

PREDICTING THE LIKELIHOOD OF INDIVIDUALS TAKING H1N1 FLU AND SEASONAL FLU VACCINES USING LOGISTIC REGRESSION MODEL AND DECISION TREE CLASSIFIER (ID3) MODELS

STUDENT: PAMELA JEPKORIR CHEBII

BUSINESS UNDERSTANDING

Introduction

Vaccination is a key public health measure used to fight infectious diseases. It is a simple, safe, and effective way of protecting a person against harmful diseases, before they come into contact with them (WHO, 2024). Vaccines provide immunization for individuals, and enough immunization in a community can further reduce the spread of diseases through "herd immunity."

Project Overview

Mpox, just like covid 19 that affected the entire world in 2020, is a new viral infection which can spread between people, mainly through close contact, and occasionally from the environment to people via things and surfaces that have been touched by a person with mpox (WHO, 2024). It is an illness caused by the monkeypox virus. 4 cases of Mpox at the time of writing this have been reported in Kenya but the vaccines for the disease are still under development. This research will help the government in determining the factors that contribute to people accepting to participate in the vaccination exercise.

The data for this research on uptake of vaccines based on opinions, demographics and doctor's recommendation is from United States National 2009 H1N1 Flu Survey collected to monitor vaccination rates during the US Government Vaccination Campaign that began in October 2009. A phone survey was used to ask people whether they had received H1N1 and seasonal flu vaccines, in conjunction with information they shared about their lives, opinions, and behaviors.

Objectives

- To predict whether people got H1N1 and Seasonal Flue Vaccines using information shared about their backgrounds, opinions and Health Behaviour.
- To determine the best performing model in predicting Vaccines

DEVELOPMENT MODELS

- Logistic Regression Model
- Decision Tree Classifier (ID3)

Performance Metrics

ROC-AUC Curve

- AUC was used because the data is highly imbalanced.

- Accuracy score could have worked well if we had balanced classes.
- Most of the people who participated in the survey had not received the vaccines and thus leading to class imbalance.

DATA UNDERSTANDING

Here, the different sets of data as downloaded from the **DRIVENDATA** data repository are described:

Training Features:

- These are the input variables that the model will use to predict the probability that people received H1N1 flu and seasonal flu vaccines. There are 35 feature columns in total, each a response to a survey question. These questions cover several different topics, such as whether people observed safe behavioral practices, their opinions about the diseases and the vaccines, and their demographics.

Training Labels:

- These are the labels corresponding to the observations in the training features. There are two target variables: `h1n1_vaccine` and `seasonal_vaccine`. Both are binary variables, with 1 indicating that a person received the respective flu vaccine and 0 indicating that a person did not receive the respective flu vaccine. This is a "multilabel" modeling task.

Test Features:

- These are the features for observations that will be used to generate the submission predictions after training a model.

Submission Format:

- This file serves as an example for how to format submission.

DESCRIPTION OF FEATURES AND LABELS

FEATURES DATASET

The features dataset has 36 columns. The first column `respondent_id` is a unique and random identifier. The remaining 35 features are described below:

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

- **h1n1_concern** - Level of concern about the H1N1 flu. 0 = Not at all concerned; 1 = Not very concerned; 2 = Somewhat concerned; 3 = Very concerned.
- **h1n1_knowledge** - Level of knowledge about H1N1 flu. 0 = No knowledge; 1 = A little knowledge; 2 = A lot of knowledge.
- **behavioral_antiviral_meds** - Has taken antiviral medications. (binary)
- **behavioral_avoidance** - Has avoided close contact with others with flu-like symptoms. (binary)
- **behavioral_face_mask** - Has bought a face mask. (binary)
- **behavioral_wash_hands** - Has frequently washed hands or used hand sanitizer. (binary)
- **behavioral_large_gatherings** - Has reduced time at large gatherings. (binary)
- **behavioral_outside_home** - Has reduced contact with people outside of own household. (binary)
- **behavioral_touch_face** - Has avoided touching eyes, nose, or mouth. (binary)
- **doctor_recc_h1n1** - H1N1 flu vaccine was recommended by doctor. (binary)

- **doctor_recc_seasonal** - Seasonal flu vaccine was recommended by doctor. (binary)
- **chronic_med_condition** - Has any of the following chronic medical conditions: asthma or an other lung condition, diabetes, a heart condition, a kidney condition, sickle cell anemia or other anemia, a neurological or neuromuscular condition, a liver condition, or a weakened immune system caused by a chronic illness or by medicines taken for a chronic illness. (binary)
- **child_under_6_months** - Has regular close contact with a child under the age of six months. (binary)
- **health_worker**- Is a healthcare worker. (binary)
- **health_insurance**- Has health insurance. (binary)
- **opinion_h1n1_vacc_effective**- Respondent's opinion about H1N1 vaccine effectiveness. 1 = Not at all effective; 2 = Not very effective; 3 = Don't know; 4 = Somewhat effective; 5 = Very effective.
- **opinion_h1n1_risk** - Respondent's opinion about risk of getting sick with H1N1 flu without vaccine. 1 = Very Low; 2 = Somewhat low; 3 = Don't know; 4 = Somewhat high; 5 = Very high.
- **opinion_h1n1_sick_from_vacc** - Respondent's worry of getting sick from taking H1N1 vaccine. 1 = Not at all worried; 2 = Not very worried; 3 = Don't know; 4 = Somewhat worried; 5 = Very worried.
- **opinion_seas_vacc_effective** - Respondent's opinion about seasonal flu vaccine effectiveness. 1 = Not at all effective; 2 = Not very effective; 3 = Don't know; 4 = Somewhat effective; 5 = Very effective.
- **opinion_seas_risk** - Respondent's opinion about risk of getting sick with seasonal flu without vaccine. 1 = Very Low; 2 = Somewhat low; 3 = Don't know; 4 = Somewhat high; 5 = Very high.
- **opinion_seas_sick_from_vacc** - Respondent's worry of getting sick from taking seasonal flu vaccine. 1 = Not at all worried; 2 = Not very worried; 3 = Don't know; 4 = Somewhat worried; 5 = Very worried.
- **age_group** - Age group of respondent.
- **education** - Self-reported education level.
- **race**- Race of respondent.
- **sex**- Sex of respondent.
- **income_poverty** - Household annual income of respondent with respect to 2008 Census poverty thresholds.
- **marital_status** - Marital status of respondent.
- **rent_or_own** - Housing situation of respondent.
- **employment_status** - Employment status of respondent.
- **hhs_geo_region**- Respondent's residence using a 10-region geographic classification defined by the U.S. Dept. of Health and Human Services. Values are represented as short random character strings.
- **census_msa** - Respondent's residence within metropolitan statistical areas (MSA) as defined by the U.S. Census.
- **household_adults** - Number of other adults in household, top-coded to 3.
- **household_children** - Number of children in household, top-coded to 3.
- **employment_industry** - Type of industry respondent is employed in. Values are represented as short random character strings.
- **employment_occupation** - Type of occupation of respondent. Values are represented as short random character strings.

Labels Dataset

There are two target variables:

- **h1n1_vaccine** - Whether respondent received H1N1 flu vaccine.
- **seasonal_vaccine** - Whether respondent received seasonal flu vaccine.

Both are binary variables: 0 = No; 1 = Yes. Some respondents didn't get either vaccine, others got only one, and some got both. This is formulated as a multilabel (and not multiclass) problem.

Importing the libraries that will be used to load and explore the data

```
In [1]: import pandas as pd
import numpy as np
```

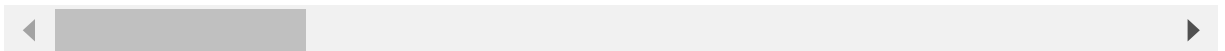
Load and explore the features dataset

```
In [2]: features_df = pd.read_csv("Data/training_set_features.csv", index_col = "respondent_id")
features_df.head()
```

Out[2]:

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

5 rows × 35 columns



Check the shape of the features dataset

```
In [3]: features_df.shape
```

Out[3]: (26707, 35)

- The dataset has 26,707 observations and 35 features.
- Each row represents a person who took part in the survey as a respondent
- The columns are the values corresponding to the participants in the survey

checking the info of the features dataset

```
In [4]: features_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26707 entries, 0 to 26706
Data columns (total 35 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   h1n1_concern                          26615 non-null  float64
1   h1n1_knowledge                        26591 non-null  float64
2   behavioral_antiviral_meds             26636 non-null  float64
3   behavioral_avoidance                  26499 non-null  float64
4   behavioral_face_mask                  26688 non-null  float64
5   behavioral_wash_hands                 26665 non-null  float64
6   behavioral_large_gatherings           26620 non-null  float64
7   behavioral_outside_home               26625 non-null  float64
8   behavioral_touch_face                 26579 non-null  float64
9   doctor_recc_h1n1                     24547 non-null  float64
10  doctor_recc_seasonal                  24547 non-null  float64
11  chronic_med_condition                 25736 non-null  float64
12  child_under_6_months                 25887 non-null  float64
13  health_worker                        25903 non-null  float64
14  health_insurance                     14433 non-null  float64
15  opinion_h1n1_vacc_effective            26316 non-null  float64
16  opinion_h1n1_risk                     26319 non-null  float64
17  opinion_h1n1_sick_from_vacc            26312 non-null  float64
18  opinion_seas_vacc_effective            26245 non-null  float64
19  opinion_seas_risk                     26193 non-null  float64
20  opinion_seas_sick_from_vacc            26170 non-null  float64
21  age_group                             26707 non-null  object
22  education                             25300 non-null  object
23  race                                  26707 non-null  object
24  sex                                   26707 non-null  object
25  income_poverty                       22284 non-null  object
26  marital_status                       25299 non-null  object
27  rent_or_own                          24665 non-null  object
28  employment_status                    25244 non-null  object
29  hhs_geo_region                       26707 non-null  object
30  census_msa                           26707 non-null  object
31  household_adults                     26458 non-null  float64
32  household_children                   26458 non-null  float64
33  employment_industry                  13377 non-null  object
34  employment_occupation                13237 non-null  object
dtypes: float64(23), object(12)
memory usage: 7.3+ MB
```

- The dataset contains both object and float datatypes.
- The features of type float are 23 in number whereas the features of object data type are 12 in number

Exploring the labels dataset

```
In [5]: # Load the data
labels_df = pd.read_csv("Data/training_set_labels.csv", index_col="respondent_id")
labels_df.head()
```

```
Out[5]:
```

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

check the shape of the labels dataset

```
In [6]: labels_df.shape
```

```
Out[6]: (26707, 2)
```

- The labels dataset has 26,707 observations just like the features dataset
- This dataset has two variables (Target Variables).

