Predicting Seasonal Flu Vaccination Uptake

Final Project Submission

- Student name: Colleta Kiilu
- Student pace: **part time**
- Scheduled project submission date/time: 23/12/2024
- Instructor name: Samuel Karu
- Blog post URL: https://github.com/Collz472/Phase_3_Project_Colleta.git

Business Understanding

Project Background

Vaccination is a critical public health intervention for controlling and preventing the spread of infectious diseases. By providing immunization at an individual level and fostering herd immunity within communities, vaccines play a very important role in safeguarding global health. In 2009, the H1N1 influenza virus, commonly known as "swine flu," caused a global pandemic, resulting in an estimated 151,000 to 575,000 deaths worldwide in its first year. To combat this pandemic, an H1N1 vaccine was made available in October 2009.

The United States National 2009 H1N1 Flu Survey, conducted in late 2009 and early 2010, collected data on vaccination uptake for both H1N1 and seasonal flu. The survey explored respondents' vaccination status alongside information about their socioeconomic and demographic backgrounds, health behaviors, and opinions on vaccine efficacy and illness risk. This project therefore seeks to guide furture public health strategies by providing valuable insights into the factors influencing the vaccination patterns through a thorough analysis of the dataset provided.

Problem Statement

Understanding the factors that influence vaccination decisions is essential for designing effective public health campaigns. Despite the availability of vaccines for the seasonal flu, uptake rates varied significantly among different population groups. The challenge lies in identifying and analyzing the individual and social factors that determine vaccine adoption, which is critical for improving vaccination rates and achieving herd immunity during pandemics.

General Objective

To understand which factors influence the uptake of the seasonal flu vaccine.

Specific Objectives

1. To determine how socioeconomic factors influence an individual's uptake of the seasonal flu vaccines.

2. To determine how behavioral factors influence an individual's uptake of the seasonal flu vaccines.

3. To determine how demographic background factors influence an individual's uptake of the seasonal flu vaccines.

4. To understand how an individual's knowledge, perception, and attitude towards seasonal flu vaccines influence the uptake of the vaccines.

Research Questions

1. How do socioeconomic factors influence an individual's uptake of the seasonal flu vaccines?

2. What is the impact of behavioral factors on an individual's uptake of the seasonal flu vaccines?

3. How do demographic background factors affect an individual's uptake of the seasonal flu vaccines?

4. How does an individual's knowledge, perception, and attitude towards seasonal flu vaccines influence the uptake of the vaccines?

Data Understanding

The National 2009 H1N1 Flu Survey (NHFS) was conducted between October 2009 and June 2010 to assess the uptake of H1N1 and seasonal flu vaccines in the United States. The survey was designed as a phone-based data collection effort, targeting a representative sample of the U.S. population. Key data collected included:

1. **Vaccination Status**: Whether respondents received the H1N1 vaccine, the seasonal flu vaccine, both, or neither.

2. **Demographic Information**: Age, gender, race/ethnicity, income level, and education.

3. **Health Behaviors**: Preventative practices such as handwashing and mask usage.

4. **Opinions and Perceptions**: Views on vaccine efficacy, risks of illness, and general attitudes toward vaccination.

5. **Socioeconomic Context**: Factors that might influence access to healthcare services, such as insurance status and geographic location.

The data is composed of approximately 26,000 instances of individual data and vaccine decision information.

Data Source

The data used for this project comes from the National 2009 H1N1 Flu Survey (NHFS) and was provided courtesy of the United States National Center for Health Statistics.

U.S. Department of Health and Human Services (DHHS). National Center for Health Statistics. The National 2009 H1N1 Flu Survey. Hyattsville, MD: Centers for Disease Control and Prevention, 2012.

The data labels and features are as described in this link.

Data Science Cycle

The CRoss Industry Standard Process for Data Mining (CRISP-DM) was used for the analyzes of data. https://www.datascience-pm.com/crisp-dm-2/

```
pip install xgboost missingno
```

```
Requirement already satisfied: xgboost in c:\users\colleta.kiilu\
appdata\local\anaconda3\envs\learn-env\lib\site-packages (1.2.1)
Requirement already satisfied: missingno in c:\users\colleta.kiilu\
appdata\local\anaconda3\envs\learn-env\lib\site-packages (0.5.2)
Requirement already satisfied: numpy in c:\users\colleta.kiilu\
appdata\local\anaconda3\envs\learn-env\lib\site-packages (from
xgboost) (1.18.5)
Requirement already satisfied: scipy in c:\users\colleta.kiilu\
appdata\local\anaconda3\envs\learn-env\lib\site-packages (from
xgboost) (1.5.0)
Requirement already satisfied: seaborn in c:\users\colleta.kiilu\
appdata\local\anaconda3\envs\learn-env\lib\site-packages (from
missingno) (0.11.0)
Requirement already satisfied: matplotlib in c:\users\colleta.kiilu\
appdata\local\anaconda3\envs\learn-env\lib\site-packages (from
missingno) (3.3.1)
Requirement already satisfied: pandas>=0.23 in c:\users\colleta.kiilu\
appdata\local\anaconda3\envs\learn-env\lib\site-packages (from
seaborn->missingno) (1.1.3)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\
colleta.kiilu\appdata\local\anaconda3\envs\learn-env\lib\site-packages
(from matplotlib->missingno) (1.2.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\
colleta.kiilu\appdata\local\anaconda3\envs\learn-env\lib\site-packages
(from matplotlib->missingno) (2.8.1)
Requirement already satisfied: cycler>=0.10 in c:\users\colleta.kiilu\
appdata\local\anaconda3\envs\learn-env\lib\site-packages (from
matplotlib->missingno) (0.10.0)
Requirement already satisfied: certifi>=2020.06.20 in c:\users\
colleta.kiilu\appdata\local\anaconda3\envs\learn-env\lib\site-packages
(from matplotlib->missingno) (2020.6.20)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!
=2.1.6,>=2.0.3 in c:\users\colleta.kiilu\appdata\local\anaconda3\envs\
learn-env\lib\site-packages (from matplotlib->missingno) (2.4.7)
Requirement already satisfied: pillow>=6.2.0 in c:\users\
colleta.kiilu\appdata\local\anaconda3\envs\learn-env\lib\site-packages
(from matplotlib->missingno) (8.0.0)
```

```
Requirement already satisfied: pytz>=2017.2 in c:\users\colleta.kiilu\appdata\local\anaconda3\envs\learn-env\lib\site-packages (from pandas>=0.23->seaborn->missingno) (2020.1)
Requirement already satisfied: six>=1.5 in c:\users\colleta.kiilu\appdata\local\anaconda3\envs\learn-env\lib\site-packages (from python-dateutil>=2.1->matplotlib->missingno) (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

```
# import Libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt



from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.metrics import accuracy_score, recall_score,
precision_score, f1_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import ExtraTreesClassifier
from xgboost import XGBClassifier
from scipy.stats.mstats import winsorize
import missingno as msno

import warnings
warnings.filterwarnings('ignore')
```

Import the data sets

```
#import datasets

target = pd.read_csv("training_set_labels.csv")
train = pd.read_csv("training_set_features.csv")
test = pd.read_csv("test_set_features.csv")
```

```
# Display the first few rows of each DataFrame to ensure data loading
was successful

print("\nTarget Labels:")
print(target.head())

print("Training Data:")
print(train.head())

print("\nTest Data:")
print(test.head())
```

```
Target Labels:
   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
Training Data:
   respondent_id  h1n1_concern  h1n1_knowledge
behavioral_antiviral_meds  \
0              0           1.0             0.0
0.0
1              1           3.0             2.0
0.0
2              2           1.0             1.0
0.0
3              3           1.0             1.0
0.0
4              4           2.0             1.0
0.0

   behavioral_avoidance  behavioral_face_mask
behavioral_wash_hands  \
0                   0.0                   0.0                    0.0

1                   1.0                   0.0                    1.0

2                   1.0                   0.0                    0.0

3                   1.0                   0.0                    1.0

4                   1.0                   0.0                    1.0


   behavioral_large_gatherings  behavioral_outside_home  \
0                          0.0                      1.0
1                          0.0                      1.0
2                          0.0                      0.0
```

```
3                          1.0                         0.0
4                          1.0                         0.0

   behavioral_touch_face  ...             income_poverty
marital_status  \
0                    1.0  ...              Below Poverty     Not
Married
1                    1.0  ...              Below Poverty     Not
Married
2                    0.0  ...  <= $75,000, Above Poverty     Not
Married
3                    0.0  ...              Below Poverty     Not
Married
4                    1.0  ...  <= $75,000, Above Poverty
Married

   rent_or_own   employment_status  hhs_geo_region
census_msa  \
0          Own  Not in Labor Force        oxchjgsf
Non-MSA
1         Rent            Employed        bhuqouqj  MSA, Not Principle
City
2          Own            Employed        qufhixun  MSA, Not Principle
City
3         Rent  Not in Labor Force        lrircsnp       MSA,
Principle City
4          Own            Employed        qufhixun  MSA, Not Principle
City

   household_adults  household_children  employment_industry  \
0               0.0                 0.0                  NaN
1               0.0                 0.0             pxcmvdjn
2               2.0                 0.0             rucpziij
3               0.0                 0.0                  NaN
4               1.0                 0.0             wxleyezf

   employment_occupation
0                    NaN
1               xgwztkwe
2               xtkaffoo
3                    NaN
4               emcorrxb

[5 rows x 36 columns]

Test Data:
   respondent_id  h1n1_concern  h1n1_knowledge
behavioral_antiviral_meds  \
0          26707           2.0             2.0
0.0
```

```
1          26708          1.0             1.0
0.0
2          26709          2.0             2.0
0.0
3          26710          1.0             1.0
0.0
4          26711          3.0             1.0
1.0

   behavioral_avoidance  behavioral_face_mask
behavioral_wash_hands  \
0                   1.0                   0.0                    1.0

1                   0.0                   0.0                    0.0

2                   0.0                   1.0                    1.0

3                   0.0                   0.0                    0.0

4                   1.0                   0.0                    1.0


   behavioral_large_gatherings  behavioral_outside_home  \
0                          1.0                      0.0
1                          0.0                      0.0
2                          1.0                      1.0
3                          0.0                      0.0
4                          1.0                      1.0

   behavioral_touch_face  ...             income_poverty
marital_status  \
0                    1.0  ...                  > $75,000     Not
Married
1                    0.0  ...              Below Poverty     Not
Married
2                    1.0  ...                  > $75,000
Married
3                    0.0  ...  <= $75,000, Above Poverty
Married
4                    1.0  ...  <= $75,000, Above Poverty     Not
Married

   rent_or_own   employment_status  hhs_geo_region
census_msa  \
0         Rent            Employed        mlyzmhmf  MSA, Not Principle
City
1         Rent            Employed        bhuqouqj
Non-MSA
2          Own            Employed        lrircsnp
Non-MSA
```

```
3          Own   Not in Labor Force        lrircsnp  MSA, Not Principle
City
4          Own            Employed         lzgpxyit
Non-MSA

   household_adults  household_children  employment_industry  \
0               1.0                 0.0             atmlpfrs
1               3.0                 0.0             atmlpfrs
2               1.0                 0.0             nduyfdeo
3               1.0                 0.0                  NaN
4               0.0                 1.0             fcxhlnwr

   employment_occupation
0               hfxkjkmi
1               xqwwgdyp
2               pvmttkik
3                    NaN
4               mxkfnird

[5 rows x 36 columns]
```

Data Exploration and data Cleaning

```
 #Examine Data Shape and Size

print("Target Shape:", target.shape)
print("Train Shape:", train.shape)
print("Test Shape:", test.shape)
```

```
Target Shape: (26707, 3)
Train Shape: (26707, 36)
Test Shape: (26708, 36)
```

The target dataset contains 26,707 rows and 3 columns

The train feature dataset contains 26,707 rows and 36 columns.

The test dataset contains 26,708 rows and 36 columns

```
# Explore Data Structure

# train df
print("train Info:")
print(train.info())

#test df
print("\ntest Info:")
print(test.info())

# target df
```

```
print("\ntarget Info:")
print(target.info())
```

```
train Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26707 entries, 0 to 26706
Data columns (total 36 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   respondent_id                26707 non-null  int64
 1   h1n1_concern                 26615 non-null  float64
 2   h1n1_knowledge               26591 non-null  float64
 3   behavioral_antiviral_meds    26636 non-null  float64
 4   behavioral_avoidance         26499 non-null  float64
 5   behavioral_face_mask         26688 non-null  float64
 6   behavioral_wash_hands        26665 non-null  float64
 7   behavioral_large_gatherings  26620 non-null  float64
 8   behavioral_outside_home      26625 non-null  float64
 9   behavioral_touch_face        26579 non-null  float64
 10  doctor_recc_h1n1             24547 non-null  float64
 11  doctor_recc_seasonal         24547 non-null  float64
 12  chronic_med_condition        25736 non-null  float64
 13  child_under_6_months         25887 non-null  float64
 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
dtypes: float64(23), int64(1), object(12)
memory usage: 7.3+ MB
None

test Info:
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26708 entries, 0 to 26707
Data columns (total 36 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   respondent_id                26708 non-null  int64  
 1   h1n1_concern                 26623 non-null  float64
 2   h1n1_knowledge               26586 non-null  float64
 3   behavioral_antiviral_meds    26629 non-null  float64
 4   behavioral_avoidance         26495 non-null  float64
 5   behavioral_face_mask         26689 non-null  float64
 6   behavioral_wash_hands        26668 non-null  float64
 7   behavioral_large_gatherings  26636 non-null  float64
 8   behavioral_outside_home      26626 non-null  float64
 9   behavioral_touch_face        26580 non-null  float64
 10  doctor_recc_h1n1             24548 non-null  float64
 11  doctor_recc_seasonal         24548 non-null  float64
 12  chronic_med_condition        25776 non-null  float64
 13  child_under_6_months         25895 non-null  float64
 14  health_worker                25919 non-null  float64
 15  health_insurance             14480 non-null  float64
 16  opinion_h1n1_vacc_effective  26310 non-null  float64
 17  opinion_h1n1_risk            26328 non-null  float64
 18  opinion_h1n1_sick_from_vacc  26333 non-null  float64
 19  opinion_seas_vacc_effective  26256 non-null  float64
 20  opinion_seas_risk            26209 non-null  float64
 21  opinion_seas_sick_from_vacc  26187 non-null  float64
 22  age_group                    26708 non-null  object 
 23  education                    25301 non-null  object 
 24  race                         26708 non-null  object 
 25  sex                          26708 non-null  object 
 26  income_poverty               22211 non-null  object 
 27  marital_status               25266 non-null  object 
 28  rent_or_own                  24672 non-null  object 
 29  employment_status            25237 non-null  object 
 30  hhs_geo_region               26708 non-null  object 
 31  census_msa                   26708 non-null  object 
 32  household_adults             26483 non-null  float64
 33  household_children           26483 non-null  float64
 34  employment_industry          13433 non-null  object 
 35  employment_occupation        13282 non-null  object 
dtypes: float64(23), int64(1), object(12)
memory usage: 7.3+ MB
None

target 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
None
```

- The training dataset contains float and object datatypes.
- The dataset shall remain as it is since it matches the test dataset data types.

```
# Explore train Summary Statistics

train.describe().T
```

```
                             count          mean          std  min
25%  \
respondent_id              26707.0  13353.000000  7709.791156  0.0
6676.5
h1n1_concern               26615.0      1.618486     0.910311  0.0
1.0
h1n1_knowledge             26591.0      1.262532     0.618149  0.0
1.0
behavioral_antiviral_meds  26636.0      0.048844     0.215545  0.0
0.0
behavioral_avoidance       26499.0      0.725612     0.446214  0.0
0.0
behavioral_face_mask       26688.0      0.068982     0.253429  0.0
0.0
behavioral_wash_hands      26665.0      0.825614     0.379448  0.0
1.0
behavioral_large_gatherings  26620.0    0.358640     0.479610  0.0
0.0
behavioral_outside_home    26625.0      0.337315     0.472802  0.0
0.0
behavioral_touch_face      26579.0      0.677264     0.467531  0.0
0.0
doctor_recc_h1n1           24547.0      0.220312     0.414466  0.0
0.0
doctor_recc_seasonal       24547.0      0.329735     0.470126  0.0
0.0
chronic_med_condition      25736.0      0.283261     0.450591  0.0
0.0
child_under_6_months       25887.0      0.082590     0.275266  0.0
0.0
health_worker              25903.0      0.111918     0.315271  0.0
0.0
health_insurance           14433.0      0.879720     0.325300  0.0
1.0
```

```
opinion_h1n1_vacc_effective  26316.0       3.850623     1.007436  1.0
3.0
opinion_h1n1_risk            26319.0       2.342566     1.285539  1.0
1.0
opinion_h1n1_sick_from_vacc  26312.0       2.357670     1.362766  1.0
1.0
opinion_seas_vacc_effective  26245.0       4.025986     1.086565  1.0
4.0
opinion_seas_risk            26193.0       2.719162     1.385055  1.0
2.0
opinion_seas_sick_from_vacc  26170.0       2.118112     1.332950  1.0
1.0
household_adults             26458.0       0.886499     0.753422  0.0
0.0
household_children           26458.0       0.534583     0.928173  0.0
0.0

                                 50%      75%      max
respondent_id                13353.0  20029.5  26706.0
h1n1_concern                     2.0      2.0      3.0
h1n1_knowledge                   1.0      2.0      2.0
behavioral_antiviral_meds        0.0      0.0      1.0
behavioral_avoidance             1.0      1.0      1.0
behavioral_face_mask             0.0      0.0      1.0
behavioral_wash_hands            1.0      1.0      1.0
behavioral_large_gatherings      0.0      1.0      1.0
behavioral_outside_home          0.0      1.0      1.0
behavioral_touch_face            1.0      1.0      1.0
doctor_recc_h1n1                 0.0      0.0      1.0
doctor_recc_seasonal             0.0      1.0      1.0
chronic_med_condition            0.0      1.0      1.0
child_under_6_months             0.0      0.0      1.0
health_worker                    0.0      0.0      1.0
health_insurance                 1.0      1.0      1.0
opinion_h1n1_vacc_effective      4.0      5.0      5.0
opinion_h1n1_risk                2.0      4.0      5.0
opinion_h1n1_sick_from_vacc      2.0      4.0      5.0
opinion_seas_vacc_effective      4.0      5.0      5.0
opinion_seas_risk                2.0      4.0      5.0
opinion_seas_sick_from_vacc      2.0      4.0      5.0
household_adults                 1.0      1.0      3.0
household_children               0.0      1.0      3.0
```

From the summary statistics, **data does not have outliers** since the data contains the binary and categorical types of data.

Check column names

```
# Print column names and data types of the "target" DataFrame
```

```python
print("\nTarget Labels Columns:")
print(target.columns)
```

```
Target Labels Columns:
Index(['respondent_id', 'h1n1_vaccine', 'seasonal_vaccine'],
dtype='object')
```

```python
# Print column names and data types of the "train" DataFrame

print("\nTrain Labels Columns:")
print(train.columns)
```

```
Train Labels Columns:
Index(['respondent_id', 'h1n1_concern', 'h1n1_knowledge',
       'behavioral_antiviral_meds', 'behavioral_avoidance',
       'behavioral_face_mask', 'behavioral_wash_hands',
       'behavioral_large_gatherings', 'behavioral_outside_home',
       'behavioral_touch_face', 'doctor_recc_h1n1',
'doctor_recc_seasonal',
       'chronic_med_condition', 'child_under_6_months',
'health_worker',
       'health_insurance', 'opinion_h1n1_vacc_effective',
'opinion_h1n1_risk',
       'opinion_h1n1_sick_from_vacc', 'opinion_seas_vacc_effective',
       'opinion_seas_risk', 'opinion_seas_sick_from_vacc',
'age_group',
       'education', 'race', 'sex', 'income_poverty', 'marital_status',
       'rent_or_own', 'employment_status', 'hhs_geo_region',
'census_msa',
       'household_adults', 'household_children',
'employment_industry',
       'employment_occupation'],
      dtype='object')
```

```python
# Print column names and data types of the "test" DataFrame
print("\nTest Labels Columns:")
print(test.columns)
```

```
Test Labels Columns:
Index(['respondent_id', 'h1n1_concern', 'h1n1_knowledge',
       'behavioral_antiviral_meds', 'behavioral_avoidance',
       'behavioral_face_mask', 'behavioral_wash_hands',
       'behavioral_large_gatherings', 'behavioral_outside_home',
       'behavioral_touch_face', 'doctor_recc_h1n1',
'doctor_recc_seasonal',
       'chronic_med_condition', 'child_under_6_months',
'health_worker',
       'health_insurance', 'opinion_h1n1_vacc_effective',
```

```
'opinion_h1n1_risk',
       'opinion_h1n1_sick_from_vacc', 'opinion_seas_vacc_effective',
       'opinion_seas_risk', 'opinion_seas_sick_from_vacc',
'age_group',
       'education', 'race', 'sex', 'income_poverty', 'marital_status',
       'rent_or_own', 'employment_status', 'hhs_geo_region',
'census_msa',
       'household_adults', 'household_children',
'employment_industry',
       'employment_occupation'],
      dtype='object')
```

Standardizing Column Names

Clean Column Names to remove spaces, special characters, and standardize case.

