Import Libraries

In [ ]:
```
#Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import glob
import re
from collections import defaultdict
```

In [ ]:
```
#To display all columns in Jupyter Notebooks
pd.set_option('display.max_columns', 500)
```

Retrieve data from Database

In [ ]:
```
#Import MongoClient
from pymongo import MongoClient

#Create a MongoClient to run the MongoDB instance
client = MongoClient('localhost', 27017)
```

In [ ]:
```
#Connect to existing datbase
db = client.NHANES_Q2
```

In [ ]:
```
db
```

Out[ ]:
```
Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=Fal
se, connect=True), 'NHANES_Q2')
```

In [ ]:
```
#Look at existing collections
col = db.list_collection_names()
col.sort()
col
```

Out[ ]:
```
['DI', 'DI_no_ohe']
```

In [ ]:
```
#Collections
di = db.DI
di_no = db.DI_no_ohe
```

In [ ]:
```
#Create dataframes from database collections
df_di = pd.DataFrame(list(di.find()))
df_di_no = pd.DataFrame(list(di_no.find()))
```

In [ ]:
```
#Variable Declarations
df = df_di.copy()
```

```
df_no = df_di_no.copy()
label = 'DIQ010'
labeldescr = 'Diabetes (0-No, 1-Yes)'
```

Reorder columns

Reorder OHE dataframe

In [ ]:
```
#Get a list of columns
cols = list(df)

#Move '_id' column to head of list using dex, pop and insert
cols.insert(0, cols.pop(cols.index('_id')))

#Move 'Year' column to back of list using index, pop and insert
cols.insert(len(df.columns)-1, cols.pop(cols.index('Year')))

#Move 'MEC18YR' column to back of list using index, pop and insert
cols.insert(len(df.columns)-1, cols.pop(cols.index('MEC18YR')))

#Move 'DRX18YR' column to back of list using index, pop and insert
cols.insert(len(df.columns)-1, cols.pop(cols.index('DRX18YR')))

#Move label column to back of list using index, pop and insert
cols.insert(len(df.columns)-1, cols.pop(cols.index(label)))
```

In [ ]:
```
#Reorder dataframe
df = df.loc[:, cols]
df.head()
```

Out[ ]:

	_id	RIDAGEYR	DMDHHSIZ	INDFMINC	DMDHREDU	DRD320GW	DRDTSODI	DRXTALCO
0	2.0	77.0	1.0	8.0	5.0	5.397605e-79	5710.03	5.397605e-79
1	5.0	49.0	3.0	11.0	4.0	1.298000e+03	3756.36	3.456000e+01
2	12.0	37.0	4.0	11.0	2.0	3.304000e+03	7511.18	5.397605e-79
3	15.0	38.0	2.0	8.0	5.0	2.478000e+03	3832.49	1.315000e+01
4	20.0	23.0	2.0	6.0	2.0	8.112500e+02	2746.43	5.397605e-79

Reorder non-OHE dataframe

In [ ]:
```
#Get a list of columns
cols = list(df_no)

#Move '_id' column to head of list using dex, pop and insert
cols.insert(0, cols.pop(cols.index('_id')))

#Move 'Year' column to back of list using index, pop and insert
cols.insert(len(df_no.columns)-1, cols.pop(cols.index('Year')))

#Move 'MEC18YR' column to back of list using index, pop and insert
cols.insert(len(df_no.columns)-1, cols.pop(cols.index('MEC18YR')))
```

```
#Move 'DRX18YR' column to back of list using index, pop and insert
cols.insert(len(df_no.columns)-1, cols.pop(cols.index('DRX18YR')))

#Move label column to back of list using index, pop and insert
cols.insert(len(df_no.columns)-1, cols.pop(cols.index(label)))
```

In [ ]:
```
#Reorder dataframe
df_no = df_no.loc[:, cols]
df_no.head()
```

Out[ ]:

	_id	RIAGENDR	RIDAGEYR	RIDRETH1	DMDBORN4	DMDCITZN	DMDHHSIZ	INDFMINC	DMD
0	2.0	1.0	77.0	3.0	1.0	1.0	1.0	8.0	
1	5.0	1.0	49.0	3.0	1.0	1.0	3.0	11.0	
2	12.0	1.0	37.0	3.0	1.0	1.0	4.0	11.0	
3	15.0	2.0	38.0	3.0	1.0	1.0	2.0	8.0	
4	20.0	2.0	23.0	1.0	1.0	1.0	2.0	6.0	

Exploratory Data Analysis

Analysis of Data

In [ ]:
```
import seaborn as sns
```

In [ ]:
```
df_no.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14121 entries, 0 to 14120
Data columns (total 58 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   _id       14121 non-null  float64
 1   RIAGENDR  14121 non-null  float64
 2   RIDAGEYR  14121 non-null  float64
 3   RIDRETH1  14121 non-null  float64
 4   DMDBORN4  14121 non-null  float64
 5   DMDCITZN  14121 non-null  float64
 6   DMDHHSIZ  14121 non-null  float64
 7   INDFMINC  14121 non-null  float64
 8   DMDHREDU  14121 non-null  float64
 9   ALQ101    14121 non-null  float64
 10  DRD320GW  14121 non-null  float64
 11  DRDTSODI  14121 non-null  float64
 12  DRXTALCO  14121 non-null  float64
 13  DRXTCAFF  14121 non-null  float64
 14  DRXTCALC  14121 non-null  float64
 15  DRXTCARB  14121 non-null  float64
 16  DRXTCHOL  14121 non-null  float64
 17  DRXTCOPP  14121 non-null  float64
 18  DRXTFIBE  14121 non-null  float64
```

```
 19  DRXTIRON  14121 non-null  float64
 20  DRXTKCAL  14121 non-null  float64
 21  DRXTMAGN  14121 non-null  float64
 22  DRXTPHOS  14121 non-null  float64
 23  DRXTPOTA  14121 non-null  float64
 24  DRXTPROT  14121 non-null  float64
 25  DRXTTFAT  14121 non-null  float64
 26  DRXTVARE  14121 non-null  float64
 27  DRXTVB1   14121 non-null  float64
 28  DRXTVB12  14121 non-null  float64
 29  DRXTVB2   14121 non-null  float64
 30  DRXTVB6   14121 non-null  float64
 31  DRXTVC    14121 non-null  float64
 32  DRXTZINC  14121 non-null  float64
 33  BPQ020    14121 non-null  float64
 34  BPXPULS   14121 non-null  float64
 35  BPXSY1    14121 non-null  float64
 36  BPXDI1    14121 non-null  float64
 37  BMXBMI    14121 non-null  float64
 38  LBXTC     14121 non-null  float64
 39  LBDHDL    14121 non-null  float64
 40  LBXTR     14121 non-null  float64
 41  LBDLDL    14121 non-null  float64
 42  PAQ635    14121 non-null  float64
 43  PAQ650    14121 non-null  float64
 44  PAQ665    14121 non-null  float64
 45  SMQ680    14121 non-null  float64
 46  SMAQUEX   14121 non-null  float64
 47  SMD410    14121 non-null  float64
 48  HID010    14121 non-null  float64
 49  HUQ010    14121 non-null  float64
 50  HUQ020    14121 non-null  float64
 51  HUQ030    14121 non-null  float64
 52  HUQ050    14121 non-null  float64
 53  HUQ070    14121 non-null  float64
 54  Year      14121 non-null  int64
 55  MEC18YR   14121 non-null  float64
 56  DRX18YR   14121 non-null  float64
 57  DIQ010    14121 non-null  float64
dtypes: float64(57), int64(1)
memory usage: 6.2 MB
```

Define Categorical & Numerical Features

In [ ]:

```
#Change columns to category
#Columns to remove:
#DRX18YR - 18 Year weight
#MEC18YR - 18 year Weight
#Year - Year of observation
#_id - Unique ID to identify individual

cat_cols = ['DMDBORN4',
            'DMDCITZN',
            'DMDHHSIZ',
            'DMDHREDU',
            'INDFMINC',
            'RIAGENDR',
            'RIDRETH1',
            'ALQ101',
```

```
             'DIQ010',
             'BPQ020',
             'BPXPULS',
             'PAQ635',
             'PAQ650',
             'PAQ665',
             'SMAQUEX',
             'SMQ680',
             'SMD410',
             'MCQ010',
             'MCQ160C',
             'MCQ220',
             'MCQ160K',
             'HID010',
             'HUQ010',
             'HUQ020',
             'HUQ030',
             'HUQ050',
             'HUQ070',
             'WHQ030',
             'WHQ040']

num_cols = ['RIDAGEYR',
             'DRD320GW',
              'DRDTSODI',
              'DRX18YR',
              'DRXTALCO',
              'DRXTCAFF',
              'DRXTCALC',
              'DRXTCARB',
              'DRXTCHOL',
              'DRXTCOPP',
              'DRXTFIBE',
              'DRXTIRON',
              'DRXTKCAL',
              'DRXTMAGN',
              'DRXTMFAT',
              'DRXTPFAT',
              'DRXTPHOS',
              'DRXTPOTA',
              'DRXTPROT',
              'DRXTSFAT',
              'DRXTTFAT',
              'DRXTVARE',
              'DRXTVB1',
              'DRXTVB12',
              'DRXTVB2',
              'DRXTVB6',
              'DRXTVC',
              'DRXTZINC',
              'BPXDI1',
              'BPXSY1',
              'LBDHDL',
              'LBDHDLSI',
              'LBDLDL',
              'LBDLDLSI',
              'LBDTCSI',
              'LBDTRSI',
              'LBXTC',
              'LBXTR',
```

```
            'BMXBMI',
            'BMXHT',
            'BMXWAIST',
            'BMXWT'
]


def recat_cols(df, col_names):
    for x in col_names:
        if x in cat_cols:
            df[x] = df[x].astype('category')
    return df

col_names = df_no.columns
df_no = recat_cols(df_no, col_names)
```

In [ ]:

```
#Recategorized columns
df_no.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14121 entries, 0 to 14120
Data columns (total 58 columns):
 #   Column    Non-Null Count  Dtype   
---  ------    --------------  -----   
 0   _id       14121 non-null  float64 
 1   RIAGENDR  14121 non-null  category
 2   RIDAGEYR  14121 non-null  float64 
 3   RIDRETH1  14121 non-null  category
 4   DMDBORN4  14121 non-null  category
 5   DMDCITZN  14121 non-null  category
 6   DMDHHSIZ  14121 non-null  category
 7   INDFMINC  14121 non-null  category
 8   DMDHREDU  14121 non-null  category
 9   ALQ101    14121 non-null  category
 10  DRD320GW  14121 non-null  float64 
 11  DRDTSODI  14121 non-null  float64 
 12  DRXTALCO  14121 non-null  float64 
 13  DRXTCAFF  14121 non-null  float64 
 14  DRXTCALC  14121 non-null  float64 
 15  DRXTCARB  14121 non-null  float64 
 16  DRXTCHOL  14121 non-null  float64 
 17  DRXTCOPP  14121 non-null  float64 
 18  DRXTFIBE  14121 non-null  float64 
 19  DRXTIRON  14121 non-null  float64 
 20  DRXTKCAL  14121 non-null  float64 
 21  DRXTMAGN  14121 non-null  float64 
 22  DRXTPHOS  14121 non-null  float64 
 23  DRXTPOTA  14121 non-null  float64 
 24  DRXTPROT  14121 non-null  float64 
 25  DRXTTFAT  14121 non-null  float64 
 26  DRXTVARE  14121 non-null  float64 
 27  DRXTVB1   14121 non-null  float64 
 28  DRXTVB12  14121 non-null  float64 
 29  DRXTVB2   14121 non-null  float64 
 30  DRXTVB6   14121 non-null  float64 
 31  DRXTVC    14121 non-null  float64 
 32  DRXTZINC  14121 non-null  float64 
 33  BPQ020    14121 non-null  category
```

```
 34  BPXPULS    14121 non-null  category
 35  BPXSY1     14121 non-null  float64
 36  BPXDI1     14121 non-null  float64
 37  BMXBMI     14121 non-null  float64
 38  LBXTC      14121 non-null  float64
 39  LBDHDL     14121 non-null  float64
 40  LBXTR      14121 non-null  float64
 41  LBDLDL     14121 non-null  float64
 42  PAQ635     14121 non-null  category
 43  PAQ650     14121 non-null  category
 44  PAQ665     14121 non-null  category
 45  SMQ680     14121 non-null  category
 46  SMAQUEX    14121 non-null  category
 47  SMD410     14121 non-null  category
 48  HID010     14121 non-null  category
 49  HUQ010     14121 non-null  category
 50  HUQ020     14121 non-null  category
 51  HUQ030     14121 non-null  category
 52  HUQ050     14121 non-null  category
 53  HUQ070     14121 non-null  category
 54  Year       14121 non-null  int64
 55  MEC18YR    14121 non-null  float64
 56  DRX18YR    14121 non-null  float64
 57  DIQ010     14121 non-null  category
dtypes: category(23), float64(34), int64(1)
memory usage: 4.1 MB
```

