Flatiron Phase 3 Seasonal Flu Vaccination Project

- Jonah Devoy
- Contact Me: [Linkedin URL (www.linkedin.com/in/jonahdevoy)](www.linkedin.com/in/jonahdevoy)

Features:

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

- h1n1_concern - Level of concern about the H1N1 flu. 0 = Not at all concerned; 1 = Not very concerned; 2 = Somewhat concerned; 3 = Very concerned.
- h1n1_knowledge - Level of knowledge about H1N1 flu. 0 = No knowledge; 1 = A little knowledge; 2 = A lot of knowledge.
- behavioral_antiviral_meds - Has taken antiviral medications. (binary)
- behavioral_avoidance - Has avoided close contact with others with flu-like symptoms. (binary)
- behavioral_face_mask - Has bought a face mask. (binary)
- behavioral_wash_hands - Has frequently washed hands or used hand sanitizer. (binary)
- behavioral_large_gatherings - Has reduced time at large gatherings. (binary)
- behavioral_outside_home - Has reduced contact with people outside of own household. (binary)
- behavioral_touch_face - Has avoided touching eyes, nose, or mouth. (binary)
- doctor_recc_h1n1 - H1N1 flu vaccine was recommended by doctor. (binary)
- doctor_recc_seasonal - Seasonal flu vaccine was recommended by doctor. (binary)
- chronic_med_condition - Has any of the following chronic medical conditions: asthma or an other lung condition,
- diabetes, a heart condition, a kidney condition, sickle cell anemia or other anemia, a neurological or neuromuscular
- condition, a liver condition, or a weakened immune system caused by a chronic illness or by medicines taken for a
- chronic illness. (binary)
- child_under_6_months - Has regular close contact with a child under the age of six months. (binary)
- health_worker - Is a healthcare worker. (binary)
- health_insurance - Has health insurance. (binary)
- opinion_h1n1_vacc_effective - Respondent's opinion about H1N1 vaccine effectiveness. 1 = Not at all effective; 2 = Not very effective; 3 = Don't know; 4 = Somewhat effective; 5 = Very effective.
- opinion_h1n1_risk - Respondent's opinion about risk of getting sick with H1N1 flu without vaccine. 1 = Very Low; 2 = Somewhat low; 3 = Don't know; 4 = Somewhat high; 5 = Very high.
- opinion_h1n1_sick_from_vacc - Respondent's worry of getting sick from taking H1N1 vaccine. 1 = Not at all worried; 2 = Not very worried; 3 = Don't know; 4 = Somewhat worried; 5 = Very worried.
- opinion_seas_vacc_effective - Respondent's opinion about seasonal flu vaccine effectiveness. 1 = Not at all effective; 2 = Not very effective; 3 = Don't know; 4 = Somewhat effective; 5 = Very effective.
- opinion_seas_risk - Respondent's opinion about risk of getting sick with seasonal flu without vaccine. 1 = Very Low; 2 = Somewhat low; 3 = Don't know; 4 = Somewhat high; 5 = Very high.
- opinion_seas_sick_from_vacc - Respondent's worry of getting sick from taking seasonal flu vaccine. 1 = Not at all worried; 2 = Not very worried; 3 = Don't know; 4 = Somewhat worried; 5 = Very worried.
- age_group - Age group of respondent.
- education - Self-reported education level.
- race - Race of respondent.
- sex - Sex of respondent.
- income_poverty - Household annual income of respondent with respect to 2008 Census poverty thresholds.
- marital_status - Marital status of respondent.
- rent_or_own - Housing situation of respondent.
- employment_status - Employment status of respondent.
- hhs_geo_region - Respondent's residence using a 10-region geographic classification defined by the U.S. Dept. of
- Health and Human Services. Values are represented as short random character strings.
- census_msa - Respondent's residence within metropolitan statistical areas (MSA) as defined by the U.S. Census.
- household_adults - Number of other adults in household, top-coded to 3.
- household_children - Number of children in household, top-coded to 3.
- employment_industry - Type of industry respondent is employed in. Values are represented as short random character strings.
- employment_occupation - Type of occupation of respondent. Values are represented as short random character strings.

In [1]:
```
# import necessary libraries

import pandas as pd
pd.set_option('display.max_columns', 100)
pd.set_option('display.float_format', lambda x: '%.3f' % x)

import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(style="darkgrid")

from ydata_profiling import ProfileReport
import missingno
%matplotlib inline
# from IPython.display import Image
## import function needed for split
from sklearn.model_selection import train_test_split


## import classes necessary for building preprocessing pipelines
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn import tree


from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
```

In [2]:
```
def calculate_null_percentage(df):


    import pandas as pd

    missing_vals = pd.DataFrame()
    missing_vals['Number of Nulls'] = df.isna().sum()
    missing_vals['% Null'] = (df.isna().sum() / len(df)) * 100

    return missing_vals


def check_unique(df, col, dropna=False):



    import pandas as pd

    unique_vals = pd.DataFrame(df[col].value_counts(dropna=dropna))

    return unique_vals
```

In [3]:
```
#conda install -c conda-forge missingno
```

In [4]:
```
#pip install -U ydata-profiling
```

In [5]:
```
# Loading in the data
df_1 = pd.read_csv('/Users/jdapeman/Documents/flu_shot_V1/CSV_FOLDER/Flu_Shot_Learning_Predict_H1N1_and_Seasonal_Flu_Va
features_df = pd.read_csv('/Users/jdapeman/Documents/flu_shot_V1/CSV_FOLDER/Flu_Shot_Learning_Predict_H1N1_and_Seasonal
labels_df = pd.read_csv('/Users/jdapeman/Documents/flu_shot_V1/CSV_FOLDER/Flu_Shot_Learning_Predict_H1N1_and_Seasonal_F
```

In [6]:
```
#profile1 = ProfileReport(df, title="Profiling Report")
#profile1
```

In [7]:
```
#profile2 = ProfileReport(features_df, title="Profiling Report")
#profile2
```

In [8]:
```
#profile3 = ProfileReport(labels_df, title="Profiling Report")
#profile3
```

In [9]:
```
# observing the first 5 rows of the features dataframe
features_df.head()
```

Out[9]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	behavioral_large
0	0	1.000	0.000	0.000	0.000	0.000	0.000	
1	1	3.000	2.000	0.000	1.000	0.000	1.000	
2	2	1.000	1.000	0.000	1.000	0.000	0.000	
3	3	1.000	1.000	0.000	1.000	0.000	1.000	
4	4	2.000	1.000	0.000	1.000	0.000	1.000	

In [10]:
```
# observing the first 5 rows of the labels dataframe
labels_df.head()
```

Out[10]:

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

In [11]:
```
# observing the first 5 rows of the test features dataframe
df_1.head()
```

Out[11]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	behavioral_large
0	26707	2.000	2.000	0.000	1.000	0.000	1.000	
1	26708	1.000	1.000	0.000	0.000	0.000	0.000	
2	26709	2.000	2.000	0.000	0.000	1.000	1.000	
3	26710	1.000	1.000	0.000	0.000	0.000	0.000	
4	26711	3.000	1.000	1.000	1.000	0.000	1.000	

In [12]:
```
# combining the lables and training features into a single dataframe
df = pd.concat([features_df, labels_df.drop('respondent_id', axis=1)], axis=1)
df
```

Out[12]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	behavioral_
0	0	1.000	0.000	0.000	0.000	0.000	0.000	
1	1	3.000	2.000	0.000	1.000	0.000	1.000	
2	2	1.000	1.000	0.000	1.000	0.000	0.000	
3	3	1.000	1.000	0.000	1.000	0.000	1.000	
4	4	2.000	1.000	0.000	1.000	0.000	1.000	
...	...	...	...	...	...	...	...	
26702	26702	2.000	0.000	0.000	1.000	0.000	0.000	
26703	26703	1.000	2.000	0.000	1.000	0.000	1.000	
26704	26704	2.000	2.000	0.000	1.000	1.000	1.000	
26705	26705	1.000	1.000	0.000	0.000	0.000	0.000	
26706	26706	0.000	0.000	0.000	1.000	0.000	0.000	

