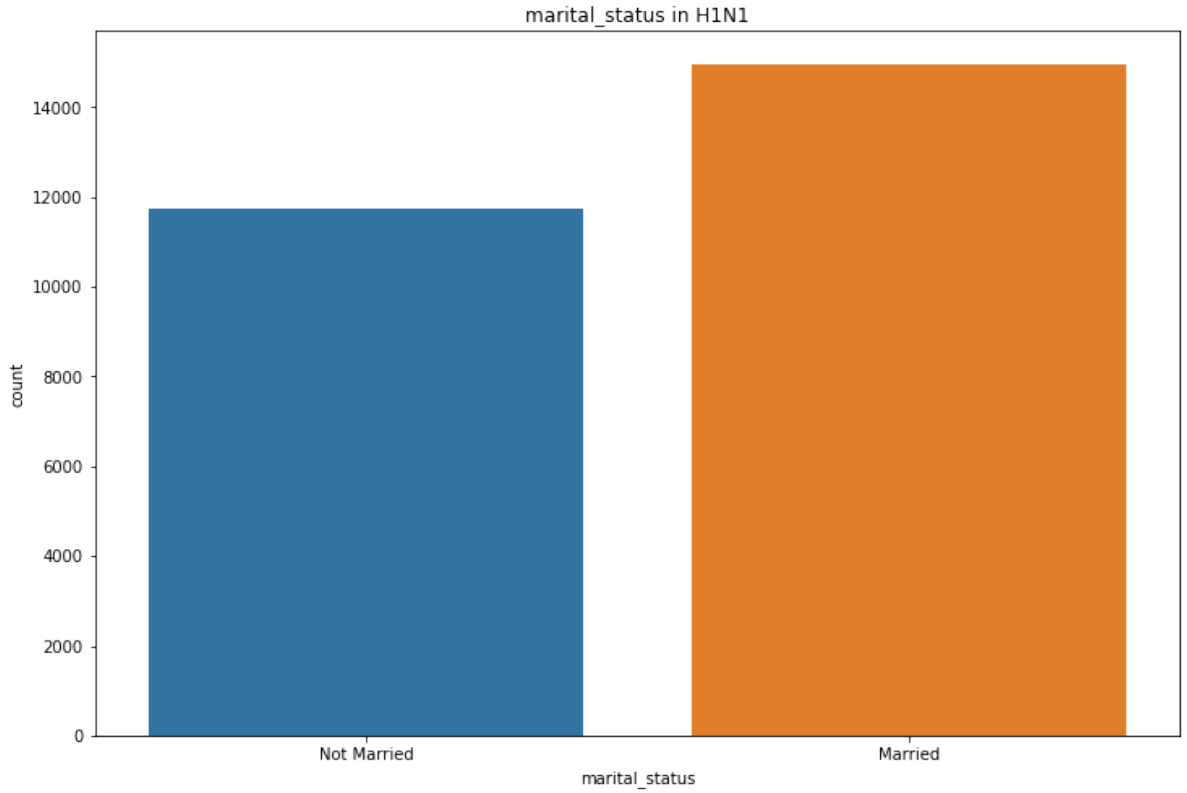
In [2038]:
```
1 value_counts('income_poverty')
```

Out[2038]:
```
<= $75,000, Above Poverty    0.644026
> $75,000                   0.254989
Below Poverty                0.100985
Name: income_poverty, dtype: float64
```

