petercvuong / Phase_3_Project Public

main

Phase_3_Project / Project 3 Final Notebook.ipynb

petercvuong Ran final notebook again to minimize output of some cells History

1 contributor

1.12 MB

Centers for Disease Control and Prevention

Contents

1. Business Understanding

1. EDA and Data Cleaning

1. Modeling
   3.1 Dummy Model
   3.2 DecisionTree Classifier
   3.3 LogisticRegression Classifier

1. Results

1. Conclusion

1. Recommendations

1. Future Considerations

1. Business Understanding

Tackling Flu and Vaccine Misinformation for the CDC

The mission statement of the CDC is **"to promote health and quality of life by preventing and controlling disease, injury, and disability."**

And so, an integral part of fulfilling this mission is to provide the population with the appropriate infromation that is needed in order to make the best informed decisions for their health.

Disease and vaccine misinformation are a major hurdle for the CDC. Lack of information or knowledge of the wrong information can lead to misinformed decisions that will compromise not only a single individual's health and safety but the population's as well.

The main goal of our project is to predict what features are most influential in determining a respondent's knowledge of the H1N1 flu and vaccine. Once we identify what features are most influential to a respondent's knowledge, we can focus on these features as areas of that the CDC should focus on in order to minimize the dangers of misinformation.

2. EDA and Data Cleaning

Data and Limitations

The data that we will be working with in this project comes from the DrivenData website. This data is a survey response from 2009.

Since this data is from a survey response, it does not give an accurate reflection of the situation and community at the time. The data also has a pretty obvious class imbalance and some clear biases.

Loading in the Data and some Initial EDA

We are looking at the vaccine data from 2009 about H1N1 flu and vaccine awareness.

The data survey response data that we got from DrivenData had about 27,000 recorded responses.

In [1]:

```
# Importing the appropriate libraries that will l
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.feature_selection import RFE
from sklearn.preprocessing import PolynomialFeatu
import statsmodels
```

```
from statsmodels.formula.api import ols
from sklearn.dummy import DummyRegressor

from scipy import stats
from sklearn.preprocessing import OneHotEncoder
from folium.plugins import FastMarkerCluster

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegress
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_sp
from sklearn.metrics import accuracy_score, reca
from sklearn.metrics import plot_confusion_matri
from sklearn.metrics import roc_auc_score, plot_
```

In [2]:

```
# Loading in the pre-split datasets that were gi
vaccinetrainingdf = pd.read_csv("data/training_s
vaccinetestdf = pd.read_csv("data/test_set_featu
vaccinelabelsdf = pd.read_csv("data/training_set
```

In [3]:

```
# Initial checking to see what data types we are
# Commenting outputs out from notebook to reduce
## vaccinetrainingdf.info()
```

As we can see there are some missing values, and some of the values are objects. Since some of the values in these columns are objects, we know that we have to one hot encode the values in order to implement them mathematically into our models.

In [4]:

```
vaccinetrainingdf.head()
```

Out[4]:

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

5 rows × 36 columns

In [5]:

```
# We see here that the data is already split alm
# Commenting outputs out from notebook to reduce
## vaccinetestdf.info()
```

In [6]:

```
# Utilizing this block of code just to display a
pd.set_option('max_columns', None)
vaccinetestdf.head()
```

Out[6]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavi
0	26707	2.0	2.0	
1	26708	1.0	1.0	
2	26709	2.0	2.0	
3	26710	1.0	1.0	
4	26711	3.0	1.0	

In [7]:

```
vaccinelabelsdf.info()
```

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

Next, we're going to be doing some data cleaning to get rid of any extraneous columns that we determined were not relevant to our business problem of tackling misinformation in flu and vaccine awareness.

In [8]:

```
# Dropping data we deemed unnecessary and irrele
columns_to_drop = ['respondent_id','h1n1_knowled
                   'employment_status', 'rent_o
                   'health_worker']
X_train = vaccinetrainingdf.copy().drop(columns_
X_test = vaccinetestdf.copy().drop(columns_to_dr
```

```
# Setting the y_train and y_test to just be the \
y_train = vaccinetrainingdf['h1n1_knowledge']
y_test = vaccinetestdf['h1n1_knowledge']
```

In [9]:

```
# Checking our data again to see that we dropped
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26707 entries, 0 to 26706
Data columns (total 20 columns):
 #   Column                       Non-Null Count
Dtype
---  ------                       --------------
-----
 0   h1n1_concern                 26615 non-null
float64
 1   behavioral_antiviral_meds    26636 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_h1n1             24547 non-null
float64
 9   doctor_recc_seasonal         24547 non-null
float64
 10  chronic_med_condition        25736 non-null
float64
 11  opinion_h1n1_vacc_effective  26316 non-null
float64
 12  opinion_h1n1_risk            26319 non-null
float64
 13  opinion_h1n1_sick_from_vacc  26312 non-null
float64
 14  opinion_seas_vacc_effective  26245 non-null
float64
 15  age_group                    26707 non-null
object
 16  education                    25300 non-null
object
 17  race                         26707 non-null
object
 18  sex                          26707 non-null
object
 19  income_poverty               22284 non-null
object
dtypes: float64(15), object(5)
memory usage: 4.1+ MB
```