26707 rows × 38 columns

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26707 entries, 0 to 26706
Data columns (total 38 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   respondent_id                26707 non-null  int64  
 1   h1n1_concern                 26615 non-null  float64
 2   h1n1_knowledge               26591 non-null  float64
 3   behavioral_antiviral_meds    26636 non-null  float64
 4   behavioral_avoidance         26499 non-null  float64
 5   behavioral_face_mask         26688 non-null  float64
 6   behavioral_wash_hands        26665 non-null  float64
 7   behavioral_large_gatherings  26620 non-null  float64
 8   behavioral_outside_home      26625 non-null  float64
 9   behavioral_touch_face        26579 non-null  float64
 10  doctor_recc_h1n1             24547 non-null  float64
 11  doctor_recc_seasonal         24547 non-null  float64
 12  chronic_med_condition        25736 non-null  float64
 13  child_under_6_months         25887 non-null  float64
 14  health_worker                25903 non-null  float64
 15  health_insurance             14433 non-null  float64
 16  opinion_h1n1_vacc_effective  26316 non-null  float64
 17  opinion_h1n1_risk            26319 non-null  float64
 18  opinion_h1n1_sick_from_vacc  26312 non-null  float64
 19  opinion_seas_vacc_effective  26245 non-null  float64
 20  opinion_seas_risk            26193 non-null  float64
 21  opinion_seas_sick_from_vacc  26170 non-null  float64
 22  age_group                    26707 non-null  object 
 23  education                    25300 non-null  object 
 24  race                         26707 non-null  object 
 25  sex                          26707 non-null  object 
 26  income_poverty               22284 non-null  object 
 27  marital_status               25299 non-null  object 
 28  rent_or_own                  24665 non-null  object 
 29  employment_status            25244 non-null  object 
 30  hhs_geo_region               26707 non-null  object 
 31  census_msa                   26707 non-null  object 
 32  household_adults             26458 non-null  float64
 33  household_children           26458 non-null  float64
 34  employment_industry          13377 non-null  object 
 35  employment_occupation        13237 non-null  object 
 36  h1n1_vaccine                 26707 non-null  int64  
 37  seasonal_vaccine             26707 non-null  int64  
dtypes: float64(23), int64(3), object(12)
memory usage: 7.7+ MB
```

In [14]:
```
df.describe()
```

Out[14]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	behavioral_
count	26707.000	26615.000	26591.000	26636.000	26499.000	26688.000	26665.000	
mean	13353.000	1.618	1.263	0.049	0.726	0.069	0.826	
std	7709.791	0.910	0.618	0.216	0.446	0.253	0.379	
min	0.000	0.000	0.000	0.000	0.000	0.000	0.000	
25%	6676.500	1.000	1.000	0.000	0.000	0.000	1.000	
50%	13353.000	2.000	1.000	0.000	1.000	0.000	1.000	
75%	20029.500	2.000	2.000	0.000	1.000	0.000	1.000	
max	26706.000	3.000	2.000	1.000	1.000	1.000	1.000	

In [15]:
```
df.corr(numeric_only = [False])
```

Out[15]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidance	behavioral_face_mask	behavioral_w
respondent_id	1.000	0.018	0.003	-0.008	0.010	-0.007	
h1n1_concern	0.018	1.000	0.063	0.090	0.234	0.156	
h1n1_knowledge	0.003	0.063	1.000	-0.011	0.089	0.030	
behavioral_antiviral_meds	-0.008	0.090	-0.011	1.000	0.049	0.146	
behavioral_avoidance	0.010	0.234	0.089	0.049	1.000	0.065	
behavioral_face_mask	-0.007	0.156	0.030	0.146	0.065	1.000	
behavioral_wash_hands	0.011	0.294	0.090	0.064	0.338	0.083	
behavioral_large_gatherings	0.005	0.255	-0.049	0.106	0.228	0.181	
behavioral_outside_home	0.009	0.246	-0.068	0.128	0.220	0.163	
behavioral_touch_face	0.008	0.248	0.086	0.071	0.335	0.104	
doctor_recc_h1n1	-0.002	0.150	0.094	0.051	0.068	0.084	
doctor_recc_seasonal	0.001	0.136	0.072	0.031	0.074	0.069	
chronic_med_condition	0.006	0.095	-0.023	0.008	0.039	0.068	
child_under_6_months	-0.005	0.050	0.022	0.029	-0.000	0.040	
health_worker	-0.003	0.034	0.170	0.009	0.001	0.070	
health_insurance	-0.013	-0.004	0.119	-0.064	0.033	-0.040	
opinion_h1n1_vacc_effective	0.006	0.240	0.121	0.030	0.112	0.038	
opinion_h1n1_risk	0.001	0.377	0.073	0.105	0.118	0.131	
opinion_h1n1_sick_from_vacc	-0.002	0.360	-0.020	0.079	0.131	0.107	
opinion_seas_vacc_effective	0.006	0.235	0.086	0.015	0.120	0.042	
opinion_seas_risk	-0.005	0.334	0.077	0.085	0.130	0.110	
opinion_seas_sick_from_vacc	0.010	0.226	-0.062	0.084	0.083	0.090	
household_adults	0.000	-0.016	0.025	0.045	0.019	0.014	
household_children	-0.004	0.051	0.051	0.085	0.040	0.006	
h1n1_vaccine	-0.003	0.122	0.118	0.041	0.048	0.070	
seasonal_vaccine	-0.005	0.155	0.120	0.006	0.076	0.050	

In [16]:
```
df.shape
```

Out[16]:
```
(26707, 38)
```

In [17]:
```
# Check for null or missing values in each column
for column in df.columns:
    null_values = df[column].isnull()

    if null_values.any():
        print(f"Null values in the column '{column}':")
        print(df[column][null_values])
        print("============================================")
```

```
Null values in the column 'h1n1_concern':
44       NaN
96       NaN
150      NaN
411      NaN
758      NaN
          ..
25788    NaN
25883    NaN
25948    NaN
26358    NaN
26471    NaN
Name: h1n1_concern, Length: 92, dtype: float64
============================================
Null values in the column 'h1n1_knowledge':
136      NaN
405      NaN
958      NaN
1026     NaN
```

In [18]:
```
# the focus for this notebook is on seasonal flu vaccines, so any columns relating to H1N1 vaccines can be dropped
# and we will drop columns unrelated to the seasonal flu vaccine
df.drop(columns=['opinion_h1n1_vacc_effective',
                 'opinion_h1n1_risk',
                 'opinion_h1n1_sick_from_vacc',
                 'doctor_recc_h1n1',
                 'h1n1_vaccine',
                 'behavioral_antiviral_meds',
                 'respondent_id'], axis=1, inplace=True)
```

In [19]:
```
df.shape
```

Out[19]: (26707, 31)

In [20]:
```
df.head()
```

Out[20]:

	h1n1_concern	h1n1_knowledge	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	behavioral_large_gatherings	behavioral_outside_home	be
0	1.000	0.000	0.000	0.000	0.000	0.000	1.000	
1	3.000	2.000	1.000	0.000	1.000	0.000	1.000	
2	1.000	1.000	1.000	0.000	0.000	0.000	0.000	
3	1.000	1.000	1.000	0.000	1.000	1.000	0.000	
4	2.000	1.000	1.000	0.000	1.000	1.000	0.000	

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26707 entries, 0 to 26706
Data columns (total 31 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   h1n1_concern                 26615 non-null  float64
 1   h1n1_knowledge               26591 non-null  float64
 2   behavioral_avoidance         26499 non-null  float64
 3   behavioral_face_mask         26688 non-null  float64
 4   behavioral_wash_hands        26665 non-null  float64
 5   behavioral_large_gatherings  26620 non-null  float64
 6   behavioral_outside_home      26625 non-null  float64
 7   behavioral_touch_face        26579 non-null  float64
 8   doctor_recc_seasonal         24547 non-null  float64
 9   chronic_med_condition        25736 non-null  float64
 10  child_under_6_months         25887 non-null  float64
 11  health_worker                25903 non-null  float64
 12  health_insurance             14433 non-null  float64
 13  opinion_seas_vacc_effective  26245 non-null  float64
 14  opinion_seas_risk            26193 non-null  float64
 15  opinion_seas_sick_from_vacc  26170 non-null  float64
 16  age_group                    26707 non-null  object 
 17  education                    25300 non-null  object 
 18  race                         26707 non-null  object 
 19  sex                          26707 non-null  object 
 20  income_poverty               22284 non-null  object 
 21  marital_status               25299 non-null  object 
 22  rent_or_own                  24665 non-null  object 
 23  employment_status            25244 non-null  object 
 24  hhs_geo_region               26707 non-null  object 
 25  census_msa                   26707 non-null  object 
 26  household_adults             26458 non-null  float64
 27  household_children           26458 non-null  float64
 28  employment_industry          13377 non-null  object 
 29  employment_occupation        13237 non-null  object 
 30  seasonal_vaccine             26707 non-null  int64  
dtypes: float64(18), int64(1), object(12)
memory usage: 6.3+ MB
```

The columns hhs_geo_region, employment_industry, and employment_occupation are random strings that have been scrambled and encoded for anonymity, the definition is not provided by the CDC. But since they are consistent and specific to the individual, they can be used to make accurate predictions.

In [22]:
```
#pip install pandasgui
```

In [23]:
```
#pip install git+https://github.com/adamerose/pandasgui.git
```

In [24]:
```
# pandas gui
#from pandasgui import show
#show(df)
```

In [25]:
```
# A function to print out the number of nulls in each column as well as the percentage of nulls
def calculate_null_percentage(df):
    missing_vals = pd.DataFrame()
    missing_vals['Number of Nulls'] = df.isna().sum()
    missing_vals['% Null'] = (df.isna().sum() / len(df)) * 100

    return missing_vals
```

In [26]:
```
calculate_null_percentage(df)
```

Out[26]:

	Number of Nulls	% Null
h1n1_concern	92	0.344
h1n1_knowledge	116	0.434
behavioral_avoidance	208	0.779
behavioral_face_mask	19	0.071
behavioral_wash_hands	42	0.157
behavioral_large_gatherings	87	0.326
behavioral_outside_home	82	0.307
behavioral_touch_face	128	0.479
doctor_recc_seasonal	2160	8.088
chronic_med_condition	971	3.636
child_under_6_months	820	3.070
health_worker	804	3.010
health_insurance	12274	45.958
opinion_seas_vacc_effective	462	1.730
opinion_seas_risk	514	1.925
opinion_seas_sick_from_vacc	537	2.011
age_group	0	0.000
education	1407	5.268
race	0	0.000
sex	0	0.000
income_poverty	4423	16.561
marital_status	1408	5.272
rent_or_own	2042	7.646
employment_status	1463	5.478
hhs_geo_region	0	0.000
census_msa	0	0.000
household_adults	249	0.932
household_children	249	0.932
employment_industry	13330	49.912
employment_occupation	13470	50.436
seasonal_vaccine	0	0.000

In [27]:
```
# Calculate the percentage of missing values for each column and then
# filter columns with less than 5% missing values and not equal to zero
def print_columns_missing_info(df):
    total_rows = len(df)
    missing_percentages = (df.isnull().sum() / total_rows) * 100
    columns_missing_info = missing_percentages[(missing_percentages < 5) & (missing_percentages > 0)]
    if len(columns_missing_info) > 0:
        for column in columns_missing_info.index:
            print(column)

print_columns_missing_info(df)
```

```
h1n1_concern
h1n1_knowledge
behavioral_avoidance
behavioral_face_mask
behavioral_wash_hands
behavioral_large_gatherings
behavioral_outside_home
behavioral_touch_face
chronic_med_condition
child_under_6_months
health_worker
opinion_seas_vacc_effective
opinion_seas_risk
opinion_seas_sick_from_vacc
household_adults
household_children
```

Notes About the Data and going forward:

- We can see the columns employment_occupation, employment_industry, and health_insurance exhibit the highest number of missing values. Among the null values in employment_occupation and employment_industry, 10,231 values correspond to individuals categorized as 'Not in Labor Force' in the employment_status column. These can be considered as N/A rather than individuals declining to answer. 1,453 observances represent unemployed individuals where the employment_occupation and employment_industry columns are appropriately labeled as 'not applicable'.
- We can also see that if an individual declined to answer if their doctor recommended one type of vaccine, they typically refused to answer about the recommendation for the other type. There is a tendency for individuals to refuse to answer questions related to having a chronic medical condition, having a child under 6 months, being a health worker, having opinion-based questions, income, education, and personal and home life topics, for reasons unknown to us.
- We should be treating missing information for specific variables as its distinctive category, instead of removing it because a non answer in itself is a kind of responce

In [28]:
```
no_null_cols = [col for col in df.columns if df[col].isna().sum()==0]
no_null_cols
```

Out[28]:
```
['age_group',
 'race',
 'sex',
 'hhs_geo_region',
 'census_msa',
 'seasonal_vaccine']
```