Most respodents are above poverty that is with $75000 and below

26. marital_status

In [2039]:
```
1 countplot('marital_status','H1N1')
```



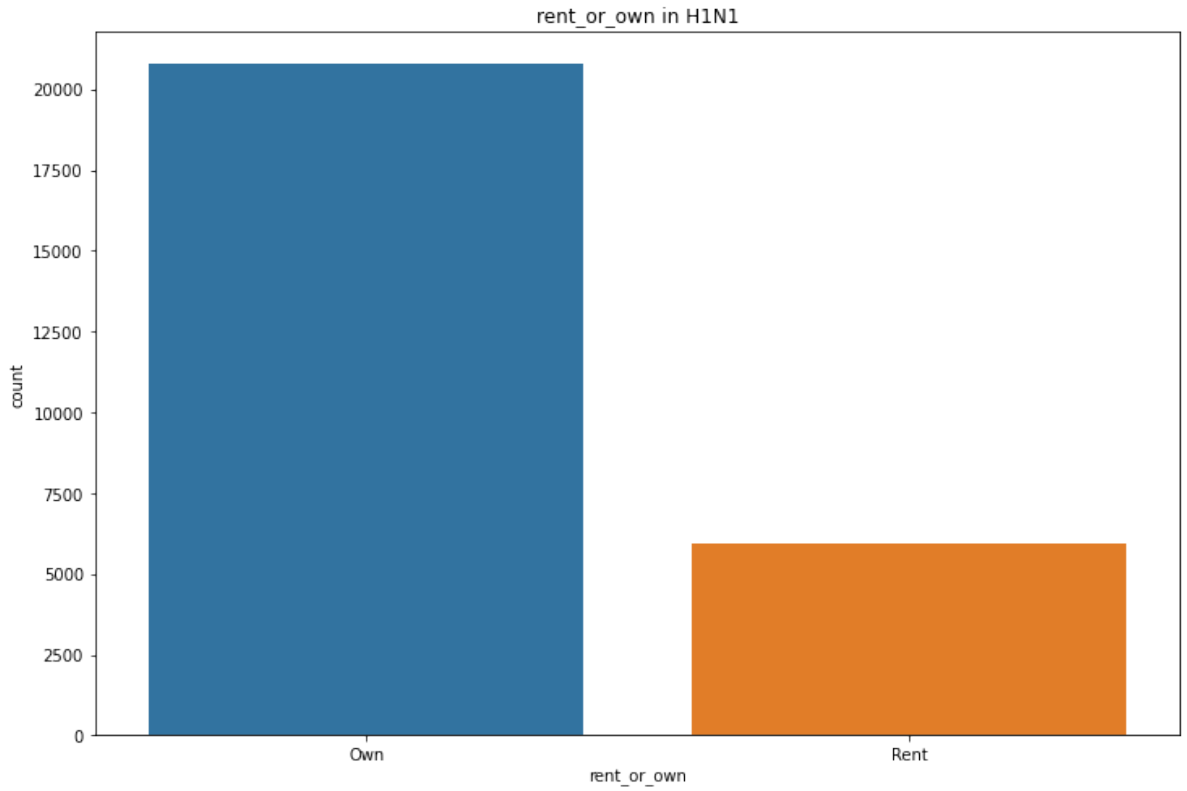
In [2040]:
```
1 value_counts('marital_status')
```

Out[2040]:
```
Married        0.560265
Not Married    0.439735
Name: marital_status, dtype: float64
```

most respodents are married

27. rent_or_own

In [2041]:
```
1 countplot('rent_or_own','H1N1')
```



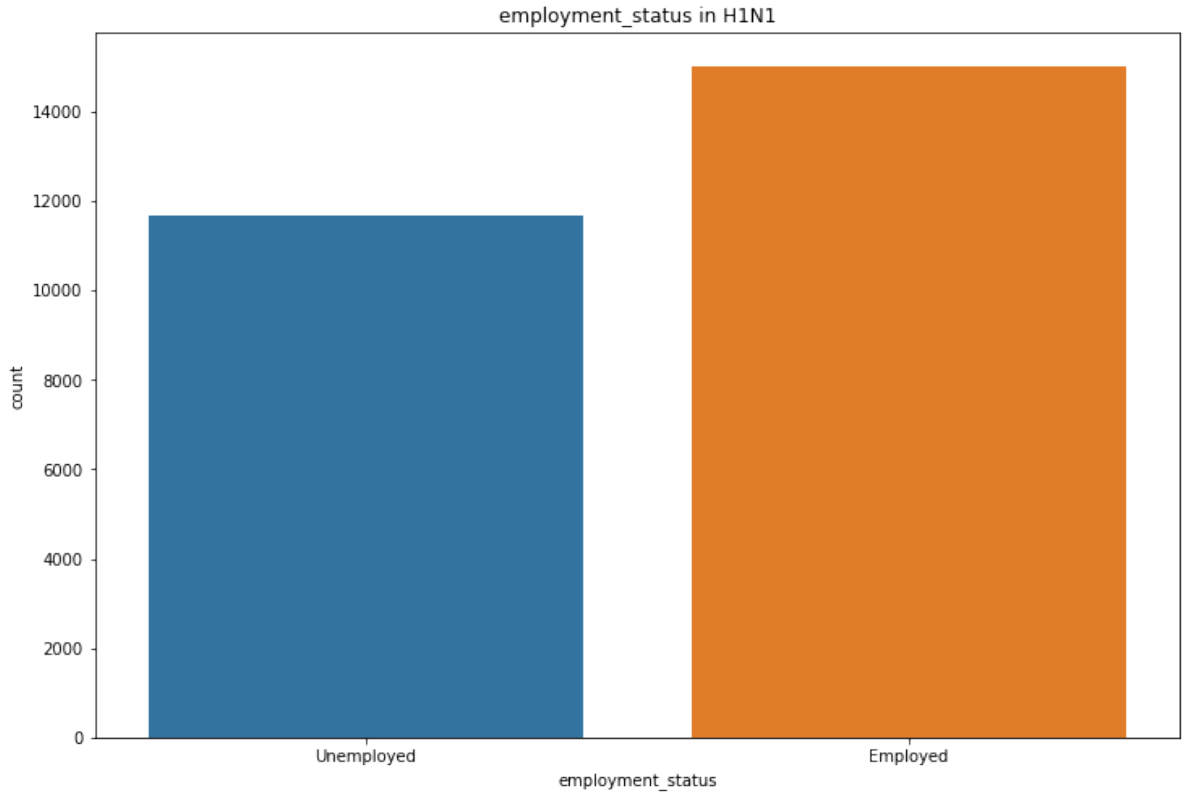
In [2042]:
```
1 value_counts('rent_or_own')
```