```
target.columns = target.columns.str.strip().str.lower().str.replace('
', '_')

train.columns = train.columns.str.strip().str.lower().str.replace(' ',
'_')

test.columns = test.columns.str.strip().str.lower().str.replace(' ',
'_')
```

Remove Unnecessary Columns

Drop information for HINI

There are two potential targets in these datasets: whether the survey respondent received the seasonal flu vaccine, or whether the respondent received the H1N1 flu vaccine. For my minimum viable project, i will choose just one of these potential targets therefore, i will drop all information related to H1N1 flu and maintain the seasonal flu vaccination. The columns are not directly relevant to the current analysis

```
# Drop columns related to H1N1 flu from the "target" DataFrame
target.drop(columns=["h1n1_vaccine"], axis=1, inplace=True)

# Drop columns related to H1N1 flu from the "train" DataFrame
train.drop(columns=["h1n1_concern","h1n1_knowledge","doctor_recc_h1n1"
,"opinion_h1n1_vacc_effective","opinion_h1n1_risk","opinion_h1n1_sick_
from_vacc"], axis=1, inplace=True)

# Drop columns related to H1N1 flu from the "test" DataFrame
test.drop(columns=["h1n1_concern","h1n1_knowledge","doctor_recc_h1n1",
"opinion_h1n1_vacc_effective","opinion_h1n1_risk","opinion_h1n1_sick_f
rom_vacc"], axis=1, inplace=True)
```

```python
# Cconfirm if the columns dropped

print("target Shape:", target.shape)
print("train Shape:", train.shape)
print("set Shape:", test.shape)
```

```
target Shape: (26707, 2)
train Shape: (26707, 30)
set Shape: (26708, 30)
```

The target dataset now contains 26,707 rows and 2 columns. 1 columns was successfuly dropped

The train feature dataset now contains 26,707 rows and 30columns. 6 columns were successfuly dropped

The test dataset now contains 26,708 rows and 30 columns. 6 columns were successfuly dropped

Check for duplicates

```python
# Check duplicates
duplicates_train = train.duplicated()
duplicates_test = test.duplicated()
num_duplicates_train = duplicates_train.sum()
num_duplicates_test = duplicates_test.sum()

print("Duplicate Rows in Train Set:", num_duplicates_train)
print("Duplicate Rows in Test Set:", num_duplicates_test)
```

```
Duplicate Rows in Train Set: 0
Duplicate Rows in Test Set: 0
```

The train and test data set does not have any duplicates

Checking for Missing values

```python
# check the percentage of missing values per column

missing=(train.isnull().sum()/len(train))*100

# show columns with the highest missing percentages
missing = missing.sort_values(ascending=False)

print("Missing values:")
print(missing)
```

```
Missing values:
employment_occupation          50.436215
employment_industry            49.912008
health_insurance               45.957989
```

```
income_poverty                 16.561201
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
household_adults                0.932340
household_children              0.932340
behavioral_avoidance            0.778822
behavioral_touch_face           0.479275
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
age_group                       0.000000
race                            0.000000
sex                             0.000000
hhs_geo_region                  0.000000
census_msa                      0.000000
respondent_id                   0.000000
dtype: float64
```

Observations

1. From the above information, the employment_occupation column, employment_industry column and the health_insurance column have the highest number of missing values at 50.4%, 49.9% and 45.96% respectively

2. From the data set, there is a strong relationship between the employement_status cloumn and the employment_occupation and employement_industry columns which have the highest null values. The exected responses in the employement_status column was either; 1) employed, 2) unemployed or 3) not in labour force. It is therfore expected that those who responded 'unemplyed' or 'not in labour force', would leave the the employment_occupation and employement_industry columns blank.

3. Therefore, 10,231 of the null values for the employment_occupation and employemnt_indstry will be viewed as "not applicable" as those are the the respondents who answered "Not in Labor Force" for employment_status. The information will be viewed as not applicable as opposed to respondendt not choosing to respond

4. Similarly, an additional 1,453 of the null values in the employment_occupation and employmet_industry representing all unemployed individuals will be viewed as not applicable.

Handling Missing values

1. Employment Columns (employment_industry and employment_occupation columns)

- For respondents marked as "Unemployed" in the `employment_status` column, the `employment_industry` will be updated to "not employed".
- For respondents marked as "Unemployed" in the `employment_status` column, the `employment_occupation` will be updated to "not employed".
- For respondents marked as "Not in Labor Force" in the `employment_status` column, the `employment_industry` will be updated to "not employed".
- For respondents marked as "Not in Labor Force" in the `employment_status` column, the `employment_occupation` is also updated to "not employed".

By labeling these cases as `not employed`, the missing values are now replaced with meaningful information, which can be utilized for further analysis without introducing any biased assumptions.

```
## if a person is unemployed, change their "employment_industry" to
"not_employed"
train.loc[train["employment_status"] == "Unemployed",
"employment_industry"] = "not employed"

## if a person is unemployed, change their "employment_occupation" to
"not_employed"
train.loc[train["employment_status"] == "Unemployed",
"employment_occupation"] = "not employed"

## if a person is not in the labor force, change their
"employment_industry" to "not_employed"
train.loc[train["employment_status"] == "Not in Labor Force",
"employment_industry"] = "not employed"

## if a person is not in the labor force, change their
"employment_occupation" to "not_employed"
train.loc[train["employment_status"] == "Not in Labor Force",
"employment_occupation"] = "not employed"
```

2. `health_insurance` Column

- Missing values in the "health_insurance" column are filled with 0, assuming that those with missing values likely do not have health insurance coverage, possibly due to financial constraints associated with a higher poverty index.

```
#Filling the missing values in health insurance with 0 assumptions is
that there is correlation between the poverty index, employeemnt status
and health cover
```

```
train["health_insurance"].fillna(0, inplace=True)

# confimr missing values again
missing=(train.isnull().sum()/len(train))*100

missing = missing.sort_values(ascending=False)

print("Missing values:")
print(missing)
```

```
Missing values:
income_poverty                  16.561201
doctor_recc_seasonal             8.087767
rent_or_own                      7.645936
employment_occupation            6.687385
employment_industry              6.163178
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
household_children               0.932340
household_adults                 0.932340
behavioral_avoidance             0.778822
behavioral_touch_face            0.479275
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
age_group                        0.000000
race                             0.000000
census_msa                       0.000000
hhs_geo_region                   0.000000
health_insurance                 0.000000
sex                              0.000000
respondent_id                    0.000000
dtype: float64
```

Check for unique values

```
total_observations = len(train)

for col in train.columns:
    print(f"Column: {col}")
```

```
    freq_percentage = train[col].value_counts(dropna=False) /
total_observations * 100
    print(freq_percentage)
    print()
```

```
Column: respondent_id
2047     0.003744
7657     0.003744
3371     0.003744
13612    0.003744
15661    0.003744
           ...
12979    0.003744
2740     0.003744
693      0.003744
6838     0.003744
0        0.003744
Name: respondent_id, Length: 26707, dtype: float64

Column: behavioral_antiviral_meds
0.0    94.862770
1.0     4.871382
NaN     0.265848
Name: behavioral_antiviral_meds, dtype: float64

Column: behavioral_avoidance
1.0    71.996106
0.0    27.225072
NaN     0.778822
Name: behavioral_avoidance, dtype: float64

Column: behavioral_face_mask
0.0    93.035534
1.0     6.893324
NaN     0.071142
Name: behavioral_face_mask, dtype: float64

Column: behavioral_wash_hands
1.0    82.431572
0.0    17.411166
NaN     0.157262
Name: behavioral_wash_hands, dtype: float64

Column: behavioral_large_gatherings
0.0    63.927060
1.0    35.747182
NaN     0.325757
Name: behavioral_large_gatherings, dtype: float64

Column: behavioral_outside_home
```

```
0.0    66.065077
1.0    33.627888
NaN     0.307036
Name: behavioral_outside_home, dtype: float64

Column: behavioral_touch_face
1.0    67.401805
0.0    32.118920
NaN     0.479275
Name: behavioral_touch_face, dtype: float64

Column: doctor_recc_seasonal
0.0    61.605572
1.0    30.306661
NaN     8.087767
Name: doctor_recc_seasonal, dtype: float64

Column: chronic_med_condition
0.0    69.068035
1.0    27.296214
NaN     3.635751
Name: chronic_med_condition, dtype: float64

Column: child_under_6_months
0.0    88.924252
1.0     8.005392
NaN     3.070356
Name: child_under_6_months, dtype: float64

Column: health_worker
0.0    86.134721
1.0    10.854832
NaN     3.010447
Name: health_worker, dtype: float64

Column: health_insurance
0.0    52.458157
1.0    47.541843
Name: health_insurance, dtype: float64

Column: opinion_seas_vacc_effective
4.0    43.542891
5.0    37.342270
2.0     8.260007
1.0     4.571835
3.0     4.553113
NaN     1.729884
Name: opinion_seas_vacc_effective, dtype: float64

Column: opinion_seas_risk
```

```
2.0    33.526791
4.0    28.569289
1.0    22.368667
5.0    11.075748
3.0     2.534916
NaN     1.924589
Name: opinion_seas_risk, dtype: float64

Column: opinion_seas_sick_from_vacc
1.0    44.445277
2.0    28.580522
4.0    18.167522
5.0     6.444003
NaN     2.010709
3.0     0.351968
Name: opinion_seas_sick_from_vacc, dtype: float64

Column: age_group
65+ Years        25.622496
55 - 64 Years    20.829745
45 - 54 Years    19.612836
18 - 34 Years    19.526716
35 - 44 Years    14.408208
Name: age_group, dtype: float64

Column: education
College Graduate    37.806568
Some College        26.371363
12 Years            21.705920
< 12 Years           8.847868
NaN                  5.268282
Name: education, dtype: float64

Column: race
White                79.462313
Black                 7.930505
Hispanic              6.571311
Other or Multiple     6.035871
Name: race, dtype: float64

Column: sex
Female    59.377691
Male      40.622309
Name: sex, dtype: float64

Column: income_poverty
<= $75,000, Above Poverty    47.841390
> $75,000                    25.498933
NaN                          16.561201
Below Poverty                10.098476
```

```
Name: income_poverty, dtype: float64

Column: marital_status
Married        50.754484
Not Married    43.973490
NaN             5.272026
Name: marital_status, dtype: float64

Column: rent_or_own
Own     70.153892
Rent    22.200172
NaN      7.645936
Name: rent_or_own, dtype: float64

Column: employment_status
Employed              50.773206
Not in Labor Force    38.308309
NaN                    5.477965
Unemployed             5.440521
Name: employment_status, dtype: float64

Column: hhs_geo_region
lzgpxyit    16.089415
fpwskwrf    12.225259
qufhixun    11.614932
oxchjgsf    10.705059
kbazzjca    10.701314
bhuqouqj    10.656382
mlyzmhmf     8.398547
lrircsnp     7.780732
atmpeygn     7.612236
dqpwygqj     4.216123
Name: hhs_geo_region, dtype: float64

Column: census_msa
MSA, Not Principle  City    43.602801
MSA, Principle City         29.445464
Non-MSA                     26.951736
Name: census_msa, dtype: float64

Column: household_adults
1.0    54.195529
0.0    30.164376
2.0    10.495376
3.0     4.212379
NaN     0.932340
Name: household_adults, dtype: float64

Column: household_children
0.0    69.914255
```

```
1.0    11.888269
2.0    10.723780
3.0     6.541356
NaN     0.932340
Name: household_children, dtype: float64

Column: employment_industry
not employed    43.748830
fcxhlnwr         9.241023
wxleyezf         6.754783
NaN              6.163178
ldnlellj         4.609278
pxcmvdjn         3.882877
atmlpfrs         3.467256
arjwrbjb         3.261317
xicduogh         3.186431
mfikgejo         2.299023
vjjrobsf         1.973265
rucpziij         1.958288
xqicxuve         1.913356
saaquncn         1.265586
cfqqtusy         1.216909
nduyfdeo         1.070880
mcubkhph         1.029693
wlfvacwt         0.805032
dotnnunm         0.752612
haxffmxo         0.554162
msuufmds         0.464298
phxvnwax         0.333246
qnlwzans         0.048676
Name: employment_industry, dtype: float64

Column: employment_occupation
not employed    43.748830
NaN              6.687385
xtkaffoo         6.657431
mxkfnird         5.650204
emcorrxb         4.755308
cmhcxjea         4.669188
xgwztkwe         4.051372
hfxkjkmi         2.868162
qxajmpny         2.051897
xqwwgdyp         1.816003
kldqjyjy         1.756094
uqqtjvyb         1.692440
tfqavkke         1.452803
ukymxvdu         1.392893
vlluhbov         1.325495
oijqvulv         1.288052
```

```
ccgxvspp        1.276819
bxpfxfdn        1.239375
haliazsg        1.108324
rcertsgn        1.033437
xzmlyyjv        0.928595
dlvbwzss        0.849964
hodpvpew        0.778822
dcjcmpih        0.554162
pvmttkik        0.366945
Name: employment_occupation, dtype: float64
```

- Since the remaining missing values are categorical variables, we will fill using `Unknown` for the `education`, `marital_status`, `rent_or_own`, and `income_poverty` columns.
- This gives a clear label for the missing values and allows the values to be treated as a distinct category during data analysis and modeling.

```
# filling with unkwown for categorical isna values

# education
train["education"].fillna("Unknown", inplace=True)

# marital status
train["marital_status"].fillna("Unknown", inplace=True)

# rent
train["rent_or_own"].fillna("Unknown", inplace=True)

# income/poverty
train["income_poverty"].fillna("Unknown", inplace=True)

# fill all the categorical variable with the modal class
train_filled = train.fillna(train.mode().iloc[0])
missing2=train_filled.isnull().sum() # confirm if the data has been filled.

missing2

respondent_id                  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_seasonal           0
chronic_med_condition          0
child_under_6_months           0
```

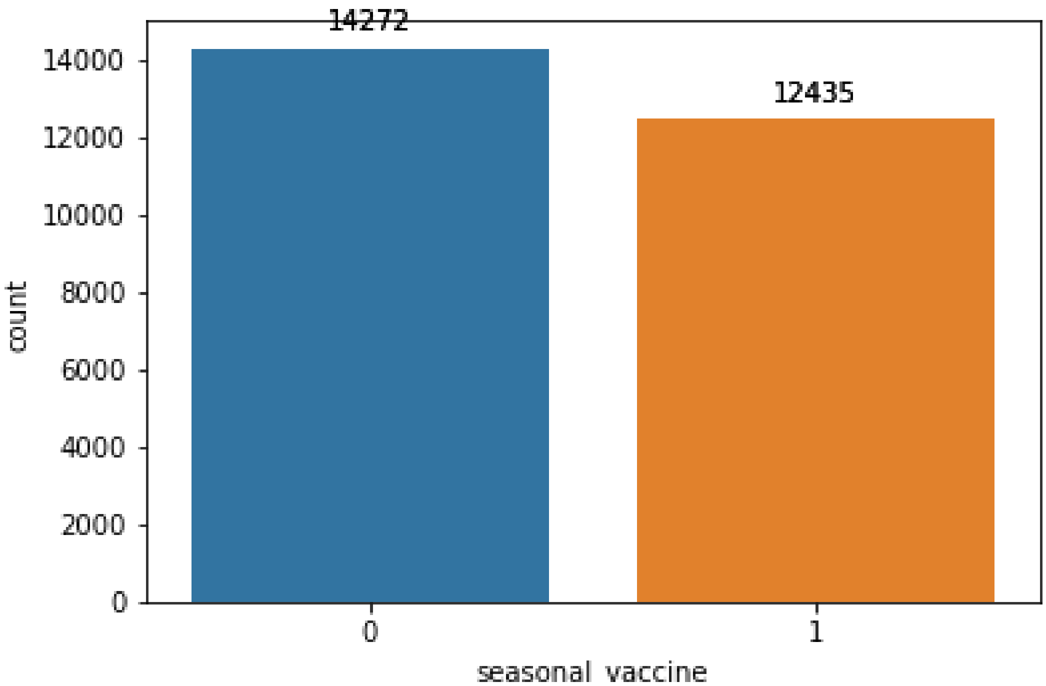
```
health_worker                  0
health_insurance               0
opinion_seas_vacc_effective    0
opinion_seas_risk              0
opinion_seas_sick_from_vacc    0
age_group                      0
education                      0
race                           0
sex                            0
income_poverty                 0
marital_status                 0
rent_or_own                    0
employment_status              0
hhs_geo_region                 0
census_msa                     0
household_adults               0
household_children             0
employment_industry            0
employment_occupation          0
dtype: int64
```

Exploratory Data Analysis (EDA)

```
# Target variables dataset exploratory

target.head()
```

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

```
# checking the distribution of the target variable

sns.countplot(data=target, x="seasonal_vaccine")

# plt.savefig("images/seasonal_vaccine_count_plot.png")

# Add data labels

ax = sns.countplot(data=target, x="seasonal_vaccine")
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center',
                xytext=(0, 10), textcoords='offset points')
```



Observations

- The number of respondents who had taken the seasonal flu vaccine (1) is lower compared to those who had not taken it (0).
- The counts of the two classes are not significantly different and are relatively close, we can consider dataset to be reasonably balanced.

More Preparation for Analysis

- First, a new DataFrame, `train_target`, is created from a merge of `train_filled` and `target` DataFrames to create a single DataFrame.
- The new DataFrame will contain all the information needed for univariate and multivariate analysis and for building machine learning models.