EXPLORATORY DATA ANALYSIS

```
In [7]: #import files for exploration
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Exploring Labels (Target Variables)

The distribution of the target variables is explored here

This will help to understand whether the target classes are balanced

```
In [8]: fig, ax = plt.subplots(2, 1, sharex=True)

total_obs = labels_df.shape[0]

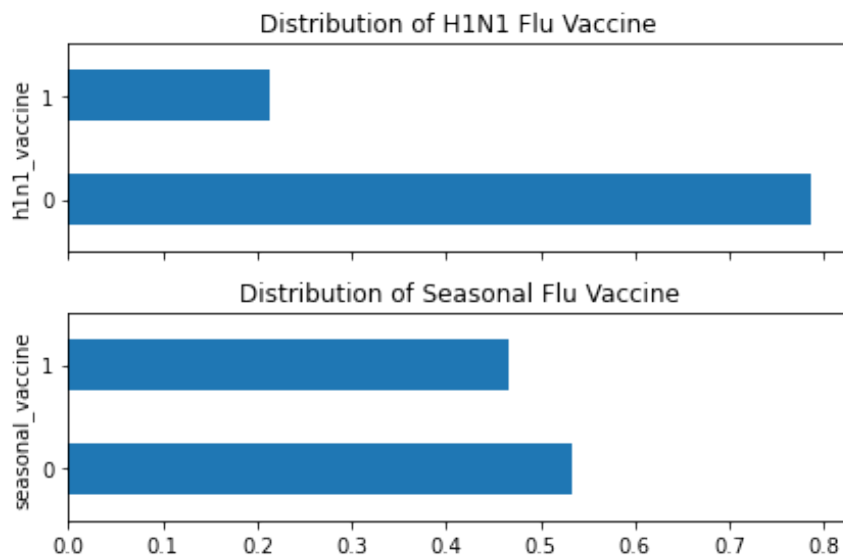
labels_df['h1n1_vaccine'].value_counts().div(total_obs).plot.barh(title="Dis
tribution of H1N1 Flu Vaccine", ax=ax[0])

ax[0].set_ylabel("h1n1_vaccine")

labels_df['seasonal_vaccine'].value_counts().div(total_obs).plot.barh(title
="Distribution of Seasonal Flu Vaccine", ax=ax[1])

ax[1].set_ylabel("seasonal_vaccine")

fig.tight_layout()
```



- From the bar chart it is clear that almost 50% of the people received the seasonal flu vaccine but only about 20% of the people received the H1N1 flu Vaccine.
- The seasonal flu vaccine has balanced classes since both class 0 (Those who did not receive the seasonal flu vaccine) and class 1 (Those that received the vaccine) have almost equal distribution.
- The H1N1 Flu Vaccine has imbalanced classes since class 0 is almost 80% whereas class 1 has only 20% distribution.

checking how the vaccination status of participants is distributed between the two types of vaccines.

The purpose of this is to get the actual figures (Distributions)


```
In [9]: pd.crosstab(labels_df["h1n1_vaccine"], labels_df["seasonal_vaccine"], margin
s=True, normalize=True)
```

```
Out[9]:
```

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

- From the table it is clear that a large percentage of the population did not receive any vaccine (49.8%) represented as 0.497810 in the table
- Approximately 29% of the population received the seasonal vaccine represented as 0.289737 in the table
- Only about 17.6% of the population received both vaccines (0.175871).
- About 3.7% (0.036582) received only the H1N1 vaccine but not the seasonal vaccine

Checking for correlation between the two target variables

```
In [10]: # code for correlation between h1n1_vaccine and seasonal_vaccine
labels_df["h1n1_vaccine"].corr(labels_df["seasonal_vaccine"])
```

```
Out[10]: 0.37714265306144684
```

- The two target variables have a positive correlation (0.377) though they are not highly correlated because the performance is below average.
- The positive correlation is demonstrated in the cross tabulation whereby 17.6% of the population had received both vaccines.

Exploring Combined data set (Both Features and Labels put together)

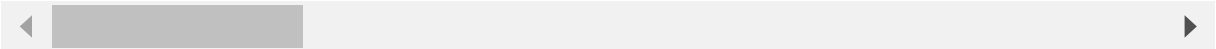
Combining the features and labels dataframes

```
In [11]: combined_df = features_df.join(labels_df)
combined_df.head()
```

```
Out[11]:
```

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

5 rows × 37 columns



check the shape of the combined dataframe

```
In [12]: combined_df.shape
```

```
Out[12]: (26707, 37)
```

- The combined df still has the same number of observations (26, 707) and 37 columns (35 features and 2 target variables)

Types of variables

The dataset contains both categorical and numerical dataset.

Here the data is divided into numerical and categorical data.

Finding categorical data

```
In [13]: categorical = [var for var in combined_df.columns if combined_df[var].dtype=
          = 'object']

          print('Total categorical variables: {} \n'.format(len(categorical)))

          print('The categorical variables are:', categorical)
```

Total categorical variables: 12

The categorical variables are: ['age_group', 'education', 'race', 'sex', 'income_poverty', 'marital_status', 'rent_or_own', 'employment_status', 'hhs_geo_region', 'census_msa', 'employment_industry', 'employment_occupation']

- There are 12 categorical variables in total.
- The columns include: age_group, education, race, sex, income_poverty, marital_status, rent_or_own, employment_status, hhs_geo_region, census_msa, employment_industry, and employment_occupation

```
In [14]: # view categorical variables
          combined_df[categorical].head()
```

Out[14]:

	age_group	education	race	sex	income_poverty	marital_status	rent_or_o
respondent_id							
0	55 - 64 Years	< 12 Years	White	Female	Below Poverty	Not Married	C
1	35 - 44 Years	12 Years	White	Male	Below Poverty	Not Married	R
2	18 - 34 Years	College Graduate	White	Male	<= \$75,000, Above Poverty	Not Married	C
3	65+ Years	12 Years	White	Female	Below Poverty	Not Married	R
4	45 - 54 Years	Some College	White	Female	<= \$75,000, Above Poverty	Married	C

Exploring Missing Values in Categorical Variables

```
In [15]: combined_df[categorical].isnull().sum()
```

```
Out[15]: 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
employment_industry 13330
employment_occupation 13470
dtype: int64
```

print categorical variables with missing values

```
In [16]: cat_missing = [var for var in categorical if combined_df[var].isnull().sum() != 0]

print(combined_df[cat_missing].isnull().sum())
```

```
education        1407
income_poverty   4423
marital_status   1408
rent_or_own      2042
employment_status 1463
employment_industry 13330
employment_occupation 13470
dtype: int64
```

- There are 7 categorical variables in the dataset which contains missing values. These are education , income_poverty , marital_status , rent_or_own employment_status , employment_industry , and employment_occupation .

Check the frequency distribution of categorical variables

In [17]: *# view frequency of categorical variables*

```
for var in categorical:
```

```
    print(combined_df[var].value_counts())
```

65+ Years	6843
55 - 64 Years	5563
45 - 54 Years	5238
18 - 34 Years	5215
35 - 44 Years	3848
Name: age_group, dtype: int64	
College Graduate	10097
Some College	7043
12 Years	5797
< 12 Years	2363
Name: education, dtype: int64	
White	21222
Black	2118
Hispanic	1755
Other or Multiple	1612
Name: race, dtype: int64	
Female	15858
Male	10849
Name: sex, dtype: int64	
<= \$75,000, Above Poverty	12777
> \$75,000	6810
Below Poverty	2697
Name: income_poverty, dtype: int64	
Married	13555
Not Married	11744
Name: marital_status, dtype: int64	
Own	18736
Rent	5929
Name: rent_or_own, dtype: int64	
Employed	13560
Not in Labor Force	10231
Unemployed	1453
Name: employment_status, dtype: int64	
lzgpxyit	4297
fpwskwrf	3265
qufhixun	3102
oxchjgsf	2859
kbazzjca	2858
bhuqouqj	2846
mlyzmhmf	2243
lrircsnp	2078
atmpeygn	2033
dqpwygqj	1126
Name: hhs_geo_region, dtype: int64	
MSA, Not Principle City	11645
MSA, Principle City	7864
Non-MSA	7198
Name: census_msa, dtype: int64	
fcxhlnwr	2468
wxleyezf	1804
ldnlellj	1231
pxcmvdjn	1037
atmlpfrs	926
arjwrbbj	871
xicduogh	851
mfikgejo	614

vjjrobsf	527
rucpzii	523
xqicxuve	511
saaquncn	338
cfqqtusy	325
nduyfdeo	286
mcubkhph	275
wlfvacwt	215
dotnnunm	201
haxffmxo	148
msuufmds	124
phxvnwax	89
qnlwzans	13

Name: employment_industry, dtype: int64

xtkaffoo	1778
mxkfnird	1509
emcorrx	1270
cmhcxjea	1247
xgwztkwe	1082
hfxkjkmi	766
qxajmpny	548
xqwwgdyp	485
kldqjy	469
uqqtjvyb	452
tfqavkke	388
ukymxvdu	372
vlluhbov	354
oijqvulv	344
ccgxvspp	341
bxpfxfdn	331
haliazsg	296
rcerts	276
xzmlyy	248
dlvbwzss	227
hodpvpew	208
dcjcpih	148
pvmttkik	98

Name: employment_occupation, dtype: int64

checking Cardinality

Cardinality helps one to know the number of unique values contained in a particular variable

```
In [18]: # check the number of variables in the categorical variables

for var in categorical:

    print(var, ' contains ', len(combined_df[var].unique()), ' labels')
```

```
age_group contains 5 labels
education contains 5 labels
race contains 4 labels
sex contains 2 labels
income_poverty contains 4 labels
marital_status contains 3 labels
rent_or_own contains 3 labels
employment_status contains 4 labels
hhs_geo_region contains 10 labels
census_msa contains 3 labels
employment_industry contains 22 labels
employment_occupation contains 24 labels
```

- Apart from employment_industry and employment_occupation that contain 22 labels and 24 labels respectively, the rest of the variables contain less than 10 labels each.
- All these variables are relatively small therefore no preprocessing needed to reduce cardinality.

Exploring Numerical Features

