PREDICTION OF SEASONAL FLU VACCINES UPTAKE ¶

- Student name: Paul Mawa Musau
- Scheduled project review date: 24/05/2023
- Student title: Data Scientist

Problem Statement

This study aims to create a prediction model to anticipate whether or not people will get a flu shot during the annual flu season.The flu season occurs on an annual basis, and each year people choose whether or not to get the flu shot.The model should be able to discover patterns and factors that influence people's vaccination decisions using historical data.The goal is to accurately predict whether an individual will get the flu shot or not by assessing data such as age groups, genders, and other relevant features.

Project Objectives

To grasp the statistical implications of variables such as age groups, genders, and the existence of children in homes, analyze them.Respond to enquiries about targeting certain subsets of the population to increase overall immunization rates.Excluding any specified aims, investigate the factors influencing individuals' decision to obtain the flu vaccination during the annual flu season.Create a predictive model to forecast whether or not people will receive the

Data Description The datasets used for this project were downloaded from [Kaggle (%22https://www.kaggle.com/datasets/arashnic/flu-data)](%22https://www.kaggle.com/datasets/arashnic/flu-data). It contains information on the social,economic and demographic backgrounds of the respondents as well as their opinions on the H1N1 and seasonal flu vaccines. The training data has 26707 rows and 36 columns. The information contained with the columns is as follows as described by the data [dictionary (%22https://www.kaggle.com/datasets/arashnic/flu-data)](%22https://www.kaggle.com/datasets/arashnic/flu-data):

No.	Column	Description
1	`respondent_id`	Unique and random identifier for the respondents
2	`h1n1_concern`	Level of concern about H1N1 flu with 0 being not concerned at all and 3 being very concerned
3	`h1n1_knowledge`	Level of knowledge about H1N1 with 0 being no knowledge and 2 being a lot of knowledge
4	`behavioral_antiviral_meds`	Has taken any antiviral medication (0-no,1-yes)
5	`behavioral_avoidance`	Has avoided close contact with anyone with flu-like symptoms (0-no,1-yes)
6	`behavioral_face_mask`	Has bought a face mask (0-no,1-yes)
7	`behavioral_wash_hands`	Has frequently washed hands or used hand sanitizer (0-no,1-yes)
8	`behavioral_large_gatherings`	Has reduced time at large gatherings (0-no,1-yes)
9	`behavioral_outside_home`	Has reduced contact with people outside of own household (0-no,1-yes)
10	`behavioral_touch_face`	Has avoided touching eyes, nose or mouth (0-no,1-yes)
11	`doctor_recc_h1n1`	H1N1 flu vaccine was recommended by doctor (0-no,1-yes)
12	`doctor_recc_seasonal`	H1N1 flu vaccine was recommended by doctor (0-no,1-yes)
13	`chronic_med_condition`	Has any of the following chronic conditions: asthma or any lung condition, a heart condition, a kidney condition, sickle cell anaemia or any other anaemia, a neurological or neouromuscular condition, a liver condition, or a weakened immune system as a result of a chronic illness or medicines taken for a chronic illness (0-no,1-yes)
14	`child_under_6_months`	Has regular close contact with a child under the age of six months (0-no,1-yes)
15	`health_worker`	Is a healthcare worker (0-no,1-yes)
16	`health_insurance`	Has health insurance (0-no,1-yes)
17	`opinion_h1n1_vacc_effective`	Respondent's opinion on the efficacy of the vaccine with 1 being not at all effective and 5 being very effective
18	`opinion_h1n1_risk`	Respondent's opinion about risk of getting sick with H1N1 flu without vaccine with 1 being very low and 5 being very high
19	`opinion_h1n1_sick_from_vacc`	Respondent's worry of getting sick from H1N1 vaccine with 1 being not worried at all and 5 being very worried
20	`opinion_seas_vacc_effective`	Respondent's opinion about seasonal flu vaccine effectiveness with 1 being not effective at all and 5 being very effective

No.	Column	Description
21	opinion_seas_risk	Respondent's opinion about risk of getting sick with seasonal flu without vaccine with 1 being very low and 5 being very high
22	opinion_seas_sick_from_vacc	Respondent's worry of getting sick from taking seasonal flu vaccine with 1 being not worried at all and 5 being very worried
23	age_group	Age group of respondents
24	education	Self-reported educational level
25	race	Race of respondent
26	sex	Sex of respondent
27	income_poverty	Household annual income of respondent with respect to 2008 Census poverty thresholds
28	marital status	Marital status of respondent
29	rent_or_own	Housing situation of respondent
30	employment_status	Employment status of respondent
31	hhs_geo_region	Respondent's residence using a 10-region geographic classification defined by the U.S. Dept. of Health and Human Services. Values are represented as short random character strings
32	census_msa	Respondent's residence within metropolitan statistical areas (MSA) as defined by the U.S. Census
33	household_adults	Number of other adults in the household, top-coded to 3
34	household_children	Number of children in the household, top-coded to 3
35	employment_industry	Type of industry respondent is employed in. Values are represented as short random character strings

DATA UNDERSTANDING

Reading the Data

In [201]:
```
# import relevant library
import pandas as pd

# load features into dataframe
df = pd.read_csv('H1N1_Flu_Vaccines.csv', index_col='respondent_id')
df.head()
```

Out[201]:

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

In [202]:
```
# import display that can display maximum columns and rows
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_rows', 200)
```

In [203]:
```
df.head()
```

Out[203]:

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

In [204]:

```
# get basic data information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26707 entries, 0 to 26706
Data columns (total 37 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   h1n1_concern                 26615 non-null  float64
 1   h1n1_knowledge               26591 non-null  float64
 2   behavioral_antiviral_meds    26636 non-null  float64
 3   behavioral_avoidance         26499 non-null  float64
 4   behavioral_face_mask         26688 non-null  float64
 5   behavioral_wash_hands        26665 non-null  float64
 6   behavioral_large_gatherings  26620 non-null  float64
 7   behavioral_outside_home      26625 non-null  float64
 8   behavioral_touch_face        26579 non-null  float64
 9   doctor_recc_h1n1             24547 non-null  float64
 10  doctor_recc_seasonal         24547 non-null  float64
 11  chronic_med_condition        25736 non-null  float64
 12  child_under_6_months         25887 non-null  float64
 13  health_worker                25903 non-null  float64
 14  health_insurance             14433 non-null  float64
 15  opinion_h1n1_vacc_effective  26316 non-null  float64
 16  opinion_h1n1_risk            26319 non-null  float64
 17  opinion_h1n1_sick_from_vacc  26312 non-null  float64
 18  opinion_seas_vacc_effective  26245 non-null  float64
 19  opinion_seas_risk            26193 non-null  float64
 20  opinion_seas_sick_from_vacc  26170 non-null  float64
 21  age_group                    26707 non-null  object 
 22  education                    25300 non-null  object 
 23  race                         26707 non-null  object 
 24  sex                          26707 non-null  object 
 25  income_poverty               22284 non-null  object 
 26  marital_status               25299 non-null  object 
 27  rent_or_own                  24665 non-null  object 
 28  employment_status            25244 non-null  object 
 29  hhs_geo_region               26707 non-null  object 
 30  census_msa                   26707 non-null  object 
 31  household_adults             26458 non-null  float64
 32  household_children           26458 non-null  float64
 33  employment_industry          13377 non-null  object 
 34  employment_occupation        13237 non-null  object 
 35  h1n1_vaccine                 26707 non-null  int64  
 36  seasonal_vaccine             26707 non-null  int64  
dtypes: float64(23), int64(2), object(12)
memory usage: 7.7+ MB
```

In [205]:
```
# preview summary statistics of columns
df.describe()
```

Out[205]:

	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	be
count	26615.000000	26591.000000	26636.000000	26499.000000	
mean	1.618486	1.262532	0.048844	0.725612	
std	0.910311	0.618149	0.215545	0.446214	
min	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	1.000000	0.000000	0.000000	
50%	2.000000	1.000000	0.000000	1.000000	
75%	2.000000	2.000000	0.000000	1.000000	
max	3.000000	2.000000	1.000000	1.000000	

In [206]:
```
# get the shape of the data

df.shape
```

Out[206]:
```
(26707, 37)
```

In [207]:
```
# get column names

df.columns
```

Out[207]:
```
Index(['h1n1_concern', 'h1n1_knowledge', 'behavioral_antiviral_meds',
       'behavioral_avoidance', 'behavioral_face_mask', 'behavioral_wash_
hands',
       'behavioral_large_gatherings', 'behavioral_outside_home',
       'behavioral_touch_face', 'doctor_recc_h1n1', 'doctor_recc_seasona
l',
       '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_ms
a',
       'household_adults', 'household_children', 'employment_industry',
       'employment_occupation', 'h1n1_vaccine', 'seasonal_vaccine'],
      dtype='object')
```

In [275]:

```
# get the missing values in percentage

missing_values = (df.isna().sum()).to_frame().sort_values(by=0, ascending=
print('Total missing values:' ,missing_values.sum()[0])
missing_values
```

```
Total missing values: 60762
```

Out[275]:

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

	0
seasonal_vaccine	0

In [209]:

```
#check the categorical columns
cat_col = df.select_dtypes(include = 'object')
cat_col
```

Out[209]:

	age_group	education	race	sex	income_poverty	marital_status	rer
respondent_id							
0	55 - 64 Years	< 12 Years	White	Female	Below Poverty	Not Married	
1	35 - 44 Years	12 Years	White	Male	Below Poverty	Not Married	
2	18 - 34 Years	College Graduate	White	Male	<= $75,000, Above Poverty	Not Married	
3	65+ Years	12 Years	White	Female	Below Poverty	Not Married	
4	45 - 54 Years	Some College	White	Female	<= $75,000, Above Poverty	Married	
...	...	...	...	...	...	...	
26702	65+ Years	Some College	White	Female	<= $75,000, Above Poverty	Not Married	
26703	18 - 34 Years	College Graduate	White	Male	<= $75,000, Above Poverty	Not Married	
26704	55 - 64 Years	Some College	White	Female	NaN	Not Married	
26705	18 - 34 Years	Some College	Hispanic	Female	<= $75,000, Above Poverty	Married	
26706	65+ Years	Some College	White	Male	<= $75,000, Above Poverty	Married	

26707 rows × 12 columns

In [210]:

```
# check for duplicated values

df.duplicated().sum()
```

Out[210]: 0

In [211]:
```
# Check the numerical columns

num_col = df.select_dtypes(exclude='object')
num_col
```

Out[211]:

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

26707 rows × 25 columns

Key Observations

- There are no duplicates in our data.
- There are `60,762` missing values from both numerical and categorical data.
- The columns `hhs_geo_region` , `employment_industry` , and `employment_occupation` are encoded with random strings; they may need to be changed to numbers for readability in order to anonymize the data.