```
#Merge the two datasets

train_target = train_filled.merge(target, on="respondent_id")

#make respondent id the index column

train_target.set_index("respondent_id", inplace=True)
train_target
```

```
               behavioral_antiviral_meds  behavioral_avoidance  \
respondent_id
0                                    0.0                   0.0
1                                    0.0                   1.0
2                                    0.0                   1.0
3                                    0.0                   1.0
```

```
4                                   0.0                   1.0
...                                 ...                   ...
26702                               0.0                   1.0
26703                               0.0                   1.0
26704                               0.0                   1.0
26705                               0.0                   0.0
26706                               0.0                   1.0

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

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

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

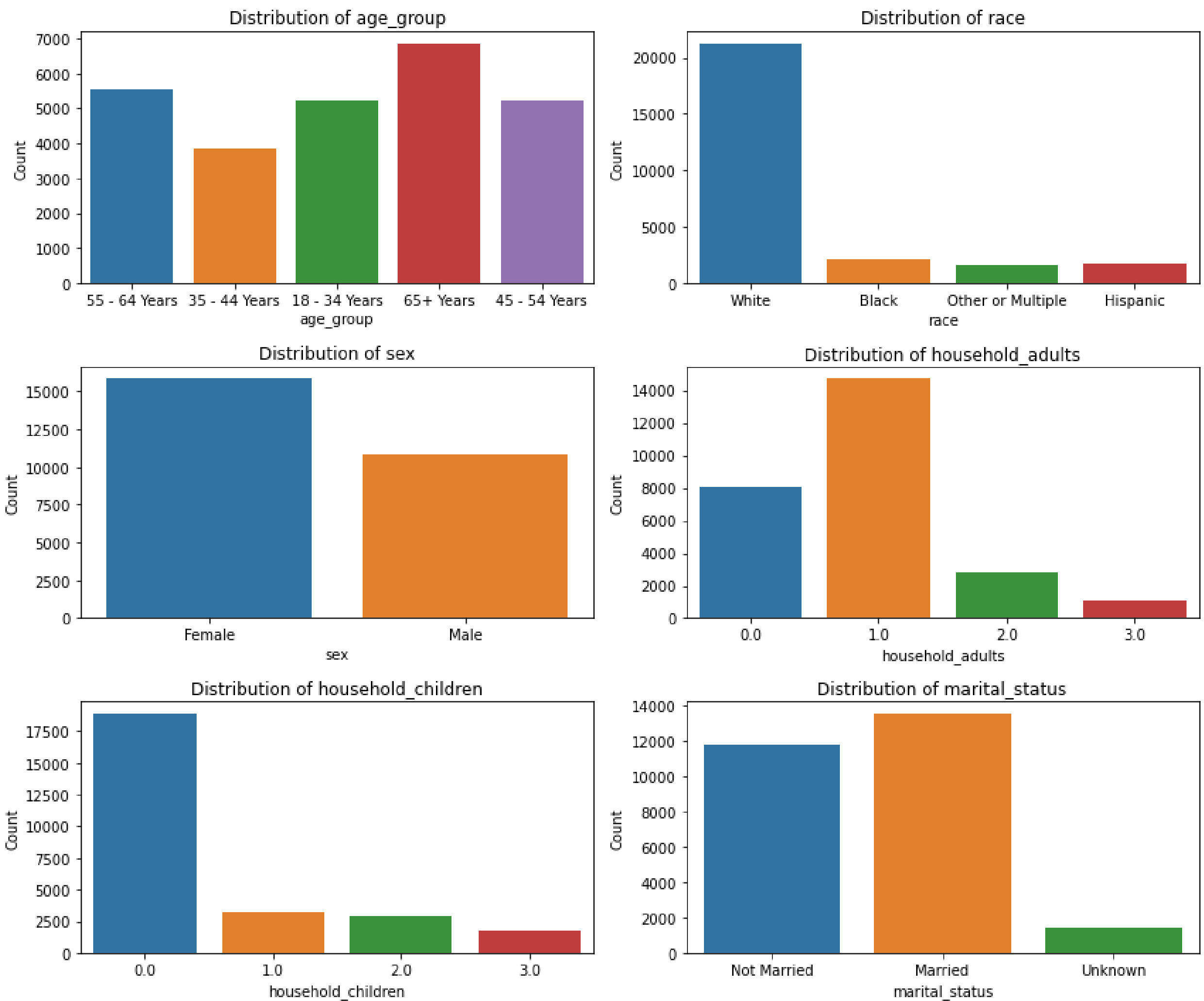
```
                  chronic_med_condition  child_under_6_months  ...  \
respondent_id                                                  ...
0                                   0.0                   0.0  ...
1                                   0.0                   0.0  ...
2                                   1.0                   0.0  ...
3                                   1.0                   0.0  ...
4                                   0.0                   0.0  ...
...                                 ...                   ...  ...
26702                               0.0                   0.0  ...
26703                               0.0                   0.0  ...
26704                               0.0                   0.0  ...
26705                               0.0                   0.0  ...
26706                               0.0                   0.0  ...

                  marital_status  rent_or_own   employment_status  \
respondent_id
0                    Not Married          Own  Not in Labor Force
1                    Not Married         Rent            Employed
2                    Not Married          Own            Employed
3                    Not Married         Rent  Not in Labor Force
4                        Married          Own            Employed
...                          ...          ...                 ...
26702                Not Married          Own  Not in Labor Force
26703                Not Married         Rent            Employed
26704                Not Married          Own            Employed
26705                    Married         Rent            Employed
26706                    Married          Own  Not in Labor Force

                  hhs_geo_region                census_msa
household_adults  \
respondent_id

0                       oxchjgsf                   Non-MSA
0.0
1                       bhuqouqj  MSA, Not Principle  City
0.0
2                       qufhixun  MSA, Not Principle  City
2.0
3                       lrircsnp       MSA, Principle City
0.0
4                       qufhixun  MSA, Not Principle  City
1.0
...                          ...                       ...              .
..
26702                   qufhixun                   Non-MSA
0.0
26703                   lzgpxyit       MSA, Principle City
1.0
26704                   lzgpxyit  MSA, Not Principle  City
0.0
```

```
26705                lrircsnp                   Non-MSA
1.0
26706                mlyzmhmf       MSA, Principle City
1.0

              household_children employment_industry
employment_occupation  \
respondent_id

0                            0.0       not employed          not
employed
1                            0.0           pxcmvdjn
xgwztkwe
2                            0.0           rucpziij
xtkaffoo
3                            0.0       not employed          not
employed
4                            0.0           wxleyezf
emcorrxb
...                          ...                ...
...
26702                        0.0       not employed          not
employed
26703                        0.0           fcxhlnwr
cmhcxjea
26704                        0.0       not employed          not
employed
26705                        0.0           fcxhlnwr
haliazsg
26706                        0.0       not employed          not
employed

              seasonal_vaccine
respondent_id
0                            0
1                            1
2                            0
3                            1
4                            0
...                        ...
26702                        0
26703                        0
26704                        1
26705                        0
26706                        0

[26707 rows x 30 columns]
```

Univariate Data Analysis

1. EDA for Demographic Features

```
# List of demographic columns to create countplots for
demographic_columns = ["age_group", "race", "sex", "household_adults", "household_children", "marital_status"]

# Calculate the number of rows and columns for subplots dynamically
num_plots = len(demographic_columns)
num_cols = 2
num_rows = (num_plots + num_cols - 1) // num_cols

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 10))

# Flatten the axes array for easier indexing
axes = axes.flatten()

# Loop through the columns and create individual countplots
for i, column in enumerate(demographic_columns):
    if i >= num_rows * num_cols:  # Check if index exceeds total number of subplots
        fig.delaxes(axes[i])
    else:
        sns.countplot(data=train_target, x=column, ax=axes[i])
        axes[i].set_xlabel(column)
        axes[i].set_ylabel("Count")
        axes[i].set_title(f"Distribution of {column}")

# Adjust the layout and spacing between subplots
plt.tight_layout()


# Show the subplots
plt.show()
```



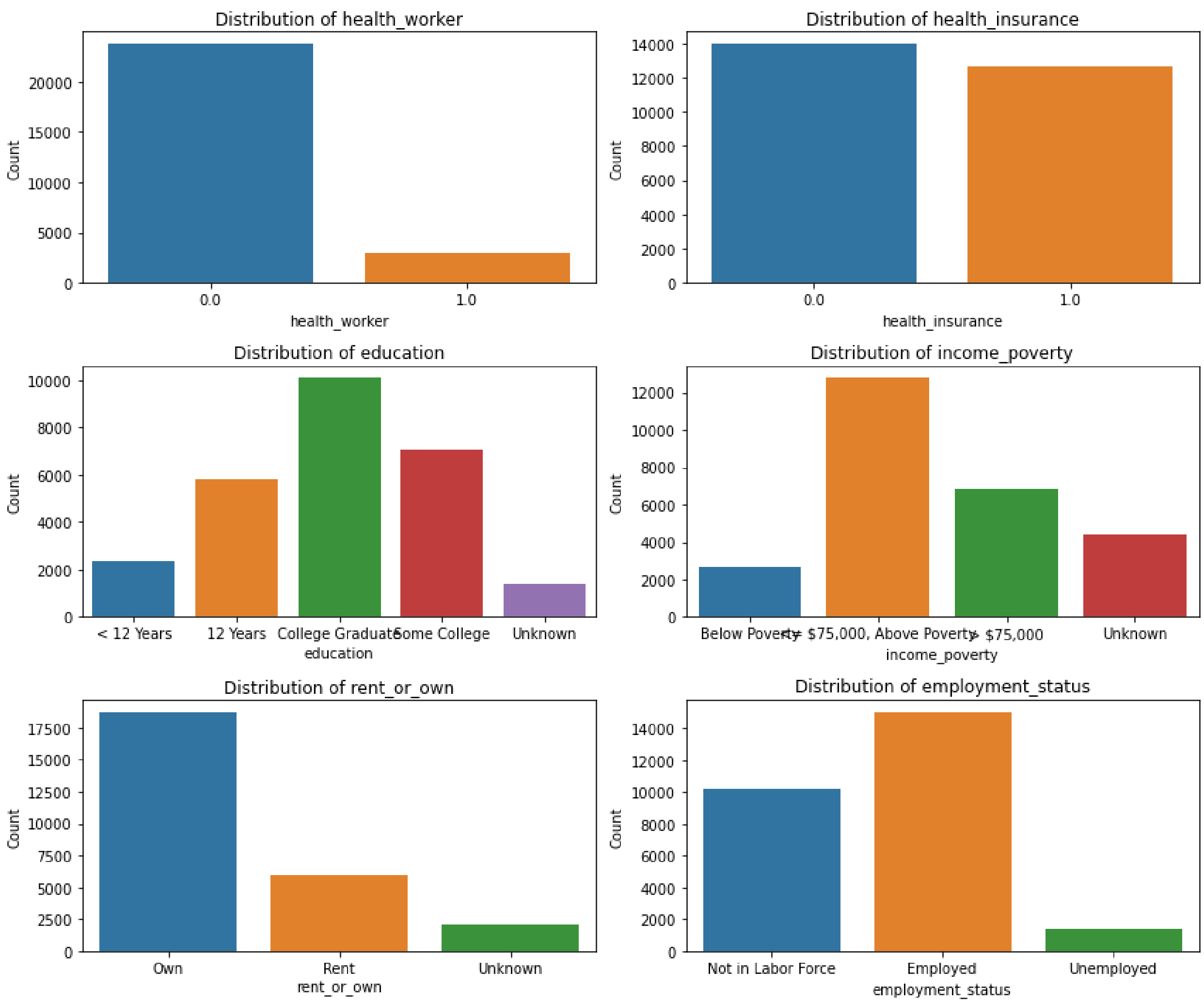
Observations on Demographics

- The respondents" age groups were normally distributed with majority being 65 years and older. This is synonymous with developed countries. Most were also female and white.
- Majority of the households had at least two adults with no child(ren) at the time of the interview.
- The number of married and unmarried respondents was almost similar.
- Majority also lived outside the Metropolitan Statitistical Area (MSA) as defined by the US Census.

2. EDA Socioeconomic Features

```python
# List of columns to create countplots for
columns_to_plot = ["health_worker", "health_insurance", "education",
                   "income_poverty", "rent_or_own",
"employment_status"]

# Calculate the number of rows and columns for subplots dynamically
num_plots = len(columns_to_plot)
```

```
num_cols = 2
num_rows = (num_plots + num_cols - 1) // num_cols

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 10))

# Flatten the axes array for easier indexing
axes = axes.flatten()

# Loop through the columns and create individual countplots
for i, column in enumerate(columns_to_plot):
    if i >= num_rows * num_cols:  # Check if index exceeds total 
number of subplots
        fig.delaxes(axes[i])
    else:
        if column == "age_group":
            sns.countplot(data=train_target, x=column, 
order=train_target["age_group"].value_counts().index, ax=axes[i])
        else:
            sns.countplot(data=train_target, x=column, ax=axes[i])
        axes[i].set_xlabel(column)
        axes[i].set_ylabel("Count")
        axes[i].set_title(f"Distribution of {column}")

# Adjust the layout and spacing between subplots
plt.tight_layout()


# Show the subplots
plt.show()
```

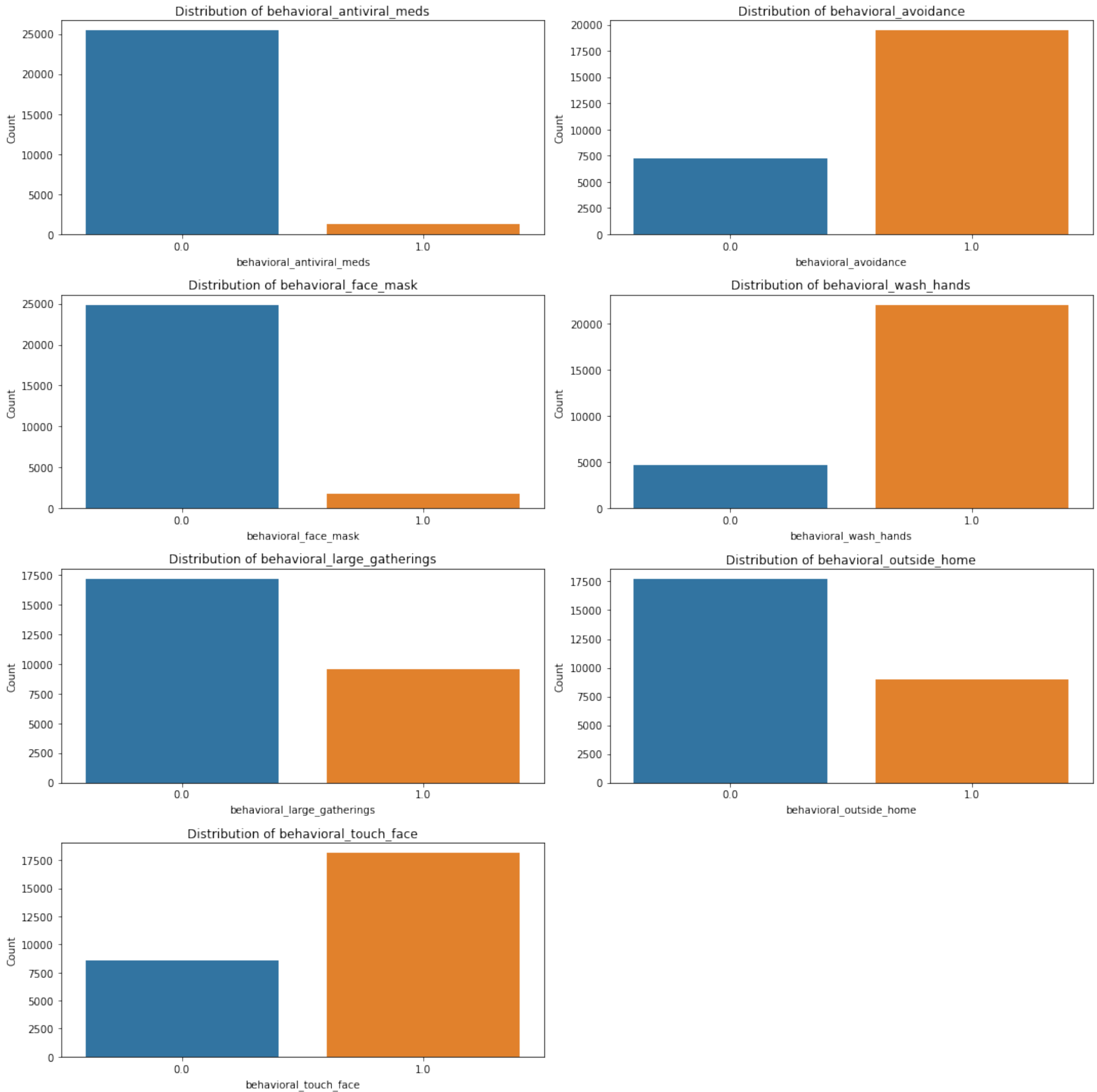


Observations

- The respondents were mostly made up of individuals in other professions other than health workers.
- The number of people with health insurance was almost similar to those who had no health insurance. However, those without were the majority. This is partly due to the assumption made earlier that the missing values were for those without insurance.
- A normal distribution in the education feature was observed with those who had a college education making up the majority.
- Most respondents had an annual household income of less than $75,000. A fair amount of respondents failed to disclose their income and were categorized as "Unknown".
- Majority of the respondents own the houses they live in.
- Majority of the respondendts were employed at the time of the survey compared to those not in labour force or unemployed