In [10]:

```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26708 entries, 0 to 26707
Data columns (total 20 columns):
 #   Column                       Non-Null Count
Dtype
---  ------                       --------------
-----
 0   h1n1_concern                 26623 non-null
float64
 1   behavioral_antiviral_meds    26629 non-null
float64
 2   behavioral_avoidance         26495 non-null
float64
 3   behavioral_face_mask         26689 non-null
float64
 4   behavioral_wash_hands        26668 non-null
float64
 5   behavioral_large_gatherings  26636 non-null
float64
 6   behavioral_outside_home      26626 non-null
float64
 7   behavioral_touch_face        26580 non-null
float64
 8   doctor_recc_h1n1             24548 non-null
float64
 9   doctor_recc_seasonal         24548 non-null
float64
 10  chronic_med_condition        25776 non-null
float64
 11  opinion_h1n1_vacc_effective  26310 non-null
float64
 12  opinion_h1n1_risk            26328 non-null
float64
 13  opinion_h1n1_sick_from_vacc  26333 non-null
float64
 14  opinion_seas_vacc_effective  26256 non-null
float64
 15  age_group                    26708 non-null
object
 16  education                    25301 non-null
object
 17  race                         26708 non-null
object
 18  sex                          26708 non-null
object
 19  income_poverty               22211 non-null
object
dtypes: float64(15), object(5)
memory usage: 4.1+ MB
```

Feature Engineering

Created some frequently used functions that we will be utilizing throughout our project

In [11]:

```
# Defined a OneHotEncoder function for ease of a
def OHE(X_train, categories):
```

```
    onehot = OneHotEncoder(sparse=False, handle_
    x_train_cat = pd.DataFrame(onehot.fit_transf
    x_train_cat.columns = onehot.get_feature_nam

    # Reset indices to avoid merging conflicts
    x_train_cat.reset_index(drop=True, inplace=T
    X_train.reset_index(drop=True, inplace=True)

    # Joined the OHE dataframe to the dataframe
    x_train_df = X_train.drop(categories, axis =
    return x_train_df

# Defined a function that takes in parameters to
def confusion_and_metrics(model, X_test, y_test,
    # Accuracy Score
    print(f"Accuracy Score: {model.score(X_test,

    # Precision Score
    print(f"Precision Score: {precision_score(y_

    # Plot confusion matrix for visualization
    plot_confusion_matrix(model, X_test, y_test)

# Defined a function to take in column name and
def print_odds(dataframe, column_name):
    # Prints out the name of the column and it's
    print(f"{column_name}: {dataframe[column_nam

    # Prints out the odds value of the column
    print(f"Odds: {np.exp(dataframe[column_name]
```

In [12]:

```
# X_train
```

SimpleImputer to Account for NaN Values

Prior to running some classification models on our data, we looked at it again and noticed that there were still a couple missing values.

In order to rectify this, we created a simple imputer to replace the NaN values with the most frequent value(otherwise known as the mode) in its respective column.

We chose to use the mode to replace these NaN values because using the mode will keep the distribution of the data consistent.

In [13]:

```
# Created a SimpleImputer to replace the NaN valu
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy = 'most_frequen
imputed_X_train = imputer.fit_transform(X_train)
```

```
imputed_X_train_df = pd.DataFrame(imputed_X_trai
```

In [14]:

```
imputed_X_test = imputer.transform(X_test)
imputed_X_test_df = pd.DataFrame(imputed_X_test)
imputed_X_test_df
```

Out[14]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	2	0	1	0	1	1	0	1	0	0	0	5	1	1
1	1	0	0	0	0	0	0	0	0	0	0	4	1	1
2	2	0	0	1	1	1	1	1	0	0	0	5	4	2
3	1	0	0	0	0	0	0	0	1	1	1	4	2	2
4	3	1	1	0	1	1	1	1	0	0	0	5	2	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
26703	1	0	1	0	1	0	0	1	1	1	0	4	2	2
26704	3	0	1	0	1	1	1	1	0	0	0	4	1	1
26705	0	0	0	0	0	0	0	0	0	0	0	4	3	1
26706	3	0	1	0	1	0	1	0	0	0	0	2	3	4
26707	2	0	0	0	1	0	0	1	1	0	0	5	1	2