In [29]:
```
# select individuals not in the labor force
not_in_labor_force = df[df['employment_status']=='Not in Labor Force']
calculate_null_percentage(not_in_labor_force)
```

	Number of Nulls	% Null
h1n1_concern	56	0.547
h1n1_knowledge	58	0.567
behavioral_avoidance	104	1.017
behavioral_face_mask	9	0.088
behavioral_wash_hands	22	0.215
behavioral_large_gatherings	46	0.450
behavioral_outside_home	49	0.479
behavioral_touch_face	67	0.655
doctor_recc_seasonal	843	8.240
chronic_med_condition	91	0.889
child_under_6_months	1	0.010
health_worker	6	0.059

In [30]:
```
# select unemployed individuals
unemployed = df[df['employment_status']=='Unemployed']
calculate_null_percentage(unemployed)
```

Out[30]:

	Number of Nulls	% Null
h1n1_concern	3	0.206
h1n1_knowledge	10	0.688
behavioral_avoidance	6	0.413
behavioral_face_mask	1	0.069
behavioral_wash_hands	0	0.000
behavioral_large_gatherings	5	0.344
behavioral_outside_home	2	0.138
behavioral_touch_face	7	0.482
doctor_recc_seasonal	94	6.469
chronic_med_condition	16	1.101
child_under_6_months	1	0.069

In [31]:
```
# takes the dataframe and column name and returns the unique values in that column as well as the
# number of each unique values.
def count_unique_values(df, col, dropna=False):
    unique_vals = pd.DataFrame(df[col].value_counts(dropna=dropna))

    return unique_vals
```

In [32]:
```
# creating not_employed from labor force
# if a person is unemployed change their 'employment_industry' to 'not_employed'
df.loc[df['employment_status'] == 'Unemployed', 'employment_industry'] = 'not employed'

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

count_unique_values(df, 'employment_industry')
```

Out[32]:

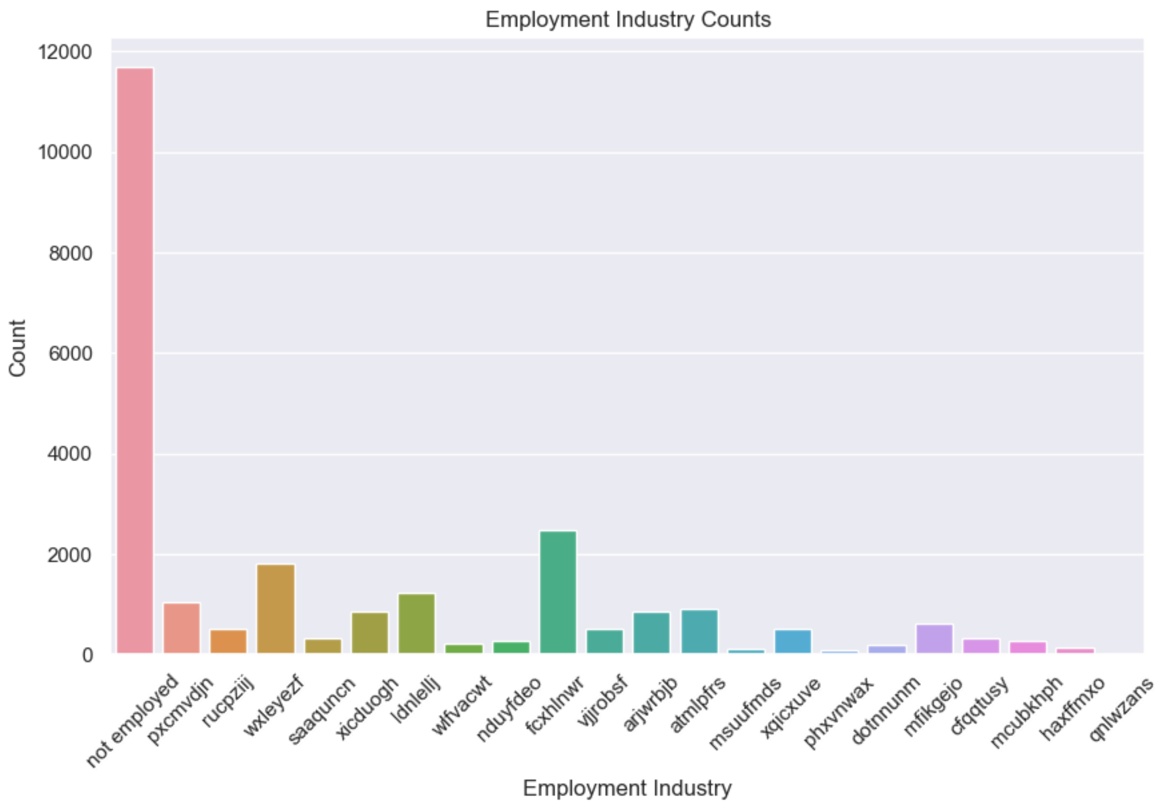
	employment_industry
not employed	11684
fcxhlnwr	2468
wxleyezf	1804
NaN	1646
ldnlellj	1231
pxcmvdjn	1037
atmlpfrs	926
arjwrbjb	871
xicduogh	851
mfikgejo	614
vjjrobsf	527
rucpziij	523
xqicxuve	511
saaquncn	338
cfqqtusy	325
nduyfdeo	286
mcubkhph	275
wlfvacwt	215
dotnnunm	201
haxffmxo	148
msuufmds	124
phxvnwax	89
qnlwzans	13

In [33]:
```
# creating not_employed from employment industry
# if a person is unemployed, change their 'employment_industry' to 'not_employed'
df.loc[df['employment_status'] == 'Unemployed', 'employment_occupation'] = 'not employed'

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

count_unique_values(df, 'employment_occupation')
```

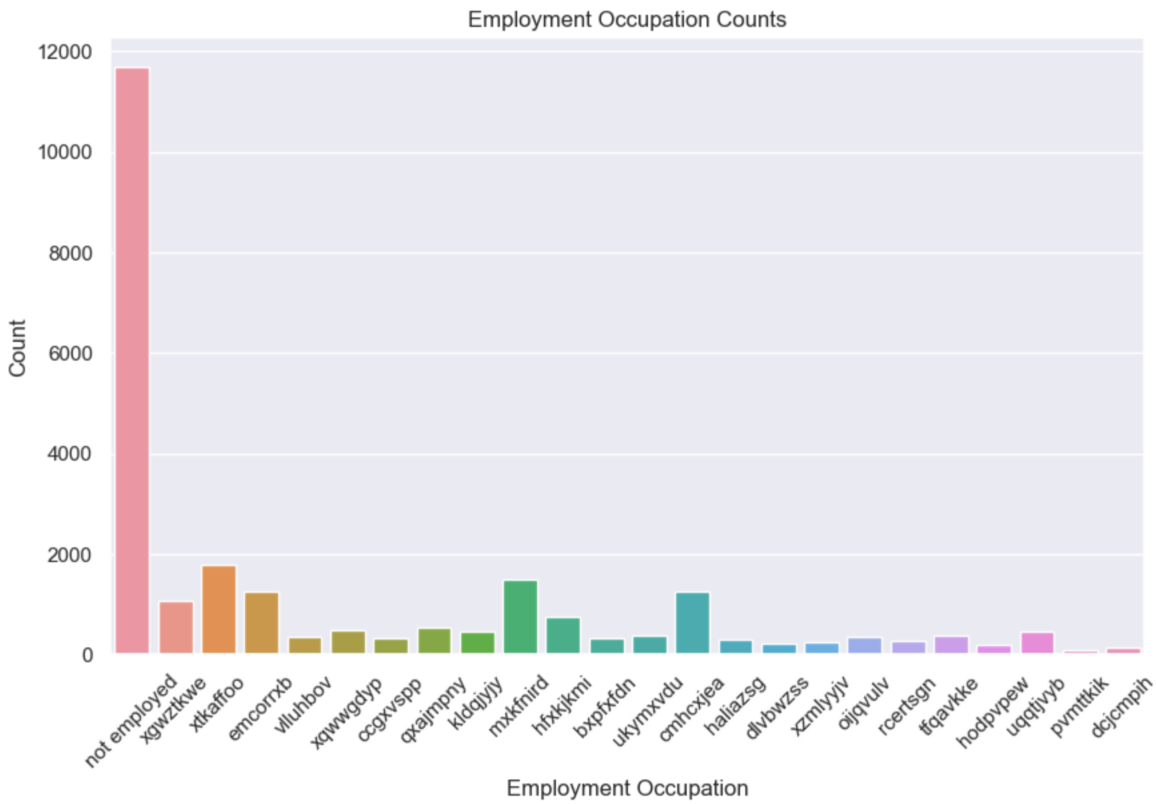
Out[33]:

	employment_occupation
not employed	11684
NaN	1786
xtkaffoo	1778
mxkfnird	1509
emcorrxb	1270
cmhcxjea	1247
xgwztkwe	1082
hfxkjkmi	766
qxajmpny	548
xqwwgdyp	485
kldqjyjy	469
uqqtjvyb	452
tfqavkke	388
ukymxvdu	372
vlluhbov	354
oijqvulv	344
ccgxvspp	341
bxpfxfdn	331
haliazsg	296
rcertsgn	276
xzmlyyjv	248
dlvbwzss	227
hodpvpew	208
dcjcmpih	148
pvmttkik	98

In [34]:
```
# Bar plot of employment industry counts
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='employment_industry')
plt.title('Employment Industry Counts')
plt.xlabel('Employment Industry')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



In [35]:
```
# Bar plot of employment occupation counts
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='employment_occupation')
plt.title('Employment Occupation Counts')
plt.xlabel('Employment Occupation')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



In [36]:
```
# columns with null between 0 and 5%, exluding values of 0
null_df = calculate_null_percentage(df)
null_df.drop(index=null_df.loc[null_df['% Null']==0].index, axis=0, inplace=True)
under_5_null = null_df.loc[null_df['% Null']<5]
under_5_null
```

Out[36]:

	Number of Nulls	% Null
h1n1_concern	92	0.344
h1n1_knowledge	116	0.434
behavioral_avoidance	208	0.779
behavioral_face_mask	19	0.071
behavioral_wash_hands	42	0.157
behavioral_large_gatherings	87	0.326
behavioral_outside_home	82	0.307
behavioral_touch_face	128	0.479
chronic_med_condition	971	3.636
child_under_6_months	820	3.070
health_worker	804	3.010
opinion_seas_vacc_effective	462	1.730
opinion_seas_risk	514	1.925
opinion_seas_sick_from_vacc	537	2.011
household_adults	249	0.932
household_children	249	0.932