Plots of Features

Categorical Features

In [ ]:
```
#Get Categorical Features Only
df_cat = df_no.copy()
df_cat = df_cat.select_dtypes(include=['category'])
df_cat = df_cat.drop([label], axis=1)
```

In [ ]:
```
df_cat.head()
```

Out[ ]:

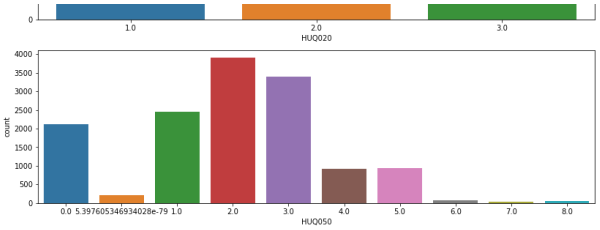
	RIAGENDR	RIDRETH1	DMDBORN4	DMDCITZN	DMDHHSIZ	INDFMINC	DMDHREDU	ALQ101
0	1.0	3.0	1.0	1.0	1.0	8.0	5.0	1.0
1	1.0	3.0	1.0	1.0	3.0	11.0	4.0	1.0
2	1.0	3.0	1.0	1.0	4.0	11.0	2.0	1.0
3	2.0	3.0	1.0	1.0	2.0	8.0	5.0	1.0
4	2.0	1.0	1.0	1.0	2.0	6.0	2.0	1.0

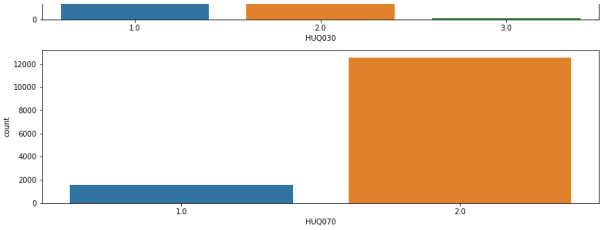
In [ ]:
```
df_cat.shape
```

Out[ ]:
```
(14121, 22)
```

In [ ]:
```
fig, ax = plt.subplots(11, 2, figsize=(30,50))
```

```
ax = ax.flatten()
for a, catplot in zip(ax, list(df_cat.columns)):
    sns.countplot(x=catplot, data=df_cat, ax=a)
plt.show()
```





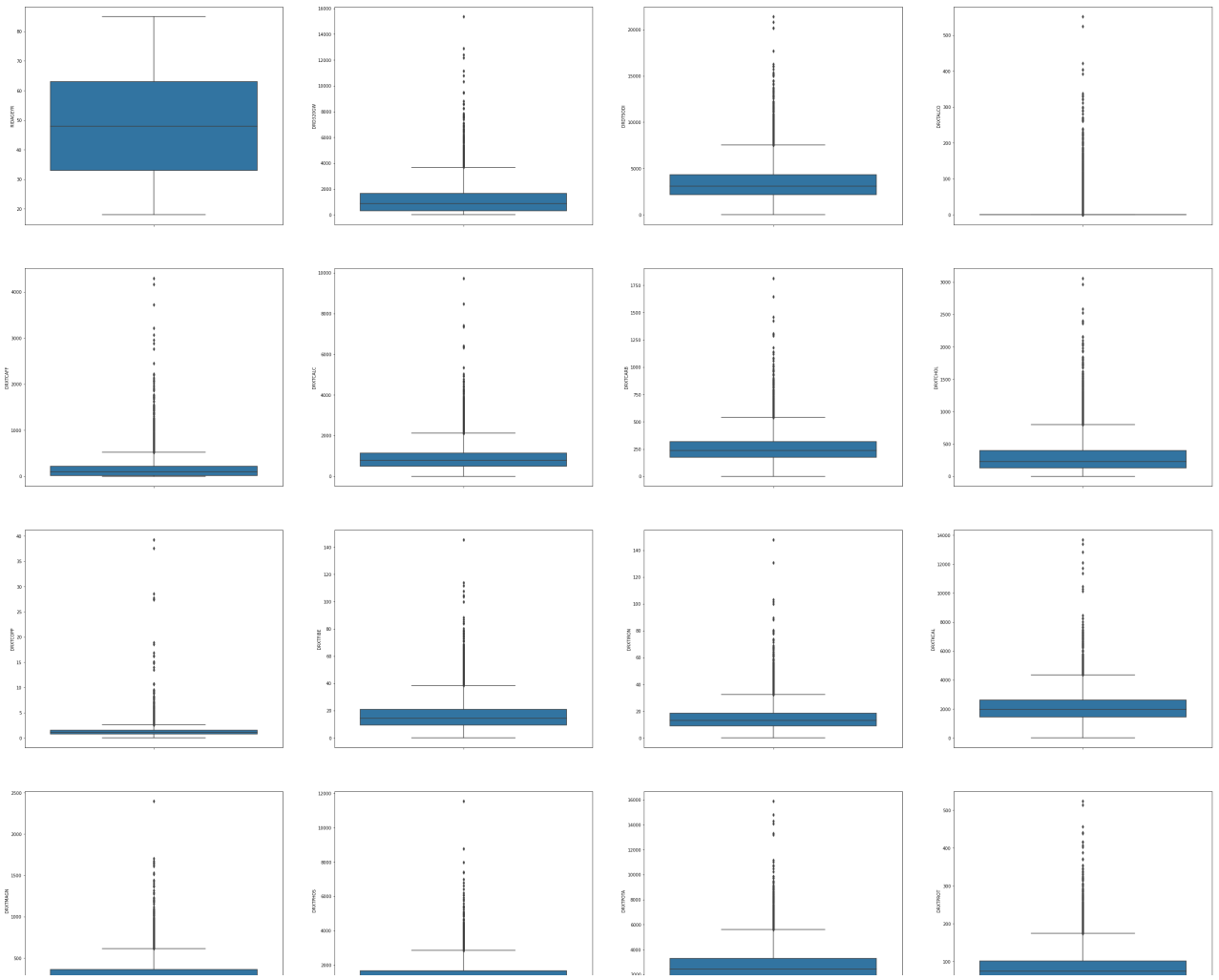
Numerical Features

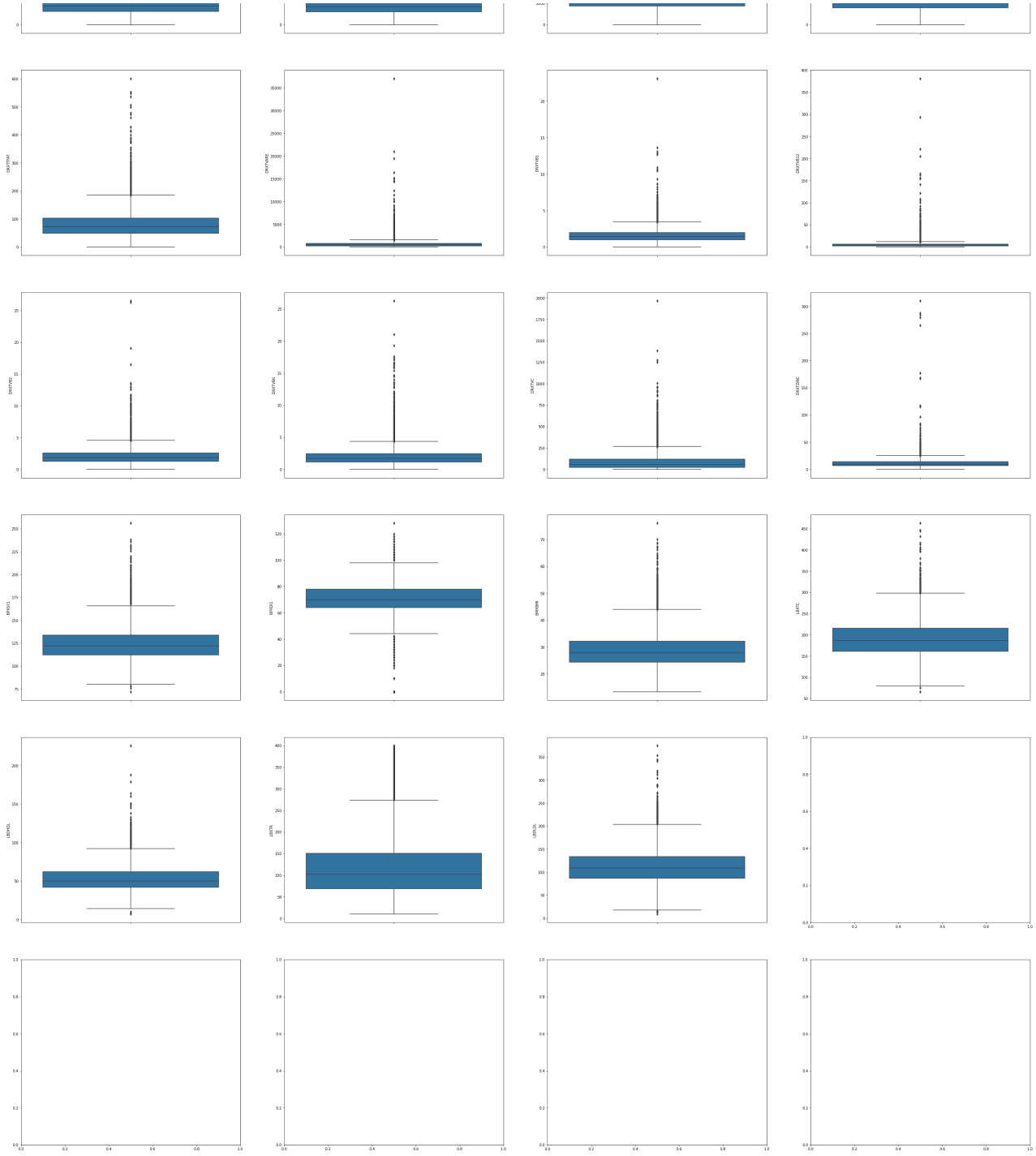
In [ ]:
```
#Get Numerical Features Only
df_num = df_no.copy()
df_num = df_num.select_dtypes(include=['float64'])
df_num = df_num.drop(['_id','MEC18YR','DRX18YR'], axis=1)
```

In [ ]:
```
df_num.shape
```

Out[ ]:
```
(14121, 31)
```

In [ ]:
```
fig, ax = plt.subplots(9, 4, figsize=(50,100))
ax = ax.flatten()
for a, bp in zip(ax, list(df_num.columns)):
    sns.boxplot(y=bp, data=df_num, ax=a)
plt.show()
```





Class Distribution

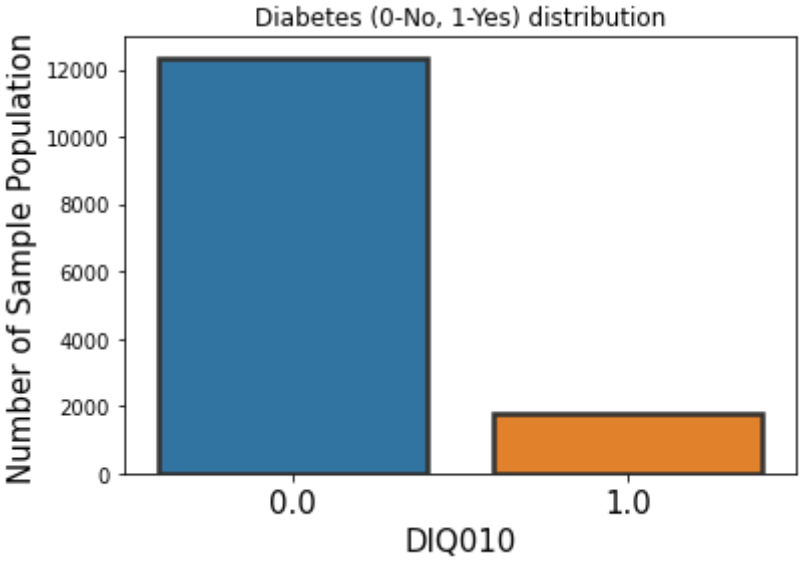
In [ ]:
```
label_count = df_no.groupby(label, as_index = False).count()
```