DATA CLEANING

In [212]:

```
# handle missing numerical values
# instantiate imputer
import numpy as np
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values=np.nan, strategy='median')

num_col.iloc[:,:] = imputer.fit_transform(num_col)
```

In [213]:

```
num_col.isna().sum()
```

Out[213]:

```
h1n1_concern                   0
h1n1_knowledge                 0
behavioral_antiviral_meds      0
behavioral_avoidance           0
behavioral_face_mask           0
behavioral_wash_hands          0
behavioral_large_gatherings    0
behavioral_outside_home        0
behavioral_touch_face          0
doctor_recc_h1n1               0
doctor_recc_seasonal           0
chronic_med_condition          0
child_under_6_months           0
health_worker                  0
health_insurance               0
opinion_h1n1_vacc_effective    0
opinion_h1n1_risk              0
opinion_h1n1_sick_from_vacc    0
opinion_seas_vacc_effective    0
opinion_seas_risk              0
opinion_seas_sick_from_vacc    0
household_adults               0
household_children             0
h1n1_vaccine                   0
seasonal_vaccine               0
dtype: int64
```

In [214]:

```
# handle missing categorical values
# instantiate imputer

imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

cat_col.iloc[:,:] = imputer.fit_transform(cat_col)
```

In [215]:

```
cat_col.isna().sum()
```

Out[215]:

```
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
employment_industry      0
employment_occupation    0
dtype: int64
```

DATA TRANSFORMATION

In [216]:
```
from sklearn.preprocessing import OrdinalEncoder

# create an instance of the OrdinalEncoder
encoder = OrdinalEncoder(categories='auto')

#fit the encoder on the categorical data
cat_encoded = encoder.fit_transform(cat_col)
cat_encoded_df = pd.DataFrame(cat_encoded)

cat_encoded_df.columns = cat_col.columns
cat_encoded_df
```

Out[216]:

	age_group	education	race	sex	income_poverty	marital_status	rent_or_own	emp
0	3.0	1.0	3.0	0.0	2.0	1.0	0.0	
1	1.0	0.0	3.0	1.0	2.0	1.0	1.0	
2	0.0	2.0	3.0	1.0	0.0	1.0	0.0	
3	4.0	0.0	3.0	0.0	2.0	1.0	1.0	
4	2.0	3.0	3.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
26702	4.0	3.0	3.0	0.0	0.0	1.0	0.0	
26703	0.0	2.0	3.0	1.0	0.0	1.0	1.0	
26704	3.0	3.0	3.0	0.0	0.0	1.0	0.0	
26705	0.0	3.0	1.0	0.0	0.0	0.0	1.0	
26706	4.0	3.0	3.0	1.0	0.0	0.0	0.0	

26707 rows × 12 columns

In [217]:
```
data_df = pd.concat([num_col, cat_encoded_df], axis=1)
data_df
```

Out[217]:

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

26707 rows × 37 columns

Key Observations

- I used the `simple Imputer` from the `sklearn library` to handle the missing values in both numerical and categorical variables.
- Went ahead and imported the `Ordinal Encoder` from the `sklearn library` to encode and transform the categorical variables to numerical.
- Since this will preserve the general distribution of categorical data, each split is handled differently using the numerical columns we used most frequently.
- Since the majority of our values are repeated classes, we chose the categorical columns with the highest frequency as our approach.
- Joined the two variables together using `concantenation` to come up with a clean dataset.

Exploratory Data Analysis(EDA)

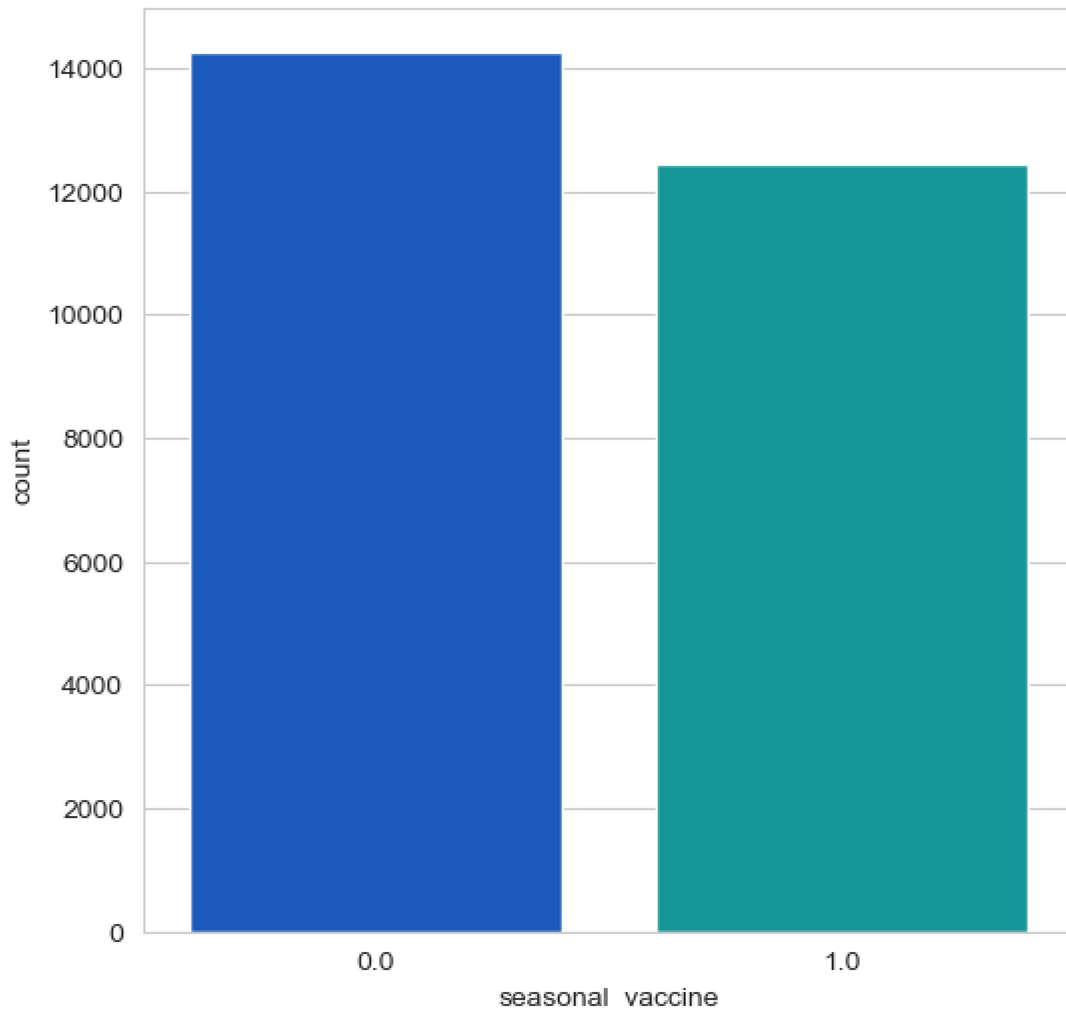
In [218]:

```
# import the relevant libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from plotly.subplots import make_subplots
import plotly.graph_objects as go

from sklearn.model_selection import train_test_split,cross_val_score,GridS
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,roc_auc_score,ConfusionMatrixDi
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.feature_selection import RFECV
from sklearn.ensemble import GradientBoostingClassifier
import joblib
```