```
In [19]: # check numerical features

numerical = [var for var in features_df.columns if features_df[var].dtype!
            = 'object']

print('There are {} numerical variables in the features dataset\n'.format(len(numerical)))

print('The numerical variables are :', numerical)
```

There are 23 numerical variables in the features dataset

The numerical variables are : ['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']

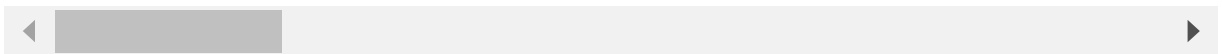
- The numeric variables in the features dataset are: 'h1n1_concern', 'h1n1_knowledge', 'behavioral_antiviral_meds', 'behavioral_avoidance', 'behavioral_face_mask', 'behavioral_wash_hands', 'behavioral_large_gatherings', 'behavioral_outside_home', 'behavioral_touch_face', 'doctor_recc_h1n1', 'doctor_recc_seasonal', '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', and 'household_children'

In [20]: `combined_df[numerical].head()`

Out[20]:

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

5 rows × 23 columns



- There are 23 numerical features in total

Checking for Missing Values in the numerical variables

```
In [21]: # Checking missing values
combined_df[numerical].isnull().sum()
```

```
Out[21]: 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
          household_adults      249
          household_children    249
          dtype: int64
```

- All the 23 variables have missing values

Imputing numeric columns using median

```
In [22]: # impute missing values with median
combined_df[numerical] = combined_df[numerical].apply(lambda x: x.fillna(x.median()))

# check whether there are still missing values.
combined_df[numerical].isnull().sum()
```

```
Out[22]: 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
household_adults         0
household_children       0
dtype: int64
```

- The data no longer has any missing values after imputing

Checking whether there is a relationship between h1n1_knowledge feature and h1n1_vaccine target variable

```
In [23]: # get the count of observations for each combination of those two variables.
counts = (combined_df[['h1n1_knowledge', 'h1n1_vaccine']]
          .groupby(['h1n1_knowledge', 'h1n1_vaccine'])
          .size()
          .unstack('h1n1_vaccine')

          )
counts
```

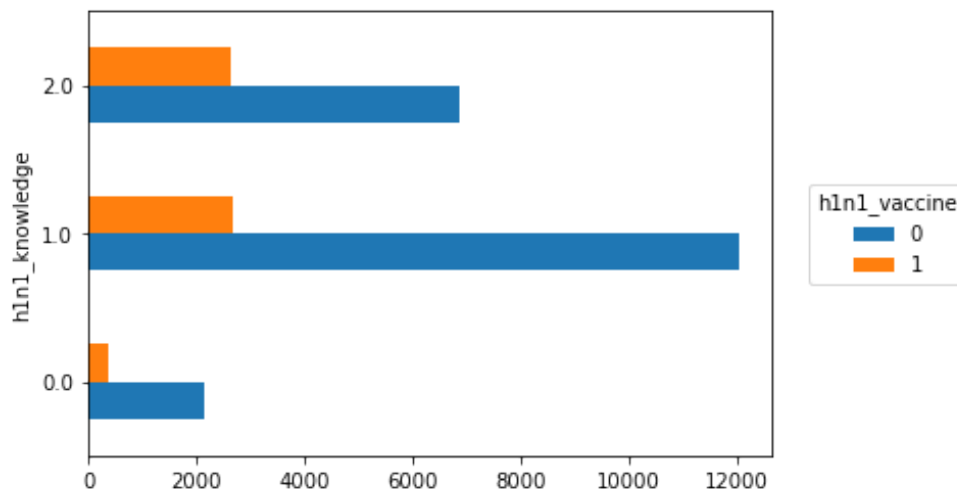
Out[23]:

	h1n1_vaccine	
	0	1
h1n1_knowledge		
0.0	2145	361
1.0	12039	2675
2.0	6849	2638

Use bar chart to compare the change in the two variables

```
In [24]: ax = counts.plot.barh()
ax.legend(
    loc='center right',
    bbox_to_anchor=(1.3, 0.5),
    title='h1n1_vaccine'
)
```

Out[24]: <matplotlib.legend.Legend at 0x1dd5441c640>



- From the bar chart, it's not easy to tell whether there is a relation between levels of knowledge of h1n1 flu and likelihood of being vaccinated since the two classes are imbalanced. That is, about 80% were not vaccinated with H1N1_Vaccine whereas 20% get vaccinated.

checking for rate of vaccination at each level

```
In [25]: h1n1_knowledge_counts = counts.sum(axis='columns')
h1n1_knowledge_counts
```

```
Out[25]: h1n1_knowledge
0.0      2506
1.0     14714
2.0      9487
dtype: int64
```

```
In [26]: # checking probabilities of h1n1_concern levels
h1n1_knowledge_probs = counts.div(h1n1_knowledge_counts, axis='index')
h1n1_knowledge_probs
```

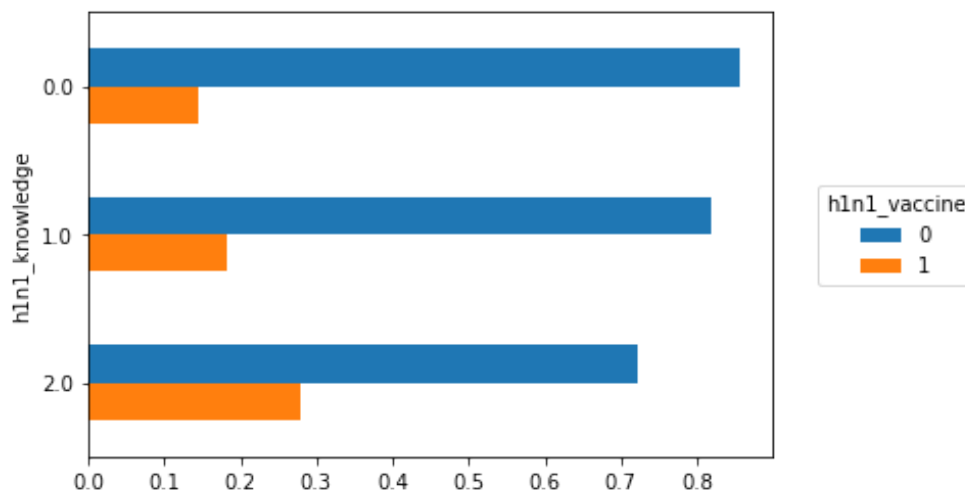
```
Out[26]:
```

	h1n1_vaccine	0	1
h1n1_knowledge			
0.0	0.855946	0.144054	
1.0	0.818200	0.181800	
2.0	0.721935	0.278065	

Bar chart for rate of vaccination for each level of h1n1_concern

```
In [27]: #
ax = h1n1_knowledge_probs.plot.barh()
ax.invert_yaxis()
ax.legend(loc='center left', bbox_to_anchor = (1.05, 0.5), title = 'h1n1_vaccine')
```

```
Out[27]: <matplotlib.legend.Legend at 0x1dd543ce2e0>
```

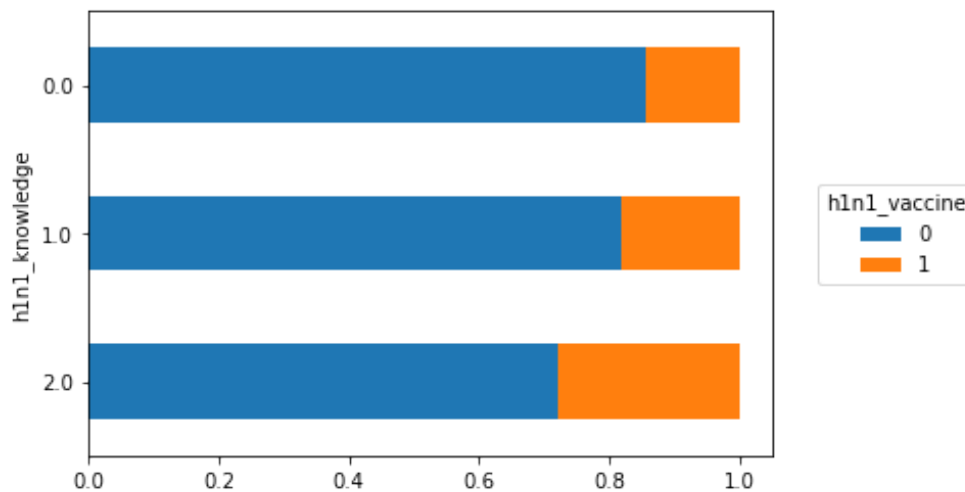


- From the bar chart it can clearly be seen that even though not many people get vaccinated with H1N1 flu vaccine, people are more likely to get vaccinated if they have more knowledge of flu vaccine.
- h1n1_knowledge can be used in modelling since there is a relationship between the level of knowledge and the number of people being vaccinated with h1n1_vaccine.

Using a stacked bar chart

```
In [28]: ax = h1n1_knowledge_probs.plot.barh(stacked=True)
ax.invert_yaxis()
ax.legend(loc='center left', bbox_to_anchor=(1.05, 0.5), title='h1n1_vaccine')
```

Out[28]: <matplotlib.legend.Legend at 0x1dd54e4b370>



plot more variables