In [37]:
```
# dropping the rows with less than 5% of null values
under_5_null_cols = list(under_5_null.index)
df.dropna(subset=under_5_null_cols, inplace=True)
df.head()
```

Out[37]:

	h1n1_concern	h1n1_knowledge	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	behavioral_large_gatherings	behavioral_outside_home	be
0	1.000	0.000	0.000	0.000	0.000	0.000	1.000	
1	3.000	2.000	1.000	0.000	1.000	0.000	1.000	
2	1.000	1.000	1.000	0.000	0.000	0.000	0.000	
3	1.000	1.000	1.000	0.000	1.000	1.000	0.000	
4	2.000	1.000	1.000	0.000	1.000	1.000	0.000	

In [38]:
```
df.shape
```

Out[38]:
```
(24939, 31)
```

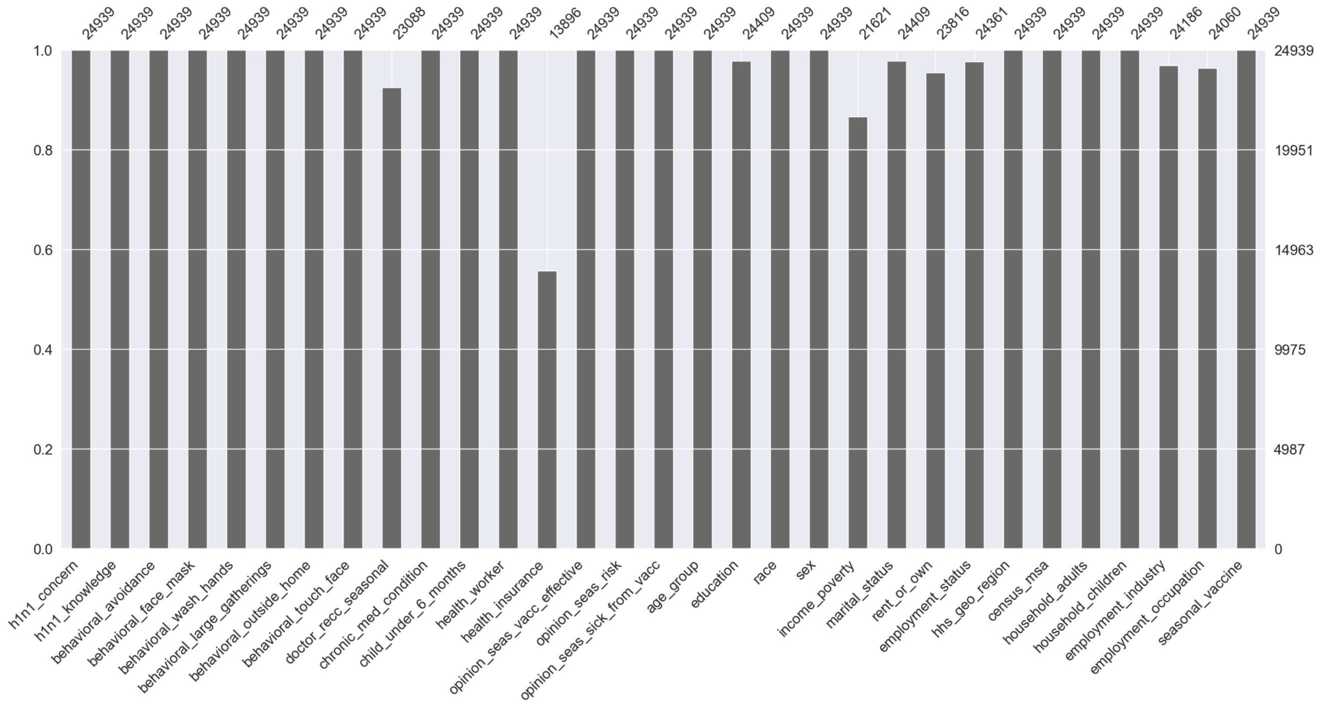
In [39]:
```
# the result is a dataframe with rows containing less than 5% of info have been dropped
calculate_null_percentage(df)
```

Out[39]:

	Number of Nulls	% Null
h1n1_concern	0	0.000
h1n1_knowledge	0	0.000
behavioral_avoidance	0	0.000
behavioral_face_mask	0	0.000
behavioral_wash_hands	0	0.000
behavioral_large_gatherings	0	0.000
behavioral_outside_home	0	0.000
behavioral_touch_face	0	0.000
doctor_recc_seasonal	1851	7.422
chronic_med_condition	0	0.000
child_under_6_months	0	0.000
health_worker	0	0.000
health_insurance	11043	44.280
opinion_seas_vacc_effective	0	0.000
opinion_seas_risk	0	0.000
opinion_seas_sick_from_vacc	0	0.000
age_group	0	0.000
education	530	2.125
race	0	0.000
sex	0	0.000
income_poverty	3318	13.304
marital_status	530	2.125
rent_or_own	1123	4.503
employment_status	578	2.318
hhs_geo_region	0	0.000
census_msa	0	0.000
household_adults	0	0.000
household_children	0	0.000
employment_industry	753	3.019
employment_occupation	879	3.525
seasonal_vaccine	0	0.000

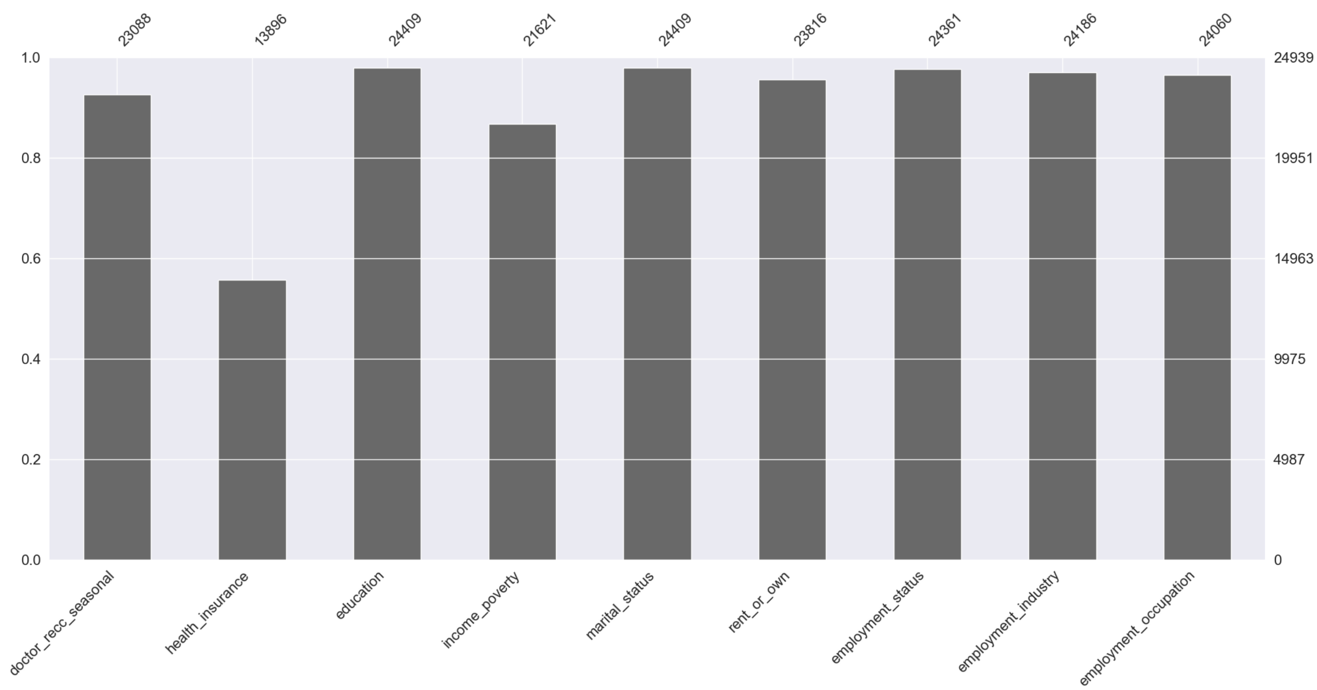
In [40]:
```
missingno.bar(df)
```

Out[40]: <Axes: >



In [41]:
```
cols_without_null = df.columns[df.isnull().sum() == 0].tolist()
missingno.bar(df.drop(columns=cols_without_null))
```

Out[41]: <Axes: >



In [42]:
```
## create a list of cols without any null values to be dropped from missingno.matrix
no_null_cols = [col for col in df.columns if df[col].isna().sum()==0]

no_null_cols
```

Out[42]:
```
['h1n1_concern',
 'h1n1_knowledge',
 'behavioral_avoidance',
 'behavioral_face_mask',
 'behavioral_wash_hands',
 'behavioral_large_gatherings',
 'behavioral_outside_home',
 'behavioral_touch_face',
 'chronic_med_condition',
 'child_under_6_months',
 'health_worker',
 'opinion_seas_vacc_effective',
 'opinion_seas_risk',
 'opinion_seas_sick_from_vacc',
 'age_group',
 'race',
 'sex',
 'hhs_geo_region',
 'census_msa',
 'household_adults',
 'household_children',
 'seasonal_vaccine']
```

- Because 44.2% declined to tell if they have health_insurance, information will become its own category for this variable.The same for the income_poverty variable, which is missing 13.4% of its values.
- The pattern of null values is so similar across the variables education, marital_status, rent_to_own, employment_status, employment_industry, and employment_occupation that dropping rows containing null values for any of these columns will drop most of the records containing null values for the other columns. Null values for these categorical variables will be filled with 'missing' as its own category since this missing information appears to represent a distinct kind of survey respondent.
- The race data is mostly white, with 19,856 out of 24,939 people sampled. The three other categorical races are Black, Hispanic, and Other or Multiple. We will combine these latter three underrepresented groups into one group for people of color.
- The sample population is slightly skewed towards women (59.6%).
- About half of the individuals surveyed declined to answer whether or not they had health insurance.