Out[2042]:
```
Own     0.777998
Rent    0.222002
Name: rent_or_own, dtype: float64
```

most respodents own houses

28.employment_status

In [2043]:
```
1 countplot('employment_status','H1N1')
```



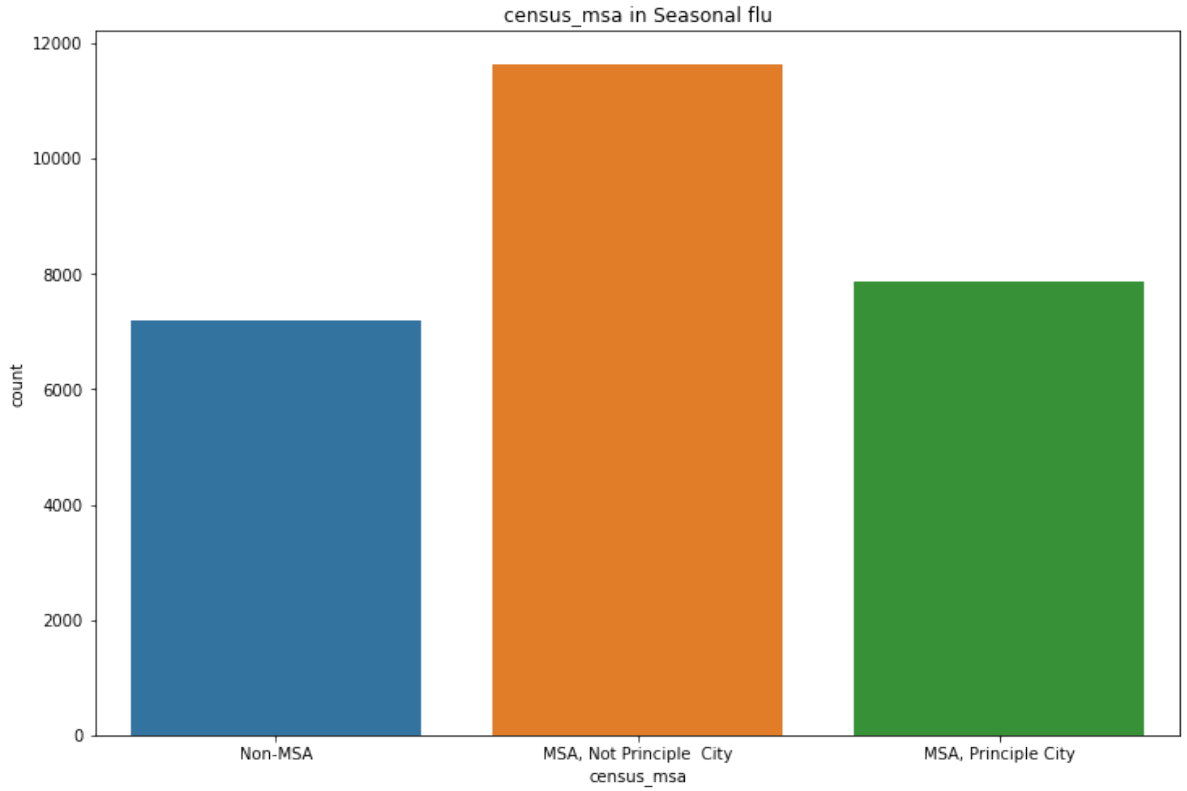
In [2044]:
```
1 value_counts('employment_status')
```

Out[2044]:
```
Employed      0.562512
Unemployed    0.437488
Name: employment_status, dtype: float64
```