3. EDA for Behavioral Features

```
#EDA for behavioral factors
# List of columns to create countplots for
columns_to_plot = ["behavioral_antiviral_meds",
"behavioral_avoidance", "behavioral_face_mask",
"behavioral_wash_hands",
                   "behavioral_large_gatherings",
"behavioral_outside_home", "behavioral_touch_face"]

# Define the number of rows and columns for subplots
num_rows = 4
num_cols = 2

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 15))

# Flatten the axes array for easier indexing
axes = axes.flatten()

# Loop through the columns and create individual countplots
for i, column in enumerate(columns_to_plot):
    sns.countplot(data=train_target, x=column, ax=axes[i])
    axes[i].set_xlabel(column)
    axes[i].set_ylabel("Count")
    axes[i].set_title(f"Distribution of {column}")

# Adjust the layout and spacing between subplots
fig.delaxes(axes[-1])

# Adjust the layout and spacing between subplots
plt.tight_layout()


# Show the subplots
plt.show()
```

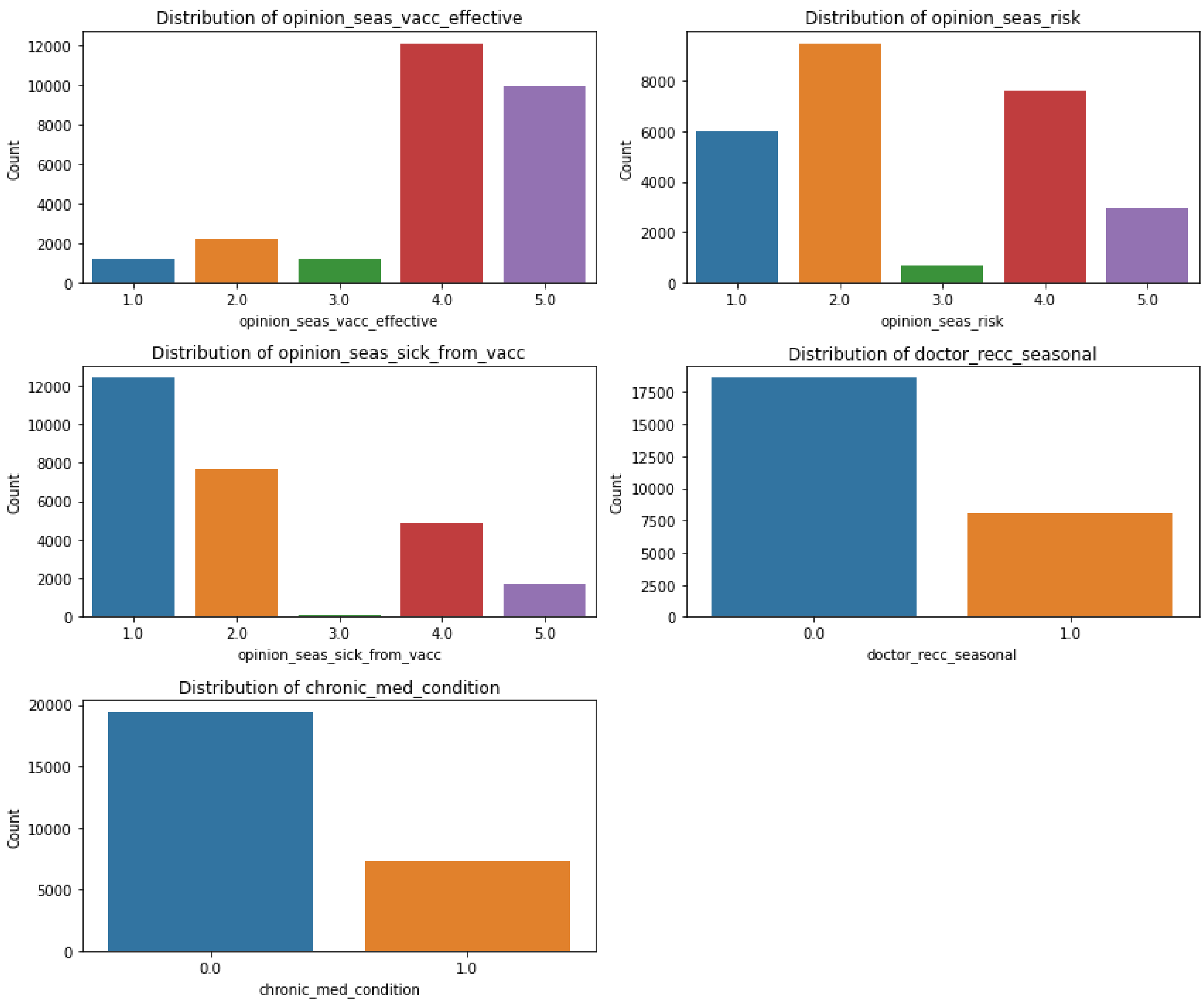


Behavioural features Observations

- Majority of the respondents appeared not to have taken any antiviral medication at the time of the survey. The response might be subjective and would require further investigation on how the question was asked.
- Most of the respondents had also avoided close contact with people who had flu-like symptoms.Most were also in the practice of washing hands and using hand sanitizer.
- Majority of the respondents were not using facemask. Similarly, most were avoiding touching their face, nose or mouth.
- There was no social-distancing among the respondents while most of the respondents spend more time outside their homes as opposed to staying indoors

4. EDA for Knowledge, Attitudes and Beliefs Towards the Vaccines

```
# List of columns to create countplots for
columns_to_plot = ["opinion_seas_vacc_effective", "opinion_seas_risk",
"opinion_seas_sick_from_vacc",
                   "doctor_recc_seasonal","chronic_med_condition"]

# Define the number of rows and columns for subplots
num_rows = 3
num_cols = 2

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 10))

# Flatten the axes array for easier indexing
axes = axes.flatten()

# Loop through the columns and create individual countplots
for i, column in enumerate(columns_to_plot):
    sns.countplot(data=train_target, x=column, ax=axes[i])
    axes[i].set_xlabel(column)
    axes[i].set_ylabel("Count")
    axes[i].set_title(f"Distribution of {column}")

# Adjust the layout and spacing between subplots
fig.delaxes(axes[-1])


# Adjust the layout and spacing between subplots
plt.tight_layout()

# Show the subplots
plt.show()
```



Observations or Knowledge, Attitudes and Beliefs Towards the Vaccines

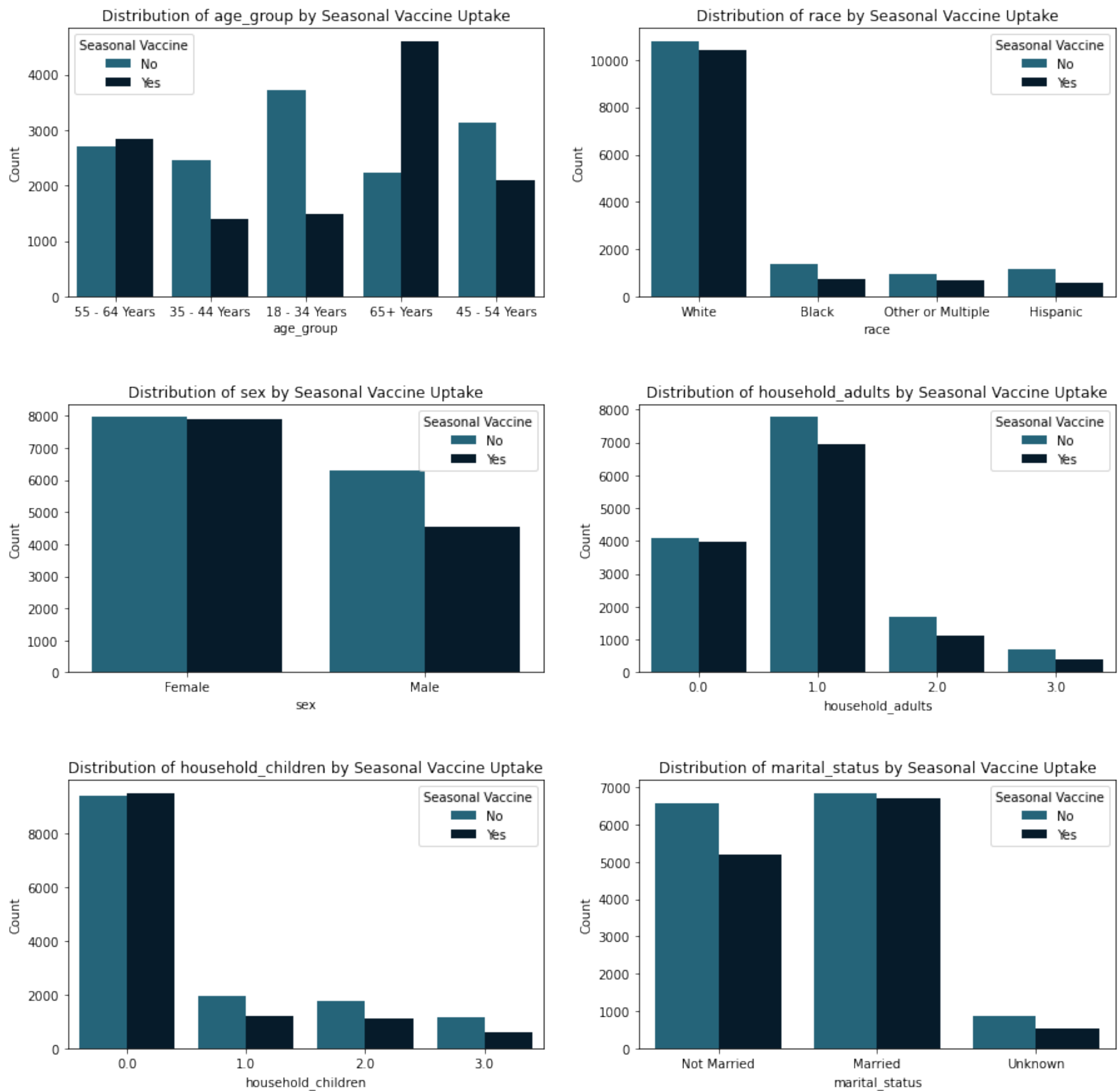
- Majority believed that seasonal vaccines were effective and they were not worried about falling sick from taking the vaccine.
- However, a large number believed that the risk of contracting the flu without a vaccine was low.
- Majority of the respondents had not received any recommendations for the vaccine from their doctors.
- Most respondents did not have a chronic medical condition.

Bivariate Analysis

1. Demographic Features vs Seasonal Vaccine Uptake

```
# columns to plot
y = ["age_group", "race", "sex", "household_adults",
"household_children", "marital_status"]

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 15))
```

```
custom_palette = ["#176B87", "#001C30"]

for i, variable in enumerate(y):
    row = i // 2
    col = i % 2
    ax = axes[row, col]
    sns.countplot(x=variable, hue="seasonal_vaccine",
data=train_target, ax=ax,palette=custom_palette)
    ax.set_title(f"Distribution of {variable} by Seasonal Vaccine
Uptake")
    ax.set_xlabel(variable)
    ax.set_ylabel("Count")
    ax.legend(title="Seasonal Vaccine", labels=["No", "Yes"])

# Adjust spacing between subplots
plt.subplots_adjust(hspace=0.4)

plt.show()
```



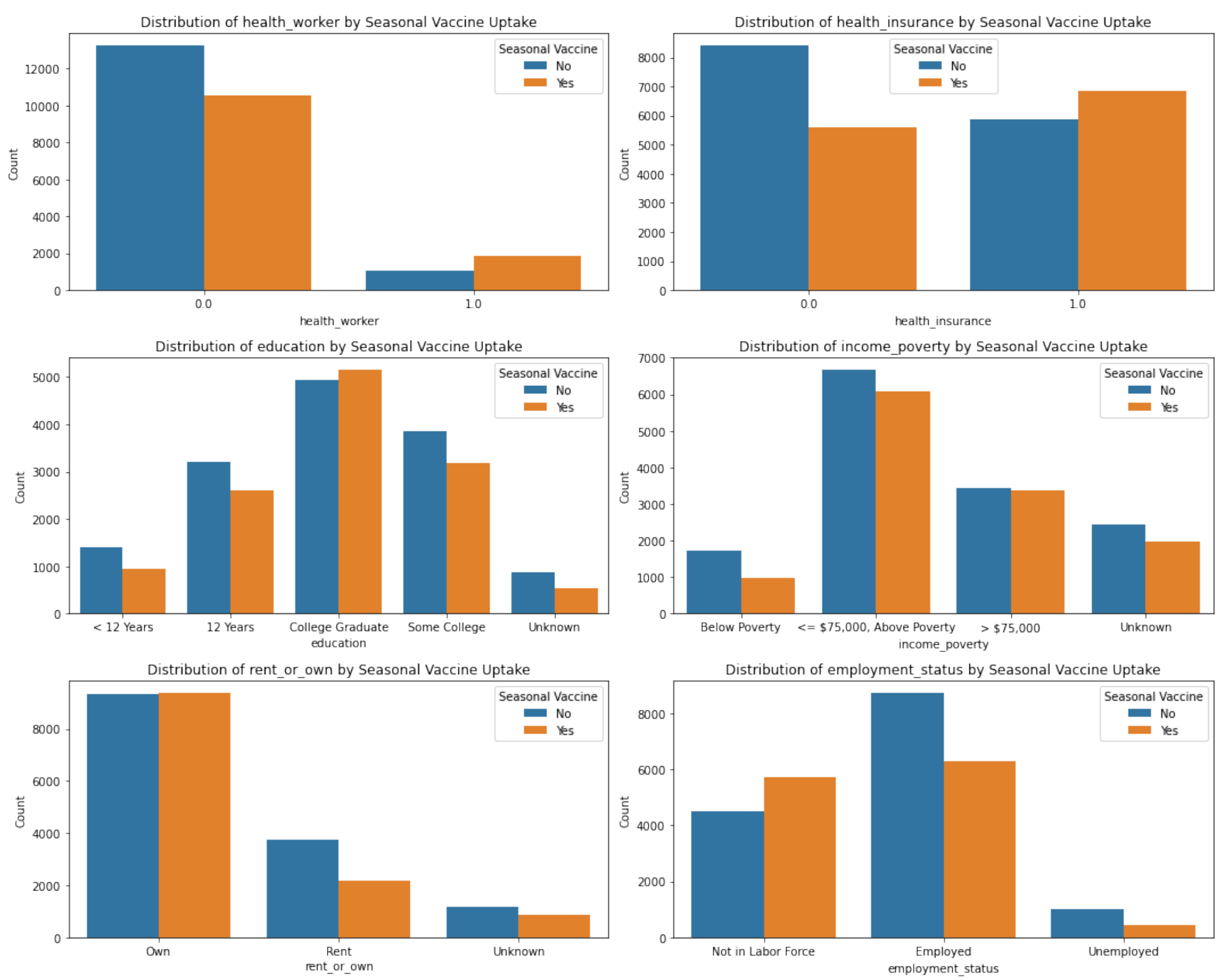
Demographic Features vs Seasonal Vaccine Uptake Observations

- The number of older people above 65 years of age were more likely to receive the vaccine compared to the younger population.
- More female and married respondents as well as people of White descent received the vaccine compared to others in their respective categories.
- Households with one adult and no children received the vaccine in more numbers than other respondents in the respective categories.

2. Socieconomic Features vs Seasonal Vaccine Uptake

```
# columns to plot
y = ["health_worker", "health_insurance", "education",
     "income_poverty", "rent_or_own", "employment_status"]
```

```python
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 12))

for i, variable in enumerate(y):
    row = i // 2
    col = i % 2
    ax = axes[row, col]
    sns.countplot(x=variable, hue="seasonal_vaccine",
data=train_target, ax=ax)
    ax.set_title(f"Distribution of {variable} by Seasonal Vaccine
Uptake")
    ax.set_xlabel(variable)
    ax.set_ylabel("Count")
    ax.legend(title="Seasonal Vaccine", labels=["No", "Yes"])

plt.tight_layout()


# Show the subplots
plt.show()
```

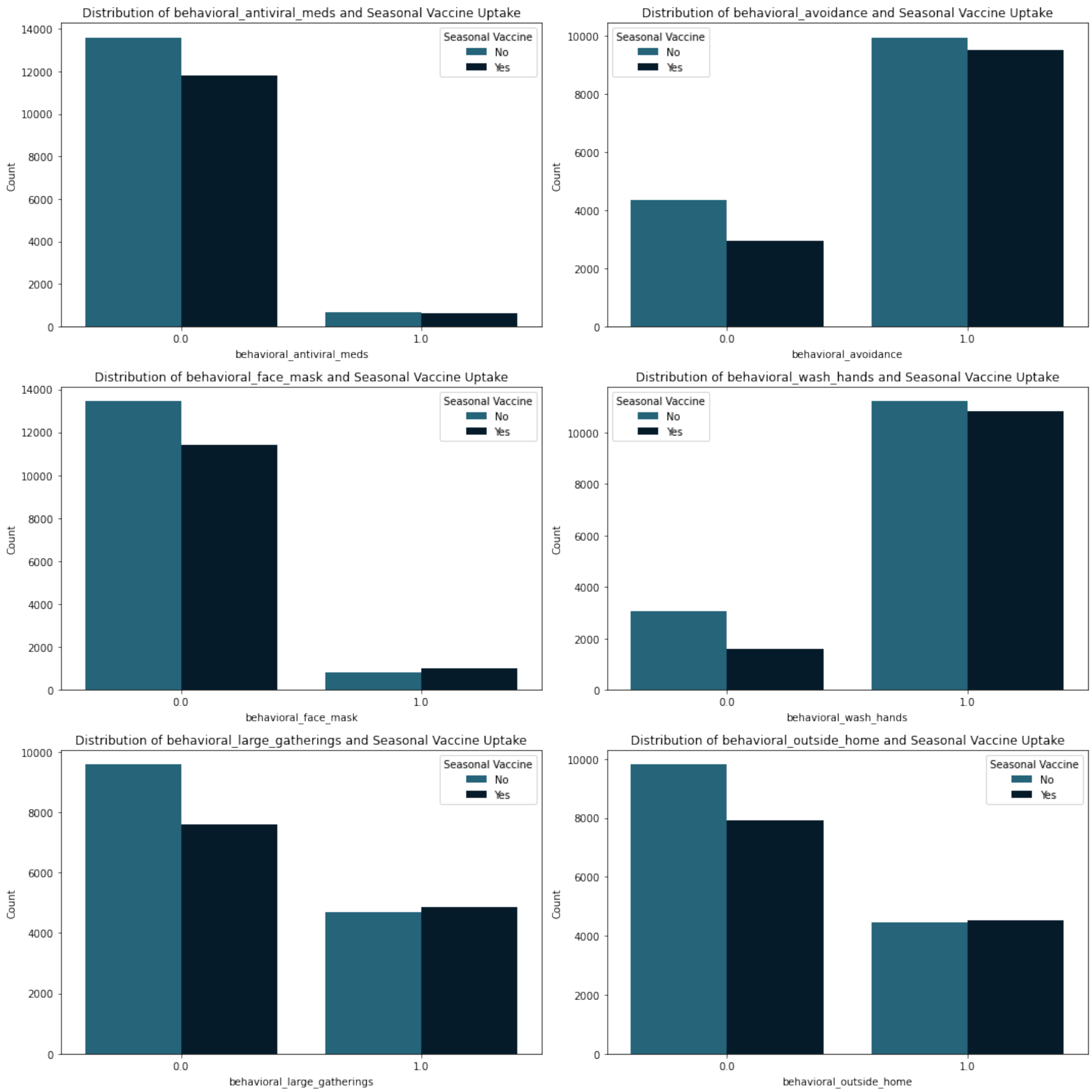


Observations for Socieconomic Features vs Seasonal Vaccine Uptake

- Majority of the health workers received vaccines as opposed to non-health workers where majority did not receive. This is highly attributed to the fact that health workers are vulnerable and are a more at risk population highly predisposed to flu
- Respondents with health insurance are likely to receive vaccines as compared to those without the insurance due to the ease of access and affordability facilitated by the insurance cover.
- Respondents with college level of education were see to be more receptive of the vaccine which could be attributed to access to information and facts about the vaccines.
- The poorer respondents and those with an annual income of <=$75,000 per household were less likely to receive the vaccine which could be attributed to access and affordability as well as lack of health insurance.
- More of those who lived in their own houses received the vaccine compared to those paying rent/with unknown housing conditions.
- The employed respondents were more likely to receive the vaccine compared to those unemployed