In [ ]:
```
#Count of class labels
class_names = list(label_count[label]) #Count the different labels
count = list(label_count['_id']) #Sum of different labels
count
```

Out[ ]:
```
[12347, 1774]
```

In [ ]:

```
#Bar Plot of Class Labels
index = np.arange(len(class_names))
sns.barplot(x=class_names, y=count, linewidth=2.5, errcolor=".2", edgecolor=".2"
plt.xlabel(label, fontsize=15)
plt.ylabel('Number of Sample Population', fontsize=15)
plt.xticks(index, class_names, fontsize=15)
plt.title(labeldescr+' distribution')
plt.show()
```



In [ ]:
```
#Find balance of labels
sum(df_no[label]==1)/(len(df_no[label]))
#0.87594 in majority class - No Diabetes
#0.12406 in minority class - Diabetes
```

Out[ ]:
```
0.12562849656539904
```

Class distribution as percentage of years

In [ ]:
```
label_perc = df_no.groupby(['Year',label], as_index = False).count()
```

In [ ]:
```
lp = label_perc[['Year', '_id',label]].copy()
```

In [ ]:
```
perc = lp.groupby(['Year', '_id']).agg({'_id': 'sum'})
perc = perc.rename(columns={ perc.columns[0]: "pcts" })
perc = perc.sort_values(by=['Year','pcts'], ascending=[True, False])
# Change: groupby Year and divide by sum
label_pcts = perc.groupby(level=0).apply(lambda x: x / float(x.sum()))
```

In [ ]:
```
#Get percentage values
lp['Percentages'] = label_pcts.values
```

In [ ]:
```
lp.head()
```

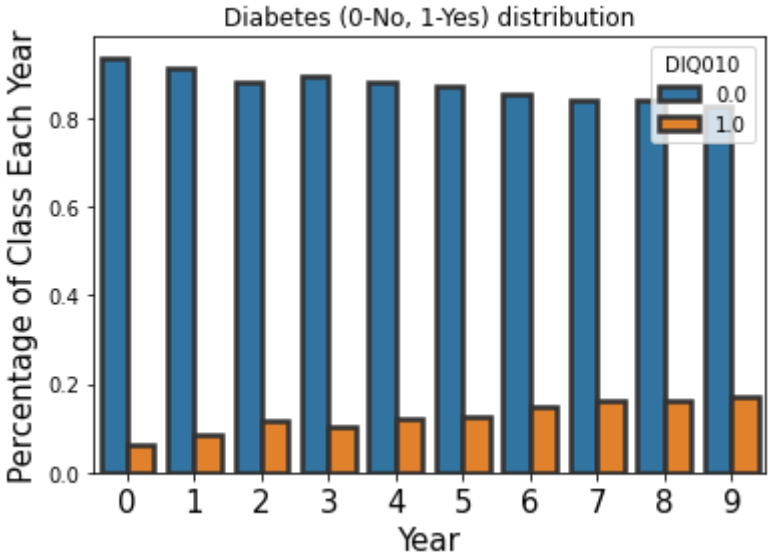
Out[ ]:

	Year	_id	DIQ010	Percentages
0	0	1178	0.0	0.937152
1	0	79	1.0	0.062848
2	1	1438	0.0	0.914758
3	1	134	1.0	0.085242
4	2	1255	0.0	0.883803

In [ ]:
```
year_names = list(lp['Year'])
class_names = list(lp[label])
```

In [ ]:
```
perc = list(lp['Percentages'])
```

In [ ]:
```
sns.barplot(x='Year', y='Percentages', data=lp, hue=label, linewidth=2.5, errcol
plt.xlabel('Year', fontsize=15)
plt.ylabel('Percentage of Class Each Year', fontsize=15)
plt.xticks(fontsize=15)
plt.title(labeldescr+' distribution')
plt.show()
```



Correlation of Numerical Features

Pearson's Coefficient

Correlation of Numerical features

In [ ]:
```
df_no.shape
```

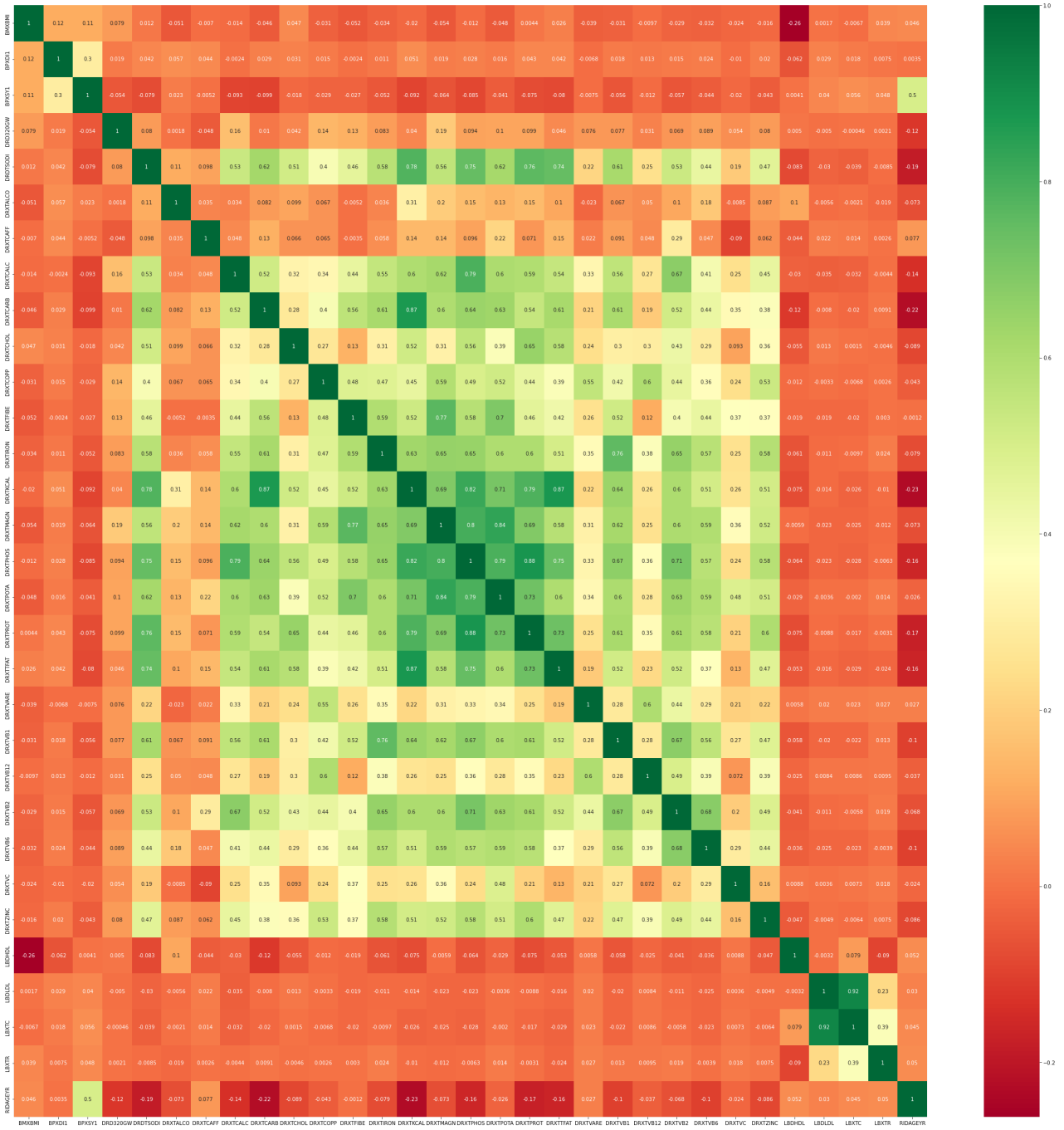
Out[ ]:
```
(14121, 58)
```

In [ ]:

```
#get Correlations of each features in dataset
x = ['_id', 'Year', 'MEC18YR', 'DRX18YR']

corrmat = df_no[df_no.columns.difference(x)].corr()
top_corr_features = corrmat.index
```

In [ ]:

```
plt.figure(figsize=(40,40))
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



In [ ]:

```
#cor_target = abs(corrmat[label])
#relevant_features = cor_target[cor_target>0.1]
#relevant_features
```

Create X and y

In [ ]:
```
#Drop variables
#Create X - drop id, label, and sample weight
X = df.drop(['_id', label], axis=1)

#Maintain id of X's
X_idx = df[['_id']]

#Create y - label
y = df[[label]]

#Create column to stratify based on year
year = df[['Year']]
```

In [ ]:
```
X.shape
```

Out[ ]:
```
(14121, 65)
```

Split Train, Test

In [ ]:
```
#Train, test, split
from sklearn.model_selection import train_test_split, cross_val_score
```

In [ ]:
```
#Stratify Train, Test based on Year Value
#80% Training, 20% Test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [ ]:
```
#y_true is test label
y_true = y_test[label]
```

In [ ]:
```
df['Year'].value_counts()
```

Out[ ]:
```
5    1584
1    1572
4    1535
7    1526
2    1420
6    1346
9    1326
8    1304
0    1257
3    1251
Name: Year, dtype: int64
```

Check if Stratified Sampling Worked

In [ ]:
```
#See if years are split evenly in training and test
```

```
xt = X_train['Year'].value_counts()
xte = X_test['Year'].value_counts()
```

In [ ]:
```
#Test is 20% of Training for each year
xte/(xte+xt)
```

Out[ ]:
```
5    0.200126
1    0.200382
4    0.200000
7    0.199869
2    0.200000
6    0.199851
9    0.199849
8    0.200153
0    0.200477
3    0.199840
Name: Year, dtype: float64
```

In [ ]:
```
#Test set label distribution
yt = y_true.value_counts()
yt
```

Out[ ]:
```
0    2464
1     361
Name: DIQ010, dtype: int64
```

In [ ]:
```
yt[0]/(yt[0]+yt[1])
```

Out[ ]:
```
0.872212389380531
```

In [ ]:
```
#Train set label distribution
ytr = y_train[label].value_counts()
ytr
```

Out[ ]:
```
0    9883
1    1413
Name: DIQ010, dtype: int64
```

In [ ]:
```
ytr[0]/(ytr[0]+ytr[1])
```

Out[ ]:
```
0.8749114730878187
```

Resample: Upsample and Downsample (Not Used)

We have imbalanced data, so we want to do upsampling and downsampling to see if it will improve the model

In [ ]:
```
from sklearn.utils import resample
```

In [ ]:
```
#Resample training data function
def resample_train(X_train, y_train, sample_type, label):
    #Concatenate our training data back together
    X_r = pd.concat([X_train, y_train], axis=1)

    #Separate minority and majority classes
    minority = X_r[X_r[label]==1]
    majority = X_r[X_r[label]==0]

    if (sample_type=='up'):
        #Upsample minority class
        resampled = resample(minority,
                             replace=True, #Sample with replacement
                             n_samples=len(majority), #Match number in majority cl
                             random_state=0) #Reproducible results
        notsampled=majority

    if (sample_type=='down'):
        #Downsample majority class
        resampled = resample(majority,
                             replace=False, #Sample without replacement
                             n_samples=len(minority), #Match minority n
                             random_state=0) #Reproducible results
        notsampled=minority

    #Combine upsample or downsample of majority and minority
    sampled = pd.concat([resampled, notsampled])
    y_train = sampled[[label]]
    X_train = sampled.drop([label], axis=1)
    return X_train, y_train
```

Sklearn Upsampled

In [ ]:
```
X_train_u, y_train_u = resample_train(X_train, y_train, 'up', label)
```

