phase-3-project (/github/kcoop610/phase-3-project/tree/main)
/ report.ipynb (/github/kcoop610/phase-3-project/tree/main/report.ipynb)

Predicting Diabetes Diagnosis

In 2015, diabetes was the seventh leading cause of death in the United States. More than 30 million Americans are living with diabetes, and 86 million are living with prediabetes, which is a serious health condition that increases a person’s risk of type 2 diabetes and other chronic diseases. The prevalence of diabetes and overweight (one of the major risk factors for diabetes) continue to increase. Substantial new efforts to prevent or control diabetes have begun, including the Diabetes Prevention Trial and the National Diabetes Education Program.

Diabetes is diagnosed by Glycated Hemoglobin (A1C) levels, which indicate a person's blood sugar levels and are measured via a blood test. However, many organizations aiming to prevent diabetes and other chronic conditions such as public health agencies and insurance providers don't have access to this sensitive information.

This report seeks to predict the presence of diabetes or prediabetes using only information that could be gathered by a survey or basic body measurements.

Using this model, stakeholders invested in public health can target participants in health and wellness programs which aim to reverse prediabetes, prevent diabetes, and manage the disease.

Potential stakeholders:

- Public health agencies
- Insurance companies
- Employer benefits coordinators

- Health & wellness companies

Data Source

National Health and Nutrition Examination Survey (NHANES)

The National Health and Nutrition Examination Survey (NHANES) (https://www.cdc.gov/nchs/nhanes/about_nhanes.htm) is a program of continuous studies designed to assess the health and nutritional status of adults and children in the United States. The survey examines a nationally representative sample of about 5,000 persons located across the country each year. The survey is unique in that it combines interviews and physical examinations. The NHANES interview includes demographic, socioeconomic, dietary, and health-related questions. The examination component consists of medical, dental, and physiological measurements, as well as laboratory tests administered by highly trained medical personnel.

NHANES is a major program of the National Center for Health Statistics (NCHS). NCHS is part of the Centers for Disease Control and Prevention (CDC) and has the responsibility for producing vital and health statistics for the Nation.

Subset for Classification Model

While NHANES collects a wealth of demographic, laboratory, and medical data, this analysis and classification model uses a subset comprised of:

- **Demographics, Smoking, Physical Activity** - surveys collected by trained interviewers using Computer-Assisted Personal Interview (CAPI) system in either English or Spanish, sometimes with assistance from an interpreter
- **Body Measures** - measured by trained health technicians in the Mobile Examination Center (MEC)
- **Pulse** - three oscillometric pulse readings were recorded by trained health technicians, then averaged together for a single representative pulse measurement
- **A1C** - whole blood specimens were processed, stored, and shipped to the University of Missouri-Columbia, MO for analysis.

Diabetes/prediabetes diagnosis was calculated based on the A1C level using the Mayo Clinic's guidelines.

As is typical for survey data, there are some filler responses such as "don't know" and "refused" in addition to null values. Each of these were evaluated, and one of the following steps was taken:

- Filled nulls/fillers with an assumed response if the number of responses in this class is low and there was a straightforward "safe" assumption. For example, "don't know" response to question about military service was assumed to be a "no."
- Created a "no_answer" class if the lack of clear answer seemed to potentially inicate some relevant finding. For example, if a person responded "don't know" to health insurance coverage, this may be a relevant predictor of their health.
- Replaced with a null value, to be addressed using an intelligent imputer method later in the process after the data has been split into training and testing samples.

In [1]:

```
import pandas as pd
pd.set_option('display.max_columns', 0)
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# styling notebook
from IPython.core.display import HTML
def css_styling():
    styles = open("./styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[1]:

In [2]:

```
demographic = pd.read_sas('./data/NHANES2017-2018_demographic.xpt')
insurance = pd.read_sas('./data/NHANES2017-2018_insurance.xpt')
measures = pd.read_sas('./data/NHANES2017-2018_body_measures.xpt')
activity = pd.read_sas('./data/NHANES2017-2018_physical_activity.xpt')
smoking = pd.read_sas('./data/NHANES2017-2018_smoking.xpt')
bp = pd.read_sas('./data/NHANES2017-2018_blood_pressure_oscillometric.x
a1c = pd.read_sas('./data/NHANES2017-2018_a1c.xpt')

# Datasets considered, but not used for this model
# chol_total = pd.read_sas('./data/NHANES2017-2018_total_cholesterol.x
# chol_hdl = pd.read_sas('./data/NHANES2017-2018_hdl_cholesterol.xpt')
# chol_ldl = pd.read_sas('./data/NHANES2017-2018_ldl_cholesterol.xpt')
# insulin = pd.read_sas('./data/NHANES2017-2018_insulin.xpt')
```

In [3]:

```
data = [a1c, demographic, insurance, measures, activity, smoking, bp]
```

In [4]:
```
for f in data:
    f.SEQN = f.SEQN.map(lambda x: int(x))
    f.set_index('SEQN', inplace=True)
    display(f.info())
    print('*****'*15)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6401 entries, 93705 to 102956
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   LBXGH   6045 non-null   float64
dtypes: float64(1)
memory usage: 100.0 KB

None

***************************************************************************
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9254 entries, 93703 to 102956
Data columns (total 45 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   SDDSRVYR  9254 non-null   float64
 1   RIDSTATR  9254 non-null   float64
 2   RIAGENDR  9254 non-null   float64
 3   RIDAGEYR  9254 non-null   float64
 4   RIDAGEMN  597 non-null    float64
 5   RIDRETH1  9254 non-null   float64
 6   RIDRETH3  9254 non-null   float64
 7   RIDEXMON  8704 non-null   float64
 8   RIDEXAGM  3433 non-null   float64
 9   DMQMILIZ  6004 non-null   float64
 10  DMQADFC   561 non-null    float64
 11  DMDBORN4  9254 non-null   float64
 12  DMDCITZN  9251 non-null   float64
 13  DMDYRSUS  1948 non-null   float64
 14  DMDEDUC3  2306 non-null   float64
 15  DMDEDUC2  5569 non-null   float64
 16  DMDMARTL  5569 non-null   float64
 17  RIDEXPRG  1110 non-null   float64
 18  SIALANG   9254 non-null   float64
 19  SIAPROXY  9254 non-null   float64
 20  SIAINTRP  9254 non-null   float64
 21  FIALANG   8780 non-null   float64
 22  FIAPROXY  8780 non-null   float64
 23  FIAINTRP  8780 non-null   float64
 24  MIALANG   6684 non-null   float64
 25  MIAPROXY  6684 non-null   float64
 26  MIAINTRP  6684 non-null   float64
 27  AIALANGA  4977 non-null   float64
 28  DMDHHSIZ  9254 non-null   float64
 29  DMDFMSIZ  9254 non-null   float64
 30  DMDHHSZA  9254 non-null   float64
 31  DMDHHSZB  9254 non-null   float64
 32  DMDHHSZE  9254 non-null   float64
 33  DMDHRGND  9254 non-null   float64
```

```
 34  DMDHRAGZ  9254 non-null   float64
 35  DMDHREDZ  8764 non-null   float64
 36  DMDHRMAZ  9063 non-null   float64
 37  DMDHSEDZ  4751 non-null   float64
 38  WTINT2YR  9254 non-null   float64
 39  WTMEC2YR  9254 non-null   float64
 40  SDMVPSU   9254 non-null   float64
 41  SDMVSTRA  9254 non-null   float64
 42  INDHHIN2  8763 non-null   float64
 43  INDFMIN2  8780 non-null   float64
 44  INDFMPIR  8023 non-null   float64
dtypes: float64(45)
memory usage: 3.2 MB

None

***********************************************************************
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9254 entries, 93703 to 102956
Data columns (total 15 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   HIQ011    9254 non-null   float64
 1   HIQ031A   4254 non-null   float64
 2   HIQ031B   1582 non-null   float64
 3   HIQ031C   58 non-null     float64
 4   HIQ031D   2527 non-null   float64
 5   HIQ031E   93 non-null     float64
 6   HIQ031F   295 non-null    float64
 7   HIQ031H   533 non-null    float64
 8   HIQ031I   301 non-null    float64
 9   HIQ031J   708 non-null    float64
 10  HIQ031AA  1 non-null      float64
 11  HIQ260    172 non-null    float64
 12  HIQ105    1141 non-null   float64
 13  HIQ270    8171 non-null   float64
 14  HIQ210    8171 non-null   float64
dtypes: float64(15)
memory usage: 1.1 MB

None

***********************************************************************
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8704 entries, 93703 to 102956
Data columns (total 20 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   BMDSTATS  8704 non-null   float64
 1   BMXWT     8580 non-null   float64
 2   BMIWT     416 non-null    float64
 3   BMXRECUM  894 non-null    float64
 4   BMIRECUM  24 non-null     float64
 5   BMXHEAD   194 non-null    float64
 6   BMIHEAD   0 non-null      float64
 7   BMXHT     8016 non-null   float64
 8   BMIHT     99 non-null     float64
 9   BMXBMI    8005 non-null   float64
```

```
 10  BMXLEG    6703 non-null   float64
 11  BMILEG    334 non-null    float64
 12  BMXARML   8177 non-null   float64
 13  BMIARML   347 non-null    float64
 14  BMXARMC   8173 non-null   float64
 15  BMIARMC   350 non-null    float64
 16  BMXWAIST  7601 non-null   float64
 17  BMIWAIST  437 non-null    float64
 18  BMXHIP    6039 non-null   float64
 19  BMIHIP    270 non-null    float64
dtypes: float64(20)
memory usage: 1.4 MB

None

*************************************************************************
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5856 entries, 93705 to 102956
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   PAQ605  5856 non-null   float64
 1   PAQ610  1389 non-null   float64
 2   PAD615  1381 non-null   float64
 3   PAQ620  5856 non-null   float64
 4   PAQ625  2439 non-null   float64
 5   PAD630  2426 non-null   float64
 6   PAQ635  5856 non-null   float64
 7   PAQ640  1439 non-null   float64
 8   PAD645  1430 non-null   float64
 9   PAQ650  5856 non-null   float64
 10  PAQ655  1434 non-null   float64
 11  PAD660  1431 non-null   float64
 12  PAQ665  5856 non-null   float64
 13  PAQ670  2308 non-null   float64
 14  PAD675  2301 non-null   float64
 15  PAD680  5846 non-null   float64
dtypes: float64(16)
memory usage: 777.8 KB

None

*************************************************************************
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6724 entries, 93705 to 102956
Data columns (total 36 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   SMQ020    5856 non-null   float64
 1   SMD030    2359 non-null   float64
 2   SMQ040    2359 non-null   float64
 3   SMQ050Q   1338 non-null   float64
 4   SMQ050U   1255 non-null   float64
 5   SMD057    1338 non-null   float64
 6   SMQ078    793 non-null    float64
 7   SMD641    1063 non-null   float64
 8   SMD650    1022 non-null   float64
 9   SMD093    1021 non-null   float64
```

```
 10  SMDUPCA   6724 non-null   object 
 11  SMD100BR  6724 non-null   object 
 12  SMD100FL  929 non-null    float64
 13  SMD100MN  929 non-null    float64
 14  SMD100LN  929 non-null    float64
 15  SMD100TR  695 non-null    float64
 16  SMD100NI  695 non-null    float64
 17  SMD100CO  695 non-null    float64
 18  SMQ621    821 non-null    float64
 19  SMD630    42 non-null     float64
 20  SMQ661    14 non-null     float64
 21  SMQ665A   3 non-null      float64
 22  SMQ665B   1 non-null      float64
 23  SMQ665C   4 non-null      float64
 24  SMQ665D   2 non-null      float64
 25  SMQ670    1035 non-null   float64
 26  SMQ848    528 non-null    float64
 27  SMQ852Q   522 non-null    float64
 28  SMQ852U   519 non-null    float64
 29  SMQ890    5856 non-null   float64
 30  SMQ895    2095 non-null   float64
 31  SMQ900    5856 non-null   float64
 32  SMQ905    1150 non-null   float64
 33  SMQ910    5856 non-null   float64
 34  SMQ915    861 non-null    float64
 35  SMAQUEX2  6724 non-null   float64
dtypes: float64(34), object(2)
memory usage: 1.9+ MB


None

*************************************************************************
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7132 entries, 93705 to 102956
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   BPAOARM   7132 non-null   object 
 1   BPAOCSZ   6144 non-null   float64
 2   BPAOMNTS  6144 non-null   float64
 3   BPXOSY1   6143 non-null   float64
 4   BPXODI1   6143 non-null   float64
 5   BPXOSY2   6123 non-null   float64
 6   BPXODI2   6123 non-null   float64
 7   BPXOSY3   6094 non-null   float64
 8   BPXODI3   6094 non-null   float64
 9   BPXOPLS1  5262 non-null   float64
 10  BPXOPLS2  5244 non-null   float64
 11  BPXOPLS3  5220 non-null   float64
dtypes: float64(11), object(1)
memory usage: 724.3+ KB

None

*************************************************************************
```

Demographics

View source for additional description and details (https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/DEMO_J.htm)

The demographics file contains information about the individual respondent, their family, and their household, including age, gender, race, education level, marital status, military service status, country of birth, US citizenship, household and family composition and income.

In [5]:
```
demographic.describe().round(1)
```

Out[5]:

	SDDSRVYR	RIDSTATR	RIAGENDR	RIDAGEYR	RIDAGEMN	RIDRETH1	RIDRETH3
count	9254.0	9254.0	9254.0	9254.0	597.0	9254.0	9254.0
mean	10.0	1.9	1.5	34.3	10.4	3.2	3.5
std	0.0	0.2	0.5	25.5	7.1	1.3	1.7
min	10.0	1.0	1.0	0.0	0.0	1.0	1.0
25%	10.0	2.0	1.0	11.0	4.0	3.0	3.0
50%	10.0	2.0	2.0	31.0	10.0	3.0	3.0
75%	10.0	2.0	2.0	58.0	17.0	4.0	4.0
max	10.0	2.0	2.0	80.0	24.0	5.0	7.0

In [6]:
```
demographic.columns
```

Out[6]:
```
Index(['SDDSRVYR', 'RIDSTATR', 'RIAGENDR', 'RIDAGEYR', 'RIDAGEMN', 'RII
       'RIDRETH3', 'RIDEXMON', 'RIDEXAGM', 'DMQMILIZ', 'DMQADFC', 'DMDI
       'DMDCITZN', 'DMDYRSUS', 'DMDEDUC3', 'DMDEDUC2', 'DMDMARTL', 'RII
       'SIALANG', 'SIAPROXY', 'SIAINTRP', 'FIALANG', 'FIAPROXY', 'FIAII
       'MIALANG', 'MIAPROXY', 'MIAINTRP', 'AIALANGA', 'DMDHHSIZ', 'DMDI
       'DMDHHSZA', 'DMDHHSZB', 'DMDHHSZE', 'DMDHRGND', 'DMDHRAGZ', 'DMI
       'DMDHRMAZ', 'DMDHSEDZ', 'WTINT2YR', 'WTMEC2YR', 'SDMVPSU', 'SDM\
       'INDHHIN2', 'INDFMIN2', 'INDFMIPIR'],
      dtype='object')
```

After reviewing the source documentation, only a handful of features which are easily interpretable and may be relevant to predicting diabetes/prediabetes were kept for analysis.

In [7]:
```
keepcols_demographic = ['RIAGENDR', 'RIDAGEYR', 'RIDRETH3', 'DMQMILIZ',
                        'DMDEDUC2', 'DMDMARTL', 'RIDEXPRG', 'DMDHHSIZ', 'II
```

In [8]:

```
keep_demographic = demographic.copy()[keepcols_demographic]
keep_demographic.rename(mapper={'RIAGENDR': 'gender',
                                'RIDAGEYR': 'age',
                                'RIDRETH3': 'race',
                                'DMQMILIZ': 'veteran_status',
                                'DMDBORN4': 'country_of_birth',
                                'DMDCITZN': 'citizen_status',
                                'DMDEDUC2': 'education',
                                'DMDMARTL': 'marital_status',
                                'RIDEXPRG': 'pregnancy_status',
                                'INDHHIN2': 'annual_household_income',
                                'INDFMPIR': 'income_poverty_ratio',
                                'DMDHHSIZ': 'household_size'},
                        axis=1, inplace=True)
keep_demographic.head()
```

Out[8]:

SEQN	gender	age	race	veteran_status	country_of_birth	citizen_status	education	mai
93703	2.0	2.0	6.0	NaN	1.0	1.0	NaN	
93704	1.0	2.0	3.0	NaN	1.0	1.0	NaN	
93705	2.0	66.0	4.0	2.0	1.0	1.0	2.0	
93706	1.0	18.0	6.0	2.0	1.0	1.0	NaN	
93707	1.0	13.0	7.0	NaN	1.0	1.0	NaN	

In [9]:

```
keep_demographic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9254 entries, 93703 to 102956
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   gender                   9254 non-null   float64
 1   age                      9254 non-null   float64
 2   race                     9254 non-null   float64
 3   veteran_status           6004 non-null   float64
 4   country_of_birth         9254 non-null   float64
 5   citizen_status           9251 non-null   float64
 6   education                5569 non-null   float64
 7   marital_status           5569 non-null   float64
 8   pregnancy_status         1110 non-null   float64
 9   household_size           9254 non-null   float64
 10  annual_household_income  8763 non-null   float64
 11  income_poverty_ratio     8023 non-null   float64
dtypes: float64(12)
memory usage: 939.9 KB
```

Questionnaire responses are coded, which makes interpretation difficult. The following cells explore each feature, decode values, and decipher response options.

In [10]:
```
keep_demographic['gender'].value_counts(dropna=False)
```

Out[10]:
```
2.0    4697
1.0    4557
Name: gender, dtype: int64
```

In [11]:
```
keep_demographic['gender'].replace({1.0: 'male', 2.0: 'female'}, inplac
```

In [12]:
```
keep_demographic['gender'].value_counts(dropna=False)
```

Out[12]:
```
female    4697
male      4557
Name: gender, dtype: int64
```

In [13]:
```
keep_demographic['age'].describe().round(2)
```

Out[13]:
```
count    9254.00
mean       34.33
std        25.50
min         0.00
25%        11.00
50%        31.00
75%        58.00
max        80.00
Name: age, dtype: float64
```

Note - Individuals aged 80 and over are topcoded at 80. In NHANES 2017-2018, the weighted mean age for participants 80+ is 85

In [14]:
```
keep_demographic['age'] = keep_demographic['age'].astype(int)
```

This model focuses on adults age 18+.

In [15]:
```
# remove if age less than 18
keep_demographic = keep_demographic[keep_demographic['age']>=18]
```

In [16]:
```
keep_demographic['age'].describe().astype(int)
```

Out[16]:
```
count    5856
mean       49
std        18
min        18
25%        33
50%        51
75%        65
max        80
Name: age, dtype: int64
```

In [17]:
```
keep_demographic['race'].value_counts(dropna=False)
```

Out[17]:
```
3.0    2032
4.0    1343
6.0     849
1.0     792
2.0     543
7.0     297
Name: race, dtype: int64
```

In [18]:
```
keep_demographic['race'].replace({1.0: 'mexican_american',
                                  2.0: 'hispanic',
                                  3.0: 'white',
                                  4.0: 'black',
                                  6.0: 'asian',
                                  7.0: 'other'},
                                 inplace=True)
```

In [19]:
```
keep_demographic['race'].value_counts(dropna=False)
```

Out[19]:
```
white               2032
black               1343
asian                849
mexican_american     792
hispanic             543
other                297
Name: race, dtype: int64
```

In [20]:
```
keep_demographic['veteran_status'].value_counts(dropna=False)
```

Out[20]:
```
2.0    5292
1.0     561
7.0       2
9.0       1
Name: veteran_status, dtype: int64
```

In [21]:
```
keep_demographic['veteran_status'].replace({1.0: 'yes',
                                            2.0: 'no',
                                            7.0: 'no',
                                            9.0: 'no',},
                                           inplace=True)
```

In [22]:
```
keep_demographic['veteran_status'].value_counts(dropna=False)
```

Out[22]:
```
no     5295
yes     561
Name: veteran_status, dtype: int64
```

In [23]:
```
keep_demographic['country_of_birth'].value_counts(dropna=False)
```

Out[23]:
```
1.0     4067
2.0     1786
77.0       2
99.0       1
Name: country_of_birth, dtype: int64
```

In [24]:
```
keep_demographic['country_of_birth'].replace({1.0: 'usa',
                                               2.0: 'other',
                                               77.0: 'usa',
                                               99.0: 'usa'},
                                              inplace=True)
```

In [25]:
```
keep_demographic['country_of_birth'].value_counts(dropna=False)
```

Out[25]:
```
usa      4070
other    1786
Name: country_of_birth, dtype: int64
```

In [26]:
```
keep_demographic['citizen_status'].value_counts(dropna=False)
```

Out[26]:
```
1.0    5030
2.0     801
7.0      15
9.0       7
NaN       3
Name: citizen_status, dtype: int64
```

In [27]:
```
# kept a "no_answer" class since this may be meaningful
keep_demographic['citizen_status'].replace({1.0: 'citizen',
                                             2.0: 'non_citizen',
                                             7.0: 'no_answer',
                                             9.0: 'no_answer',
                                             np.nan: 'no_answer'},
                                            inplace=True)
```

In [28]:
```
keep_demographic['education'].value_counts(dropna=False)
```

Out[28]:
```
4.0    1778
5.0    1336
3.0    1325
2.0     638
1.0     479
NaN     287
9.0      11
7.0       2
Name: education, dtype: int64
```

In [29]:
```
# these nulls are not as easy to fill in, so I'll use an imputer at a :
keep_demographic['education'].replace({1.0: 'no_diploma',
                                       2.0: 'no_diploma',
                                       3.0: 'highschool_grad',
                                       4.0: 'some_college',
                                       5.0: 'college_grad',
                                       7.0: np.nan,
                                       9.0: np.nan}, inplace=True)
```

In [30]:
```
keep_demographic['marital_status'].value_counts(dropna=False)
```

Out[30]:
```
1.0     2737
5.0     1006
3.0      641
6.0      515
2.0      462
NaN      287
4.0      202
77.0       6
Name: marital_status, dtype: int64
```

In [31]:
```
keep_demographic['marital_status'].replace({1.0: 'married or living wit
                                            2.0: 'widowed',
                                            3.0: 'divorced/separated',
                                            4.0: 'divorced/separated',
                                            5.0: 'never married',
                                            6.0: 'married or living wit
                                            77.0: 'no_answer'}, inplace
```

In [32]:
```
keep_demographic[keep_demographic['marital_status'].isna()].sort_values
```

Out[32]:

	gender	age	race	veteran_status	country_of_birth	citizen_status	edu
SEQN							
93706	male	18	asian	no	usa	citizen	
98618	male	18	mexican_american	no	other	non_citizen	
98695	male	18	black	no	usa	citizen	
98697	male	18	white	no	usa	citizen	
98829	male	18	mexican_american	no	usa	citizen	
...	...	...	...	...	...	...	
98866	male	19	white	no	usa	citizen	
95585	male	19	mexican_american	no	usa	citizen	
98938	female	19	white	no	usa	citizen	
99013	female	19	mexican_american	no	usa	citizen	
102837	female	19	hispanic	no	usa	citizen	

287 rows × 12 columns

All null values in the marital status column are 18 or 19 year olds, so I will fill the nulls with "never married."

In [33]:
```
keep_demographic['marital_status'].fillna('never married', inplace=True
```

In [34]:
```
keep_demographic['marital_status'].value_counts(dropna=False)
```

Out[34]:
```
married or living with partner    3252
never married                     1293
divorced/separated                 843
widowed                            462
no_answer                            6
Name: marital_status, dtype: int64
```

In [35]:
```
keep_demographic['pregnancy_status'].value_counts(dropna=False)
```

Out[35]:
```
NaN    4746
2.0     966
3.0      89
1.0      55
Name: pregnancy_status, dtype: int64
```

Per the data source, respondents were coded with a value 3 "could not be determined" if (1) the respondent reported did not know her pregnancy status and the urine test was negative, or (2) if the respondent was interviewed but not examined

Because the urine test was negative in first scenario, the responded is assumed to be not pregnant.

Respondents who fall into the second scenario will be removed from the sample later on when data files are merged, as it's likely that the "examination" would have also included the A1C test. Because this is a supervised learning activity, only respondents with valid A1C levels will be kept in the sample.

Null values are assumed to indicate "not pregnant".

Because pregnancy may cause fluctuations in weight and body measurements, respondents who are pregnant are dropped from the sample.

In [36]:
```
keep_demographic = keep_demographic[keep_demographic['pregnancy_status'
```

In [37]:
```
keep_demographic.drop(columns='pregnancy_status', inplace=True)
```

In [38]:
```
keep_demographic.head()
```

Out[38]:

	gender	age	race	veteran_status	country_of_birth	citizen_status	education	di
SEQN								
93705	female	66	black	no	usa	citizen	no_diploma	di
93706	male	18	asian	no	usa	citizen	NaN	
93708	female	66	asian	no	other	citizen	no_diploma	
93709	female	75	black	no	usa	citizen	some_college	
93711	male	56	asian	no	other	citizen	college_grad	

While multiple income-related questions were surveyed, the feature most meaningful for this model is annual household income. This number would include the combined incomes of all members of a household, which may be comprised of a single family/individual, more than one family, more than one unrelated individuals, or any combination.

Income ranges were captured as the response to this question. In order to create a continuous numeric variable, these ranges were replaced with the highest number in the range for purposes of this model. For example, a household reporting annual income between 45,000 and 54,999 was recoded to 55,000. The 100k+ category was coded as 150,000.

Some respondents only reported whether their income was under 20k (197 total respondents), or above 20k (69 total respondents). Those reponding <20k were recoded as 15,000 since this is likely a larger range with more actual values falling below the maximum. Those responding >20k were recoded as 45,000.

In [39]:
```
keep_demographic['annual_household_income'].value_counts(dropna=False)
```

Out[39]:
```
15.0    988
6.0     574
7.0     570
14.0    497
8.0     389
NaN     345
5.0     344
4.0     335
9.0     331
10.0    277
3.0     264
12.0    220
2.0     164
1.0     164
99.0    143
77.0    121
13.0     75
Name: annual_household_income, dtype: int64
```

In [40]:
```
# these nulls are not as easy to fill in, so I'll use an imputer at a :
keep_demographic['annual_household_income'].replace({1.0: 5000,
                                                     2.0: 10000,
                                                     3.0: 15000,
                                                     4.0: 20000,
                                                     5.0: 25000,
                                                     6.0: 35000,
                                                     7.0: 45000,
                                                     8.0: 55000,
                                                     9.0: 65000,
                                                     10.0: 75000,
                                                     12.0: 15000,
                                                     13.0: 45000,
                                                     14.0: 100000,
                                                     15.0: 150000,
                                                     77.0: np.nan,
                                                     99.0: np.nan}, inp
```

In [41]:
```
keep_demographic['annual_household_income'].value_counts(dropna=False)
```

Out[41]:
```
150000.0    988
45000.0     645
NaN         609
35000.0     574
100000.0    497
15000.0     484
55000.0     389
25000.0     344
20000.0     335
65000.0     331
75000.0     277
5000.0      164
10000.0     164
Name: annual_household_income, dtype: int64
```

In [42]:
```
keep_demographic['household_size'].value_counts(dropna=False)
```

Out[42]:
```
2.0    1787
3.0    1057
4.0     918
1.0     806
5.0     638
6.0     334
7.0     261
Name: household_size, dtype: int64
```

In [43]:
```
keep_demographic['household_size'] = keep_demographic['household_size']
```

In [44]:
```
keep_demographic['income_poverty_ratio'].describe().round(2)
```

Out[44]:
```
count    4975.00
mean        2.52
std         1.61
min         0.00
25%         1.18
50%         2.08
75%         4.06
max         5.00
Name: income_poverty_ratio, dtype: float64
```

In [45]:
```
keep_demographic['income_poverty_ratio'].isna().sum()
```

Out[45]:
```
826
```

Per the data source:

> Income-Poverty Ratio was calculated by dividing family (or individual) income by the poverty guidelines specific to the survey year. The value was not computed if the respondent only reported income as < 20,000 or ≥ 20,000. If family income was reported as a more detailed category, the midpoint of the range was used to compute the ratio. Values at or above 5.00 were coded as 5.00 or more because of disclosure concerns. The values were not computed if the income data was missing.

In [46]:
```
keep_demographic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5801 entries, 93705 to 102956
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   gender                   5801 non-null   object 
 1   age                      5801 non-null   int64  
 2   race                     5801 non-null   object 
 3   veteran_status           5801 non-null   object 
 4   country_of_birth         5801 non-null   object 
 5   citizen_status           5801 non-null   object 
 6   education                5501 non-null   object 
 7   marital_status           5801 non-null   object 
 8   household_size           5801 non-null   int64  
 9   annual_household_income  5192 non-null   float64
 10  income_poverty_ratio     4975 non-null   float64
dtypes: float64(2), int64(2), object(7)
memory usage: 543.8+ KB
```

Health Insurance

View source for additional description and details (https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/HIQ_J.htm)

The Health Insurance questionnaire provides respondent-level interview data on insurance coverage, type of insurance coverage, coverage of prescription drugs, and uninsured status during the past 12 months.

In [47]:
```
insurance
```

Out[47]:

	HIQ011	HIQ031A	HIQ031B	HIQ031C	HIQ031D	HIQ031E	HIQ031F	HIQ031H
SEQN								
93703	1.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN
93704	1.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN
93705	1.0	NaN	15.0	NaN	17.0	NaN	NaN	NaN
93706	1.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN
93707	1.0	NaN	NaN	NaN	17.0	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
102952	1.0	14.0	15.0	NaN	NaN	NaN	NaN	NaN
102953	1.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN
102954	1.0	NaN	NaN	NaN	17.0	NaN	NaN	NaN
102955	1.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN
102956	1.0	NaN	NaN	NaN	NaN	NaN	NaN	21.0

9254 rows × 15 columns

In [48]:
```
insurance.columns
```

Out[48]:
```
Index(['HIQ011', 'HIQ031A', 'HIQ031B', 'HIQ031C', 'HIQ031D', 'HIQ031E',
       'HIQ031F', 'HIQ031H', 'HIQ031I', 'HIQ031J', 'HIQ031AA', 'HIQ260
       'HIQ105', 'HIQ270', 'HIQ210'],
      dtype='object')
```

In [49]:
```
keepcols_insurance = ['HIQ011', 'HIQ031A', 'HIQ031B', 'HIQ031C', 'HIQ0:
       'HIQ031F', 'HIQ031H', 'HIQ031I', 'HIQ031J', 'HIQ031AA', 'HIQ270'

