FlatIron Phase 3 Project : H1N1 Vaccines

Business Problem

New York State Department of Health wants to increase H1N1 vaccination rate because it struggles to vaccinate the population. New York Department of Health wants to increase future public health effort to increase public vaccination rate by having an understanding of how people's backgrounds, opinions, and health behaviors are related to their personal vaccination patterns.

install library that is needed for this notebook

In [242]:
```
# data analysis and wrangling
import pandas as pd
import numpy as np
import random as rnd
import math
from matplotlib import pyplot as plt
from scipy import stats as stats

# visualization
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
%matplotlib inline

# scaling and train test split
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.preprocessing import MinMaxScaler

# pipeline setup

from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import MaxAbsScaler
from sklearn.compose import ColumnTransformer
from sklearn.impute import KNNImputer
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

# cross validation
from sklearn.model_selection import KFold
# Import the evaluation matrics
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_sco

from sklearn.metrics import precision_score, recall_score, plot_confusion_matrix
from sklearn.model_selection import train_test_split, GridSearchCV,\
cross_val_score, RandomizedSearchCV


from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, roc_curve, auc
from sklearn.preprocessing import OneHotEncoder
from sklearn import tree

# evaluation on test data
from sklearn import metrics
from sklearn.metrics import mean_squared_error,mean_absolute_error,explained_vari
from sklearn.metrics import classification_report,confusion_matrix
# import library for Gradient Boosting
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
```

In [ ]:

```

```

In [243]:

```
file_path = "\\Users\\eggfr\\Flatiron\\Flatiron_phase3_project\\data\\H1N1_Flu_Va
project3_raw_df = pd.read_csv(file_path)
```

Identifying Features and Target and investigate the non vaccinate group.

Once the data is loaded into a pandas dataframe, the next step is identifying which columns represent features and which column represents the target. In the cell below, assign X to be the features and y to be the target. Remember that X should not contain the target.

In [244]:
```
project3_raw_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
```

Data Undestanding and Identifying Features and Target

Once the data is loaded into a pandas dataframe, the next step is identifying which columns represent features and which column represents the target. In this project, we are going to focus on predicting whether people got H1N1 vaccine using data collected in the National 2009 H1N1 Flu Survey which can be found from this link [https://www.kaggle.com/datasets/arashnic/flu-data (https://www.kaggle.com/datasets/arashnic/flu-data)](https://www.kaggle.com/datasets/arashnic/flu-data). In the cell below, assign X to be the features and y to be the target, which is project3_raw_2_df['h1n1_vaccine']. Also, this is not an extremely inbalanced dataset, around 78% of the responses is not vaccinated. For all binary variables: 0 = No; 1 = Yes.

There is 26707 total rows of data. There is 36 columns of features. The first column respondent_id is a unique and random identifier. The remaining 35 features are described below. In this data set, seasonal flu study is also surveyed. Since we are focused on H1N1 vaccination, those data are not going to be used n be dropped.

Here are the data libarary for each feature.

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 [245]:
```
print(project3_raw_df["h1n1_vaccine"].value_counts())
print()
print("Percentages")
print(project3_raw_df["h1n1_vaccine"].value_counts(normalize=True))
```

```
0    21033
1     5674
Name: h1n1_vaccine, dtype: int64

Percentages
0    0.787546
1    0.212454
Name: h1n1_vaccine, dtype: float64
```

In [246]:
```
y = project3_raw_df['h1n1_vaccine']
X = project3_raw_df.drop(columns=['h1n1_vaccine'], axis=1)
```

In [247]:
```
project3_raw_df['h1n1_vaccine'].value_counts()
```

Out[247]:
```
0    21033
1     5674
Name: h1n1_vaccine, dtype: int64
```

EDA analysis-

Let's check on the mean and standard deviation for the vaccination group and the unvaccination group. People who are vaccianted have higher score on most categories in regard to prevent h1n1 or aware of h1n1.

In [248]:
```
aggs = project3_raw_df.groupby('h1n1_vaccine').agg(['mean', 'std'])
aggs
```

```
C:\Users\eggfr\AppData\Local\Temp\ipykernel_7372\273236371.py:1: FutureWarning:
['age_group', 'education', 'race', 'sex', 'income_poverty', 'marital_status',
'rent_or_own', 'employment_status', 'hhs_geo_region', 'census_msa', 'employment
_industry', 'employment_occupation'] did not aggregate successfully. If any err
or is raised this will raise in a future version of pandas. Drop these columns/
ops to avoid this warning.
  aggs = project3_raw_df.groupby('h1n1_vaccine').agg(['mean', 'std'])
```

Out[248]:

	respondent_id		**h1n1_concern**		**h1n1_knowledge**		**behavioral_antivi**	
	mean	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**
h1n1_vaccine								
0	13366.133885	7704.999816	1.560815	0.910159	1.224653	0.615697	0.044305	
1	13304.313888	7728.011741	1.832096	0.878564	1.402866	0.606937	0.065722	

2 rows × 50 columns

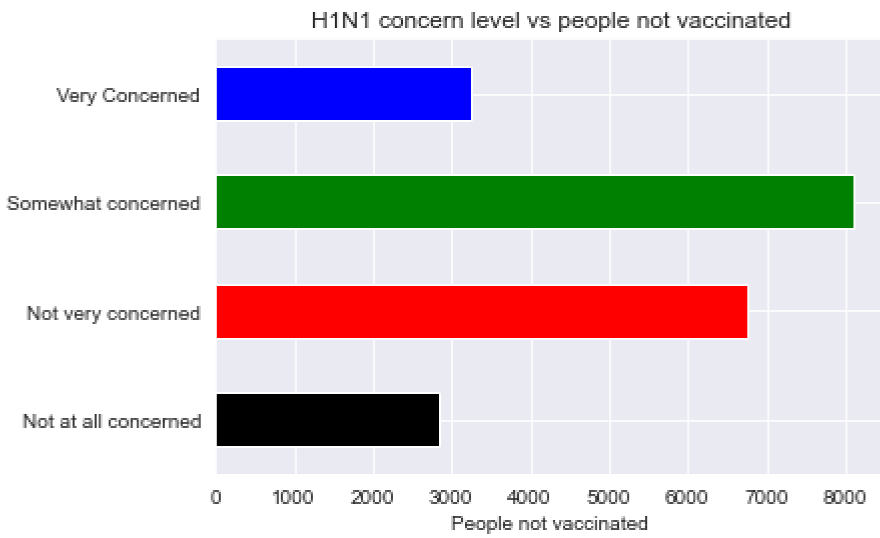
Let's take a look on h1n1_concern and the total amount of people vaccinated. Interresting, people who are in the middle of the concern levels have >50% of people of the whole survey of not being vaccinated. Note: 0 = Not at all concerned; 1 = Not very concerned; 2 = Somewhat concerned; 3 = Very concerned.

In [249]:
```
filt = project3_raw_df['h1n1_vaccine'] == 0
project3_raw_df.loc[filt]['h1n1_concern'].value_counts()
```

Out[249]:
```
2.0    8102
1.0    6756
3.0    3250
0.0    2849
Name: h1n1_concern, dtype: int64
```

In [250]:
```
ax = project3_raw_df.loc[filt]['h1n1_concern'].value_counts().sort_index(ascendir
#ax.set_xlabel("H1N1_ concern level")
plt.yticks((0, 1, 2, 3), ('Not at all concerned', 'Not very concerned', 'Somewhat
ax.set_xlabel("People not vaccinated")
ax.set_title("H1N1 concern level vs people not vaccinated")
```

Out[250]:
```
Text(0.5, 1.0, 'H1N1 concern level vs people not vaccinated')
```

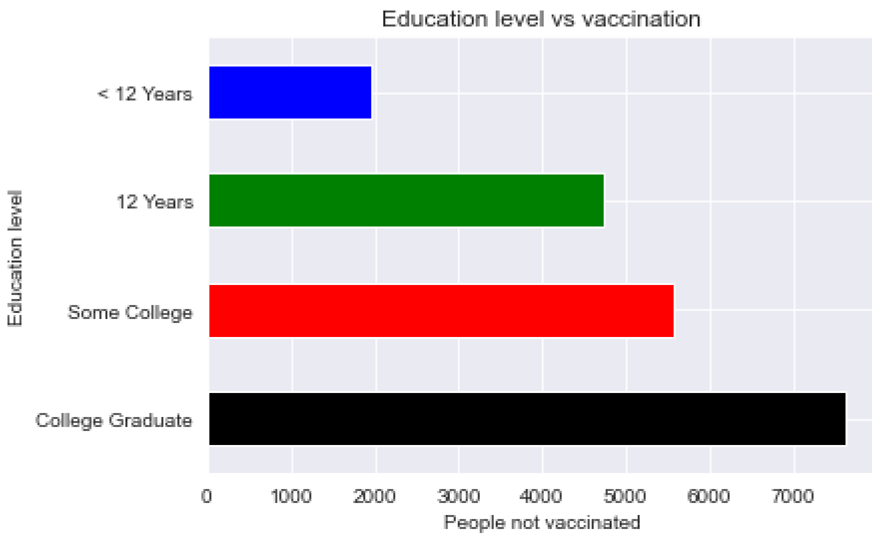


In [251]:
```
filt = project3_raw_df['h1n1_vaccine'] == 0
project3_raw_df.loc[filt]['education'].value_counts()
```

Out[251]:
```
College Graduate    7614
Some College        5579
12 Years            4726
< 12 Years          1968
Name: education, dtype: int64
```