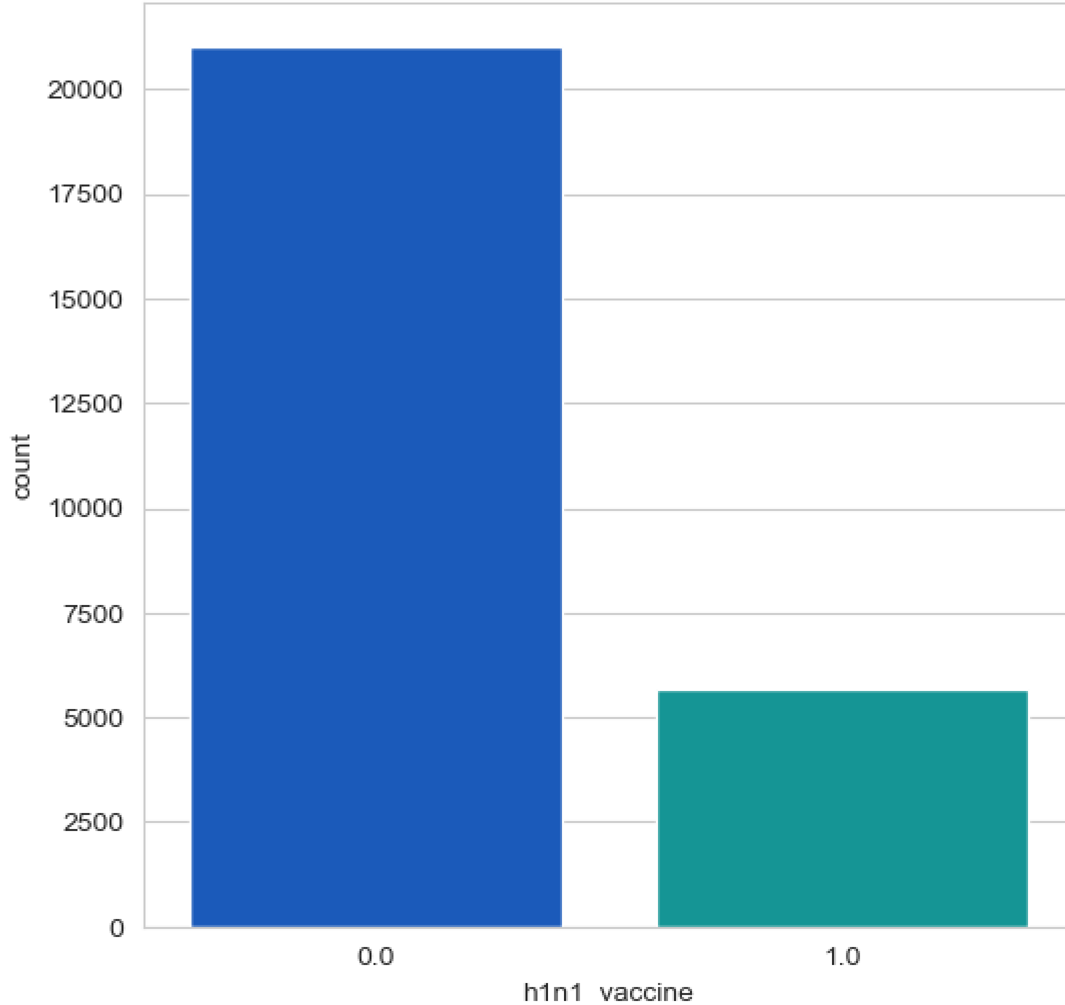
Graph depicting the balance / imbalance of the size distributions of h1n1_vaccine and seasonal_vaccine

In [219]:

```
def histo_plot(column):
    sns.set_style("whitegrid")
    fig,axes = plt.subplots(figsize = (6,6))
    sns.countplot(column, palette="winter")
histo_plot(data_df.seasonal_vaccine)
```



In [220]:
```
histo_plot(data_df.h1n1_vaccine)
```



In [221]:
```
data_df.seasonal_vaccine.value_counts()
```

Out[221]:
```
0.0    14272
1.0    12435
Name: seasonal_vaccine, dtype: int64
```

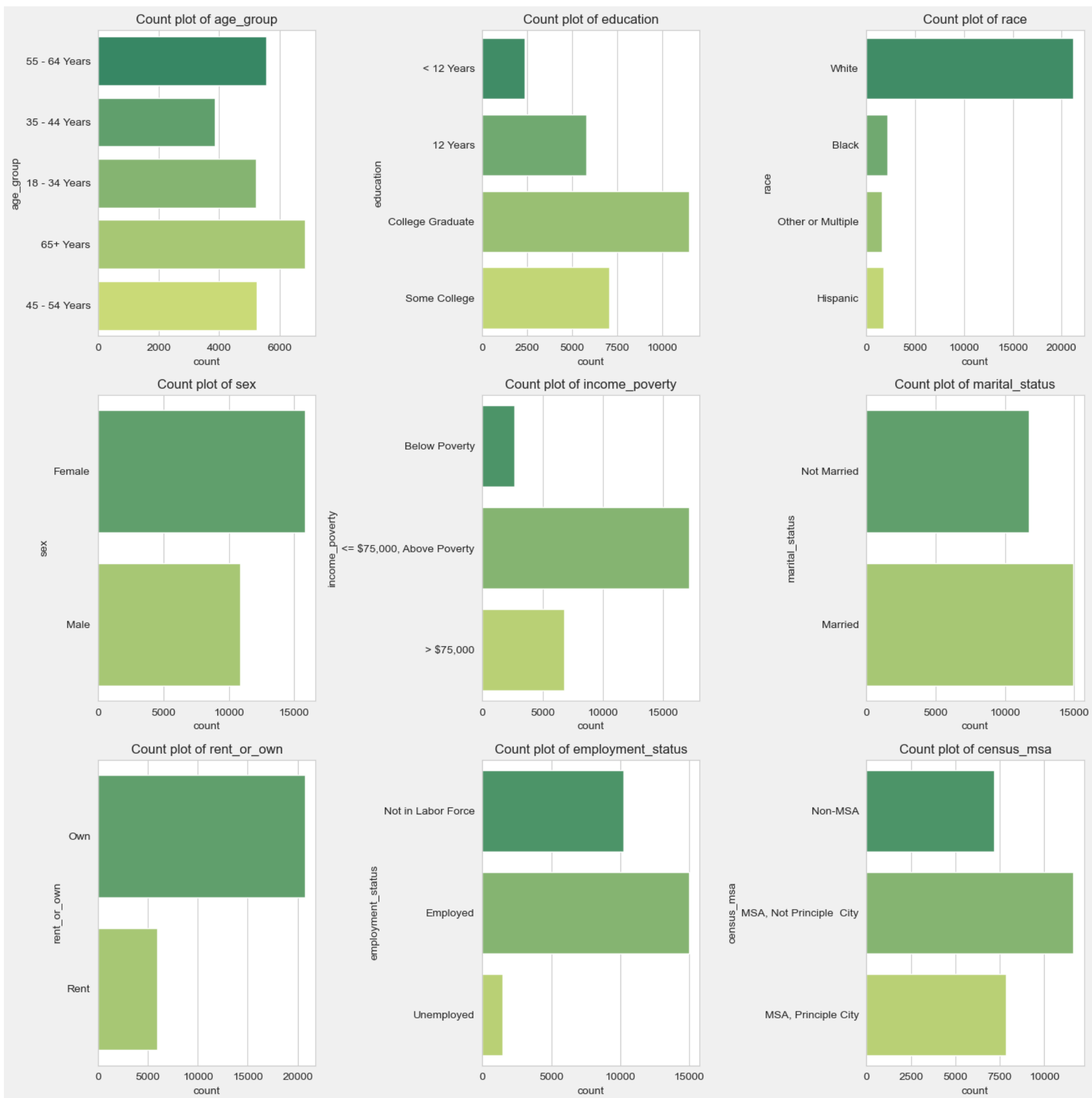
In [222]:
```
data_df.h1n1_vaccine.value_counts()
```

Out[222]:
```
0.0    21033
1.0     5674
Name: h1n1_vaccine, dtype: int64
```

I decided to go with `'seasonal_vaccine'` as my target variable since it is less imbalanced.

Univariate Analysis

In [223]:
```
cat_col = cat_col.drop(['employment_industry', 'employment_occupation', 'h

count_plot = cat_col.columns.to_list()

fig, axes = plt.subplots(nrows=3 , ncols=3, figsize=(14, 14))
fig.patch.set_facecolor('#F0F0F0')
# Iterate over the columns and corresponding axes
for column, ax in zip(count_plot, axes.flatten()):
    sns.countplot(data=cat_col, y=column, ax=ax, palette="summer")
    ax.set_title(f'Count plot of {column}')

plt.tight_layout()
plt.show()
```



Key Observations

- The majority of responders are above the age of 65, while the difference between the categories is fairly small.
- The majority of respondents are college graduates.
- White people account for roughly 80% of the dataset.
- Almost 60% of those polled are female.
- Nearly 60% of respondents earn between the poverty level and around $75,000 per year.
- The majority of respondents are employed.
- Majority of respondents own their own houses.
- most of the respondents are from the MSA,Not Principle City.
- Almost 60% of respondents are married.

Bivariate Analysis

In [277]:

```
# Create subplots with titles
fig = make_subplots(rows=3, cols=3, subplot_titles=('Race', 'Sex', 'Educat

# Define variables and titles
variables = ['race', 'sex', 'education', 'age_group', 'marital_status', '
titles = ['Race', 'Sex', 'Education', 'Age Group', 'Marital Status', 'Chro

# Loop through the variables and add violin plots to the subplots
for i, var in enumerate(variables):
    row = (i // 3) + 1
    col = (i % 3) + 1
    fig.add_trace(go.Violin(x=df[var], y=df['seasonal_vaccine'], name=titl

# Update the layout
fig.update_layout(height=1000, width=900, title_text="Violin Subplots", te
```

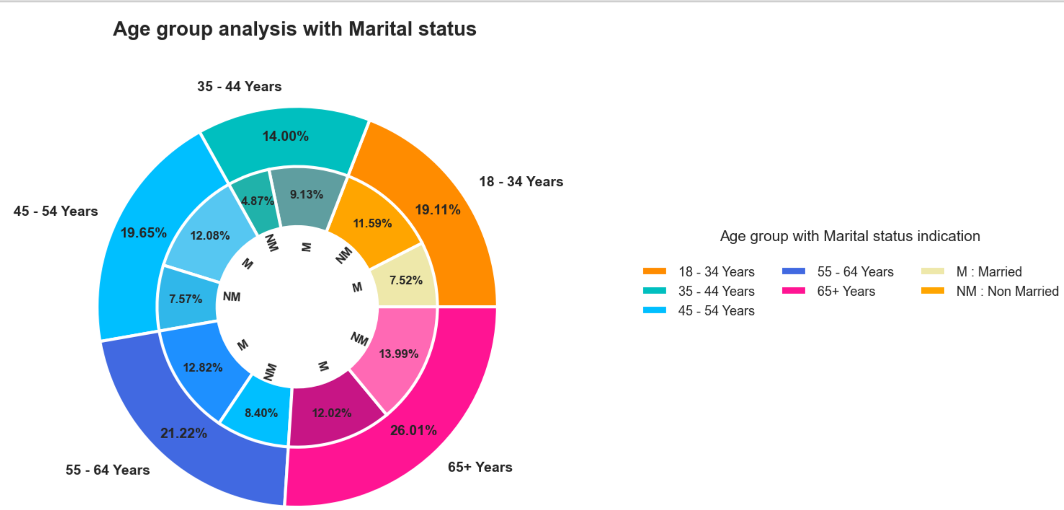
Key Observations

- `Health insurance` - it appears that people without health insurance did not receive the seasonal vaccine in big numbers when compared to those with insurance who were evenly distributed.
- `Health workers` - In comparison to non-health employees, the majority of health workers received the seasonal vaccine.
- `Marital status` - The majority of the unmarried population received the immunizations, but it is evenly distributed in the married class.
- `Chronic med condition` - Those taking chronic medicine had a greater intake of the vaccine than none, while those without chronic illnesses had a higher rate of not taking the vaccine.