In [43]:
```
#profile_new_V1 = ProfileReport(df, title="Profiling Report")
#profile_new_V1
```

- We are going to make a variable 'behav_score' that represents how much an individual has done to avoid the flu by summing up all behavioral variables. Taking the sum across these columns, a higher score represents a more cautious person.
- We are going to make a variable 'behav_to_risk' that describes the ratio of how much a person has done to avoid the flu in considering their perception of the risk of getting the flu without the vaccine. The numerator is behav_score + 1 to distinguish people who are not doing anything to avoid the flu but vary by how concerned they are about getting sick without the vaccine. The denominator is the rating of risk perception, opinion_seas_risk. An individual with a very low score is someone who has done little to avoid the flu but is very concerned about getting sick without the vaccine. An individual with a score on the upper end has done a lot to behaviorally minimize their risk of exposure and is not very concerned about getting sick without the vaccine. This kind of person may be less likely to get the vaccine, even if they think it's effective because they feel they're doing enough to avoid getting the flu on thier own

In [44]:
```
behavior_cols = [x for x in df.columns if 'behavioral' in x]
behavior_cols
```

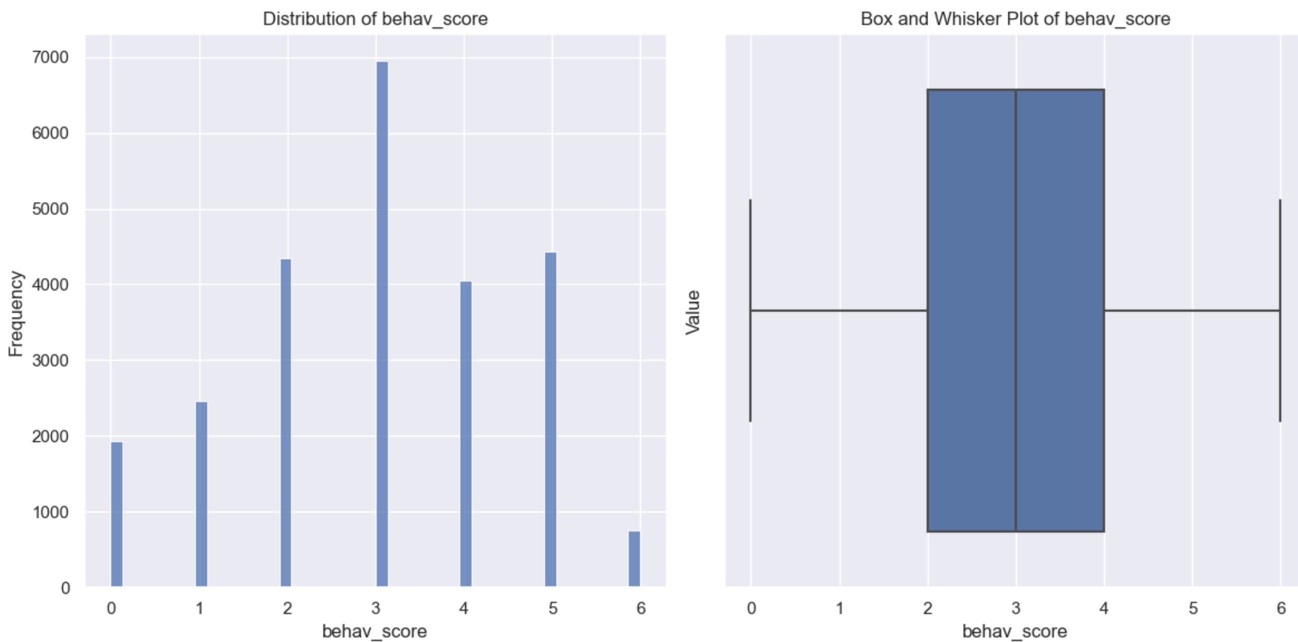
Out[44]:
```
['behavioral_avoidance',
 'behavioral_face_mask',
 'behavioral_wash_hands',
 'behavioral_large_gatherings',
 'behavioral_outside_home',
 'behavioral_touch_face']
```

In [45]:
```
df['behav_score'] = df[behavior_cols].sum(axis=1)
df.columns
```

Out[45]:
```
Index(['h1n1_concern', 'h1n1_knowledge', 'behavioral_avoidance',
       'behavioral_face_mask', 'behavioral_wash_hands',
       'behavioral_large_gatherings', 'behavioral_outside_home',
       'behavioral_touch_face', 'doctor_recc_seasonal',
       'chronic_med_condition', 'child_under_6_months', 'health_worker',
       'health_insurance', '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',
       'seasonal_vaccine', 'behav_score'],
      dtype='object')
```

In [46]:
```
# Get unique values and their counts
def check_column(df, column_name):
    unique_values = df[column_name].value_counts().reset_index()
    unique_values.columns = [column_name, 'Count']

    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

    sns.histplot(df[column_name].dropna(), kde=False, ax=axes[0])
    axes[0].set_xlabel(column_name)
    axes[0].set_ylabel('Frequency')
    axes[0].set_title('Distribution of {}'.format(column_name))

    sns.boxplot(x=column_name, data=df, ax=axes[1])
    axes[1].set_xlabel(column_name)
    axes[1].set_ylabel('Value')
    axes[1].set_title('Box and Whisker Plot of {}'.format(column_name))

    plt.tight_layout()
    plt.show()

    return unique_values
```

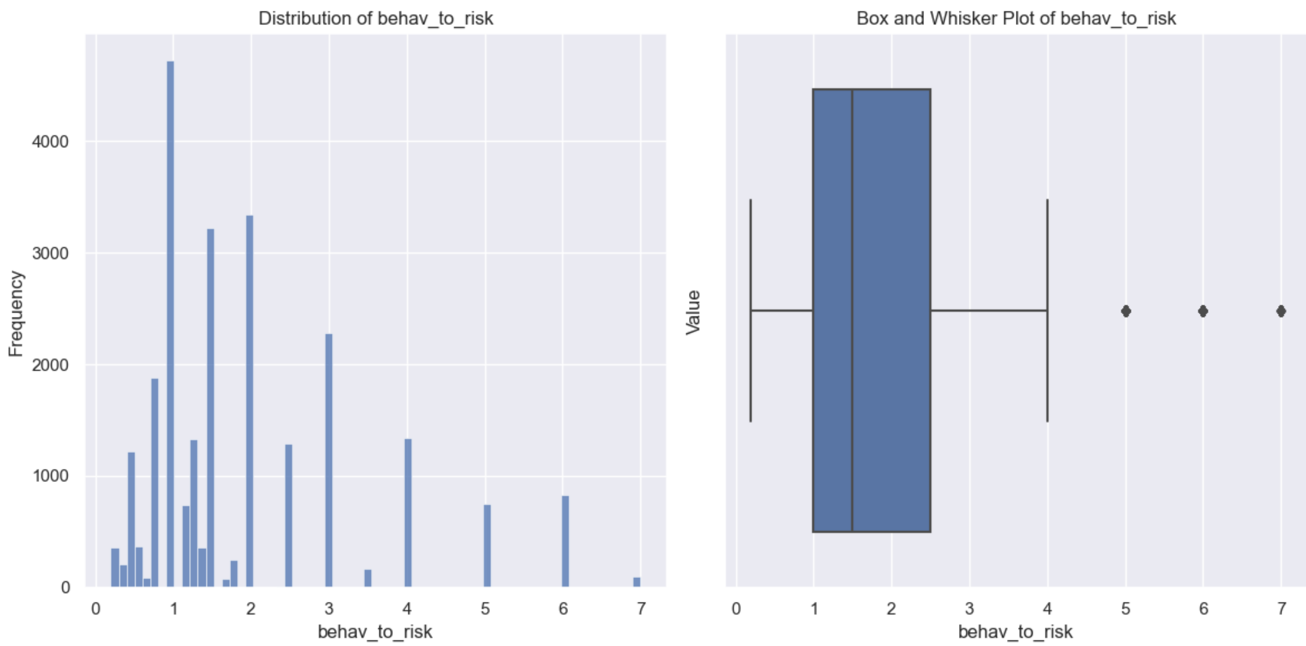
In [47]:
```
check_column(df, 'behav_score')
```



Out[47]:

	behav_score	Count
0	3.000	6952
1	5.000	4440
2	2.000	4348
3	4.000	4052
4	1.000	2465
5	0.000	1935
6	6.000	747

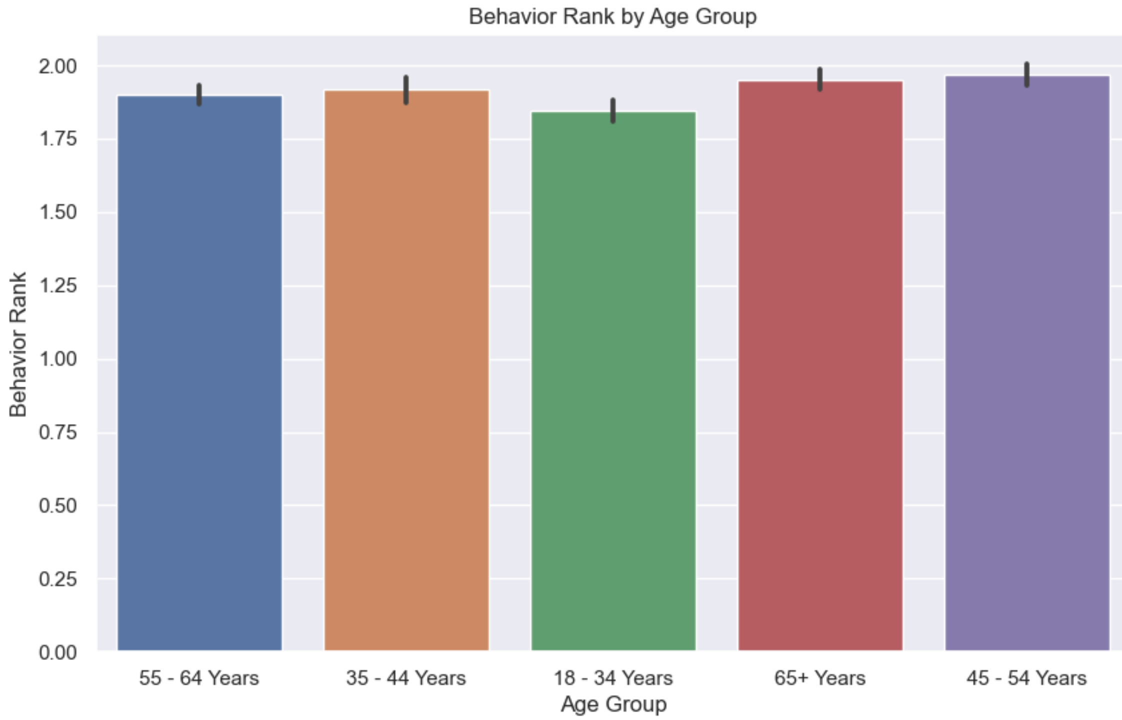
In [48]:
```
df['behav_to_risk'] = (df['behav_score'] + 1) / df['opinion_seas_risk']
check_column(df, 'behav_to_risk')
```