Most respodents are employed

29 census_msa

In [2045]:
```
1 countplot('census_msa','H1NI')
```



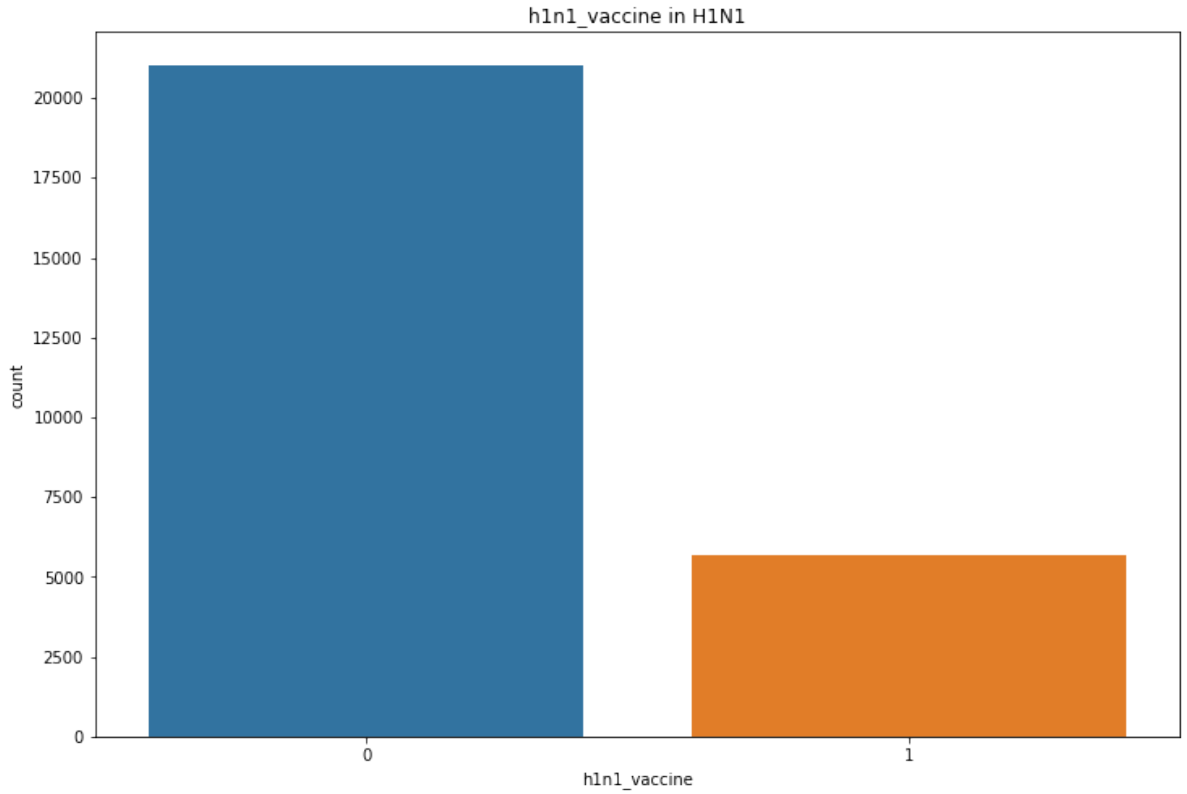
In [2046]:
```
1 value_counts('census_msa')
```

Out[2046]:
```
MSA, Not Principle  City    0.436028
MSA, Principle City         0.294455
Non-MSA                     0.269517
Name: census_msa, dtype: float64
```

most people not from MSA(metropolitan statistical area)