In [252]:
```
ax = project3_raw_df.loc[filt]['education'].value_counts().plot(kind = 'barh',col
ax.set_ylabel("Education level")
ax.set_xlabel("People not vaccinated")
ax.set_title("Education level vs vaccination")
```

Out[252]:
```
Text(0.5, 1.0, 'Education level vs vaccination')
```

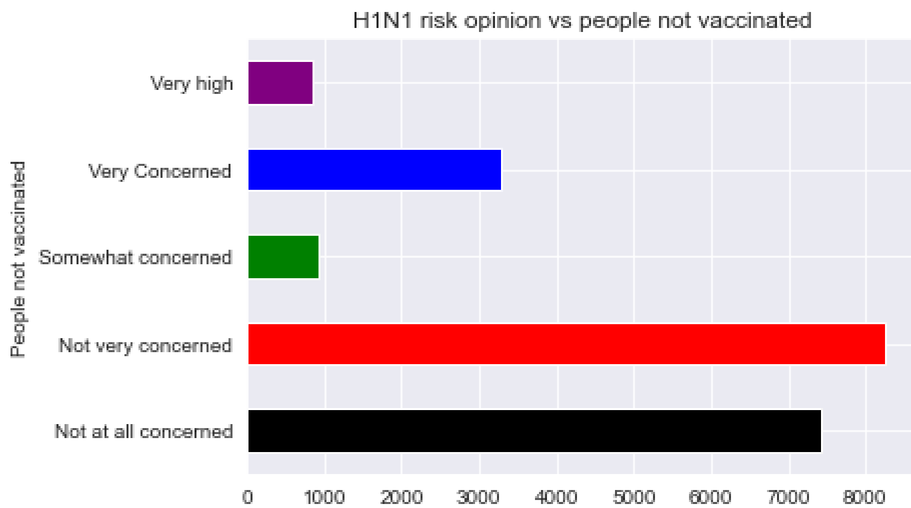


In [253]:
```
filt = project3_raw_df['h1n1_vaccine'] == 0
project3_raw_df.loc[filt]['opinion_h1n1_risk'].value_counts().sort_index(ascendin
```

Out[253]:
```
1.0    7420
2.0    8253
3.0     923
4.0    3279
5.0     856
Name: opinion_h1n1_risk, dtype: int64
```

In [254]:
```
ax = project3_raw_df.loc[filt]['opinion_h1n1_risk'].value_counts().sort_index(asc
#ax.set_xlabel("H1N1_ concern level")
plt.yticks((0, 1, 2,3,4), ('Not at all concerned', 'Not very concerned', 'Somewha
ax.set_ylabel("People not vaccinated")
ax.set_title("H1N1 risk opinion vs people not vaccinated")
```

Out[254]:
```
Text(0.5, 1.0, 'H1N1 risk opinion vs people not vaccinated')
```



In [255]:
```
unvac_df = project3_raw_df.loc[filt]
```

Train/Test Split

Separating data into training and testing sets is an important part of evaluating the models.Most of the data is used for training, and a smaller portion of the data is used for testing. For this analysis: we only split data into train and test. 75% of the data is for training and 25% for test. Also, the data split happened before we even do any EDA analysis to prevent data leakage. There is 20030 row of datas for the train set and 6677 rows of the data for test set before any data cleaning or analysis is done.

In [256]:
```
#create train-test set using 75%-25% ratio for the train set and test set and set
x_train, x_test, y_train, y_test = train_test_split(X, y ,test_size=0.25,random_s
# shape of train and test splits
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[256]:
```
((20030, 37), (6677, 37), (20030,), (6677,))
```

Data preprocessing and imputing missing value for data

There is no duplication for the train set and test set.

In [257]:
```
x_train.duplicated().sum()
```

Out[257]:
```
0
```

In [258]:
```
x_test.duplicated().sum()
```

Out[258]:
```
0
```

Impute missing value with most frequent value or mean when the feature has less than 5% of missing data.

Let's check which features has missing value. For any column that has missing value with less than 1100 (~5% of the data), we are going to impute it with most frequent value for categorical variables and mean for numerical variables in the pipeline. We need to have different different strategy for
health_insurance,income_poverty,doctor_recc_h1n1,rent_or_own,employment_occupation,employm

In [259]:
```
x_train.isnull().sum().sort_values(ascending=False)
```

Out[259]:
```
employment_occupation          10074
employment_industry             9974
health_insurance                9233
income_poverty                  3269
doctor_recc_h1n1                1635
doctor_recc_seasonal            1635
rent_or_own                     1512
employment_status               1081
education                       1040
marital_status                  1038
chronic_med_condition            717
child_under_6_months             605
health_worker                    597
opinion_seas_sick_from_vacc      407
opinion_seas_risk                387
opinion_seas_vacc_effective      349
opinion_h1n1_sick_from_vacc      301
opinion_h1n1_vacc_effective      299
opinion_h1n1_risk                292
household_children               188
household_adults                 188
behavioral_avoidance             157
behavioral_touch_face             98
h1n1_knowledge                    87
behavioral_large_gatherings       70
h1n1_concern                      67
behavioral_outside_home           58
behavioral_antiviral_meds         56
behavioral_wash_hands             36
behavioral_face_mask              14
census_msa                         0
respondent_id                      0
hhs_geo_region                     0
sex                                0
race                               0
age_group                          0
seasonal_vaccine                   0
dtype: int64
```

There is no clear way to impute the missing value for people who have health insurance. A KNN imputer is used to impute missing value from data that are similar to missing data

In [260]:
```
x_train['health_insurance'].value_counts()
```

Out[260]:
```
1.0    9514
0.0    1283
Name: health_insurance, dtype: int64
```

By looking at the value count, most people is in the between the <=75000 and poverty group. Hence, it is reasonable to just impute the missing value to that group

In [261]:
```
x_train['income_poverty'].value_counts()
```

Out[261]:
```
<= $75,000, Above Poverty    9671
> $75,000                    5095
Below Poverty                1995
Name: income_poverty, dtype: int64
```

For doctor recommendation- It would be reasonable to assume those N/A would be that the doctors didnt say anything (so didnt recommand about the H1N1 vaccination). Hence, the missing value is imputed to 0 value.

In [262]:
```
x_train['doctor_recc_h1n1'].value_counts()
```

Out[262]:
```
0.0    14318
1.0     4077
Name: doctor_recc_h1n1, dtype: int64
```

There is a lot of missing data for employment_occupation and employment_industry, and also these features are classified with some code. For this study, we will drop these features, but it should be checked back and study to see how these features affect vaccination status.

In [263]:
```
x_train['rent_or_own'].value_counts()
```

Out[263]:
```
Own     14094
Rent     4424
Name: rent_or_own, dtype: int64
```

In [264]:
```
x_train['employment_occupation'].value_counts()
```

Out[264]:
```
xtkaffoo    1316
mxkfnird    1139
cmhcxjea     959
emcorrxb     942
xgwztkwe     813
hfxkjkmi     582
qxajmpny     414
xqwwgdyp     371
kldqjyjy     363
uqqtjvyb     337
tfqavkke     280
ukymxvdu     278
vlluhbov     263
ccgxvspp     262
oijqvulv     252
bxpfxfdn     251
haliazsg     227
rcertsgn     213
xzmlyyjv     190
dlvbwzss     172
hodpvpew     144
dcjcmpih     117
pvmttkik      71
Name: employment_occupation, dtype: int64
```

In [265]:
```
x_train['employment_industry'].value_counts(normalize = True)
```

Out[265]:
```
fcxhlnwr    0.187848
wxleyezf    0.131961
ldnlellj    0.092979
pxcmvdjn    0.078361
atmlpfrs    0.071201
xicduogh    0.064439
arjwrbjb    0.063842
mfikgejo    0.045744
vjjrobsf    0.038982
rucpziij    0.038882
xqicxuve    0.037689
saaquncn    0.025358
cfqqtusy    0.023767
nduyfdeo    0.020784
mcubkhph    0.019889
wlfvacwt    0.015414
dotnnunm    0.013624
haxffmxo    0.011635
msuufmds    0.009348
phxvnwax    0.007160
qnlwzans    0.001094
Name: employment_industry, dtype: float64
```

In [266]:
```
x_train['h1n1_concern'].value_counts()
```

Out[266]:
```
2.0    7915
1.0    6149
3.0    3448
0.0    2451
Name: h1n1_concern, dtype: int64
```

I am going to make a list of the following categorical variable so I can prepare a list for the feature for the one hot encoding.

In [267]:
```
print("age_group")
print(x_train.age_group.unique())

print("education")
print(x_train.education.unique())

print("race")
print(x_train.race.unique())

print("income_poverty")
print(x_train.income_poverty.unique())

print("marital_status")
print(x_train.marital_status.unique())

print("rent_or_own")
print(x_train.rent_or_own.unique())

print("employment_status")
print(x_train.employment_status.unique())

print("hhs_geo_region")
print(x_train.hhs_geo_region.unique())

print("census_msa")
print(x_train.census_msa.unique())

print("employment_industry")
print(x_train.employment_industry.unique())

print("employment_occupation")
print(x_train.employment_occupation.unique())
```

```
age_group
['18 - 34 Years' '45 - 54 Years' '55 - 64 Years' '65+ Years'
 '35 - 44 Years']
education
['12 Years' 'Some College' 'College Graduate' nan '< 12 Years']
race
['White' 'Hispanic' 'Black' 'Other or Multiple']
income_poverty
[nan '<= $75,000, Above Poverty' 'Below Poverty' '> $75,000']
marital_status
['Not Married' 'Married' nan]
rent_or_own
['Own' nan 'Rent']
employment_status
['Not in Labor Force' 'Employed' nan 'Unemployed']
hhs_geo_region
['oxchjgsf' 'lzgpxyit' 'kbazzjca' 'mlyzmhmf' 'bhuqouqj' 'lrircsnp'
 'atmpeygn' 'fpwskwrf' 'dqpwygqj' 'qufhixun']
census_msa
['Non-MSA' 'MSA, Not Principle  City' 'MSA, Principle City']
employment_industry
[nan 'fcxhlnwr' 'wlfvacwt' 'mcubkhph' 'xqicxuve' 'wxleyezf' 'mfikgejo'
 'arjwrbjb' 'pxcmvdjn' 'rucpziij' 'nduyfdeo' 'ldnlellj' 'atmlpfrs'
 'saaquncn' 'cfqqtusy' 'xicduogh' 'haxffmxo' 'vjjrobsf' 'dotnnunm'
```