mapper_insurance = {'HIQ011': 'coverage_status',
          'HIQ031A': 'covered_private',
          'HIQ031B': 'covered_medicare',
          'HIQ031C': 'covered_medigap',
          'HIQ031D': 'covered_medicaid',
          'HIQ031E': 'covered_chip',
          'HIQ031F': 'covered_military',
          'HIQ031H': 'covered_state',
          'HIQ031I': 'covered_other_gov',
          'HIQ031J': 'covered_single_service',
          'HIQ031AA': 'not_covered',
          'HIQ270': 'prescription_coverage',
          'HIQ210': 'uninsured_in_last_year'}
```

In [50]:
```
keep_insurance = insurance.copy()[keepcols_insurance]
keep_insurance.rename(mapper=mapper_insurance, axis=1, inplace=True)
keep_insurance
```

Out[50]:

	coverage_status	covered_private	covered_medicare	covered_medigap	covered_m
SEQN					
93703	1.0	14.0	NaN	NaN	
93704	1.0	14.0	NaN	NaN	
93705	1.0	NaN	15.0	NaN	
93706	1.0	14.0	NaN	NaN	
93707	1.0	NaN	NaN	NaN	
...	...	...	...	...	
102952	1.0	14.0	15.0	NaN	
102953	1.0	14.0	NaN	NaN	
102954	1.0	NaN	NaN	NaN	
102955	1.0	14.0	NaN	NaN	
102956	1.0	NaN	NaN	NaN	

9254 rows × 13 columns

In [51]:
```
keep_insurance['coverage_status'].value_counts(dropna=False)
```

Out[51]:
```
1.0    8157
2.0    1072
9.0      18
7.0       7
Name: coverage_status, dtype: int64
```

In [52]:
```
# kept a "no_answer" class as this may be meaningful
keep_insurance['coverage_status'].replace({1.0: 'insured',
                                           2.0: 'uninsured',
                                           7.0: 'no_answer',
                                           9.0: 'no_answer'}, inplace=True)
```

In [53]:
```
keep_insurance['coverage_status'].value_counts(dropna=False)
```

Out[53]:
```
insured      8157
uninsured    1072
no_answer      25
Name: coverage_status, dtype: int64
```

In [54]:
```
keep_insurance['covered_private'].value_counts(dropna=False)
```

Out[54]:
```
NaN     5000
14.0    4188
99.0      63
77.0       3
Name: covered_private, dtype: int64
```

In [55]:
```
keep_insurance['covered_private'].replace({14.0: 'yes',
                                            99.0: 'no',
                                            77.0: 'no',
                                            np.nan: 'no'}, inplace=True)
```

In [56]:
```
keep_insurance['covered_medicare'].value_counts(dropna=False)
```

Out[56]:
```
NaN     7672
15.0    1582
Name: covered_medicare, dtype: int64
```

In [57]:
```
keep_insurance['covered_medicare'].replace({15.0: 'yes', np.nan: 'no'},
```

In [58]:
```
keep_insurance['covered_medigap'].value_counts(dropna=False)
```

Out[58]:
```
NaN     9196
16.0      58
Name: covered_medigap, dtype: int64
```

In [59]:
```
keep_insurance['covered_medigap'].replace({16.0: 'yes', np.nan: 'no'},
```

In [60]:
```
keep_insurance['covered_medicaid'].value_counts(dropna=False)
```

Out[60]:
```
NaN     6727
17.0    2527
Name: covered_medicaid, dtype: int64
```

In [61]:
```
keep_insurance['covered_medicaid'].replace({17.0: 'yes', np.nan: 'no'},
```

In [62]:
```
keep_insurance['covered_chip'].value_counts(dropna=False)
```

Out[62]:
```
NaN     9161
18.0      93
Name: covered_chip, dtype: int64
```

In [63]:
```
keep_insurance['covered_chip'].replace({18.0: 'yes', np.nan: 'no'}, inp
```

In [64]:
```
keep_insurance['covered_military'].value_counts(dropna=False)
```

Out[64]:
```
NaN     8959
19.0     295
Name: covered_military, dtype: int64
```

In [65]:
```
keep_insurance['covered_military'].replace({19.0: 'yes', np.nan: 'no'},
```

In [66]:
```
keep_insurance['covered_state'].value_counts(dropna=False)
```

Out[66]:
```
NaN     8721
21.0     533
Name: covered_state, dtype: int64
```

In [67]:
```
keep_insurance['covered_state'].replace({21.0: 'yes', np.nan: 'no'}, ir
```

In [68]:
```
keep_insurance['covered_other_gov'].value_counts(dropna=False)
```

Out[68]:
```
NaN     8953
22.0     301
Name: covered_other_gov, dtype: int64
```

In [69]:
```
keep_insurance['covered_other_gov'].replace({22.0: 'yes', np.nan: 'no'}
```

In [70]:
```
keep_insurance['covered_single_service'].value_counts(dropna=False)
```

Out[70]:
```
NaN     8546
23.0     708
Name: covered_single_service, dtype: int64
```

In [71]:
```
keep_insurance['covered_single_service'].replace({23.0: 'yes', np.nan:
```

In [72]:
```
keep_insurance['not_covered'].value_counts(dropna=False)
```

Out[72]:
```
NaN     9253
40.0       1
Name: not_covered, dtype: int64
```

In [73]:
```
#seems to be an anomoly, so I drop this row
keep_insurance[keep_insurance['not_covered']==40.0]
keep_insurance = keep_insurance[keep_insurance['not_covered']!=40.0]
```

In [74]:
```
# dropping the column since all values are now null
keep_insurance.drop(columns='not_covered', inplace=True)
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
  return super().drop(
```

In [75]:
```
keep_insurance.head()
```

Out[75]:

	coverage_status	covered_private	covered_medicare	covered_medigap	covered_me
SEQN					
93703	insured	yes	no	no	
93704	insured	yes	no	no	
93705	insured	no	yes	no	
93706	insured	yes	no	no	
93707	insured	no	no	no	

In [76]:
```
keep_insurance['prescription_coverage'].value_counts(dropna=False)
```

Out[76]:
```
1.0    7678
NaN    1082
2.0     379
9.0     110
7.0       4
Name: prescription_coverage, dtype: int64
```

In [77]:
```
keep_insurance['prescription_coverage'].replace({1.0: 'yes',
                                          2.0: 'no',
                                          7.0: 'no_answer',
                                          9.0: 'no_answer',
                                          np.nan: 'no_answer'}, inplace=
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
  return super().replace(
```

In [78]:
```
keep_insurance['prescription_coverage'].value_counts(dropna=False)
```

Out[78]:
```
yes          7678
no_answer    1196
no            379
Name: prescription_coverage, dtype: int64
```

In [79]:
```
keep_insurance['uninsured_in_last_year'].value_counts(dropna=False)
```

Out[79]:
```
2.0    7591
NaN    1082
1.0     561
9.0      18
7.0       1
Name: uninsured_in_last_year, dtype: int64
```

In [80]:

```
keep_insurance['uninsured_in_last_year'].replace({1.0: 'yes',
                                                  2.0: 'no',
                                                  7.0: 'no_answer',
                                                  9.0: 'no_answer'}, inplace=Tr
```

In [81]:

```
keep_insurance[keep_insurance['uninsured_in_last_year'].isna()]
```

Out[81]:

	coverage_status	covered_private	covered_medicare	covered_medigap	covered_m
SEQN					
93712	uninsured	no	no	no	
93717	uninsured	no	no	no	
93729	no_answer	no	no	no	
93730	uninsured	no	no	no	
93743	uninsured	no	no	no	
...	...	...	...	...	
102884	uninsured	no	no	no	
102898	uninsured	no	no	no	
102918	uninsured	no	no	no	
102921	uninsured	no	no	no	
102922	uninsured	no	no	no	

1082 rows × 12 columns

In [82]:

```
# determined that respondents who indicated their "coverage status" is
# have been uninsured in the last 12 months

for ind, row in keep_insurance.iterrows():
    if keep_insurance['coverage_status'][ind] == 'uninsured':
        keep_insurance['uninsured_in_last_year'][ind] = 'yes'
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [83]:
```
keep_insurance[keep_insurance['uninsured_in_last_year'].isna()]
```

Out[83]:

	coverage_status	covered_private	covered_medicare	covered_medigap	covered_m
SEQN					
93729	no_answer	no	no	no	
93902	no_answer	no	no	no	
93981	no_answer	no	no	no	
95301	no_answer	no	no	no	
95515	no_answer	no	no	no	
95567	no_answer	no	no	no	
95659	no_answer	no	no	no	
96934	no_answer	no	no	no	
97033	no_answer	no	no	no	
97306	no_answer	no	no	no	
97486	no_answer	no	no	no	
97531	no_answer	no	no	no	
97774	no_answer	no	no	no	
98146	no_answer	no	no	no	
98618	no_answer	no	no	no	
98628	no_answer	no	no	no	
98926	no_answer	no	no	no	
98962	no_answer	no	no	no	
99065	no_answer	no	no	no	
99576	no_answer	no	no	no	
100464	no_answer	no	no	no	
100601	no_answer	no	no	no	
100950	no_answer	no	no	no	
100981	no_answer	no	no	no	
101975	no_answer	no	no	no	

In [84]:
```
keep_insurance['uninsured_in_last_year'].fillna('no_answer', inplace=Tr
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
  return super().fillna(
```

In [85]:
```
keep_insurance['uninsured_in_last_year'].value_counts(dropna=False)
```

Out[85]:
```
no           7578
yes          1631
no_answer      44
Name: uninsured_in_last_year, dtype: int64
```

In [86]:
```
keep_insurance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9253 entries, 93703 to 102956
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   coverage_status         9253 non-null   object
 1   covered_private         9253 non-null   object
 2   covered_medicare        9253 non-null   object
 3   covered_medigap         9253 non-null   object
 4   covered_medicaid        9253 non-null   object
 5   covered_chip            9253 non-null   object
 6   covered_military        9253 non-null   object
 7   covered_state           9253 non-null   object
 8   covered_other_gov       9253 non-null   object
 9   covered_single_service  9253 non-null   object
 10  prescription_coverage   9253 non-null   object
 11  uninsured_in_last_year  9253 non-null   object
dtypes: object(12)
memory usage: 1.2+ MB
```

Body Measures

[View source for additional description and details (https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/BMX_J.htm)](https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/BMX_J.htm)

Body measures including height, weight, waist and hip circumference were taken for adult participants. Body Mass Index (BMI) was calculated as weight in kilograms divided by height in meters squared, and then rounded to one decimal place.

In [87]:
```
measures
```

Out[87]:

SEQN	BMDSTATS	BMXWT	BMIWT	BMXRECUM	BMIRECUM	BMXHEAD	BMIHEAD	BMXI
93703	1.0	13.7	3.0	89.6	NaN	NaN	NaN	8
93704	1.0	13.9	NaN	95.0	NaN	NaN	NaN	9
93705	1.0	79.5	NaN	NaN	NaN	NaN	NaN	15
93706	1.0	66.3	NaN	NaN	NaN	NaN	NaN	17
93707	1.0	45.4	NaN	NaN	NaN	NaN	NaN	15
...	...	...	...	...	...	...	...	
102952	1.0	49.0	NaN	NaN	NaN	NaN	NaN	15
102953	1.0	97.4	NaN	NaN	NaN	NaN	NaN	16
102954	1.0	69.1	NaN	NaN	NaN	NaN	NaN	16
102955	1.0	111.9	NaN	NaN	NaN	NaN	NaN	15
102956	1.0	111.5	NaN	NaN	NaN	NaN	NaN	17

8704 rows × 20 columns

In [88]:
```
measures.columns
```

Out[88]:
```
Index(['BMDSTATS', 'BMXWT', 'BMIWT', 'BMXRECUM', 'BMIRECUM', 'BMXHEAD',
       'BMIHEAD', 'BMXHT', 'BMIHT', 'BMXBMI', 'BMXLEG', 'BMILEG', 'BMX
       'BMIARML', 'BMXARMC', 'BMIARMC', 'BMXWAIST', 'BMIWAIST', 'BMXHI
       'BMIHIP'],
      dtype='object')
```

In [89]:
```
keepcols_measures = ['BMXWT', 'BMXHT', 'BMXBMI', 'BMXWAIST', 'BMXHIP']
```

In [90]:
```
keep_measures = measures.copy()[keepcols_measures]
keep_measures.rename(mapper={'BMXWT': 'weight_kg',
                             'BMXHT': 'height_cm',
                             'BMXBMI': 'BMI',
                             'BMXWAIST': 'waist_circumference_cm',
                             'BMXHIP': 'hip_circumference_cm'},
                     axis=1, inplace=True)
keep_measures
```

Out[90]:

	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_cm
SEQN					
93703	13.7	88.6	17.5	48.2	NaN
93704	13.9	94.2	15.7	50.0	NaN
93705	79.5	158.3	31.7	101.8	110.0
93706	66.3	175.7	21.5	79.3	94.4
93707	45.4	158.4	18.1	64.1	83.0
...	...	...	...	...	...
102952	49.0	156.5	20.0	82.2	87.3
102953	97.4	164.9	35.8	114.8	112.8
102954	69.1	162.6	26.1	86.4	102.7
102955	111.9	156.6	45.6	113.5	128.3
102956	111.5	175.8	36.1	122.0	110.0

8704 rows × 5 columns

In [91]:
```
keep_measures['weight_kg'].describe().round(1)
```

Out[91]:
```
count    8580.0
mean       65.1
std        32.9
min         3.2
25%        43.1
50%        67.8
75%        85.6
max       242.6
Name: weight_kg, dtype: float64
```

In [92]:
```
keep_measures['weight_kg'].isna().sum()
```

Out[92]:
```
124
```

In [93]:
```
keep_measures[keep_measures['weight_kg'].isna()]
```

Out[93]:

	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_cm
SEQN					
93777	NaN	NaN	NaN	NaN	NaN
93816	NaN	NaN	NaN	NaN	NaN
93935	NaN	NaN	NaN	NaN	NaN
93955	NaN	NaN	NaN	NaN	NaN
94310	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...
102590	NaN	NaN	NaN	NaN	NaN
102610	NaN	96.2	NaN	NaN	NaN
102671	NaN	NaN	NaN	NaN	NaN
102684	NaN	NaN	NaN	NaN	NaN
102864	NaN	NaN	NaN	NaN	NaN

124 rows × 5 columns

Because many of the 77 rows with missing weights are also missing heights and BMIs, they are removed from the dataset

In [94]:
```
keep_measures = keep_measures[~keep_measures['weight_kg'].isna()]
keep_measures
```

Out[94]:

	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_cm
SEQN					
93703	13.7	88.6	17.5	48.2	NaN
93704	13.9	94.2	15.7	50.0	NaN
93705	79.5	158.3	31.7	101.8	110.0
93706	66.3	175.7	21.5	79.3	94.4
93707	45.4	158.4	18.1	64.1	83.0
...	...	...	...	...	...
102952	49.0	156.5	20.0	82.2	87.3
102953	97.4	164.9	35.8	114.8	112.8
102954	69.1	162.6	26.1	86.4	102.7
102955	111.9	156.6	45.6	113.5	128.3
102956	111.5	175.8	36.1	122.0	110.0

8580 rows × 5 columns

In [95]:
```
keep_measures['weight_kg'].isna().sum()
```

Out[95]: 0

In [96]:
```
keep_measures['height_cm'].isna().sum()
```

Out[96]: 575

In [97]:
```
keep_measures[keep_measures['height_cm'].isna()]
```

Out[97]:

SEQN	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_cm
93710	10.2	NaN	NaN	NaN	NaN
93720	10.6	NaN	NaN	NaN	NaN
93748	9.3	NaN	NaN	NaN	NaN
93749	8.3	NaN	NaN	NaN	NaN
93764	9.2	NaN	NaN	NaN	NaN
...	...	...	...	...	...
102897	12.6	NaN	NaN	NaN	NaN
102919	7.2	NaN	NaN	NaN	NaN
102927	9.1	NaN	NaN	NaN	NaN
102936	10.2	NaN	NaN	NaN	NaN
102942	8.9	NaN	NaN	NaN	NaN

575 rows × 5 columns

Because a these rows seem to be missing a lot of data, they are removed from the dataset.

In [98]:
```
keep_measures = keep_measures[~keep_measures['height_cm'].isna()]
keep_measures['height_cm'].isna().sum()
```

Out[98]: 0

In [99]:
```
keep_measures['BMI'].isna().sum()
```

Out[99]: 0

In [100]:
```
keep_measures['waist_circumference_cm'].isna().sum()
```

Out[100]: 419

In [101]:
```
keep_measures['hip_circumference_cm'].isna().sum()
```

Out[101]: 1975

Waist and hip circumference will be filled during the imputing step to come.

Physical Activity

[View source for additional description and details (https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/PAQ_J.htm)](https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/PAQ_J.htm)

Adult participants were surveyed on their general physical activity levels.

In [102]:

```
activity
```

Out[102]:

	PAQ605	PAQ610	PAD615	PAQ620	PAQ625	PAD630	PAQ635	PAQ640	PAD64
SEQN									
93705	2.0	NaN	NaN	2.0	NaN	NaN	2.0	NaN	NaI
93706	2.0	NaN	NaN	2.0	NaN	NaN	1.0	5.0	45.(
93708	2.0	NaN	NaN	2.0	NaN	NaN	2.0	NaN	NaI
93709	2.0	NaN	NaN	1.0	2.0	180.0	2.0	NaN	NaI
93711	2.0	NaN	NaN	2.0	NaN	NaN	1.0	5.0	60.(
...	...	...	...	...	...	...	...	...	..
102950	2.0	NaN	NaN	2.0	NaN	NaN	2.0	NaN	NaI
102952	2.0	NaN	NaN	2.0	NaN	NaN	2.0	NaN	NaI
102953	1.0	3.0	240.0	1.0	3.0	240.0	2.0	NaN	NaI
102954	2.0	NaN	NaN	2.0	NaN	NaN	2.0	NaN	NaI
102956	2.0	NaN	NaN	2.0	NaN	NaN	2.0	NaN	NaI

5856 rows × 16 columns

In [103]:

```
activity.columns
```

Out[103]:

```
Index(['PAQ605', 'PAQ610', 'PAD615', 'PAQ620', 'PAQ625', 'PAD630', 'PA(
       'PAQ640', 'PAD645', 'PAQ650', 'PAQ655', 'PAD660', 'PAQ665', 'PA(
       'PAD675', 'PAD680'],
      dtype='object')
```

In [104]:

```
keepcols_activity = ['PAD615', 'PAQ610', 'PAD630', 'PAQ625', 'PAQ640',
                     'PAQ655', 'PAD675', 'PAQ670', 'PAD680']
mapper_activity = {'PAD615':'work_vigorous_minperday',
                   'PAQ610': 'work_vigorous_daysperweek',
                   'PAD630': 'work_moderate_minperday',
                   'PAQ625': 'work_moderate_daysperweek',
                   'PAD645': 'transportation_minperday',
                   'PAQ640': 'transportation_daysperweek',
                   'PAD660': 'recreation_vigorous_minperday',
                   'PAQ655': 'recreation_vigorous_daysperweek',
                   'PAD675': 'recreation_moderate_minperday',
                   'PAQ670': 'recreation_moderate_daysperweek',
                   'PAD680': 'sedentary_minsperday'}
```

In [105]:
```
keep_activity = activity.copy()[keepcols_activity].rename(mapper=mappe
keep_activity
```

Out[105]:

	work_vigorous_minperday	work_vigorous_daysperweek	work_moderate_minperday
SEQN			
93705	0.0	0.0	0.0
93706	0.0	0.0	0.0
93708	0.0	0.0	0.0
93709	0.0	0.0	180.0
93711	0.0	0.0	0.0
...	...	...	...
102950	0.0	0.0	0.0
102952	0.0	0.0	0.0
102953	240.0	3.0	240.0
102954	0.0	0.0	0.0
102956	0.0	0.0	0.0

5856 rows × 11 columns

In [106]:
```
days_cols = ['work_vigorous_daysperweek', 'work_moderate_daysperweek',
             'recreation_vigorous_daysperweek', 'recreation_moderate_day
mins_cols = ['work_vigorous_minperday', 'work_moderate_minperday', 'tra
             'recreation_vigorous_minperday', 'recreation_moderate_minpe
```

All the columns in units=days and all the columns in units=minutes have the same dummy values, so they are analyzed together. These are especially important to fix now in order to engineer a smaller number of features which summarize each respondent's vigorous, moderate, and sedentary activity levels.