26708 rows × 20 columns

In [15]:

```
# After doing the imputation and renaming, checke
imputed_X_test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26708 entries, 0 to 26707
Data columns (total 20 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       26708 non-null  object
 1   1       26708 non-null  object
 2   2       26708 non-null  object
 3   3       26708 non-null  object
 4   4       26708 non-null  object
 5   5       26708 non-null  object
 6   6       26708 non-null  object
 7   7       26708 non-null  object
 8   8       26708 non-null  object
 9   9       26708 non-null  object
 10  10      26708 non-null  object
 11  11      26708 non-null  object
 12  12      26708 non-null  object
 13  13      26708 non-null  object
 14  14      26708 non-null  object
 15  15      26708 non-null  object
 16  16      26708 non-null  object
 17  17      26708 non-null  object
 18  18      26708 non-null  object
 19  19      26708 non-null  object
dtypes: object(20)
memory usage: 4.1+ MB
```

After imputing the data, we recognize that the column names have disappeared but the indices are still there. To resolve this, we create a dictionary with the original column names and call the rename function on this new data frame's columns.

In [16]:

```python
# Extracting column names into a dictionary
dictionary_of_names = {columns: index for index,

# Flipping the column keys and values
dictionary_of_names_flipped = {dictionary_of_name

# Checking to see if the column names were extrac
dictionary_of_names_flipped
```

Out[16]:

```
{0: 'h1n1_concern',
 1: 'behavioral_antiviral_meds',
 2: 'behavioral_avoidance',
 3: 'behavioral_face_mask',
 4: 'behavioral_wash_hands',
 5: 'behavioral_large_gatherings',
 6: 'behavioral_outside_home',
 7: 'behavioral_touch_face',
 8: 'doctor_recc_h1n1',
 9: 'doctor_recc_seasonal',
 10: 'chronic_med_condition',
 11: 'opinion_h1n1_vacc_effective',
 12: 'opinion_h1n1_risk',
```

```
13: 'opinion_h1n1_sick_from_vacc',
14: 'opinion_seas_vacc_effective',
15: 'age_group',
16: 'education',
17: 'race',
18: 'sex',
19: 'income_poverty'}
```

In [17]:

```
# Created new variables for the training and tes
imputed_X_train_df_plus_column_names = imputed_X_
imputed_X_test_df_plus_column_names = imputed_X_
```

In [18]:

```
# Calling the new dataframe variables to check i
imputed_X_test_df_plus_column_names
```

Out[18]:

	h1n1_concern	behavioral_antiviral_meds	behaviora
0	2	0	
1	1	0	
2	2	0	
3	1	0	
4	3	1	
...	...	...	
26703	1	0	
26704	3	0	
26705	0	0	
26706	3	0	
26707	2	0	

26708 rows × 20 columns

In [19]:

```
imputed_X_train_df_plus_column_names
```

Out[19]:

	h1n1_concern	behavioral_antiviral_meds	behaviora
0	1	0	

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

26707 rows × 20 columns

In [20]:

```
X_train
```

Out[20]:

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

26707 rows × 20 columns

After doing some initial data cleaning and making sure that our data was uniform, we next want to address the problem of having the `object` type in our columns. In order to address this, we apply a OneHotEncoder to these object columns of `age_group, education, race, sex, and income_poverty`.

We check the values in each of these object columns to see how many variables will be OneHotEncoded.

In [21]:
```
X_train['age_group'].value_counts()
```

Out[21]:
```
65+ Years        6843
55 - 64 Years    5563
45 - 54 Years    5238
18 - 34 Years    5215
35 - 44 Years    3848
Name: age_group, dtype: int64
```

In [22]:
```
X_train['education'].value_counts()
```

Out[22]:
```
College Graduate    10097
Some College         7043
12 Years             5797
< 12 Years           2363
Name: education, dtype: int64
```

In [23]:
```
X_train['race'].value_counts()
```

Out[23]:
```
White                21222
Black                 2118
Hispanic              1755
Other or Multiple     1612
Name: race, dtype: int64
```

In [24]:
```
X_train['sex'].value_counts()
```

Out[24]:
```
Female    15858
Male      10849
Name: sex, dtype: int64
```

In [25]:
```
X_train['income_poverty'].value_counts()
```

Out[25]:
```
<= $75,000, Above Poverty    12777
> $75,000                     6810
Below Poverty                 2697
Name: income_poverty, dtype: int64
```

```
Name: income_poverty, dtype: int64
```

After counting the values, we see that **18** columns will be added

In [26]:

```
# Called the OHE function we made and assigned ne
ohe_training_df = OHE(imputed_X_train_df_plus_co]
ohe_test_df = OHE(imputed_X_test_df_plus_column_i
ohe_training_df
```

Out[26]:

	h1n1_concern	behavioral_antiviral_meds	behaviora
0	1	0	
1	3	0	
2	1	0	
3	1	0	
4	2	0	
...	...	...	
26702	2	0	
26703	1	0	
26704	2	0	
26705	1	0	
26706	0	0	

26707 rows × 33 columns

We finished OneHotEncoding the object values and now we have to bin the target values.