```
In [29]: # creating a function to factor several variables
def vaccination_rate_plot(col, target, data, ax=None):
    counts = (combined_df[[target, col]].groupby([target, col]).size().unstack(target))
    group_counts = counts.sum(axis='columns')
    probabilities = counts.div(group_counts, axis='index')
    probabilities.plot(kind="barh", stacked=True, ax=ax)
    ax.invert_yaxis()
    ax.legend().remove()
```

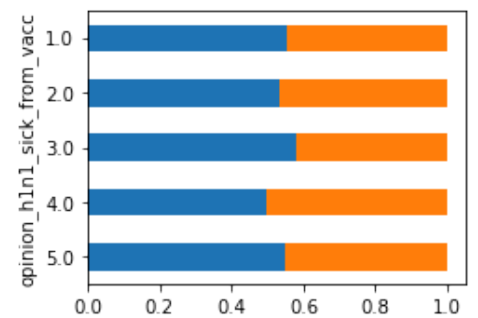
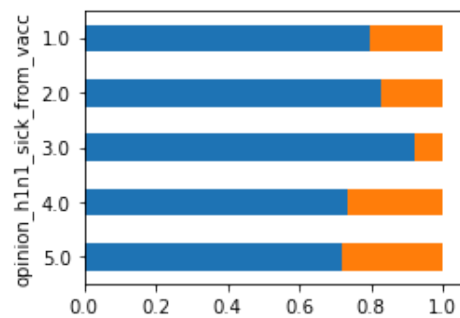
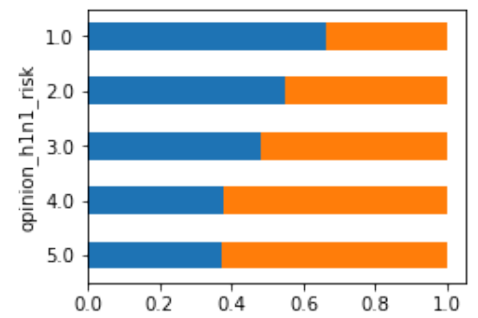
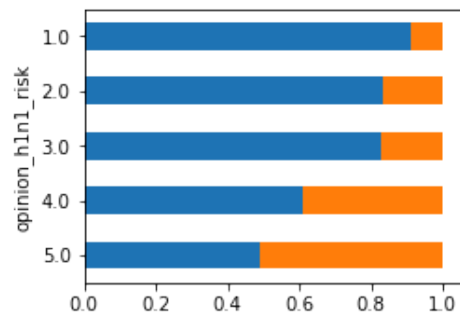
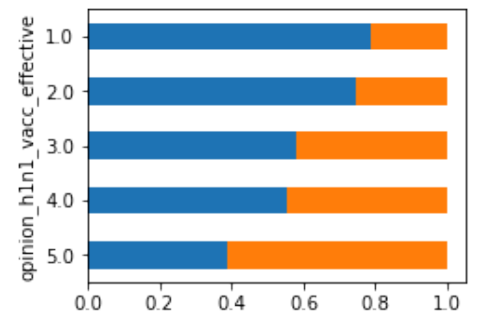
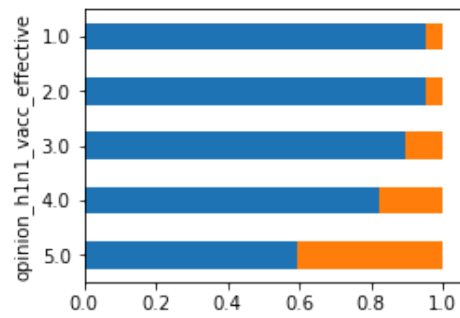
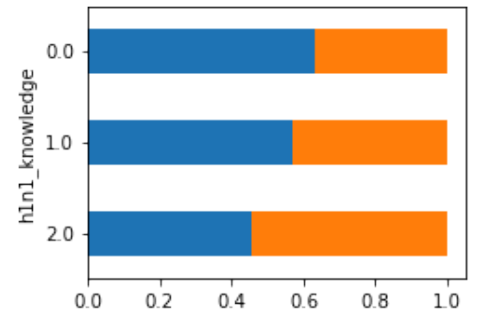
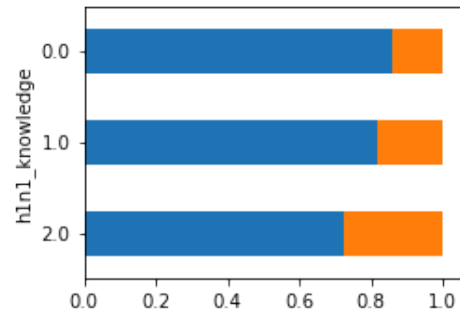
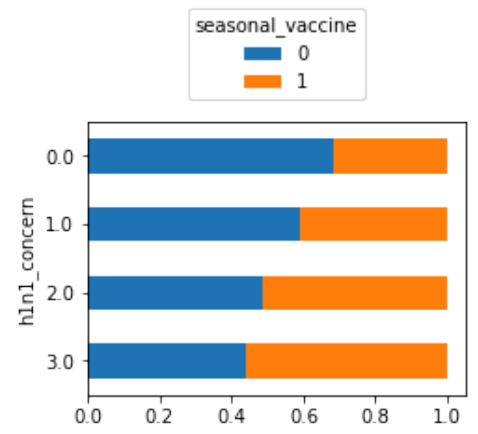
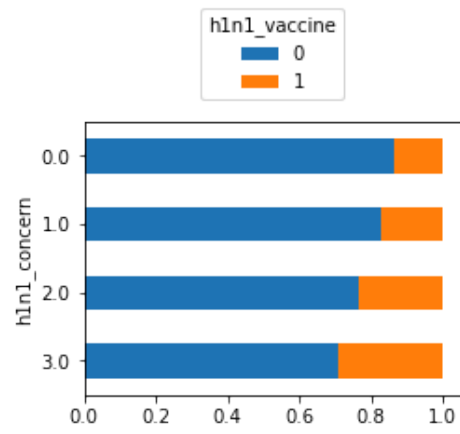
plotting selected columns against the target variables

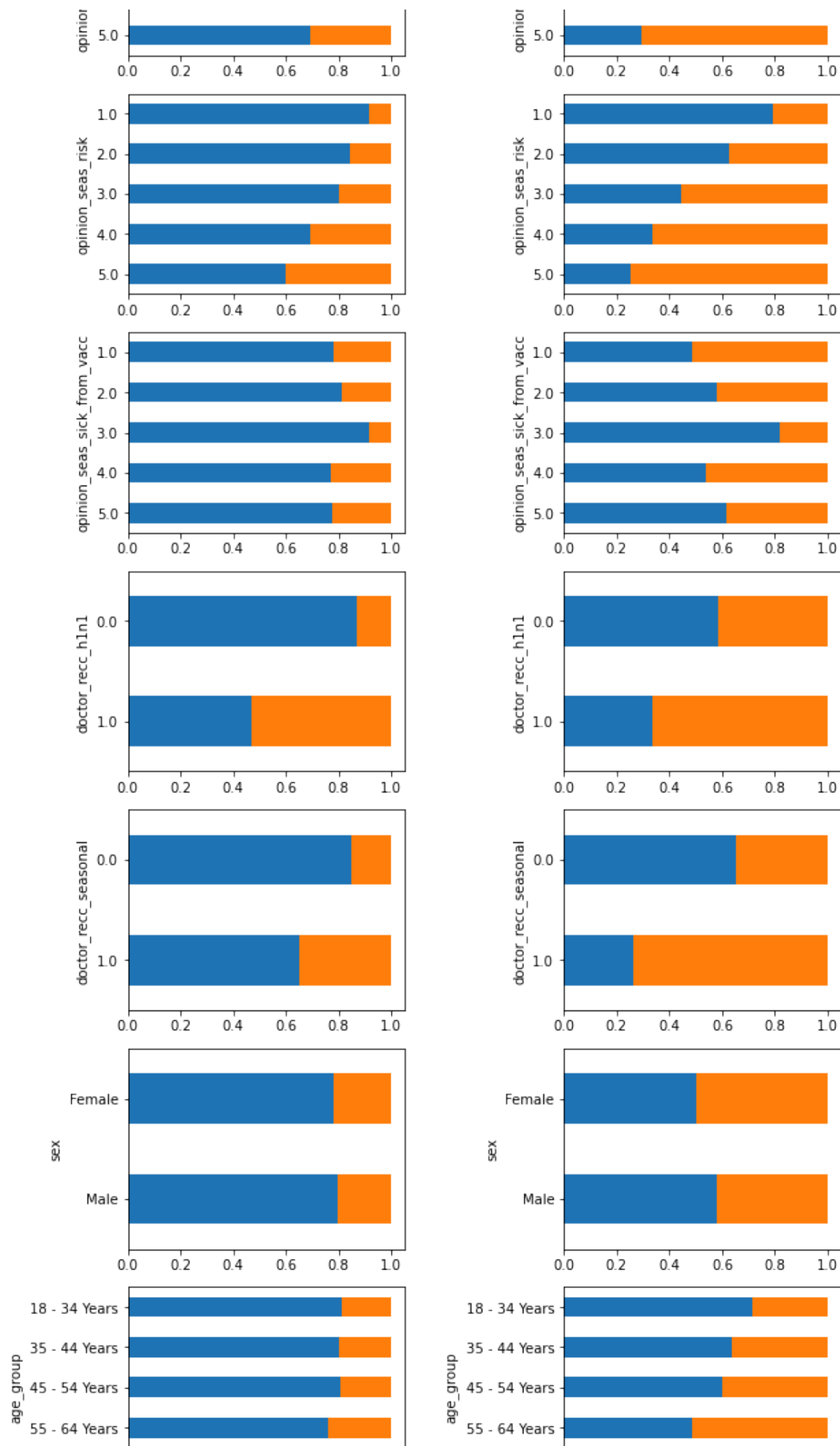
```
In [30]: # code to select columns to plot
selected_columns = [
    'h1n1_concern',
    'h1n1_knowledge',
    'opinion_h1n1_vacc_effective',
    'opinion_h1n1_risk',
    'opinion_h1n1_sick_from_vacc',
    'opinion_seas_vacc_effective',
    'opinion_seas_risk',
    'opinion_seas_sick_from_vacc',
    'doctor_recc_h1n1',
    'doctor_recc_seasonal',
    'sex',
    'age_group',
    'race',
]
```

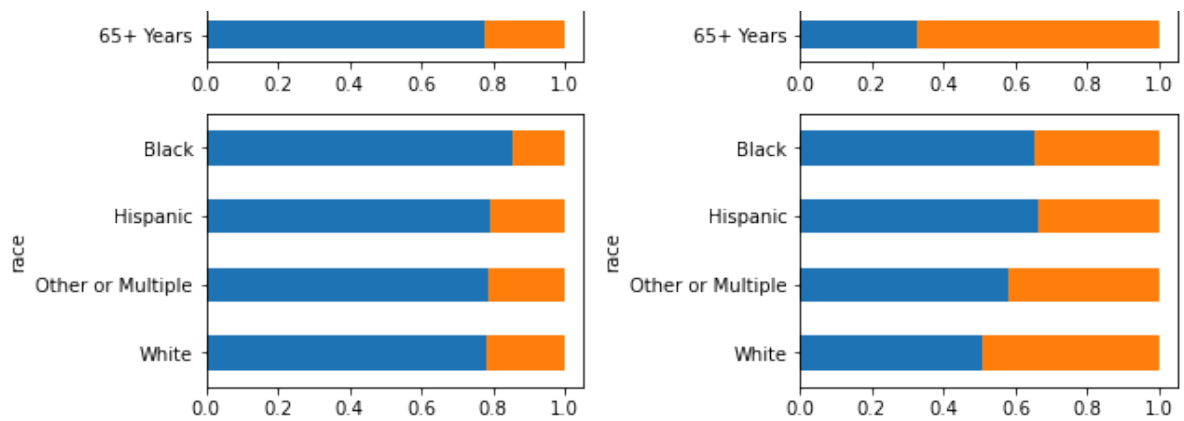
```
In [31]: # code to plot numeric variables against the target variables