In [ ]:
```
y_train_u[y_train_u[label]==1].shape
```

Out[ ]:
```
(9883, 1)
```

In [ ]:
```
y_train_u[y_train_u[label]==0].shape
```

Out[ ]:
```
(9883, 1)
```

Sklearn Downsampled

In [ ]:
```
X_train_d, y_train_d = resample_train(X_train, y_train, 'down', label)
```

In [ ]:
```
y_train_d[y_train_d[label]==1].shape
```

Out[ ]:
```
(1413, 1)
```

In [ ]:

```
y_train_d[y_train_d[label]==0].shape
```

Out[ ]:
```
(1413, 1)
```

Imbalance Learn - SMOTE (Up) and TomekLinks (Down)

Upsampling - SMOTE

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

In [ ]:
```
sm = SMOTE(random_state=1)
```

In [ ]:
```
X_sm, y_sm = sm.fit_resample(X_train, y_train.values.ravel())
```

In [ ]:
```
#Before upsampling - Majority
len(y_train[y_train[label]==0])
```

Out[ ]:
```
9883
```

In [ ]:
```
#Before upsampling - Minority
len(y_train[y_train[label]==1])
```

Out[ ]:
```
1413
```

In [ ]:
```
#After upsampling, majority and minority match
sum(y_sm==0)
```

Out[ ]:
```
9883
```

In [ ]:
```
sum(y_sm==1)
```

Out[ ]:
```
9883
```

Downsampling - Tomeklinks

In [ ]:
```
from imblearn.under_sampling import TomekLinks
```

In [ ]:
```
#tLinks = TomekLinks(random_state = 0, return_indices = True)
tLinks = TomekLinks()
```

In [ ]:
```
#X_tl, y_tl, id_tl = tLinks.fit_sample(X_train, y_train.values.ravel())
X_tl, y_tl = tLinks.fit_resample(X_train, y_train.values.ravel())
```

In [ ]:
```
#Before downsampling - Majority
len(y_train[y_train[label]==0])
```

Out[ ]: 9883

In [ ]:
```
#Before downsampling - Minority
len(y_train[y_train[label]==1])
```

Out[ ]: 1413

In [ ]:
```
#After downsampling, majority decreases in neighbors
sum(y_tl==0)
```

Out[ ]: 9403

In [ ]:
```
sum(y_tl==1)
```

Out[ ]: 1413

Clean Columns for Training Data

Regular Training and Test

In [ ]:
```
X_cols = list(X_train.columns)

#Get MEC18YR from training & test data
Xtr_sw = X_train[['MEC18YR']]
Xts_sw = X_test[['MEC18YR']]

#Get DRX18YR from training & test data
Xtr_dsw = X_train[['DRX18YR']]
Xts_dsw = X_test[['DRX18YR']]

#Get Year from training & test data
Xtr_yr = X_train[['Year']]
Xts_yr = X_test[['Year']]

#Drop columns for training
X_train = X_train.drop(['MEC18YR','DRX18YR','Year'], axis=1)
X_test = X_test.drop(['MEC18YR','DRX18YR','Year'], axis=1)
```

SMOTE

In [ ]:
```
X_sm = pd.DataFrame(X_sm, columns = X_cols)
```

In [ ]:
```
X_sm.head()
```

Out[ ]:

	RIDAGEYR	DMDHHSIZ	INDFMINC	DMDHREDU	DRD320GW	DRDTSODI	DRXTALCO	L
0	38.0	3.0	9.0	4.0	8.700000e+02	1446.00	2.070000e+01	2.88
1	55.0	1.0	6.0	4.0	1.184200e+03	2066.00	3.730000e+01	3.03
2	24.0	4.0	3.0	1.0	5.397605e-79	1148.58	5.397605e-79	1.24
3	71.0	5.0	2.0	2.0	3.110600e+02	3920.00	4.500000e+00	1.20
4	55.0	5.0	11.0	4.0	4.129200e+03	8838.00	5.397605e-79	5.3

In [ ]:
```
#Get MEC18YR from training
Xsm_sw = X_sm[['MEC18YR']]

#Get DRX18YR from training
Xsm_dsw = X_sm[['DRX18YR']]

#Get Year from training
Xsm_yr = X_sm[['Year']]

#Drop columns for training
X_sm = X_sm.drop(['MEC18YR','DRX18YR','Year'], axis=1)
```

TomekLinks

In [ ]:
```
X_tl = pd.DataFrame(X_tl, columns = X_cols)
```

In [ ]:
```
#Get MEC18YR from training
Xtl_sw = X_tl[['MEC18YR']]

#Get DRX18YR from training
Xtl_dsw = X_tl[['DRX18YR']]

#Get Year from training
Xtl_yr = X_tl[['Year']]

#Drop sample weight
X_tl = X_tl.drop(['MEC18YR','DRX18YR','Year'], axis=1)
```

Feature Selection

In [ ]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

Drop MEC sample weight for X for feature selection

In [ ]:
```
Xd = df_no.drop(['_id',label,'MEC18YR','DRX18YR','Year'], axis=1)
yd = df_no[label]
```

Feature Selection

In [ ]:
```
#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=30)
fit = bestfeatures.fit(Xd,yd)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(Xd.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
       Specs         Score
19  DRXTKCAL  71856.306275
22  DRXTPOTA  22334.886274
10  DRDTSODI  19060.586205
13  DRXTCALC  18891.120855
21  DRXTPHOS  16480.268606
14  DRXTCARB  13080.767992
25  DRXTVARE   9550.342221
1   RIDAGEYR   6416.967430
11  DRXTALCO   3760.483344
30    DRXTVC   3173.602016
```

OHE Feature Selection

In [ ]:
```
Xd = df.drop(['_id',label,'MEC18YR','DRX18YR','Year'], axis=1)
yd = df[label]
```

In [ ]:
```
#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(Xd,yd)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(Xd.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
       Specs         Score
14  DRXTKCAL  71856.306275
17  DRXTPOTA  22334.886274
5   DRDTSODI  19060.586205
8   DRXTCALC  18891.120855
16  DRXTPHOS  16480.268606
9   DRXTCARB  13080.767992
20  DRXTVARE   9550.342221
0   RIDAGEYR   6416.967430
6   DRXTALCO   3760.483344
25    DRXTVC   3173.602016
```

Data Analysis: Random Forest and XGBoost

Import Machine Learning Libraries

In [ ]:
```
#Models
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
from xgboost import plot_importance

#Grid Search
from sklearn.model_selection import GridSearchCV

#Evaluation Metrics
from sklearn.metrics import accuracy_score, confusion_matrix, precision_recall_f
from sklearn.metrics import mean_squared_error, classification_report, f1_score
```

```
/Users/ashwinithirukkonda/opt/anaconda3/envs/sklearn-env/lib/python3.10/site-pac
kages/xgboost/compat.py:36: FutureWarning: pandas.Int64Index is deprecated and w
ill be removed from pandas in a future version. Use pandas.Index with the approp
riate dtype instead.
  from pandas import MultiIndex, Int64Index
```

Random Forest

GridSearch: Define parameters and run

Regular Training

In [ ]:
```
#clf = RandomForestClassifier(random_state=1)

#Define Parameter Grid for GridSearchCV
# param_grid = {
#     'n_estimators' : [200],
#     'max_features' : ['auto'],
#     'max_depth' : [4, 6, 8, 15],
#     'criterion' : ['gini'],
#     'class_weight' : [{0:0.1, 1:0.9},'balanced']
# }
```

In [ ]:
```
#cv_rfc = GridSearchCV(estimator=clf, param_grid=param_grid, scoring='f1', verbo
```

In [ ]:
```
#cv_rfc.fit(X_train, y_train.values.ravel())
```

In [ ]:
```
#cv_rfc.best_params_
# {'class_weight': 'balanced',
#  'criterion': 'gini',
#  'max_depth': 8,
#  'max_features': 'auto',
#  'n_estimators': 200}
```

In [ ]:
```
#cv_rfc.best_estimator_
# RandomForestClassifier(bootstrap=True, class_weight={0: 0.1, 1: 0.9},
#             criterion='gini', max_depth=6, max_features='auto',
#             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,
#             n_estimators=200, n_jobs=None, oob_score=False, random_state=1,
#             verbose=0, warm_start=False)
```

In [ ]:
```
#cv_rfc.best_score_
#0.46175239302855975
```

Upsampling: SMOTE

In [ ]:
```
#clf_u = RandomForestClassifier(random_state=1)
```

In [ ]:
```
#cv_rfc_u = GridSearchCV(estimator=clf_u, param_grid=param_grid, scoring='f1', v
```

In [ ]:
```
#cv_rfc_u.fit(X_sm, y_sm)
```

In [ ]:
```
#cv_rfc_u.best_params_
# {'class_weight': 'balanced',
#  'criterion': 'gini',
#  'max_depth': 15,
#  'max_features': 'auto',
#  'n_estimators': 200}
```

In [ ]:
```
#cv_rfc_u.best_estimator_
# RandomForestClassifier(bootstrap=True, class_weight='balanced',
#             criterion='gini', max_depth=15, max_features='auto',
#             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,
#             n_estimators=200, n_jobs=None, oob_score=False, random_state=1,
#             verbose=0, warm_start=False)
```

In [ ]:
```
#cv_rfc_u.best_score_
#0.9025614637840615
```

Downsampling: TomekLinks

In [ ]:
```
#clf_d = RandomForestClassifier(random_state=1)
```

In [ ]:
```
#cv_rfc_d = GridSearchCV(estimator= clf_d, param_grid=param_grid, scoring='f1',
```

In [ ]:
```
#cv_rfc_d.fit(X_tl, y_tl)
```

In [ ]:
```
#cv_rfc_d.best_params_
```

```
# {'class_weight': 'balanced',
#  'criterion': 'gini',
#  'max_depth': 8,
#  'max_features': 'auto',
#  'n_estimators': 200}
```

In [ ]:
```
#cv_rfc_d.best_estimator_
# RandomForestClassifier(bootstrap=True, class_weight='balanced',
#             criterion='gini', max_depth=8, max_features='auto',
#             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,
#             n_estimators=200, n_jobs=None, oob_score=False, random_state=1,
#             verbose=0, warm_start=False)
```

In [ ]:
```
#cv_rfc_d.best_score_
#0.4753574582260989
```

Define Classifier with Best Parameters and Fit

Regular Training

In [ ]:
```
'''
clf_r = RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=8, max_features='auto',
            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,
            n_estimators=200, n_jobs=None, oob_score=False, random_state=1,
            verbose=0, warm_start=False)
'''

clf_r = RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=8, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0, min_samples_leaf=1,
            n_estimators=200, n_jobs=None, oob_score=False, random_state=1,
            verbose=0, warm_start=False)
```

In [ ]:
```
clf_r.fit(X_train, y_train.values.ravel())
```

Out[ ]:
```
RandomForestClassifier(class_weight='balanced', max_depth=8, n_estimators=200,
                       random_state=1)
```

Upsampling: SMOTE

In [ ]:
```
'''
clf_s = RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=15, max_features='auto',
            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,
            n_estimators=200, n_jobs=None, oob_score=False, random_state=1,
```

```
            verbose=0, warm_start=False)
'''

clf_s = RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=15, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=200, n_jobs=None, oob_score=False, random_state=1,
            verbose=0, warm_start=False)
```

In [ ]:
```
clf_s.fit(X_sm, y_sm)
```

Out[ ]:
```
RandomForestClassifier(class_weight='balanced', max_depth=15, n_estimators=200,
                       random_state=1)
```

Downsampling: TomekLinks

In [ ]:
```
clf_t = RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=8, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=200, n_jobs=None, oob_score=False, random_state=1,
            verbose=0, warm_start=False)
```

In [ ]:
```
clf_t.fit(X_tl, y_tl)
```

Out[ ]:
```
RandomForestClassifier(class_weight='balanced', max_depth=8, n_estimators=200,
                       random_state=1)
```