30.h1n1_vaccine

In [2047]:
```
1 countplot('h1n1_vaccine','H1N1')
```



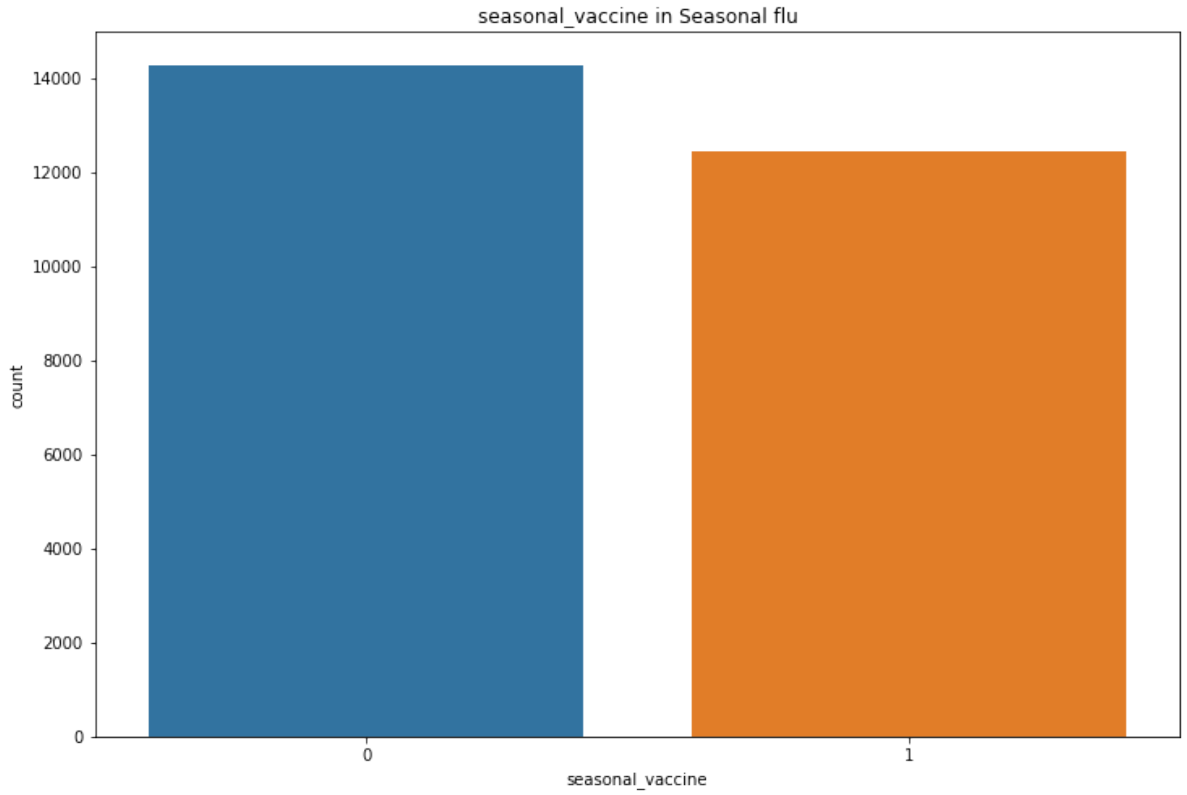
In [2048]:
```
1 value_counts('h1n1_vaccine')
```

Out[2048]:
```
0    0.787546
1    0.212454
Name: h1n1_vaccine, dtype: float64
```

Most respodents(78%) did not receive the H1N1 vaccine

31.seasonal_vaccine

In [2049]:
```
countplot('seasonal_vaccine','seasonal')
```



In [2050]:
```
value_counts('seasonal_vaccine')
```

Out[2050]:
```
0    0.534392
1    0.465608
Name: seasonal_vaccine, dtype: float64
```

most people(53%) did not receive the seasonal flu vaccine

4.One hot Encoding for categorical variables

In [2051]:
```
#categorical data
categoricals = training.select_dtypes(include='object')
#numeric data
numerics = training.drop(categoricals.columns,axis=1)

#one hot encoding
ohe = OneHotEncoder(drop='first',sparse_output=False)
train_encoded = pd.DataFrame(ohe.fit_transform(categoricals),columns=ohe

#combine encoded values with numerics columns
train_encoded = pd.concat([numerics,train_encoded],axis=1)
```

5. Feature Selection

In [2052]:
```
#predictor variables
X =  train_encoded.drop(['h1n1_vaccine','seasonal_vaccine','respondent_i
#target variables
y_h1n1 = train_encoded['h1n1_vaccine']
y_seasonal = train_encoded['seasonal_vaccine']
selector = SelectKBest(f_classif, k=20)
X_new_h1n1 = selector.fit_transform(X,y_h1n1)
X_new_seasonal = selector.fit_transform(X,y_seasonal)

```

6.Modelling

Logistic Regression

In [2053]:
```
#use logistic regression since this is a binary prediction
logreg = LogisticRegression(C=100, class_weight='balanced',random_state=

#split data for modelling h1n1
X_train_h1n1,X_test_h1n1,y_train_h1n1,y_test_h1n1 =  train_test_split(X_

#split data for modelling seasonal
X_train_seasonal,X_test_seasonal,y_train_seasonal,y_test_seasonal =  tra

#train for h1n1 vaccine
model_log_h1n1 = logreg.fit(X_train_h1n1,y_train_h1n1)
model_log_h1n1

# train for seasonal vaccine
model_log_seasonal = logreg.fit(X_train_seasonal,y_train_seasonal)
model_log_seasonal
```

Out[2053]:
```
LogisticRegression(C=100, class_weight='balanced', random_state=42,
                   solver='liblinear')
```