Based on the data dictionary, our target values reside in the `h1n1_knowledge` column where the responses are recorded as such:

- `0 = No knowledge`
- `1 = A Little Knowledge`
- `2 = A Lot of Knowledge`

For our project, we are going to bin the 0s and 1s together because those who respond as having little to no knowledge of the flu and vaccine are most prone to misinformation.

We will then be turning all the 2 responses into 1s so that we have a simple binary categorization where:

- `0 = Little/No Knowledge`
- `1 = A Lot of Knowledge`

In [27]:
```
# Instead of calling SimpleImputer and removing t
# imputation which replaced all the NaN values wi
# which in this case would be 1.0 (little knowled
y_train.replace(np.nan, 1.0, inplace = True)
y_test.replace(np.nan, 1.0, inplace = True)
```

In [28]:
```
# Checking to see if we replaced the NaN values c
y_test.isna().value_counts()
```

Out[28]:
```
False    26708
Name: h1n1_knowledge, dtype: int64
```

In [29]:
```
y_train.isna().value_counts()
```

Out[29]:
```
False    26707
Name: h1n1_knowledge, dtype: int64
```

In [30]:
```
# Binning all the 1.0s with the 0.0s
y_train.replace(1.0, 0.0, inplace = True)
y_test.replace(1.0, 0.0, inplace = True)
# Replacing all the 2.0s with 1.0s
y_train.replace(2.0, 1.0, inplace = True)
y_test.replace(2.0, 1.0, inplace = True)
```

In [31]:
```
# Checking to see if we replaced our values corre
y_train.value_counts()
```

Out[31]:
```
0.0    17220
1.0     9487
Name: h1n1_knowledge, dtype: int64
```

In [32]:
```
y_test.value_counts()
```

Out[32]:
```
0.0    17193
1.0     9515
Name: h1n1_knowledge, dtype: int64
```

In [33]:
```
# Commenting outputs out from notebook to reduce
# ohe_training_df.info()
```

SMOTE for Class Imbalance

After we bin our target and features together, we recognize that our target class is severely imbalanced.

To address thisn class imbalance, we implement SMOTE to undersample our 0 class.

In [34]:

```
# Since our data is severly imbalanced, we utili
# Since we SMOTE our training dataset, we must S

from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSa
from collections import Counter

over = SMOTE(sampling_strategy=0.7)
under = RandomUnderSampler(sampling_strategy=0.8

X_smote, y_smote = under.fit_resample(ohe_traini
X_test_smote, y_test_smote = under.fit_resample(

counter = Counter(y_train)
test_counter = Counter(y_test_smote)
print(counter)
print(test_counter)
```

```
Counter({0.0: 17220, 1.0: 9487})
Counter({0.0: 11893, 1.0: 9515})
```

Checking for Preliminary Feature Importance

For the final part of our EDA and Data cleaning, we want to check and see what features are seemingly most important to our respondents.

In [35]:

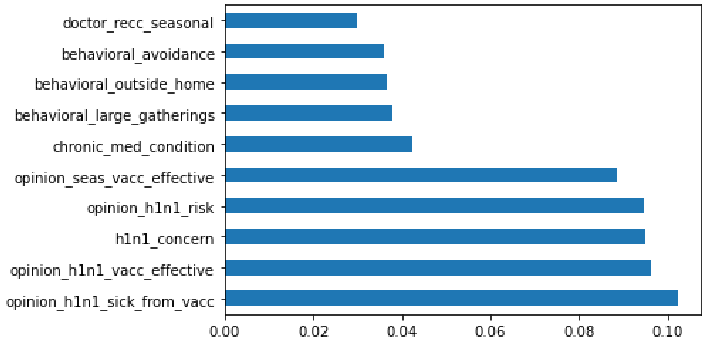
```
# Feature columns
X = ohe_training_df.iloc[:,0:33]
# Target column - H1N1 Knowledge
y = vaccinetrainingdf.iloc[:,2]

from sklearn.ensemble import ExtraTreesClassifie
import matplotlib.pyplot as plt

# Instantiate model
modelfeatures = ExtraTreesClassifier()
modelfeatures.fit(X,y)
print(modelfeatures.feature_importances_) # use i
# Plot graph of feature importances for better v
feat_importances = pd.Series(modelfeatures.featu
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