Feature Importance from Random Forest

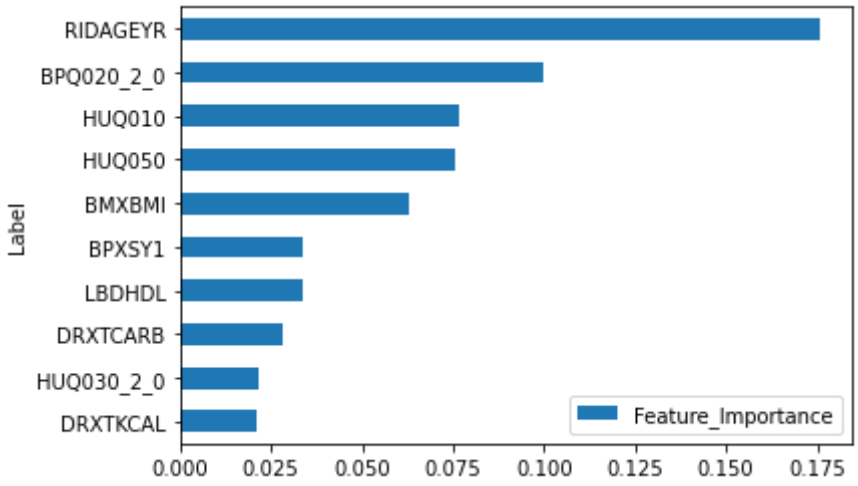
Regular Training

In [ ]:
```
rf_output = pd.DataFrame(clf_r.feature_importances_) #Get importance of features
rf_output['Label'] = X_train.columns
rf_output.columns = ['Feature_Importance', 'Label']
```

In [ ]:
```
rf_oo_r = rf_output.nlargest(10, 'Feature_Importance')
```

In [ ]:
```
rf_oo_r_s = rf_oo_r.sort_values(by=['Feature_Importance'])
```

In [ ]:
```
ax_r = rf_oo_r_s.plot.barh(y='Feature_Importance', x='Label')
```



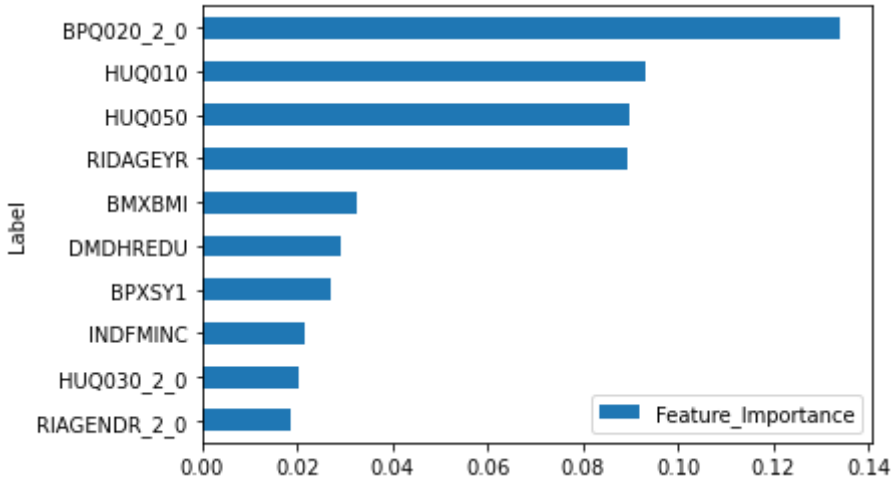
Upsampling: SMOTE

In [ ]:
```
rf_output = pd.DataFrame(clf_s.feature_importances_)
rf_output['Label'] = X_sm.columns
rf_output.columns = ['Feature_Importance', 'Label']
```

In [ ]:
```
rf_oo_u = rf_output.nlargest(10, 'Feature_Importance')
```

In [ ]:
```
rf_oo_u_s = rf_oo_u.sort_values(by=['Feature_Importance'])
```

In [ ]:
```
ax_u = rf_oo_u_s.plot.barh(y='Feature_Importance', x='Label')
```



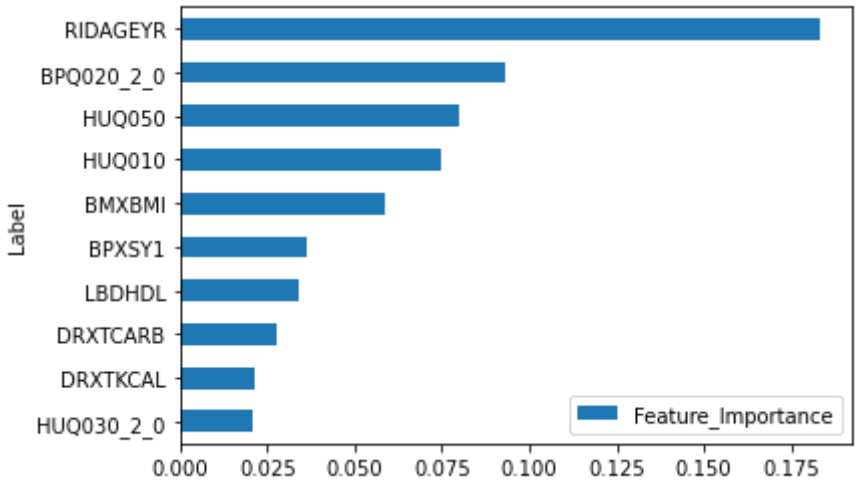
Downsampling: TomekLinks

In [ ]:
```
rf_output = pd.DataFrame(clf_t.feature_importances_)
rf_output['Label'] = X_tl.columns
rf_output.columns = ['Feature_Importance', 'Label']
```

In [ ]:
```
rf_oo_d = rf_output.nlargest(10, 'Feature_Importance')
```

In [ ]:
```
rf_oo_d_s = rf_oo_d.sort_values(by=['Feature_Importance'])
```

In [ ]:
```
ax_d = rf_oo_d_s.plot.barh(y='Feature_Importance', x='Label')
```



Model Evaluation

Confusion Matrix & Classification Reports

Regular Training

In [ ]:
```
pred_r = clf_r.predict(X_test)
```

In [ ]:
```
print('Confusion Matrix:')
pd.crosstab(y_true, pred_r, rownames=['True'], colnames=['Predicted'], margins=T
```

Out[ ]:
Confusion Matrix:

Predicted	0	1	All
True			
0	2007	457	2464
1	119	242	361
All	2126	699	2825

In [ ]:
```
r_output = '{} \n Accuracy: {}'
r = r_output.format(classification_report(y_true, pred_r, labels=[0,1]), accurac
print(r)
```

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87      2464
           1       0.35      0.67      0.46       361
```

```
    accuracy                           0.80      2825
   macro avg       0.65      0.74      0.67      2825
weighted avg       0.87      0.80      0.82      2825

 Accuracy: 0.7961061946902654
```

Upsampling: SMOTE

In [ ]:
```
pred_s = clf_s.predict(X_test)
```

In [ ]:
```
print('Confusion Matrix:')
pd.crosstab(y_true, pred_s, rownames=['True'], colnames=['Predicted'], margins=T
```

```
Confusion Matrix:
```

Out[ ]:

Predicted	0	1	All
True			
0	2283	181	2464
1	217	144	361
All	2500	325	2825

In [ ]:
```
u_output = '{} \n Accuracy: {}'
u = u_output.format(classification_report(y_true, pred_s, labels=[0,1]), accurac
print(u)
```

```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92      2464
           1       0.44      0.40      0.42       361

    accuracy                           0.86      2825
   macro avg       0.68      0.66      0.67      2825
weighted avg       0.85      0.86      0.86      2825

 Accuracy: 0.8591150442477876
```

Downsampling: TomekLinks

In [ ]:
```
pred_t = clf_t.predict(X_test)
```

In [ ]:
```
print('Confusion Matrix:')
pd.crosstab(y_true, pred_t, rownames=['True'], colnames=['Predicted'], margins=T
```

```
Confusion Matrix:
```

Out[ ]:

Predicted	0	1	All
True			
0	1995	469	2464
1	120	241	361

Predicted	0	1	All
True			
All	2115	710	2825

In [ ]:
```
d_output = '{} \n Accuracy: {}'
d = d_output.format(classification_report(y_true, pred_t, labels=[0,1]), accurac
print(d)
```

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87      2464
           1       0.34      0.67      0.45       361

    accuracy                           0.79      2825
   macro avg       0.64      0.74      0.66      2825
weighted avg       0.87      0.79      0.82      2825

 Accuracy: 0.791504424778761
```

ROC AUC Score

Regular Training

In [ ]:
```
prob_y_2_r = clf_r.predict_proba(X_test)
prob_y_2_r = [p[1] for p in prob_y_2_r]
print('ROC AUC Score: ', roc_auc_score(y_true, prob_y_2_r))
```

```
ROC AUC Score:  0.8425032377594704
```

Upsampling: SMOTE

In [ ]:
```
prob_y_2_s = clf_s.predict_proba(X_test)
prob_y_2_s = [p[1] for p in prob_y_2_s]
print('ROC AUC Score: ', roc_auc_score(y_true, prob_y_2_s))
```

```
ROC AUC Score:  0.8316803521962801
```

Downsampling: TomekLinks

In [ ]:
```
prob_y_2_t = clf_t.predict_proba(X_test)
prob_y_2_t = [p[1] for p in prob_y_2_t]
print('ROC AUC Score: ', roc_auc_score(y_true, prob_y_2_t))
```

```
ROC AUC Score:  0.8404953771989783
```

Adjust Predictions Based on Probability Threshold

In [ ]:
```
#Import Metrics
from sklearn.metrics import recall_score, precision_score
```

In [ ]:
```
#Custom cutoff probability
def cutoff_predict(clf, X, cutoff):
    return (clf.predict_proba(X)[:,1]> cutoff).astype(int)

#Custom scoring function
def custom_f1(cutoff):
    def f1_cutoff(clf, X, y):
        ypred = cutoff_predict(clf, X, cutoff)
        return f1_score(y, ypred)
    return f1_cutoff
```

Regular Training

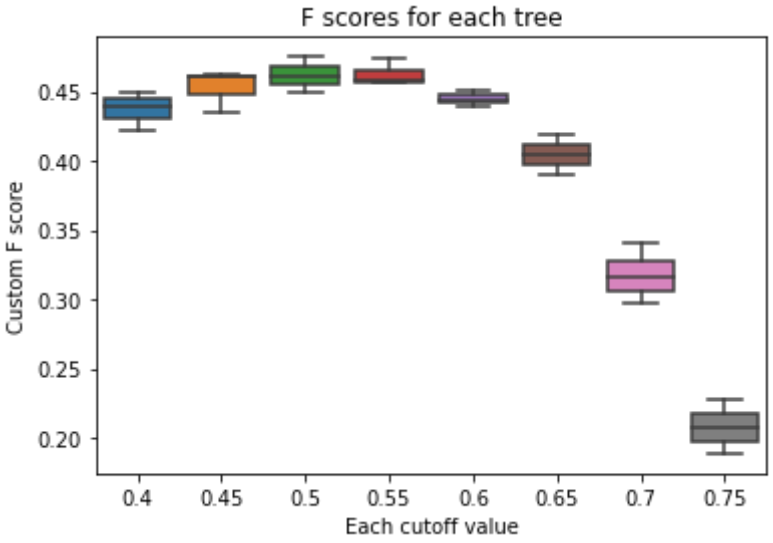
In [ ]:
```
#scores_r = []
cutoffs_r = np.arange(0.4, 0.8, 0.05)

# for cutoff in cutoffs_r:
#     validated = cross_val_score(clf_r, X_train, y_train.values.ravel(), cv = 3
#     scores_r.append(validated)
```

In [ ]:
```
# scores_r output
# [array([0.42226047, 0.44942529, 0.4394111 ]),
#  array([0.43506922, 0.46235139, 0.46065259]),
#  array([0.44885496, 0.47509579, 0.46130952]),
#  array([0.45749343, 0.45614035, 0.47445887]),
#  array([0.45061728, 0.43947101, 0.444     ]),
#  array([0.40399002, 0.38970588, 0.41893491]),
#  array([0.31578947, 0.29714286, 0.34054834]),
#  array([0.20701169, 0.22734761, 0.18867925])]
```

In [ ]:
```
scores_r_saved = [[0.42226047, 0.44942529, 0.4394111 ],
 [0.43506922, 0.46235139, 0.46065259],
 [0.44885496, 0.47509579, 0.46130952],
 [0.45749343, 0.45614035, 0.47445887],
 [0.45061728, 0.43947101, 0.444     ],
 [0.40399002, 0.38970588, 0.41893491],
 [0.31578947, 0.29714286, 0.34054834],
 [0.20701169, 0.22734761, 0.18867925]]
```