```
 'msuufmds' 'qnlwzans' 'phxvnwax']
employment_occupation
[nan 'oijqvulv' 'hfxkjkmi' 'ukymxvdu' 'mxkfnird' 'kldqjyjy' 'xtkaffoo'
 'emcorrxb' 'xgwztkwe' 'cmhcxjea' 'tfqavkke' 'xqwwgdyp' 'vlluhbov'
 'ccgxvspp' 'hodpvpew' 'uqqtjvyb' 'haliazsg' 'qxajmpny' 'bxpfxfdn'
 'xzmlyyjv' 'rcertsgn' 'dlvbwzss' 'dcjcmpih' 'pvmttkik']
```

In [268]:
```
x_train.shape
```

Out[268]:
```
(20030, 37)
```

In [269]:
```
print(x_train.health_insurance.unique())
```

```
[nan  1.  0.]
```

Dropping unused column. respondent_id is dropped since it is not going to be used in the analysis. 'opinion_seas_vacc_effective','opinion_seas_risk','opinion_seas_sick_from_vacc' are also dropped since the analysis is focused on H1N1 vaccine prediction. Employment_occupation and employment_industry are dropped as well.

In [270]:
```
x_train = x_train.drop(columns=['respondent_id','employment_occupation','employme
x_train.shape
```

Out[270]:
```
(20030, 31)
```

Pipeline

Now we need to set a pipeline for our data with the imputing staregy from the discussion above. We will set up a numeric pipeline for numerical variable. Feautres with missing value will be imputed by mean. Afterwards, it will be fed into a standard scaler for scaling.

In [271]:
```
numeric_pipeline = Pipeline([('numimputer', SimpleImputer(strategy = 'mean')), ('
```

We set up different ordinal pipelines for different categorical oridnal variables as they have different categorical groups. We first impute the missing value with the startegy mentioned above with the simpleImputer. Then, we encode it with ordinal encoder, and then scale it with standard scaler.

In [272]:
```
age_list = ['18 - 34 Years', '35 - 44 Years','45 - 54 Years', '55 - 64 Years', '6
income_list = ['Below Poverty','<= $75,000, Above Poverty','> $75,000']
emp_stat_list = ['Not in Labor Force' ,'Unemployed','Employed']
edu_list =['< 12 Years','12 Years', 'Some College', 'College Graduate']
census_list = ['Non-MSA', 'MSA, Not Principle  City', 'MSA, Principle City']
hhs_list = ['oxchjgsf', 'lzgpxyit', 'kbazzjca', 'mlyzmhmf', 'bhuqouqj', 'lrircsnp
'atmpeygn', 'fpwskwrf', 'dqpwygqj', 'qufhixun']
```

In [273]:
```
ordinal_age_pipeline = Pipeline([
    ('ordimputer', SimpleImputer(strategy = 'most_frequent')),
    ('ordenc', OrdinalEncoder(categories = [age_list])),
    ('ordnorm', StandardScaler())])
```

In [274]:
```
ordinal_income_pipeline = Pipeline([
    ('ordimputer', SimpleImputer(strategy = 'most_frequent')),
    ('ordenc', OrdinalEncoder(categories = [income_list])),
    ('ordnorm', StandardScaler())])
```

In [275]:
```
ordinal_emp_status_pipeline = Pipeline([
    ('ordimputer', SimpleImputer(strategy = 'most_frequent')),
    ('ordenc', OrdinalEncoder(categories = [emp_stat_list])),
    ('ordnorm', StandardScaler())])
```

In [276]:
```
ordinal_edu_pipeline = Pipeline([
    ('ordimputer', SimpleImputer(strategy = 'most_frequent')),
    ('ordenc', OrdinalEncoder(categories = [edu_list])),
    ('ordnorm', StandardScaler())])
```

In [277]:
```
ordinal_census_pipeline = Pipeline([
    ('ordimputer', SimpleImputer(strategy = 'most_frequent')),
    ('ordenc', OrdinalEncoder(categories = [census_list])),
    ('ordnorm', StandardScaler())])
```

In [278]:
```
ordinal_hhs_pipeline = Pipeline([
    ('ordimputer', SimpleImputer(strategy = 'most_frequent')),
    ('ordenc', OrdinalEncoder(categories = [hhs_list])),
    ('ordnorm', StandardScaler())])
```

Lastly, we set up nominal pipeline using Onehotcoder for the categorical nominal variables. We first impute it with simpleimputer (KNNImputer for nominal_insurance_pipeline). Then, we one hot encode it with OneHotEncoder, and then the data is scaled with MaxAbsScaler.

In [279]:
```
nominal_pipeline = Pipeline([
    ('onehotimputer', SimpleImputer(strategy = 'most_frequent')),
    ('onehotenc', OneHotEncoder(sparse = False, drop = 'first')),
    ('onehotnorm', MaxAbsScaler())])
```

In [280]:
```
nominal_insurance_pipeline = Pipeline([
    ('onehotimputer', KNNImputer(n_neighbors=5)),
    ('onehotenc', OneHotEncoder(sparse = False, drop = 'first')),
    ('onehotnorm', MaxAbsScaler())])
```

In [281]:
```
nominal_doc_rec_pipeline = Pipeline([
    ('onehotimputer', SimpleImputer(strategy = 'constant',fill_value=0)),
    ('onehotenc', OneHotEncoder(sparse = False, drop = 'first')),
    ('onehotnorm', MaxAbsScaler())])
```

Now, we unite different pipeline with the column transformer so we can specify columns each pipeline acts on.

In [282]:

```
num_cols = x_train.select_dtypes(['int', 'float']).columns
nom_resp_cols = ['behavioral_antiviral_meds','behavioral_avoidance','behavioral_t

ct = ColumnTransformer(
    [ ("ordinalpipe", ordinal_age_pipeline, ['age_group']),
       ("ordinalpipe2", ordinal_income_pipeline, ['income_poverty']),
       ("ordinalpipe3", ordinal_emp_status_pipeline, ['employment_status']),
       ("ordinalpipe4", ordinal_edu_pipeline, ['education']),
       ("ordinalpipe5", ordinal_census_pipeline, ['census_msa']),
       ("ordinalpipe6", ordinal_hhs_pipeline, ['hhs_geo_region']),
       ("nominalpipe", nominal_pipeline,nom_resp_cols),
       ("nominalpipe2", nominal_insurance_pipeline,['health_insurance']),
       ("nominalpipe3", nominal_doc_rec_pipeline,['doctor_recc_h1n1']),
       ("numpipe", numeric_pipeline, num_cols)])


     #("nominalpipe", nominal_pipeline,nom_resp_cols),
     #("numpipe", numeric_pipeline, num_cols)])
```

In [283]:

```
x_train_clean = pd.DataFrame(ct.fit_transform(x_train))
x_train_clean.shape
```

Out[283]: (20030, 46)

In [284]:

```
x_train_clean.describe()
```

count	2.003000e+04	2.003000e+04	2.003000e+04	2.003000e+04	2.003000e+04	2.003000e+04
mean	-4.951938e-17	-1.721389e-15	-7.693408e-18	1.343242e-16	-1.015118e-15	4.935809e-16
std	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00
min	-1.496388e+00	-2.010144e+00	-1.234496e+00	-2.040060e+00	-1.359660e+00	-1.399982e+00
25%	-8.113928e-01	-2.694097e-01	-1.234496e+00	-1.038432e+00	-1.359660e+00	-1.061858e+00
50%	-1.263975e-01	-2.694097e-01	8.580827e-01	-3.680469e-02	-2.935569e-02	-4.748591e-02
75%	1.243593e+00	1.471325e+00	8.580827e-01	9.648229e-01	1.300948e+00	9.668860e-01
max	1.243593e+00	1.471325e+00	8.580827e-01	9.648229e-01	1.300948e+00	1.643134e+00

8 rows × 46 columns

In [285]:
```
ct
```

```
'44 '

'Years',

'45 '

'- '

'54 '

'Years',

'55 '

'- '

'64 '

'Years',
```

In [286]:
```
ct.named_transformers_
```

```
 'ordinalpipe5': Pipeline(steps=[('ordimputer', SimpleImputer(strategy='most_
frequent')),