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.

a)Clean and encode test data

In [2054]:

```
null_percentage = (test_data.isna().sum()/test_data.shape[0]) * 100
#identify columns with 50 + null values
columns_to_drop = null_percentage[null_percentage>49.9].index
print(f'columns_to_drop: {columns_to_drop}')
#Drop those columns from the dataframe
test_data.drop(columns=columns_to_drop,inplace=True)
#check missing values again
((test_data.isna().sum()/test_data.shape[0]) * 100).sort_values(ascendin

#fill numerics with median and categorical with mode
for column in test_data.columns:
    if test_data[column].dtype=='object':
        test_data[column].fillna(test_data[column].mode()[0],inplace=Tru
    elif test_data[column].dtype =='float64':
        test_data[column].fillna(test_data[column].median(),inplace=True

test_data.drop(columns=['hhs_geo_region','household_children','household
test_data.replace('Not in Labor Force','Unemployed',inplace=True)

#categorical data
categoricals_test = test_data.select_dtypes(include='object')
#numeric data
numerics_test = test_data.drop(categoricals.columns,axis=1)

#one hot encoding
ohe = OneHotEncoder(drop='first',sparse_output=False)
test_encoded = pd.DataFrame(ohe.fit_transform(categoricals_test),columns

#combine encoded values with numerics columns
test_encoded = pd.concat([numerics_test,test_encoded],axis=1)

X_test =  test_encoded.drop('respondent_id',axis=1)

X_new_test = selector.transform(X_test)
```

```
columns_to_drop: Index(['employment_occupation'], dtype='object')
```

b) Predict test data

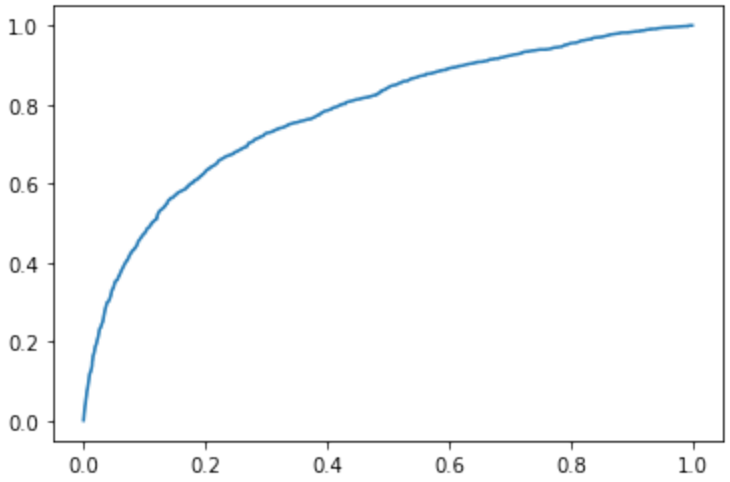
In [2055]:

```
y_predicted_h1n1 = model_log_h1n1.predict(X_test_h1n1)
y_predicted_seasonal = model_log_seasonal.predict(X_test_seasonal)
```

c)Evaluate Model performance

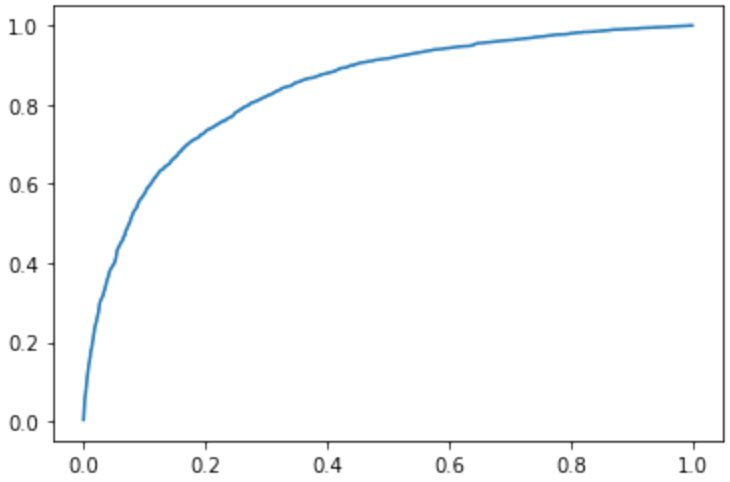
In [2057]:
```
#h1n1
y_score_h1n1 = model_log_h1n1.decision_function(X_test_h1n1)
fpr,tpr,threshhold = roc_curve(y_test_h1n1,y_score_h1n1)
# sns.lineplot(x=fpr,y=tpr)
h1n1_auc = auc(fpr,tpr)
h1n1_auc
```