fig, ax = plt.subplots(len(selected_columns), 2, figsize=(9, len(selected_columns)*2.5))
for idx, col in enumerate(selected_columns):
    vaccination_rate_plot(col, 'h1n1_vaccine', combined_df, ax=ax[idx, 0] )
    vaccination_rate_plot(col, 'seasonal_vaccine', combined_df, ax=ax[idx, 1])
ax[0, 0].legend(loc='lower center', bbox_to_anchor=(0.5, 1.05), title='h1n1_vaccine')
ax[0, 1].legend(loc='lower center', bbox_to_anchor=(0.5, 1.05), title='seasonal_vaccine')

fig.tight_layout()
```





- From the above, it is clear that concern, knowledge, opinion and doctor's recommendation of vaccines have a strong correlation with both the h1n1_vaccine and the seasonal_vaccine.
- age_group has a strong correlation with seasonal_vaccine whereby a large number of older people tend to accept the vaccine but doesn't show a strong correlation with the h1n1_vaccine.
- sex and race have high correlation with the seasonal_vaccine but does not show a strong correlation with the h1n1_vaccine.

Model Development

Model 1: Building Logistic Regression Model

```
In [32]: from sklearn.linear_model import LogisticRegression
from sklearn.multioutput import MultiOutputClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

Preprocessing

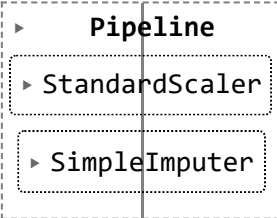
Instead of filling missing data separately then transforming the data using standard scaler, the two steps (scaling and imputing the data) will be combined together using pipeline to make work easier

Median will be used to fill missing values

```
In [33]: # combine preprocessing into pipeline object on numeric data
numerical_preprocessing_steps = Pipeline([('standard_scaler', StandardScaler()),
                                          ('simple_imputer', SimpleImputer(strategy='median'))])

numerical_preprocessing_steps
```

```
Out[33]:
```



```

  Pipeline
  |
  +-- StandardScaler
  |
  +-- SimpleImputer

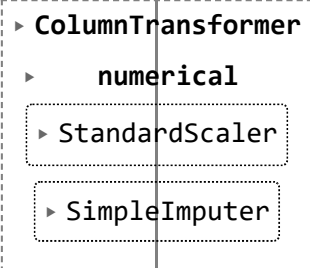
```

preprocess the data using column transformer

```
In [34]: # creating preprocessor stage of final pipeline
preprocessor = ColumnTransformer(transformers=[("numerical", numerical_preprocessing_steps, numerical)],
                                remainder="drop")

preprocessor
```

```
Out[34]:
```



```

  ColumnTransformer
  |
  +-- numerical
  |   |
  |   +-- StandardScaler
  |   |
  |   +-- SimpleImputer

```

Estimator

Since the data has two different target variables, a Multioutput classifier will be used as an estimator

```
In [35]: # code for estimator
estimators = MultiOutputClassifier(
    estimator=LogisticRegression(penalty="l2", C=1)
)

estimators
```

```
Out[35]:
└─ MultiOutputClassifier
  └─ estimator: LogisticRegression
    └─ LogisticRegression
```

combining estimator and preprocessor to form a complete pipeline

```
In [36]: model_1 = Pipeline([
    ("preprocessor", preprocessor),
    ("estimators", estimators),
])

model_1
```

```
Out[36]:
└─ Pipeline
  └─ preprocessor: ColumnTransformer
    └─ numerical
      └─ StandardScaler
      └─ SimpleImputer
  └─ estimators: MultiOutputClassifier
    └─ estimator: LogisticRegression
      └─ LogisticRegression
```

Model Training and Evaluation

The model will be trained and evaluated by splitting the train dataset into train and test.

Later the model will be evaluated using the separate test data that was provided as a separate file

```
In [37]: X_train, X_test, y_train, y_test = train_test_split(
    features_df,
    labels_df,
    test_size=0.3,
    shuffle=True,
    stratify=labels_df, # enforce even splits
    random_state=42
)
```

Model training

```
In [38]: # Train model
model_1.fit(X_train, y_train)

# Predict on evaluation set
y_preds = model_1.predict_proba(X_test)
y_preds
```

```
Out[38]: [array([[0.95138922, 0.04861078],
                [0.854483  , 0.145517  ],
                [0.97502413, 0.02497587],
                ...,
                [0.70545918, 0.29454082],
                [0.41456829, 0.58543171],
                [0.93665301, 0.06334699]]),
array([[0.93569526, 0.06430474],
        [0.69169398, 0.30830602],
        [0.98163324, 0.01836676],
        ...,
        [0.64042428, 0.35957572],
        [0.08724774, 0.91275226],
        [0.46803151, 0.53196849]])]
```

checking the shape of the

```
In [39]: print("test_probabilities[0].shape", y_preds[0].shape)
print("test_probabilities[1].shape", y_preds[1].shape)

test_probabilities[0].shape (8013, 2)
test_probabilities[1].shape (8013, 2)
```

- There are two arrays containing the (number of observations, 2) whereby the first one is for h1n1_vaccine whereas the second one is for the seasonal_vaccine which have the probabilities of 0 and 1 respectively

```
In [40]: # Display index 1 for each of the two arrays
y_prediction = pd.DataFrame(
    {
        "h1n1_vaccine": y_preds[0][:, 1],
        "seasonal_vaccine": y_preds[1][:, 1],
    },
    index = y_test.index
)
y_prediction.head()
```

```
Out[40]:
```

	h1n1_vaccine	seasonal_vaccine
respondent_id		
7572	0.048611	0.064305
3586	0.145517	0.308306
14114	0.024976	0.018367
2426	0.926849	0.951727
13147	0.087216	0.940136

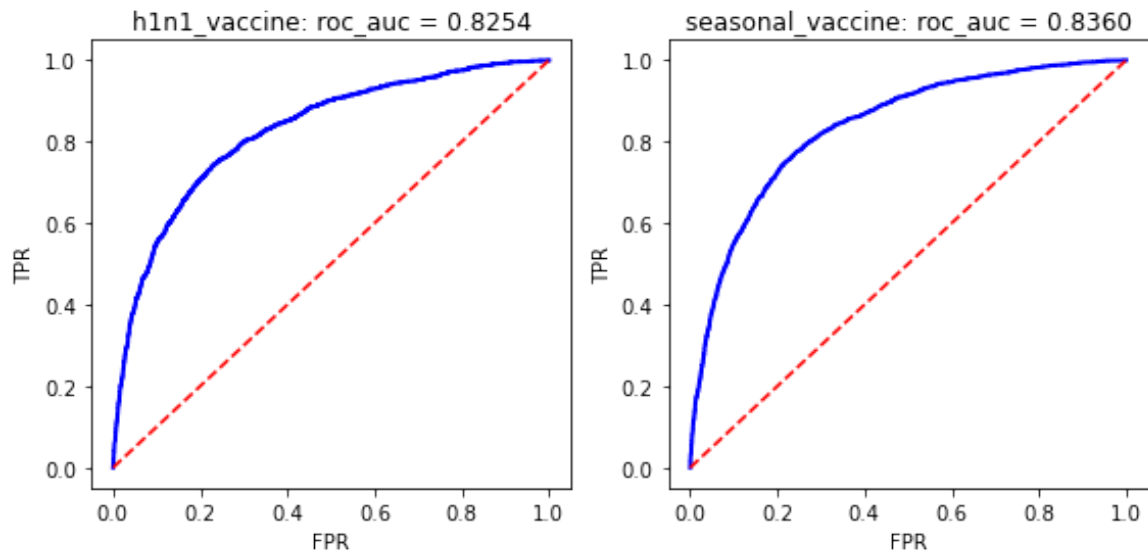
Plotting ROC Curves for Multilabel dataset for Test data

```
In [41]: # Function to create ROC Curve.
def plot_roc(y_true, y_score, label_name, ax):
    fpr, tpr, thresholds = roc_curve(y_true, y_score)
    ax.plot(fpr, tpr, color = 'blue', lw=2)
    ax.plot([0, 1], [0, 1], color='red', linestyle='--')
    ax.set_ylabel('TPR')
    ax.set_xlabel('FPR')
    ax.set_title(f"{label_name}: roc_auc = {roc_auc_score(y_true, y_score):.4f}")
```

```
In [42]: # Plotting ROC Curve

fig, ax = plt.subplots(1, 2, figsize=(8, 4))
plot_roc(y_test['h1n1_vaccine'], y_prediction['h1n1_vaccine'], 'h1n1_vaccine', ax=ax[0])
plot_roc(y_test['seasonal_vaccine'], y_prediction['seasonal_vaccine'], 'seasonal_vaccine', ax=ax[1])

fig.tight_layout()
```



- Both models seem to be performing well on test data since the AUC score on h1n1_vaccine is 0.8254 and the AUC score on seasonal_vaccine is 0.8360.
- The average performance of the model based on the two different target variables is 83%