```
[0.09506911 0.01447003 0.03583344 0.01769623 0.02
254268 0.03772919
 0.03655344 0.02747728 0.02085327 0.02988187 0.04
221946 0.09611108
 0.09469669 0.10224987 0.08850742 0.01540307 0.01
607095 0.01713853
 0.01701695 0.01558146 0.01093478 0.01123703 0.02
```

```
630258 0.00947193
 0.00769741 0.0072257  0.00790023 0.01453193 0.01
239084 0.01235294
 0.01141553 0.01665281 0.0087843 ]
```



We notice that the top 5 features that are most important to survey respondents are:

- `opinion_h1n1_sick_from_vacc`
- `opinion_h1n1_vacc_effective`
- `h1n1_concern`
- `opinion_h1n1_risk`
- `opinion_seas_vacc_effective`

3. Modeling

Now that our data is cleaned, we can go into the modeling.

3.1 Model 1 (Dummy Classifier)

First we want to create a DummyClassifier model that will serve as the baseline for our model performance comparison. A DummyClassifier model in this case would mean that based on the given data, the dummy model would correctly identify our predictions **50%** of the time.

In [36]:

```
# Created Dummy Classifier model to look at simp
from sklearn.dummy import DummyClassifier
dummy = DummyClassifier()
dummy.fit(X_smote, y_smote)
y_pred = dummy.predict(X_smote)
y_test_pred = dummy.predict(X_test_smote)
y_pred_df = pd.DataFrame(y_pred)
dummy.score(X_test_smote, y_test_smote)
```

```
C:\Users\Beter\anaconda3\envs\learn-env\lib\site-
packages\sklearn\dummy.py:131: FutureWarning: The
```

```
packages\sklearn\dummy.py:131: FutureWarning: The
default value of strategy will change from strati
fied to prior in 0.24.
  warnings.warn("The default value of strategy wi
ll change from "
```

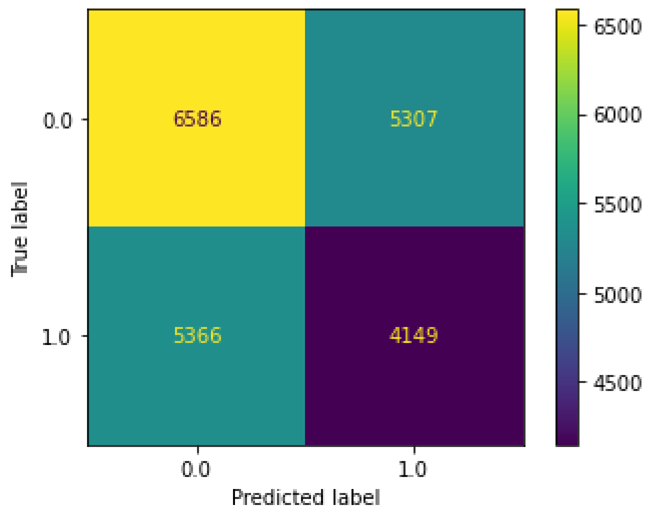
Out[36]:

```
0.5091554559043349
```

In [37]:

```python
# Called function and printed out confusion and p
confusion_and_metrics(dummy, X_test_smote, y_test
```

```
Accuracy Score: 0.508
Precision Score: 0.442
```



We print out the `dummy.score` to see that the accuracy score is about 50%, just as we expected.

We call our `confusion_and_metrics` function that we defined above in order to produce the evaluation metrics of Accuracy and Precision and a confusion matrix for easier visualization.

3.2 Model 2 (Decision Tree Classifier)

Next we will create an inferential DecisionTree Classifier in order to identify our **most important features**. After we identify our most important features, we can then run a classifiying LogisticRegression model to measure our predictions on the dataset.

In [38]:

```python
# Displayed cross validation score for the dummy
from sklearn.model_selection import train_test_sp
```

```
dummy_cross_val = cross_val_score(dummy, X_smote

dummy_cross_val
```

```
C:\Users\Beter\anaconda3\envs\learn-env\lib\site-
packages\sklearn\dummy.py:131: FutureWarning: The
default value of strategy will change from strati
fied to prior in 0.24.
  warnings.warn("The default value of strategy wi
ll change from "
```

Out[38]:

```
array([0.50480206, 0.51628016, 0.50995549, 0.5050
3631, 0.51253221])
```

In [39]:

```
from sklearn.tree import DecisionTreeClassifier,
from sklearn.model_selection import GridSearchCV

dt = DecisionTreeClassifier (random_state = 10)
dt.fit(X_smote, y_smote)
y_dt_pred = dt.predict(X_smote)
y_dt_test_pred = dt.predict(X_test_smote)
dt.score(X_test_smote, y_test_smote)
```

Out[39]:

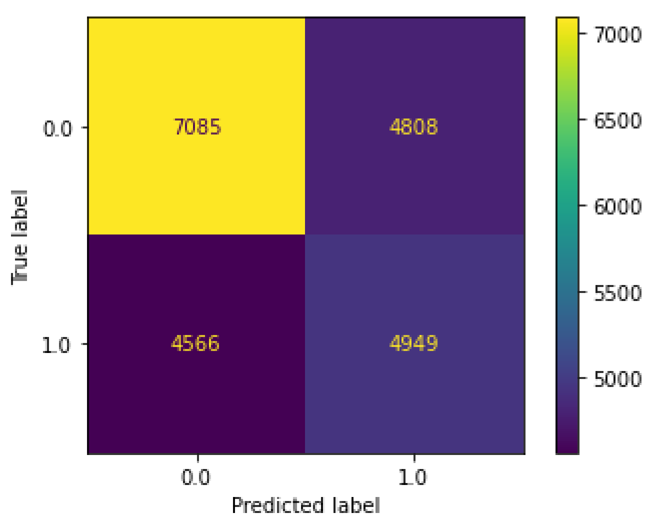
```
0.562126307922272
```

Similarly like how we checked the DummyClassifier, we also check the plot the confusion matrix and check the metrics of our baseline decision tree.

In [40]:

```
# Called function and printed out confusion matr
confusion_and_metrics(dt, X_test_smote, y_test_sı
```

```
Accuracy Score: 0.562
Precision Score: 0.507
```



After running our initial DecisionTree Classifier, we got an accuracy score of about **56%**. As you can see, this accuracy score is only about 6% better than the baseline.

Implementing GridSearchCV to Find Optimal Hyperparameters

We decided to use a `GridSearchCV` in order find the best hyperparameters to pass into our DecisionTree Classifier so that we can find the most important features to focus on.

In [41]:
```
# Created grid paramater to perform a GridSearch
grid = {
    'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'min_samples_split': [100, 500, 1000, 5000],
    'min_samples_leaf': [100, 500, 1000, 5000]
}
# initializing our grid search with the grid para
gs = GridSearchCV(estimator = dt, param_grid = gr
gs.fit(ohe_training_df, y_train)
```

Out[41]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifi
er(random_state=10),
             param_grid={'max_depth': [1, 2, 3,
4, 5, 6, 7, 8, 9, 10],
                         'min_samples_leaf': [10
0, 500, 1000, 5000],
                         'min_samples_split': [10
0, 500, 1000, 5000]})
```

After we run our `GridSearchCV`, we print out the `best_params_`, `best_score_`, and the `best_estimator_` to get the optimal parameters and metrics based on the grid search results.

In [42]:
```
gs.best_params_
```

Out[42]:
```
{'max_depth': 7, 'min_samples_leaf': 100, 'min_sa
mples_split': 100}
```

In [43]:
```
gs.best_score_
```

Out[43]:
```
0.6770884108866996
```

In [44]:
```
gs.best_estimator_.score(ohe_test_df, y_test)
```

Out[44]:
```
0.6696495432080276
```

In [45]:
```
gs.n_features_in_
```

Out[45]:
```
33
```

In [46]:
```
# Converted the results from the GridSearch to a
```

```
# converted the results from the GridSearch to a
pd.DataFrame(gs.cv_results_)
```

Out[46]:

	mean_fit_time	std_fit_time	mean_score_time	std_sco
0	0.052703	0.004094	0.010736	(
1	0.047641	0.001899	0.010346	(
2	0.051506	0.002567	0.011032	(
3	0.055478	0.012545	0.010883	(
4	0.051519	0.003832	0.011048	(
...	...	...	...	
155	0.055637	0.002057	0.009565	(
156	0.050849	0.002426	0.009955	(
157	0.049724	0.003120	0.009994	(
158	0.047685	0.001417	0.009380	(
159	0.049302	0.001878	0.010146	(

160 rows × 16 columns

After we figure out the best parameters, we create a new DecisionTree Classifier and pass in the GridSearchCV results in order to produce the metrics and a confusion matrix for easier visualization.