In [ ]:
```
#Melt and make scores into dataframe
scores_rdf = pd.DataFrame(scores_r_saved.copy())
#Rename columns
scores_rdf['Cutoff'] = pd.DataFrame(cutoffs_r).round(2)
#Create cutoff column values
scores_rdf = pd.melt(scores_rdf, id_vars='Cutoff')
scores_rdf.columns = ['Cutoff', 'CV', 'F Score']
```

In [ ]:
```
#Boxplot of 3-CV to determine F Score
sns.boxplot(x='Cutoff', y='F Score', data = scores_rdf)
plt.title('F scores for each tree')
plt.xlabel('Each cutoff value')
plt.ylabel('Custom F score')
plt.show()
```



Looks like the optimal cutoff is 0.50

In [ ]:
```
cutoff_r = 0.50
```

Upsampling: SMOTE

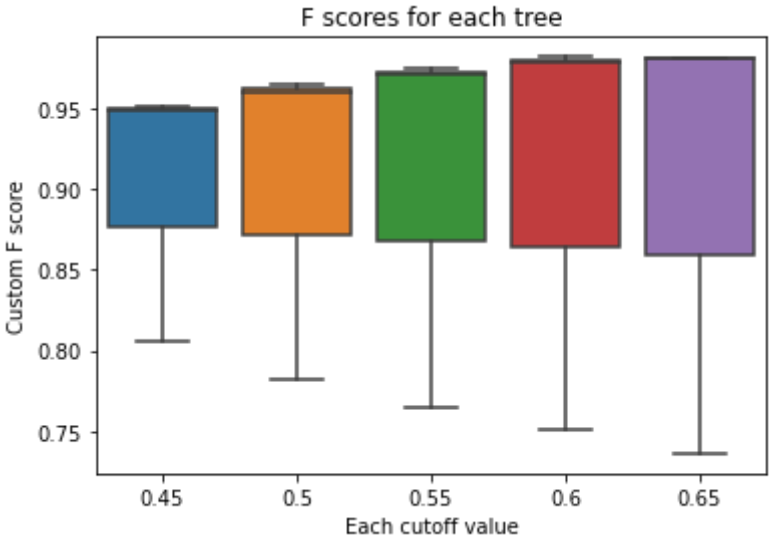
In [ ]:
```
#scores_u = []
cutoffs_u = np.arange(0.45, 0.7, 0.05)

# for cutoff in cutoffs_u:
#     validated = cross_val_score(clf_s, X_sm, y_sm, cv = 3, scoring = custom_f1
#     scores_u.append(validated)
```

In [ ]:
```
# scores_u output
# [array([0.80553269, 0.94864211, 0.95104518]),
#  array([0.78202095, 0.96023965, 0.96544158]),
#  array([0.76506128, 0.97131826, 0.97446572]),
#  array([0.75039411, 0.97880056, 0.98204769]),
#  array([0.73613087, 0.9815415 , 0.98143687])]
```

In [ ]:
```
scores_u_saved = [[0.80553269, 0.94864211, 0.95104518],
 [0.78202095, 0.96023965, 0.96544158],
 [0.76506128, 0.97131826, 0.97446572],
 [0.75039411, 0.97880056, 0.98204769],
 [0.73613087, 0.9815415 , 0.98143687]]
```

In [ ]:
```
#Melt and make scores into dataframe
scores_udf = pd.DataFrame(scores_u_saved.copy())
#Rename columns
scores_udf['Cutoff'] = pd.DataFrame(cutoffs_u).round(2)
#Create cutoff column values
scores_udf = pd.melt(scores_udf, id_vars='Cutoff')
scores_udf.columns = ['Cutoff', 'CV', 'F Score']
```

In [ ]:
```
#Boxplot of 3-CV to determine F Score
sns.boxplot(x='Cutoff', y='F Score', data = scores_udf)
plt.title('F scores for each tree')
plt.xlabel('Each cutoff value')
plt.ylabel('Custom F score')
plt.show()
```



In [ ]:
```
scores_udf.groupby('Cutoff').mean()
```

Out[ ]:

Cutoff	F Score
0.45	0.901740
0.50	0.902567
0.55	0.903615
0.60	0.903747
0.65	0.899703

Looks like the optimal cutoff is 0.60

In [ ]:
```
cutoff_u = 0.60
```

Downsampling: TomekLinks

In [ ]:
```
# scores_d = []
cutoffs_d = np.arange(0.4, 0.75, 0.05)

# for cutoff in cutoffs_d:
#     validated = cross_val_score(clf_t, X_tl, y_tl, cv = 3, scoring = custom_f1
#     scores_d.append(validated)
```

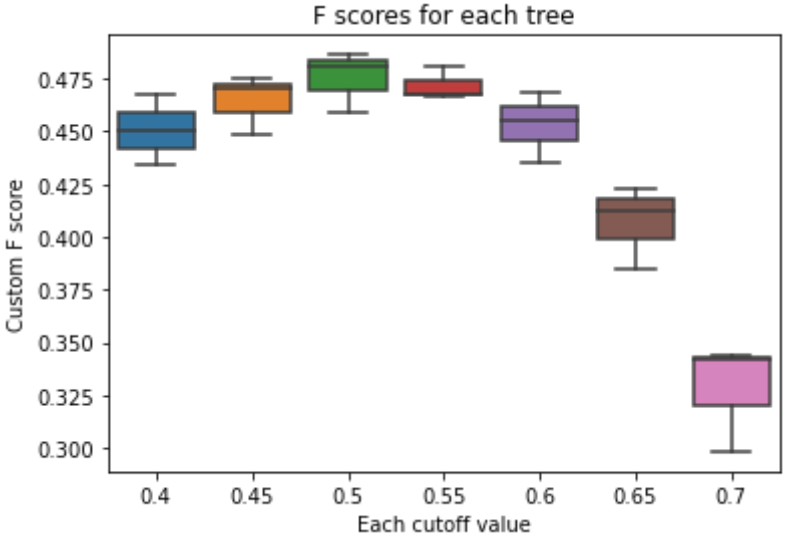
In [ ]:
```
# scores_d output
# [array([0.43390634, 0.46798324, 0.45026178]),
```

```
#  array([0.44848485, 0.47482993, 0.47028086]),
#  array([0.45927075, 0.48631744, 0.48048967]),
#  array([0.46778464, 0.46684832, 0.48070175]),
#  array([0.45556691, 0.43505808, 0.46843177]),
#  array([0.41277641, 0.38509317, 0.42316785]),
#  array([0.34383954, 0.29824561, 0.34188034])]
```

In [ ]:
```
scores_d_saved = [[0.43390634, 0.46798324, 0.45026178],
 [0.44848485, 0.47482993, 0.47028086],
 [0.45927075, 0.48631744, 0.48048967],
 [0.46778464, 0.46684832, 0.48070175],
 [0.45556691, 0.43505808, 0.46843177],
 [0.41277641, 0.38509317, 0.42316785],
 [0.34383954, 0.29824561, 0.34188034]]
```

In [ ]:
```
#Melt and make scores into dataframe
scores_ddf = pd.DataFrame(scores_d_saved.copy())
#Rename columns
scores_ddf['Cutoff'] = pd.DataFrame(cutoffs_d).round(2)
#Create cutoff column values
scores_ddf = pd.melt(scores_ddf, id_vars='Cutoff')
scores_ddf.columns = ['Cutoff', 'CV', 'F Score']
```

In [ ]:
```
#Boxplot of 3-CV to determine F Score
sns.boxplot(x='Cutoff', y='F Score', data = scores_ddf)
plt.title('F scores for each tree')
plt.xlabel('Each cutoff value')
plt.ylabel('Custom F score')
plt.show()
```



In [ ]:
```
scores_ddf.groupby('Cutoff').mean()
```

Out[ ]:

Cutoff	F Score
0.40	0.450717

Cutoff	F Score
0.45	0.464532
0.50	0.475359
0.55	0.471778
0.60	0.453019
0.65	0.407012
0.70	0.327988

Looks like the optimal is 0.50

In [ ]:
```
cutoff_d = 0.50
```

Model Evaluation based on Optimal Probability Cutoff

Regular Training

In [ ]:
```
pred_rc = np.where(pd.DataFrame(prob_y_2_r)>cutoff_r, 1, 0).flatten()
```

In [ ]:
```
cutoff_r
```

Out[ ]:
```
0.5
```

In [ ]:
```
print('Confusion Matrix:')
pd.crosstab(y_true, pred_rc, rownames=['True'], colnames=['Predicted'], margins=
```

Out[ ]:
```
Confusion Matrix:
```

Predicted	0	1	All
True			
0	2007	457	2464
1	119	242	361
All	2126	699	2825

In [ ]:
```
rc_output = '{} \n Accuracy: {}'
rc = rc_output.format(classification_report(y_true, pred_rc, labels=[0,1]), accu
print(rc)
```

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87      2464
           1       0.35      0.67      0.46       361

    accuracy                           0.80      2825
```

```
   macro avg       0.65      0.74      0.67      2825
weighted avg       0.87      0.80      0.82      2825

 Accuracy: 0.7961061946902654
```

In [ ]:
```
#Compare with default cutoff 0.5
print(r)
```

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87      2464
           1       0.35      0.67      0.46       361

    accuracy                           0.80      2825
   macro avg       0.65      0.74      0.67      2825
weighted avg       0.87      0.80      0.82      2825

 Accuracy: 0.7961061946902654
```

Upsampling: SMOTE

In [ ]:
```
pred_uc = np.where(pd.DataFrame(prob_y_2_s)>cutoff_u, 1, 0).flatten()
```

In [ ]:
```
cutoff_u
```

Out[ ]:
```
0.6
```

In [ ]:
```
print('Confusion Matrix:')
pd.crosstab(y_true, pred_uc, rownames=['True'], colnames=['Predicted'], margins=
```

```
Confusion Matrix:
```

Out[ ]:

Predicted	0	1	All
True			
0	2369	95	2464
1	288	73	361
All	2657	168	2825

In [ ]:
```
uc_output = '{} \n Accuracy: {}'
uc = uc_output.format(classification_report(y_true, pred_uc, labels=[0,1]), accu
print(uc)
```

```
              precision    recall  f1-score   support

           0       0.89      0.96      0.93      2464
           1       0.43      0.20      0.28       361

    accuracy                           0.86      2825
   macro avg       0.66      0.58      0.60      2825
weighted avg       0.83      0.86      0.84      2825

 Accuracy: 0.8644247787610619
```

In [ ]:
```
#Compare with default cutoff 0.5
print(u)
```

```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92      2464
           1       0.44      0.40      0.42       361

    accuracy                           0.86      2825
   macro avg       0.68      0.66      0.67      2825
weighted avg       0.85      0.86      0.86      2825

 Accuracy: 0.8591150442477876
```

Downsampling: TomekLinks

In [ ]:
```
pred_dc = np.where(pd.DataFrame(prob_y_2_t)>cutoff_d, 1, 0).flatten()
```

In [ ]:
```
cutoff_d
```

Out[ ]:
```
0.5
```

In [ ]:
```
print('Confusion Matrix:')
pd.crosstab(y_true, pred_dc, rownames=['True'], colnames=['Predicted'], margins=
```

```
Confusion Matrix:
```

Out[ ]:

Predicted	0	1	All
True			
0	1995	469	2464
1	120	241	361
All	2115	710	2825

In [ ]:
```
dc_output = '{} \n Accuracy: {}'
dc = dc_output.format(classification_report(y_true, pred_dc, labels=[0,1]), accu
print(dc)
```

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87      2464
           1       0.34      0.67      0.45       361

    accuracy                           0.79      2825
   macro avg       0.64      0.74      0.66      2825
weighted avg       0.87      0.79      0.82      2825

 Accuracy: 0.791504424778761
```

In [ ]:
```
print(d)
```

```
                precision    recall  f1-score   support

           0       0.94      0.81      0.87      2464
           1       0.34      0.67      0.45       361

    accuracy                           0.79      2825
   macro avg       0.64      0.74      0.66      2825
weighted avg       0.87      0.79      0.82      2825

Accuracy: 0.791504424778761
```

XGBOOST

In [ ]:
```
# A parameter grid for XGBoost
# params = {
#         'max_depth': [3, 4, 6],
#         'subsample': [1.0],
#         'colsample_bytree': [0.6, 1.0],
#         'scale_pos_weight': [8, 10, 12],
#         'learning_rate' : [0.1],
#         'n_estimators' : [200]
#         }
```

Regular Training

GridSearch

In [ ]:
```
#xg_class_r = xgb.XGBClassifier(objective ='binary:logistic', random_state = 1,
```

In [ ]:
```
#cv_xgr = GridSearchCV(estimator=xg_class_r, param_grid=params, scoring='f1', ve
```

In [ ]:
```
#cv_xgr.fit(X_train, y_train.values.ravel())
```

In [ ]:
```
#cv_xgr.best_params_
# {'colsample_bytree': 1.0,
#  'learning_rate': 0.1,
#  'max_depth': 4,
#  'n_estimators': 200,
#  'scale_pos_weight': 8,
#  'subsample': 1.0}
```

In [ ]:
```
#cv_xgr.best_score_
#0.46577333079332783
```

Best Parameter

In [ ]:
```
#cv_xgr.best_estimator_
```

In [ ]:
```
```

```
xg_clf_r = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel
       colsample_bytree=1.0, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=4, min_child_weight=1, missing=1, n_estimators=200,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=1,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=8, seed=None,
       silent=True, subsample=1.0, verbose=1)
```

Predict

In [ ]:
```
xg_clf_r.fit(X_train, y_train.values.ravel())
```

```
[19:29:56] WARNING: ../src/learner.cc:576:
Parameters: { "silent", "verbose" } might not be used.