Out[2057]: 0.779794702844236



In [2058]:
```
#seasonal
y_score_seasonal = model_log_seasonal.decision_function(X_test_seasonal)
fpr,tpr,threshhold = roc_curve(y_test_seasonal,y_score_seasonal)
# sns.lineplot(x=fpr,y=tpr)
seasonal_auc = auc(fpr,tpr)
seasonal_auc
```

Out[2058]: 0.8414155805063176



In [2059]:
```
#average of seasonal auc and    H1n1
model_performance =  (h1n1_auc+seasonal_auc)/2
model_performance
```

Out[2059]: 0.8106051416752769

mse, r2

In [2060]:
```
mse_h1n1_train  = mean_squared_error(y_train_h1n1,model_log_h1n1.predict
mse_h1n1_test = mean_squared_error(y_test_h1n1,y_predicted_h1n1)

mse_seasonal_train  = mean_squared_error(y_train_seasonal,model_log_seas
mse_seasonal_test = mean_squared_error(y_test_seasonal,y_predicted_seaso

# r2_Score_h1n1 = r2_score(y_train_h1n1,model_log_h1n1.predict(X_train_h
# r2_Score_seasonal = r2_score(y_train_seasonal,model_log_seasonal.predi

r_Squared_h1n1 = model_log_h1n1.score(X_train_h1n1,y_train_h1n1)
r_Squared_seasonal = model_log_seasonal.score(X_train_seasonal,y_train_s

print(f'r_Squared_h1n1: {r_Squared_h1n1}')
print(f'r_Squared_seasonal: {r_Squared_seasonal}')

print(f'mse_h1n1_train: {mse_h1n1_train}')
print(f'mse_h1n1_test: {mse_h1n1_test}')

print(f'mse_seasonal_train: {mse_seasonal_train}')
print(f'mse_seasonal_test: {mse_seasonal_test}')


```

```
r_Squared_h1n1: 0.2973681395100032
r_Squared_seasonal: 0.7651118005777254
mse_h1n1_train: 0.7026318604899968
mse_h1n1_test: 0.6986147510295769
mse_seasonal_train: 0.2348881994222745
mse_seasonal_test: 0.2332459752901535
```

d)Predict our test data from file

In [2061]:
```
test_predict_h1n1 = model_log_h1n1.predict(X_new_test)
test_predict_seasonal =  model_log_seasonal.predict(X_new_test)
### create daframe for the predictions
predictions = pd.DataFrame({
    'h1n1_vaccine': test_predict_h1n1,
    'seasonal_vaccine': test_predict_seasonal
},index=range(26707, 26707 + len(test_predict_h1n1)))

#make index a column
predictions = predictions.reset_index()

#rename index column to respodent id
predictions.rename(columns={
    'index':'respondent_id'
},inplace=True)
```

In [2063]:
```
#write predictions dataframe to csv file
predictions.to_csv('h1n1_seasonal_flu_Vaccinepredictions.csv',index=Fals
```