In [107]:
```
for col in days_cols:
    print(col)
    display(keep_activity[col].value_counts())
    print('*****'*15)
```

```
work_vigorous_daysperweek

0.0     4467
5.0      384
3.0      232
2.0      178
6.0      160
4.0      153
7.0      151
1.0      130
99.0       1
Name: work_vigorous_daysperweek, dtype: int64

***************************************************************************
work_moderate_daysperweek

0.0     3417
5.0      702
3.0      401
7.0      331
2.0      308
4.0      307
6.0      210
1.0      174
99.0       6
Name: work_moderate_daysperweek, dtype: int64

***************************************************************************
transportation_daysperweek

0.0     4417
7.0      410
5.0      332
3.0      230
2.0      177
4.0      135
6.0       87
1.0       64
99.0       4
Name: transportation_daysperweek, dtype: int64

***************************************************************************
recreation_vigorous_daysperweek

0.0     4422
3.0      385
2.0      269
4.0      264
5.0      207
1.0      163
7.0       73
6.0       72
```

```
99.0       1
Name: recreation_vigorous_daysperweek, dtype: int64

************************************************************************
recreation_moderate_daysperweek

0.0     3548
3.0      606
2.0      491
5.0      321
4.0      293
1.0      253
7.0      250
6.0       91
99.0       3
Name: recreation_moderate_daysperweek, dtype: int64

************************************************************************
```

In [108]:
```
for col in mins_cols:
    print(col)
    display(keep_activity[col].value_counts())
    print('*****'*15)
```

```
work_vigorous_minperday

0.0       4475
120.0      234
60.0       194
240.0      177
180.0      130
480.0      107
30.0       106
300.0       93
360.0       80
420.0       41
10.0        40
600.0       38
15.0        31
20.0        31
90.0        23
540.0       15
45.0        14
9999.0       6
150.0        4
40.0         4
720.0        4
25.0         2
80.0         1
12.0         1
840.0        1
35.0         1
780.0        1
140.0        1
660.0        1
Name: work_vigorous_minperday, dtype: int64

***************************************************************************
work_moderate_minperday

0.0       3430
120.0      449
60.0       362
240.0      317
180.0      277
30.0       206
300.0      150
480.0      131
360.0      125
10.0        76
20.0        75
15.0        55
90.0        39
420.0       38
600.0       34
45.0        28
```

```
540.0       14
9999.0      13
40.0        11
720.0        6
150.0        6
25.0         4
50.0         2
660.0        1
12.0         1
230.0        1
70.0         1
35.0         1
840.0        1
140.0        1
210.0        1
Name: work_moderate_minperday, dtype: int64

**********************************************************************
transportation_minperday

0.0       4426
30.0       333
60.0       270
20.0       176
10.0       154
15.0       119
120.0      118
40.0        45
180.0       38
45.0        36
240.0       30
90.0        28
25.0        21
300.0        8
480.0        8
360.0        6
420.0        6
50.0         5
80.0         4
9999.0       4
12.0         3
35.0         3
14.0         2
540.0        2
75.0         1
16.0         1
22.0         1
660.0        1
150.0        1
42.0         1
17.0         1
209.0        1
110.0        1
600.0        1
230.0        1
Name: transportation_minperday, dtype: int64

**********************************************************************
```

```
recreation_vigorous_minperday

0.0       4425
60.0       464
120.0      240
30.0       210
45.0       110
90.0       110
180.0       90
20.0        62
40.0        29
240.0       23
15.0        18
10.0        13
25.0        11
35.0        10
150.0       10
50.0         5
75.0         5
80.0         4
300.0        4
18.0         2
480.0        2
360.0        2
23.0         1
12.0         1
11.0         1
209.0        1
420.0        1
55.0         1
9999.0       1
Name: recreation_vigorous_minperday, dtype: int64

*******************************************************************
recreation_moderate_minperday

0.0       3555
60.0       641
30.0       557
120.0      260
20.0       194
45.0       123
15.0       102
90.0        80
180.0       77
10.0        73
40.0        59
240.0       54
300.0       16
25.0        16
35.0         8
50.0         7
360.0        6
80.0         4
75.0         4
480.0        4
12.0         3
22.0         2
```

```
420.0       2
150.0       2
540.0       1
55.0        1
16.0        1
28.0        1
100.0       1
210.0       1
9999.0      1
Name: recreation_moderate_minperday, dtype: int64

**************************************************************************
```

In [109]:

```
for col in days_cols:
    keep_activity[col].replace({77.0: 0, 99.0:0}, inplace=True)
for col in mins_cols:
    keep_activity[col].replace({7777.0: 0, 9999.0: 0}, inplace=True)
```

In [110]:
```
for col in days_cols:
    print(col)
    display(keep_activity[col].value_counts())
    print('_____'*20)
```

```
work_vigorous_daysperweek

0.0    4468
5.0     384
3.0     232
2.0     178
6.0     160
4.0     153
7.0     151
1.0     130
Name: work_vigorous_daysperweek, dtype: int64

____________________________________________________________________________________________________
work_moderate_daysperweek

0.0    3423
5.0     702
3.0     401
7.0     331
2.0     308
4.0     307
6.0     210
1.0     174
Name: work_moderate_daysperweek, dtype: int64

____________________________________________________________________________________________________
transportation_daysperweek

0.0    4421
7.0     410
5.0     332
3.0     230
2.0     177
4.0     135
6.0      87
1.0      64
Name: transportation_daysperweek, dtype: int64

____________________________________________________________________________________________________
recreation_vigorous_daysperweek

0.0    4423
3.0     385
2.0     269
4.0     264
5.0     207
1.0     163
7.0      73
6.0      72
Name: recreation_vigorous_daysperweek, dtype: int64

____________________________________________________________________________________________________
recreation_moderate_daysperweek
```

```
0.0    3551
3.0     606
2.0     491
5.0     321
4.0     293
1.0     253
7.0     250
6.0      91
Name: recreation_moderate_daysperweek, dtype: int64
```

In [111]:
```
for col in mins_cols:
    print(col)
    display(keep_activity[col].value_counts())
    print('_____'*20)
```

```
work_vigorous_minperday

0.0      4481
120.0     234
60.0      194
240.0     177
180.0     130
480.0     107
30.0      106
300.0      93
360.0      80
420.0      41
10.0       40
600.0      38
20.0       31
15.0       31
90.0       23
540.0      15
45.0       14
150.0       4
40.0        4
720.0       4
25.0        2
80.0        1
12.0        1
840.0       1
35.0        1
780.0       1
140.0       1
660.0       1
Name: work_vigorous_minperday, dtype: int64
____________________________________________________________________________________________________
work_moderate_minperday

0.0      3443
120.0     449
60.0      362
240.0     317
180.0     277
30.0      206
300.0     150
480.0     131
360.0     125
10.0       76
20.0       75
15.0       55
90.0       39
420.0      38
600.0      34
45.0       28
540.0      14
```

```
40.0       11
150.0       6
720.0       6
25.0        4
50.0        2
660.0       1
12.0        1
230.0       1
70.0        1
35.0        1
840.0       1
140.0       1
210.0       1
Name: work_moderate_minperday, dtype: int64
```

```
transportation_minperday

0.0      4430
30.0      333
60.0      270
20.0      176
10.0      154
15.0      119
120.0     118
40.0       45
180.0      38
45.0       36
240.0      30
90.0       28
25.0       21
300.0       8
480.0       8
360.0       6
420.0       6
50.0        5
80.0        4
35.0        3
12.0        3
540.0       2
14.0        2
75.0        1
16.0        1
22.0        1
660.0       1
150.0       1
42.0        1
17.0        1
209.0       1
110.0       1
600.0       1
230.0       1
Name: transportation_minperday, dtype: int64
```

```
recreation_vigorous_minperday

0.0      4426
```

```
60.0      464
120.0     240
30.0      210
45.0      110
90.0      110
180.0      90
20.0       62
40.0       29
240.0      23
15.0       18
10.0       13
25.0       11
150.0      10
35.0       10
50.0        5
75.0        5
80.0        4
300.0       4
18.0        2
480.0       2
360.0       2
209.0       1
12.0        1
420.0       1
23.0        1
55.0        1
11.0        1
Name: recreation_vigorous_minperday, dtype: int64
```

```
recreation_moderate_minperday

0.0      3556
60.0      641
30.0      557
120.0     260
20.0      194
45.0      123
15.0      102
90.0       80
180.0      77
10.0       73
40.0       59
240.0      54
300.0      16
25.0       16
35.0        8
50.0        7
360.0       6
80.0        4
75.0        4
480.0       4
12.0        3
420.0       2
150.0       2
22.0        2
540.0       1
55.0        1
```

```
16.0        1
28.0        1
100.0       1
210.0       1
Name: recreation_moderate_minperday, dtype: int64
```

Three features were engineered to summarize activity levels by multiplying days per week times minutes per day of each activity type, then adding work and recreation activity together.

In [112]:

```
keep_activity['vigorous_activity_minsperweek'] =\
            (keep_activity['work_vigorous_daysperweek']*keep_activity[
            (keep_activity['recreation_vigorous_daysperweek']*keep_act:

keep_activity['moderate_activity_minsperweek'] =\
            (keep_activity['work_moderate_daysperweek']*keep_activity[
            (keep_activity['recreation_moderate_daysperweek']*keep_act:
            (keep_activity['transportation_daysperweek']*keep_activity[

keep_activity
```

Out[112]:

	work_vigorous_minperday	work_vigorous_daysperweek	work_moderate_minperday
SEQN			
93705	0.0	0.0	0.0
93706	0.0	0.0	0.0
93708	0.0	0.0	0.0
93709	0.0	0.0	180.0
93711	0.0	0.0	0.0
...	...	...	...
102950	0.0	0.0	0.0
102952	0.0	0.0	0.0
102953	240.0	3.0	240.0
102954	0.0	0.0	0.0
102956	0.0	0.0	0.0

5856 rows × 13 columns

In [113]:

```
keep_activity2 = keep_activity[['sedentary_minsperday',
                                'vigorous_activity_minsperweek',
                                'moderate_activity_minsperweek']]
keep_activity2
```

Out[113]:

	sedentary_minsperday	vigorous_activity_minsperweek	moderate_activity_minsperw
SEQN			
93705	300.0	0.0	1
93706	240.0	0.0	2
93708	120.0	0.0	1
93709	600.0	0.0	3
93711	420.0	16.0	3
...	...	...	
102950	60.0	0.0	
102952	120.0	0.0	3
102953	360.0	720.0	7
102954	600.0	0.0	
102956	720.0	0.0	

5856 rows × 3 columns

In [114]:

```
keep_activity2.describe().round(2)
```

Out[114]:

	sedentary_minsperday	vigorous_activity_minsperweek	moderate_activity_minsperwee
count	5856.00	5856.00	5856.0
mean	388.89	222.20	491.
std	771.32	603.94	777.0
min	0.00	0.00	0.0
25%	180.00	0.00	0.0
50%	300.00	0.00	150.0
75%	480.00	25.00	600.0
max	9999.00	5056.00	5880.0

Blood Pressure

[View source for additional description and details (https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/BPXO_J.htm)](https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/BPXO_J.htm)

Pulse is captured in the blood pressure data file. Because this model seeks to predict diabetes based only on easily-accessible questionnaire data or body measurements, pulse is the only measurement used from this file. For the purposes of this analysis, an average of three oscillometric measurements was calculated.

In [115]:
```
bp
```

Out[115]:

SEQN	BPAOARM	BPAOCSZ	BPAOMNTS	BPXOSY1	BPXODI1	BPXOSY2	BPXODI2	BPXO
93705	b'R'	4.0	-20.0	164.0	66.0	165.0	66.0	1
93706	b'R'	3.0	138.0	126.0	74.0	128.0	68.0	1
93707	b'R'	2.0	12.0	136.0	71.0	133.0	72.0	1
93708	b'R'	3.0	22.0	146.0	82.0	142.0	76.0	1
93709	b'R'	4.0	58.0	120.0	83.0	124.0	81.0	1
...	...	...	...	...	...	...	...	
102952	b'R'	3.0	97.0	154.0	92.0	144.0	84.0	1
102953	b'R'	4.0	-57.0	135.0	91.0	133.0	86.0	1
102954	b''	3.0	-101.0	123.0	75.0	119.0	71.0	1
102955	b'R'	5.0	-88.0	92.0	64.0	97.0	64.0	
102956	b'R'	4.0	30.0	143.0	100.0	146.0	101.0	1

7132 rows × 12 columns

In [116]:
```
bp.columns
```

Out[116]:
```
Index(['BPAOARM', 'BPAOCSZ', 'BPAOMNTS', 'BPXOSY1', 'BPXODI1', 'BPXOSY:
       'BPXODI2', 'BPXOSY3', 'BPXODI3', 'BPXOPLS1', 'BPXOPLS2', 'BPXOP]
      dtype='object')
```

In [117]:
```
keepcols_bp = ['BPXOPLS1', 'BPXOPLS2', 'BPXOPLS3']
mapper_bp = {'BPXOPLS1': 'pulse_1', 'BPXOPLS2': 'pulse_2', 'BPXOPLS3':
```

In [118]:

```
pulse = bp[keepcols_bp]
pulse.rename(mapper=mapper_bp, axis=1, inplace=True)
pulse
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
  return super().rename(
```

Out[118]:

	pulse_1	pulse_2	pulse_3
SEQN			
93705	52.0	51.0	49.0
93706	76.0	83.0	73.0
93707	100.0	89.0	91.0
93708	67.0	65.0	71.0
93709	64.0	62.0	61.0
...	...	...	...
102952	88.0	84.0	74.0
102953	76.0	79.0	78.0
102954	NaN	NaN	NaN
102955	71.0	71.0	76.0
102956	72.0	74.0	75.0

7132 rows × 3 columns

In [119]:
```
pulse['avg_pulse'] = ((pulse['pulse_1'] + pulse['pulse_2'] + pulse['pul
pulse = pd.DataFrame(pulse['avg_pulse'])
pulse
```

```
<ipython-input-119-5c63e71f4110>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
  pulse['avg_pulse'] = ((pulse['pulse_1'] + pulse['pulse_2'] + pulse[']
```

Out[119]:

	avg_pulse
SEQN	
93705	50.7
93706	77.3
93707	93.3
93708	67.7
93709	62.3
...	...
102952	82.0
102953	77.7
102954	NaN
102955	72.7
102956	73.7

7132 rows × 1 columns

Nicotine Usage

[View source for additional description and details (https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/SMQ_J.htm#Component_Description)](https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/SMQ_J.htm#Component_Description)

The Smoking - Cigarette Use dataset provides a history of cigarette use, age at initiation, past 30-day use, cigarette brand, sub-brand and other related details. Questions on ever use of cigars, smokeless tobacco, and electronic nicotine delivery systems (including e-cigarettes) were added to NHANES in 2015-16.

Based on the source data available and domain knowledge, four nicotine usage features were defined as:

- **Lifetime cigarette smoker** - respondent has smoked at least 100 cigarettes in their lifetime

- **Current cigarette smoker** - respondent currently smokes every day or some days
- **E-cigarette smoker** - respondent has smoked an ecigarette at least once in the last 30 days
- **Smokeless tobacco user** - respondent has used smokeless tobacco at least once in the last 30 days

Because there are no obvious assumptions with which to fill null values, they will be addressed at the imputing step.

In [120]:

```
smoking.head()
```

Out[120]:

	SMQ020	SMD030	SMQ040	SMQ050Q	SMQ050U	SMD057	SMQ078	SMD641	SM
SEQN									
93705	1.0	16.0	3.0	30.0	4.0	5.0	NaN	NaN	
93706	2.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
93707	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
93708	2.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
93709	1.0	15.0	1.0	NaN	NaN	NaN	1.0	30.0	

In [121]:

```
keepcols_smoking = ['SMQ020', 'SMQ040', 'SMQ905', 'SMQ915']
mapper_smoking = {'SMQ020': 'lifetime_cigarette_smoker',
                  'SMQ040': 'current_cigarette_smoker',
                  'SMQ905': 'ecig_smoker',
                  'SMQ915': 'smokeless_tobacco_user'}
```

In [122]:
```
keep_smoking = smoking[keepcols_smoking].rename(mapper=mapper_smoking,
keep_smoking.round(1)
```

Out[122]:

SEQN	lifetime_cigarette_smoker	current_cigarette_smoker	ecig_smoker	smokeless_toba
93705	1.0	3.0	NaN	
93706	2.0	NaN	NaN	
93707	NaN	NaN	NaN	
93708	2.0	NaN	NaN	
93709	1.0	1.0	0.0	
...	...	...	...	
102952	2.0	NaN	NaN	
102953	1.0	3.0	0.0	
102954	2.0	NaN	NaN	
102955	NaN	NaN	NaN	
102956	1.0	1.0	NaN	

6724 rows × 4 columns

In [123]:
```
keep_smoking['lifetime_cigarette_smoker'].value_counts(dropna=False)
```

Out[123]:
```
2.0    3497
1.0    2359
NaN     868
Name: lifetime_cigarette_smoker, dtype: int64
```

In [124]:
```
keep_smoking['lifetime_cigarette_smoker'].replace({1.0: 'yes', 2.0: 'nc
```

In [125]:
```
keep_smoking['current_cigarette_smoker'].value_counts(dropna=False)
```

Out[125]:
```
NaN    4365
3.0    1338
1.0     805
2.0     216
Name: current_cigarette_smoker, dtype: int64
```

In [126]:
```
keep_smoking['current_cigarette_smoker'].replace({1.0: 'yes',
                                                   2.0: 'yes',
                                                   3.0: 'no'}, inplace=True)
```

In [127]:
```
keep_smoking['current_cigarette_smoker'].value_counts(dropna=False)
```

Out[127]:
```
NaN    4365
no     1338
yes    1021
Name: current_cigarette_smoker, dtype: int64
```

In [128]:
```
keep_smoking['ecig_smoker'].value_counts(dropna=False)
```

Out[128]:
```
NaN             5574
5.397605e-79     844
1.000000e+00     102
3.000000e+01      55
2.000000e+00      43
3.000000e+00      21
5.000000e+00      20
1.500000e+01      19
1.000000e+01      11
4.000000e+00      10
2.000000e+01       7
2.100000e+01       3
7.000000e+00       3
9.000000e+00       3
1.200000e+01       2
6.000000e+00       2
2.500000e+01       2
2.800000e+01       1
1.400000e+01       1
9.900000e+01       1
Name: ecig_smoker, dtype: int64
```

In [129]:
```
for ind, row in keep_smoking.iterrows():
    if keep_smoking['ecig_smoker'][ind] > 0:
        keep_smoking['ecig_smoker'][ind] = 'yes'
```

```
<ipython-input-129-0391f3294e35>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
  keep_smoking['ecig_smoker'][ind] = 'yes'
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
  iloc._setitem_with_indexer(indexer, value)
```

In [130]:
```
keep_smoking['smokeless_tobacco_user'].value_counts(dropna=False)
```

Out[130]:
```
NaN             5863
5.397605e-79     720
3.000000e+01      71
1.000000e+00      22
2.000000e+00      17
4.000000e+00       6
5.000000e+00       4
3.000000e+00       4
2.000000e+01       4
1.000000e+01       4
1.500000e+01       3
7.000000e+00       2
2.500000e+01       1
1.200000e+01       1
9.000000e+00       1
2.800000e+01       1
Name: smokeless_tobacco_user, dtype: int64
```

In [131]:
```
for ind, row in keep_smoking.iterrows():
    if keep_smoking['smokeless_tobacco_user'][ind] > 0:
        keep_smoking['smokeless_tobacco_user'][ind] = 'yes'
```

```
<ipython-input-131-d865f51e87fd>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
  keep_smoking['smokeless_tobacco_user'][ind] = 'yes'
```

A1C

View source for additional description and details (https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/GHB_J.htm)

Per the Mayo Clinic (https://www.mayoclinic.org/diseases-conditions/diabetes/diagnosis-treatment/drc-20371451#:~:text=A%20fasting%20blood%20sugar%20level,separate%20tests%2C%20you%2 the Glycated Hemoglobin (A1C) test indicates a person's average blood sugar level for the past two to three months. It measures the percentage of blood sugar attached to hemoglobin, the oxygen-carrying protein in red blood cells. The higher your blood sugar levels, the more hemoglobin you'll have with sugar attached.

- A1C level of 6.5% or higher on two separate tests indicates that you have diabetes
- A1C between 5.7 and 6.4 % indicates prediabetes
- A1C below 5.7 is considered normal

NHANES examined participants age 12+ provided a blood sample for analysis.

In [132]:
```
a1c
```

Out[132]:

	LBXGH
SEQN	
93705	6.2
93706	5.2
93707	5.6
93708	6.2
93709	6.3
...	...
102952	7.4
102953	5.9
102954	5.2
102955	5.5
102956	5.4

6401 rows × 1 columns

In [133]:
```
a1c.rename(mapper={'LBXGH': 'glycohemoglobin'}, axis=1, inplace=True)
```

In [134]:
```
a1c.isna().sum()
```

Out[134]:
```
glycohemoglobin    356
dtype: int64
```

Because this is our target in a supervised learning task, all nulls were removed.

In [135]:
```
a1c.dropna(inplace=True)
```

One approach is to create binary classes for normal vs elevated A1C levels. The multi-variate approach would require the model to predict one of three classes: normal, prediabetic, or diabetic.

For the initial model, we'll work with binary classes, though a multiclass model may also be of interest in the future.

Special care is taken when encoding the diagnosis values to help the model's interpretability - 0=normal, 1=diabetic/prediabetic

In [136]:
```
def diagnose_multiclass(a1c_value):
    diagnosis = None
    if a1c_value >= 6.5:
        diagnosis = 'Diabetic'
    if (a1c_value >= 5.7) & (a1c_value < 6.5):
        diagnosis = 'Prediabetic'
    if a1c_value < 5.7:
        diagnosis = 'Normal'
    return diagnosis

def diagnose_binary(a1c_value):
    '''
    Returns binary values; 0 indicates normal, 1 indicates diabetic/pre
    '''
    diagnosis = None
    if (a1c_value >= 5.7):
        diagnosis = 1
    if a1c_value < 5.7:
        diagnosis = 0
    return diagnosis
```

In [137]:
```
a1c['diagnosis'] = a1c.glycohemoglobin.map(lambda x: diagnose_binary(x)
a1c
```

Out[137]:

	glycohemoglobin	diagnosis
SEQN		
93705	6.2	1
93706	5.2	0
93707	5.6	0
93708	6.2	1
93709	6.3	1
...	...	...
102952	7.4	1
102953	5.9	1
102954	5.2	0
102955	5.5	0
102956	5.4	0

6045 rows × 2 columns

In [138]:
```
display(a1c.diagnosis.value_counts())
display(a1c.diagnosis.value_counts(normalize=True).round(2))
```

```
0    3659
1    2386
Name: diagnosis, dtype: int64
```

```
0    0.61
1    0.39
Name: diagnosis, dtype: float64
```

Merging and Preprocessing Data

Once all data files are loaded and initial processing complete, some additional steps are needed before the sample can be modeled.

The preprocessing workflow:

1. **Merge data files into one dataframe,** ensuring all samples in the final set have valid/unimputed A1C / diagnosis values. Body measures were determined to be the second-most critical data to the model since prior research shows that weight/BMI is a key risk factor for diabetes. Appropriate steps are taken at the merging step to ensure the final combined dataset has as many natural/unimputed body measures as possible without too much data leakage.
2. **Encode categorical features** with number values (0=False, 1=True) so model algorithms can interpret the data.
3. Split the dataset into **training and testing samples.** Models will be fit to the training sample, then validated against the testing sample.

    - It is important that this step occurs before any processing that uses the sample data as a whole to define other values (ex. some feature engineering, null value imputation, normalization, etc.), since the testing data should imitate real-world new/novel data as closely as possible to evaluate how good the model actually is.
4. **Fill remaining null values** using a model that guesses the missing value based on the rest of the dataset.
5. **50/50 balance the number of diabetic and nondiabetic respondents represented in the training data**, so the model doesn't accidentally perform well just based on the laws of probability.
6. **Standard-scale** the continuous data in order to compare feature importances.

Merge

In [139]:

```
# Concatenated all but two most important dfs - a1c (target) and measu
concat = pd.concat([keep_demographic, keep_insurance, keep_activity2, p
                    join='outer', axis=1)
concat
```

Out[139]:

SEQN	gender	age	race	veteran_status	country_of_birth	citizen_status	
93703	NaN	NaN	NaN	NaN	NaN	NaN	
93704	NaN	NaN	NaN	NaN	NaN	NaN	
93705	female	66.0	black	no	usa	citizen	
93706	male	18.0	asian	no	usa	citizen	
93707	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	
102952	female	70.0	asian	no	other	citizen	hig
102953	male	42.0	mexican_american	no	other	non_citizen	hig
102954	female	41.0	black	no	usa	citizen	
102955	NaN	NaN	NaN	NaN	NaN	NaN	
102956	male	38.0	white	no	usa	citizen	

9254 rows × 31 columns

In [140]:

```
# inner merge on 2 most important dfs in order to only keep IDs where l
inner_merge = pd.merge(a1c, keep_measures, how='inner', on='SEQN')
display(inner_merge.head())
inner_merge.index.value_counts().sum()
```

SEQN	glycohemoglobin	diagnosis	weight_kg	height_cm	BMI	waist_circumference_cm
93705	6.2	1	79.5	158.3	31.7	101.8
93706	5.2	0	66.3	175.7	21.5	79.3
93707	5.6	0	45.4	158.4	18.1	64.1
93708	6.2	1	53.5	150.2	23.7	88.2
93709	6.3	1	88.8	151.1	38.9	113.0

Out[140]: 5951

In [141]:

```
# left merge the concatenated df onto the merged a1c and body measures
mega_df = pd.merge(inner_merge, concat, how='left', on='SEQN')

# drop the lab results
mega_df.drop(columns='glycohemoglobin', inplace=True)
```

In [142]:
```
display(mega_df.head())
display(mega_df.info())
```

	diagnosis	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_
SEQN						
93705	1	79.5	158.3	31.7	101.8	1
93706	0	66.3	175.7	21.5	79.3	(
93707	0	45.4	158.4	18.1	64.1	(
93708	1	53.5	150.2	23.7	88.2	(
93709	1	88.8	151.1	38.9	113.0	1:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5951 entries, 93705 to 102956
Data columns (total 37 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   diagnosis                     5951 non-null   int64  
 1   weight_kg                     5951 non-null   float64
 2   height_cm                     5951 non-null   float64
 3   BMI                           5951 non-null   float64
 4   waist_circumference_cm        5738 non-null   float64
 5   hip_circumference_cm          5751 non-null   float64
 6   gender                        5128 non-null   object 
 7   age                           5128 non-null   float64
 8   race                          5128 non-null   object 
 9   veteran_status                5128 non-null   object 
 10  country_of_birth              5128 non-null   object 
 11  citizen_status                5128 non-null   object 
 12  education                     4884 non-null   object 
 13  marital_status                5128 non-null   object 
 14  household_size                5128 non-null   float64
 15  annual_household_income       4655 non-null   float64
 16  income_poverty_ratio          4466 non-null   float64
 17  coverage_status               5950 non-null   object 
 18  covered_private               5950 non-null   object 
 19  covered_medicare              5950 non-null   object 
 20  covered_medigap               5950 non-null   object 
 21  covered_medicaid              5950 non-null   object 
 22  covered_chip                  5950 non-null   object 
 23  covered_military              5950 non-null   object 
 24  covered_state                 5950 non-null   object 
 25  covered_other_gov             5950 non-null   object 
 26  covered_single_service        5950 non-null   object 
 27  prescription_coverage         5950 non-null   object 
 28  uninsured_in_last_year        5950 non-null   object 
 29  sedentary_minsperday          5175 non-null   float64
 30  vigorous_activity_minsperweek 5175 non-null   float64
 31  moderate_activity_minsperweek 5175 non-null   float64
 32  avg_pulse                     4500 non-null   float64
 33  lifetime_cigarette_smoker     5175 non-null   object 
 34  current_cigarette_smoker      2100 non-null   object 
```

```
 35  ecig_smoker                    1027 non-null   object
 36  smokeless_tobacco_user         762 non-null    object
dtypes: float64(13), int64(1), object(23)
memory usage: 1.7+ MB