Multivariate Analysis

In [225]:

```
plt.figure(figsize=(12,8))
plt.title("Age group analysis with Marital status\n\n",fontsize=20,fontwei
plt.pie([4835,3543,4971,5369,6581], radius=1,
        colors=['darkorange', 'c','deepskyblue','royalblue',"deeppink"],
        labels=['18 - 34 Years','35 - 44 Years','45 - 54 Years','55 - 64 Y
        autopct='%.2f%%',
        pctdistance=0.85, textprops = {"fontsize":14,"fontweight":"bold"},
        wedgeprops=dict(width=0.6, edgecolor='white',linewidth=3))

plt.pie([1902,2933,2311,1232,3057,1914,3243,2126,3042,3539], radius=0.7,
        colors=['palegoldenrod','orange', 'cadetblue', 'lightseagreen',
                '#56C7F2','#30B7EA','dodgerblue', 'deepskyblue',"mediumvic
        wedgeprops=dict(width=0.3, edgecolor='w',linewidth=3),textprops =
         labels=['M','NM','M','NM','M','NM','M',
                 'NM','M','NM'],autopct='%.2f%%',
        pctdistance=0.8, labeldistance=0.4,rotatelabels= True)

legend = plt.legend(bbox_to_anchor=(1, 0.7),
          labels=['18 - 34 Years','35 - 44 Years','45 - 54 Years','55 - 64
                  "M : Married","NM : Non Married"],
          title = "Age group with Marital status indication\n",
          ncol=3,
          fontsize=13)
legend.set_title("Age group with Marital status indication\n",prop={"size'
legend.draw_frame(False)

# Set the desired background color
background_color = 'white'  # Replace with your desired color

# Set the style with the desired background color
sns.set_style("whitegrid", {'axes.facecolor': background_color})

plt.axis('equal')
plt.show()
```



Key Observations

- In the pie chart above I was just trying to visualize and see the relation between different age groups and marital status in percentage.
- In overall `65 years` and above have the highest percentage in receiving vaccination compared to other age-groups.
- Respondents that are `65 years` and not married appear to have a fairly higher percentage of receiving the vaccine compared to married respondents.
- Respondents that are `55-64 years` and married appear to have a higher percentage of receiving the vaccine compared to non-married respondents.
- Respondents that are `45-54 years` and married appear to have a higher percentage of receiving the vaccine compared to non-married respondents.
- Respondents that are `35-44 years` and married appear to have a higher percentage of receiving the vaccine compared to non-married respondents.
- Respondents that are `18-34 years` and not married appear to have a higher percentage of receiving the vaccine compared to married respondents.

Feature Selection

In [226]:

```
# Importing necessary modules for feature selection
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# Set the Target and Independent variable

X = data_df.drop(['h1n1_vaccine', 'seasonal_vaccine'], axis=1)
y = data_df.seasonal_vaccine

# select K best to extract best features
best_features = SelectKBest(chi2,k=15)
fit = best_features.fit(X, y)

scores = pd.DataFrame(fit.scores_)
x_columns= pd.DataFrame(X.columns)

data_df_score = pd.concat([x_columns,scores], axis=1)
data_df_score.columns =['data_df1','scores']
data_df_score
```

Out[226]:

	data_df1	scores
0	h1n1_concern	325.148276
1	h1n1_knowledge	115.652237
2	behavioral_antiviral_meds	0.918545
3	behavioral_avoidance	42.141343
4	behavioral_face_mask	62.214095
5	behavioral_wash_hands	58.594846
6	behavioral_large_gatherings	69.677957
7	behavioral_outside_home	50.332544
8	behavioral_touch_face	123.369866
9	doctor_recc_h1n1	839.733852
10	doctor_recc_seasonal	2421.579654
11	chronic_med_condition	557.625394
12	child_under_6_months	4.427147
13	health_worker	383.862558
14	health_insurance	27.094311
15	opinion_h1n1_vacc_effective	286.280388
16	opinion_h1n1_risk	866.169464
17	opinion_h1n1_sick_from_vacc	16.260163
18	opinion_seas_vacc_effective	991.273021
19	opinion_seas_risk	2794.888237
20	opinion_seas_sick_from_vacc	80.551802
21	household_adults	71.806568
22	household_children	538.442999
23	age_group	1997.217625
24	education	5.762772
25	race	91.674338
26	sex	94.243768
27	income_poverty	38.808131
28	marital_status	26.849772
29	rent_or_own	215.311327
30	employment_status	67.380193
31	hhs_geo_region	14.872176
32	census_msa	7.253763
33	employment_industry	1669.849661
34	employment_occupation	7.185226

In [227]:
```
# Preview top 15 scores
print(data_df_score.nlargest(15, 'scores'))
```

```
                     data_df1       scores
19           opinion_seas_risk  2794.888237
10         doctor_recc_seasonal  2421.579654
23                   age_group  1997.217625
33         employment_industry  1669.849661
18  opinion_seas_vacc_effective   991.273021
16           opinion_h1n1_risk   866.169464
9             doctor_recc_h1n1   839.733852
11        chronic_med_condition   557.625394
22          household_children   538.442999
13               health_worker   383.862558
0                 h1n1_concern   325.148276
15  opinion_h1n1_vacc_effective   286.280388
29                 rent_or_own   215.311327
8         behavioral_touch_face   123.369866
1               h1n1_knowledge   115.652237
```

Checking for Multicollinearity

In [228]:
```
# check for correlation
corr = data_df.corr()['seasonal_vaccine'].sort_values(ascending = False)
corr_data = corr[(corr > 0.1)]
corr_data
```

Out[228]:
```
seasonal_vaccine               1.000000
opinion_seas_risk              0.386916
h1n1_vaccine                   0.377143
doctor_recc_seasonal           0.360696
opinion_seas_vacc_effective    0.358869
age_group                      0.277454
opinion_h1n1_risk              0.215650
opinion_h1n1_vacc_effective    0.203187
doctor_recc_h1n1               0.198560
chronic_med_condition          0.169465
h1n1_concern                   0.154488
health_worker                  0.126977
health_insurance               0.124929
behavioral_touch_face          0.119925
h1n1_knowledge                 0.119779
behavioral_wash_hands          0.112254
race                           0.101743
Name: seasonal_vaccine, dtype: float64
```

In [229]:
```
# subsetting the final dataframe to desired columns
final_data_df=data_df.loc[:,['opinion_seas_risk','doctor_recc_seasonal','a
              'household_children','health_worker','h1n1_concern','opinion
final_data_df.head()
```

Out[229]:

	opinion_seas_risk	doctor_recc_seasonal	age_group	opinion_seas_vacc_effective	opinio
0	1.0	0.0	3.0	2.0	
1	2.0	0.0	1.0	4.0	
2	1.0	0.0	0.0	4.0	
3	4.0	1.0	4.0	5.0	
4	1.0	0.0	2.0	3.0	

In [230]:
```
# check the shape of the subset
final_data_df.shape
```

Out[230]: (26707, 14)

Key Observations

- I used `Kbest` from the `sklearn library` to extract the best features.
- I checked the features that scored the best.
- Did some correlations of the features with our target variable `seasonal_vaccine` in percentage.
- I hard coded a subset of features with high correlations and best score from the data and assigned them a new variable name `final_data_df` for modeling.