In [47]:

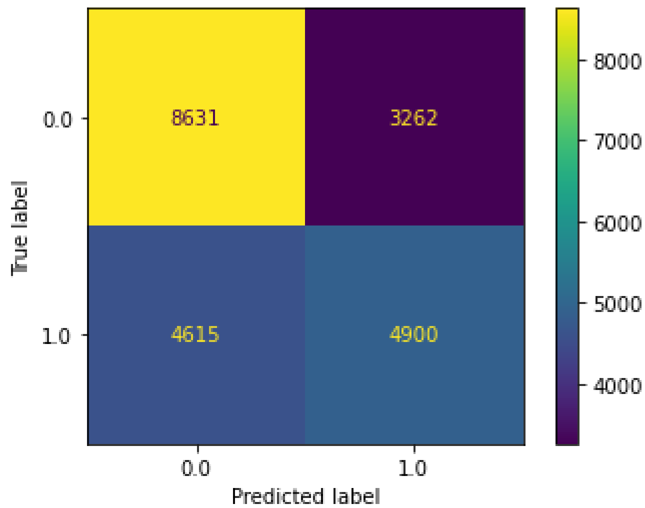
```
dt2 = DecisionTreeClassifier(max_depth = 7, min_
dt2.fit(X_smote, y_smote)
y_dt2_pred = dt.predict(X_smote)
y_dt2_test_pred = dt.predict(X_test_smote)
dt2.score(X_test_smote, y_test_smote)
```

```
0.6320534379671151
```

In [48]:

```
confusion_and_metrics(dt2, X_test_smote, y_test_s
```

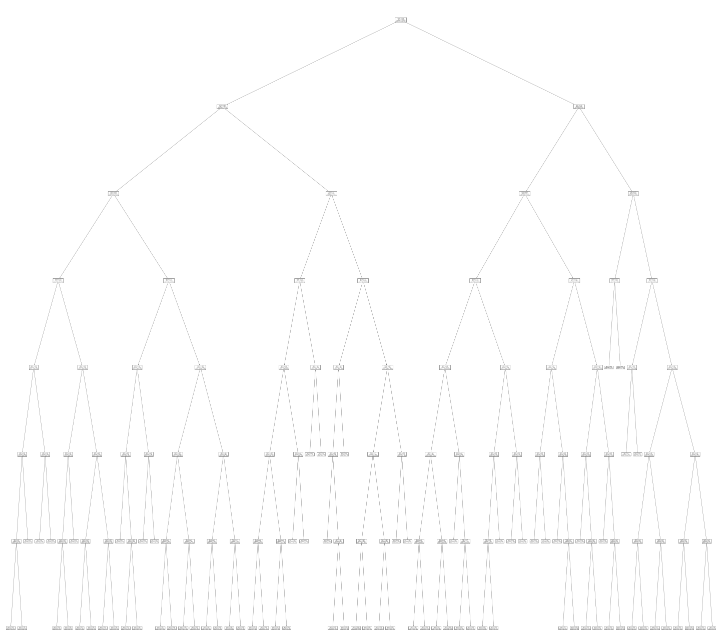
```
Accuracy Score: 0.632
Precision Score: 0.507
```



We also plotted the DecisionTree and exported it out to `out.pdf` for better visualization in a PDF Reader program.

In [49]:

```python
f, ax = plt.subplots(figsize=(100, 100))
plot_tree(dt2, ax=ax);
# plt.savefig('out.pdf')
```



From this DecisionTree, we recognize that our X[21] is one of the most important features for us to split our data. We exported the DecisionTree and took a closer look at the features to split on. Next we will run a LogisticRegression model.

3.3 Model 3 (Logistic Regression)

After running our DecisionTree Classifier, we implemented a `LogisticRegression` model to find our best predicitions on H1N1 knowledge.

In [50]:

```
# Importing the appropriate library
from sklearn.linear_model import LogisticRegress
model = LogisticRegression(random_state=42)
model.fit(X_smote, y_smote)
y_lr_pred = model.predict(X_smote)
y_lr_test_pred = model.predict(X_test_smote)
model.score(X_test_smote, y_test_smote)
```

```
C:\Users\Beter\anaconda3\envs\learn-env\lib\site-
packages\sklearn\linear_model\_logistic.py:762: C
onvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or s
cale the data as shown in:
    https://scikit-learn.org/stable/modules/prepr
ocessing.html
Please also refer to the documentation for altern
ative solver options:
    https://scikit-learn.org/stable/modules/linea
r_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[50]:

```
0.6391068759342302
```

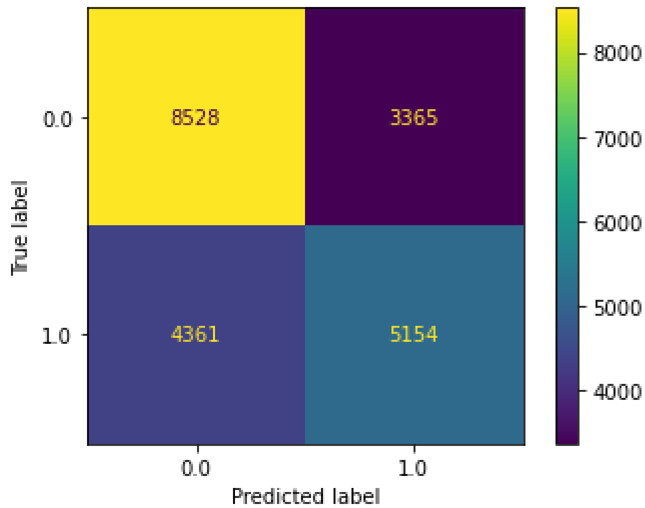
We plot out the confusion matrix and produce the metrics to see that our model is accurate in predicting whether a respondent is knowledgeable about H1N1 or not about **64%** of the time. This is a **14%** increase from our baseline model. The precision of this model also increased about **10%** meaning that our model correctly identifies knowledgeable respondents **60%** of the time.

The precision increase to 60% is important to us because in our models we would like to focus more on those who responded that they are knowledgeable about H1N1 Flu and Vaccine, but in reality they are not knowledgeable at all (*False Negative*).

In [51]:

```
confusion_and_metrics(model, X_test_smote, y_tes
```

```
Accuracy Score: 0.639
Precision Score: 0.605
```



4. Results

Interpreting LogisticRegression Results

We found that our LogisticRegression model produced the highest accuracy score of **64%** and the highest precision score of **60%**.

We want to also identify the coefficients in this array produced by our model. We want to identify the lowest coefficient and take the power of that coefficient in order to produce an odds value.

In the cell below, we are simply extracting the column names from the dataset, inputting them into a dictionary, and flipping the dictionary values. We then rename the columns in this coefficient dataframe to reflect the appropriate changes.

In [52]:

```
# Extracting column names into a dictionary
model_column_names = {c: i for i, c in enumerate

# Flipping the column keys and values
model_column_names_flipped = {model_column_names

# Turning the coefficients array into a datafram
model_coef = pd.DataFrame(model.coef_)
```

```
model_coef.rename(model_column_names_flipped, ax
# Checking to see if the rename was done correct
model_coef.columns
# Those with an education of less than 12 years
```

Out[52]:

```
Index(['h1n1_concern', 'behavioral_antiviral_med
s', 'behavioral_avoidance',
       'behavioral_face_mask', 'behavioral_wash_h
ands',
       'behavioral_large_gatherings', 'behavioral
_outside_home',
       'behavioral_touch_face', 'doctor_recc_h1n
1', 'doctor_recc_seasonal',
       'chronic_med_condition', 'opinion_h1n1_vac
c_effective',
       'opinion_h1n1_risk', 'opinion_h1n1_sick_fr
om_vacc',
       'opinion_seas_vacc_effective', 'age_group_
18 - 34 Years',
       'age_group_35 - 44 Years', 'age_group_45 -
54 Years',
       'age_group_55 - 64 Years', 'age_group_65+
Years', 'education_12 Years',
       'education_< 12 Years', 'education_College
Graduate',
       'education_Some College', 'race_Black', 'r
ace_Hispanic',
       'race_Other or Multiple', 'race_White', 's
ex_Female', 'sex_Male',
       'income_poverty_<= $75,000, Above Povert
y', 'income_poverty_> $75,000',
       'income_poverty_Below Poverty'],
      dtype='object')
```

In [53]:

```
# Printing out new dataframe to check if renaming
model_coef
```

Out[53]:

	h1n1_concern	behavioral_antiviral_meds	behavioral_avo
0	0.034431	0.083164	0.

We identify that from this array, the column of `education_< 12 Years` was our lowest coefficient. To make sense of this number we take the exponent of this value.

In [54]:

```
# Calling function that defined above to intepre
print_odds(model_coef, 'education_< 12 Years')
```

```
education_< 12 Years: -0.7209944344950889
Odds: 0.4862684533209686
```

From this we see that those with an `education_< 12 Years` are about **0.49x** as likely to be knowledgeable about the H1N1 flu and vaccine.

In [55]:

```
print_odds(model_coef, 'income_poverty_Below Pove
```

```
income_poverty_Below Poverty: -0.3401334253652462
7
Odds: 0.7116753608826263
```

From this we can see that respondents who indicate `income_poverty_Below Poverty` are about **0.69x** as likely to be knowledgeable about the H1N1 Flu and Vaccine.

In [56]:

```
print_odds(model_coef, 'sex_Male')
```

```
sex_Male: -0.12456103754841111
Odds: 0.8828843706242121
```

From this we can see that respondents who indicate `sex_Male` are about **0.85x** as likely to be knowledgeable about the H1N1 Flu and Vaccine.

Visualizations

Below we will be graphing some of the important features that can lead to misinformation: **overall knowledge of H1N1** and **concern for H1N1** as they both can lead to misinformation.

In [57]:

```
def change_width(ax, new_value) :
```