3. Behavioral Features vs Seasonal Vaccine Uptake

```
# columns to plot
y = ["behavioral_antiviral_meds", "behavioral_avoidance",
"behavioral_face_mask",
     "behavioral_wash_hands", "behavioral_large_gatherings",
"behavioral_outside_home"]

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 15))

# Define custom color palette
#custom_palette = ["#454545", "#FF6000"]

for i, variable in enumerate(y):
    row = i // 2
    col = i % 2
    ax = axes[row, col]
    sns.countplot(x=variable, hue="seasonal_vaccine",
data=train_target, ax=ax, palette=custom_palette)
    ax.set_title(f"Distribution of {variable} and Seasonal Vaccine
Uptake")
    ax.set_xlabel(variable)
    ax.set_ylabel("Count")
    ax.legend(title="Seasonal Vaccine", labels=["No", "Yes"])

plt.tight_layout()

plt.show()
```



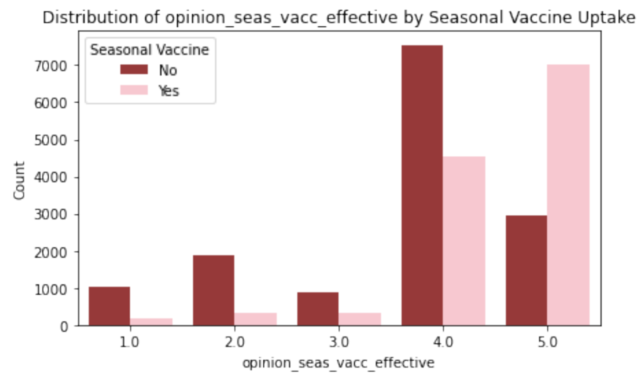
Behavioral Features vs Seasonal Vaccine Uptake Observations

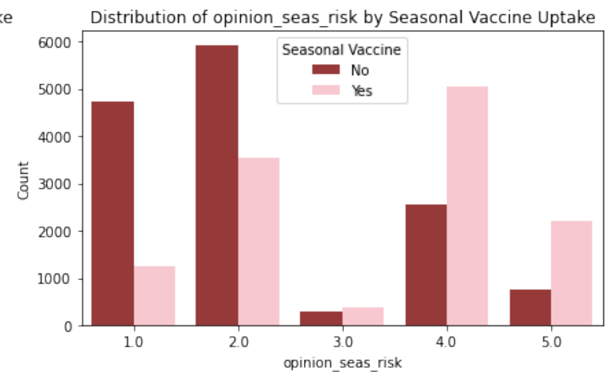
- Generally, behavioral factors did not have much influence on whether the respondents took the vaccines.
- Whether the respondent was on antiviral medication, avoided contact with people showing flu symptoms even outside the home or washed hands/used santizers often and vice versa, the general outome was that less people ad received the vaccine.

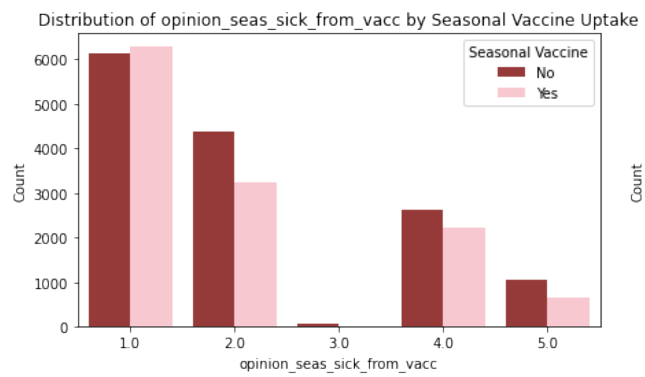
4. Knowledge, Attitudes and Beliefs Towards Vaccines vs Seasonal Vaccine Uptake

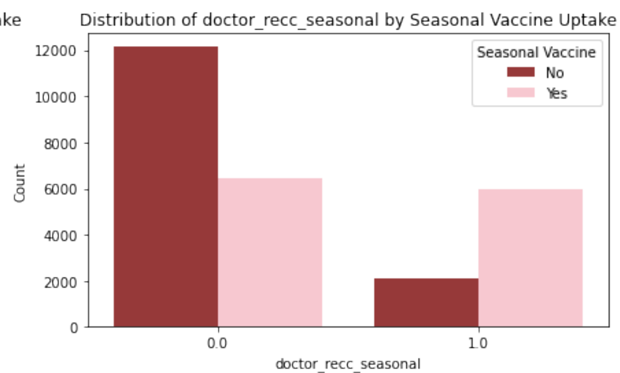
```
# columns to plot
y = ["opinion_seas_vacc_effective", "opinion_seas_risk",
"opinion_seas_sick_from_vacc",
```

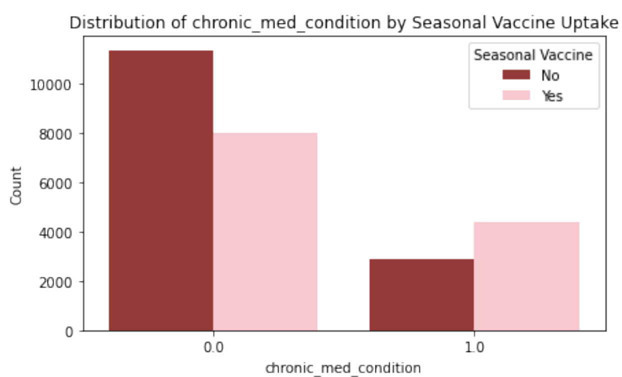
```
     "doctor_recc_seasonal", "chronic_med_condition"]

num_plots = len(y)
num_cols = 2
num_rows = (num_plots + num_cols - 1) // num_cols

fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(15,
15))
custom_palette = ["brown", "pink"]

for i, variable in enumerate(y):
    if i >= num_plots:
        break

    row = i // num_cols
    col = i % num_cols

    ax = axes[row, col]
    sns.countplot(x=variable, hue="seasonal_vaccine",
data=train_target, ax=ax, palette=custom_palette)
    ax.set_title(f"Distribution of {variable} by Seasonal Vaccine
Uptake")
    ax.set_xlabel(variable)
    ax.set_ylabel("Count")
    ax.legend(title="Seasonal Vaccine", labels=["No", "Yes"])

# Adjust spacing between subplots
plt.subplots_adjust(hspace=0.4)

# Remove any extra blank subplot
if num_plots < num_cols * num_rows:
    fig.delaxes(axes.flatten()[num_plots])

plt.show()
```











Knowledge, Attitudes and Beliefs Towards Vaccines vs Seasonal Vaccine Uptake Observations

- The majority of respondents who received the vaccine hold the opinion that it is effective.
- A general observation is that the lower the opinion towards the vaccine factor, the lower the vaccine uptake. A respondent was less likely to receive the vaccine if they:
  - did not believe that there is a risk of getting sick with seasonal flu without vaccine;
  - was not worried of getting sick from taking seasonal flu vaccine;
  - did not get a doctor's recommendation to take the vaccine, and;
  - had no chronic medical condition.

Feature Engineering

Data Encoding

First, the data types are displayed once again to determine the affected features.

```
train_target.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26707 entries, 0 to 26706
Data columns (total 30 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   behavioral_antiviral_meds    26707 non-null  float64
 1   behavioral_avoidance         26707 non-null  float64
 2   behavioral_face_mask         26707 non-null  float64
 3   behavioral_wash_hands        26707 non-null  float64
 4   behavioral_large_gatherings  26707 non-null  float64
 5   behavioral_outside_home      26707 non-null  float64
 6   behavioral_touch_face        26707 non-null  float64
 7   doctor_recc_seasonal         26707 non-null  float64
 8   chronic_med_condition        26707 non-null  float64
 9   child_under_6_months         26707 non-null  float64
 10  health_worker                26707 non-null  float64
 11  health_insurance             26707 non-null  float64
 12  opinion_seas_vacc_effective  26707 non-null  float64
 13  opinion_seas_risk            26707 non-null  float64
 14  opinion_seas_sick_from_vacc  26707 non-null  float64
 15  age_group                    26707 non-null  object
 16  education                    26707 non-null  object
 17  race                         26707 non-null  object
 18  sex                          26707 non-null  object
 19  income_poverty               26707 non-null  object
 20  marital_status               26707 non-null  object
 21  rent_or_own                  26707 non-null  object
 22  employment_status            26707 non-null  object
 23  hhs_geo_region               26707 non-null  object
 24  census_msa                   26707 non-null  object
 25  household_adults             26707 non-null  float64
 26  household_children           26707 non-null  float64
 27  employment_industry          26707 non-null  object
 28  employment_occupation        26707 non-null  object
 29  seasonal_vaccine             26707 non-null  int64
dtypes: float64(17), int64(1), object(12)
memory usage: 7.6+ MB
```

```
train_target.head()
```

```
               behavioral_antiviral_meds  behavioral_avoidance  \
respondent_id
```

```
0                                    0.0                     0.0
1                                    0.0                     1.0
2                                    0.0                     1.0
3                                    0.0                     1.0
4                                    0.0                     1.0

               behavioral_face_mask  behavioral_wash_hands  \
respondent_id
0                               0.0                    0.0
1                               0.0                    1.0
2                               0.0                    0.0
3                               0.0                    1.0
4                               0.0                    1.0

               behavioral_large_gatherings  behavioral_outside_home  \
respondent_id
0                                      0.0                      1.0
1                                      0.0                      1.0
2                                      0.0                      0.0
3                                      1.0                      0.0
4                                      1.0                      0.0

               behavioral_touch_face  doctor_recc_seasonal  \
respondent_id
0                                1.0                   0.0
1                                1.0                   0.0
2                                0.0                   0.0
3                                0.0                   1.0
4                                1.0                   0.0

               chronic_med_condition  child_under_6_months  ...  \
respondent_id                                               ...
0                                0.0                   0.0  ...
1                                0.0                   0.0  ...
2                                1.0                   0.0  ...
3                                1.0                   0.0  ...
4                                0.0                   0.0  ...

               marital_status  rent_or_own   employment_status  \
respondent_id
0                 Not Married          Own  Not in Labor Force
1                 Not Married         Rent            Employed
2                 Not Married          Own            Employed
3                 Not Married         Rent  Not in Labor Force
4                     Married          Own            Employed

               hhs_geo_region                census_msa
household_adults  \
respondent_id
```

```
0                     oxchjgsf                   Non-MSA
0.0
1                     bhuqouqj  MSA, Not Principle  City
0.0
2                     qufhixun  MSA, Not Principle  City
2.0
3                     lrircsnp       MSA, Principle City
0.0
4                     qufhixun  MSA, Not Principle  City
1.0

              household_children employment_industry
employment_occupation  \
respondent_id

0                            0.0        not employed         not
employed
1                            0.0            pxcmvdjn
xgwztkwe
2                            0.0            rucpziij
xtkaffoo
3                            0.0        not employed         not
employed
4                            0.0            wxleyezf
emcorrxb

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

[5 rows x 30 columns]
```

One-Hot Encoding

Columns - "age_group", "education", "race", "sex", "marital_status", "rent_or_own", "employment_status", "census_msa", and "income_poverty."

```
from sklearn.preprocessing import OneHotEncoder
encoded_df = train_target[["age_group", "education", "race", "sex",
"marital_status", "rent_or_own", "employment_status",
                           "census_msa", "income_poverty"]]
#using one-hot encoding to create dummy column
ohe = OneHotEncoder(handle_unknown="ignore", sparse=False)
data_enc1= ohe.fit_transform(encoded_df)

#converting the finding into dataframe
```

```
# data_enc1.todense()

#getting feature names
feature_names = ohe.get_feature_names(encoded_df.columns)

# geting feature names in a dataframe
data_encoded = pd.DataFrame(data_enc1, columns=feature_names)
data_encoded.head()

   age_group_18 - 34 Years  age_group_35 - 44 Years  age_group_45 - 54 
Years  \
0                      0.0                      0.0                      
0.0   
1                      0.0                      1.0                      
0.0   
2                      1.0                      0.0                      
0.0   
3                      0.0                      0.0                      
0.0   
4                      0.0                      0.0                      
1.0   

   age_group_55 - 64 Years  age_group_65+ Years  education_12 Years  \
0                      1.0                  0.0                 0.0   
1                      0.0                  0.0                 1.0   
2                      0.0                  0.0                 0.0   
3                      0.0                  1.0                 1.0   
4                      0.0                  0.0                 0.0   

   education_< 12 Years  education_College Graduate  education_Some 
College  \
0                   1.0                         0.0                   
0.0   
1                   0.0                         0.0                   
0.0   
2                   0.0                         1.0                   
0.0   
3                   0.0                         0.0                   
0.0   
4                   0.0                         0.0                   
1.0   

   education_Unknown  ...  employment_status_Employed  \
0                0.0  ...                         0.0   
1                0.0  ...                         1.0   
2                0.0  ...                         1.0   
3                0.0  ...                         0.0   
4                0.0  ...                         1.0   

   employment_status_Not in Labor Force  employment_status_Unemployed
```

```
\
0                                    1.0                            0.0

1                                    0.0                            0.0

2                                    0.0                            0.0

3                                    1.0                            0.0

4                                    0.0                            0.0


   census_msa_MSA, Not Principle  City  census_msa_MSA, Principle City
\
0                                   0.0                             0.0

1                                   1.0                             0.0

2                                   1.0                             0.0

3                                   0.0                             1.0

4                                   1.0                             0.0


   census_msa_Non-MSA  income_poverty_<= $75,000, Above Poverty  \
0                 1.0                                       0.0
1                 0.0                                       0.0
2                 0.0                                       1.0
3                 0.0                                       0.0
4                 0.0                                       1.0

   income_poverty_> $75,000  income_poverty_Below Poverty  \
0                       0.0                           1.0
1                       0.0                           1.0
2                       0.0                           0.0
3                       0.0                           1.0
4                       0.0                           0.0

   income_poverty_Unknown
0                     0.0
1                     0.0
2                     0.0
3                     0.0
4                     0.0

[5 rows x 32 columns]
```

- A copy of the DataFrame was made to avoid overwritng the main dataset during data manipulation.