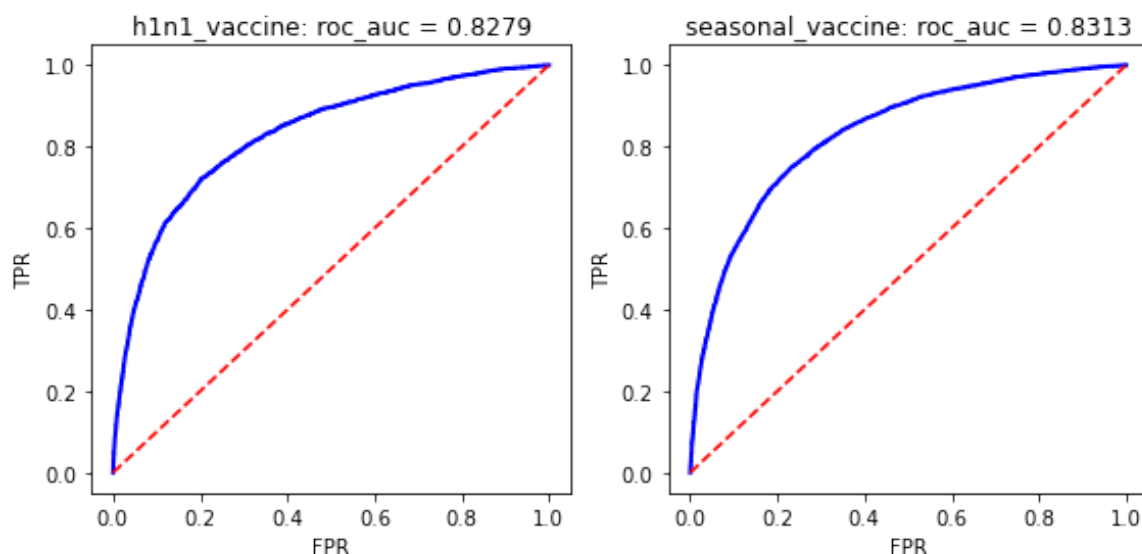
ROC_Curve for Train Data

```
In [43]: # Performance of the model base on train score
y_train_probas = model_1.predict_proba(X_train)

# Display index 1 for each of the two arrays
y_predict = pd.DataFrame(
    {
        "h1n1_vaccine": y_train_probas[0][:, 1],
        "seasonal_vaccine": y_train_probas[1][:, 1],
    },
    index = y_train.index
)
# Plotting ROC Curve

fig, ax = plt.subplots(1, 2, figsize=(8, 4))
plot_roc(y_train['h1n1_vaccine'], y_predict['h1n1_vaccine'], 'h1n1_vaccine',
ax=ax[0])
plot_roc(y_train['seasonal_vaccine'], y_predict['seasonal_vaccine'], 'seasonal_vaccine', ax=ax[1])

fig.tight_layout()
```



```
In [44]: # Average score for roc_auc
print(f"The roc_auc_score for train score is {roc_auc_score(y_train, y_predict)}")
print(f"The roc_auc_score for test score is {roc_auc_score(y_test, y_prediction)}")
```

The roc_auc_score for train score is 0.8296219970279013
The roc_auc_score for test score is 0.8307292673111957

Comments on Logistic Regression

- From the scores (83.07%) it is evident that based on the information that was shared about people's backgrounds, opinions and health behaviours, people took H1N1 and Seasonal Flu Vaccines.
- It therefore implies that a person's background, Opinion and Health Behaviour affect the uptake of a vaccine.
- The performance of the train data is 82.96% whereas the test data performance is 83.07% showing that the performance of the model in both train and test data is almost similar. This shows that the model does not overfit.
- The model performance is good but it is not perfect since it has an 83% performance out of the total 100%

To test whether a different model would perform better, a Decision Tree Classification Model (ID3) will be developed.

Model 2: Decision Tree Classifier Vanilla Model

Since the performance of the models are almost similar on H1N1 and Seasonal Flu Vaccine target variables, the Decision Tree Model will be developed using only one target variable.

A simple Decision Tree Model (Vanilla Model / Baseline Model) is built here

Importing Necessary Libraries

```
In [45]: from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve, roc_auc_score, auc
from sklearn.preprocessing import OneHotEncoder
from sklearn import tree
```

Selecting Features and Target Variable

This model will use the columns that were previously selected based on domain knowledge and also one target variable instead of a multilabel problem which was done in the Logistic Regression.

The decision to use one target variable was arrived at after the first model proved to be performing almost similarly.

```
In [46]: # Selecting Features
selected_cols_df = combined_df[selected_columns]
selected_cols_df.head()
```

```
Out[46]:
```

	h1n1_concern	h1n1_knowledge	opinion_h1n1_vacc_effective	opinion_h1n1_risk
respondent_id				
0	1.0	0.0	3.0	1.0
1	3.0	2.0	5.0	4.0
2	1.0	1.0	3.0	1.0
3	1.0	1.0	3.0	3.0
4	2.0	1.0	3.0	3.0

Checking whether there are missing values in the selected columns

```
In [47]: selected_cols_df.isna().sum()
```

```
Out[47]: h1n1_concern          0
h1n1_knowledge          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
doctor_recc_h1n1        0
doctor_recc_seasonal    0
sex                     0
age_group              0
race                   0
dtype: int64
```

Comments

- None of the selected columns have missing values.
- It is okay to proceed to feature selection since the data is clean

Selecting Features and Target Variable

```
In [48]: # selected the features and the target variables
X = selected_cols_df
y = combined_df['seasonal_vaccine']

#splitting data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

```
In [49]: # check the data types of the data

X.dtypes
```

```
Out[49]: h1n1_concern          float64
h1n1_knowledge          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
doctor_recc_h1n1         float64
doctor_recc_seasonal     float64
sex                      object
age_group                object
race                    object
dtype: object
```

- The sex , age_group and race columns are categorical columns (object).
- The rest of the columns are numerical columns (float64).

Encode categorical data as numbers

Since the data currently contains both categorical and numerical data, all the data need to be encoded as numbers. For this, the sklearn's `OneHotEncoder` from `preprocessing` will be used.

```
In [50]: # One-hot encode the training data and show the resulting DataFrame with proper column names
ohe = OneHotEncoder()

ohe.fit(X_train)
X_train_ohe = ohe.transform(X_train).toarray()

# show the result of the ohe
ohe_df = pd.DataFrame(X_train_ohe, columns=ohe.get_feature_names_out(X_train.columns))

ohe_df.head()
```

Out[50]:

	h1n1_concern_0.0	h1n1_concern_1.0	h1n1_concern_2.0	h1n1_concern_3.0	h1n1_knowledge.
0	0.0	0.0	1.0	0.0	
1	0.0	0.0	1.0	0.0	
2	0.0	0.0	0.0	1.0	
3	0.0	1.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	

5 rows × 52 columns

Train the decision tree

```
In [51]: # Create the classifier, fit it on the training data and make predictions on the test set
model_2 = DecisionTreeClassifier(criterion='entropy', random_state=42)

model_2.fit(X_train_ohe, y_train)
```

Out[51]:

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=42)
```

Evaluating the Vanilla Model using ROC_AUC Score

Creating a function for AUC_ROC Curve

```
In [52]: def plot_roc_auc(model, X_test, y_test):
# Get predicted probabilities for the positive class (usually the second
column in binary classification)
y_preds = model.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,
y_preds)

# Calculate the AUC
roc_auc = auc(false_positive_rate, true_positive_rate)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(false_positive_rate, true_positive_rate, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.grid(True)

# Show the plot
plt.show()

# Return the AUC value
return roc_auc
```

```

In [53]: X_test_ohe = ohe.transform(X_test)

# Plot ROC AUC for model_2
auc_value = plot_roc_auc(model_2, X_test_ohe, y_test)
print(f'roc_auc for test data: {auc_value:.4f}')

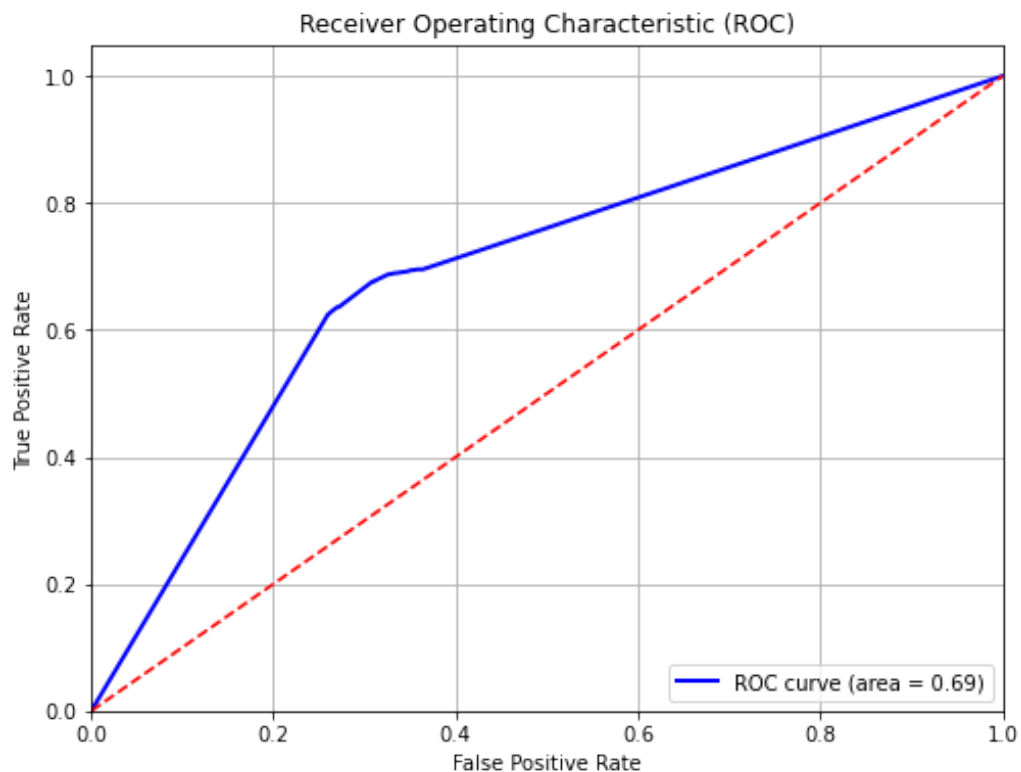
y_train_probs = model_2.predict_proba(X_train_ohe)[: , 1]

# Calculate the ROC curve
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, y_train_probs)

# Calculate the AUC
roc_auc_train = auc(false_positive_rate, true_positive_rate)

print(f'roc_auc for training data: {roc_auc_train:.4f}')