Out[48]:

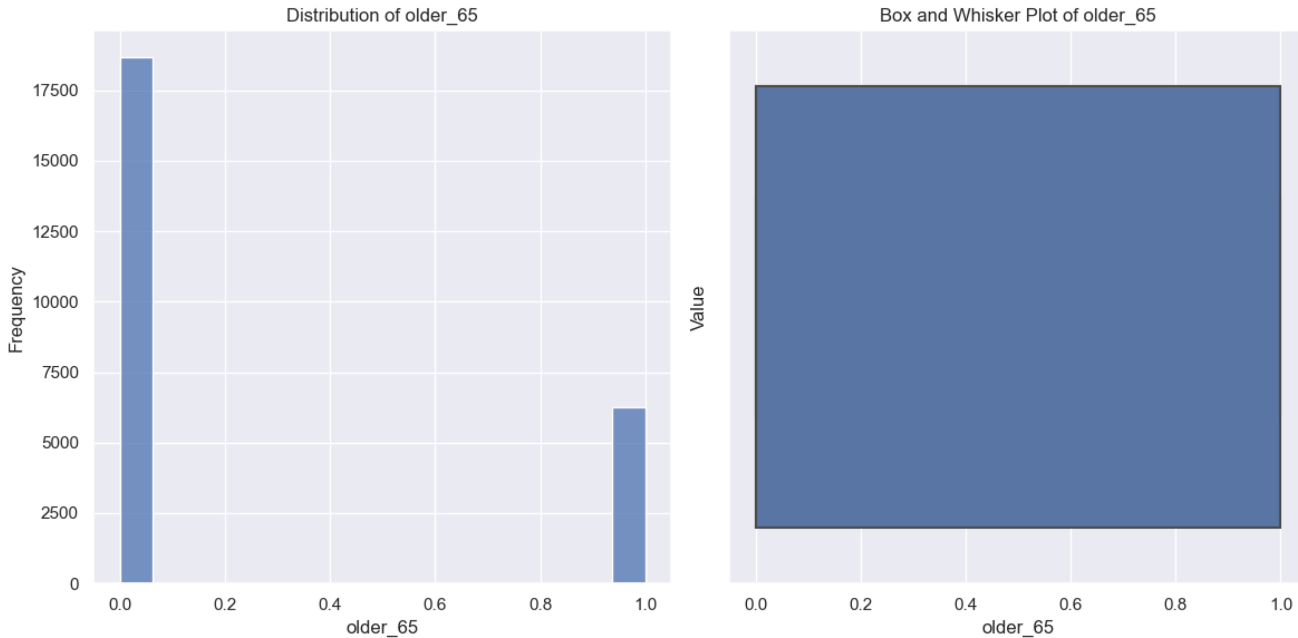
	behav_to_risk	Count
0	1.000	4721
1	2.000	3345
2	1.500	3222
3	3.000	2281
4	4.000	1344
5	1.250	1326
6	2.500	1290
7	0.500	1218
8	0.750	1179
9	6.000	829
10	5.000	748
11	1.200	742
12	0.800	700
13	0.600	365
14	0.250	283
15	1.750	254
16	1.400	217
17	3.500	168
18	0.400	146
19	1.333	142
20	7.000	98
21	0.667	91
22	1.667	77
23	0.200	76
24	0.333	67
25	2.333	10

In [49]:
```
# Plotting 'behavior_rank' and 'age_group'
plt.figure(figsize=(10, 6))
sns.barplot(x='age_group', y='behav_to_risk', data=df)
plt.xlabel('Age Group')
plt.ylabel('Behavior Rank')
plt.title('Behavior Rank by Age Group')
plt.show()
```



In [50]:
```
# Create a function for whether or not an individual is 65 years or older as this
# represents a group at higher risk for serious complications from the flu.
def is_older_65(row):
    if row['age_group'] == '65+ Years':
        return 1
    else:
        return 0
```

In [51]:
```
# 'older_65' variable for whether or not an individual is 65 years or older as this represents a group at higher risk
# for serious complications from the flu.
df['older_65'] = df.apply(lambda x: is_older_65(x), axis=1)

# check counts of unique values in new col and plot distribution
check_column(df, 'older_65')
```



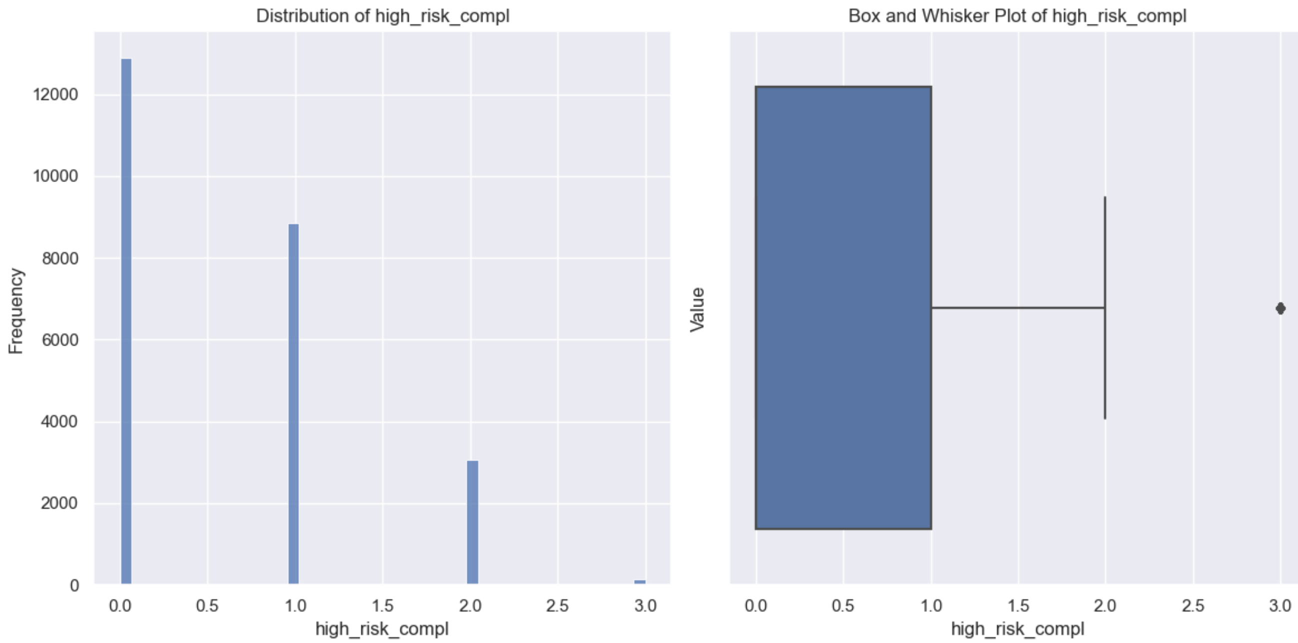
Out[51]:

	older_65	Count
0	0	18678
1	1	6261

- Create a variable 'high_risk_compl' if an individual's overall risk for developing flu-related complications. According to the CDC, people 65 years and older, children 6 months or younger, and people with chronic medical conditions are at higher risk for the flu

In [52]:
```
# function to calculate score for high risk of complications
def calc_high_risk(row):
    risk = 0
    if row['older_65'] == 1:
        risk += 1
    if row['child_under_6_months'] == 1:
        risk += 1
    if row['chronic_med_condition'] == 1:
        risk += 1
    return risk
```

In [53]:
```
# create new column 'high_risk_compl'
df['high_risk_compl'] = df.apply(lambda x: calc_high_risk(x), axis=1)

check_column(df, 'high_risk_compl')
```



Out[53]:

	high_risk_compl	Count
0	0	12894
1	1	8852
2	2	3051
3	3	142

Making a categorical variable that bins people with multiple high-risk factors (high_risk_compl > 1) into one 'high risk' category, assigning 0 to 'low risk' and 1 to 'med risk' Because of the high variations of risk factors per person

In [54]:
```
df['high_risk_cat'] = df['high_risk_compl'].map({0:'low risk', 1:'med risk', 2:'high risk', 3:'high risk'})
df['high_risk_cat'].value_counts()
```

Out[54]:
```
low risk     12894
med risk      8852
high risk     3193
Name: high_risk_cat, dtype: int64
```

In [55]:
```
df['doctor_recc_seasonal'] = df['doctor_recc_seasonal'].map({1.0: '1', 0.0: '0'})
df['doctor_recc_seasonal'].value_counts(dropna=False)
```

Out[55]:
```
0      15420
1       7668
NaN     1851
Name: doctor_recc_seasonal, dtype: int64
```

In [56]:
```
df['health_insurance'] = df['health_insurance'].map({1.0: '1', 0.0: '0'})
df['health_insurance'].value_counts(dropna=False)
```

Out[56]:
```
1      12224
NaN    11043
0       1672
Name: health_insurance, dtype: int64
```

In [57]:
```
# define a function to return make combine people of color
def race_func(row):
    if row['race'] == 'White':
        return 'White'
    else:
        return 'POC'
```

In [58]:
```
df['race'] = df.apply(lambda x: race_func(x), axis=1)
df['race'].value_counts(dropna=False)
```