Train Test Split

In [231]:
```
# split the data into train and test sets
# I have assigned 75% of the original data on the training set and 25% on
# create a copy of the data set

model = final_data_df.copy()

# Define X and y
X = model.drop('seasonal_vaccine', axis=1)
y = model.seasonal_vaccine

# set the random seed to be 0
random_seed = 0

#split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
```

MODEL BUILDING

- We are going to define a few functions that will help us with model selection
- We start with our base model which in this case we will be using `DecisionTreeClassifier`

Base Model

DecisionTree

- The function below will aid to plot a learning curve to evaluate the performance on training and validation sets

In [283]:

```
from sklearn.model_selection import learning_curve

def plot_learning_curve(model):

    # Plotting the learning curve

    # Generate the learning curve using the learning_curve function

    train_sizes, train_scores, test_scores = learning_curve(
        model, X_train, y_train, scoring='accuracy')

    # Calculate the mean and standard deviation of the training scores

    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)

    # Calculate the mean and standard deviation of the validation scores

    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    # Plot the training scores and fill the area between the upper and low

    plt.figure(figsize=(10, 8))
    plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training sco
    plt.fill_between(
        train_sizes, train_mean - train_std, train_mean + train_std, alpha

    # Plot the validation scores and fill the area between the upper and l

    plt.plot(train_sizes, test_mean, 'o-', color='g', label='Validation sc
    plt.fill_between(
        train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.1


    # Set the x-axis label


    plt.xlabel('Training examples')

    # Set the y-axis label

    plt.ylabel('Score')

    # Add a legend to the plot

    plt.legend(loc='best')

    # Add a grid to the plot

    plt.grid(True)
    plt.show()
```

Create a Pipeline

In [232]:
```
# import the necessary libraries
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import log_loss

# setting my pipeline

pipeline = Pipeline([('scaler' , StandardScaler()),
                            ('tree', DecisionTreeClassifier(random_state=

# setting basic parameter
grid = [{'tree__criterion': ['entropy'],
         }]
```

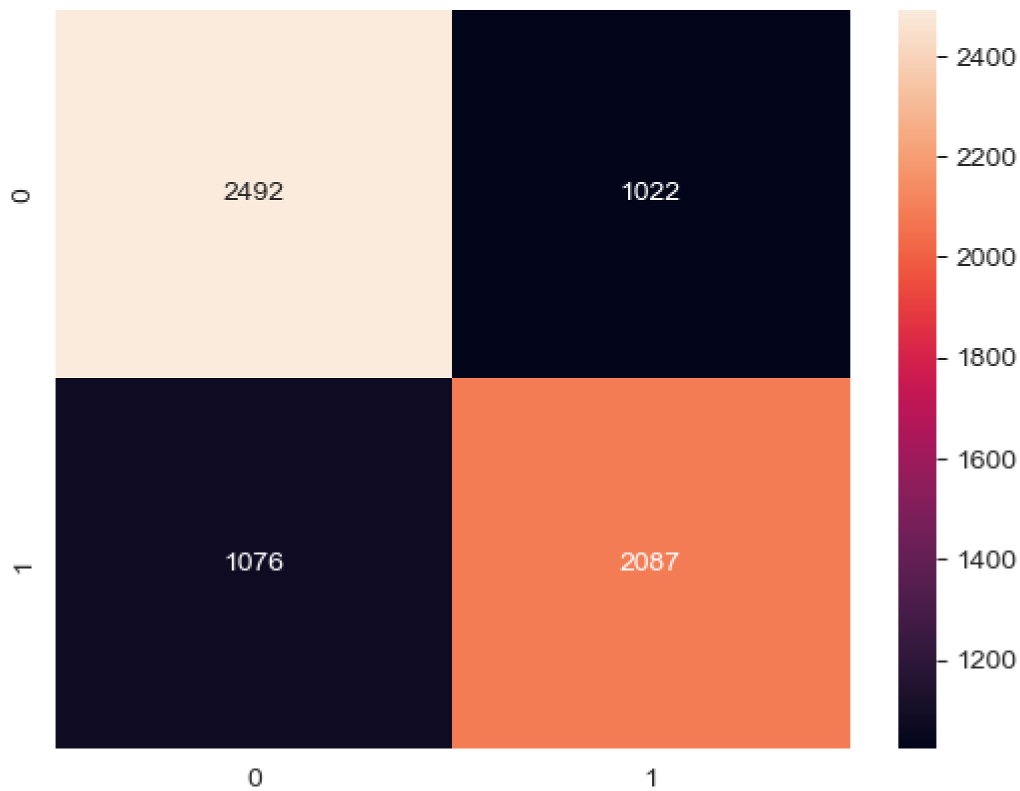
In [233]:
```
gridsearch = GridSearchCV(estimator=pipeline,
                          param_grid= grid,
                          scoring='accuracy',
                          cv=5)
```

In [234]:

```
def model(pipeline):
    # fit the model
    pipeline.fit(X_train, y_train)

    # predict the train and test set
    y_train_pred = pipeline.predict(X_train)
    y_test_pred = pipeline.predict(X_test)

    # test the accuracy
    acc_train = accuracy_score(y_train_pred, y_train)
    acc_test = accuracy_score(y_test_pred, y_test)

    # Print the scores
    print(f'The Model Train accuracy is: {acc_train:.3f}')
    print(f'The Model Test accuracy is: {acc_test:.3f}')
    print('\n')
    print('----------------------------------------------------------------

    # print the report
    print(classification_report(y_test,y_test_pred))

model(gridsearch)
```

```
The Model Train accuracy is: 0.916
The Model Test accuracy is: 0.707


----------------------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.72      0.75      0.73      3568
         1.0       0.69      0.66      0.68      3109

    accuracy                           0.71      6677
   macro avg       0.71      0.70      0.70      6677
weighted avg       0.71      0.71      0.71      6677
```

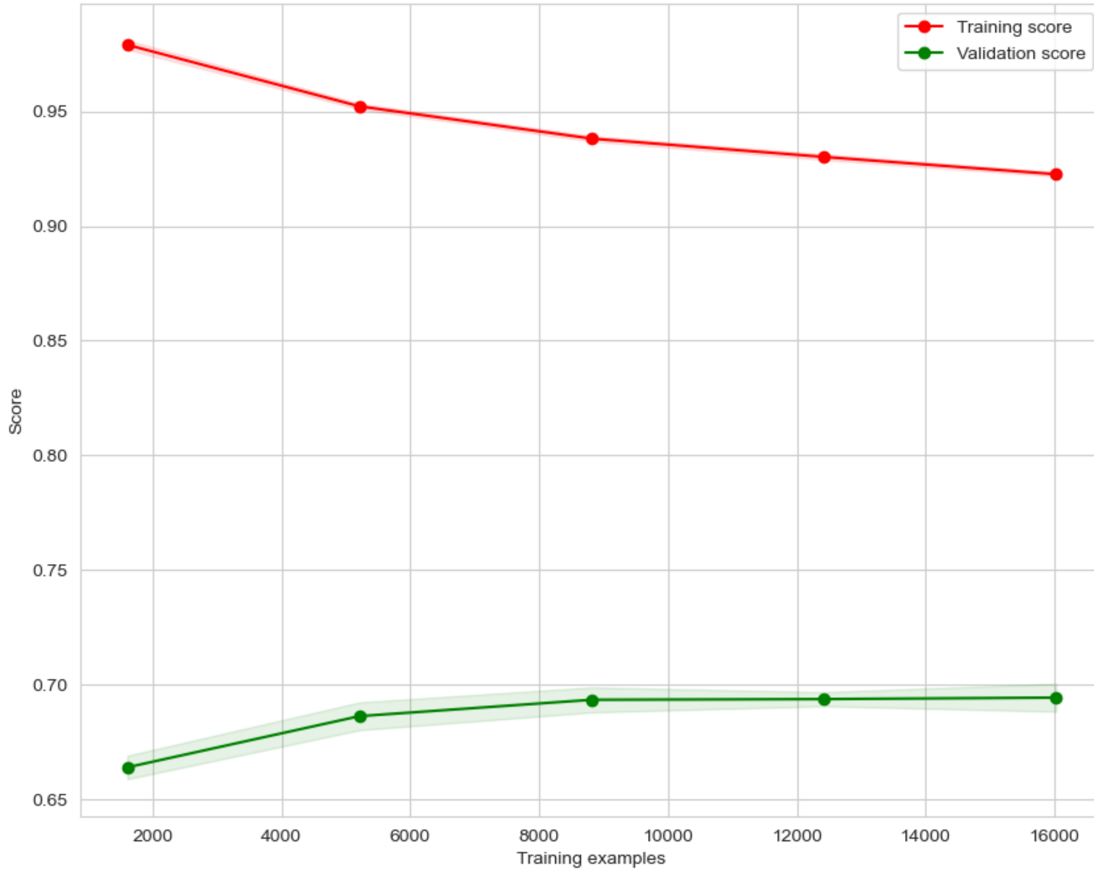
In [235]:
```
# creating confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_test_pred)
# Making the Confusion Matrix
cm = confusion_matrix(y_test_pred, y_test)
sns.heatmap(cm,annot=True, fmt='g')
plt.savefig('confusion.png')
```



In [289]:
```
#performing cross validation
cross_val = cross_val_score(pipeline,X_train,y_train,cv=10)
print(f"Cross Validation Accuracy: {round(cross_val.mean()*100,4)}%")
```

```
Cross Validation Accuracy: 73.7244%
```

In [284]:
```
# plotting the learning curve

plot_learning_curve(gridsearch)
```



Key Observations

- The model shows good performance on the training dataset, achieving an accuracy of `91.6%` .
- However, when applied to the test dataset and unseen data, the accuracy drops to `70.7%` which significantly means our base model is overfitting.
- The cross-validation accuracy provides further evidence of the model's overall performance, indicating that it performs consistently at around `73.72%` accuracy across different subsets of the data.
- The learning curve and accuracy measures show that the model is overfitting, as evidenced by the reduction in training score and reduced cross-validation accuracy.