                 ('ordenc',
                  OrdinalEncoder(categories=[['Non-MSA',
                                              'MSA, Not Principle  City',
                                              'MSA, Principle City']])),
                 ('ordnorm', StandardScaler())]),
 'ordinalpipe6': Pipeline(steps=[('ordimputer', SimpleImputer(strategy='most_
frequent')),
                 ('ordenc',
                  OrdinalEncoder(categories=[['oxchjgsf', 'lzgpxyit', 'kbazzj
ca',
                                              'mlyzmhmf', 'bhuqouqj', 'lrircs
np',
                                              'atmpeygn', 'fpwskwrf', 'dqpwyg
qj',
                                              'qufhixun']])),
                 ('ordnorm', StandardScaler())]),
 'nominalpipe': Pipeline(steps=[('onehotimputer', SimpleImputer(strategy='mos
t_frequent')),
```

Baseline model

Lets check the data with a dummyclassifier.

In [287]:
```
steps=[('preprocessing', ct),
       ('classifier', DummyClassifier(strategy='most_frequent'))]
```

In [288]:
```
baseline_pipe = Pipeline(steps)
```

In [289]:
```
baseline_pipe.fit(x_train, y_train)
```

Out[289]:
```
Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('ordinalpipe',
                                                  Pipeline(steps=[('ordimpute
r',
                                                                   SimpleImpute
r(strategy='most_frequent')),
                                                                  ('ordenc',
                                                                   OrdinalEncod
er(categories=[['18 '
'- '
'34 '
'Years',
'35 '
'- '
'44 '
'Years',
'45 '
'- '
'54 '
'Years',
'55 '
'- '
'64 '
'Years',
'65+ '
'Years']])),
                                                                  ('ordnorm',
                                                                   StandardScal
er())]),
                                                  ['age_group']),
                                                 ('ordinalpipe2',
                                                  Pipeli...
       'behavioral_touch_face', 'doctor_recc_h1n1', 'doctor_recc_seasonal',
       'chronic_med_condition', 'child_under_6_months', 'health_worker',
       'health_insurance', 'opinion_h1n1_vacc_effective', 'opinion_h1n1_risk',
       'opinion_h1n1_sick_from_vacc', 'household_adults', 'household_children',
       'seasonal_vaccine'],
```

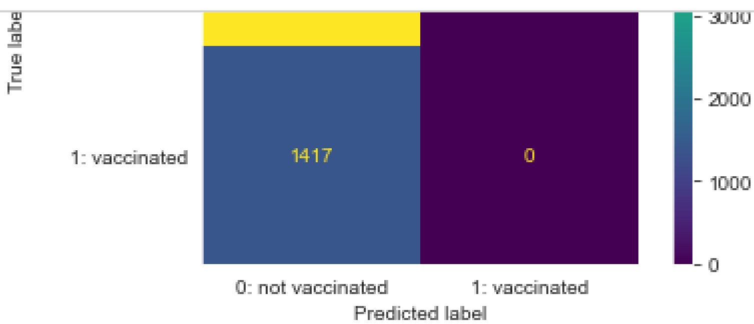
```
                dtype='object'))])),
                          ('classifier', DummyClassifier(strategy='most_frequent'))])
```

In [290]:
```
y_pred0= baseline_pipe.predict(x_test)
```

In [291]:
```

status_labels = ['0: not vaccinated', '1: vaccinated']
plot_confusion_matrix(baseline_pipe,x_test,y_test,display_labels = status_labels)
plt.grid(False)
plt.title('Confusion Matrix - Baseline')
plt.show()

baseline_classification_report = classification_report(y_test, y_pred0)
print(baseline_classification_report)
```

```
           1       0.00      0.00      0.00      1417

    accuracy                           0.79      6677
   macro avg       0.39      0.50      0.44      6677
weighted avg       0.62      0.79      0.69      6677










C:\Users\eggfr\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
   warn prf(average, modifier, msg start, len(result))
```

In [292]:
```
acc = accuracy_score(y_test,y_pred0) * 100
print('Accuracy is :{0}'.format(acc))

pre = precision_score(y_test,y_pred0) * 100
print('precision is :{0}'.format(pre))

# Check the AUC for predictions
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred0)
roc_auc = auc(false_positive_rate, true_positive_rate)
print('\nAUC is :{0}'.format(round(roc_auc, 2)))

# Create and print a confusion matrix
print('\nConfusion Matrix')
print('----------------')
#pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=T
from sklearn.metrics import plot_confusion_matrix


plot_confusion_matrix(baseline_pipe,x_test,y_test,display_labels = status_labels)
plt.grid(False)
plt.title('Confusion Matrix - Baseline Model ')
plt.show()
logreg_classification_report = classification_report(y_test, y_pred0)
print(logreg_classification_report)
```




```
              precision    recall  f1-score   support

           0       0.79      1.00      0.88      5260
           1       0.00      0.00      0.00      1417

    accuracy                           0.79      6677
   macro avg       0.39      0.50      0.44      6677
weighted avg       0.62      0.79      0.69      6677
```

In [293]:
```
print(baseline_pipe.score(x_train,y_train))
print(baseline_pipe.score(x_test,y_test))
```

```
0.7874687968047928
0.7877789426389097
```

The classification reports 78% for true negative and 0% for ture positive test (vaccinated). We are focusing on the True positive, True negative and False Positive when evaluating model because our stake holders foucs on more vaccination.Hence, precision and accuracy are our key metrics for our evaluation. This model is just for model comparison for later.

Model 1 Logisitc Regression Model

In [294]:
```
steps = [('preprocess', ct),
            ('logreg',
                        LogisticRegression(random_state=42))]

model1_pipe = Pipeline(steps)
```

In [295]:
```
model1_pipe
```

Out[295]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('ordinalpipe',
                                                  Pipeline(steps=[('ordimpute
r',
                                                                   SimpleImpu
ter(strategy='most_frequent')),
                                                                  ('ordenc',
                                                                   OrdinalEnc
oder(categories=[['18 '

'- '

'34 '

'Years',

'35 '

'- '

'44 '

'Years',

'45 '

'- '

'54 '

'Years',

'55 '

'- '

'64 '

'Years',

'65+ '

'Years']])),
                                                                  ('ordnorm',
                                                                   StandardSc
aler())]),
                                                  ['age_group']),
                                                 ('ordinalpipe2',
                                                  Pipeline(...
       'behavioral_touch_face', 'doctor_recc_h1n1', 'doctor_recc_seasonal',
       'chronic_med_condition', 'child_under_6_months', 'health_worker',
       'health_insurance', 'opinion_h1n1_vacc_effective', 'opinion_h1n1_ris
k',
       'opinion_h1n1_sick_from_vacc', 'household_adults', 'household_childre
n',
```

```
                 'seasonal_vaccine'],
                dtype='object'))])),
                         ('logreg', LogisticRegression(random_state=42))])
```

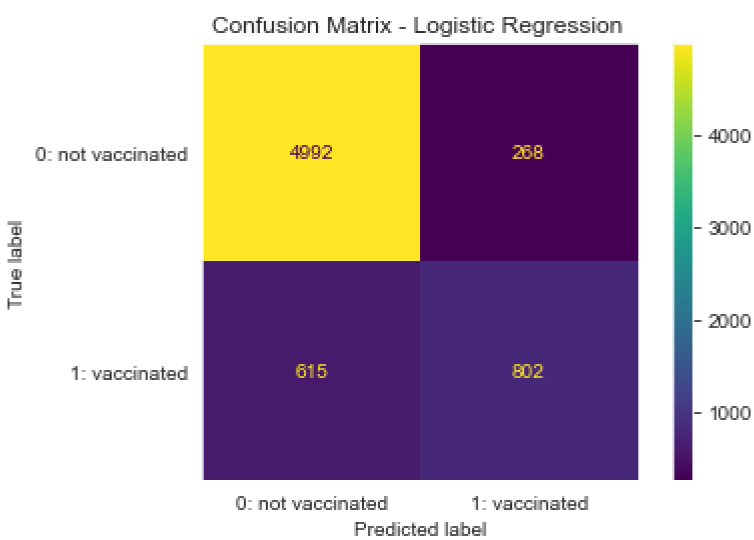
In [296]:
```
model1_pipe.fit(x_train,y_train)
```

```
C:\Users\eggfr\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
814: ConvergenceWarning: lbfgs 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://sciki
t-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
```

In [297]:
```
y_pred = model1_pipe.predict(x_test)
```

In [298]:
```
acc = accuracy_score(y_test,y_pred) * 100
print('Accuracy is :{0}'.format(acc))

pre = precision_score(y_test,y_pred) * 100
print('precision is :{0}'.format(pre))

# Check the AUC for predictions
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
print('\nAUC is :{0}'.format(round(roc_auc, 2)))

# Create and print a confusion matrix
print('\nConfusion Matrix')
print('----------------')
#pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=T
from sklearn.metrics import plot_confusion_matrix


plot_confusion_matrix(model1_pipe,x_test,y_test,display_labels = status_labels)
plt.grid(False)
plt.title('Confusion Matrix - Logistic Regression ')
plt.show()
logreg_classification_report = classification_report(y_test, y_pred)
print(logreg_classification_report)
```

```
Accuracy is :86.7754979781339
precision is :74.95327102803738

AUC is :0.76

Confusion Matrix
----------------

C:\Users\eggfr\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: Fut
ureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confus
ion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the cla
ss methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.f
rom_estimator.
  warnings.warn(msg, category=FutureWarning)
```



```
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      5260
           1       0.75      0.57      0.64      1417

    accuracy                           0.87      6677
   macro avg       0.82      0.76      0.78      6677
weighted avg       0.86      0.87      0.86      6677
```

Logistic Regression did pretty good on precision (75%) when comparing to the dummy model. Test and train model have similar R2 score, so Regression model doesnt really overfit.