  This could be a false alarm, with some parameters getting used by language bin
dings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such ca
ses.


[19:29:56] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from
'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the ol
d behavior.
```

Out[ ]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1.0,
              enable_categorical=False, gamma=0, gpu_id=-1,
              importance_type=None, interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=4,
              min_child_weight=1, missing=1, monotone_constraints='()',
              n_estimators=200, n_jobs=1, nthread=1, num_parallel_tree=1,
              predictor='auto', random_state=1, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=8, seed=1, silent=True, subsample=1.0,
              tree_method='exact', validate_parameters=1, ...)
```

In [ ]:
```
pred_xgr = xg_clf_r.predict(X_test)
```

Evaluation

In [ ]:
```
print('Confusion Matrix:')
pd.crosstab(y_true, pred_xgr, rownames=['True'], colnames=['Predicted'], margins
```

```
Confusion Matrix:
```

Out[ ]:

Predicted	0	1	All
True			
0	1982	482	2464
1	105	256	361
All	2087	738	2825

In [ ]:
```
rxg_output = '{} \n Accuracy: {}'
```

```
rxg = rxg_output.format(classification_report(y_true, pred_xgr, labels=[0,1]), a
print(rxg)
```

```
              precision    recall  f1-score   support

           0       0.95      0.80      0.87      2464
           1       0.35      0.71      0.47       361

    accuracy                           0.79      2825
   macro avg       0.65      0.76      0.67      2825
weighted avg       0.87      0.79      0.82      2825

 Accuracy: 0.792212389380531
```

Compare with Random Forest

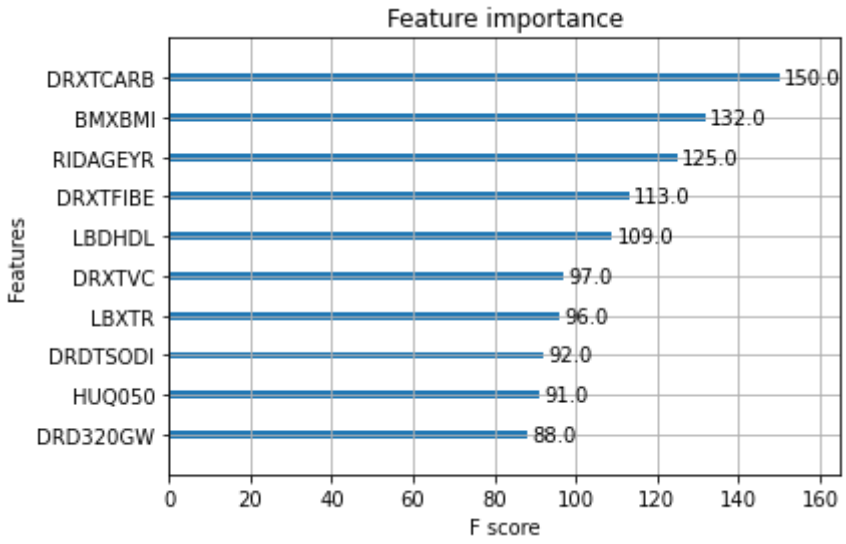
In [ ]:
```
print(r)
```

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87      2464
           1       0.35      0.67      0.46       361

    accuracy                           0.80      2825
   macro avg       0.65      0.74      0.67      2825
weighted avg       0.87      0.80      0.82      2825

 Accuracy: 0.7961061946902654
```

Feature Importance for XGBoost

In [ ]:
```
plot_importance(xg_clf_r, max_num_features = 10)
```

Out[ ]:
```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='Fe
atures'>
```



Upsampling: SMOTE

GridSearch

In [ ]:

```
#xg_class_u = xgb.XGBClassifier(objective ='binary:logistic', random_state = 1,
```

In [ ]:
```
#cv_xgu = GridSearchCV(estimator=xg_class_u, param_grid=params, scoring='f1', ve
```

In [ ]:
```
#cv_xgu.fit(X_sm, y_sm)
```

In [ ]:
```
#cv_xgu.best_params_
# {'colsample_bytree': 0.6,
#  'learning_rate': 0.1,
#  'max_depth': 6,
#  'n_estimators': 200,
#  'scale_pos_weight': 8,
#  'subsample': 1.0}
```

In [ ]:
```
#cv_xgu.best_score_
#0.8602420336517165
```

Best Parameter

In [ ]:
```
#cv_xgu.best_estimator_
```

In [ ]:
```
xg_clf_u = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel
       colsample_bytree=0.6, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=6, min_child_weight=1, missing=1, n_estimators=200,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=1,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=8, seed=None,
       silent=True, subsample=1.0, verbose=1)
```

Predict

In [ ]:
```
xg_clf_u.fit(X_sm, y_sm)
```

```
[19:31:40] WARNING: ../src/learner.cc:576:
Parameters: { "silent", "verbose" } might not be used.

  This could be a false alarm, with some parameters getting used by language bin
dings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such ca
ses.


[19:31:40] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from
'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the ol
d behavior.
```

Out[ ]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.6,
              enable_categorical=False, gamma=0, gpu_id=-1,
```

```
importance_type=None, interaction_constraints='',
learning_rate=0.1, max_delta_step=0, max_depth=6,
min_child_weight=1, missing=1, monotone_constraints='()',
n_estimators=200, n_jobs=1, nthread=1, num_parallel_tree=1,
predictor='auto', random_state=1, reg_alpha=0, reg_lambda=1,
scale_pos_weight=8, seed=1, silent=True, subsample=1.0,
tree_method='exact', validate_parameters=1, ...)
```

In [ ]:
```
pred_xgu = xg_clf_u.predict(X_test)
```

Evaluation

In [ ]:
```
print('Confusion Matrix:')
pd.crosstab(y_true, pred_xgu, rownames=['True'], colnames=['Predicted'], margins
```

```
Confusion Matrix:
```

Out[ ]:

Predicted	0	1	All
True			
0	2062	402	2464
1	139	222	361
All	2201	624	2825

In [ ]:
```
uxg_output = '{} \n Accuracy: {}'
uxg = uxg_output.format(classification_report(y_true, pred_xgu, labels=[0,1]), a
print(uxg)
```

```
              precision    recall  f1-score   support

           0       0.94      0.84      0.88      2464
           1       0.36      0.61      0.45       361

    accuracy                           0.81      2825
   macro avg       0.65      0.73      0.67      2825
weighted avg       0.86      0.81      0.83      2825

 Accuracy: 0.8084955752212389
```

Compare with Random Forest

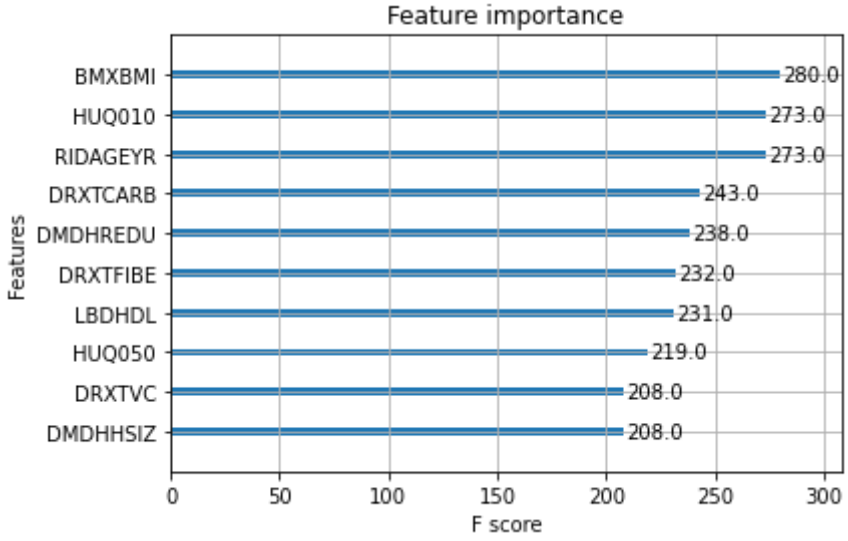
In [ ]:
```
print(u)
```

```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92      2464
           1       0.44      0.40      0.42       361

    accuracy                           0.86      2825
   macro avg       0.68      0.66      0.67      2825
weighted avg       0.85      0.86      0.86      2825

 Accuracy: 0.8591150442477876
```

Feature Importance for XGBoost

In [ ]:
```
plot_importance(xg_clf_u, max_num_features = 10)
```

Out[ ]:
```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='Features'>
```



Downsampling: TomekLinks

GridSearch

In [ ]:
```
#xg_class_d = xgb.XGBClassifier(objective ='binary:logistic', random_state = 1,
```

In [ ]:
```
#cv_xgd = GridSearchCV(estimator=xg_class_d, param_grid=params, scoring='f1', ve
```

In [ ]:
```
#cv_xgd.fit(X_tl, y_tl)
```

In [ ]:
```
#cv_xgd.best_params_
# {'colsample_bytree': 1.0,
#  'learning_rate': 0.1,
#  'max_depth': 4,
#  'n_estimators': 200,
#  'scale_pos_weight': 8,
#  'subsample': 1.0}
```

In [ ]:
```
#cv_xgd.best_score_
#0.4728752502035096
```

Best Parameter

In [ ]:
```
#cv_xgd.best_estimator_
```

In [ ]:
```
xg_clf_d = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel
       colsample_bytree=1.0, gamma=0, learning_rate=0.1, max_delta_step=0,
```

```
        max_depth=4, min_child_weight=1, missing=1, n_estimators=200,
        n_jobs=1, nthread=None, objective='binary:logistic', random_state=1,
        reg_alpha=0, reg_lambda=1, scale_pos_weight=8, seed=None,
        silent=True, subsample=1.0, verbose=1)
```

Predict

In [ ]:
```
xg_clf_d.fit(X_tl, y_tl)
```

```
[19:32:58] WARNING: ../src/learner.cc:576:
Parameters: { "silent", "verbose" } might not be used.

  This could be a false alarm, with some parameters getting used by language bin
dings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such ca
ses.


[19:32:58] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from
'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the ol
d behavior.
```

Out[ ]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1.0,
              enable_categorical=False, gamma=0, gpu_id=-1,
              importance_type=None, interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=4,
              min_child_weight=1, missing=1, monotone_constraints='()',
              n_estimators=200, n_jobs=1, nthread=1, num_parallel_tree=1,
              predictor='auto', random_state=1, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=8, seed=1, silent=True, subsample=1.0,
              tree_method='exact', validate_parameters=1, ...)
```

In [ ]:
```
pred_xgd = xg_clf_d.predict(X_test)
```

Evaluation

In [ ]:
```
#Creating a confusion matrix
print('Confusion Matrix:')
pd.crosstab(y_true, pred_xgd, rownames=['True'], colnames=['Predicted'], margins
```