Random Forest

In [278]:

```
from sklearn.ensemble import RandomForestClassifier

X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size= (


pipeline1 = Pipeline([('scaler' , StandardScaler()),
                            ('tree1', RandomForestClassifier())])

# setting basic parameter
grid = [{'tree1__criterion': ['entropy','gini'],
         'tree1__n_estimators': [len(range(100))],
         'tree1__max_depth':[2,3,4,5]}]

gridsearch1 = GridSearchCV(estimator=pipeline1,
                           param_grid= grid,
                           scoring='accuracy',
                           cv=5)

model(gridsearch1)
```

```
The Model Train accuracy is: 0.765
The Model Test accuracy is: 0.767


----------------------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.77      0.81      0.79      3568
         1.0       0.77      0.72      0.74      3109

    accuracy                           0.77      6677
   macro avg       0.77      0.76      0.76      6677
weighted avg       0.77      0.77      0.77      6677
```

In [237]:

```
print(gridsearch1.best_params_)
```

```
{'tree1__criterion': 'gini', 'tree1__max_depth': 5, 'tree1__n_estimator
s': 100}
```

Evaluate the model using the best parameters

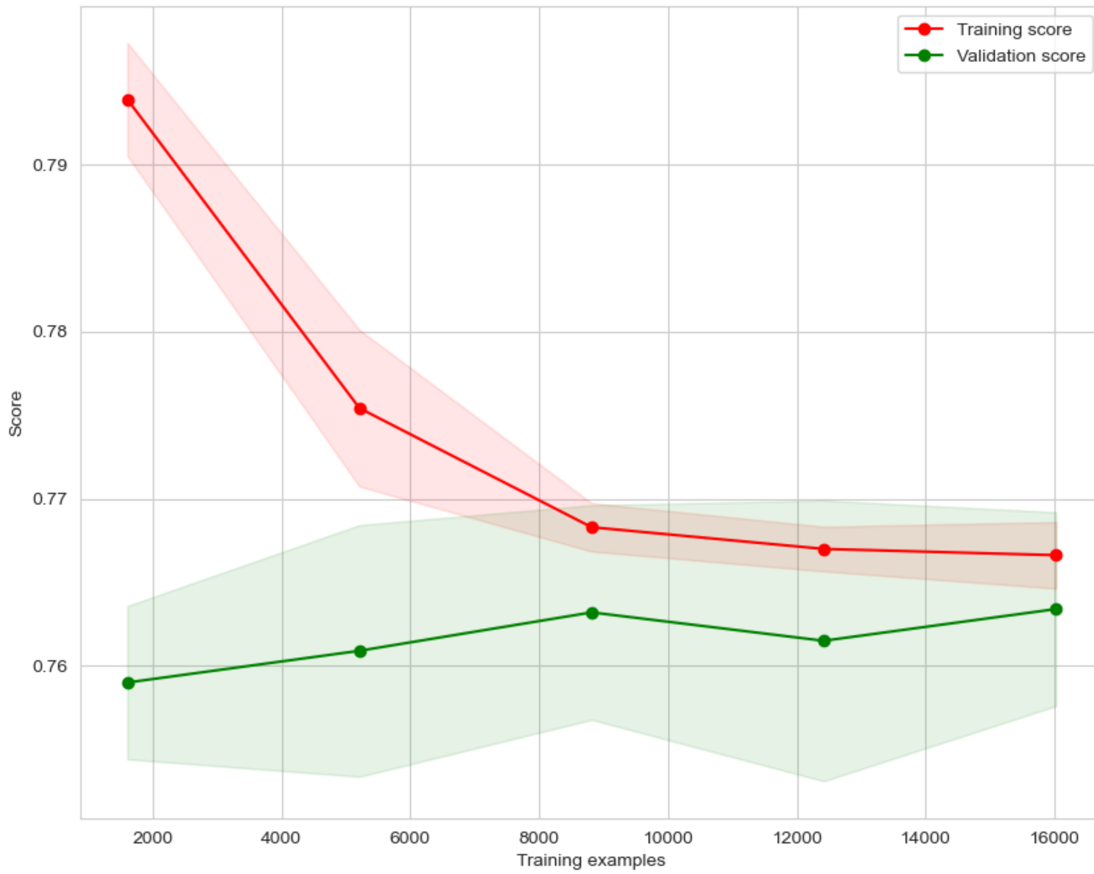
In [279]:
```
grid = [{'tree1__criterion': ['gini'],
         'tree1__n_estimators': [100],
         'tree1__max_depth':[5]}]

gridsearch1 = GridSearchCV(estimator=pipeline1,
                           param_grid= grid,
                           scoring='accuracy',
                           cv=5)

model(gridsearch1)
```

```
The Model Train accuracy is: 0.764
The Model Test accuracy is: 0.769


----------------------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.77      0.81      0.79      3568
         1.0       0.77      0.72      0.74      3109

    accuracy                           0.77      6677
   macro avg       0.77      0.77      0.77      6677
weighted avg       0.77      0.77      0.77      6677
```

In [288]:
```
#performing cross validation
cross_val = cross_val_score(pipeline1,X_train1,y_train1,cv=10)
print(f"Cross Validation Accuracy: {round(cross_val.mean()*100,4)}%")
```

```
Cross Validation Accuracy: 73.5034%
```

In [285]:
```
# plotting the learning curve

plot_learning_curve(gridsearch1)
```



Key Observations

- The model has a training accuracy of `76.4%` and a test accuracy of `76.9%` and a cross validation accuracy of `73.50%` .
- The learning curve shows that generalization is improving initially, with the test score increasing and the training score falling.
- The test score, however, gradually plateaus and begins to drop, indicating limits in catching complicated patterns and probable overfitting.
- The temporary dip suggests that there may be certain instances in the validation set where the model struggles to make accurate predictions.
- The following increase in the validation score suggests that the model adjusts and improves its performance in such difficult situations as well.
- To achieve the requisite levels of accuracy, the model must be refined through hyperparameter tuning.

Gradient Boosting

In [243]:
```
# instantiate gradient boosting
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size= 0


pipeline2 = Pipeline([('scaler' , StandardScaler()),
                    ('gradient', GradientBoostingClassifier())])

# setting basic parameter
grid2 = [{'gradient__loss' : ['log_loss', 'deviance'],
         'gradient__learning_rate': [ 0.2, 0.3, 0.4,],
         'gradient__n_estimators': [len(range(120))]}]


gridsearch2 = GridSearchCV(estimator=pipeline2,
                           param_grid= grid2,
                           scoring='accuracy',
                           cv=5)

model(gridsearch2)
```

```
The Model Train accuracy is: 0.778
The Model Test accuracy is: 0.770


----------------------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.78      0.80      0.79      3568
         1.0       0.76      0.73      0.75      3109

    accuracy                           0.77      6677
   macro avg       0.77      0.77      0.77      6677
weighted avg       0.77      0.77      0.77      6677
```

In [253]:
```
print(gridsearch2.best_params_)
```

```
{'gradient__learning_rate': 0.2, 'gradient__loss': 'deviance', 'gradient
__n_estimators': 120}
```

Evaluate the model using the best parameters

In [252]:

```
# setting basic parameter
grid2 = [{'gradient__loss' : [ 'deviance'],
         'gradient__learning_rate': [ 0.2],
         'gradient__n_estimators': [120]}]


gridsearch2 = GridSearchCV(estimator=pipeline2,
                           param_grid= grid2,
                           scoring='accuracy',
                           cv=5)

model(gridsearch2)
```

```
The Model Train accuracy is: 0.778
The Model Test accuracy is: 0.770


----------------------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.78      0.80      0.79      3568
         1.0       0.76      0.73      0.75      3109

    accuracy                           0.77      6677
   macro avg       0.77      0.77      0.77      6677
weighted avg       0.77      0.77      0.77      6677
```

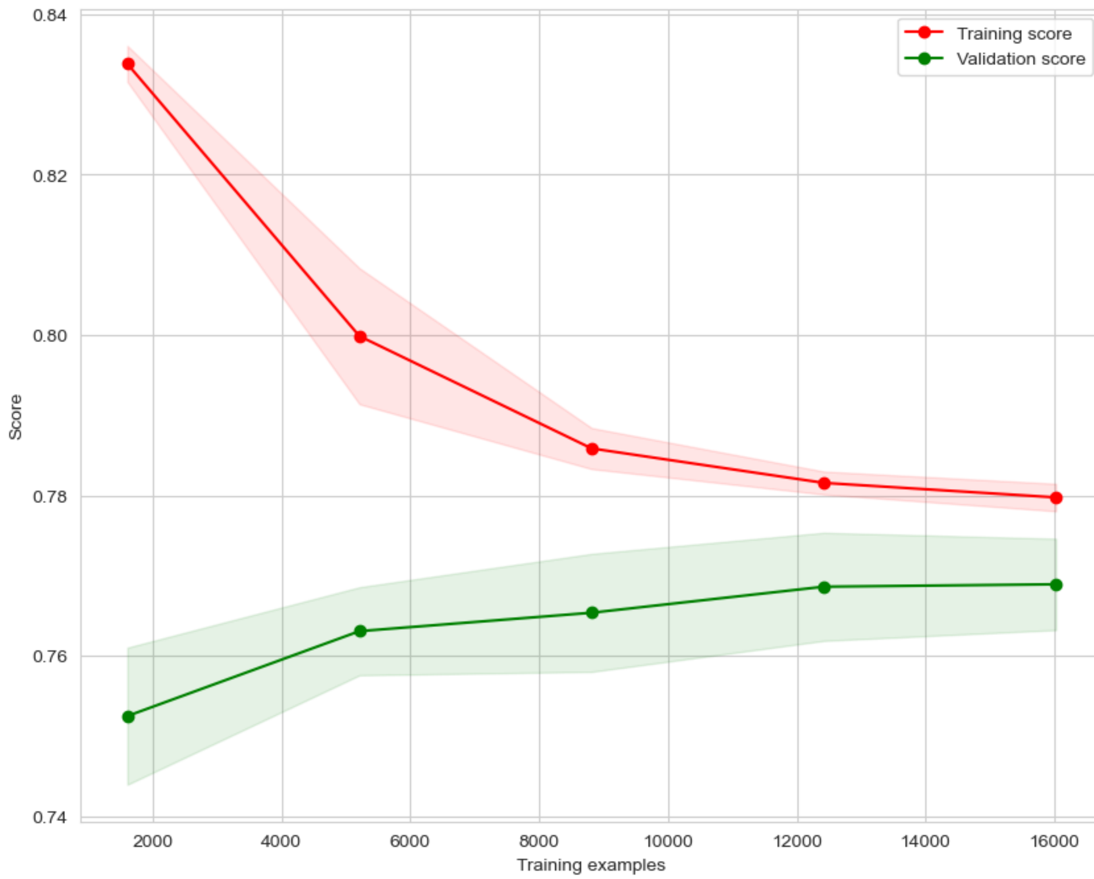
In [287]:

```
#performing cross validation
cross_val = cross_val_score(pipeline2,X_train2,y_train2,cv=10)
print(f"Cross Validation Accuracy: {round(cross_val.mean()*100,4)}%")
```

```
Cross Validation Accuracy: 76.953%
```

In [286]:

```
# plotting the learning curve

plot_learning_curve(gridsearch2)
```



Key Observations

- The model has a training accuracy of `77.8%` and a test accuracy of `77%` and a cross validation accuracy of `76.95%` .
- The learning curve shows that generalization is improving, with the test score increasing and the training score falling.
- These findings show that the model performed well in terms of accuracy on both the training and test datasets, with consistent performance evaluated by cross-validation.
- The rising validation score indicates that the model generalizes well to unseen data, as it consistently improves its predictive accuracy.