In [299]:
```
print(model1_pipe.score(x_train,y_train))
print(model1_pipe.score(x_test,y_test))
```

```
0.8651522715926111
0.867754979781339
```

In [300]:
```
model1_pipe.named_steps["preprocess"]
```

```
'44 '

'Years',

'45 '

'- '

'54 '

'Years',

'55 '

'- '

'64 '

'Years',
```

In [301]:
```
model1_pipe.steps
```

Out[301]:
```
[('preprocess',
  ColumnTransformer(transformers=[('ordinalpipe',
                                   Pipeline(steps=[('ordimputer',
                                                    SimpleImputer(strategy='mos
t_frequent')),
                                                   ('ordenc',
                                                    OrdinalEncoder(categories=
[['18 '

'- '

'34 '

'Years',

'35 '

'- '

'44 '

'Years',

'45 '

'- '

'54 '

'Years',

'55 '

'- '

'64 '

'Years',

'65+ '

'Years']])),
                                                   ('ordnorm',
                                                    StandardScaler())]),
                                   ['age_group']),
                                  ('ordinalpipe2',
                                   Pipeline(steps=[('ordimputer',
                                                    SimpleImp...
         'behavioral_large_gatherings', 'behavioral_outside_home',
         'behavioral_touch_face', 'doctor_recc_h1n1', 'doctor_recc_seasonal',
         'chronic_med_condition', 'child_under_6_months', 'health_worker',
         'health_insurance', 'opinion_h1n1_vacc_effective', 'opinion_h1n1_ris
k',
         'opinion_h1n1_sick_from_vacc', 'household_adults', 'household_childre
n',
```

```
         'seasonal_vaccine'],
        dtype='object'))])),
 ('logreg', LogisticRegression(random_state=42))]
```

Model 2.1 KNN neighbors Classifier.

We are going to investigate KNN neighbors Classifier first. At first, we are going to use the default parameter.

In [308]:
```
# let's define a new pipeline object

steps = [('preprocess', ct),
            ('knn',KNeighborsClassifier())]

model2_pipe = Pipeline(steps)
```

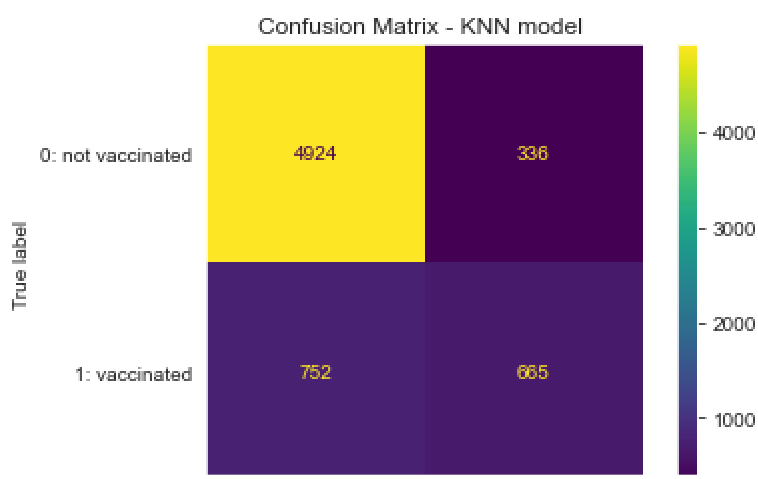
In [309]:
```
model2_pipe.fit(x_train,y_train)
```

Out[309]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('ordinalpipe',
                                                  Pipeline(steps=[('ordimpute
r',
                                                                   SimpleImpute
r(strategy='most_frequent')),
                                                                  ('ordenc',
                                                                   OrdinalEncod
er(categories=[['18 '

'- '

'34 '

'Years',

'35 '

'- '

'44 '

'Years',

'45 '

'- '

'54 '

'Years',

'55 '

'- '

'64 '

'Years',

'65+ '

'Years']])),
                                                                  ('ordnorm',
                                                                   StandardScal
er())]),
                                                  ['age_group']),
                                                 ('ordinalpipe2',
                                                  Pipeline(...
       'behavioral_large_gatherings', 'behavioral_outside_home',
       'behavioral_touch_face', 'doctor_recc_h1n1', 'doctor_recc_seasonal',
       'chronic_med_condition', 'child_under_6_months', 'health_worker',
       'health_insurance', 'opinion_h1n1_vacc_effective', 'opinion_h1n1_risk',
       'opinion_h1n1_sick_from_vacc', 'household_adults', 'household_children',
       'seasonal_vaccine'],
```

```
                 dtype='object'))])),
                ('knn', KNeighborsClassifier())])
```

In [310]:
```
y_pred2 = model2_pipe.predict(x_test)
```

In [311]:
```
acc = accuracy_score(y_test,y_pred2) * 100
print('Accuracy is :{0}'.format(acc))

pre = precision_score(y_test,y_pred2) * 100
print('precision is :{0}'.format(pre))

# Check the AUC for predictions
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred2)
roc_auc = auc(false_positive_rate, true_positive_rate)
print('\nAUC is :{0}'.format(round(roc_auc, 2)))

# Create and print a confusion matrix
print('\nConfusion Matrix')
print('----------------')
#pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=T
from sklearn.metrics import plot_confusion_matrix


plot_confusion_matrix(model2_pipe,x_test,y_test,display_labels = status_labels)
plt.grid(False)
plt.title('Confusion Matrix - KNN model ')
plt.show()
KNN_classification_report = classification_report(y_test, y_pred2)
print(KNN_classification_report)
```



Precision score has a little bit drop off when compare to the regression model. We will test out the n neighbor parameter n k parameter to see if we can improve the result.

In [312]:
```
print(model2_pipe.score(x_train,y_train))
print(model2_pipe.score(x_test,y_test))
```

```
0.8806789815277084
0.8370525685187958
```

Running the code below w pipe_grid as the range to tune the knn n_neighbors, knn_p parameter to fine tune the knn model. It takes a lot of time to run the model, so it get hided out in this notebook (but feel free to run the model). The best parameter is {'knn__n_neighbors': 15, 'knn__p': 1}.

In [313]:
```
#pipe_grid = {
 #            'knn__n_neighbors': [3, 11, 15],
  #           'knn__p': [1, 2]}
```

In [314]:
```
#gs_pipe = GridSearchCV(estimator=model2_pipe,
#                     param_grid=pipe_grid)
```

In [315]:
```
#gs_pipe.fit(x_train, y_train);
```

In [316]:
```
#gs_pipe.best_params_
```

Running the code

{'knn__n_neighbors': 15, 'knn__p': 1}

In [317]:
```
steps = [('preprocess', ct),
            ('knn',KNeighborsClassifier(n_neighbors=15, p= 1 ))]

model2a_pipe = Pipeline(steps)
```

In [318]:
```
model2a_pipe.fit(x_train,y_train)
```

Out[318]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('ordinalpipe',
                                                  Pipeline(steps=[('ordimpute
r',
                                                                   SimpleImpu
ter(strategy='most_frequent')),
                                                                  ('ordenc',
                                                                   OrdinalEnc
oder(categories=[['18 '

'- '

'34 '

'Years',

'35 '

'- '

'44 '

'Years',

'45 '

'- '

'54 '

'Years',

'55 '

'- '

'64 '

'Years',

'65+ '

'Years']])),
                                                                  ('ordnorm',
                                                                   StandardSc
aler())]),
                                                  ['age_group']),
                                                 ('ordinalpipe2',
                                                  Pipeline(...
       'behavioral_touch_face', 'doctor_recc_h1n1', 'doctor_recc_seasonal',
       'chronic_med_condition', 'child_under_6_months', 'health_worker',
       'health_insurance', 'opinion_h1n1_vacc_effective', 'opinion_h1n1_ris
k',
       'opinion_h1n1_sick_from_vacc', 'household_adults', 'household_childre
n',
```

```
             'seasonal_vaccine'],
            dtype='object'))])),
                      ('knn', KNeighborsClassifier(n_neighbors=15, p=1))])
```

In [319]:
```
y_pred2a = model2a_pipe.predict(x_test)
```

In [320]:
```
acc = accuracy_score(y_test,y_pred2a) * 100
print('Accuracy is :{0}'.format(acc))

pre = precision_score(y_test,y_pred2a) * 100
print('precision is :{0}'.format(pre))

# Check the AUC for predictions
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred2a)
roc_auc = auc(false_positive_rate, true_positive_rate)
print('\nAUC is :{0}'.format(round(roc_auc, 2)))

# Create and print a confusion matrix
print('\nConfusion Matrix')
print('----------------')
#pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=T
from sklearn.metrics import plot_confusion_matrix


plot_confusion_matrix(model2a_pipe,x_test,y_test,display_labels = status_labels)
plt.grid(False)
plt.title('Confusion Matrix - KNN model Optimum Parameter')
plt.show()
KNN_classification_report = classification_report(y_test, y_pred2a)
print(KNN_classification_report)
```

```
Accuracy is :85.11307473416205
precision is :73.16538882803944

AUC is :0.71

Confusion Matrix
----------------

C:\Users\eggfr\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: F
utureWarning: Function plot_confusion_matrix is deprecated; Function `plot_co
nfusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of t
he class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixD
isplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Precision score improved compared to the default KNN model but it is still has a slighly lower accuracy and precision score compare to the logistic regression model.

In [321]:
```
print(model2a_pipe.score(x_train,y_train))
print(model2a_pipe.score(x_test,y_test))
```