None
```

One-Hot Encode

In [143]:
```
mega_df_encoded = pd.get_dummies(mega_df, drop_first=True, dtype='float
mega_df_encoded.head()
```

Out[143]:

	diagnosis	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_
SEQN						
93705	1	79.5	158.3	31.7	101.8	1
93706	0	66.3	175.7	21.5	79.3	[illegible]
93707	0	45.4	158.4	18.1	64.1	[illegible]
93708	1	53.5	150.2	23.7	88.2	[illegible]
93709	1	88.8	151.1	38.9	113.0	1:

Train-Test Split

In [144]:
```
from sklearn.model_selection import train_test_split
```

In [145]:
```
rs = 610
y = mega_df_encoded['diagnosis']
X = mega_df_encoded.drop(columns='diagnosis')
X_train , X_test, y_train, y_test = train_test_split(X, y, test_size=0.
```

In [146]:
```
print(f'X Train Shape: {X_train.shape}\nX Test Shape: {X_test.shape}')

display(X_train.head())
display(y_train.head())
```

```
X Train Shape: (4760, 47)
X Test Shape: (1191, 47)
```

	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_cm	age
SEQN						
102783	64.8	156.8	26.4	96.8	93.5	52.0
99996	84.6	164.2	31.4	105.0	113.5	51.0
99721	74.8	174.0	24.7	87.5	105.8	20.0
97169	87.9	163.1	33.0	111.8	119.2	67.0
94465	106.2	185.4	30.9	104.7	109.4	31.0

```
SEQN
102783    1
99996     1
99721     0
97169     1
94465     0
Name: diagnosis, dtype: int64
```

Impute Remaining Null Values

Scikit-learn's experimental IterativeImputer class models each feature with null values as a function of other features to intelligently fill missing values. This is an experimental class, but performed better than the KNNImputer. This class may change; please review the documentation here. (https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html)

In [147]:
```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
```

In [148]:
```
X_train.isna().sum()
```

Out[148]:
```
weight_kg                                       0
height_cm                                       0
BMI                                             0
waist_circumference_cm                        180
hip_circumference_cm                          170
age                                           660
household_size                                660
annual_household_income                      1039
income_poverty_ratio                         1185
sedentary_minsperday                          622
vigorous_activity_minsperweek                 622
moderate_activity_minsperweek                 622
avg_pulse                                    1153
gender_male                                     0
race_black                                      0
race_hispanic                                   0
race_mexican_american                           0
race_other                                      0
race_white                                      0
veteran_status_yes                              0
country_of_birth_usa                            0
citizen_status_no_answer                        0
citizen_status_non_citizen                      0
education_highschool_grad                       0
education_no_diploma                            0
education_some_college                          0
marital_status_married or living with partner   0
marital_status_never married                    0
marital_status_no_answer                        0
marital_status_widowed                          0
coverage_status_no_answer                       0
coverage_status_uninsured                       0
covered_private_yes                             0
covered_medicare_yes                            0
covered_medigap_yes                             0
covered_medicaid_yes                            0
covered_chip_yes                                0
covered_military_yes                            0
covered_state_yes                               0
covered_other_gov_yes                           0
covered_single_service_yes                      0
prescription_coverage_no_answer                 0
prescription_coverage_yes                       0
uninsured_in_last_year_no_answer                0
uninsured_in_last_year_yes                      0
lifetime_cigarette_smoker_yes                   0
current_cigarette_smoker_yes                    0
dtype: int64
```

In [149]:
```
# fit_transform the training data, but just transform the testing data
imp = IterativeImputer(random_state=rs)
X_train_imputed = pd.DataFrame(imp.fit_transform(X_train, y_train),
                               columns=X_train.columns, index=X_train.i

X_test_imputed = pd.DataFrame(imp.transform(X_test), columns=X_test.col

display(X_train_imputed.head())
display(y_train.head())
```

	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_cm	age
SEQN						
102783	64.8	156.8	26.4	96.8	93.5	52.0
99996	84.6	164.2	31.4	105.0	113.5	51.0
99721	74.8	174.0	24.7	87.5	105.8	20.0
97169	87.9	163.1	33.0	111.8	119.2	67.0
94465	106.2	185.4	30.9	104.7	109.4	31.0

```
SEQN
102783    1
99996     1
99721     0
97169     1
94465     0
Name: diagnosis, dtype: int64
```

In [150]:
```
X_train_imputed.isna().sum()
```

Out[150]:
```
weight_kg                                        0
height_cm                                        0
BMI                                              0
waist_circumference_cm                           0
hip_circumference_cm                             0
age                                              0
household_size                                   0
annual_household_income                          0
income_poverty_ratio                             0
sedentary_minsperday                             0
vigorous_activity_minsperweek                    0
moderate_activity_minsperweek                    0
avg_pulse                                        0
gender_male                                      0
race_black                                       0
race_hispanic                                    0
race_mexican_american                            0
race_other                                       0
race_white                                       0
veteran_status_yes                               0
country_of_birth_usa                             0
citizen_status_no_answer                         0
citizen_status_non_citizen                       0
education_highschool_grad                        0
education_no_diploma                             0
education_some_college                           0
marital_status_married or living with partner    0
marital_status_never married                     0
marital_status_no_answer                         0
marital_status_widowed                           0
coverage_status_no_answer                        0
coverage_status_uninsured                        0
covered_private_yes                              0
covered_medicare_yes                             0
covered_medigap_yes                              0
covered_medicaid_yes                             0
covered_chip_yes                                 0
covered_military_yes                             0
covered_state_yes                                0
covered_other_gov_yes                            0
covered_single_service_yes                       0
prescription_coverage_no_answer                  0
prescription_coverage_yes                        0
uninsured_in_last_year_no_answer                 0
uninsured_in_last_year_yes                       0
lifetime_cigarette_smoker_yes                    0
current_cigarette_smoker_yes                     0
dtype: int64
```

In [151]:
```
X_test_imputed.head()
```

Out[151]:

	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_cm	age
SEQN						
98959	72.4	164.6	26.7	100.2	97.8	75.0
102761	71.6	165.3	26.2	80.4	97.4	25.0
100216	52.8	158.0	21.2	74.0	89.1	42.0
99151	69.2	162.2	26.3	83.1	104.3	22.0
99679	90.1	172.9	30.1	102.5	101.8	61.0

In [152]:
```
X_test_imputed.isna().sum()
```

Out[152]:
```
weight_kg                                     0
height_cm                                     0
BMI                                           0
waist_circumference_cm                        0
hip_circumference_cm                          0
age                                           0
household_size                                0
annual_household_income                       0
income_poverty_ratio                          0
sedentary_minsperday                          0
vigorous_activity_minsperweek                 0
moderate_activity_minsperweek                 0
avg_pulse                                     0
gender_male                                   0
race_black                                    0
race_hispanic                                 0
race_mexican_american                         0
race_other                                    0
race_white                                    0
veteran_status_yes                            0
country_of_birth_usa                          0
citizen_status_no_answer                      0
citizen_status_non_citizen                    0
education_highschool_grad                     0
education_no_diploma                          0
education_some_college                        0
marital_status_married or living with partner 0
marital_status_never married                  0
marital_status_no_answer                      0
marital_status_widowed                        0
coverage_status_no_answer                     0
coverage_status_uninsured                     0
covered_private_yes                           0
covered_medicare_yes                          0
covered_medigap_yes                           0
covered_medicaid_yes                          0
covered_chip_yes                              0
covered_military_yes                          0
covered_state_yes                             0
covered_other_gov_yes                         0
covered_single_service_yes                    0
prescription_coverage_no_answer               0
prescription_coverage_yes                     0
uninsured_in_last_year_no_answer              0
uninsured_in_last_year_yes                    0
lifetime_cigarette_smoker_yes                 0
current_cigarette_smoker_yes                  0
dtype: int64
```

Class Balance

In [153]:
```
# checking for class imbalance in the target feature
print(y_train.value_counts())
print('*****'*15)
print(y_train.value_counts(normalize=True).round(3))
```

```
0    2888
1    1872
Name: diagnosis, dtype: int64
***************************************************************************
0    0.607
1    0.393
Name: diagnosis, dtype: float64
```

In [154]:
```
from imblearn.over_sampling import SMOTE
```

In [155]:
```
smote = SMOTE(n_jobs=3)
X_train_bal, y_train_bal = smote.fit_sample(X_train_imputed, y_train)
```

In [156]:
```
display(X_train_bal.head())
display(y_train_bal.head())
```

	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_cm	age	hous
0	64.8	156.8	26.4	96.8	93.5	52.0	
1	84.6	164.2	31.4	105.0	113.5	51.0	
2	74.8	174.0	24.7	87.5	105.8	20.0	
3	87.9	163.1	33.0	111.8	119.2	67.0	
4	106.2	185.4	30.9	104.7	109.4	31.0	

```
0    1
1    1
2    0
3    1
4    0
Name: diagnosis, dtype: int64
```

In [157]:
```
print(y_train_bal.value_counts())
print('*****'*15)
print(y_train_bal.value_counts(normalize=True).round(3))
```

```
1    2888
0    2888
Name: diagnosis, dtype: int64
***************************************************************************
1    0.5
0    0.5
Name: diagnosis, dtype: float64
```

In [158]:
```
# reset variables to simplify model process on unscaled data
X_train = X_train_bal.copy()
y_train = pd.DataFrame(y_train_bal.copy())
X_test = X_test_imputed.copy()
# y_test hasn't changed
```

In [159]:
```
display(X_train.head())
display(y_train.head())
```

	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_cm	age	hous
0	64.8	156.8	26.4	96.8	93.5	52.0	
1	84.6	164.2	31.4	105.0	113.5	51.0	
2	74.8	174.0	24.7	87.5	105.8	20.0	
3	87.9	163.1	33.0	111.8	119.2	67.0	
4	106.2	185.4	30.9	104.7	109.4	31.0	

	diagnosis
0	1
1	1
2	0
3	1
4	0

In [160]:
```
display(X_test.head())
display(y_test.head())
```

	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_cm	age
SEQN						
98959	72.4	164.6	26.7	100.2	97.8	75.0
102761	71.6	165.3	26.2	80.4	97.4	25.0
100216	52.8	158.0	21.2	74.0	89.1	42.0
99151	69.2	162.2	26.3	83.1	104.3	22.0
99679	90.1	172.9	30.1	102.5	101.8	61.0

```
SEQN
98959     1
102761    0
100216    0
99151     0
99679     0
Name: diagnosis, dtype: int64
```

Normalize Data

In [161]:
```
from sklearn.preprocessing import StandardScaler
```

In [162]:
```
scaler = StandardScaler()
# fit transform on train data
X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train), columns=X_

# just transform on test data
X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns=X_test.c
```

Classification Models

There are many modeling algorithms available, each with their strengths and weaknesses. Each model will be fit to the training sample, then validated on the testing sample. Key metrics will be captured along the way in a summary dataframe to compare and contrast performance. A custom function has been written to quickly evaluate each model.

Recall is the primary metric for evaluating this particular model as this represents, *out of all the true diabetic/prediabetic people, how often did the model predict correctly.*

- The downside of missing an at-risk person is that maybe they won't get the preventative care or health information they need, and they end up developing diabetes or progressing from prediabetic to diabetic.
- For stakeholders, this means their program did not reach the population it would most help, which translates into both public health and monetary loss in the long term (i.e., more claims to insurance companies, increased hospital utilization, etc.)
- For the individual, this relates to missed opportunity to participate in a program that may have built health literacy and provided them the support they needed to reverse prediabetes or manage diabets. This can translate to lower quality of life, greater personal health expenses, and even loss of life.

Accuracy is a more general metric of the model's holistic performance, representing the *percentage of predictions the model gets correct.* While this metric will be looked at, it will not be the deciding evaluation metric.

Precision is a metric to pay less attention to, as this represents *how often the model guessed someone is diabetic/prediabetic and the person truly is diabetic/prediabetic.*

- The downside of getting a diabetic prediction wrong is that someone not at risk for diabetes/prediabetes may be targeted for a program.
- For the stakeholder, this just means they might spend attitional resources targeting someone who doesn't truly NEED their programs. Total monetary impact could be calculated by multiplying the per-person spend on advertising/outreach by the number of "false negatives" the model predicts.
- For the person, there is no downside to having access to health and wellness programs!

F1 Score is a metric that combines precision and recall, also called *the harmonic mean of precision and recall.*

In [163]:
```
from sklearn.metrics import classification_report, plot_confusion_matr
from sklearn.metrics import accuracy_score, roc_curve, auc, f1_score, r
```

In [164]:
```
model_stats = pd.DataFrame(columns=['Model', 'Label', 'Train Recall', '
                                    'Train AUC', 'Test AUC',
                                    'Train Accuracy', 'Test Accuracy']]
```

In [165]:
```
def check_fit(model, X_train, y_train, X_test, y_test):
    '''
    Fits model to training data, then looks for overfitting by compari
    accuracy and F1 score of the training sample to the same metrics o

    Inputs:
    model (unfit)
    X_train - pandas dataframe comprised of training predictors
    y_train - pandas dataframe comprised of training target
    X_test - pandas dataframe comprised of testing predictors
    y_test - pandas dataframe comprised of testing target

    Returns:
    Dataframe with columns for evaluation metrics, rows for the traini
    '''
    model.fit(X_train, y_train)
    stats = pd.DataFrame(index=['Training', 'Testing'], columns=['Accu
    y_train_pred = model.predict(X_train)
    stats['Accuracy']['Training'] = accuracy_score(y_train, y_train_pre
    stats['F1 Score']['Training'] = f1_score(y_train, y_train_pred)
    y_test_pred = model.predict(X_test)
    stats['Accuracy']['Testing'] = accuracy_score(y_test, y_test_pred)
    stats['F1 Score']['Testing'] = f1_score(y_test, y_test_pred)
    return stats

def plot_roc(fpr_test, tpr_test):
    '''
    Plots the receiver operating curve for a binary classification mode

    Inputs:
    fpr_test - false positive rate associated with a model
    tpr_test - true positive rate associated with a model

    Returns:
    matplotlib plot showing a model's receiver operating curve (ROC) c
    '''
    test_auc = auc(fpr_test, tpr_test)
    print('AUC: {}'.format(test_auc))
    plt.figure(figsize=(6,4))
    plt.plot(fpr_test, tpr_test, color='darkorange', label='ROC curve')
    plt.plot([0,1], [0,1], color='navy', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show()

def evaluate_classifier(model, X_train, y_train, X_test, y_test,
                        cmap='Blues', model_stats=None, track=True, lab
    '''
    Overall evaluation of a model's performance using multiple metrics
    Can be used as part of an iterative modeling process to capture me
    and compare easily.

    Inputs:
    model - unfit model object with any parameters already passed
```

```
    X_train - pandas dataframe comprised of training predictors
    y_train - pandas dataframe comprised of training target
    X_test - pandas dataframe comprised of testing predictors
    y_test - pandas dataframe comprised of testing target
    *
    cmap='Blues' - color scheme for resulting confusion matrix *must be
        recognized by matplotlib https://matplotlib.org/stable/tutorial
    model_stats=None - dataframe in which to capture summary metrics. I
        create a new dataframe.
    track=True - boolean, whether to write and return the summary data
        summary metrics will be appended to the dataframe passed in mod
        If False, summary metrics will not be captured or returned.
    label='' = optional additional label to include in model_stats summ
        differentiate this model from others in your iterative process

    Returns:
    Model fit to training data and evaluation metrics including accurac
    confusion matrix, receiver operating curve.
    If track=True, model_stats dataframe is returned.
    '''
    model.fit(X_train, y_train)

    print('CHECK FOR OVERFITTING - seek for test data to perform almost
    display(check_fit(model, X_train, y_train, X_test, y_test))
    print('*****'*15)

    y_test_pred = model.predict(X_test)
    test_acc = accuracy_score(y_test, y_test_pred)
    y_train_pred = model.predict(X_train)
    train_acc = accuracy_score(y_train, y_train_pred)

    print('\nCHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to ma
    print(classification_report(y_test, y_test_pred))
    plot_confusion_matrix(model, X_test, y_test, normalize='all', cmap=
    print('*****'*15)

    print('\nCHECK Test ROC CURVE - seek to maximize area under the cur
    y_score_train = model.predict_proba(X_train)
    fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_score
    train_auc = auc(fpr_train, tpr_train)

    y_score_test = model.predict_proba(X_test)
    fpr_test, tpr_test, thresholds_test = roc_curve(y_test, y_score_tes
    test_auc = auc(fpr_test, tpr_test)

    plot_roc(fpr_test, tpr_test)

    tn, fp, fn, tp = confusion_matrix(y_test, y_test_pred).ravel()
    fnr = fn/(fn+tp)

    if model_stats is None:
        model_stats = pd.DataFrame(columns=['Model', 'Label', 'Train Re
                                            'False Negatives',
                                            'Train AUC', 'Test AUC',
                                            'Train Accuracy', 'Test Acc

    stats_dict = {}
```

```
    stats_dict['Model'] = str(model)
    stats_dict['Label'] = label
    try:
        stats_dict['Train AUC'] = train_auc.round(4)
    except:
        stats_dict['Train AUC'] = 'could not compute'
    try:
        stats_dict['Test AUC'] = test_auc.round(4)
    except:
        stats_dict['Test AUC'] = 'could not compute'
    try:
        stats_dict['Train Accuracy'] = train_acc.round(4)
    except:
        stats_dict['Train Accuracy'] = 'could not compute'
    try:
        stats_dict['Test Accuracy'] = test_acc.round(4)
    except:
        stats_dict['Test Accuracy'] = 'could not compute'
    try:
        stats_dict['Train Recall'] = recall_score(y_train, y_train_pre
    except:
        stats_dict['Train Recall'] = 'could not compute'
    try:
        stats_dict['Test Recall'] = recall_score(y_test, y_test_pred).r
    except:
        stats_dict['Test Recall'] = 'could not compute'
    try:
        stats_dict['False Negatives'] = fn
    except:
        stats_dict['False Negatives'] = 'could not compute'

    if track:
        model_stats = model_stats.append(pd.Series(stats_dict), ignore_
        return model_stats
```

Dummy Model - *for comparison only*

The dummy model uses simple rules (most_frequent, stratified, etc.) to make predictions. This is used as a baseline only to compare more intelligent models against.

In [166]:

```
from sklearn.dummy import DummyClassifier
```

In [167]:

```
dummy_clf = DummyClassifier(strategy='stratified')
```

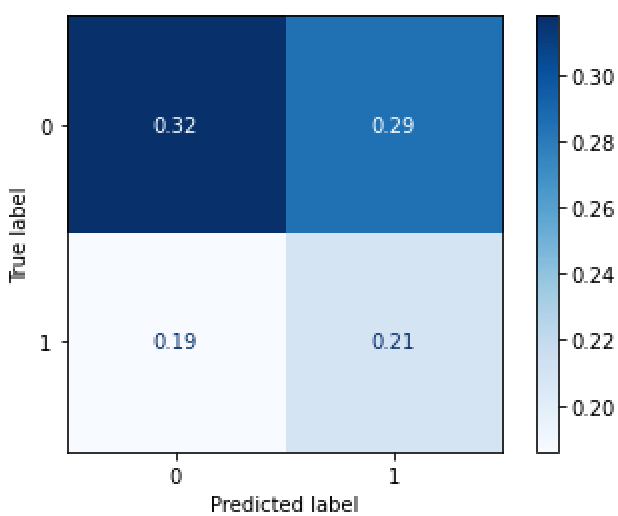
Scaled data

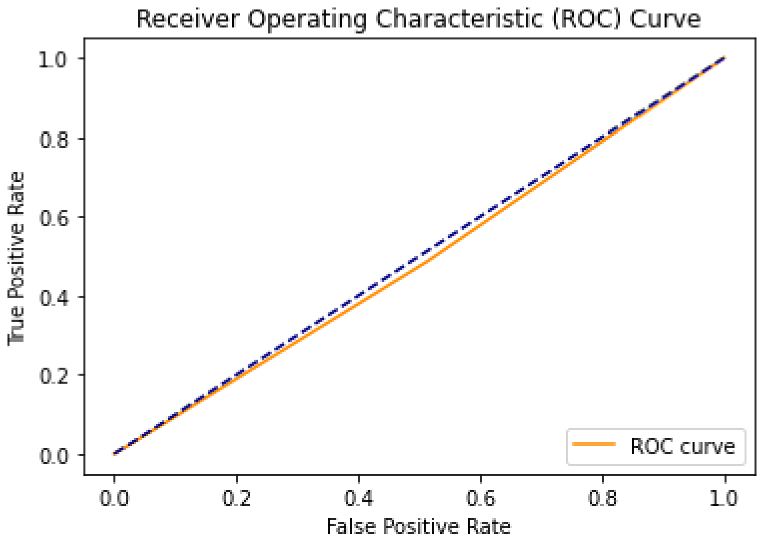
In [168]:
```
evaluate_classifier(dummy_clf, X_train_scaled, y_train, X_test_scaled,
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well a:
```

	Accuracy	F1 Score
Training	0.500173	0.50164
Testing	0.497901	0.430476

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.62      0.53      0.57       719
           1       0.41      0.51      0.46       472

    accuracy                           0.52      1191
   macro avg       0.52      0.52      0.51      1191
weighted avg       0.54      0.52      0.52      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.48667375828009707
```





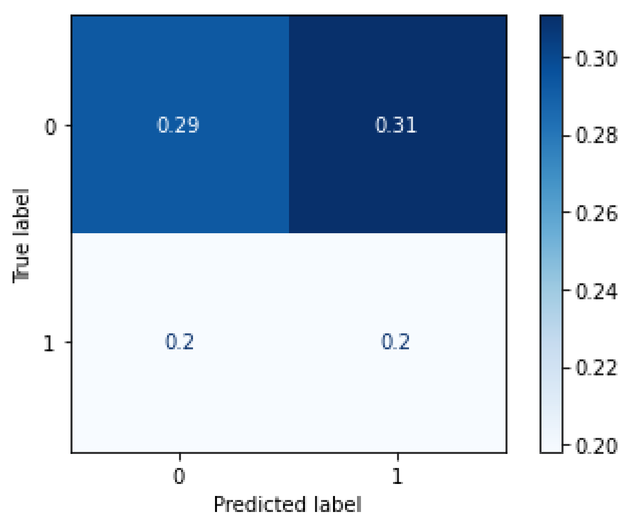
Unscaled data

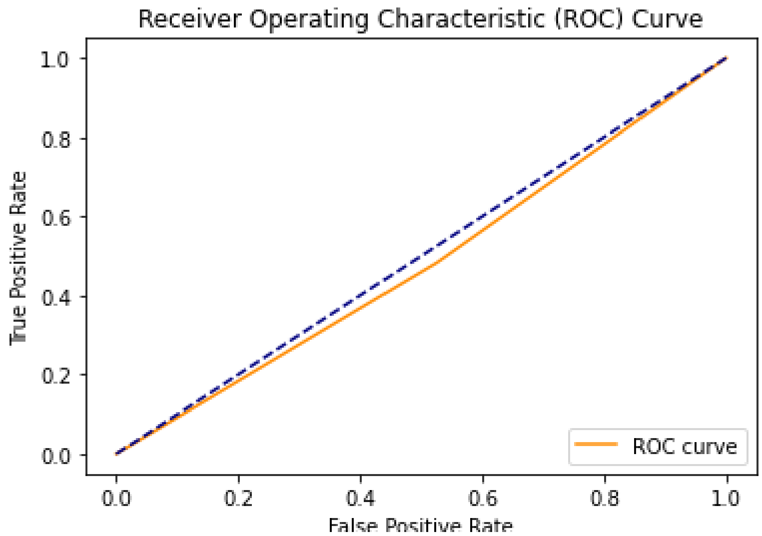
In [169]:
```
evaluate_classifier(dummy_clf, X_train, y_train, X_test, y_test, track=
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well a:
```

	Accuracy	F1 Score
Training	0.514889	0.514553
Testing	0.502939	0.460838

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal]

              precision    recall  f1-score   support

           0       0.59      0.50      0.54       719
           1       0.38      0.47      0.42       472

    accuracy                           0.48      1191
   macro avg       0.48      0.48      0.48      1191
weighted avg       0.50      0.48      0.49      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.47866033332547564
```





As expected, the dummy models using the stratification strategy both performed no better than random chance.

Logistic Regression

The logistic regression model guesses the probability that an element belongs to the normal class or the diabetic class based on the relationships learned between the target and predictors in the training sample.

In [170]:
```
from sklearn.linear_model import LogisticRegressionCV
```

In [171]:
```
log_clf = LogisticRegressionCV(random_state=rs, cv=3)
```

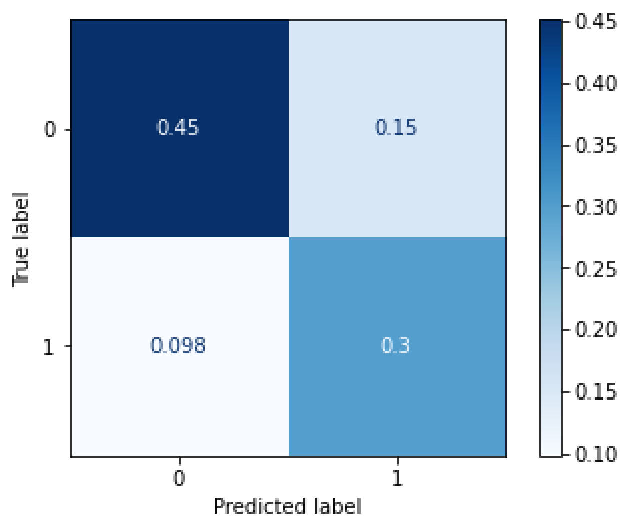
Scaled data

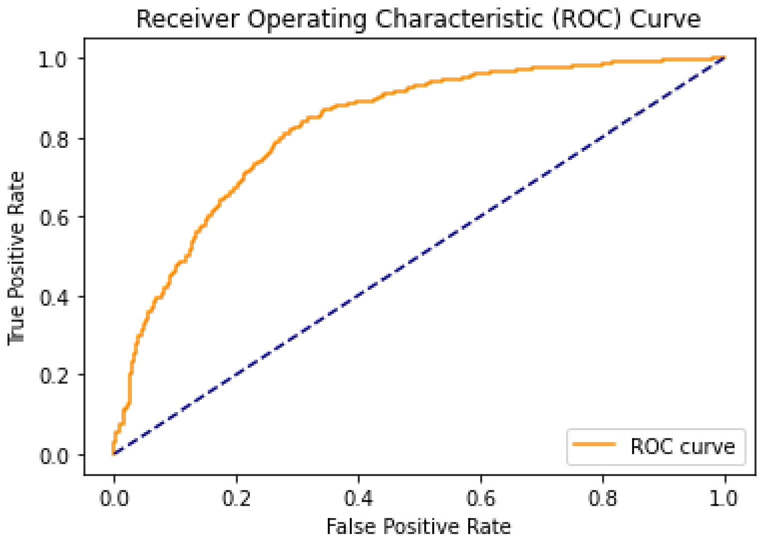
In [172]:
```
model_stats = evaluate_classifier(log_clf, X_train_scaled, y_train, X_t
                                  model_stats=model_stats, label='defau
model_stats
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)