Out[58]:
```
White    19856
POC       5083
Name: race, dtype: int64
```

In [59]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24939 entries, 0 to 26706
Data columns (total 36 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   h1n1_concern                 24939 non-null  float64
 1   h1n1_knowledge               24939 non-null  float64
 2   behavioral_avoidance         24939 non-null  float64
 3   behavioral_face_mask         24939 non-null  float64
 4   behavioral_wash_hands        24939 non-null  float64
 5   behavioral_large_gatherings  24939 non-null  float64
 6   behavioral_outside_home      24939 non-null  float64
 7   behavioral_touch_face        24939 non-null  float64
 8   doctor_recc_seasonal         23088 non-null  object 
 9   chronic_med_condition        24939 non-null  float64
 10  child_under_6_months         24939 non-null  float64
 11  health_worker                24939 non-null  float64
 12  health_insurance             13896 non-null  object 
 13  opinion_seas_vacc_effective  24939 non-null  float64
 14  opinion_seas_risk            24939 non-null  float64
 15  opinion_seas_sick_from_vacc  24939 non-null  float64
 16  age_group                    24939 non-null  object 
 17  education                    24409 non-null  object 
 18  race                         24939 non-null  object 
 19  sex                          24939 non-null  object 
 20  income_poverty               21621 non-null  object 
 21  marital_status               24409 non-null  object 
 22  rent_or_own                  23816 non-null  object 
 23  employment_status            24361 non-null  object 
 24  hhs_geo_region               24939 non-null  object 
 25  census_msa                   24939 non-null  object 
 26  household_adults             24939 non-null  float64
 27  household_children           24939 non-null  float64
 28  employment_industry          24186 non-null  object 
 29  employment_occupation        24060 non-null  object 
 30  seasonal_vaccine             24939 non-null  int64  
 31  behav_score                  24939 non-null  float64
 32  behav_to_risk                24939 non-null  float64
 33  older_65                     24939 non-null  int64  
 34  high_risk_compl              24939 non-null  int64  
 35  high_risk_cat                24939 non-null  object 
dtypes: float64(18), int64(3), object(15)
memory usage: 7.0+ MB
```

In [60]:
```
# create df with remaining null values filled in with 'missing' for vizualizations
df_missing = df.fillna(value='missing')
df_missing.head()
```

Out[60]:

	h1n1_concern	h1n1_knowledge	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	behavioral_large_gatherings	behavioral_outside_home	be
0	1.000	0.000	0.000	0.000	0.000	0.000	1.000	
1	3.000	2.000	1.000	0.000	1.000	0.000	1.000	
2	1.000	1.000	1.000	0.000	0.000	0.000	0.000	
3	1.000	1.000	1.000	0.000	1.000	1.000	0.000	
4	2.000	1.000	1.000	0.000	1.000	1.000	0.000	

In [61]:
```
df.corr()
```

Out[61]:

	h1n1_concern	h1n1_knowledge	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	behavioral_large_gatherings	b
h1n1_concern	1.000	0.060	0.236	0.159	0.297	0.258	
h1n1_knowledge	0.060	1.000	0.082	0.035	0.087	-0.047	
behavioral_avoidance	0.236	0.082	1.000	0.063	0.337	0.231	
behavioral_face_mask	0.159	0.035	0.063	1.000	0.081	0.178	
behavioral_wash_hands	0.297	0.087	0.337	0.081	1.000	0.193	
behavioral_large_gatherings	0.258	-0.047	0.231	0.178	0.193	1.000	
behavioral_outside_home	0.247	-0.067	0.223	0.164	0.191	0.585	
behavioral_touch_face	0.249	0.084	0.333	0.104	0.364	0.254	
chronic_med_condition	0.096	-0.020	0.040	0.068	0.032	0.104	
child_under_6_months	0.048	0.023	-0.003	0.039	0.035	0.021	
health_worker	0.033	0.170	-0.001	0.069	0.053	-0.033	
opinion_seas_vacc_effective	0.235	0.081	0.116	0.044	0.139	0.080	
opinion_seas_risk	0.333	0.076	0.130	0.110	0.173	0.133	
opinion_seas_sick_from_vacc	0.223	-0.063	0.084	0.093	0.089	0.136	
household_adults	-0.019	0.018	0.015	0.013	0.004	-0.035	
household_children	0.050	0.048	0.038	0.003	0.043	-0.011	
seasonal_vaccine	0.160	0.121	0.079	0.051	0.114	0.065	
behav_score	0.394	0.040	0.616	0.336	0.579	0.704	
behav_to_risk	-0.027	-0.045	0.275	0.101	0.244	0.319	
older_65	0.018	-0.123	-0.020	0.002	-0.002	0.092	
high_risk_compl	0.090	-0.078	0.012	0.059	0.032	0.128	

In [62]:
```
feats_to_drop = ['older_65', 'high_risk_compl']
df.drop(columns=feats_to_drop, axis=1, inplace=True)
df.head()
```

Out[62]:

	h1n1_concern	h1n1_knowledge	behavioral_avoidance	behavioral_face_mask	behavioral_wash_hands	behavioral_large_gatherings	behavioral_outside_home	be
0	1.000	0.000	0.000	0.000	0.000	0.000	1.000	
1	3.000	2.000	1.000	0.000	1.000	0.000	1.000	
2	1.000	1.000	1.000	0.000	0.000	0.000	0.000	
3	1.000	1.000	1.000	0.000	1.000	1.000	0.000	
4	2.000	1.000	1.000	0.000	1.000	1.000	0.000	

We need to process numerical and categorical variables differently, but right now some categorical variables are still showing up as numeric because NaNs haven't been filled in with 'missing'. This can be done as part of the preprocessing pipeline.

In [63]:
```
null_df = calculate_null_percentage(df)
miss_val_cols = list(null_df.loc[null_df['% Null']>0].index)
miss_val_cols
```

Out[63]:
```
['doctor_recc_seasonal',
 'health_insurance',
 'education',
 'income_poverty',
 'marital_status',
 'rent_or_own',
 'employment_status',
 'employment_industry',
 'employment_occupation']
```

these all need to have null values filled with 'missing' so they will all be changed to categorical features

In [64]:
```
# list of all columns that are currently a object
obj_cols = list(df.select_dtypes('O').columns)
cat_cols = list(set(obj_cols + miss_val_cols))
cat_cols
```

Out[64]:
```
['census_msa',
 'employment_industry',
 'high_risk_cat',
 'income_poverty',
 'sex',
 'health_insurance',
 'employment_status',
 'employment_occupation',
 'education',
 'doctor_recc_seasonal',
 'age_group',
 'marital_status',
 'hhs_geo_region',
 'race',
 'rent_or_own']
```

In [65]:
```
num_cols = [col for col in df.drop('seasonal_vaccine', axis=1).columns if col not in cat_cols]
num_cols
```

Out[65]:
```
['h1n1_concern',
 'h1n1_knowledge',
 'behavioral_avoidance',
 'behavioral_face_mask',
 'behavioral_wash_hands',
 'behavioral_large_gatherings',
 'behavioral_outside_home',
 'behavioral_touch_face',
 'chronic_med_condition',
 'child_under_6_months',
 'health_worker',
 'opinion_seas_vacc_effective',
 'opinion_seas_risk',
 'opinion_seas_sick_from_vacc',
 'household_adults',
 'household_children',
 'behav_score',
 'behav_to_risk']
```

In [66]:
```
# define target variable
target = 'seasonal_vaccine'

# separate of features (X) and target (y) for train-test-split
X = df.drop(columns=target, axis=1).copy()
y = df[target].copy()

# split the data into training and test sets prior to preprocessing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

## check for class imbalance across all sets of y
print('**original**\n', y.value_counts(normalize=True), '\n------\n')
print('**y_train**\n', y_train.value_counts(normalize=True), '\n------\n')
print('**y_test**\n', y_test.value_counts(normalize=True), '\n------\n')
```

```
**original**
 0    0.531
1    0.469
Name: seasonal_vaccine, dtype: float64
------

**y_train**
 0    0.534
1    0.466
Name: seasonal_vaccine, dtype: float64
------

**y_test**
 0    0.523
1    0.477
Name: seasonal_vaccine, dtype: float64
------
```

LogisticRegression

In [67]:
```
# transforming numerical columns
num_transformer = Pipeline(steps = [('scaler', StandardScaler())])


# transforming categorical columns and missing

cat_transformer = Pipeline(steps = [('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
                                    ('encoder', OneHotEncoder(handle_unknown='ignore',
                                                              sparse_output=False))])


preprocessing = ColumnTransformer(transformers=[('num', num_transformer, num_cols),
                                                ('cat', cat_transformer, cat_cols)])


model1 = Pipeline([('preproc', preprocessing), ('model', LogisticRegression())])

model1.fit(X_train,y_train)

print(model1.score(X_test,y_test))
preds = model1.predict(X_test)
confusion_matrix(y_test,preds)
```

```
0.7825180433039294

/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[67]:
```
array([[2636,  628],
       [ 728, 2243]])
```

In [68]:
```
param_grid = {
    'model__penalty': ['l1', 'l2'],
    'model__C': [0.1, 1.0, 10.0]
}

grid_search = GridSearchCV(model1, param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_

print("Best parameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)

best_model_score = best_model.score(X_test, y_test)
print("Best model score:", best_model_score)

best_preds = best_model.predict(X_test)
confusion_matrix(y_test, best_preds)
```

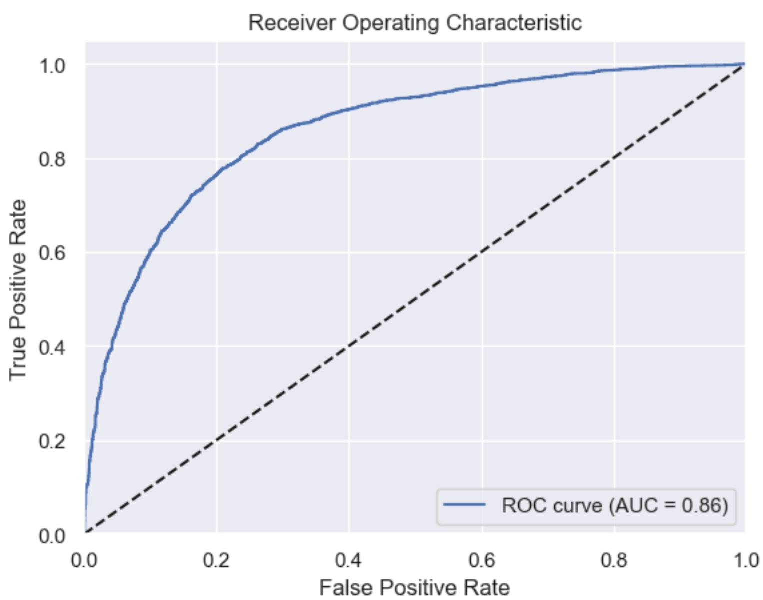
```
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/prep in
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
```

```
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
15 fits failed out of a total of 30.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
```

```
15 fits failed with the following error:
Traceback (most recent call last):
  File "/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/model_selection/_validation.py", line 686, in
_fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/pipeline.py", line 405, in fit
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py", line 54, in _check
_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/model_selection/_search.py:952: UserWarning: One or mo
re of the test scores are non-finite: [       nan 0.78090207        nan 0.78116945        nan 0.78090216]
  warnings.warn(

Best parameters: {'model__C': 1.0, 'model__penalty': 'l2'}
Best score: 0.7811694519609986
Best model score: 0.7825180433039294

/Users/jdapeman/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[68]:
```
array([[2636,  628],
       [ 728, 2243]])
```

In [69]:
```
# Compute predicted probabilities for positive class
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

probs = best_model.predict_proba(X_test)[:, 1]