- The merged train_target_float DataFrame has a combination of numerical columns from train_target_copy and the one-hot encoded binary columns from data_encoded.
- This merged DataFrame has all the features represented in numerical format.

```
train_target_copy = train_target.copy()
train_target_copy.head()
```

```
               behavioral_antiviral_meds  behavioral_avoidance  \
respondent_id
0                                    0.0                   0.0
1                                    0.0                   1.0
2                                    0.0                   1.0
3                                    0.0                   1.0
4                                    0.0                   1.0

               behavioral_face_mask  behavioral_wash_hands  \
respondent_id
0                               0.0                    0.0
1                               0.0                    1.0
2                               0.0                    0.0
3                               0.0                    1.0
4                               0.0                    1.0

               behavioral_large_gatherings  behavioral_outside_home  \
respondent_id
0                                      0.0                      1.0
1                                      0.0                      1.0
2                                      0.0                      0.0
3                                      1.0                      0.0
4                                      1.0                      0.0

               behavioral_touch_face  doctor_recc_seasonal  \
respondent_id
0                                1.0                   0.0
1                                1.0                   0.0
2                                0.0                   0.0
3                                0.0                   1.0
4                                1.0                   0.0

               chronic_med_condition  child_under_6_months  ...  \
respondent_id                                               ...
0                                0.0                   0.0  ...
1                                0.0                   0.0  ...
2                                1.0                   0.0  ...
3                                1.0                   0.0  ...
4                                0.0                   0.0  ...

               marital_status  rent_or_own   employment_status  \
respondent_id
0                 Not Married          Own  Not in Labor Force
1                 Not Married         Rent            Employed
```

```
2                   Not Married          Own             Employed
3                   Not Married         Rent  Not in Labor Force
4                       Married          Own             Employed

               hhs_geo_region                census_msa
household_adults  \
respondent_id

0                    oxchjgsf                   Non-MSA
0.0
1                    bhuqouqj  MSA, Not Principle  City
0.0
2                    qufhixun  MSA, Not Principle  City
2.0
3                    lrircsnp       MSA, Principle City
0.0
4                    qufhixun  MSA, Not Principle  City
1.0

              household_children employment_industry
employment_occupation  \
respondent_id

0                            0.0        not employed          not
employed
1                            0.0            pxcmvdjn
xgwztkwe
2                            0.0            rucpziij
xtkaffoo
3                            0.0        not employed          not
employed
4                            0.0            wxleyezf
emcorrxb

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

[5 rows x 30 columns]
```

```python
columns_to_drop = ["age_group", "education", "race", "sex",
"marital_status", "rent_or_own", "employment_status",
                   "census_msa", "income_poverty", "hhs_geo_region",
"employment_industry", "employment_occupation"]
train_target_copy.drop(columns_to_drop, axis=1, inplace=True)
train_target_copy.head()
```

```
               behavioral_antiviral_meds  behavioral_avoidance  \
respondent_id
0                                    0.0                   0.0
1                                    0.0                   1.0
2                                    0.0                   1.0
3                                    0.0                   1.0
4                                    0.0                   1.0

               behavioral_face_mask  behavioral_wash_hands  \
respondent_id
0                               0.0                    0.0
1                               0.0                    1.0
2                               0.0                    0.0
3                               0.0                    1.0
4                               0.0                    1.0

               behavioral_large_gatherings  behavioral_outside_home  \
respondent_id
0                                      0.0                      1.0
1                                      0.0                      1.0
2                                      0.0                      0.0
3                                      1.0                      0.0
4                                      1.0                      0.0

               behavioral_touch_face  doctor_recc_seasonal  \
respondent_id
0                                1.0                   0.0
1                                1.0                   0.0
2                                0.0                   0.0
3                                0.0                   1.0
4                                1.0                   0.0

               chronic_med_condition  child_under_6_months
health_worker  \
respondent_id

0                                0.0                   0.0
0.0
1                                0.0                   0.0
0.0
2                                1.0                   0.0
0.0
3                                1.0                   0.0
0.0
4                                0.0                   0.0
0.0

               health_insurance  opinion_seas_vacc_effective  \
respondent_id
0                           1.0                          2.0
```

```
1                           1.0                            4.0
2                           0.0                            4.0
3                           0.0                            5.0
4                           0.0                            3.0

               opinion_seas_risk  opinion_seas_sick_from_vacc  \
respondent_id
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

               household_adults  household_children  seasonal_vaccine

respondent_id

0                           0.0                 0.0                 0

1                           0.0                 0.0                 1

2                           2.0                 0.0                 0

3                           0.0                 0.0                 1

4                           1.0                 0.0                 0
```

```
train_target_float = pd.merge(train_target_copy, data_encoded,
left_index=True, right_index=True)
train_target_float.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26707 entries, 0 to 26706
Data columns (total 50 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   behavioral_antiviral_meds                 26707 non-null  float64
 1   behavioral_avoidance                      26707 non-null  float64
 2   behavioral_face_mask                      26707 non-null  float64
 3   behavioral_wash_hands                     26707 non-null  float64
 4   behavioral_large_gatherings               26707 non-null  float64
 5   behavioral_outside_home                   26707 non-null  float64
 6   behavioral_touch_face                     26707 non-null  float64
 7   doctor_recc_seasonal                      26707 non-null  float64
 8   chronic_med_condition                     26707 non-null  float64
 9   child_under_6_months                      26707 non-null  float64
 10  health_worker                             26707 non-null  float64
 11  health_insurance                          26707 non-null  float64
 12  opinion_seas_vacc_effective               26707 non-null  float64
 13  opinion_seas_risk                         26707 non-null  float64
```

```
 14  opinion_seas_sick_from_vacc                 26707 non-null  float64
 15  household_adults                            26707 non-null  float64
 16  household_children                          26707 non-null  float64
 17  seasonal_vaccine                            26707 non-null  int64
 18  age_group_18 - 34 Years                     26707 non-null  float64
 19  age_group_35 - 44 Years                     26707 non-null  float64
 20  age_group_45 - 54 Years                     26707 non-null  float64
 21  age_group_55 - 64 Years                     26707 non-null  float64
 22  age_group_65+ Years                         26707 non-null  float64
 23  education_12 Years                          26707 non-null  float64
 24  education_< 12 Years                        26707 non-null  float64
 25  education_College Graduate                  26707 non-null  float64
 26  education_Some College                      26707 non-null  float64
 27  education_Unknown                           26707 non-null  float64
 28  race_Black                                  26707 non-null  float64
 29  race_Hispanic                               26707 non-null  float64
 30  race_Other or Multiple                      26707 non-null  float64
 31  race_White                                  26707 non-null  float64
 32  sex_Female                                  26707 non-null  float64
 33  sex_Male                                    26707 non-null  float64
 34  marital_status_Married                      26707 non-null  float64
 35  marital_status_Not Married                  26707 non-null  float64
 36  marital_status_Unknown                      26707 non-null  float64
 37  rent_or_own_Own                             26707 non-null  float64
 38  rent_or_own_Rent                            26707 non-null  float64
 39  rent_or_own_Unknown                         26707 non-null  float64
 40  employment_status_Employed                  26707 non-null  float64
 41  employment_status_Not in Labor Force        26707 non-null  float64
 42  employment_status_Unemployed                26707 non-null  float64
 43  census_msa_MSA, Not Principle  City         26707 non-null  float64
 44  census_msa_MSA, Principle City              26707 non-null  float64
 45  census_msa_Non-MSA                          26707 non-null  float64
 46  income_poverty_<= $75,000, Above Poverty    26707 non-null  float64
 47  income_poverty_> $75,000                    26707 non-null  float64
 48  income_poverty_Below Poverty                26707 non-null  float64
 49  income_poverty_Unknown                      26707 non-null  float64
dtypes: float64(49), int64(1)
memory usage: 11.6 MB
```

Multivariate Analysis using Correlation

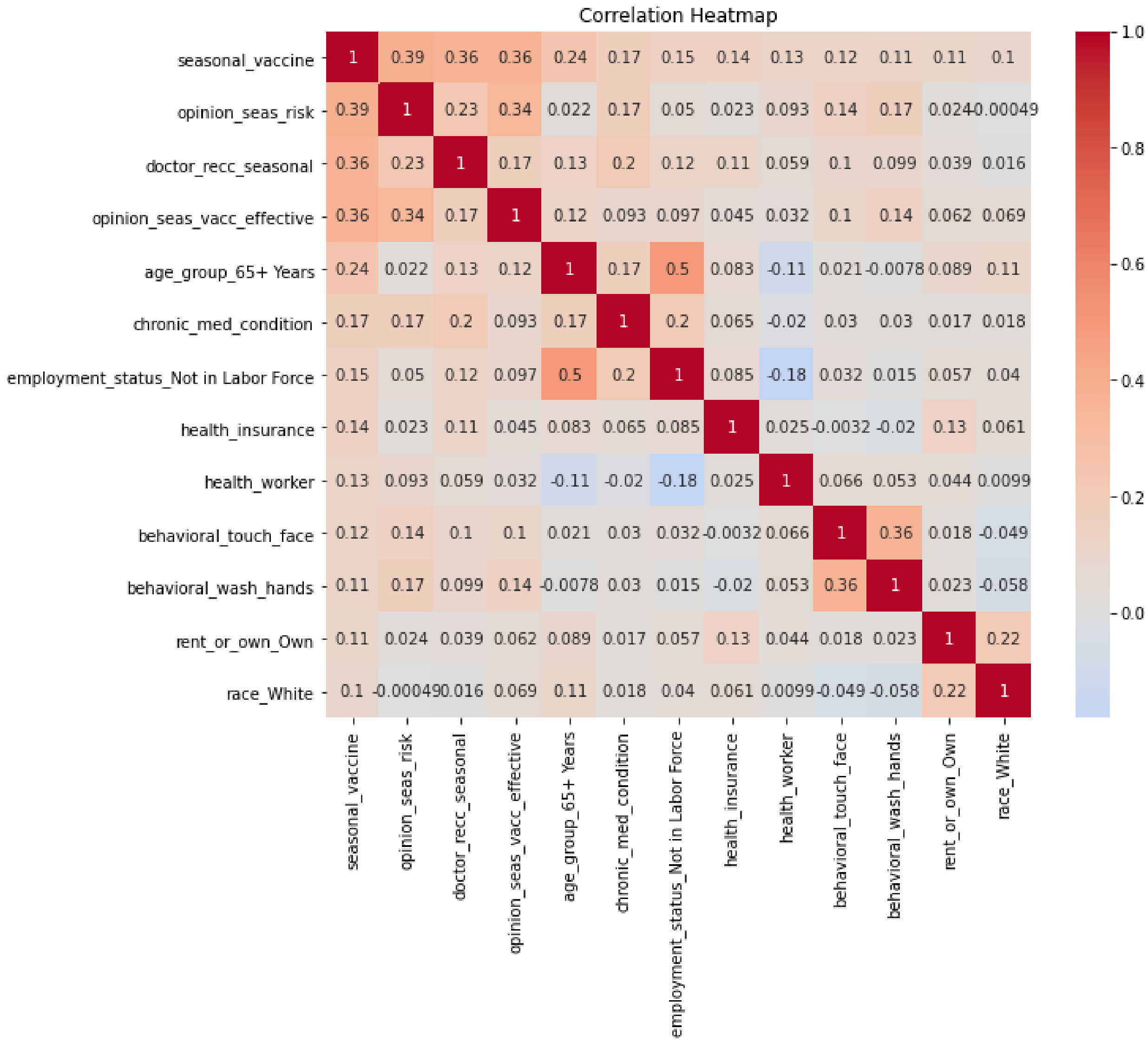
- Thirteen (13) features had a correlation greater than 10% (0.1) with the target.

```
# Set the data for corr
corr = train_target_float.corr()
['seasonal_vaccine'].sort_values(ascending = False)
corr = corr[(corr > 0.1)] # correlation greater than 0.1
columns = corr.index.tolist()

# df with only the selected columns
```

```
corr_df = train_target_float[columns] # columns with correlation > 0.1

corr
```

```
seasonal_vaccine                              1.000000
opinion_seas_risk                             0.386916
doctor_recc_seasonal                          0.360696
opinion_seas_vacc_effective                   0.358869
age_group_65+ Years                           0.244830
chronic_med_condition                         0.169465
employment_status_Not in Labor Force          0.145819
health_insurance                              0.138161
health_worker                                 0.126977
behavioral_touch_face                         0.119925
behavioral_wash_hands                         0.112254
rent_or_own_Own                               0.108002
race_White                                    0.100314
Name: seasonal_vaccine, dtype: float64
```

```
# plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_df.corr(), annot=True, cmap='coolwarm', center=0)
plt.title("Correlation Heatmap")
# plt.savefig("images/corr_map.png")

plt.show()
```



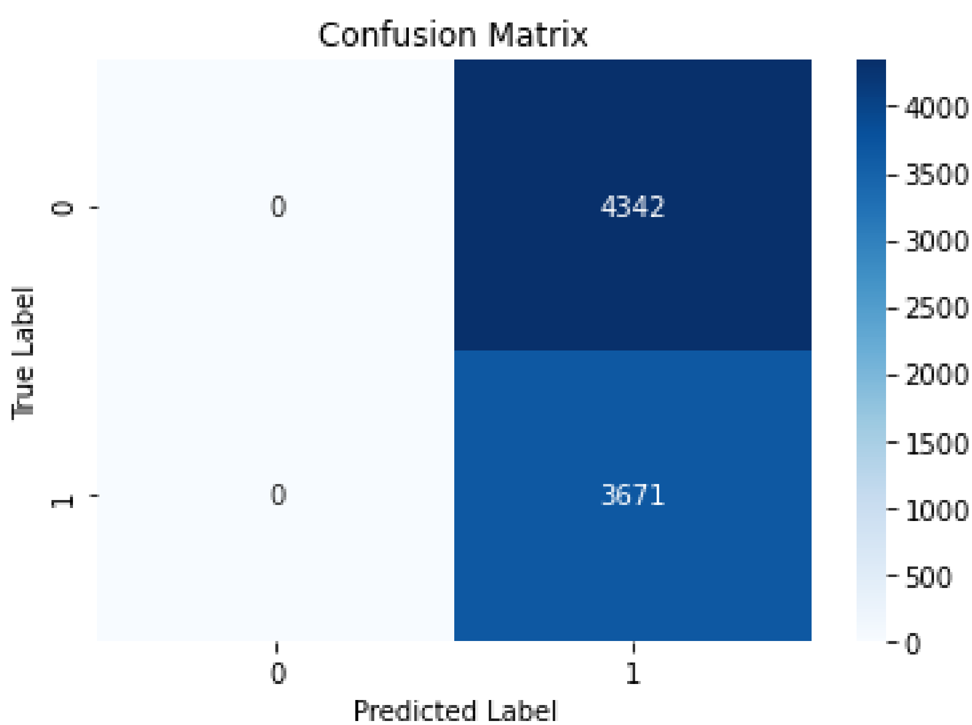
Modelling

Baseline Model

- Before conducting any modeling on the data, a "dummy" model that always predicts the positive class is first used.
  - "negative" is defined as a 0 (not received vaccine) and "positive" as a 1 (received thye vaccine).
  - Focus is on the test data, since this is will be used to evaluate the actual model as well.

```
# split data into train and test, claze size=0.3
X = train_target_float.drop(columns=["seasonal_vaccine"], axis=1)
y = train_target_float["seasonal_vaccine"]
```

```
# Perform train test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=42,test_size=0.3)

# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Training the dummy classifier
dummy_classifier = DummyClassifier(strategy="constant", constant=1)
dummy_classifier.fit(X_train_scaled, y_train)

# Make predictions
y_pred = dummy_classifier.predict(X_test_scaled)

# Create confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")

plt.show()
```



```
# Evaluation of the baseline model

#Accuracy=TP+TN/TP+TN+FP+FN
TP=3671
TN=0
FP=4342
FN=0
baseline_accuracy=(TP+TN)/(TP+FP+TN+FN)
print("Baseline Accuracy: " ,baseline_accuracy)

#precision TP/TP+FP
baseline_precision=(TP/(TP+FP))
print("Baseline Precision: " ,baseline_precision)

baseline_recall=(TP/(TP+FN))
print("Baseline Recall: " ,baseline_recall)

baseline_F1score=(2*baseline_precision*
baseline_recall)/(baseline_precision+baseline_recall)
print("Baseline F1score: " ,baseline_F1score)
```

```
Baseline Accuracy:  0.4581305378759516
Baseline Precision:  0.4581305378759516
Baseline Recall:  1.0
Baseline F1score:  0.6283806915439919
```

Baseline Model Observations

1. **Baseline Accuracy** is approximately 45.81%, it means that the dummy classifier, correctly predicts around 45.81% of instances in the test data.

2. **Baseline Precision** is also 45.81%. It is equal to the accuracy since the dummy classifier always predicts the positive class.

3. **Baseline Recall** is 100%. Since the dummy classifier always predicts the positive class, it correctly identifies all the actual positive instances.

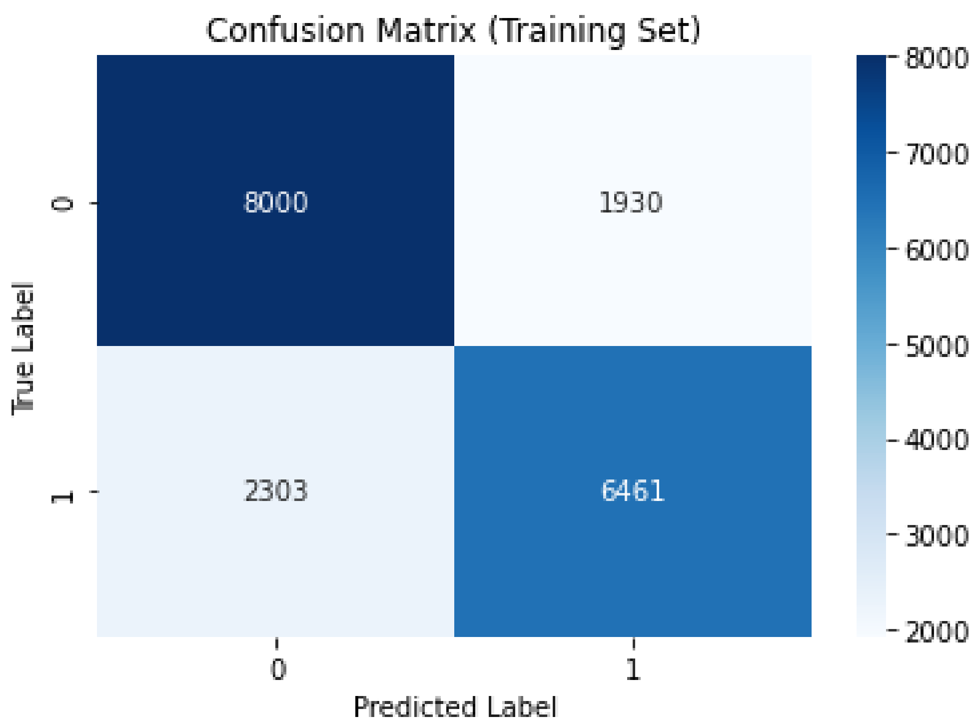
4. **Baseline F1-score** is approximately 62.84%. A higher F1-score would indicate a better balance between precision and recall.

These metrics will be used to reference the performance of subsequent models, hoping that they will outperform the baseline model.