```



```

roc_auc for test data: 0.6902
roc_auc for training data: 0.9962

```

Comments

- The model performance using the decision tree is lower compared to the performance of the Logistic Regression Model.
- The score roc_auc is 69% whereas the score in the Logistic Regression is 83%.
- This might be because the number of features selected in the Logistic Regression Model were higher compared to the features selected for Decision Tree.
- The difference in score might also be caused by overfitting in the decision tree.
- The model performance on Training score is 99.62% indicating that the model is overfitted
- To solve the overfitting problem, tree pruning / hyperparameter tuning will be done in the next model

Model 3: Tree Pruning using Max_Depth

Similar Features that were used to develop the vanilla model are used here. The only difference is that the model is pruned to prevent overfitting


```

In [54]: # Create the classifier, fit it on the training data and tune using max_dept
h=3
model_3 = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)

#fit the model
model_3.fit(X_train_ohe, y_train)

# Evaluate the model
X_test_ohe = ohe.transform(X_test)

# Plot ROC AUC for model_4
auc_value = plot_roc_auc(model_3, X_test_ohe, y_test)

y_train_probs = model_3.predict_proba(X_train_ohe)[: , 1]

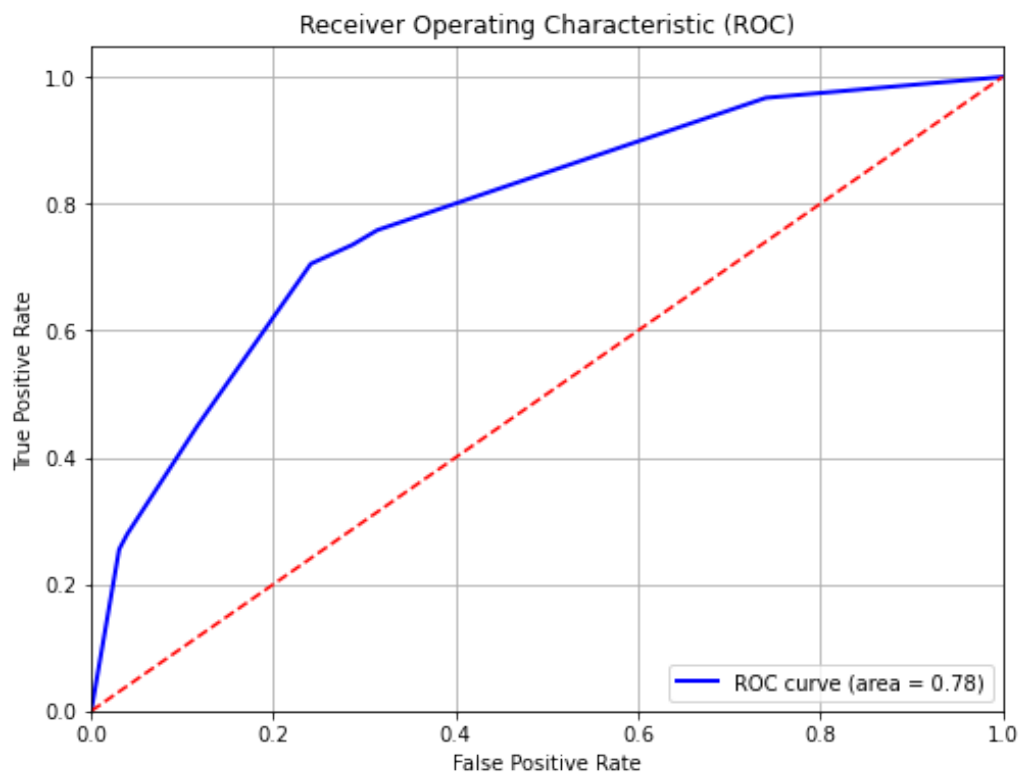
# Calculate the ROC curve
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, y_train_probs)

# Calculate the AUC
roc_auc_train = auc(false_positive_rate, true_positive_rate)

#print AUC value and the roc_auc
print(f'roc_auc for test data: {auc_value:.4f}')

print(f'roc_auc for training data: {roc_auc_train:.4f}')

```



```

roc_auc for test data: 0.7828
roc_auc for training data: 0.7856

```

Comments

- The model has significantly improved (78% performance score) compared to the Vanilla decision tree model (69% performance score) that was not pruned.
- The train (78.56%) and test (78.28%) performance are almost similar unlike the Decision Tree Baseline Model that had a performance of 99% on train and 69% on test. This indicates that pruning helps to control overfitting
- The Logistic Regression Model that had two target variables still performs better (83% performance score) compared to the two Decision Tree models.
- This might be attributed to the size of the Logistic Regression Model that had a higher number of Features compared to the features selected for Decision Tree.

Model 4: Pruned Decision Tree (ID3) with more Features Using 1 Target Variable

Here, all the numerical columns that were used to develop the Logistic Regression Model are used.

The categorical variables which had shown positive correlation with seasonal_vaccine are also used. These are sex , age_group and race columns.

```
In [55]: # select numerical features
numeric_cols_df = combined_df[numerical]

# select categorical features from the selected columns
# the columns were previously selected during EDA and used in Model 2 & 3
categorical_cols = selected_cols_df.columns[selected_cols_df.dtypes == "object"]

# Data Frame for selected columns
selected_categorical_cols_df = combined_df[categorical_cols]

print(numeric_cols_df)
print(selected_categorical_cols_df)
```

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

respondent_id	behavioral_avoidance	behavioral_face_mask \
0	0.0	0.0
1	1.0	0.0
2	1.0	0.0
3	1.0	0.0
4	1.0	0.0
...
26702	1.0	0.0
26703	1.0	0.0
26704	1.0	1.0
26705	0.0	0.0
26706	1.0	0.0

respondent_id	behavioral_wash_hands	behavioral_large_gatherings \
0	0.0	0.0
1	1.0	0.0
2	0.0	0.0
3	1.0	1.0
4	1.0	1.0
...
26702	0.0	0.0
26703	1.0	0.0
26704	1.0	1.0
26705	0.0	0.0
26706	0.0	0.0

respondent_id	behavioral_outside_home	behavioral_touch_face \
0	1.0	1.0
1	1.0	1.0
2	0.0	0.0
3	0.0	0.0
4	0.0	1.0
...
26702	1.0	0.0
26703	0.0	0.0
26704	0.0	1.0
26705	0.0	1.0
26706	0.0	0.0

respondent_id	doctor_recc_h1n1	...	health_worker	health_insurance	\
0	0.0	...	0.0	1.0	
1	0.0	...	0.0	1.0	
2	0.0	...	0.0	1.0	
3	0.0	...	0.0	1.0	
4	0.0	...	0.0	1.0	
...	
26702	0.0	...	0.0	1.0	
26703	1.0	...	1.0	1.0	
26704	0.0	...	0.0	1.0	
26705	0.0	...	0.0	0.0	
26706	0.0	...	0.0	1.0	

respondent_id	opinion_h1n1_vacc_effective	opinion_h1n1_risk	\
0	3.0	1.0	
1	5.0	4.0	
2	3.0	1.0	
3	3.0	3.0	
4	3.0	3.0	
...	
26702	3.0	1.0	
26703	4.0	2.0	
26704	4.0	4.0	
26705	3.0	1.0	
26706	5.0	1.0	

respondent_id	opinion_h1n1_sick_from_vacc	opinion_seas_vacc_effective	\
0	2.0	2.0	
1	4.0	4.0	
2	1.0	4.0	
3	5.0	5.0	
4	2.0	3.0	
...	
26702	1.0	5.0	
26703	2.0	5.0	
26704	2.0	5.0	
26705	2.0	2.0	
26706	1.0	5.0	

respondent_id	opinion_seas_risk	opinion_seas_sick_from_vacc	\
0	1.0	2.0	
1	2.0	4.0	
2	1.0	2.0	
3	4.0	1.0	
4	1.0	4.0	
...	
26702	2.0	2.0	
26703	1.0	1.0	
26704	4.0	2.0	
26705	1.0	2.0	
26706	1.0	1.0	

respondent_id	household_adults	household_children
0	0.0	0.0
1	0.0	0.0
2	2.0	0.0
3	0.0	0.0
4	1.0	0.0
...
26702	0.0	0.0
26703	1.0	0.0
26704	0.0	0.0
26705	1.0	0.0
26706	1.0	0.0

[26707 rows x 23 columns]

respondent_id	sex	age_group	race
0	Female	55 - 64 Years	White
1	Male	35 - 44 Years	White
2	Male	18 - 34 Years	White
3	Female	65+ Years	White
4	Female	45 - 54 Years	White
...
26702	Female	65+ Years	White
26703	Male	18 - 34 Years	White
26704	Female	55 - 64 Years	White
26705	Female	18 - 34 Years	Hispanic
26706	Male	65+ Years	White

[26707 rows x 3 columns]

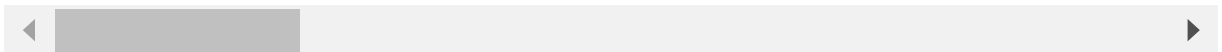
- There are two different dataframes that need to be joined before using them to develop the fourth Model

```
In [56]: # combining the dataframe
large_df = numeric_cols_df.join(selected_categorical_cols_df)
large_df.head()
```

```
Out[56]:
```

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

5 rows × 26 columns



- The new data frame consists of 26 different features and it has both numeric and categorical data (23 numeric and 3 categorical).
- The data need to be encoded using OneHotEncoder so as to convert all the features to numeric type.

```
In [57]: # selected the features and the target variables
X = large_df
y = combined_df['seasonal_vaccine']

#splitting data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, r
andom_state = 42)
```

Encoding the Data using OneHotEncoder

This will convert the entire dataset to numerical

```
In [58]: # One-hot encode the training data and show the resulting DataFrame with proper column names
ohe = OneHotEncoder()

ohe.fit(X_train)
X_train_ohe = ohe.transform(X_train).toarray()

#encode the test data
X_test_ohe = ohe.transform(X_test).toarray()

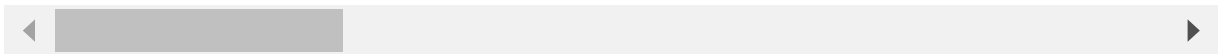
#show the result of the ohe on train dataset
ohe_df = pd.DataFrame(X_train_ohe, columns=ohe.get_feature_names_out(X_train.columns))

ohe_df.head()
```

Out[58]:

	h1n1_concern_0.0	h1n1_concern_1.0	h1n1_concern_2.0	h1n1_concern_3.0	h1n1_knowledge.
0	0.0	0.0	1.0	0.0	
1	0.0	0.0	1.0	0.0	
2	0.0	0.0	0.0	1.0	
3	0.0	1.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	

5 rows × 82 columns



Hyperparameter Tuning: Identify maximum tree depth

To create the best decision tree model, the optimal depth for the model had to be identified.