CHECK FOR OVERFITTING - seek for test data to perform almost as well as

/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)
```

	Accuracy	F1 Score
Training	0.76108	0.767833
Testing	0.74895	0.703667

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.82      0.75      0.78       719
           1       0.66      0.75      0.70       472

    accuracy                           0.75      1191
   macro avg       0.74      0.75      0.74      1191
weighted avg       0.76      0.75      0.75      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8276973668701822
```





Out[172]:

	Model	Label	Train Recall	Test Recall	Train AUC	Test AUC	Train Accuracy	Test Accuracy	Nega
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254	0.8277	0.7611	0.749	

Unscaled data

In [173]:
```
model_stats = evaluate_classifier(log_clf, X_train, y_train, X_test, y_
                                  model_stats=model_stats, label='defau
model_stats
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as show
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as show
    https://scikit-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-
  n_iter_i = _check_optimize_result(
```

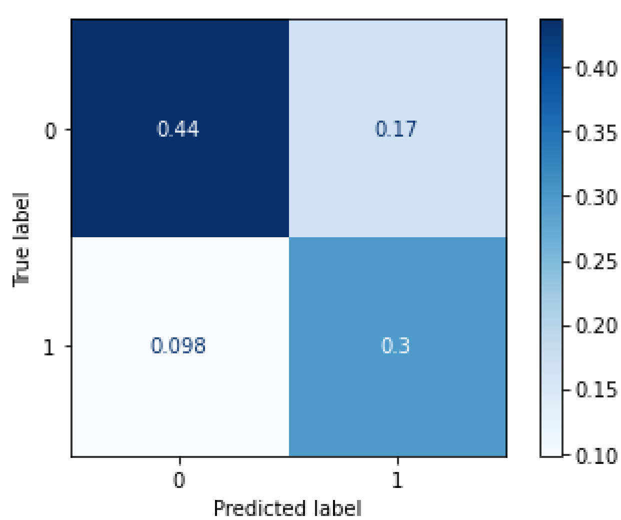
```
CHECK FOR OVERFITTING - seek for test data to perform almost as well as
```

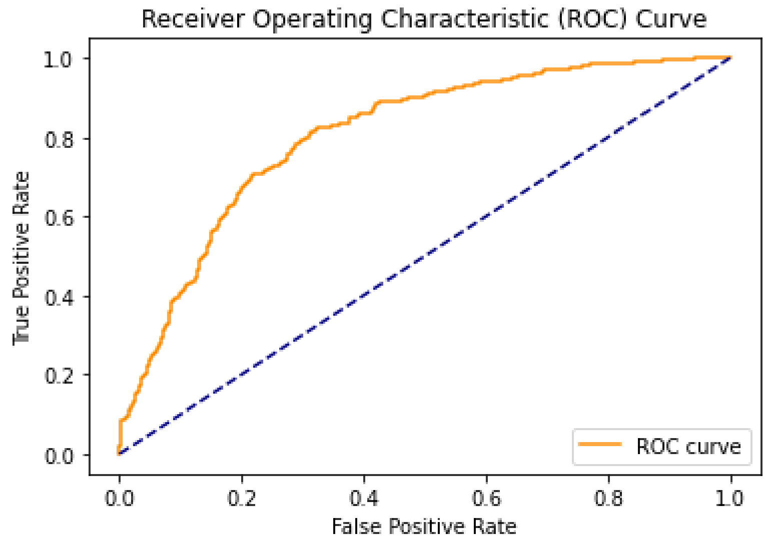
```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as show
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as show
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as show
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as show
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as show
    https://scikit-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-
  n_iter_i = _check_optimize_result(
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
```

	Accuracy	F1 Score
Training	0.746537	0.753867
Testing	0.735516	0.692683

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recall

              precision    recall  f1-score   support

           0       0.82      0.72      0.77       719
           1       0.64      0.75      0.69       472

    accuracy                           0.74      1191
   macro avg       0.73      0.74      0.73      1191
weighted avg       0.75      0.74      0.74      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8055827302515264
```





Out[173]:

	Model	Label	Train Recall	Test Recall	Train AUC	Test AUC	Train Accuracy	Test Accuracy	Neg
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254	0.8277	0.7611	0.7490	
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044	0.8056	0.7465	0.7355	

In [174]:

```
log_clf_coefs = pd.DataFrame(log_clf.coef_[0], index=X_train.columns).r
print('Positively related coefficients:')
display(log_clf_coefs[log_clf_coefs[0]>=0.01].sort_values(0, ascending=
print('***'*15)
print('\nNegatively related coefficients:')
display(log_clf_coefs[log_clf_coefs[0]<=-.01])
```

```
Positively related coefficients:
```

	0
age	0.063
weight_kg	0.033
waist_circumference_cm	0.030
avg_pulse	0.010

```
*********************************************

Negatively related coefficients:
```

	0
height_cm	-0.033
hip_circumference_cm	-0.040

The scaled dataset performed decently well with 76% training accuracy, 74.9% testing accuracy, and 0.82 AUC which is quite an improvement on the dummy model's 0.53.

72.6% of diabetic/prediabetic individuals were correctly predicted by the logistic regression model.

K-Nearest Neighbors

The K-Nearest Neighbors modeling technique predicts the class of an observation by evaluating data points mathematically close to the one in question. "k" number of neighbors are found, then the average of their classes determines the model's vote for the element in question.

In [175]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [176]:
```
# using defaults of 5 nearest neighbors, uniform weights, and euclidea
knn_clf = KNeighborsClassifier()
```

Scaled data

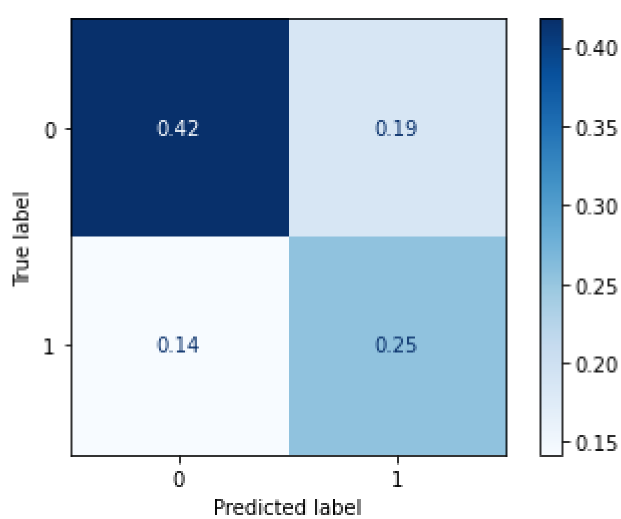
In [177]:

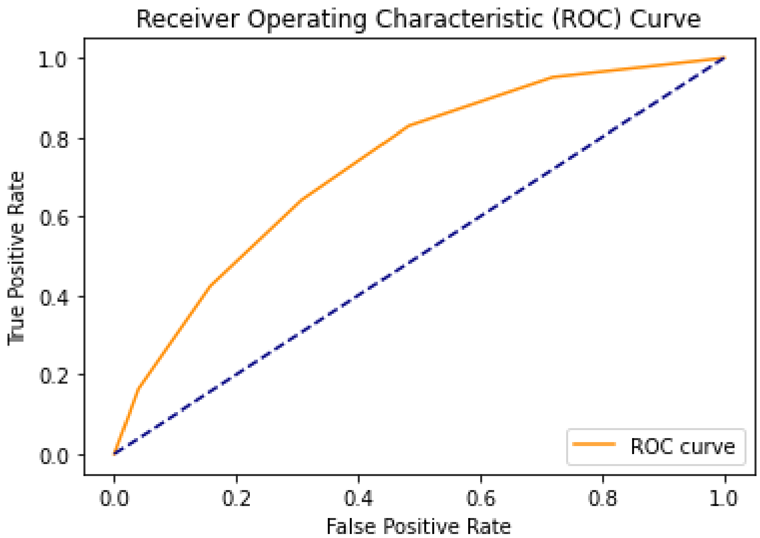
```
model_stats = evaluate_classifier(knn_clf, X_train_scaled, y_train, X_t
                                  model_stats=model_stats, label='defau
model_stats
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well as

<ipython-input-165-0759a685b866>:77: DataConversionWarning: A column-ve
  model.fit(X_train, y_train)
<ipython-input-165-0759a685b866>:16: DataConversionWarning: A column-ve
  model.fit(X_train, y_train)
```

	Accuracy	F1 Score
Training	0.822368	0.828943
Testing	0.672544	0.608434

```
************************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.75      0.69      0.72       719
           1       0.58      0.64      0.61       472

    accuracy                           0.67      1191
   macro avg       0.66      0.67      0.66      1191
weighted avg       0.68      0.67      0.67      1191

************************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.7312283420947172
```





Out[177]:

	Model	Label	Train Recall	Test Recall	Train AUC	Test AUC	Train Accuracy	Test Accuracy	Neg
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254	0.8277	0.7611	0.7490	
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044	0.8056	0.7465	0.7355	
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064	0.7312	0.8224	0.6725	

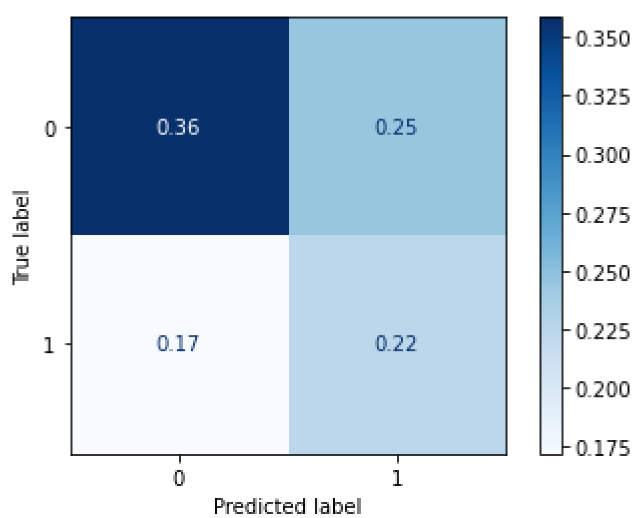
Unscaled data

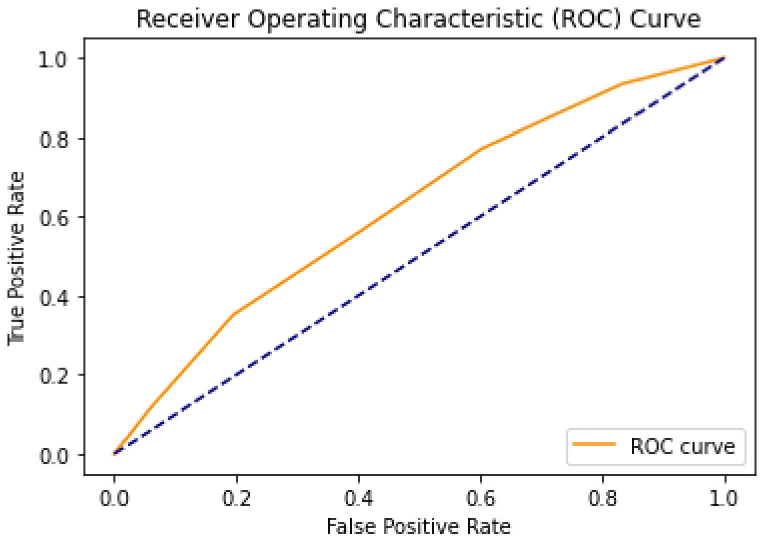
In [178]:
```
model_stats = evaluate_classifier(knn_clf, X_train, y_train, X_test, y_
                                  model_stats=model_stats, label='defau
model_stats
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well a

<ipython-input-165-0759a685b866>:77: DataConversionWarning: A column-ve
  model.fit(X_train, y_train)
<ipython-input-165-0759a685b866>:16: DataConversionWarning: A column-ve
  model.fit(X_train, y_train)
```

	Accuracy	F1 Score
Training	0.773546	0.787524
Testing	0.582704	0.517944

```
***********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.68      0.59      0.63       719
           1       0.48      0.57      0.52       472

    accuracy                           0.58      1191
   macro avg       0.58      0.58      0.58      1191
weighted avg       0.60      0.58      0.59      1191

***********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.6215332618278683
```





Out[178]:

	Model	Label	Train Recall	Test Recall	Train AUC	Test AUC	Train Accuracy	Test Accuracy	Neg
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254	0.8277	0.7611	0.7490	
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044	0.8056	0.7465	0.7355	
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064	0.7312	0.8224	0.6725	
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619	0.6215	0.7735	0.5827	

In [179]:

```
np.asarray(y_test)
```

Out[179]:

```
array([1, 0, 0, ..., 0, 0, 0])
```

In [180]:

```
# look for best n_neighbors param

best_k = 0
best_score = 0.0
for k in range(1, 25):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, np.asarray(y_train).reshape(len(y_train), 
    preds = knn.predict(X_test_scaled)
    recall = recall_score(y_test, preds)
    if recall > best_score:
        best_k = k
        best_score = recall
```

In [181]:
```
print(f'Best K: {best_k}')
print(f'Recall Score: {best_score}')
```

```
Best K: 21
Recall Score: 0.739406779661017
```

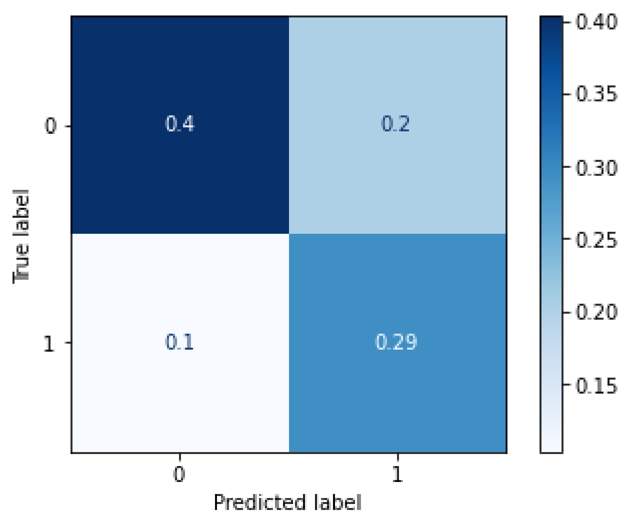
In [182]:
```
knn_clf2 = KNeighborsClassifier(n_neighbors=23)
```

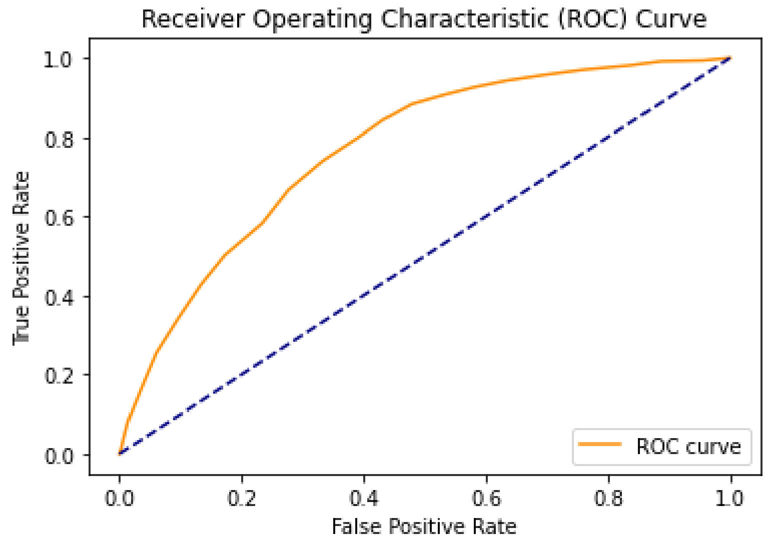
In [183]:
```
model_stats = evaluate_classifier(knn_clf2, X_train_scaled, y_train, X_
                                  model_stats=model_stats, label='k=23,
model_stats
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well a:

<ipython-input-165-0759a685b866>:77: DataConversionWarning: A column-v
  model.fit(X_train, y_train)
<ipython-input-165-0759a685b866>:16: DataConversionWarning: A column-v
  model.fit(X_train, y_train)
```

	Accuracy	F1 Score
Training	0.757791	0.773515
Testing	0.696054	0.658491

```
***********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.80      0.67      0.73       719
           1       0.59      0.74      0.66       472

    accuracy                           0.70      1191
   macro avg       0.69      0.70      0.69      1191
weighted avg       0.72      0.70      0.70      1191

***********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.7695805143678838
```





Out[183]:

	Model	Label	Train Recall	Test Recall	Train AUC	Test AUC	Train Accuracy	T Accu
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254	0.8277	0.7611	0.7
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044	0.8056	0.7465	0.7
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064	0.7312	0.8224	0.6
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619	0.6215	0.7735	0.5
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405	0.7696	0.7578	0.6

Decision Tree

The decision tree classification method selects the feature that best splits the sample space in two, creates two (or more) branches based on this feature, then continues branching and splitting until the dataset has been cleanly partitioned (or until max_depth or min_samples_split parameters, if set).

Using ensemble methods, a "random forest" of decision trees can be created which uses the laws of probability and central limit theorem to reach a consensus prediction based on the combination of multiple models trained on different sample splits.

Decision trees do not require scaled data, so these models are only run using the unscaled samples.

Single Tree Classifier

In [184]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

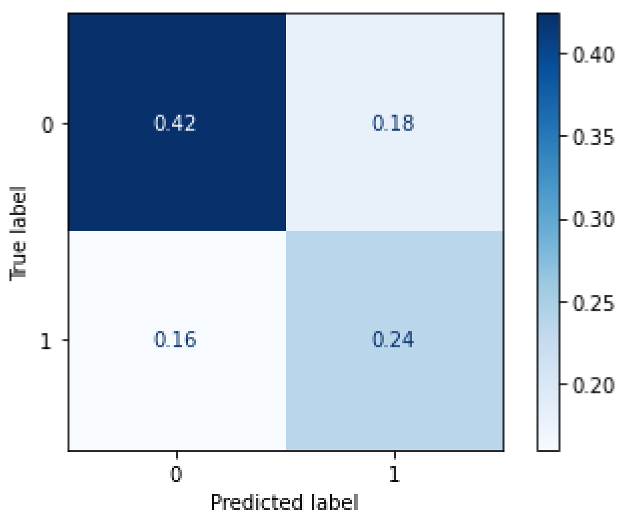
In [185]:
```
# first use default params of gini impurity, no max depth, 2 min sample
    # 1 min samples leaf, no max features, and no max leaf nodes
dt_clf = DecisionTreeClassifier(random_state=rs)
```

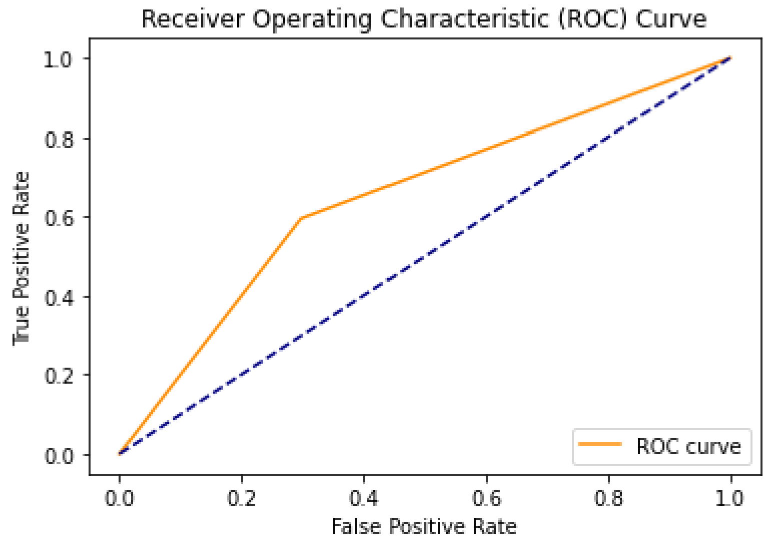
In [186]:
```
model_stats = evaluate_classifier(dt_clf, X_train, y_train, X_test, y_t
                                  model_stats=model_stats, label='defau
model_stats
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well as
```

	Accuracy	F1 Score
Training	1	1
Testing	0.65995	0.581179

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.73      0.70      0.71       719
           1       0.57      0.60      0.58       472

    accuracy                           0.66      1191
   macro avg       0.65      0.65      0.65      1191
weighted avg       0.66      0.66      0.66      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.6488516890219466
```





Out[186]:

	Model	Label	Train Recall	Test Recall	Train AUC	Test AUC	Train Accuracy	A
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254	0.8277	0.7611	
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044	0.8056	0.7465	
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064	0.7312	0.8224	
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619	0.6215	0.7735	
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405	0.7696	0.7578	
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000	0.6489	1.0000	

The decision tree was quite overfit, as decision trees with no max features or max depth parameters tend to be.

A grid search can help find the best combination of parameters to use.

In [187]:
```
dt_clf.get_params()
```

Out[187]:
```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'presort': 'deprecated',
 'random_state': 610,
 'splitter': 'best'}
```

In [188]:
```
dt_clf.get_depth()
```

Out[188]:
```
26
```

In [189]:
```
dt_clf.get_n_leaves()
```

Out[189]:
```
875
```

In [190]:
```
pd.DataFrame(dt_clf.feature_importances_, index=X_train.columns).sort_v
```

Out[190]:

	index	0
0	age	0.260515
1	waist_circumference_cm	0.098869
2	avg_pulse	0.060168
3	hip_circumference_cm	0.054435
4	weight_kg	0.050502
5	height_cm	0.046357
6	BMI	0.044330
7	income_poverty_ratio	0.042859
8	moderate_activity_minsperweek	0.036375
9	race_white	0.033420
10	sedentary_minsperday	0.030992
11	vigorous_activity_minsperweek	0.028127
12	household_size	0.026048
13	annual_household_income	0.021833
14	covered_medicare_yes	0.020437
15	race_black	0.018065
16	gender_male	0.015350
17	country_of_birth_usa	0.013352
18	citizen_status_non_citizen	0.010983
19	lifetime_cigarette_smoker_yes	0.008160
20	education_no_diploma	0.008020
21	covered_private_yes	0.007143
22	education_some_college	0.006409
23	covered_medicaid_yes	0.006075
24	marital_status_married or living with partner	0.006008
25	current_cigarette_smoker_yes	0.005481
26	education_highschool_grad	0.004723
27	marital_status_never married	0.004042
28	covered_single_service_yes	0.003782
29	prescription_coverage_no_answer	0.003774
30	veteran_status_yes	0.003459
31	covered_other_gov_yes	0.002670
32	covered_state_yes	0.002667
33	race_mexican_american	0.001857

	index	0
34	uninsured_in_last_year_yes	0.001690
35	covered_military_yes	0.001629
36	covered_medigap_yes	0.001583
37	race_hispanic	0.001516
38	marital_status_widowed	0.001508
39	uninsured_in_last_year_no_answer	0.001267
40	covered_chip_yes	0.000854
41	race_other	0.000646
42	coverage_status_no_answer	0.000626
43	prescription_coverage_yes	0.000549
44	marital_status_no_answer	0.000540
45	coverage_status_uninsured	0.000305
46	citizen_status_no_answer	0.000000

In [191]:

```
from sklearn.model_selection import GridSearchCV, cross_val_score
```

In [192]:

```
dt_param_grid = {'criterion': ['entropy', 'gini'],
                'max_depth': [None, 5, 10, 15, 20],
                'min_samples_leaf': [1, 2, 3, 4],
                'max_features': [None, 5, 10, 15, 20]}
```

In [193]:

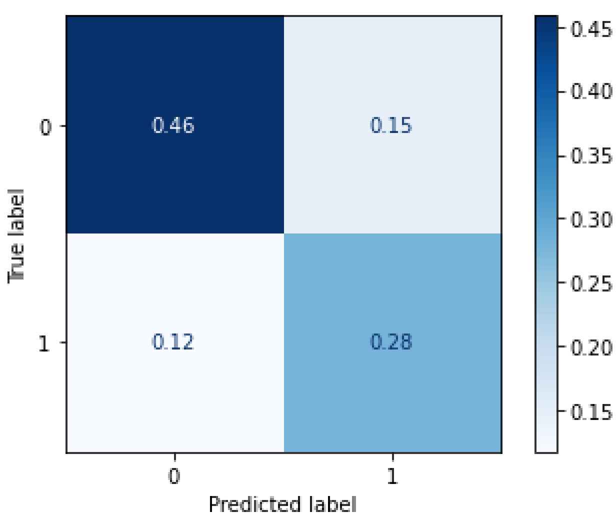
```
dt_grid_search = GridSearchCV(dt_clf, dt_param_grid, cv=3, return_trai
```

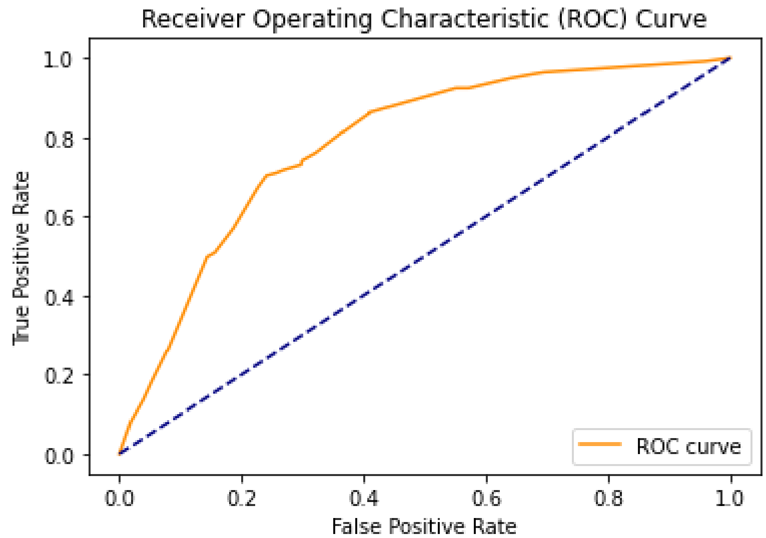
In [194]:
```
model_stats = evaluate_classifier(dt_grid_search, X_train, y_train, X_t
                                   model_stats=model_stats, label='unsca
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well as
```

	Accuracy	F1 Score
Training	0.776835	0.780894
Testing	0.737196	0.679632

```
************************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.80      0.76      0.78       719
           1       0.66      0.70      0.68       472

    accuracy                           0.74      1191
   macro avg       0.73      0.73      0.73      1191
weighted avg       0.74      0.74      0.74      1191

************************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.784237170269442
```





In [195]:
```
model_stats
```

Out[195]:

	Model	Label	Train Recall	Test Recall	Train AUC	Test AUC	Train Accuracy	A
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254	0.8277	0.7611	
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044	0.8056	0.7465	
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064	0.7312	0.8224	
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619	0.6215	0.7735	
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405	0.7696	0.7578	
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000	0.6489	1.0000	
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464	0.7842	0.7768	

In [196]:
```
dt_grid_search.best_params_
```

Out[196]:
```
{'criterion': 'gini',
 'max_depth': 5,
 'max_features': None,
 'min_samples_leaf': 4}
```

In [197]:
```
dt_clf2 = DecisionTreeClassifier(criterion='entropy', max_depth=5, max_
                                random_state=rs)
```

Bagged Trees

In [198]:
```
from sklearn.ensemble import BaggingClassifier
```

In [199]:
```
rf_clf = BaggingClassifier(base_estimator=dt_clf2, n_estimators=25, ran
```

In [200]:
```
model_stats = evaluate_classifier(rf_clf, X_train, y_train, X_test, y_t
                                  model_stats=model_stats,
                                   label='25 bagged trees with base est:
model_stats
```

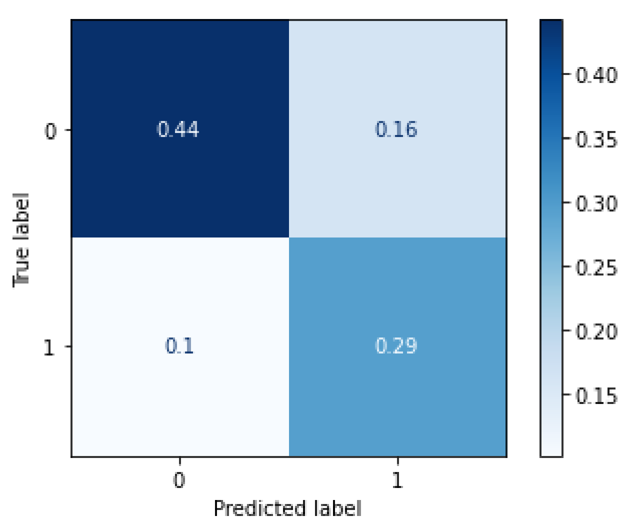
```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)
```

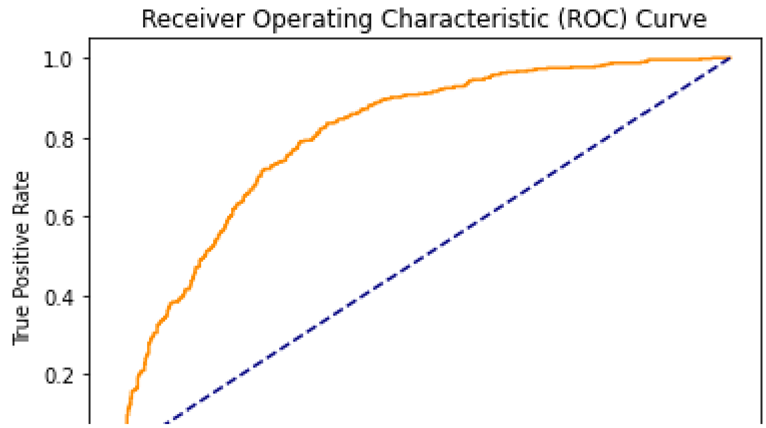
```
CHECK FOR OVERFITTING - seek for test data to perform almost as well a:
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)
```

	Accuracy	F1 Score
Training	0.784626	0.794516
Testing	0.735516	0.689655

```
***********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal]

              precision    recall  f1-score   support

           0       0.81      0.73      0.77       719
           1       0.64      0.74      0.69       472

    accuracy                           0.74      1191
   macro avg       0.73      0.74      0.73      1191
weighted avg       0.75      0.74      0.74      1191

***********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8077323141840127
```





Out[200]:

	Model	Label	Train Recall	Test Recall	Train AUC	
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254	0
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044	0
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064	0
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619	0
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405	0
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000	0
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464	0
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676	0

Random Forest

In [201]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [202]:

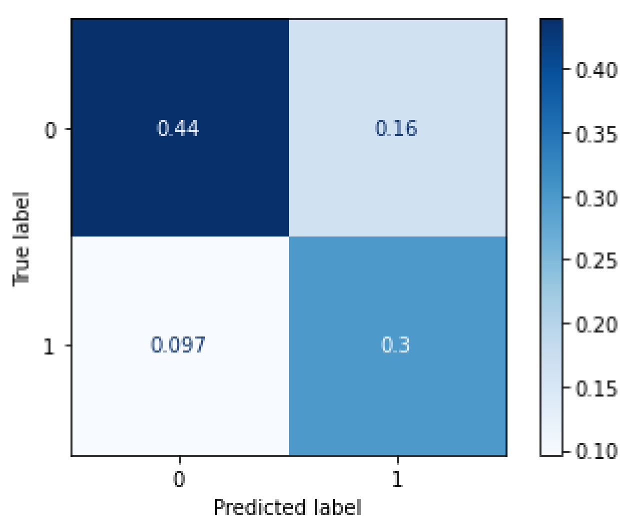
```
rf_clf2 = RandomForestClassifier(n_estimators=50, criterion='entropy',
                                 min_samples_leaf=4, random_state=rs)
```

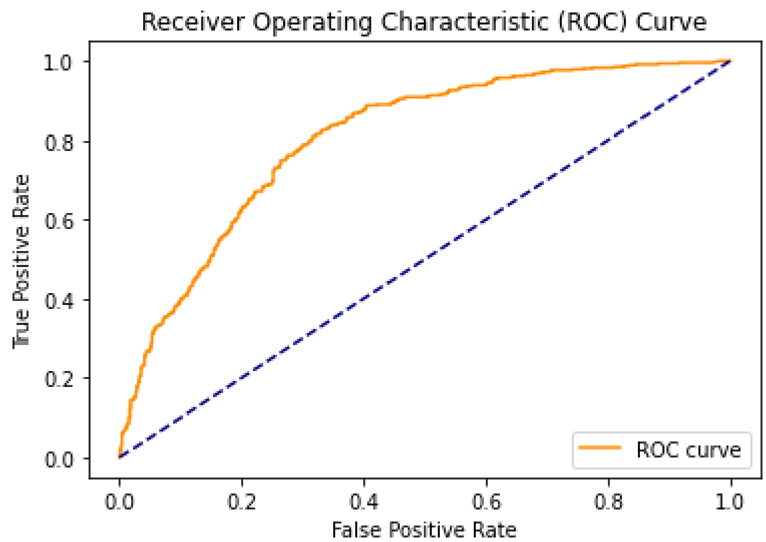
In [203]:
```
model_stats = evaluate_classifier(rf_clf2, X_train, y_train, X_test, y_
                                  model_stats=model_stats, label='RF wit
```

```
<ipython-input-165-0759a685b866>:77: DataConversionWarning: A column-ve
  model.fit(X_train, y_train)


CHECK FOR OVERFITTING - seek for test data to perform almost as well as

<ipython-input-165-0759a685b866>:16: DataConversionWarning: A column-ve
  model.fit(X_train, y_train)
```

	Accuracy	F1 Score
Training	0.782548	0.794098
Testing	0.738875	0.696585

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.82      0.73      0.77       719
           1       0.65      0.76      0.70       472

    accuracy                           0.74      1191
   macro avg       0.73      0.74      0.73      1191
weighted avg       0.75      0.74      0.74      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8053514179297989
```





This is the best model so far, with only a 9.3% false negative rate and .76 recall score on testing data. It doesn't seem too overfit since the testing data is doing comparable to the training data.

Boosting Ensemble Methods

Boosting ensemble methods are similar to random forests in that they leverage many individual models to come to one overall prediction. Where random forests are comprised on many decision trees which are individually viable models, boosting methods train many decision trees with a max_depth=1, meaning they are optimized on only one split. Each subsequent model focuses on optimizing what the last model got wrong by weighting incorrectly classified instances higher.

AdaBoost

In [204]:
```
from sklearn.ensemble import AdaBoostClassifier
```

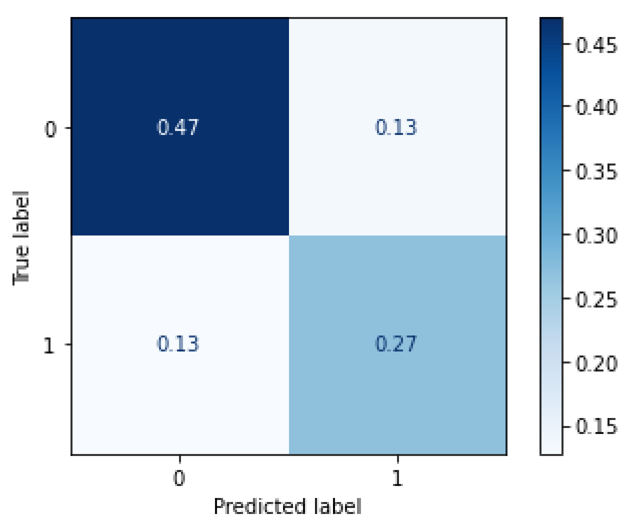
In [205]:
```
# using default params of n_estimators=50 and learning_rate=1
ab_clf = AdaBoostClassifier(random_state=rs)
```

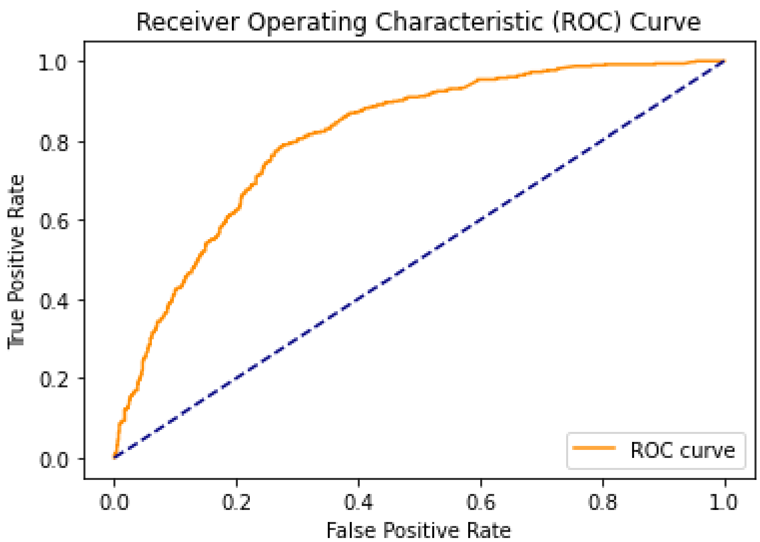
In [206]:
```
model_stats = evaluate_classifier(ab_clf, X_train, y_train, X_test, y_t
                                  model_stats=model_stats, label='defau
model_stats
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)

CHECK FOR OVERFITTING - seek for test data to perform almost as well a

/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)
```

	Accuracy	F1 Score
Training	0.784453	0.788157
Testing	0.738035	0.672269

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.79      0.78      0.78       719
           1       0.67      0.68      0.67       472

    accuracy                           0.74      1191
   macro avg       0.73      0.73      0.73      1191
weighted avg       0.74      0.74      0.74      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8089198156573396
```





Out[206]:

	Model	Label	Train Recall	Test Recall	Train AUC	
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254	0
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044	0
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064	0
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619	0
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405	0
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000	0
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464	0
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676	0
8	RandomForestClassifier(criterion='entropy', ma...	RF with gridsearch best params, 50 trees	0.8386	0.7564	0.8690	0
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.8019	0.6780	0.8744	0

GradientBoosting

In [207]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [208]:
```
# using default params of learning_rate=1.0, n_estimators=100, criteri
gb_clf = GradientBoostingClassifier(random_state=rs)
```

In [209]:
```
model_stats = evaluate_classifier(gb_clf, X_train, y_train, X_test, y_t
                                  model_stats=model_stats, label='defaul
model_stats
```

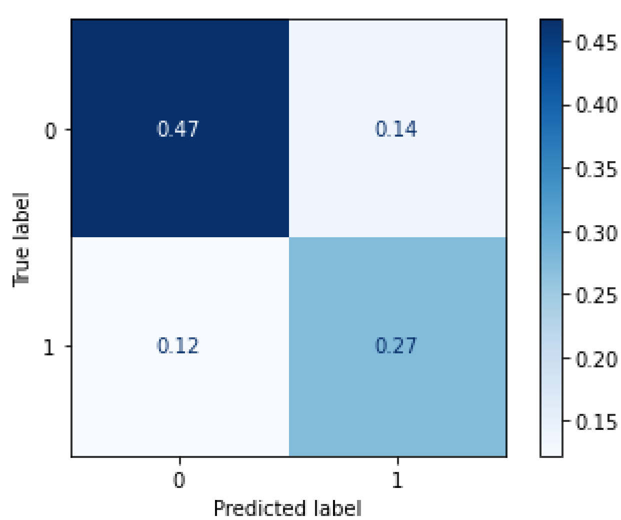
```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)
```

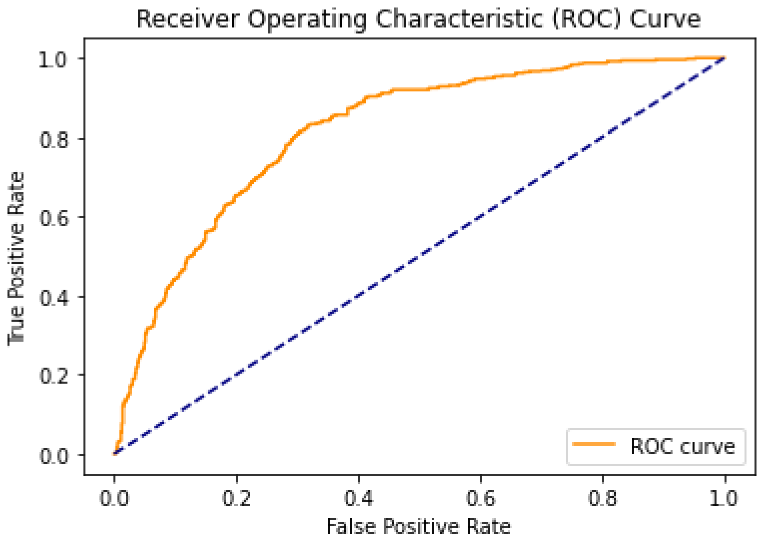
```
CHECK FOR OVERFITTING - seek for test data to perform almost as well as
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)
```

	Accuracy	F1 Score
Training	0.820291	0.823169
Testing	0.740554	0.67846

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recall

              precision    recall  f1-score   support

           0       0.79      0.77      0.78       719
           1       0.67      0.69      0.68       472

    accuracy                           0.74      1191
   macro avg       0.73      0.73      0.73      1191
weighted avg       0.74      0.74      0.74      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8148897362155536
```





Out[209]:

	Model	Label	Train Recall	Test Recall	Train AUC
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676
8	RandomForestClassifier(criterion='entropy', ma...	RF with gridsearch best params, 50 trees	0.8386	0.7564	0.8690
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.8019	0.6780	0.8744
10	GradientBoostingClassifier(random_state=610)	default params, unscaled data	0.8366	0.6907	0.9097

XGBoost - unscaled

In [210]:
```
from xgboost import XGBClassifier
```

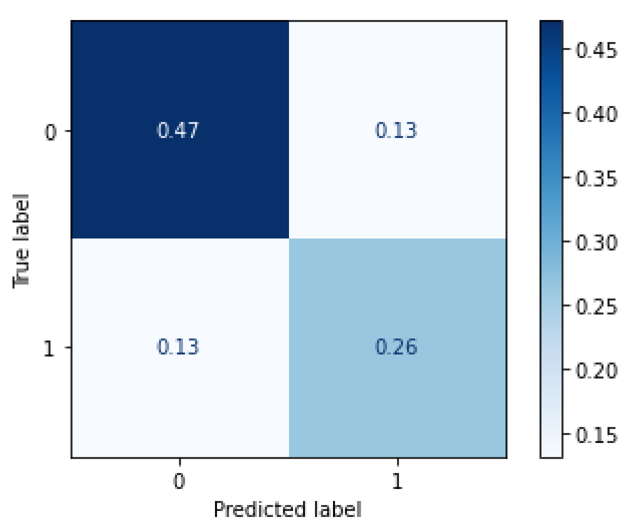
In [211]:
```
xgb_clf = XGBClassifier(random_state=rs)
```

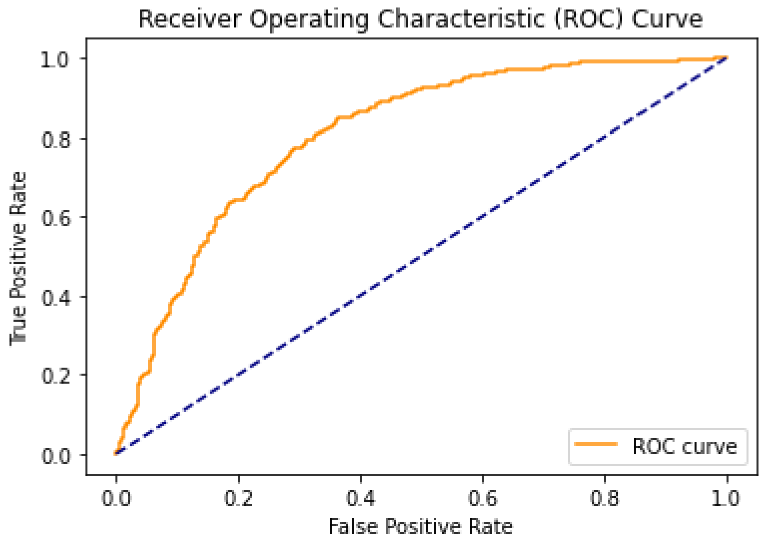
In [212]:
```
model_stats = evaluate_classifier(xgb_clf, X_train, y_train, X_test, y_
                                   model_stats=model_stats, label='defau
model_stats
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)

CHECK FOR OVERFITTING - seek for test data to perform almost as well a
```

	Accuracy	F1 Score
Training	0.991863	0.991887
Testing	0.736356	0.667373

```
************************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.78      0.78      0.78       719
           1       0.67      0.67      0.67       472

    accuracy                           0.74      1191
   macro avg       0.72      0.72      0.72      1191
weighted avg       0.74      0.74      0.74      1191

************************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8055355837910468
```





Out[212]:

	Model	Label	Train Recall	Test Recall	Train AUC
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676
8	RandomForestClassifier(criterion='entropy', ma...	RF with gridsearch best params, 50 trees	0.8386	0.7564	0.8690
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.8019	0.6780	0.8744
10	GradientBoostingClassifier(random_state=610)	default params, unscaled data	0.8366	0.6907	0.9097
11	XGBClassifier(base_score=0.5, booster='gbtree'...	default params, unscaled	0.9948	0.6674	0.9997

XGBoost model is quite overfit, and both AdaBoost and GradientBoosting classifiers are good but not as good as the tuned random forest in recall score.

Support Vector Machine

Support vector machines maximize the decision boundary between points to balance underfitting and overfitting. The slack parameter determines the balance between prioritizing accuracy or maximum margin.

In [213]:
```
from sklearn.svm import SVC
```

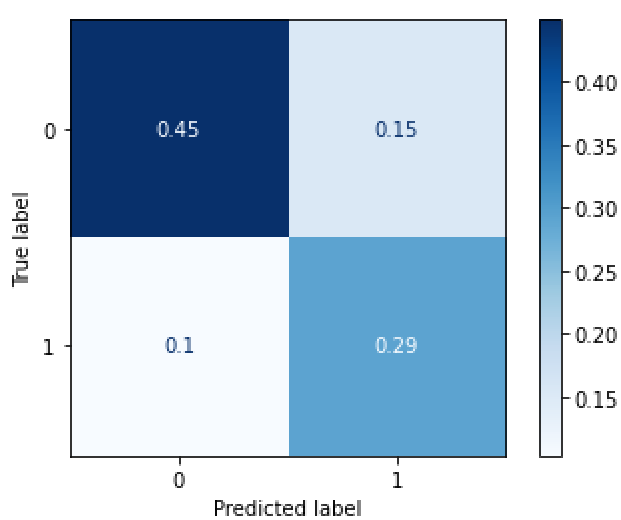
In [214]:
```
svc_clf = SVC(probability=True, random_state=rs)
```

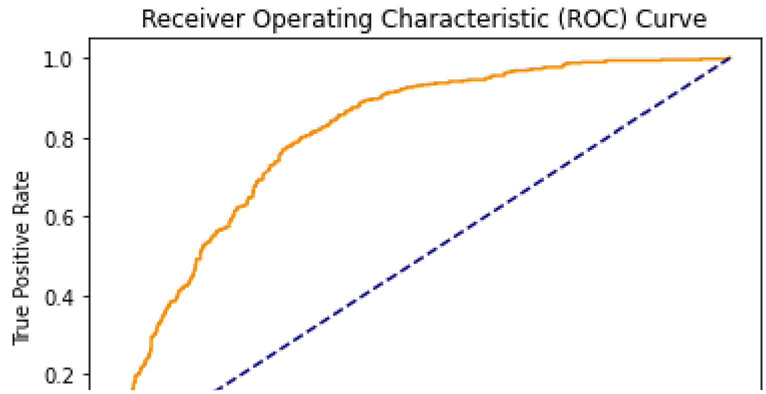
In [215]:
```
model_stats = evaluate_classifier(svc_clf, X_train_scaled, y_train, X_t
                                  model_stats=model_stats, label='scalec
model_stats
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)

CHECK FOR OVERFITTING - seek for test data to perform almost as well a:

/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)
```

	Accuracy	F1 Score
Training	0.838643	0.844355
Testing	0.741394	0.693227

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.81      0.74      0.78       719
           1       0.65      0.74      0.69       472

    accuracy                           0.74      1191
   macro avg       0.73      0.74      0.73      1191
weighted avg       0.75      0.74      0.74      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8146171707409066
```





Out[215]:

	Model	Label	Train Recall	Test Recall	Train AUC
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676
8	RandomForestClassifier(criterion='entropy', ma...	RF with gridsearch best params, 50 trees	0.8386	0.7564	0.8690
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.8019	0.6780	0.8744
10	GradientBoostingClassifier(random_state=610)	default params, unscaled data	0.8366	0.6907	0.9097
11	XGBClassifier(base_score=0.5, booster='gbtree'...	default params, unscaled	0.9948	0.6674	0.9997
12	SVC(probability=True, random_state=610)	scaled	0.8753	0.7373	0.9145

Feature Re-Selection

The first dataset had a lot of features, many of which the best-performing model - tuned random forest - didn't use since the max_features parameter is set to 20.

The next iterations will limit the feature set and rerun models to see if performance improves without as much noise in the predictor set.

Preprocessing

In [216]:

```
# find what features the best model was using
dt_clf2_features = pd.DataFrame(dt_clf2.fit(X_train, y_train).feature_
dt_clf2_features[dt_clf2_features[0]>0]
```

Out[216]:

	0
weight_kg	0.016537
BMI	0.015308
waist_circumference_cm	0.317853
hip_circumference_cm	0.006131
age	0.155861
moderate_activity_minsperweek	0.035357
race_black	0.121119
race_white	0.011548
country_of_birth_usa	0.047186
citizen_status_non_citizen	0.001877
marital_status_married or living with partner	0.034141
covered_medicare_yes	0.237082

In [217]:

```
mega_df.head()
```

Out[217]:

	diagnosis	weight_kg	height_cm	BMI	waist_circumference_cm	hip_circumference_
SEQN						
93705	1	79.5	158.3	31.7	101.8	1
93706	0	66.3	175.7	21.5	79.3	
93707	0	45.4	158.4	18.1	64.1	
93708	1	53.5	150.2	23.7	88.2	
93709	1	88.8	151.1	38.9	113.0	1

In [218]:
```
mega_df.columns
```

Out[218]:
```
Index(['diagnosis', 'weight_kg', 'height_cm', 'BMI', 'waist_circumfere
       'hip_circumference_cm', 'gender', 'age', 'race', 'veteran_statu
       'country_of_birth', 'citizen_status', 'education', 'marital_sta
       'household_size', 'annual_household_income', 'income_poverty_ra
       'coverage_status', 'covered_private', 'covered_medicare',
       'covered_medigap', 'covered_medicaid', 'covered_chip',
       'covered_military', 'covered_state', 'covered_other_gov',
       'covered_single_service', 'prescription_coverage',
       'uninsured_in_last_year', 'sedentary_minsperday',
       'vigorous_activity_minsperweek', 'moderate_activity_minsperweek
       'avg_pulse', 'lifetime_cigarette_smoker', 'current_cigarette_sm
       'ecig_smoker', 'smokeless_tobacco_user'],
      dtype='object')
```

In [219]:
```
# choose the unencoded columns that map to the features used in the be
# not including height and weight, as BMI is a combination of the two
keepcols = ['diagnosis', 'BMI', 'gender', 'age', 'race',
            'waist_circumference_cm', 'country_of_birth', 'citizen_sta
            'education', 'marital_status', 'coverage_status', 'avg_pul

len(keepcols)
```

Out[219]: 12

In [220]:
```
mega_df2 = mega_df[keepcols]
y2 = mega_df2['diagnosis']
X2 = mega_df2.drop(columns='diagnosis')
```

In [221]:
```
# encode categorical columns
X2 = pd.get_dummies(X2, drop_first=True, dtype='float')
X2.head()
```

Out[221]:

	BMI	age	waist_circumference_cm	avg_pulse	gender_male	race_black	race_hisp
SEQN							
93705	31.7	66.0	101.8	50.7	0.0	1.0	
93706	21.5	18.0	79.3	77.3	1.0	0.0	
93707	18.1	NaN	64.1	93.3	0.0	0.0	
93708	23.7	66.0	88.2	67.7	0.0	0.0	
93709	38.9	75.0	113.0	62.3	0.0	1.0	

In [222]:
```
# Train test split
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_s
```

In [223]:
```
# impute nulls
X2_train_imputed = pd.DataFrame(imp.fit_transform(X2_train, y2_train),
                                columns=X2_train.columns, index=X2_trai

X2_test_imputed = pd.DataFrame(imp.transform(X2_test), columns=X2_test.
                               index=X2_test.index)

display(X2_train_imputed.head())
display(X2_test_imputed.head())
```

	BMI	age	waist_circumference_cm	avg_pulse	gender_male	race_black	race_his
SEQN							
102783	26.4	52.0	96.8	81.300000	0.0	0.0	
99996	31.4	51.0	105.0	62.700000	0.0	1.0	
99721	24.7	20.0	87.5	80.000000	0.0	1.0	
97169	33.0	67.0	111.8	71.476713	0.0	1.0	
94465	30.9	31.0	104.7	74.907934	1.0	0.0	