# Compute ROC curve values
fpr, tpr, thresholds = roc_curve(y_test, probs)

# Compute AUC score
auc_score = roc_auc_score(y_test, probs)

# Plot ROC curve
plt.plot(fpr, tpr, label='ROC curve (AUC = {:.2f})'.format(auc_score))
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```



RandomForestClassifier

In [70]:
```
# transforming numerical columns
num_transformer = Pipeline(steps = [('scaler', StandardScaler())])


# transforming categorical columns and missing

cat_transformer = Pipeline(steps = [('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
                                    ('encoder', OneHotEncoder(handle_unknown='ignore',
                                                              sparse_output=False))])


preprocessing = ColumnTransformer(transformers=[('num', num_transformer, num_cols),
                                                ('cat', cat_transformer, cat_cols)])


model12 = Pipeline([('preproc', preprocessing), ('model', RandomForestClassifier())])

model12.fit(X_train,y_train)

print(model12.score(X_test,y_test))
preds = model12.predict(X_test)
confusion_matrix(y_test,preds)
```

```
0.7797914995990377
```

Out[70]:
```
array([[2662,  602],
       [ 771, 2200]])
```

In [71]:
```python
param_grid = {
    'model__criterion': ['gini', 'entropy'],
    'model__max_depth': [1, 2, 5, 10],
    'model__min_samples_split': [1, 5, 10, 20]
}

gs_RFC = GridSearchCV(model12, param_grid, cv=3)
gs_RFC.fit(X_train,y_train)

gs_RFC.best_params_
```

Out[71]:
```
{'model__criterion': 'entropy',
 'model__max_depth': 10,
 'model__min_samples_split': 5}
```

KNeighborsClassifier

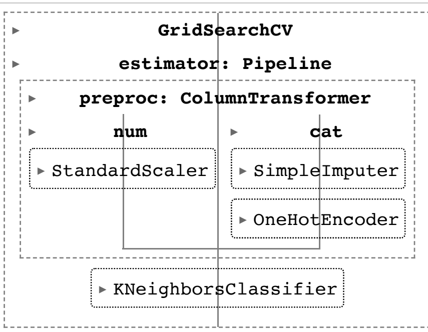
In [72]:
```python
# transforming numerical columns
num_transformer = Pipeline(steps = [('scaler', StandardScaler())])


# transforming categorical columns and missing

cat_transformer = Pipeline(steps = [('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
                                    ('encoder', OneHotEncoder(handle_unknown='ignore',
                                                              sparse_output=False))])


preprocessing = ColumnTransformer(transformers=[('num', num_transformer, num_cols),
                                                ('cat', cat_transformer, cat_cols)])


model13 = Pipeline([('preproc', preprocessing), ('model', KNeighborsClassifier())])

model13.fit(X_train,y_train)

print(model1.score(X_test,y_test))
preds = model1.predict(X_test)
confusion_matrix(y_test,preds)
```

```
0.7825180433039294
```

Out[72]:
```
array([[2636,  628],
       [ 728, 2243]])
```

In [73]:
```python
# fitting the model for grid search

param_grid = {
    'model__n_neighbors': range(2,10,2)}

gs_knn = GridSearchCV(model13, param_grid, cv=3)
gs_knn.fit(X_train,y_train)
```

Out[73]:

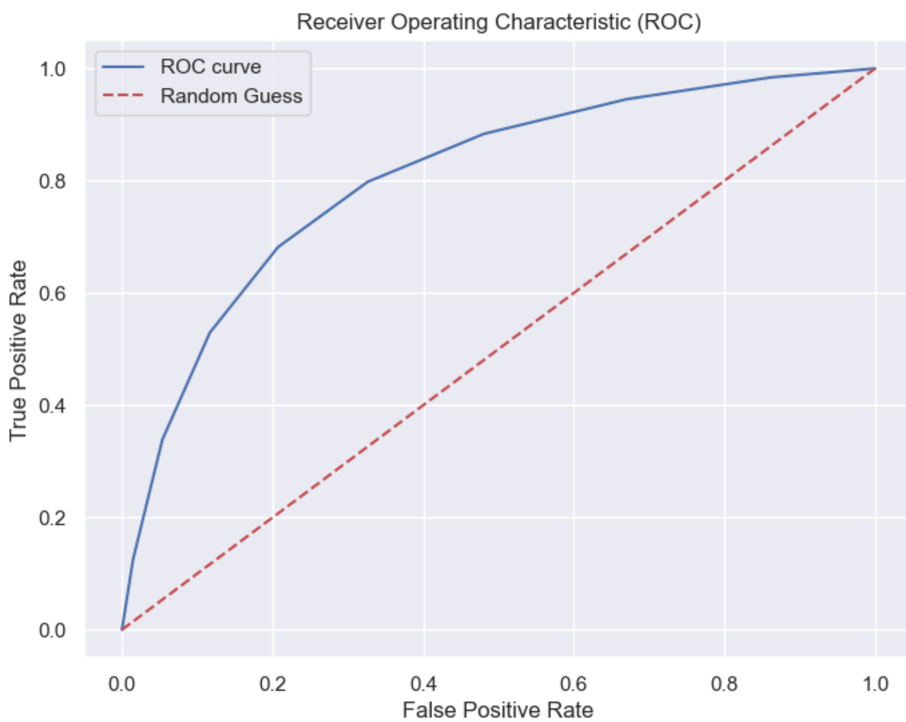


In [74]:
```python
print(gs_knn.best_params_)
gs_knn.best_score_
```

```
{'model__n_neighbors': 8}
```

Out[74]:
```
0.7323563505680664
```

In [75]:
```
# plot a ROC curve for the KNeighborsClassifier

# Get predicted probabilities for the positive class
probs = gs_knn.predict_proba(X_test)[:, 1]

# Calculate the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, probs)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'r--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend()
plt.show()
```



DecisionTreeClassifier

In [76]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, roc_curve, auc
from sklearn.preprocessing import OneHotEncoder
from sklearn import tree

# transforming numerical columns
num_transformer = Pipeline(steps = [('scaler', StandardScaler())])


# transforming categorical columns and missing

cat_transformer = Pipeline(steps = [('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
                                    ('encoder', OneHotEncoder(handle_unknown='ignore',
                                                              sparse_output=False))])


preprocessing = ColumnTransformer(transformers=[('num', num_transformer, num_cols),
                                                ('cat', cat_transformer, cat_cols)])


model14 = Pipeline([('preproc', preprocessing), ('model', DecisionTreeClassifier())])

model14.fit(X_train,y_train)

print(model1.score(X_test,y_test))
preds = model1.predict(X_test)
confusion_matrix(y_test,preds)
```

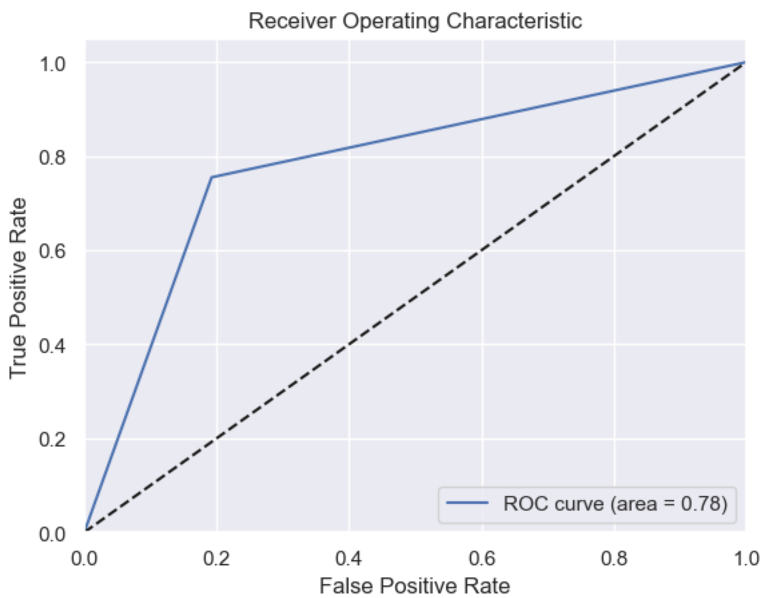
```
0.7825180433039294
```

Out[76]:
```
array([[2636,  628],
       [ 728, 2243]])
```

In [77]:
```


param_grid = {
    'model__criterion': ['gini', 'entropy'],
    'model__max_depth': [1, 2, 5, 10],
    'model__min_samples_split': [1, 5, 10, 20]
}

gs_tree = GridSearchCV(model14, param_grid, cv=3)
gs_tree.fit(X_train,y_train)

gs_tree.best_params_
```

Out[77]:
```
{'model__criterion': 'gini',
 'model__max_depth': 5,
 'model__min_samples_split': 20}
```

In [78]:
```
# plot a ROC curve for the Decision tree classifier
fpr, tpr, thresholds = roc_curve(y_test, preds)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



In [79]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score
```

Feature Importance

best model: LogisticRegression

In [80]:
```
feature_importance = model1.named_steps['model'].coef_[0]
feature_names = model1.named_steps['preproc'].transformers_[1][1].named_steps['encoder'].get_feature_names_out(cat_cols

# Create a DataFrame to store the feature importance
importance_df = pd.DataFrame({'Feature': np.concatenate((num_cols, feature_names)), 'Importance': feature_importance})

# Sort the DataFrame by importance values in descending order
importance_df = importance_df.sort_values('Importance', ascending=False)

# Select the top 5 features
top_features = importance_df.head(15)

# Print the top 5 features and their importance values
print(top_features)
```

```
                          Feature  Importance
26      employment_industry_haxffmxo       1.530
63    employment_occupation_dcjcmpih       1.530
91            doctor_recc_seasonal_1       0.894
97               age_group_65+ Years       0.841
12                 opinion_seas_risk       0.711
31      employment_industry_msuufmds       0.692
11       opinion_seas_vacc_effective       0.646
84    employment_occupation_xzmlyyjv       0.411
54              health_insurance_1       0.362
34      employment_industry_phxvnwax       0.352
25      employment_industry_fcxhlnwr       0.237
29      employment_industry_mfikgejo       0.206
96           age_group_55 - 64 Years       0.200
105          hhs_geo_region_kbazzjca       0.191
21      employment_industry_arjwrbjb       0.190
```

In [81]:
```
# Print top_features on a bar graph and selecting the relevant features
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=top_features)
plt.title('Top 5 Features')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```