```
0.8662006989515726
0.8511307473416205
```

In [322]:
```
#knn_best_classification_report = classification_report(y_test, y_pred2a)
#print(knn_best_classification_report)
```

Model 3: Decision Tree

In this classifier, we are using evaulating with decision trees. We will start with default parameters with a random_state = 42. We should expect overfitting on the train set data by default.

In [323]:
```
steps =[('preprocess', ct),
                      ('dt',
                       DecisionTreeClassifier(random_state = 42))]
model3_pipe = Pipeline(steps)
```

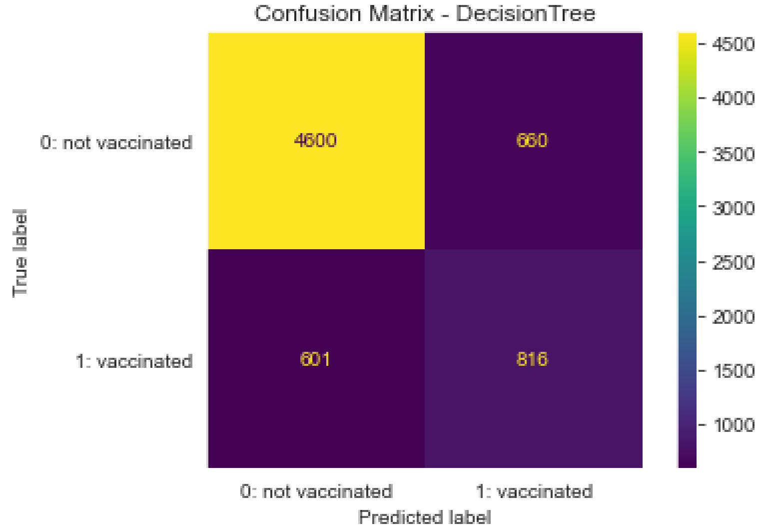
In [324]:
```
model3_pipe.fit(x_train,y_train)
```

```
'- '

'44 '

'Years',

'45 '

'- '

'54 '

'Years',

'55 '

'- '

'64 '
```

In [325]:
```
y_pred3 = model3_pipe.predict(x_test)
```

In [326]:
```
acc = accuracy_score(y_test,y_pred3) * 100
print('Accuracy is :{0}'.format(acc))

pre = precision_score(y_test,y_pred3) * 100
print('precision is :{0}'.format(pre))

# Check the AUC for predictions
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred3)
roc_auc = auc(false_positive_rate, true_positive_rate)
print('\nAUC is :{0}'.format(round(roc_auc, 2)))

# Create and print a confusion matrix
print('\nConfusion Matrix')
print('----------------')
#pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=T
from sklearn.metrics import plot_confusion_matrix


plot_confusion_matrix(model3_pipe,x_test,y_test,display_labels = status_labels)
plt.grid(False)
plt.title('Confusion Matrix - DecisionTree')
plt.show()
dt_classification_report = classification_report(y_test, y_pred3)
print(dt_classification_report)
```

```
Accuracy is :81.11427287704058
precision is :55.28455284552846

AUC is :0.73

Confusion Matrix
----------------



C:\Users\eggfr\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: Fut
ureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confus
ion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the cla
ss methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.f
rom_estimator.
  warnings.warn(msg, category=FutureWarning)
```




```
              precision    recall  f1-score   support

           0       0.88      0.87      0.88      5260
           1       0.55      0.58      0.56      1417

    accuracy                           0.81      6677
   macro avg       0.72      0.73      0.72      6677
weighted avg       0.81      0.81      0.81      6677
```

In [327]:
```
print(model3_pipe.score(x_train,y_train))
print(model3_pipe.score(x_test,y_test))
```

```
0.9999500748876685
0.8111427287704058
```

Decisiontree model was overfitting as expected, and it also didnt score high on accuracy or precision. Lets see if we can improve the model by adjusting the hyper parameter.

In [328]:
```
#pipe3_grid = {
     #       'dt__criterion':['gini','entropy'],
    #       'dt__max_depth': [30,35,40],
     #        'dt__min_samples_split': [5,6],
      #       'dt__min_samples_leaf': [2,3]
   # }
```

In [329]:
```
#gs_pipe = GridSearchCV(estimator=model3_pipe,
                     # param_grid=pipe3_grid,
                     # cv = 3,
                     #   n_jobs=-1
                     #  )
```

In [330]:
```
#gs_pipe.fit(x_train, y_train)
```

{'dt__criterion': 'gini', 'dt__max_depth': 30, 'dt__min_samples_leaf': 3, 'dt__min_samples_split': 5}

We use GridSearchCV to find the optimum parameters for criterion,max_depth,min_sample_leaf,and min_samples_split for the DecisionTree model. The code works but highlighted so it wont take a long time to run the notebook.

In [331]:
```
steps =[('preprocess', ct),
                      ('dt',
                       DecisionTreeClassifier(criterion = 'gini',
                                              max_depth =30,
                                              min_samples_leaf = 3,
                                              min_samples_split = 5,
                                                random_state = 42))]
model3a_pipe = Pipeline(steps)
```

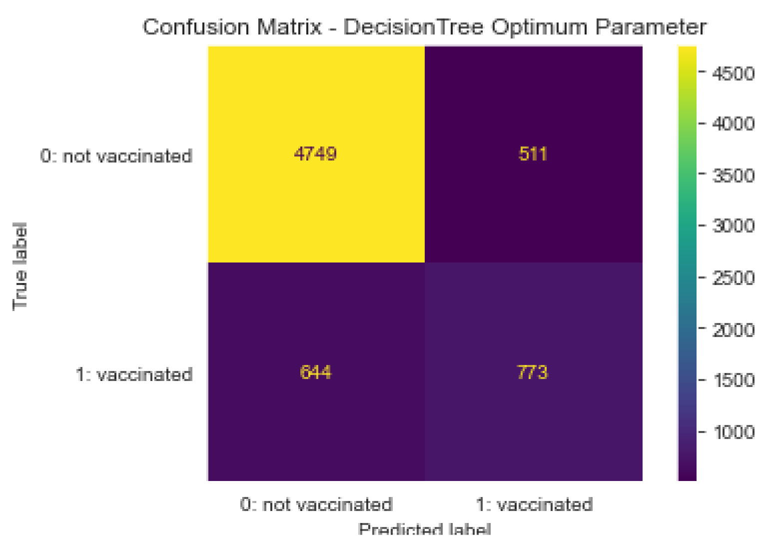
In [332]:
```
model3a_pipe.fit(x_train,y_train)
```

Out[332]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('ordinalpipe',
                                                  Pipeline(steps=[('ordimpute
r',
                                                                   SimpleImpute
r(strategy='most_frequent')),
                                                                  ('ordenc',
                                                                   OrdinalEncod
er(categories=[['18 '
'- '
'34 '
'Years',
'35 '
'- '
'44 '
'Years',
'45 '
'- '
'54 '
'Years',
'55 '
'- '
'64 '
'Years',
'65+ '
'Years']])),
                                                                  ('ordnorm',
                                                                   StandardScal
er())]),
                                                  ['age_group']),
                                                 ('ordinalpipe2',
                                                  Pipeline(...
       'behavioral_touch_face', 'doctor_recc_h1n1', 'doctor_recc_seasonal',
       'chronic_med_condition', 'child_under_6_months', 'health_worker',
       'health_insurance', 'opinion_h1n1_vacc_effective', 'opinion_h1n1_risk',
       'opinion_h1n1_sick_from_vacc', 'household_adults', 'household_children',
       'seasonal_vaccine'],
      dtype='object'))])),
```

```
                        ('dt',
                         DecisionTreeClassifier(max_depth=30, min_samples_leaf=3,
                                                min_samples_split=5,
                                                random_state=42))])
```

In [333]:
```
y_pred3a = model3a_pipe.predict(x_test)
```

In [334]:
```
acc = accuracy_score(y_test,y_pred3a) * 100
print('Accuracy is :{0}'.format(acc))

pre = precision_score(y_test,y_pred3a) * 100
print('precision is :{0}'.format(pre))

# Check the AUC for predictions
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred3a)
roc_auc = auc(false_positive_rate, true_positive_rate)
print('\nAUC is :{0}'.format(round(roc_auc, 2)))

# Create and print a confusion matrix
print('\nConfusion Matrix')
print('----------------')
#pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=T
from sklearn.metrics import plot_confusion_matrix


plot_confusion_matrix(model3a_pipe,x_test,y_test,display_labels = status_labels)
plt.grid(False)
plt.title('Confusion Matrix - DecisionTree Optimum Parameter')
plt.show()
dt_classification_report = classification_report(y_test, y_pred3a)
print(dt_classification_report)
```



In [335]:
```
print(model3a_pipe.score(x_train,y_train))
print(model3a_pipe.score(x_test,y_test))
```

```
0.9422366450324513
0.8270181219110379
```

The model is overfitting but in a less degree with the parameter adjustment. Accuracy and precision improve slightly but it has a lower score than logistic regression model.

Model 4: AdaBoostClassifier

All the models we've learned so far are Strong Learners -- models with the goal of doing as well as possible on the classification or regression task they are given. The term Weak Learner refers to simple models that do only slightly better than random chance. We also test the Adaboostclassifier with default parameters with random_state = 42).