	BMI	age	waist_circumference_cm	avg_pulse	gender_male	race_black	race_his
SEQN							
98959	26.7	75.0	100.2	61.3	1.0	0.0	
102761	26.2	25.0	80.4	45.7	1.0	1.0	
100216	21.2	42.0	74.0	77.0	0.0	0.0	
99151	26.3	22.0	83.1	71.7	0.0	0.0	
99679	30.1	61.0	102.5	56.7	1.0	0.0	

In [224]:
```
# balance classes
y2_train.value_counts(normalize=True)
```

Out[224]:
```
0    0.606723
1    0.393277
Name: diagnosis, dtype: float64
```

In [225]:
```
# balance classes
X2_train_bal, y2_train_bal = smote.fit_sample(X2_train_imputed, y2_tra
```

In [226]:
```
y2_train_bal.value_counts(normalize=True)
```

Out[226]:
```
1    0.5
0    0.5
Name: diagnosis, dtype: float64
```

In [227]:
```
# reset variables
X2_train = X2_train_bal.copy()
X2_test = X2_test_imputed.copy()
y2_train = y2_train_bal.copy()
```

In [228]:
```
# fit transform on train data
X2_train_scaled = pd.DataFrame(scaler.fit_transform(X2_train), columns=

# just transform on test data
X2_test_scaled = pd.DataFrame(scaler.transform(X2_test), columns=X2_tes
```

Modeling

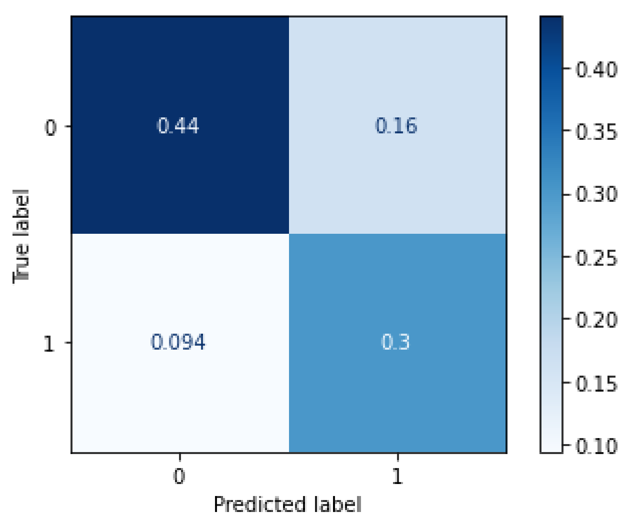
Logistic Regression

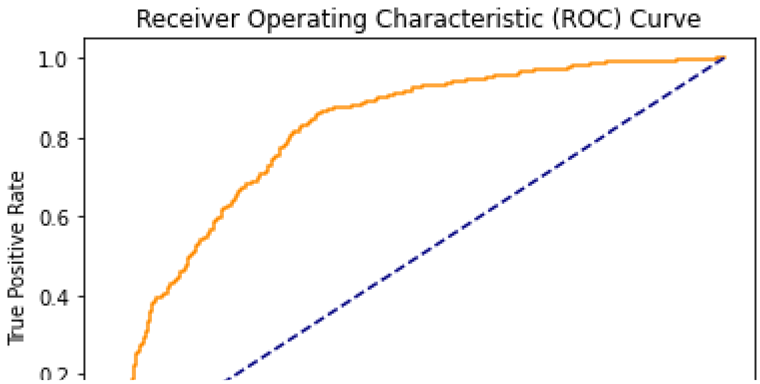
In [229]:
```
log_clf = LogisticRegressionCV(random_state=rs, cv=3)
model_stats = evaluate_classifier(log_clf, X2_train_scaled, y2_train, X
                                  label='limited feature set; scaled',
model_stats
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well as
```

	Accuracy	F1 Score
Training	0.746537	0.75287
Testing	0.743073	0.701754

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recall

              precision    recall  f1-score   support

           0       0.82      0.73      0.77       719
           1       0.65      0.76      0.70       472

    accuracy                           0.74      1191
   macro avg       0.74      0.75      0.74      1191
weighted avg       0.76      0.74      0.75      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8177170505174324
```





Out[229]:

	Model	Label	Train Recall	Test Recall	Train AUC
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676
8	RandomForestClassifier(criterion='entropy', ma...	RF with gridsearch best params, 50 trees	0.8386	0.7564	0.8690
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.8019	0.6780	0.8744
10	GradientBoostingClassifier(random_state=610)	default params, unscaled data	0.8366	0.6907	0.9097
11	XGBClassifier(base_score=0.5, booster='gbtree'...	default params, unscaled	0.9948	0.6674	0.9997
12	SVC(probability=True, random_state=610)	scaled	0.8753	0.7373	0.9145
13	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; scaled	0.7722	0.7627	0.8124

In [230]:
```
model_stats = evaluate_classifier(log_clf, X2_train, y2_train, X2_test,
                                  label='limited feature set; unscaled
model_stats
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

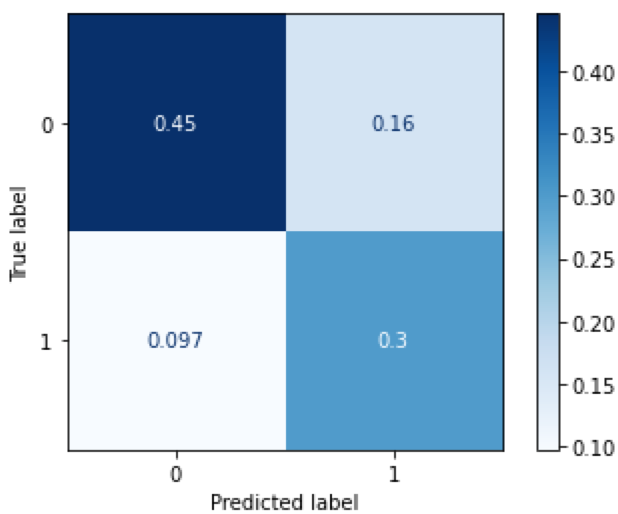
```
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(


CHECK FOR OVERFITTING - seek for test data to perform almost as well as

/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
```

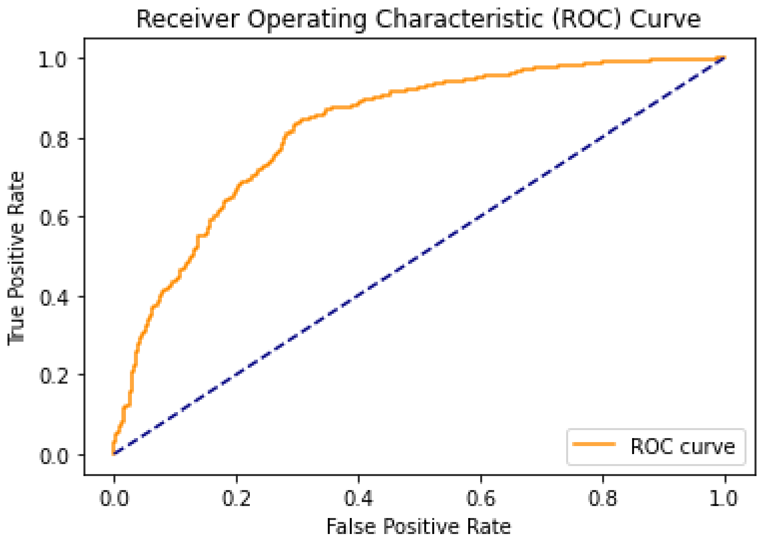
```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
    https://scikit-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-
  n_iter_i = _check_optimize_result(
```

	Accuracy	F1 Score
Training	0.74446	0.749831
Testing	0.744752	0.700787

```
*****************************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.82      0.74      0.78       719
           1       0.65      0.75      0.70       472

    accuracy                           0.74      1191
   macro avg       0.74      0.75      0.74      1191
weighted avg       0.75      0.74      0.75      1191

*****************************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8214033143961716
```





Out[230]:

	Model	Label	Train Recall	Test Recall	Train AUC
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676
8	RandomForestClassifier(criterion='entropy', ma...	RF with gridsearch best params, 50 trees	0.8386	0.7564	0.8690
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.8019	0.6780	0.8744
10	GradientBoostingClassifier(random_state=610)	default params, unscaled data	0.8366	0.6907	0.9097
11	XGBClassifier(base_score=0.5, booster='gbtree'...	default params, unscaled	0.9948	0.6674	0.9997
12	SVC(probability=True, random_state=610)	scaled	0.8753	0.7373	0.9145
13	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; scaled	0.7722	0.7627	0.8124
14	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; unscaled	0.7659	0.7542	0.8136

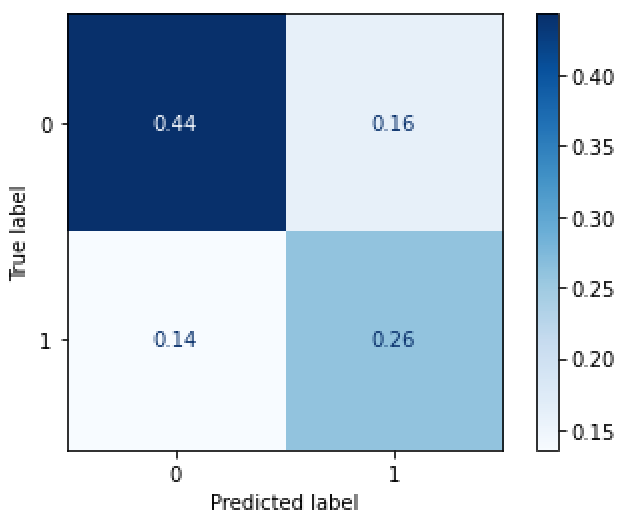
K-Nearest Neighbors

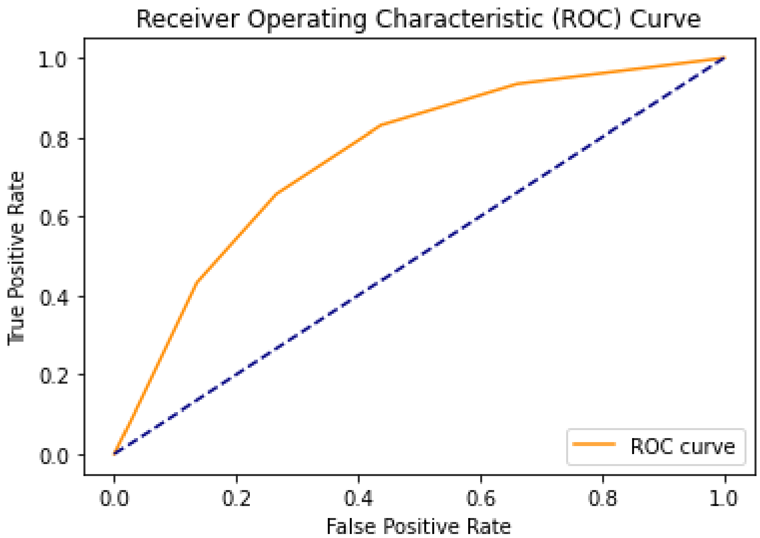
In [231]:
```
model_stats = evaluate_classifier(knn_clf, X2_train_scaled, y2_train, X
                                  label='limited feature set; scaled',
model_stats
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well as
```

	Accuracy	F1 Score
Training	0.823061	0.825478
Testing	0.70361	0.637205

```
*********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recall

              precision    recall  f1-score   support

           0       0.77      0.73      0.75       719
           1       0.62      0.66      0.64       472

    accuracy                           0.70      1191
   macro avg       0.69      0.70      0.69      1191
weighted avg       0.71      0.70      0.70      1191

*********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.7530144268169067
```





Out[231]:

	Model	Label	Train Recall	Test Recall	Train AUC
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676
8	RandomForestClassifier(criterion='entropy', ma...	RF with gridsearch best params, 50 trees	0.8386	0.7564	0.8690
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.8019	0.6780	0.8744
10	GradientBoostingClassifier(random_state=610)	default params, unscaled data	0.8366	0.6907	0.9097
11	XGBClassifier(base_score=0.5, booster='gbtree'...	default params, unscaled	0.9948	0.6674	0.9997
12	SVC(probability=True, random_state=610)	scaled	0.8753	0.7373	0.9145
13	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; scaled	0.7722	0.7627	0.8124
14	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; unscaled	0.7659	0.7542	0.8136
15	KNeighborsClassifier()	limited feature set; scaled	0.8369	0.6568	0.9081

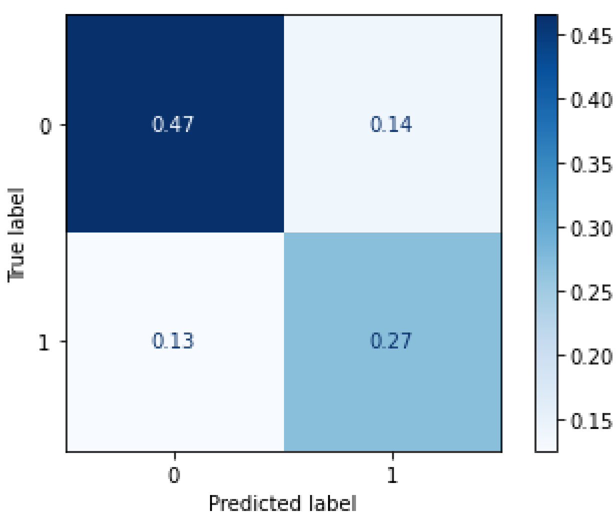
Boosting Ensemble Methods

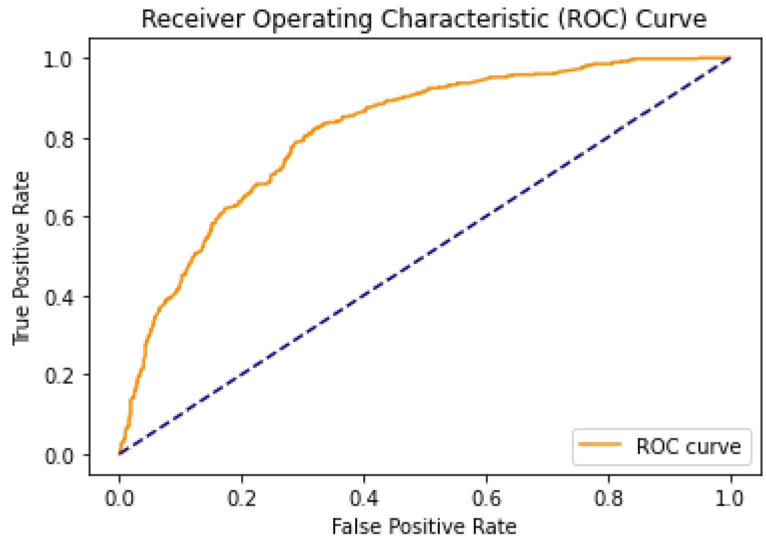
In [232]:
```
# Adaboost
model_stats = evaluate_classifier(ab_clf, X2_train, y2_train, X2_test, 
                                  label='limited feature set; unscaled')
model_stats
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well as
```

	Accuracy	F1 Score
Training	0.774758	0.779304
Testing	0.735516	0.671533

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recall

              precision    recall  f1-score   support

           0       0.79      0.77      0.78       719
           1       0.66      0.68      0.67       472

    accuracy                           0.74      1191
   macro avg       0.72      0.73      0.73      1191
weighted avg       0.74      0.74      0.74      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8120211687607553
```





Out[232]:

	Model	Label	Train Recall	Test Recall	Train AUC
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676
8	RandomForestClassifier(criterion='entropy', ma...	RF with gridsearch best params, 50 trees	0.8386	0.7564	0.8690
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.8019	0.6780	0.8744
10	GradientBoostingClassifier(random_state=610)	default params, unscaled data	0.8366	0.6907	0.9097
11	XGBClassifier(base_score=0.5, booster='gbtree'...	default params, unscaled	0.9948	0.6674	0.9997
12	SVC(probability=True, random_state=610)	scaled	0.8753	0.7373	0.9145
13	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; scaled	0.7722	0.7627	0.8124
14	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; unscaled	0.7659	0.7542	0.8136
15	KNeighborsClassifier()	limited feature set; scaled	0.8369	0.6568	0.9081

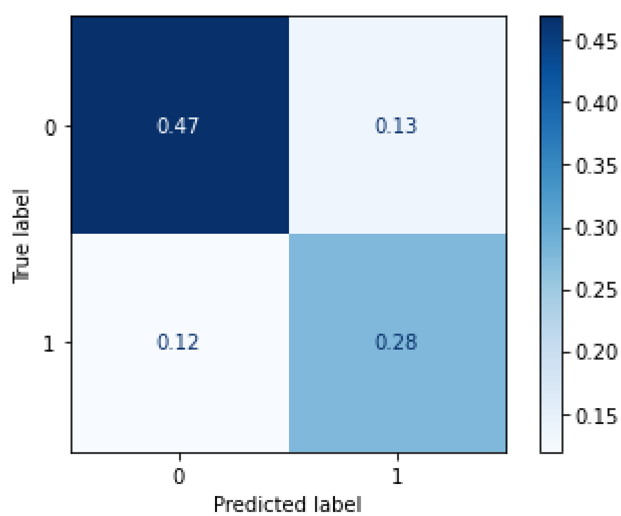
Model	Label	Train Recall	Test Recall	Train AUC

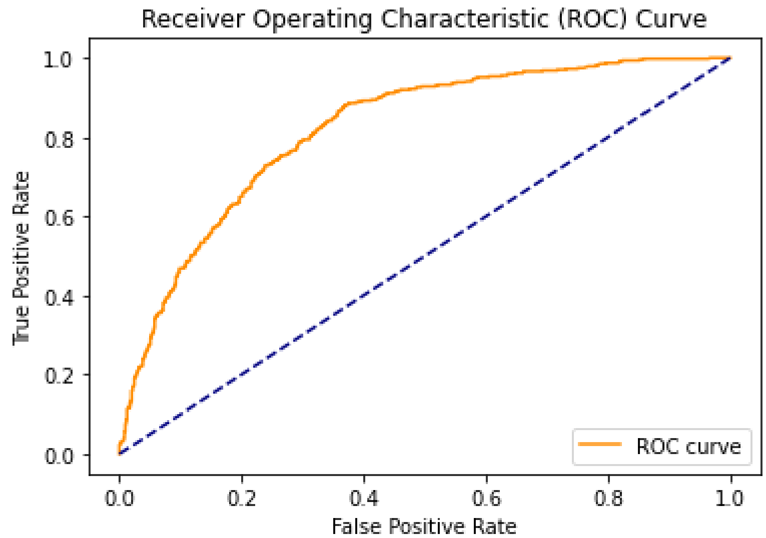
In [233]:
```
# Gradient boost
model_stats = evaluate_classifier(gb_clf, X2_train, y2_train, X2_test,
                                  label='limited feature set, unscaled')
model_stats
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well a:
```

	Accuracy	F1 Score
Training	0.810942	0.813206
Testing	0.746432	0.686071

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal]

              precision    recall  f1-score   support

           0       0.80      0.78      0.79       719
           1       0.67      0.70      0.69       472

    accuracy                           0.75      1191
   macro avg       0.74      0.74      0.74      1191
weighted avg       0.75      0.75      0.75      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8187852125126707
```





Out[233]:

	Model	Label	Train Recall	Test Recall	Train AUC
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676
8	RandomForestClassifier(criterion='entropy', ma...	RF with gridsearch best params, 50 trees	0.8386	0.7564	0.8690
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.8019	0.6780	0.8744
10	GradientBoostingClassifier(random_state=610)	default params, unscaled data	0.8366	0.6907	0.9097
11	XGBClassifier(base_score=0.5, booster='gbtree'...	default params, unscaled	0.9948	0.6674	0.9997
12	SVC(probability=True, random_state=610)	scaled	0.8753	0.7373	0.9145
13	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; scaled	0.7722	0.7627	0.8124
14	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; unscaled	0.7659	0.7542	0.8136
15	KNeighborsClassifier()	limited feature set; scaled	0.8369	0.6568	0.9081

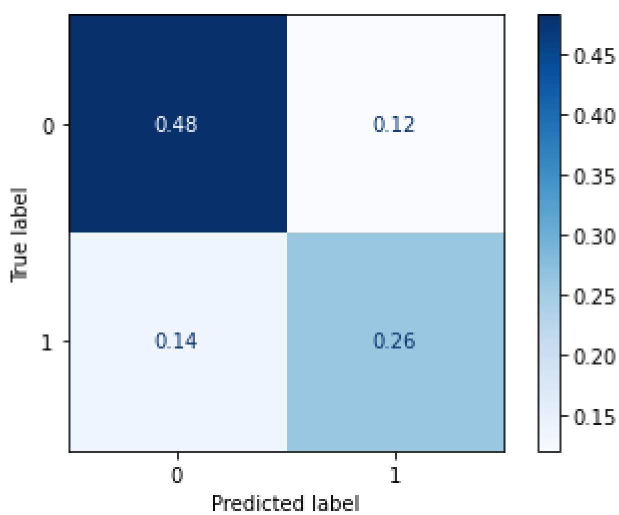
	Model	Label	Train Recall	Test Recall	Train AUC
16	AdaBoostClassifier(random_state=610)	limited feature set; unscaled	0.7954	0.6822	0.8636
17	GradientBoostingClassifier(random_state=610)	limited feature set, unscaled	0.8231	0.6992	0.8959

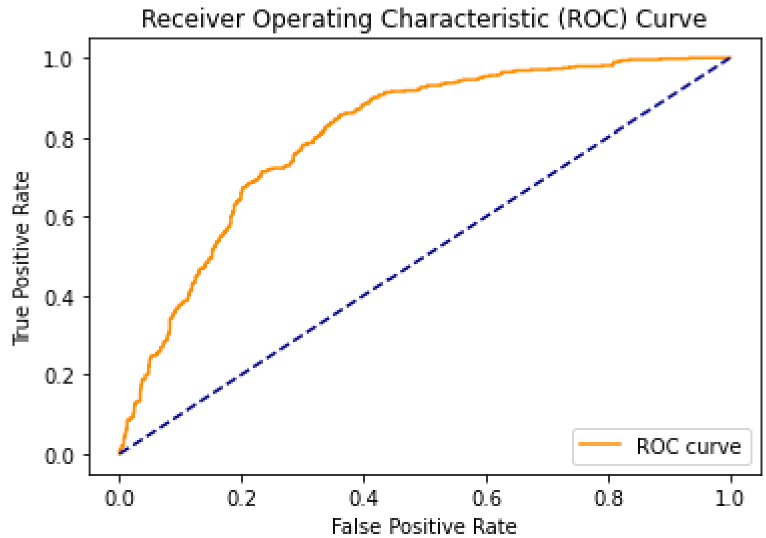
In [234]:
```
# XBG
model_stats = evaluate_classifier(xgb_clf, X2_train, y2_train, X2_test,
                                  label='limited feature set, unscaled')
model_stats
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well as
```

	Accuracy	F1 Score
Training	0.956891	0.956808
Testing	0.743913	0.67027

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recall

              precision    recall  f1-score   support

           0       0.78      0.80      0.79       719
           1       0.68      0.66      0.67       472

    accuracy                           0.74      1191
   macro avg       0.73      0.73      0.73      1191
weighted avg       0.74      0.74      0.74      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8048681667098843
```





Out[234]:

	Model	Label	Train Recall	Test Recall	Train AUC
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676
8	RandomForestClassifier(criterion='entropy', ma...	RF with gridsearch best params, 50 trees	0.8386	0.7564	0.8690
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.8019	0.6780	0.8744
10	GradientBoostingClassifier(random_state=610)	default params, unscaled data	0.8366	0.6907	0.9097
11	XGBClassifier(base_score=0.5, booster='gbtree'...	default params, unscaled	0.9948	0.6674	0.9997
12	SVC(probability=True, random_state=610)	scaled	0.8753	0.7373	0.9145
13	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; scaled	0.7722	0.7627	0.8124
14	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; unscaled	0.7659	0.7542	0.8136
15	KNeighborsClassifier()	limited feature set; scaled	0.8369	0.6568	0.9081

	Model	Label	Train Recall	Test Recall	Train AUC
16	AdaBoostClassifier(random_state=610)	limited feature set; unscaled	0.7954	0.6822	0.8636
17	GradientBoostingClassifier(random_state=610)	limited feature set, unscaled	0.8231	0.6992	0.8959
18	XGBClassifier(base_score=0.5, booster='gbtree'...	limited feature set, unscaled	0.9550	0.6568	0.9927

Support Vector Machine

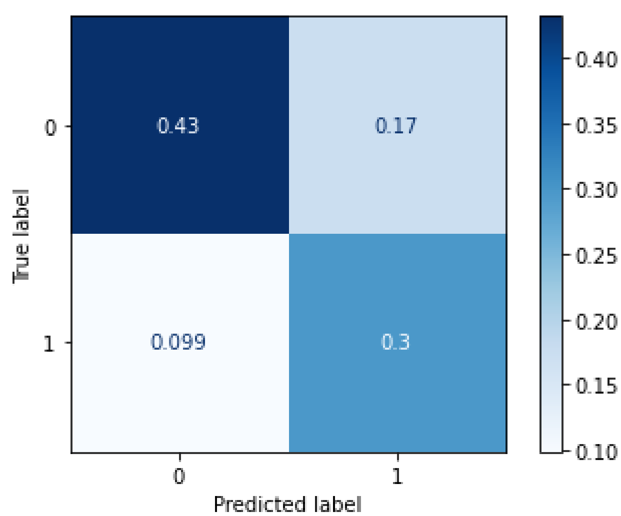
In [235]:

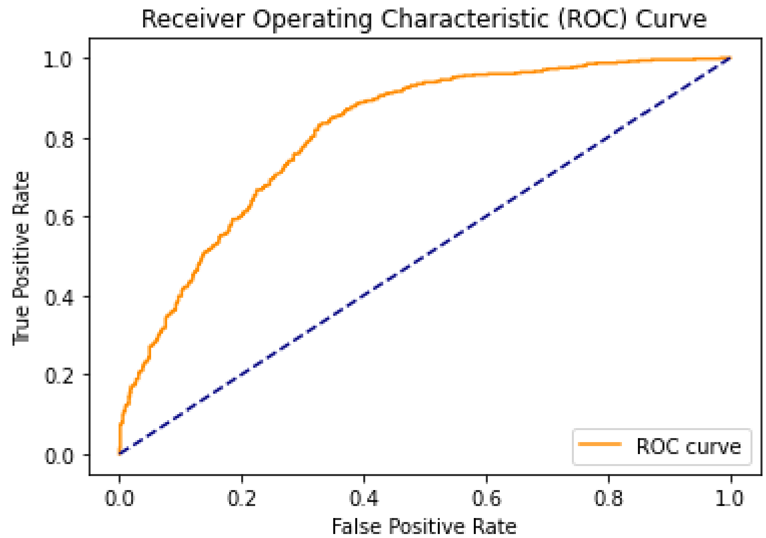
```
model_stats = evaluate_classifier(svc_clf, X2_train_scaled, y2_train, X
                                  model_stats=model_stats, label='limite
model_stats
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well as
```

	Accuracy	F1 Score
Training	0.80367	0.813241
Testing	0.728799	0.686712

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recall

              precision    recall  f1-score   support

           0       0.81      0.71      0.76       719
           1       0.63      0.75      0.69       472

    accuracy                           0.73      1191
   macro avg       0.72      0.73      0.72      1191
weighted avg       0.74      0.73      0.73      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8092321609580161
```