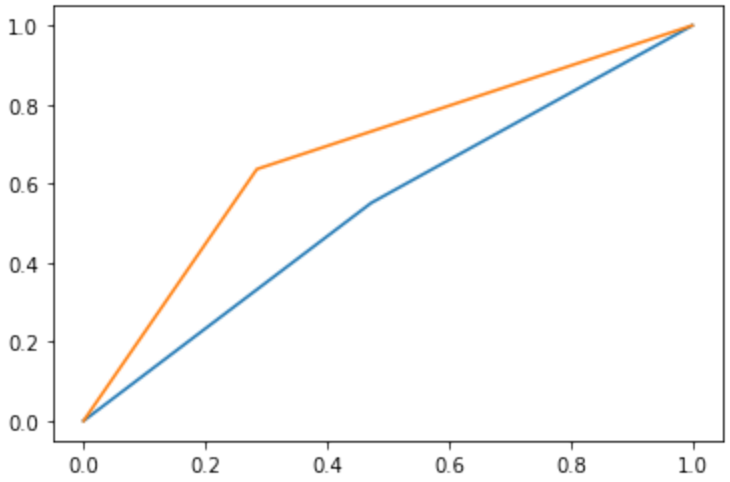
Mpdel 2: Decision Tree

In [ ]:

```
#use logistic regression since this is a binary prediction
clf = DecisionTreeClassifier(random_state=10)

#split data for modelling h1n1
X_train_h1n1,X_test_h1n1,y_train_h1n1,y_test_h1n1 =  train_test_split(X_

#split data for modelling seasonal
X_train_seasonal,X_test_seasonal,y_train_seasonal,y_test_seasonal =  tra

#train for h1n1 vaccine
model_log_h1n1 = clf.fit(X_train_h1n1,y_train_h1n1)
model_log_h1n1

# train for seasonal vaccine
model_log_seasonal = clf.fit(X_train_seasonal,y_train_seasonal)
model_log_seasonal


y_predicted_h1n1 = model_log_h1n1.predict(X_test_h1n1)
y_predicted_seasonal = model_log_seasonal.predict(X_test_seasonal)

#h1n1
# y_score_h1n1 = model_log_h1n1.decision_function(X_test_h1n1)
fpr,tpr,threshhold = roc_curve(y_test_h1n1,y_predicted_h1n1)
sns.lineplot(x=fpr,y=tpr)
h1n1_auc = auc(fpr,tpr)
h1n1_auc

#seasonal
# y_score_seasonal = model_log_seasonal.decision_function(X_test_seasona
fpr,tpr,threshhold = roc_curve(y_test_seasonal,y_predicted_seasonal)
sns.lineplot(x=fpr,y=tpr)
seasonal_auc = auc(fpr,tpr)
seasonal_auc

model_performance =  (h1n1_auc+seasonal_auc)/2
model_performance

mse_h1n1_train  = mean_squared_error(y_train_h1n1,model_log_h1n1.predict
mse_h1n1_test = mean_squared_error(y_test_h1n1,y_predicted_h1n1)

mse_seasonal_train  = mean_squared_error(y_train_seasonal,model_log_seas
mse_seasonal_test = mean_squared_error(y_test_seasonal,y_predicted_seaso

# r2_Score_h1n1 = r2_score(y_train_h1n1,model_log_h1n1.predict(X_train_h
# r2_Score_seasonal = r2_score(y_train_seasonal,model_log_seasonal.predi

r_Squared_h1n1 = model_log_h1n1.score(X_train_h1n1,y_train_h1n1)
r_Squared_seasonal = model_log_seasonal.score(X_train_seasonal,y_train_s

print(f'r_Squared_h1n1: {r_Squared_h1n1}')
print(f'r_Squared_seasonal: {r_Squared_seasonal}')

print(f'mse_h1n1_train: {mse_h1n1_train}')
print(f'mse_h1n1_test: {mse_h1n1_test}')

print(f'mse_seasonal_train: {mse_seasonal_train}')
```

```
58 print(f'mse_seasonal_test: {mse_seasonal_test}')
59
60
```

```
r_Squared_h1n1: 0.5376591419706858
r_Squared_seasonal: 0.9687065368567455
mse_h1n1_train: 0.46234085802931424
mse_h1n1_test: 0.46761512542119055
mse_seasonal_train: 0.03129346314325452
mse_seasonal_test: 0.3207288156745289
```



###

Hyper parameter tuning

In [ ]:

```
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_auc_score, make_scorer

# Define parameter grid
param_grid = [
    # Logistic Regression parameters
    {
        'model': [LogisticRegression(solver='liblinear', random_state=42
        'model__C': [0.01, 0.1, 1, 10, 100],
        'model__class_weight': ['balanced', None],
    },
    # Decision Tree parameters
    {
        'model': [DecisionTreeClassifier(random_state=42)],
        'model__max_depth': [5, 10, 20, None],
        'model__min_samples_split': [2, 5, 10],
        'model__min_samples_leaf': [1, 2, 5],
    }
]

# Define pipeline
pipeline = Pipeline([
    ('model', LogisticRegression())  # Placeholder; actual model chosen
])

# Use AUC as the evaluation metric
scorer = make_scorer(roc_auc_score, needs_proba=True)

# Grid Search CV
grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid, sc

# Fit on H1N1 dataset
grid_search.fit(X_train_h1n1, y_train_h1n1)

# Best model and parameters
best_model_h1n1 = grid_search.best_estimator_
best_params_h1n1 = grid_search.best_params_

print(f"Best Model for H1N1: {best_model_h1n1}")
print(f"Best Parameters for H1N1: {best_params_h1n1}")

```

```
Fitting 5 folds for each of 46 candidates, totalling 230 fits
Best Model for H1N1: Pipeline(steps=[('model',
                 LogisticRegression(C=100, class_weight='balanced',
                                    random_state=42, solver='liblinear'))])
Best Parameters for H1N1: {'model': LogisticRegression(random_state=42, solve
r='liblinear'), 'model__C': 100, 'model__class_weight': 'balanced'}
```

In [ ]:

```

```