In [336]:
```
steps =[('preprocess', ct),
                     ('ab_clf',
                     AdaBoostClassifier(random_state=42))]

model4_pipe = Pipeline(steps)
```

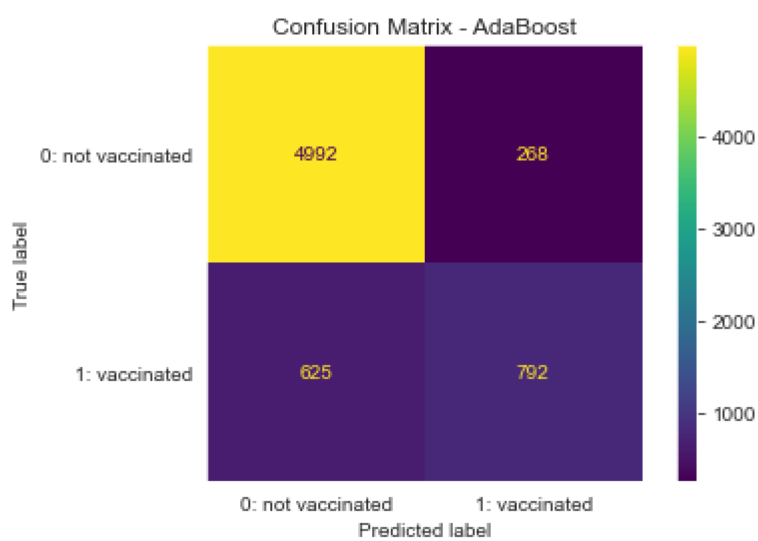
In [337]:
```
model4_pipe.fit(x_train,y_train)
```

Out[337]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('ordinalpipe',
                                                  Pipeline(steps=[('ordimpute
r',
                                                                   SimpleImpu
ter(strategy='most_frequent')),
                                                                  ('ordenc',
                                                                   OrdinalEnc
oder(categories=[['18 '

'- '

'34 '

'Years',

'35 '

'- '

'44 '

'Years',

'45 '

'- '

'54 '

'Years',

'55 '

'- '

'64 '

'Years',

'65+ '

'Years']])),
                                                                  ('ordnorm',
                                                                   StandardSc
aler())]),
                                                  ['age_group']),
                                                 ('ordinalpipe2',
                                                  Pipeline(...
       'behavioral_touch_face', 'doctor_recc_h1n1', 'doctor_recc_seasonal',
       'chronic_med_condition', 'child_under_6_months', 'health_worker',
       'health_insurance', 'opinion_h1n1_vacc_effective', 'opinion_h1n1_ris
k',
       'opinion_h1n1_sick_from_vacc', 'household_adults', 'household_childre
n',
```

```
                 'seasonal_vaccine'],
                dtype='object'))])),
                ('ab_clf', AdaBoostClassifier(random_state=42))])
```

In [338]:
```
y_pred4 = model4_pipe.predict(x_test)
```

In [339]:
```
# Calculate accuracy
acc = accuracy_score(y_test,y_pred4) * 100
print('Accuracy is :{0}'.format(acc))

pre = precision_score(y_test,y_pred4) * 100
print('precision is :{0}'.format(pre))

# Check the AUC for predictions
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred4)
roc_auc = auc(false_positive_rate, true_positive_rate)
print('\nAUC is :{0}'.format(round(roc_auc, 2)))

# Create and print a confusion matrix
print('\nConfusion Matrix')
print('----------------')
#pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=T
from sklearn.metrics import plot_confusion_matrix


plot_confusion_matrix(model4_pipe,x_test,y_test,display_labels = status_labels)
plt.grid(False)
plt.title('Confusion Matrix - AdaBoost')
plt.show()
Ada_classification_report = classification_report(y_test, y_pred4)
print(Ada_classification_report)
```



In [340]:
```
print(model4_pipe.score(x_train,y_train))
print(model4_pipe.score(x_test,y_test))
```

```
0.8652521218172741
0.8662573011831661
```

Adaboost seems to have the best score so far in precision and accuracy. Let's see if we can improve the score by adjusting parameter.

We use GridSearchCV to find the optimum parameters for n_estimators n learning rate for the AdaBoost model. The code works but highlighted so it wont take a long time to run the notebook.

In [341]:
```

#pipe4_grid = {
#             'ab_clf__n_estimators':[60,90,120],
#              'ab_clf__learning_rate': [1,2]
#                 }

```

In [342]:
```
#gs_pipe = GridSearchCV(estimator=model4_pipe,
 #                      param_grid=pipe4_grid,
 #                      cv = 3,
 #                        n_jobs=-1
 #                      )
```

In [343]:
```
#gs_pipe.fit(x_train, y_train)
```

In [344]:
```
#gs_pipe.best_params_
```

{'ab_clf__learning_rate': 1, 'ab_clf__n_estimators': 90}

That is the best parameter for the AdaBoost parameter when learning rate and n_estimator are investigated.Lets run this parameter again- so we dont have to spend time on gridsearch to run the notebook again.

In [345]:
```
steps =[('preprocess', ct),
                     ('ab_clf',
                     AdaBoostClassifier(learning_rate = 1,
                                            n_estimators = 90,
                                          random_state=42))]


model4a_pipe = Pipeline(steps)
```

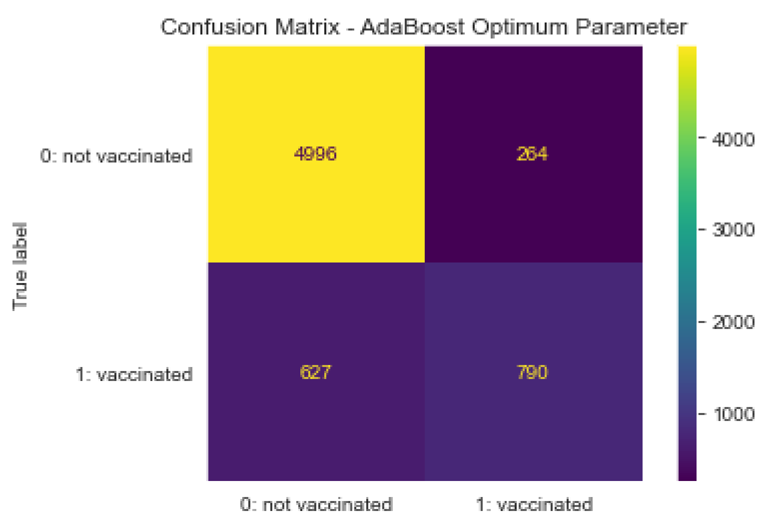
In [346]:
```
model4a_pipe.fit(x_train,y_train)
```

Out[346]:
```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('ordinalpipe',
                                                  Pipeline(steps=[('ordimpute
r',
                                                                   SimpleImpute
r(strategy='most_frequent')),
                                                                  ('ordenc',
                                                                   OrdinalEncod
er(categories=[['18 '

'- '

'34 '

'Years',

'35 '

'- '

'44 '

'Years',

'45 '

'- '

'54 '

'Years',

'55 '

'- '

'64 '

'Years',

'65+ '

'Years']])),
                                                                  ('ordnorm',
                                                                   StandardScal
er())]),
                                                  ['age_group']),
                                                 ('ordinalpipe2',
                                                  Pipeline(...
       'behavioral_touch_face', 'doctor_recc_h1n1', 'doctor_recc_seasonal',
       'chronic_med_condition', 'child_under_6_months', 'health_worker',
       'health_insurance', 'opinion_h1n1_vacc_effective', 'opinion_h1n1_risk',
       'opinion_h1n1_sick_from_vacc', 'household_adults', 'household_children',
       'seasonal_vaccine'],
      dtype='object'))])),
```

```
                                  ('ab_clf',
                                   AdaBoostClassifier(learning_rate=1, n_estimators=90,
                                                      random_state=42))])
```

In [347]:
```
y_pred4a = model4a_pipe.predict(x_test)
```

In [348]:
```
acc = accuracy_score(y_test,y_pred4a) * 100
print('Accuracy is :{0}'.format(acc))

pre = precision_score(y_test,y_pred4a) * 100
print('precision is :{0}'.format(pre))

# Check the AUC for predictions
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred4a)
roc_auc = auc(false_positive_rate, true_positive_rate)
print('\nAUC is :{0}'.format(round(roc_auc, 2)))

# Create and print a confusion matrix
print('\nConfusion Matrix')
print('----------------')
#pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=T
from sklearn.metrics import plot_confusion_matrix


plot_confusion_matrix(model4a_pipe,x_test,y_test,display_labels = status_labels)
plt.grid(False)
plt.title('Confusion Matrix - AdaBoost Optimum Parameter')
plt.show()
Ada_classification_report = classification_report(y_test, y_pred4a)
print(Ada_classification_report)
```



Not much improvement is made as the optimum learning rate remain at 1. It is expected to see the model didnt improve much from default parameter. A different parameter should be investigated to see if that change would improve the score.

In [349]:
```
print(model4a_pipe.score(x_train,y_train))
print(model4a_pipe.score(x_test,y_test))
```