Out[235]:

	Model	Label	Train Recall	Test Recall	Train AUC
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676
8	RandomForestClassifier(criterion='entropy', ma...	RF with gridsearch best params, 50 trees	0.8386	0.7564	0.8690
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.8019	0.6780	0.8744
10	GradientBoostingClassifier(random_state=610)	default params, unscaled data	0.8366	0.6907	0.9097
11	XGBClassifier(base_score=0.5, booster='gbtree'...	default params, unscaled	0.9948	0.6674	0.9997
12	SVC(probability=True, random_state=610)	scaled	0.8753	0.7373	0.9145
13	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; scaled	0.7722	0.7627	0.8124
14	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; unscaled	0.7659	0.7542	0.8136
15	KNeighborsClassifier()	limited feature set; scaled	0.8369	0.6568	0.9081

	Model	Label	Train Recall	Test Recall	Train AUC
16	AdaBoostClassifier(random_state=610)	limited feature set; unscaled	0.7954	0.6822	0.8636
17	GradientBoostingClassifier(random_state=610)	limited feature set, unscaled	0.8231	0.6992	0.8959
18	XGBClassifier(base_score=0.5, booster='gbtree'...	limited feature set, unscaled	0.9550	0.6568	0.9927
19	SVC(probability=True, random_state=610)	limited feature set SVC scaled	0.8549	0.7500	0.8838

In [236]:

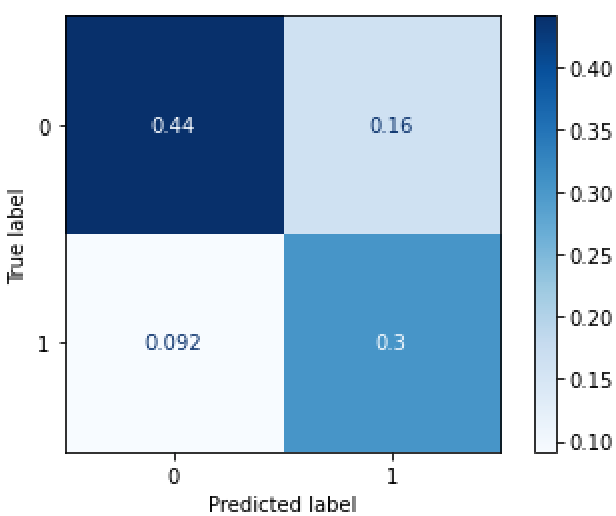
```
svc_clf2 = SVC(kernel='linear', random_state=rs, probability=True)
```

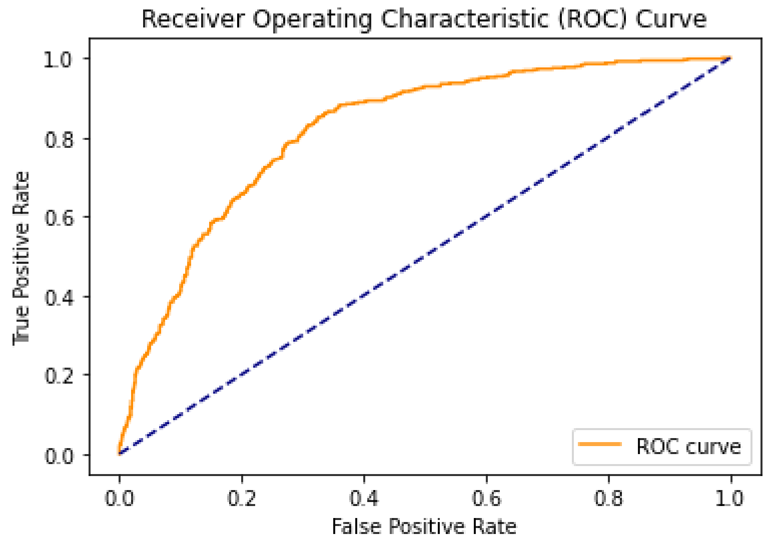
In [237]:
```
model_stats = evaluate_classifier(svc_clf2, X2_train_scaled, y2_train,
                                  model_stats=model_stats, label='lineaı
model_stats
```

```
CHECK FOR OVERFITTING - seek for test data to perform almost as well a:
```

	Accuracy	F1 Score
Training	0.746191	0.753696
Testing	0.746432	0.706226

```
**********************************************************************

CHECK ACCURACY, PRECISION, RECALL, & F1 SCORE - seek to maximize recal

              precision    recall  f1-score   support

           0       0.83      0.73      0.78       719
           1       0.65      0.77      0.71       472

    accuracy                           0.75      1191
   macro avg       0.74      0.75      0.74      1191
weighted avg       0.76      0.75      0.75      1191

**********************************************************************

CHECK Test ROC CURVE - seek to maximize area under the curve

AUC: 0.8189458051436789
```





Out[237]:

	Model	Label	Train Recall	Test Recall	Train AUC
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.7902	0.7521	0.8254
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.7763	0.7521	0.8044
2	KNeighborsClassifier()	defaul params, scaled data	0.8608	0.6419	0.9064
3	KNeighborsClassifier()	default params, unscaled data	0.8393	0.5657	0.8619
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.8272	0.7394	0.8405
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.0000	0.5953	1.0000
6	GridSearchCV(cv=3, estimator=DecisionTreeClass...	unscaled	0.7954	0.7034	0.8464
7	BaggingClassifier(base_estimator=DecisionTreeC...	25 bagged trees with base estimator=gridsearch...	0.8328	0.7415	0.8676
8	RandomForestClassifier(criterion='entropy', ma...	RF with gridsearch best params, 50 trees	0.8386	0.7564	0.8690
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.8019	0.6780	0.8744
10	GradientBoostingClassifier(random_state=610)	default params, unscaled data	0.8366	0.6907	0.9097
11	XGBClassifier(base_score=0.5, booster='gbtree'...	default params, unscaled	0.9948	0.6674	0.9997
12	SVC(probability=True, random_state=610)	scaled	0.8753	0.7373	0.9145
13	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; scaled	0.7722	0.7627	0.8124
14	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; unscaled	0.7659	0.7542	0.8136
15	KNeighborsClassifier()	limited feature set; scaled	0.8369	0.6568	0.9081

	Model	Label	Train Recall	Test Recall	Train AUC
16	AdaBoostClassifier(random_state=610)	limited feature set; unscaled	0.7954	0.6822	0.8636
17	GradientBoostingClassifier(random_state=610)	limited feature set, unscaled	0.8231	0.6992	0.8959
18	XGBClassifier(base_score=0.5, booster='gbtree'...	limited feature set, unscaled	0.9550	0.6568	0.9927
19	SVC(probability=True, random_state=610)	limited feature set SVC scaled	0.8549	0.7500	0.8838
20	SVC(kernel='linear', probability=True, random_...	linear SVC limited feature set scaled	0.7767	0.7691	0.8122

Evaluation & Interpretation

As previously mentioned, the most important evaluation metric for this report is Test Recall, or how many true positive diagnoses the model predicted in new data it hadn't seen before. As such, the next cells sort and style the iteratively populated `model_stats` dataframe to see which model performed the best.

In [238]:
```
model_stats['False Negatives Normalized'] = model_stats['False Negative
model_stats.sort_values('Test Recall', ascending=False, inplace=True)
```

In [239]:

```
# sort the model_stats df that has been collecting model results itera
# highlight the highest values of all columns except false negatives, w
model_stats.style.highlight_max(subset=['Test Recall', 'Test AUC', 'Tes
                   .highlight_min(subset=['False Negatives', 'False Negat
```

Out[239]:

	Model	Label	R
20	SVC(kernel='linear', probability=True, random_state=610)	linear SVC limited feature set scaled	0.77
13	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; scaled	0.77
8	RandomForestClassifier(criterion='entropy', max_depth=5, max_features=20, min_samples_leaf=4, n_estimators=50, random_state=610)	RF with gridsearch best params, 50 trees	0.83
14	LogisticRegressionCV(cv=3, random_state=610)	limited feature set; unscaled	0.76
1	LogisticRegressionCV(cv=3, random_state=610)	default params, unscaled data	0.77
0	LogisticRegressionCV(cv=3, random_state=610)	default params, scaled data	0.79
19	SVC(probability=True, random_state=610)	limited feature set SVC scaled	0.85
7	BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entropy', max_depth=5, max_features=20, min_samples_leaf=4, random_state=610), n_estimators=25, random_state=610)	25 bagged trees with base estimator=gridsearch best params	0.83
4	KNeighborsClassifier(n_neighbors=23)	k=23, scaled data	0.82
12	SVC(probability=True, random_state=610)	scaled	0.87
6	GridSearchCV(cv=3, estimator=DecisionTreeClassifier(random_state=610), param_grid={'criterion': ['entropy', 'gini'], 'max_depth': [None, 5, 10, 15, 20], 'max_features': [None, 5, 10, 15, 20], 'min_samples_leaf': [1, 2, 3, 4]}, return_train_score=True)	unscaled	0.79
17	GradientBoostingClassifier(random_state=610)	limited feature set, unscaled	0.82
10	GradientBoostingClassifier(random_state=610)	default params, unscaled data	0.83
16	AdaBoostClassifier(random_state=610)	limited feature set; unscaled	0.79
9	AdaBoostClassifier(random_state=610)	default params, unscaled	0.80
11	XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1, importance_type='gain', interaction_constraints='', learning_rate=0.300000012, max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=610, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact', validate_parameters=1, verbosity=None)	default params, unscaled	0.99
15	KNeighborsClassifier()	limited feature set; scaled	0.83

	Model	Label	R
18	XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1, importance_type='gain', interaction_constraints='', learning_rate=0.300000012, max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=610, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact', validate_parameters=1, verbosity=None)	limited feature set, unscaled	0.95
2	KNeighborsClassifier()	defaul params, scaled data	0.86
5	DecisionTreeClassifier(random_state=610)	default params, unscaled	1.00
3	KNeighborsClassifier()	default params, unscaled data	0.83

Overall, the best models are consistently performing better than random chance and simple stratification models.

Test Recall

The maximum test recall of all models is consistently around .75-.8, meaning 75-80% of true diabetic/prediabetic diagnoses are correctly predicted by the model. The best performing models resulted in between 8.5-9.5% false negatives, or diabetics who were incorrectly predicted to be healthy.

Test Accuracy

The highest accuracy scores are consistently between .71 and .76, indicating that the model predicts the correct diagnosis about 71-76% of the time.

Interpreting the Support Vector Machine

The linear Support Vector Machine generates coefficients for each feature which, together with a calculated intercept, draw a decision boundary that predicts the target class. These coefficients can be used to understand how the model is using each feature to come to a prediction.

In [240]:
```
svc_coefs = pd.DataFrame(svc_clf2.fit(X2_train_scaled, y2_train).coef_,
                                    columns=X2_train_scaled.columns).T
svc_coefs.rename(columns={0: 'coefficient'}, inplace=True)
```

In [241]:
```
def relationship(x):
    y=None
    if x>0:
        y='positive'
    elif x<0:
        y='negative'
    else:
        y='no relationship'
    return y
```

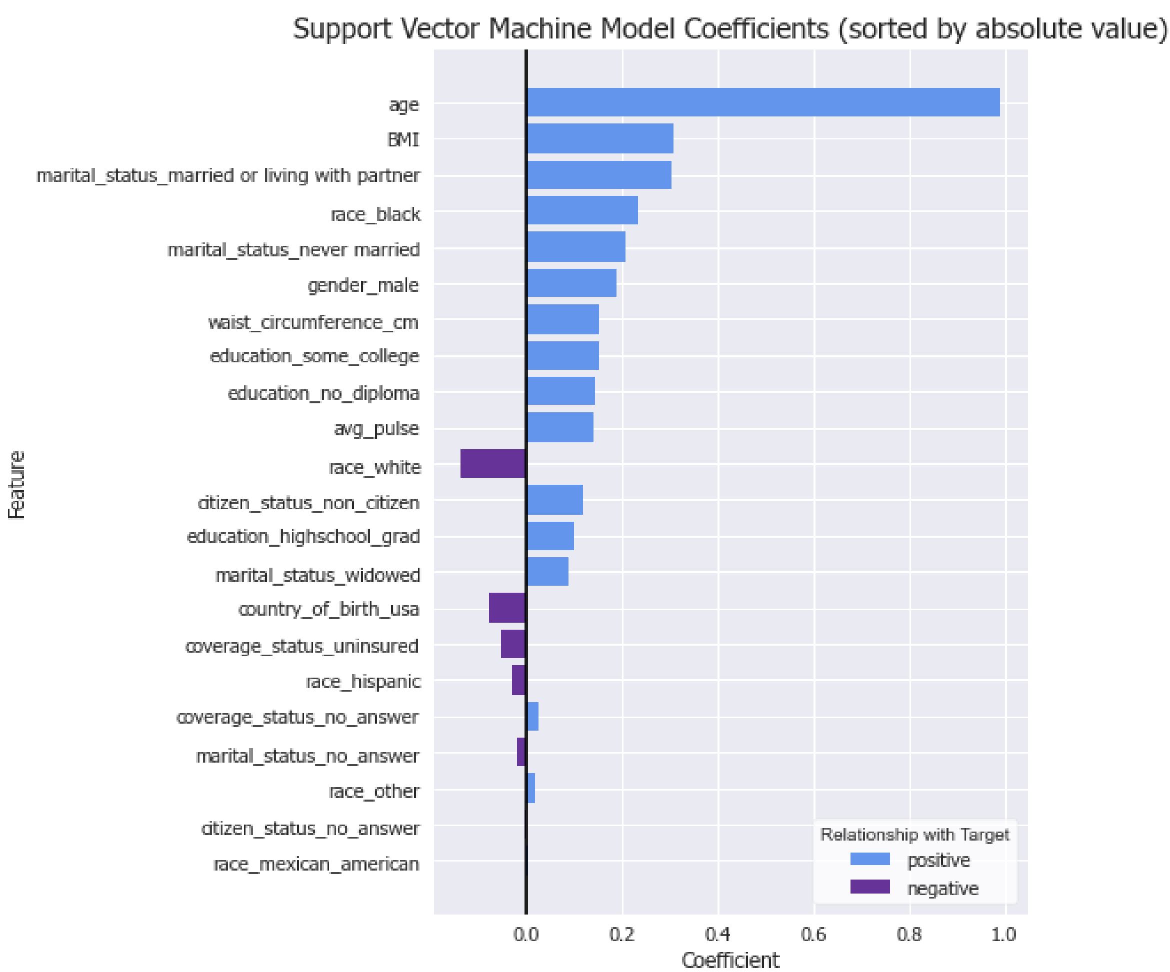
In [242]:
```
svc_coefs['relationship'] = svc_coefs['coefficient'].map(lambda x: rela
svc_coefs
```

Out[242]:

	coefficient	relationship
BMI	0.305900	positive
age	0.990892	positive
waist_circumference_cm	0.149740	positive
avg_pulse	0.139696	positive
gender_male	0.188301	positive
race_black	0.231212	positive
race_hispanic	-0.031562	negative
race_mexican_american	0.004261	positive
race_other	0.017383	positive
race_white	-0.138459	negative
country_of_birth_usa	-0.079173	negative
citizen_status_no_answer	-0.004494	negative
citizen_status_non_citizen	0.116101	positive
education_highschool_grad	0.098058	positive
education_no_diploma	0.144461	positive
education_some_college	0.149711	positive
marital_status_married or living with partner	0.304178	positive
marital_status_never married	0.207266	positive
marital_status_no_answer	-0.019246	negative
marital_status_widowed	0.087126	positive
coverage_status_no_answer	0.025564	positive
coverage_status_uninsured	-0.051003	negative

In [243]:
```
svc_coefs_abs = svc_coefs.iloc[svc_coefs.coefficient.abs().argsort()]
```

In [244]:
```
# Plot coefficients used in the best-performing model (SVC) in order o

# Shoutout to Shane Lynn for introducing me to Patches
# Source: https://www.shanelynn.ie/bar-plots-in-python-using-pandas-da

colors = {'positive':'cornflowerblue', 'negative':'rebeccapurple'}
from matplotlib.patches import Patch

plt.style.use('seaborn')
plt.figure(figsize=(6,9), facecolor='white')
plt.barh(y=svc_coefs_abs.index, width=svc_coefs_abs.coefficient,
        color=svc_coefs_abs['relationship'].replace(colors))
plt.axvline(x=0, color='black')
plt.xlabel('Coefficient', fontfamily='tahoma', fontsize=12)
plt.xticks(fontsize=11, fontfamily='tahoma')
plt.ylabel('Feature', fontfamily='tahoma', fontsize=12)
plt.yticks(fontsize=11, fontfamily='tahoma')
plt.title('Support Vector Machine Model Coefficients (sorted by absolut
          fontdict = {'family': 'tahoma', 'size':16})
plt.legend(
    [Patch(facecolor=colors['positive']),
     Patch(facecolor=colors['negative'])],
    ['positive', 'negative'],
    loc='lower right', frameon=True, facecolor='white', title='Relatio
    prop={'family': 'tahoma', 'size': 11})
plt.savefig('./images/svm_coefs.jpg', bbox_inches='tight', pad_inches=
plt.show();
```



Features with positive coefficients have a positive relationship with diagnosis. For example, higher age values have a higher probability of diabetes/prediabetes, and lower age values have a lower probability of diabetes/prediabetes.

Features with negative coefficients have a negative relationship with the diagosis. For example, when a person's race is not white (race_white value is 0), there is a higher probability of diabetes/prediabetes, and when race_white value is 1, there is a lower probability of diabetes/prediabetes.

It is important to note that these are simply observations, not to imply causation. There are many factors beyond those evaluated here that together influence a person's health and circumstances.

Interpretting the Logistic Regression

Similarly, the logistic regression model is best interpreted using coefficients.

In [245]:

```
lr_coefs = pd.DataFrame(log_clf.fit(X_train_scaled, y_train).coef_,
                                 columns=X_train_scaled.columns).T

lr_coefs.rename(columns={0: 'coefficient'}, inplace=True)
```

```
/Users/kristincooper/opt/anaconda3/envs/learn-env/lib/python3.8/site-pa
  return f(**kwargs)
```

In [246]:
```
lr_coefs['relationship'] = lr_coefs['coefficient'].map(lambda x: relat:
lr_coefs
```

Out[246]:

	coefficient	relationship
weight_kg	-0.003969	negative
height_cm	-0.031489	negative
BMI	0.423391	positive
waist_circumference_cm	0.502465	positive
hip_circumference_cm	-0.279188	negative
age	1.069344	positive
household_size	0.009377	positive
annual_household_income	0.014223	positive
income_poverty_ratio	0.050109	positive
sedentary_minsperday	-0.023882	negative
vigorous_activity_minsperweek	0.003185	positive
moderate_activity_minsperweek	-0.024660	negative
avg_pulse	0.147236	positive
gender_male	0.140738	positive
race_black	0.277205	positive
race_hispanic	-0.057839	negative
race_mexican_american	-0.031829	negative
race_other	0.016785	positive
race_white	-0.183077	negative
veteran_status_yes	-0.086975	negative
country_of_birth_usa	-0.208223	negative
citizen_status_no_answer	-0.018978	negative
citizen_status_non_citizen	0.056412	positive
education_highschool_grad	0.120277	positive
education_no_diploma	0.190555	positive
education_some_college	0.158730	positive
marital_status_married or living with partner	0.197218	positive
marital_status_never married	0.101474	positive
marital_status_no_answer	-0.018574	negative
marital_status_widowed	0.048336	positive
coverage_status_no_answer	0.001104	positive
coverage_status_uninsured	-0.115808	negative
covered_private_yes	-0.018917	negative

	coefficient	relationship
covered_medicare_yes	0.017615	positive
covered_medigap_yes	-0.006686	negative
covered_medicaid_yes	-0.041275	negative
covered_chip_yes	0.008534	positive
covered_military_yes	0.026312	positive
covered_state_yes	0.041381	positive
covered_other_gov_yes	0.018733	positive
covered_single_service_yes	-0.051231	negative
prescription_coverage_no_answer	0.075814	positive
prescription_coverage_yes	-0.005378	negative
uninsured_in_last_year_no_answer	0.006595	positive
uninsured_in_last_year_yes	-0.014603	negative
lifetime_cigarette_smoker_yes	0.022808	positive
current_cigarette_smoker_yes	0.042493	positive

In [247]:
```
lr_coefs_abs = lr_coefs.iloc[lr_coefs.coefficient.abs().argsort()]
```

In [248]:
```
# Plot coefficients used in the logistic regression cv in order of abs

# Shoutout to Shane Lynn for introducing me to Patches
# Source: https://www.shanelynn.ie/bar-plots-in-python-using-pandas-da

colors = {'positive':'cornflowerblue', 'negative':'rebeccapurple'}
from matplotlib.patches import Patch

plt.style.use('seaborn')
plt.figure(figsize=(6,14), facecolor='white')
plt.barh(y=lr_coefs_abs.index, width=lr_coefs_abs.coefficient,
        color=lr_coefs_abs['relationship'].replace(colors))
plt.axvline(x=0, color='black')
plt.xlabel('Coefficient', fontfamily='tahoma', fontsize=12)
plt.xticks(fontsize=11, fontfamily='tahoma')
plt.ylabel('Feature', fontfamily='tahoma', fontsize=12)
plt.yticks(fontsize=11, fontfamily='tahoma')
plt.title('Logistic Regression Model Coefficients (sorted by absolute
          fontdict = {'family': 'tahoma', 'size':16})
plt.legend(
    [Patch(facecolor=colors['positive']),
     Patch(facecolor=colors['negative'])],
    ['positive', 'negative'],
    loc='lower right', frameon=True, facecolor='white', title='Relatio
    prop={'family': 'tahoma', 'size': 11})
plt.savefig('./images/lr_coefs.jpg', pad_inches=3)
plt.show();
```

Caveats / Future Enhancements

Caveats:

- Coefficients represent observed relationships of a relatively small sample and should not be considered causal. There are countless factors that influence a person's health such as family history, living environment, and many more.

- This model is not intended to predict diabetes in children under 18 or pregnant women
- This model was trained on a sample comprised only of individuals in the USA. Some data, notably insurance coverage status, is quite specific to this sample.

Future Enhancements:

- Create a multivariate classifier to differentiate between prediabetes and diabetes.
- Incorporate time-series using data from prior year NHANES.

Appendix

Data reviewed but not used

Lab - Cholesterol

https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/TCHOL_J.htm (https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/TCHOL_J.htm)

https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/TRIGLY_J.htm (https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/TRIGLY_J.htm)

https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/HDL_J.htm (https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/HDL_J.htm)

A complete cholesterol test — also called a lipid panel or lipid profile — is a blood test that can measure the amount of cholesterol and triglycerides in your blood. The blood lipids measurements in NHANES include total cholesterol, high-density lipoprotein cholesterol (HDL-C), low-density lipoproteins cholesterol (LDL-C), and triglycerides.

- **Total cholesterol** - sum of your blood's cholesterol content
  - Less than 200 mg/dL - desirable
  - 200-239 mg/dL - borderline
  - Greater than 240 mg/dL - high
- **High-density lipoprotein (HDL) cholesterol** - the "good" cholesterol because it helps carry away LDL cholesterol, thus keeping arteries open and your blood flowing more freely.
  - Greater than 60 mg/dL - best
  - 40-59 mg/dL - good
  - Less than 50 (women) or 40 (men) - poor
- **Low-density lipoprotein (LDL) cholesterol** "bad" cholesterol; too much of it in your blood causes the buildup of fatty deposits (plaques) in your arteries (atherosclerosis), which reduces blood flow. These plaques sometimes rupture and can lead to a heart attack or stroke.
  - Less than 100 mg/dL - optimal
  - 100-129 mg/dL - high for those with coronary artery disease
  - 130-159 mg/dL - borderline
  - Greater than 160 mg/dL - high

  - Greater than 190 mg/dL - very high
- **Triglycerides** - a type of fat in the blood created from calories your body doesn't need. High triglyceride levels are associated with several factors, including being overweight, eating too many sweets or drinking too much alcohol, smoking, being sedentary, or having diabetes with elevated blood sugar levels.
  - Less than 150 mg/dL - desirable
  - 150-199 mg/dL - borderline
  - 200-499 mg/dL - high
  - Greater than 500 mg/dL - very high

https://www.mayoclinic.org/tests-procedures/cholesterol-test/about/pac-20384601 (https://www.mayoclinic.org/tests-procedures/cholesterol-test/about/pac-20384601)

In [249]:
```
# chol_total = chol_total.drop(columns='LBDTCSI').rename(mapper={'LBXT(
# chol_total
```

In [250]:
```
# chol_hdl = chol_hdl.drop(columns='LBDHDDSI').rename(mapper={'LBDHDD'.
# chol_hdl
```

In [251]:
```
# chol_ldl
```

In [252]:
```
# chol_ldl.describe().round(1)
```

In [253]:
```
# chol_ldl.columns
```

In [254]:
```
# keepcols_chol_ldl = ['LBXTR', 'LBDLDL']
# mapper_chol_ldl = {'LBXTR': 'triglyceride', 'LBDLDL': 'cholesterol_l(
```

In [255]:
```
# keep_chol_ldl = chol_ldl[keepcols_chol_ldl].rename(mapper=mapper_cho.
# keep_chol_ldl
```

In [256]:
```
# chol1 = pd.merge(keep_chol_ldl, chol_hdl, how='outer', on='SEQN')
# chol1
```

In [257]:
```
# cholesterol = pd.merge(chol1, chol_total, how='outer', on='SEQN')
# cholesterol
```

Lab - Insulin

In [258]:
```
# insulin
```

In [259]:
```
# keep_insulin = pd.DataFrame(insulin['LBXIN'])
# keep_insulin.rename(mapper={'LBXIN': 'insulin'}, axis=1, inplace=Tru(
```

In [260]:
```
# keep_insulin
```