The graph showing the performance of a model as a function of the training process, AUC (Area Under the Curve), for both the training set (in blue) and the test set (in red) as the training progresses is created here.

The graph will show overfitting, underfitting and optimal values.


```

In [59]: # Identify the optimal tree depth for given data
max_depths = list(range(1, 33))
train_results = []
test_results = []

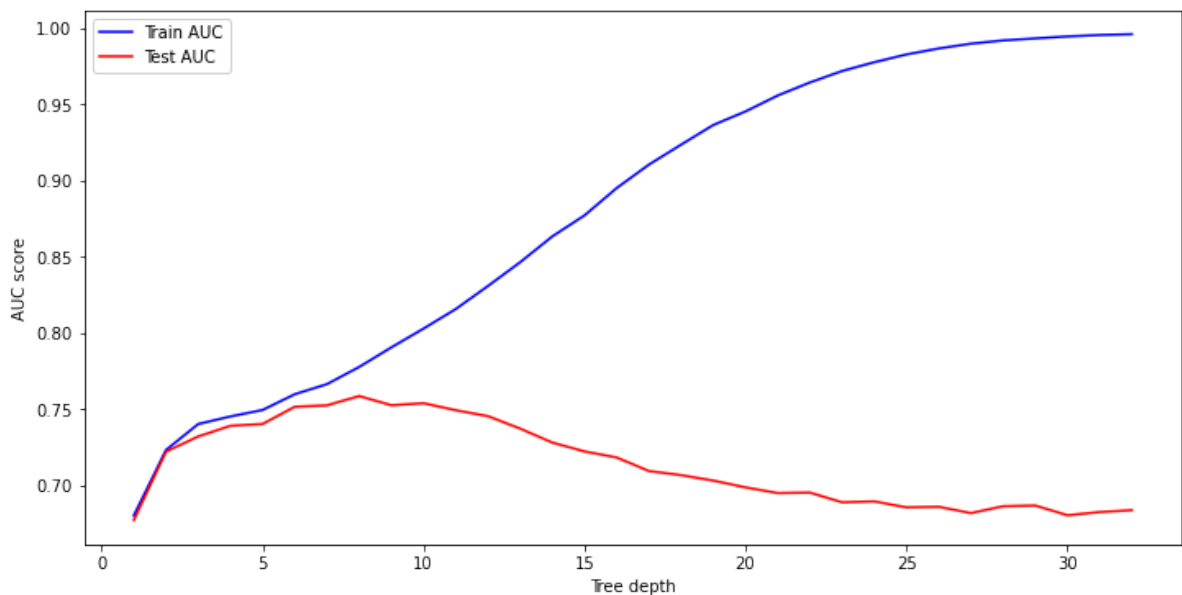
for max_depth in max_depths:
    dt = DecisionTreeClassifier(criterion='entropy', max_depth=max_depth, random_state=42)
    dt.fit(X_train_ohe, y_train)
    train_pred = dt.predict(X_train_ohe)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, train_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)

    # Add auc score to previous train results
    train_results.append(roc_auc)
    y_pred = dt.predict(X_test_ohe)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)

    # Add auc score to previous test results
    test_results.append(roc_auc)

plt.figure(figsize=(12,6))
plt.plot(max_depths, train_results, 'b', label='Train AUC')
plt.plot(max_depths, test_results, 'r', label='Test AUC')
plt.ylabel('AUC score')
plt.xlabel('Tree depth')
plt.legend()
plt.show()

```



Comments on the Maximum Depth Graph

- The AUC score on the training data (**Blue Curve**) increases continuously as the model is trained since the model is learning the patterns in the training data. The model improves at predicting the outcomes for the training set as it continues to learn the data.
- The AUC score on the test data (**Red Curve**) initially increases along with the training AUC, indicating that the model is generalizing well to unseen data at the start until it starts to decline the depth of about 7 even as the Train AUC continues to rise.
- The decline in the performance of the Test Data whereas the training data performance continues to increase indicate that the model has become overfitted.
- 7 is therefore the optimal depth that will be used to tune the model

Develop and Test the Model

```

In [60]: # Create the ID3 classifier and prune the model using max_depth=8
model_4 = DecisionTreeClassifier(criterion='entropy', max_depth=7, random_state=42)

#fit the model
model_4.fit(X_train_ohe, y_train)

# Plot ROC AUC for model_4
auc_value = plot_roc_auc(model_4, X_test_ohe, y_test)

y_train_probs = model_4.predict_proba(X_train_ohe)[: , 1]

# Calculate the ROC curve
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, y_train_probs)

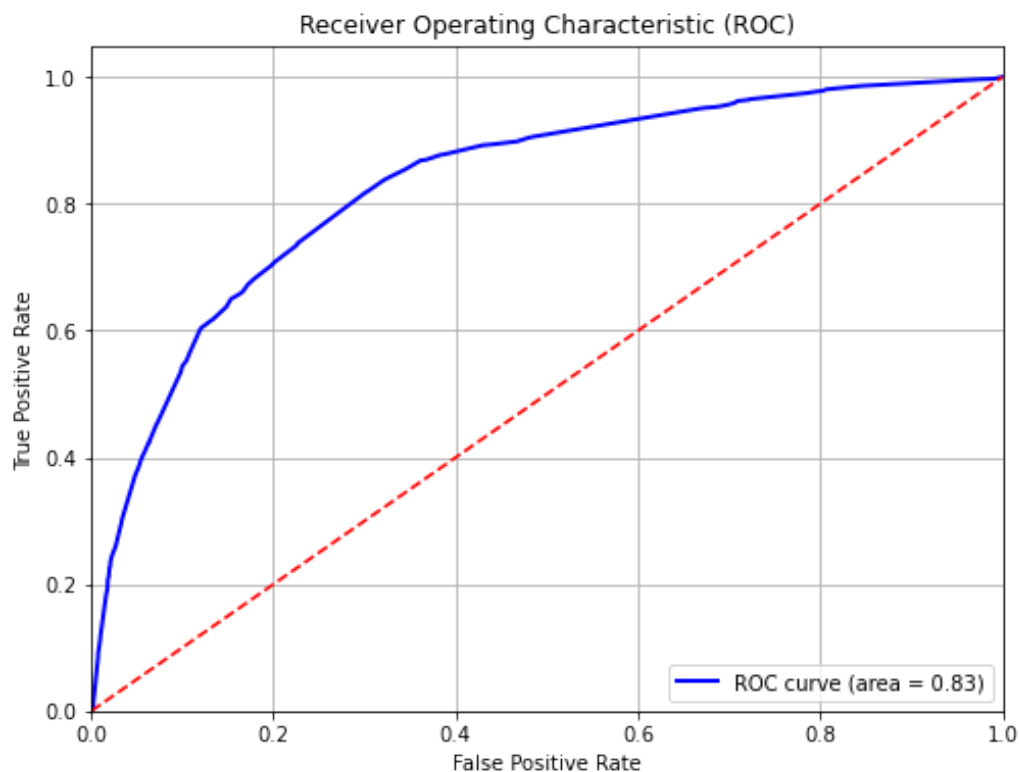
# Calculate the AUC
roc_auc_train = auc(false_positive_rate, true_positive_rate)

# print ROC_AUC values for test and train data

print(f'roc_auc for test data: {auc_value:.4f}')

print(f'roc_auc for training data: {roc_auc_train:.4f}')

```



```

roc_auc for test data: 0.8312
roc_auc for training data: 0.8491

```

Comments on Model 4 Performance

- The performance of the model has significantly improved (83% auc_roc score) compared to the second and third models which were both decision tree models but had a performance of 69% and 78% respectively.
- This improvement in performance is attributed to the use of more features and the use of correct depth limit achieved through hyperparameter tuning.
- The model however, has the same performance when compared to the Logistic Regression model that had a performance score of 83% despite model four having more features than the Logistic Regression Model.
- To further improve the performance of the Decision Tree Classifier, other tree pruning methods like (minimum samples with leaf split, minimum leaf sample size, maximum leaf nodes, and maximum features) can be used instead of using maximum depth alone.

FINDINGS AND RECOMMENDATIONS

FINDINGS

1. Logistic Regression Model has an average roc_auc score of 83.07% indicating that the model performs well in predicting H1N1 and Seasonal Flu Vaccines uptake
2. The Vanilla Decision Tree Model has an roc_auc score of 69% on test data which is a poor performance compared to the logistic regression model. This indicates that use of less variables without hyperparameter tuning results in poor model performance in Decision Trees.
3. The Vanilla model having a test score of 69% and a train score of 99.6% indicates that the model is overfitted.
4. The tuned decision tree using similar features as the Vanilla model had a roc_auc score of 78% indicating that proper pruning of decision tree can lead to good performance of a model since it helps to reduce overfitting.
5. The tuned decision tree that was developed using an optimal depth with three more features (sex , race and age_group) compared to the logistic regression that had only 23 features performed better than all the models (roc_auc of 83.12%). This indicates that with proper tuning and increasing the number of features the model can perform better in predicting vaccines.

Conclusions

1. A person's background, Opinion and Health Behaviour can be used to predict whether someone can accept a vaccine.
2. The government and health practitioners should focus on proper sensitization since the doctor's recommendation of vaccines had significant correlation with both vaccines.

Recommendations

1. When performing hyperparameter tuning for the decision tree classifier, only the Maximum Depth was used. To improve the performance of the Decision Tree Classifier, other tree pruning methods like (minimum samples with leaf split, minimum leaf sample size, maximum leaf nodes, and maximum features) can be used instead of using maximum depth alone.
2. The decision tree classifier used only one target variable (seasonal_vaccine). Both outcomes can be included in the decision tree model to determine whether the model performance will improve.