```
0.8657014478282576
0.8665568369028006
```

Model Comparison

In [350]:
```
columns = ['Model', 'Train Accuracy', 'Test Accuracy']
Models = ['Model 0: Baseline', 'Model 1: Logistic Regression', 'Model 2: KNN', 'M
train_accuracy_scores = [0.787,0.865,0.880, 0.866,0.991,0.942,0.866,0.866]
test_accuracy_scores = [0.787,0.867,0.837,0.851,0.811, 0.827,0.866, 0.866]
accuracy_scores = list(zip(Models, train_accuracy_scores, test_accuracy_scores))
accuracy_scores_df = pd.DataFrame(accuracy_scores, columns = columns)
accuracy_scores_df.sort_values(by=['Test Accuracy'], ascending=False)
```

Out[350]:

	Model	Train Accuracy	Test Accuracy
1	Model 1: Logistic Regression	0.865	0.867
6	Model 4: AdaBoost	0.866	0.866
7	Model 4a: AdaBoost OptimumPar	0.866	0.866
3	Model 2a: KNN OptimumPar	0.866	0.851
2	Model 2: KNN	0.880	0.837
5	Model 3a: DecisionTree OptimumPar	0.942	0.827
4	Model 3: DecisionTree	0.991	0.811
0	Model 0: Baseline	0.787	0.787

```
Test score remain
```

Classification Report (accuracy, precision) summary

Our stake holder is focusing on increasing the vaccination, so false negative isn't a concern. Hence, we would be focusing on accuracy and precision.

In [351]:
```
columns = ['Model', 'Precision', 'Accuracy','AUC']
Models = ['Model 0: Baseline', 'Model 1: Logistic Regression', 'Model 2: KNN', 'M
test_accuracy_scores = [0.787,0.867,0.837,0.851,0.811, 0.827,0.866, 0.866]
precision_score = [0.5,0.76,0.66,0.73,0.55,0.6,0.74,0.74]
auc_score = [0.5,0.76,0.7,0.71,0.73,0.72,0.75,0.75]
class_scores = list(zip(Models, test_accuracy_scores, precision_score,auc_score))
class_scores_df = pd.DataFrame(class_scores, columns = columns)
class_scores_df.sort_values(by=['Precision'], ascending=False)
```

Out[351]:

	Model	Precision	Accuracy	AUC
1	Model 1: Logistic Regression	0.867	0.76	0.76
6	Model 4: AdaBoost	0.866	0.74	0.75
7	Model 4a: AdaBoost OptimumPar	0.866	0.74	0.75
3	Model 2a: KNN OptimumPar	0.851	0.73	0.71
2	Model 2: KNN	0.837	0.66	0.70
5	Model 3a: DecisionTree OptimumPar	0.827	0.60	0.72
4	Model 3: DecisionTree	0.811	0.55	0.73
0	Model 0: Baseline	0.787	0.50	0.50

Logisitic Model has the highest precision and accuracy score. Regression Model also has a highest auc score, so it is a slighlty better categorized method

Top 10 features from the logisitic Regression Model

The get_feature_name function is a work by Johannes Haupt. I didnt write that function to get the column name from the column transfromer.
[https://johaupt.github.io/blog/columnTransformer_feature_names.html (https://johaupt.github.io/blog/columnTransformer_feature_names.html)](https://johaupt.github.io/blog/columnTransformer_feature_names.html)

In [352]:
```
import warnings
import sklearn
import pandas as pd
def get_feature_names(column_transformer):
    """Get feature names from all transformers.
    Returns
    -------
    feature_names : list of strings
        Names of the features produced by transform.
    """
    # Remove the internal helper function
    #check_is_fitted(column_transformer)

    # Turn loopkup into function for better handling with pipeline later
    def get_names(trans):
        # >> Original get_feature_names() method
        if trans == 'drop' or (
                hasattr(column, '__len__') and not len(column)):
            return []
        if trans == 'passthrough':
            if hasattr(column_transformer, '_df_columns'):
                if ((not isinstance(column, slice))
                        and all(isinstance(col, str) for col in column)):
                    return column
                else:
                    return column_transformer._df_columns[column]
            else:
                indices = np.arange(column_transformer._n_features)
                return ['x%d' % i for i in indices[column]]
        if not hasattr(trans, 'get_feature_names'):
        # >>> Change: Return input column names if no method avaiable
            # Turn error into a warning
            warnings.warn("Transformer %s (type %s) does not "
                                 "provide get_feature_names. "
                                 "Will return input column names if available"
                                 % (str(name), type(trans).__name__))
            # For transformers without a get_features_names method, use the input
            # names to the column transformer
            if column is None:
                return []
            else:
                return [name + "__" + f for f in column]

        return [name + "__" + f for f in trans.get_feature_names()]

    ### Start of processing
    feature_names = []

    # Allow transformers to be pipelines. Pipeline steps are named differently, s
    if type(column_transformer) == sklearn.pipeline.Pipeline:
        l_transformers = [(name, trans, None, None) for step, name, trans in colu
    else:
        # For column transformers, follow the original method
        l_transformers = list(column_transformer._iter(fitted=True))
```

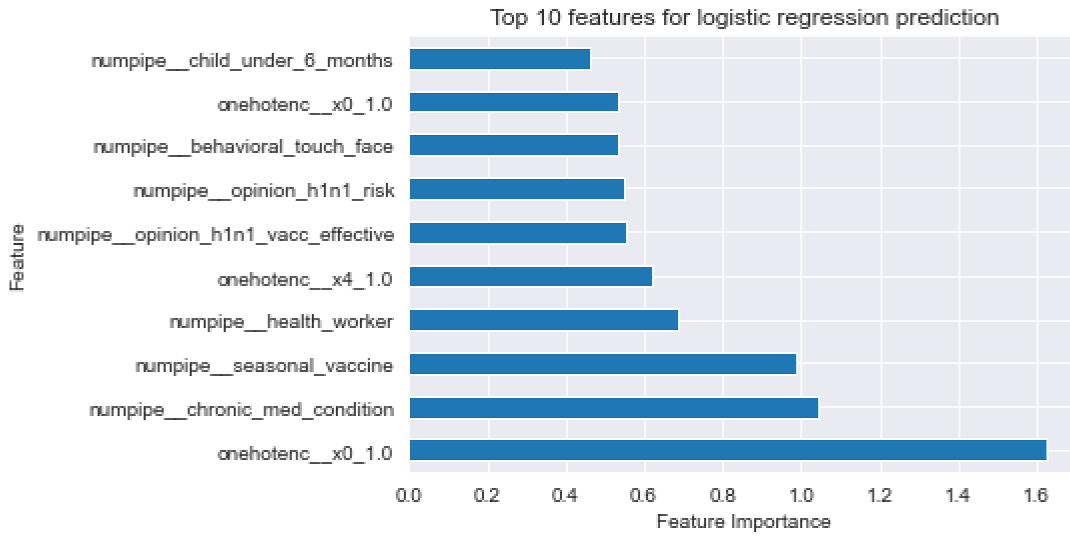
```
    for name, trans, column, _ in l_transformers:
        if type(trans) == sklearn.pipeline.Pipeline:
            # Recursive call on pipeline
            _names = get_feature_names(trans)
            # if pipeline has no transformer that returns names
            if len(_names)==0:
                _names = [name + "__" + f for f in column]
            feature_names.extend(_names)
        else:
            feature_names.extend(get_names(trans))

    return feature_names
```

In [353]:

```
df1 = pd.DataFrame(model1_pipe.named_steps['logreg'].coef_.flatten(), index=get_f
```

```
C:\Users\eggfr\AppData\Local\Temp\ipykernel_7372\1483046653.py:33: UserWarnin
g: Transformer ordnorm (type StandardScaler) does not provide get_feature_nam
es. Will return input column names if available
  warnings.warn("Transformer %s (type %s) does not "
C:\Users\eggfr\AppData\Local\Temp\ipykernel_7372\1483046653.py:33: UserWarnin
g: Transformer onehotimputer (type SimpleImputer) does not provide get_featur
e_names. Will return input column names if available
  warnings.warn("Transformer %s (type %s) does not "
C:\Users\eggfr\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: F
utureWarning: Function get_feature_names is deprecated; get_feature_names is
deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_ou
t instead.
  warnings.warn(msg, category=FutureWarning)
C:\Users\eggfr\AppData\Local\Temp\ipykernel_7372\1483046653.py:33: UserWarnin
g: Transformer onehotnorm (type MaxAbsScaler) does not provide get_feature_na
mes. Will return input column names if available
  warnings.warn("Transformer %s (type %s) does not "
C:\Users\eggfr\AppData\Local\Temp\ipykernel_7372\1483046653.py:33: UserWarnin
g: Transformer onehotimputer (type KNNImputer) does not provide get_feature_n
ames. Will return input column names if available
```

In [354]:
```
df2 = df1.sort_values([0], ascending=False)
ax = df2.head(10).plot(kind='barh')
ax.set_xlabel("Feature Importance")
ax.set_ylabel("Feature")
ax.set_title("Top 10 features for logistic regression prediction")
ax.get_legend().remove()
```



Recommendation

Everyone know about H1N1 disease, but people are not highly concerned about it. People with chronical medical condition, have seasonal vaccine, who is a health worker, and with children under 6 months are in the top 10 highest feature scores for the Logistic Regression prediction. It shows that if people are aware of their health condition, they are highly going to get vaccinated. From our EDA analysis,there is a lot of people doesnt think H1N1 are risky without vaccination. Therefore, even they are some concerns about H1N1, but they are not vaccinated as they don't think its risky with vaccination. I would suggest we target people with high educational group about the H1N1 risk to boost the H1n1 vaccination rate.

Future work

In this study, employment_occupation and employment_industry due to data issues, and I think they should be considered to further study to see if working condition would have an effect on higher H1N1 risk awareness to provide a higher vaccination rate. Also, more complicated models such as Random Forrest should be used to investigate the data.