AdaBoost

In [241]:
```
from sklearn.ensemble import AdaBoostClassifier

X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size= (

pipeline3 = Pipeline([('scaler' , StandardScaler()),
                      ('ADA', AdaBoostClassifier())])

cv= 7
# setting basic parameter
ada = [{'ADA__learning_rate': [0.1, 0.2, 0.3, 0.4,],
        'ADA__n_estimators': [len(range(50))]}]

adaboost = GridSearchCV(estimator=pipeline3,
                        param_grid= ada,
                        scoring='accuracy',
                        cv=cv)

model(adaboost)
```

```
The Model Train accuracy is: 0.766
The Model Test accuracy is: 0.769


----------------------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.77      0.81      0.79      3568
         1.0       0.77      0.72      0.74      3109

    accuracy                           0.77      6677
   macro avg       0.77      0.77      0.77      6677
weighted avg       0.77      0.77      0.77      6677
```

In [254]:
```
print(adaboost.best_params_)
```

```
{'ADA__learning_rate': 0.4, 'ADA__n_estimators': 50}
```

Evaluate the model using the best parameters

In [255]:
```
# setting basic parameter
ada = [{'ADA__learning_rate': [ 0.4],
         'ADA__n_estimators': [50]}]

adaboost = GridSearchCV(estimator=pipeline3,
                          param_grid= ada,
                          scoring='accuracy',
                          cv=cv)

model(adaboost)
```

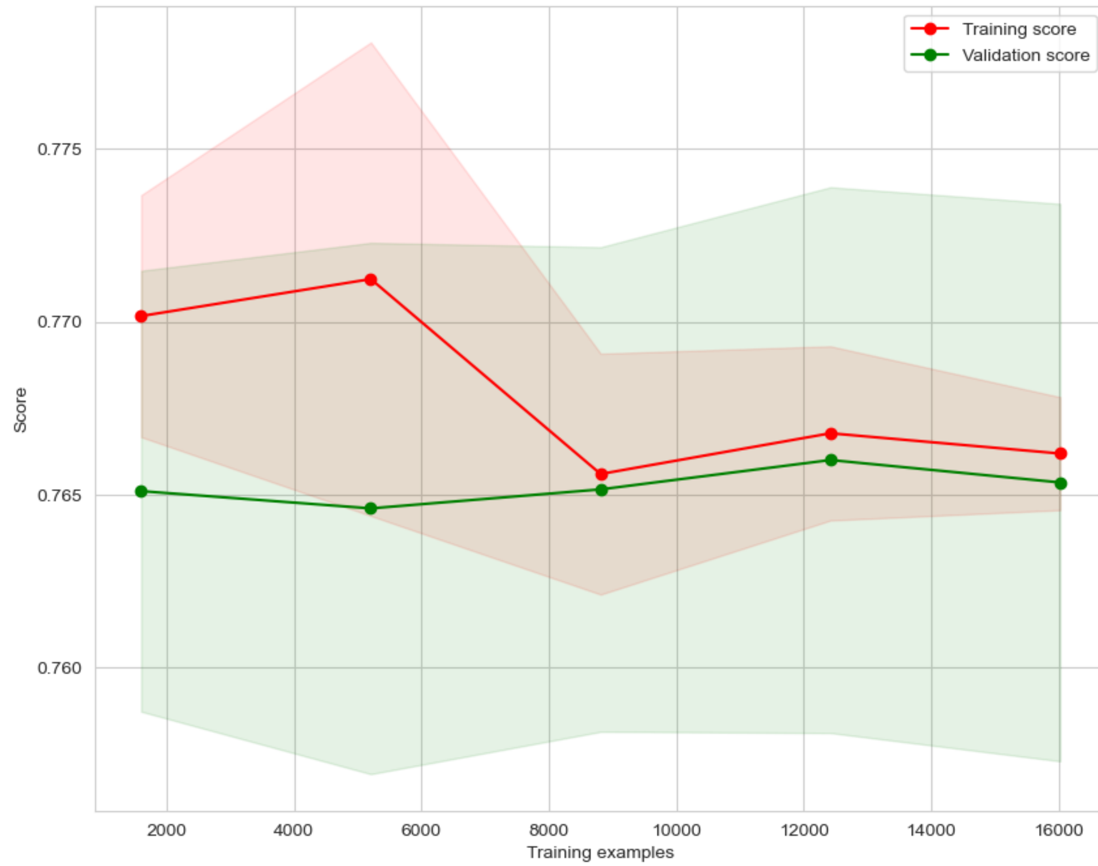
```
The Model Train accuracy is: 0.766
The Model Test accuracy is: 0.769


----------------------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.77      0.81      0.79      3568
         1.0       0.77      0.72      0.74      3109

    accuracy                           0.77      6677
   macro avg       0.77      0.77      0.77      6677
weighted avg       0.77      0.77      0.77      6677
```

In [257]:
```
#performing cross validation
cross_val = cross_val_score(pipeline3,X_train3,y_train3,cv=10)
print(f"Cross Validation Accuracy: {round(cross_val.mean()*100,4)}%")
```

```
Cross Validation Accuracy: 76.5751%
```

In [291]:

```
# plotting the learning curve

plot_learning_curve(adaboost)
```



Key Observations

- The model has a training accuracy of `76.6%` and a test accuracy of `76.9%` and a cross validation accuracy of `76.575%` .
- The learning curve shows that the training score starts with a high note suggesting overfitting however, it suddenly declines.
- This indicates that the model initially struggles to fit the training data and may not capture all of the patterns available in the data.
- However, as the model receives more training data and learns from it, the training score gradually improves.
- Both the train and test set reach a point where the model start to drop gradually.

XGBoosting

In [242]:
```
import xgboost as xgb

xgb.XGBClassifier()

# splitting the dataset
X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size= 0


pipeline4 = Pipeline([('scaler' , StandardScaler()),
                            ('xgb', xgb.XGBClassifier())])

cv= 5
# setting basic parameters
xgb = [{'xgb__eta': [0.1, 0.2, 0.3, 0.4,],
         'xgb__gamma': [len(range(1,50))],
         'xgb__max_depth': [len(range(1,10))],
         'xgb__subsample': [len(range(0,1))],
         'xgb__booster': ['gbtree','dart']}]

xgboost = GridSearchCV(estimator=pipeline4,
                      param_grid= xgb,
                      scoring='accuracy',
                      cv=cv)

model(xgboost)
```

```
The Model Train accuracy is: 0.768
The Model Test accuracy is: 0.770


----------------------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.79      0.79      0.79      3568
         1.0       0.75      0.75      0.75      3109

    accuracy                           0.77      6677
   macro avg       0.77      0.77      0.77      6677
weighted avg       0.77      0.77      0.77      6677
```

In [258]:
```
print(xgboost.best_params_)
```

```
{'xgb__booster': 'gbtree', 'xgb__eta': 0.2, 'xgb__gamma': 49, 'xgb__max_
depth': 9, 'xgb__subsample': 1}
```

In [260]:
```
# setting best parameters
xgb = [{'xgb__eta': [ 0.2],
         'xgb__gamma': [49],
         'xgb__max_depth': [9],
         'xgb__subsample': [1],
         'xgb__booster': ['gbtree']}]

xgboost = GridSearchCV(estimator=pipeline4,
                      param_grid= xgb,
                      scoring='accuracy',
                      cv=cv)

model(xgboost)
```