Model 1. Logistic Regression

```
# fit the model in logistic regression

# Instantiate the model
model1 = LogisticRegression(random_state=42)
```

```
# Fit the model on the scaled data
model1.fit(X_train_scaled, y_train)

# Make predictions on the training data
y_train_pred1 = model1.predict(X_train_scaled)

# Create confusion matrix
cm = confusion_matrix(y_train, y_train_pred1)

# Plot confusion matrix
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix (Training Set)")


plt.show()
```



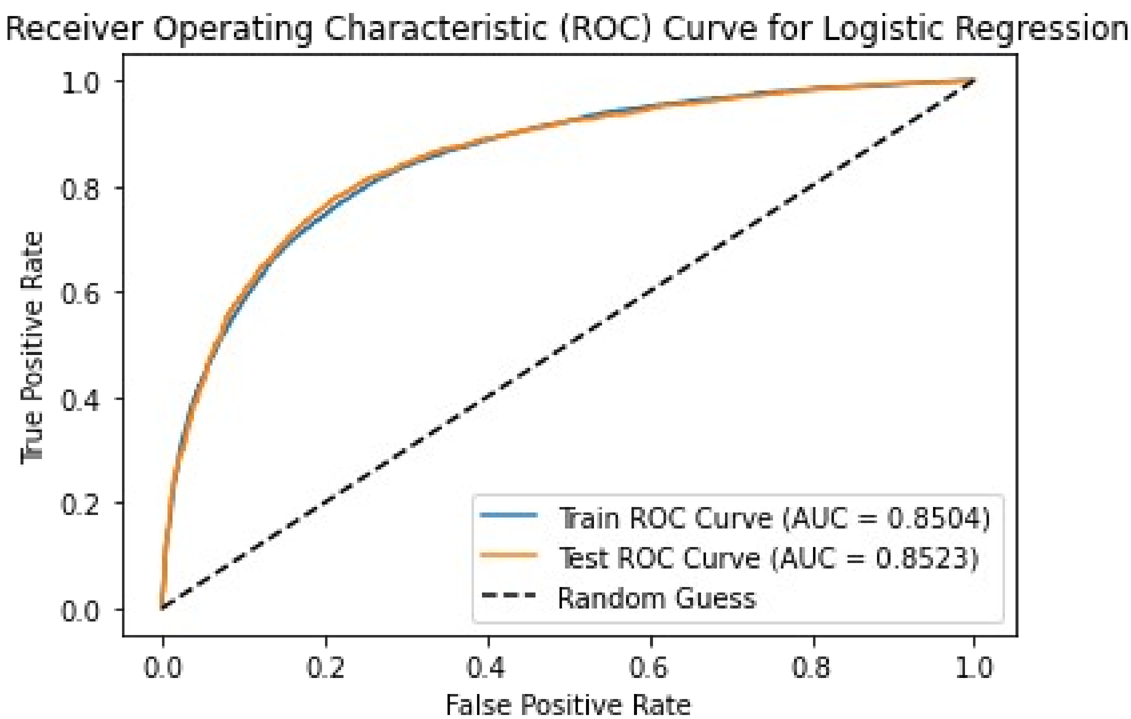

```
# cross validate the model using 3 kfolds
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(model1, X_train_scaled, y_train, cv=3)
print("Cross-Validation Scores:", cv_scores)

average_cv_score = cv_scores.mean()
print("Average Cross-Validation Score:", average_cv_score * 100) # in
percentage
```

```
Cross-Validation Scores: [0.76813222 0.77210721 0.77820575]
Average Cross-Validation Score: 77.28150573893545
```

- The scores indicate that the model is approximately 77.28% accurate in its performance.

```
#evaluation of model
y_pred = model1.predict(X_test_scaled)
model1_accuracy = accuracy_score(y_test, y_pred)
model1_recall = recall_score(y_test, y_pred)
model1_precision = precision_score(y_test, y_pred)
model1_f1 = f1_score(y_test, y_pred)

print(f"""
Accuracy Fitted Model 1: {model1_accuracy:1.3f}
Recall Fitted Model 1: {model1_recall:1.3f}
Precision Fitted Model 1: {model1_precision:1.3f}
F1 Score Fitted Model 1: {model1_f1:1.3f}
""")


Accuracy Fitted Model 1: 0.782
Recall Fitted Model 1: 0.739
Precision Fitted Model 1: 0.774
F1 Score Fitted Model 1: 0.756


# calcluate ROC

# Obtain the predicted probabilities for the positive class
y_test_prob = model1.predict_proba(X_test_scaled)[:, 1]
y_train_prob = model1.predict_proba(X_train_scaled)[:, 1]

# Calculate the false positive rate (fpr), true positive rate (tpr),
and thresholds
train_fpr_l, train_tpr_l, thresholds = roc_curve(y_train,
y_train_prob)
test_fpr_l, test_tpr_l, thresholds = roc_curve(y_test, y_test_prob)

# Calculate the AUC score
auc_score_model1_train = roc_auc_score(y_train, y_train_prob)
auc_score_model1_test = roc_auc_score(y_test, y_test_prob)

print("Train AUC Score", auc_score_model1_train)
print("Test AUC Score", auc_score_model1_test)

# Plot the ROC curve
plt.plot(train_fpr_l, train_tpr_l, label="Train ROC Curve (AUC =
{:.4f})".format(auc_score_model1_train))
plt.plot(test_fpr_l, test_tpr_l, label="Test ROC Curve (AUC =
{:.4f})".format(auc_score_model1_test))
plt.plot([0, 1], [0, 1], "k--", label="Random Guess")
```

```
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curve for Logistic
Regression")
plt.legend(loc="lower right")


plt.show()

Train AUC Score 0.8503931732533946
Test AUC Score 0.8522644587822867
```



Logistic Regression Observations

- An **accuracy** of 0.782 means that the model is correctly predicting the seasonal vaccine outcome for around 78% of the samples in the test data.
- A **recall** of 0.739 indicates that the model is able to correctly identify around 74% of the positive instances (those who received the flu vaccine) in the test data.
- A **precision** of 0.774 implies that around 77% of the instances predicted as positive by the model are actually true positives.
- The **F1 score** combines both precision and recall into a single metric. With an F1 score of 0.756, it suggests a balanced performance between precision and recall. These metrics are a good indication that the logistic regression model is providing reasonably accurate predictions on the uptake of the seasonal flu vaccine.

The **ROC curve** above shows a AUC score of 0.8523 on the test set, revealing that the model is quite good on distinguishing between those who received the seasonal flu vaccine or not (positives and negatives).

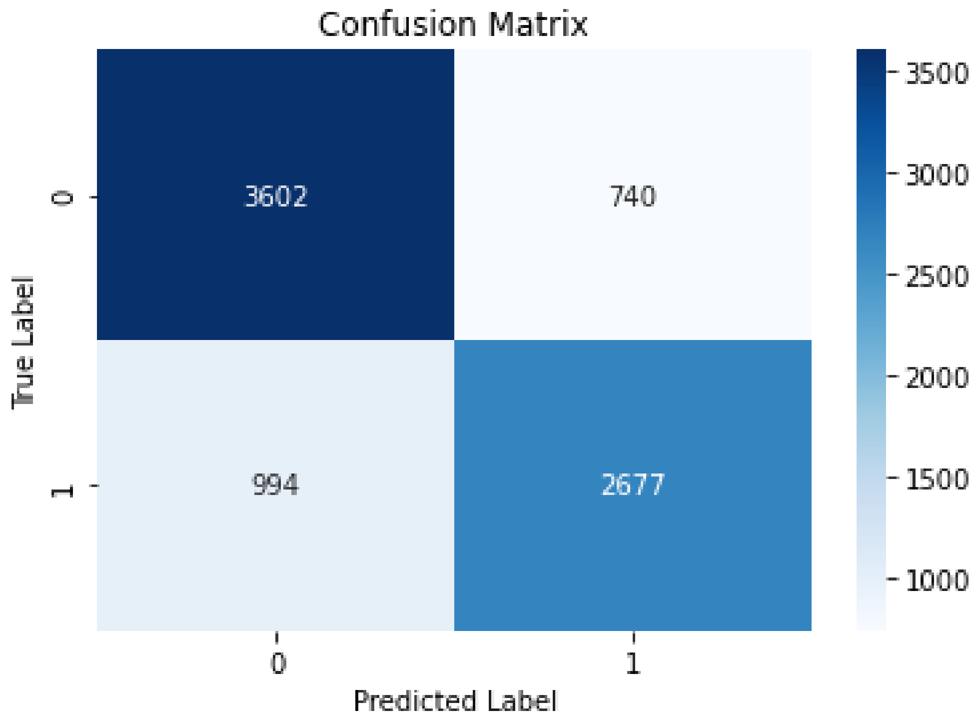
- The score is close to 1, meaning that the predictive power of the model can be trusted.

Model 2 - Random Forest

```
# Perform feature engineering or transformation

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# instantiate
model2 = RandomForestClassifier(random_state=42)

# Perform hyperparameter tuning using grid search
param_grid = {"n_estimators": [100, 200, 300], "max_depth": [None, 5,
10]}
grid_search = GridSearchCV(model2, param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)
best_model2 = grid_search.best_estimator_

# Fit the best model on the scaled data
best_model2.fit(X_train_scaled, y_train)
```

```
RandomForestClassifier(max_depth=10, n_estimators=200,
random_state=42)
```

```
y_pred2 = best_model2.predict(X_test_scaled)

# Create confusion matrix
cm = confusion_matrix(y_test, y_pred3)

# Plot confusion matrix
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")

plt.show()
```

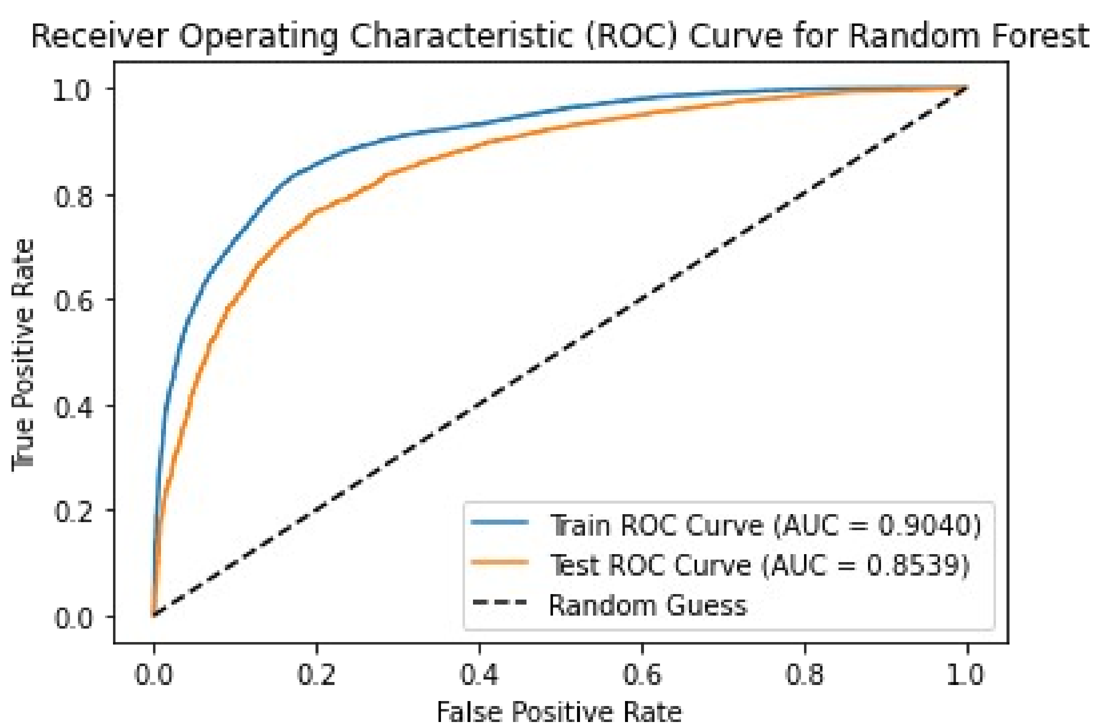



```
#evaluation of model
y_pred = best_model2.predict(X_test_scaled)
model2_accuracy = accuracy_score(y_test, y_pred)
model2_recall = recall_score(y_test, y_pred)
model2_precision = precision_score(y_test, y_pred)
model2_f1 = f1_score(y_test, y_pred)

print(f"""
Accuracy Fitted Model: {model2_accuracy:1.3f}
Recall Fitted Model: {model2_recall:1.3f}
Precision Fitted Model: {model2_precision:1.3f}
F1 Score Fitted Model: {model2_f1:1.3f}
""")
```

```
Accuracy Fitted Model: 0.784
Recall Fitted Model: 0.729
Precision Fitted Model: 0.783
F1 Score Fitted Model: 0.755
```

```
# Predict on training and test sets
training_preds2 = best_model2.predict_proba(X_train_scaled)[:, 1]
test_preds2 = best_model2.predict_proba(X_test_scaled)[:, 1]

# Calculate false positive rate (fpr), true positive rate (tpr), and
thresholds for ROC curve
training_fpr_r, training_tpr_r, _ = roc_curve(y_train,
```

```
training_preds2)
test_fpr_r, test_tpr_r, _ = roc_curve(y_test, test_preds2)

# Calculate the AUC score
training_auc_model2 = roc_auc_score(y_train, training_preds2)
test_auc_model2 = roc_auc_score(y_test, test_preds2)
print("Train AUC Score: {:.4f}". format(training_auc_model2))
print("Test AUC Score: {:.4f}".format(test_auc_model2))

# Plot the ROC curve
plt.plot(training_fpr_r, training_tpr_r, label="Train ROC Curve (AUC =
{:.4f})".format(training_auc_model2))
plt.plot(test_fpr_r, test_tpr_r, label="Test ROC Curve (AUC =
{:.4f})".format(test_auc_model2))
plt.plot([0, 1], [0, 1], "k--", label="Random Guess")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curve for Random
Forest")
plt.legend(loc="lower right")


plt.show()

Train AUC Score: 0.9040
Test AUC Score: 0.8539
```



Random Forest Observations

- **Accuracy**: the model correctly predicted vaccine uptake with a score of 78.4%.
- **Recall**: 72.9% of actual vaccine recipients were correctly identified.

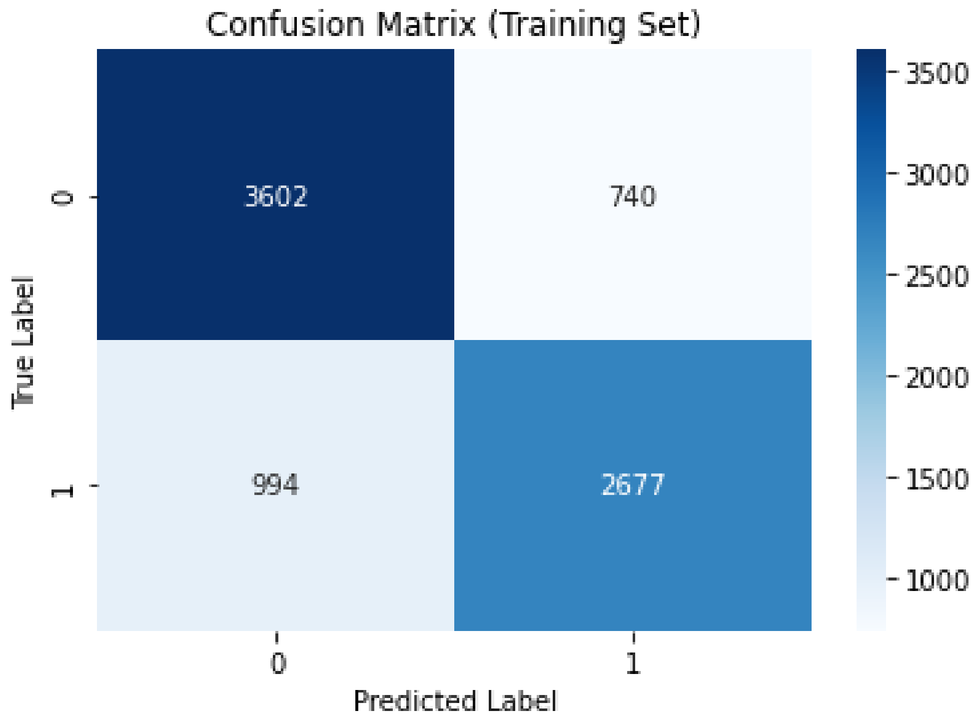
- **Precision**: 78.3% of those predicted to have taken the vaccine actually took the vaccine.
- **F1-Score**: at 75.5%, it is a good balance between precision and recall.

This model's **ROC curve** shows AUC of 0.8539, the highest so far. The model is more effective in predicting the seasonal vaccine uptake based on the provided features.It is able to differentiate between vaccine recipients and non-recipients effectively.

Model 3 - Decision Trees

```
# Test set predictions
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

# instantiate
model3 = DecisionTreeClassifier(criterion="gini", max_depth=5)

# fit the model on train data
model3.fit(X_train_scaled, y_train)

y_pred3 = model3.predict(X_test_scaled)

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred2)

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix (Training Set)")
```

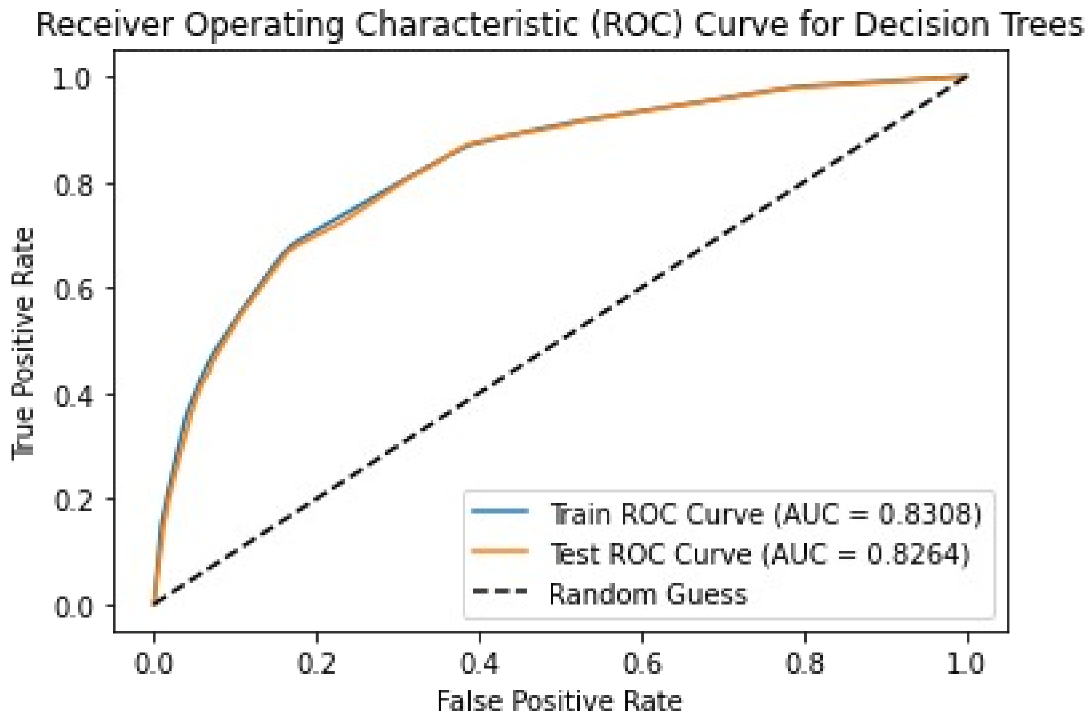
```
Text(0.5, 1.0, 'Confusion Matrix (Training Set)')
```




```
#evaluation of model
y_pred = model3.predict(X_test_scaled)
model3_accuracy = accuracy_score(y_test, y_pred)
model3_recall = recall_score(y_test, y_pred)
model3_precision = precision_score(y_test, y_pred)
model3_f1 = f1_score(y_test, y_pred)

print(f"""
Accuracy Fitted Model: {model3_accuracy:1.3f}
Recall Fitted Model: {model3_recall:1.3f}
Precision Fitted Model: {model3_precision:1.3f}
F1 Score Fitted Model: {model3_f1:1.3f}
""")


Accuracy Fitted Model: 0.758
Recall Fitted Model: 0.675
Precision Fitted Model: 0.769
F1 Score Fitted Model: 0.719


# ROC and AUC
# calcluate ROC

# Obtain the predicted probabilities for the positive class
y_test_pred = model3.predict_proba(X_test_scaled)[:, 1]
y_train_pred = model3.predict_proba(X_train_scaled)[:, 1]
```

```
# Calculate the false positive rate (fpr), true positive rate (tpr),
and thresholds
training_fpr_d, training_tpr_d, _  = roc_curve(y_train, y_train_pred)
test_fpr_d, test_tpr_d, _  = roc_curve(y_test, y_test_pred)

# Calculate the AUC score
auc_score_model3_test = roc_auc_score(y_test, y_test_pred)
auc_score_model3_train = roc_auc_score(y_train, y_train_pred)

print("Train AUC Score", auc_score_model3_train)
print("Test AUC Score", auc_score_model3_test)

# Plot the ROC curve
plt.plot(training_fpr_d, training_tpr_d, label="Train ROC Curve (AUC =
{:.4f})".format(auc_score_model3_train))
plt.plot(test_fpr_d, test_tpr_d, label="Test ROC Curve (AUC =
{:.4f})".format(auc_score_model3_test))
plt.plot([0, 1], [0, 1], "k--", label="Random Guess")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curve for Decision
Trees")
plt.legend(loc="lower right")


plt.show()
```

```
Train AUC Score 0.8308051327342516
Test AUC Score 0.8264301813572109
```



Decision Trees Observations

- **Accuracy**: overall, accuracy stands at approximately 0.76, indicating that around 76% of samples are correctly predicted.
- **Recall** shows that about 67.5% of those who received the vaccine were identified.
- **Precision** indicates the model got 76.9% in prediciting vaccine recipients as actual recipients.
- **F1-Score**of 71.9% represents precision and recall in a balanced way.