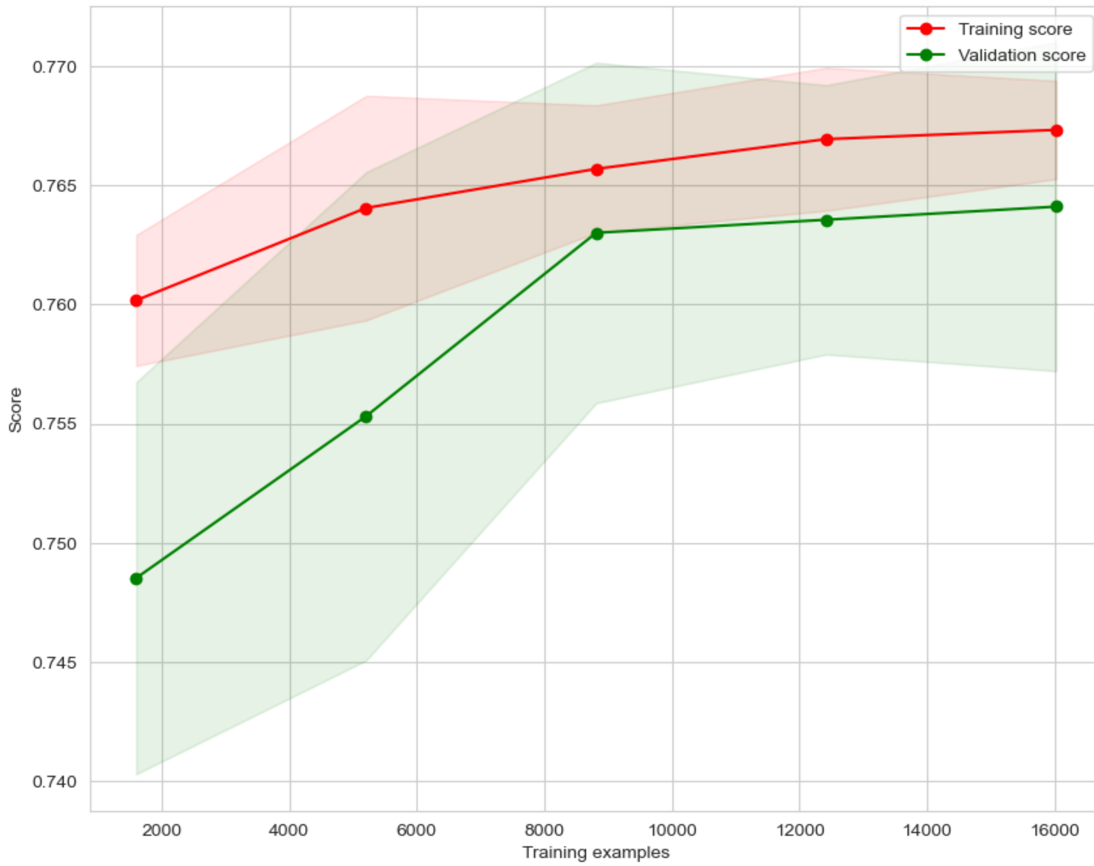
```
The Model Train accuracy is: 0.768
The Model Test accuracy is: 0.770


----------------------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.79      0.79      0.79      3568
         1.0       0.75      0.75      0.75      3109

    accuracy                           0.77      6677
   macro avg       0.77      0.77      0.77      6677
weighted avg       0.77      0.77      0.77      6677
```

In [261]:
```
#performing cross validation
cross_val = cross_val_score(pipeline4,X_train4,y_train4,cv=10)
print(f"Cross Validation Accuracy: {round(cross_val.mean()*100,4)}%")
```

```
Cross Validation Accuracy: 75.7114%
```

In [292]:

```
# plotting the learning curve

plot_learning_curve(xgboost)
```



Key Observations

- The model has a training accuracy of `76.8%` and a test accuracy of `77%` and a cross validation accuracy of `75.71%` .
- The learning curve shows that the training score starts with a high note suggesting overfitting however, it suddenly declines.
- These results indicate that the model performs consistently across different datasets, as the training and test accuracies are similar.
- The achieved accuracies are rather high, showing that the model predicts the outcome variable successfully.
- The model is learning from the training data and improving its performance on both the training and validation sets.
- Both scores are increasing, which implies that the model benefits from more training and has the potential to become more accurate.

MODEL EVALUATION

In [293]:

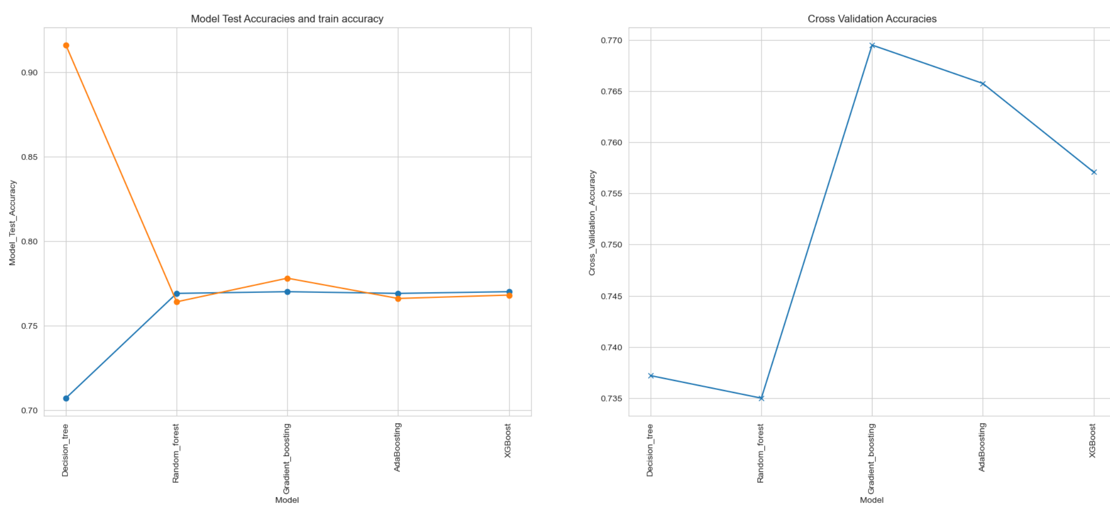
```
# Define the data and columns
data = np.array([
    ['Decision_tree', 0.916, 0.707, 0.7372],
    ['Random_forest', 0.764, 0.769, 0.735],
    ['Gradient_boosting', 0.778, 0.77, 0.7695],
    ['AdaBoosting', 0.766, 0.769, 0.76575],
    ['XGBoost', 0.768, 0.77, 0.7571]])

# Create a DataFrame with the given data
scores_data = pd.DataFrame(data)

# Assign column names to the DataFrame
scores_data.columns = ['Model', 'Model_Train_Accuracy', 'Model_Test_Accura


columns_to_convert = ['Model_Train_Accuracy', 'Model_Test_Accuracy', 'Cros
scores_data[columns_to_convert] = scores_data[columns_to_convert].astype(

scores_data
```

Out[293]:

	Model	Model_Train_Accuracy	Model_Test_Accuracy	Cross_Validation_Accurac
0	Decision_tree	0.916	0.707	0.7372
1	Random_forest	0.764	0.769	0.7350
2	Gradient_boosting	0.778	0.770	0.7695
3	AdaBoosting	0.766	0.769	0.7657
4	XGBoost	0.768	0.770	0.7571

In [295]:

```
# Create subplots with 1 row and 2 columns
fig, axes = plt.subplots(1, 2, figsize=(22, 8))  # Adjust the figsize as n

# Plot Model_Test_Accuracy in the first subplot
axes[0].plot(scores_data['Model'], scores_data['Model_Test_Accuracy'], mar
axes[0].plot(scores_data['Model'], scores_data['Model_Train_Accuracy'], ma

axes[0].set_xlabel('Model')
axes[0].set_ylabel('Model_Test_Accuracy')
axes[0].set_title('Model Test Accuracies and train accuracy')
axes[0].tick_params(axis='x', rotation=90)
axes[0].grid(True)

# Plot Cross_Validation_Accuracy in the second subplot
axes[1].plot(scores_data['Model'], scores_data['Cross_Validation_Accuracy
axes[1].set_xlabel('Model')
axes[1].set_ylabel('Cross_Validation_Accuracy')
axes[1].set_title('Cross Validation Accuracies')
axes[1].tick_params(axis='x', rotation=90)
axes[1].grid(True)
```



Use above subplots and learning curve to find best model

Measures of success:

- Accuracy and ability to generalize over unseen data.
- Potential to learn from more data and not be at risk of over-fitting.

Filtering:

- The Accuracy plot above clearly shows that Decision Tree does not match this criteria, as it has the lowest accuracy score.
- This is followed by Random Forest and AdaBoosting, both of which have low accuracy and hence do not make the cut.
- We are left with Gradient Boost and XGBoost.

- XGBoost has lower cross validation score compared to Gradient Boost.
- As more data is trained for the gradient boosting model, the validation score steadily increases along with the training score. It is adjusting to perform better in difficult situations while increasing the accuracy of its predictions.
- From the above analysis, Gradient boost has better prospects on performance as more data is used.

BEST MODEL : GRADIENT BOOST

Pickle

- This is done to avoid having to retrain the model.

In [296]:

```
# import necessary library
import joblib

# Create an instance of the GradientBoostingClassifier with specified para

gradient_boosting = GradientBoostingClassifier(loss='deviance', learning_r

# Save the trained model using joblib.dump

joblib.dump(gradient_boosting, 'gradient_boosting.pkl')
```

Out[296]:

```
['gradient_boosting.pkl']
```

Conclusion

- While this dataset contains some intriguing insights, it is heavily weighted in favor of specific communities and classifications.
- The gradient boosting model was optimal despite the fact that the dataset contains biases within our features. The data needs to be better balanced.
- The data should be better sampled to avoid biases and ensure that all groups are adequately represented, and it would benefit from more data since accuracy increased as more data was submitted.

Recommendations

- Careful examination of the significance of identified predictors, such as opinion_seas_risk, will help understand underlying factors and ensure fairness in decision-making
- According to the models, people's opinions have a significant impact on their chance to take immunizations. Given this, the following suggestions can be made:
- Campaigns should be launched to raise public knowledge about the effectiveness of the seasonal flu vaccine as well as the hazards connected with the virus.
- It would be beneficial to emphasize the safety of the vaccines for public use.

- The seasonal flu vaccine is more likely to be used by older persons. The younger population could, therefore, be targeted for such campaigns.
- To ensure better data gathering, a broader range of demographic groups should be included, resulting in a more diverse and balanced dataset.
- It is recommended to validate conclusions by combining external data from diverse sources to improve generalizability.
- Techniques like as oversampling, undersampling, or the use of weighted loss functions