As per the **ROC curve**, this model has AUC of 0.8264, which is slightly lower than the Logistic Regression model. It is still a commendable performance of predicting positives as positives and negatives as negatives.

Model 4. XGBoost Algorithm

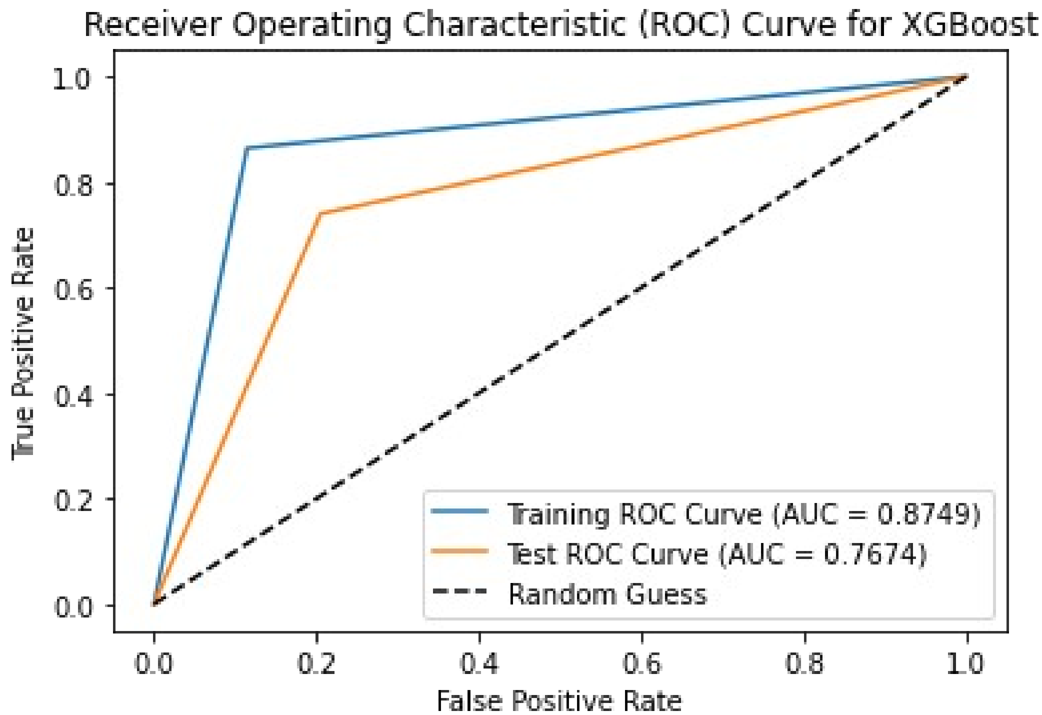
```
# import library
from xgboost import XGBClassifier

# Instantiate XGBClassifier
model4 = XGBClassifier()

# Fit XGBClassifier
model4.fit(X_train_scaled, y_train)

# Predict on training and test sets
training_preds = model4.predict_proba(X_train_scaled)[:, 1]
test_preds = model4.predict_proba(X_test_scaled)[:, 1]
```

```python
# Predict on training and test sets
training_preds = model4.predict(X_train_scaled)
test_preds = model4.predict(X_test_scaled)

# Accuracy of training and test sets
training_accuracy = accuracy_score(y_train, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print("Training Accuracy: {:.4}".format(training_accuracy))
print("Validation accuracy: {:.4}".format(test_accuracy))
print()

# evaluation
# Calculating precision, recall, and F1-score for the validation set
model4_precision = precision_score(y_test, test_preds)
model4_recall = recall_score(y_test, test_preds)
model4_f1 = f1_score(y_test, test_preds)

print("Precision: {:.4f}".format(model4_precision))
print("Recall: {:.4f}".format(model4_recall))
print("F1-Score: {:.4f}".format(model4_f1))
```

```
Training Accuracy: 0.8755
Validation accuracy: 0.7697

Precision: 0.7532
Recall: 0.7399
F1-Score: 0.7465
```

```python
# Calculate false positive rate (fpr), true positive rate (tpr), and
thresholds for ROC curve
training_fpr_x, training_tpr_x, _ = roc_curve(y_train, training_preds)
test_fpr_x, test_tpr_x, _ = roc_curve(y_test, test_preds)

# Calculate AUC scores for training and test sets
training_auc_model4 = roc_auc_score(y_train, training_preds)
test_auc_model4 = roc_auc_score(y_test, test_preds)

# Print AUC Score
print("Train AUC Score: {:.4f}". format(training_auc_model4))
print("Test AUC Score: {:.4f}".format(test_auc_model4))

# Plot the ROC curve
plt.plot(training_fpr_x, training_tpr_x, label="Training ROC Curve
(AUC = {:.4f})".format(training_auc_model4))
plt.plot(test_fpr_x, test_tpr_x, label="Test ROC Curve (AUC =
{:.4f})".format(test_auc_model4))
plt.plot([0, 1], [0, 1], "k--", label="Random Guess")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
plt.title("Receiver Operating Characteristic (ROC) Curve for XGBoost")
plt.legend(loc="lower right")

plt.show()

Train AUC Score: 0.8749
Test AUC Score: 0.7674
```



XGBoost Algorithm Observations

- This model has performed better than Logistic Regression and Decision Trees. It is highly likely that the algorithm has lived up to its performance characteristics of capturing complex patterns in data to provide high and improved accuracy. The results are as follows:
  - **Training Accuracy** of 87.55% suggests that this model was able to classify approcximately 87.55% of the samples in the training data.
  - **Validation Accuracy** of 76.97% suggests that this model was able to classify approcximately 76.97% of the samples in the test data.
  - **Recall**: 73.99% of actual vaccine recipients (actual positives) were correctly identified.
  - **Precision**: at 75.32%, the model correcty identified vaccine recipients as true positives.
  - **F1-Score**: at 74.65%, it is a good balance between precision and recall.

Analysis of the **ROC curve** reveals that the model was the least powerful in prediciting the test data. An AUC of 0.7674 on the test data, while still high, is the least among the four models.

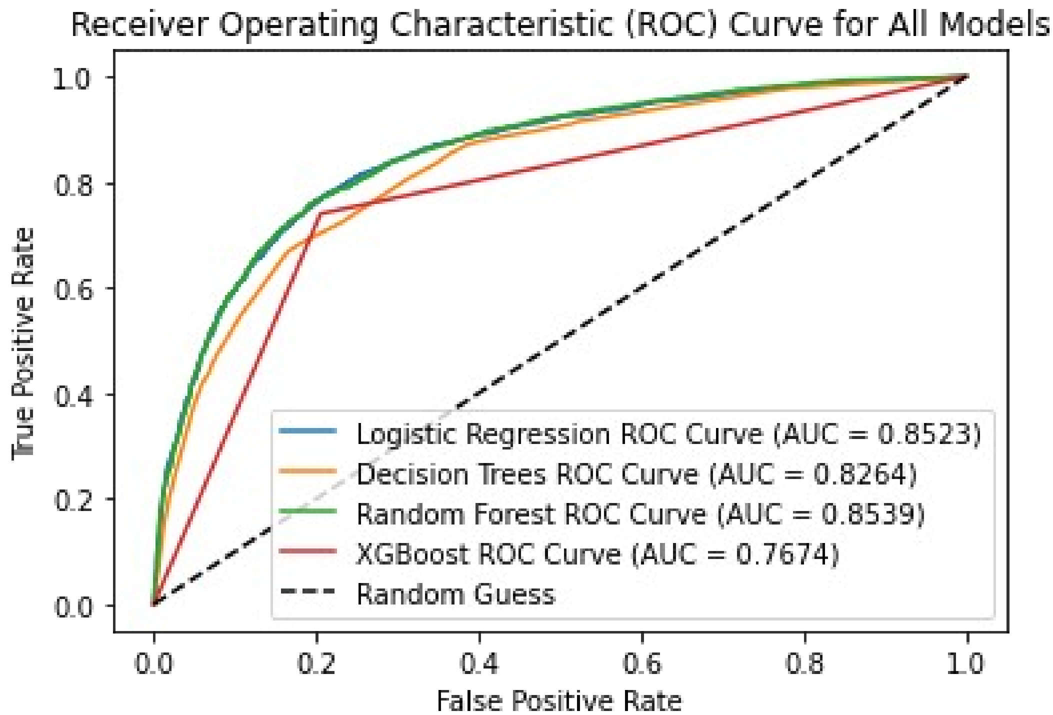
Model Evaluation Summary

```
# create the summary df and define columns
scores = pd.DataFrame(np.array([
        ['Logistic Regression', 78.2, 73.9, 77.4, 85.0, 85.2],
         ['Random Forest', 78.4, 72.9, 78.3, 90.4, 85.4],
        ['Decision Tree', 75.8, 67.5, 76.9, 83.1, 82.6],
       ['XGBoost', 77.0, 74.0, 75.3, 87.5, 76.7]
]))
scores.columns = ["Model", "Accuracy", "Recall", "Precision",
"Training AUC Score", "Test AUC Score"]
scores

                 Model Accuracy Recall Precision Training AUC Score  \
0  Logistic Regression     78.2   73.9      77.4               85.0
1        Random Forest     78.4   72.9      78.3               90.4
2        Decision Tree     75.8   67.5      76.9               83.1
3              XGBoost     77.0   74.0      75.3               87.5

  Test AUC Score
0           85.2
1           85.4
2           82.6
3           76.7
```

- The **Random Forest Model** (best_model3) has demonstrated commendable performance in predicting the uptake of the seasonal flu vaccine. It has strong evaluation metrics and an ROC curve with strong discriminatory power.

Final ROC Curve (all combined for test data)

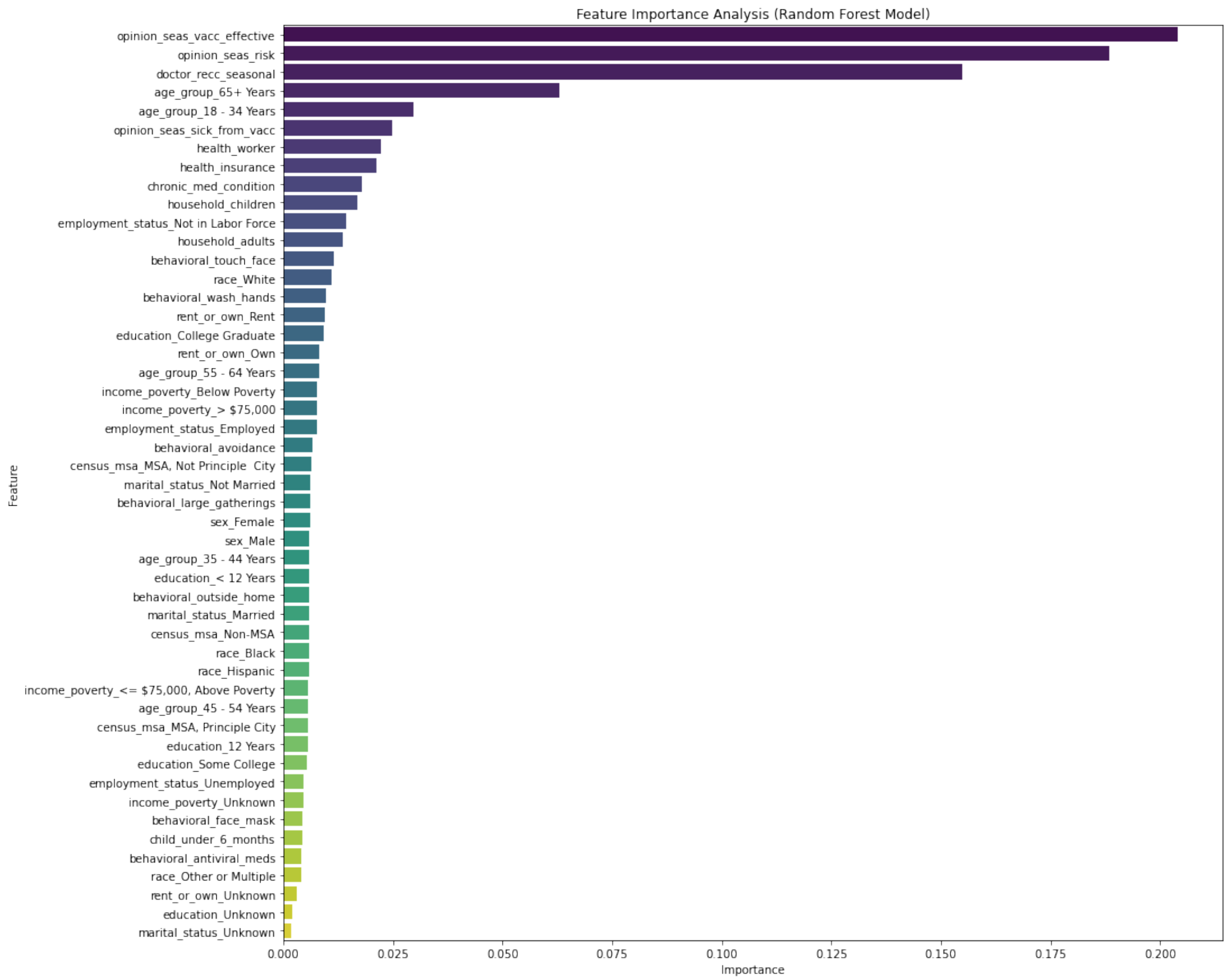
```
plt.plot(test_fpr_l, test_tpr_l, label="Logistic Regression ROC Curve
(AUC = {:.4f})".format(auc_score_model1_test))
plt.plot(test_fpr_d, test_tpr_d, label="Decision Trees ROC Curve (AUC
= {:.4f})".format(auc_score_model2_test))
plt.plot(test_fpr_r, test_tpr_r, label="Random Forest ROC Curve (AUC =
{:.4f})".format(test_auc_model3))
plt.plot(test_fpr_x, test_tpr_x, label="XGBoost ROC Curve (AUC =
{:.4f})".format(test_auc_model4))

plt.plot([0, 1], [0, 1], "k--", label="Random Guess")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curve for All
Models")
plt.legend(loc="lower right")

plt.show()
```



Feature Importance Analysis

- This step will help to further understand the most important features when predicting seasonal flu vaccine uptake.

```
# Get feature importances from the Random Forest model (Model 3)
importance_scores = best_model3.feature_importances_

# DataFrame with feature names and importance scores
feature_importance_df = pd.DataFrame({"Feature": X_train.columns,
"Importance": importance_scores})

# Sort the features by importance in descending order
feature_importance_df =
feature_importance_df.sort_values(by="Importance", ascending=False)

# Plotting feature importance
plt.figure(figsize=(15, 15))
sns.barplot(x="Importance", y="Feature", data=feature_importance_df,
palette="viridis")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.title("Feature Importance Analysis (Random Forest Model)")
plt.show()
```




```
# Set a threshold for feature importance
threshold = 0.05

# Select the features with importance scores above the threshold
selected_features =
feature_importance_df.loc[feature_importance_df["Importance"] >
threshold, "Feature"]
selected_features

12    opinion_seas_vacc_effective
13              opinion_seas_risk
7            doctor_recc_seasonal
21            age_group_65+ Years
Name: Feature, dtype: object
```

Feature Importance Observations
The following were the top features influencing the uptake of the seasonal vaccine in order of importance:

1. **`opinion_seas_vacc_effective`**

- As the most influential feature, a respondent's opinion on whether the seasonal flu vaccine was effective mattered a lot.
- This feature is under objective 4

1. `opinion_seas_risk`

- An individual's opinion about risk of getting sick with seasonal flu without the vaccine was the second most important feature in determining the indivuals uptake of the vaccine.
- This feature is under objective 4.

1. `doctor_recc_seasonal`

- This is the third most influential feature. It suggests that a doctor recommending the seasonal flu vaccine had a significant influence on vaccine uptake.
- This feature is under objective 4.

1. `age_group_65+ Years`

- Older respondents (65 and above) seemed to prioritize vaccination as the feature played a big role in the decision on vaccine uptake. Other concerns such as health may have influenced respondents in this age group.
- This is a demographic feature.

These are the features that play the most significant role in predicting how likely an individual is to get the seasonal flu vaccine out of all available features.

Conclusion

The Data

- The data required a lot of exploration and engineering. Some important features had missing values that would have made the data biased. Other features showed a bias, such as the race feature that was heavily tilted towards the white race.

The Models

- The models' performance was almost similar. However, with each iteration, the scores and accuracies changed. **Random Forest** emerged the best with an AUC score of 0.8539 on the test data.
- The process of choosing and optimizing model hyperparameters was time consuming.
- A better balance on the data may give more promising results.

In summary:

- The role of healthcare professionals can never be downplayed as evidenced by the fact that a doctor's recommendation to get the flu vaccine played a big role.
- Public perception is also important. How people view and feel about vaccines is a big influence on vaccine uptake.
- Age is also a factor. Older people tended to get the seasonal flue vaccine more than younger people.

Recommendations

1. Embrace personalized outreach as a campaign tool so as to target individuals and mould their perception towards immunization.
2. Public campaigns should be geared towards bringing onboard more younger people as it seems that they are less likely to get the seasonal flu vaccines.
3. The public health sector should continue encouraging doctors to recommend suitable vaccines to their clients. This modelling and analysis process has shown that people are highly likely to listen to their doctor's advice.

For further improvements:

1. Conduct more feature engineering to get more insight on features influencing uptake of the vaccine.
2. Using more recent data to create predictions, especially after the recent Covid-19 pandemic, may provide better outlooks on the results.