```
Confusion Matrix:
```

Out[ ]:

Predicted	0	1	All
True			
0	1971	493	2464
1	104	257	361
All	2075	750	2825

In [ ]:
```
dxg_output = '{} \n Accuracy: {}'
dxg = dxg_output.format(classification_report(y_true, pred_xgd, labels=[0,1]), a
```

```
print(dxg)
```

```
              precision    recall  f1-score   support

           0       0.95      0.80      0.87      2464
           1       0.34      0.71      0.46       361

    accuracy                           0.79      2825
   macro avg       0.65      0.76      0.67      2825
weighted avg       0.87      0.79      0.82      2825

 Accuracy: 0.7886725663716814
```

Compare with Random Forest

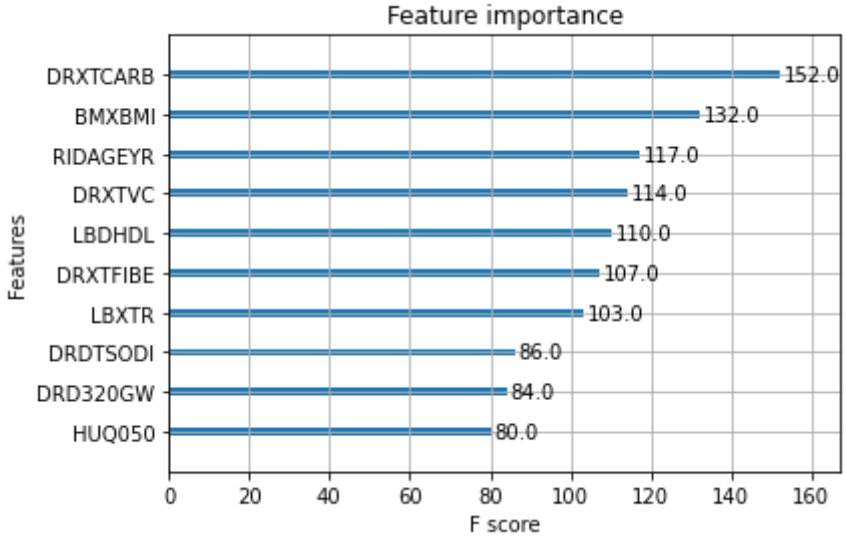
In [ ]:
```
print(d)
```

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87      2464
           1       0.34      0.67      0.45       361

    accuracy                           0.79      2825
   macro avg       0.64      0.74      0.66      2825
weighted avg       0.87      0.79      0.82      2825

 Accuracy: 0.791504424778761
```

Feature Importance for XGBoost

In [ ]:
```
plot_importance(xg_clf_d, max_num_features = 10)
```

Out[ ]:
```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='Features'>
```



Important Features: Risk Factors

Feature Selection: Chi-Square

In [ ]:
```
chi2_fi = featureScores.nlargest(10,'Score')
chi2_fi = chi2_fi.rename(columns={'Specs': 'Label', 'Score': 'Feature_Importance
chi2_fi
```

Out[ ]:

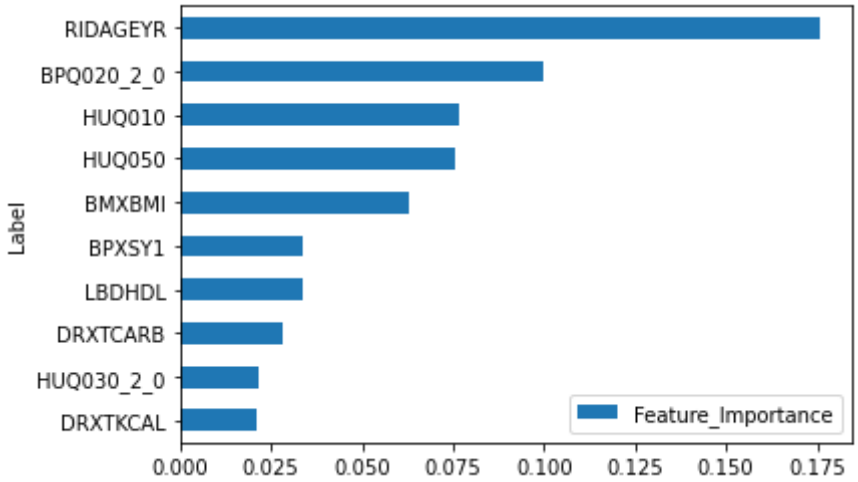
	Label	Feature_Importance
14	DRXTKCAL	71856.306275
17	DRXTPOTA	22334.886274
5	DRDTSODI	19060.586205
8	DRXTCALC	18891.120855
16	DRXTPHOS	16480.268606
9	DRXTCARB	13080.767992
20	DRXTVARE	9550.342221
0	RIDAGEYR	6416.967430
6	DRXTALCO	3760.483344
25	DRXTVC	3173.602016

Random Forest

Regular:

In [ ]:
```
ax_r.figure
```

Out[ ]:



In [ ]:
```
rfr_fi = rf_oo_r.copy()
rfr_fi
```

Out[ ]:

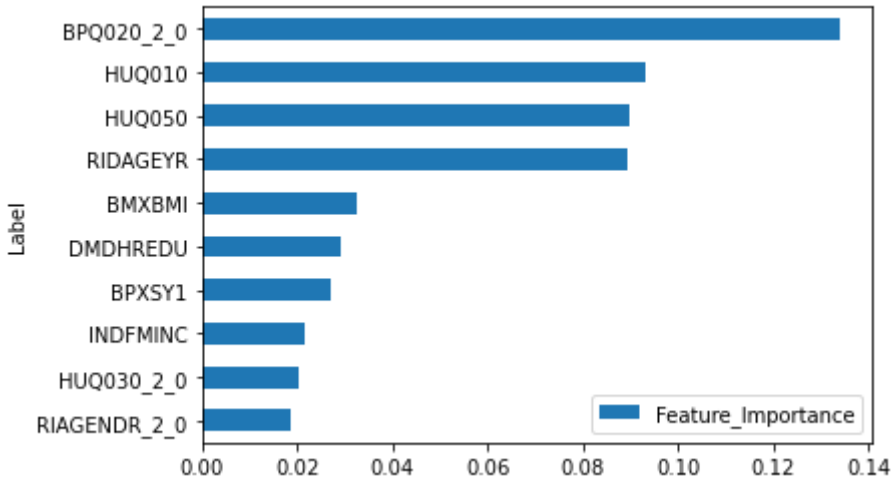
	Feature_Importance	Label
0	0.175557	RIDAGEYR
44	0.099665	BPQ020_2_0
34	0.076619	HUQ010
35	0.075479	HUQ050

	Feature_Importance	Label
29	0.062725	BMXBMI
27	0.033821	BPXSY1
31	0.033681	LBDHDL
9	0.028464	DRXTCARB
59	0.021536	HUQ030_2_0
14	0.021050	DRXTKCAL

Upsample:

In [ ]:
```
ax_u.figure
```

Out[ ]:



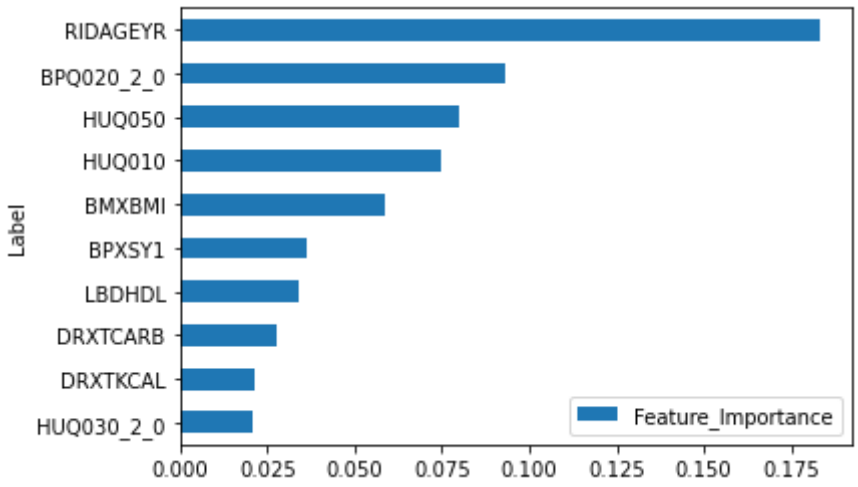
In [ ]:
```
rfu_fi = rf_oo_u.copy()
rfu_fi
```

Out[ ]:

	Feature_Importance	Label
44	0.134156	BPQ020_2_0
34	0.093164	HUQ010
35	0.089661	HUQ050
0	0.089192	RIDAGEYR
29	0.032461	BMXBMI
3	0.029392	DMDHREDU
27	0.027064	BPXSY1
2	0.021497	INDFMINC
59	0.020299	HUQ030_2_0
36	0.018513	RIAGENDR_2_0

Downsample:

In [ ]:
```
ax_d.figure
```

Out[ ]:


In [ ]:
```
rfd_fi = rf_oo_d.copy()
rfd_fi
```

Out[ ]:

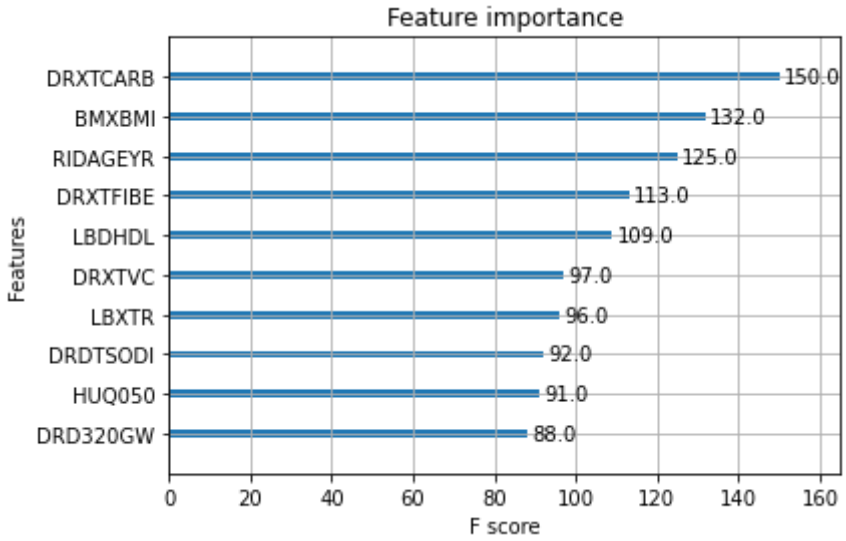
	Feature_Importance	Label
0	0.182928	RIDAGEYR
44	0.092933	BPQ020_2_0
35	0.080155	HUQ050
34	0.074754	HUQ010
29	0.058951	BMXBMI
27	0.036251	BPXSY1
31	0.033742	LBDHDL
9	0.027906	DRXTCARB
14	0.021677	DRXTKCAL
59	0.020714	HUQ030_2_0

XGBoost

Regular

In [ ]:
```
plot_importance(xg_clf_r, max_num_features = 10)
```

Out[ ]:
```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='Features'>
```



In [ ]:
```
xgr_fs = xg_clf_r.get_booster().get_score(importance_type='weight')
xgr_fs = pd.DataFrame(xgr_fs.items(), columns = ['Label', 'Feature_Importance'])
xgr_fs = xgr_fs.nlargest(10, 'Feature_Importance')
xgr_fs
```

Out[ ]:

	Label	Feature_Importance
9	DRXTCARB	150.0
29	BMXBMI	132.0
0	RIDAGEYR	125.0
12	DRXTFIBE	113.0
31	LBDHDL	109.0
25	DRXTVC	97.0
32	LBXTR	96.0
5	DRDTSODI	92.0
35	HUQ050	91.0
4	DRD320GW	88.0

In [ ]:
```
xgr_fi = pd.DataFrame(data = {'Feature_Importance':xg_clf_r.feature_importances_
xgr_fi = xgr_fi.nlargest(10,'Feature_Importance')
xgr_fi
```

Out[ ]:

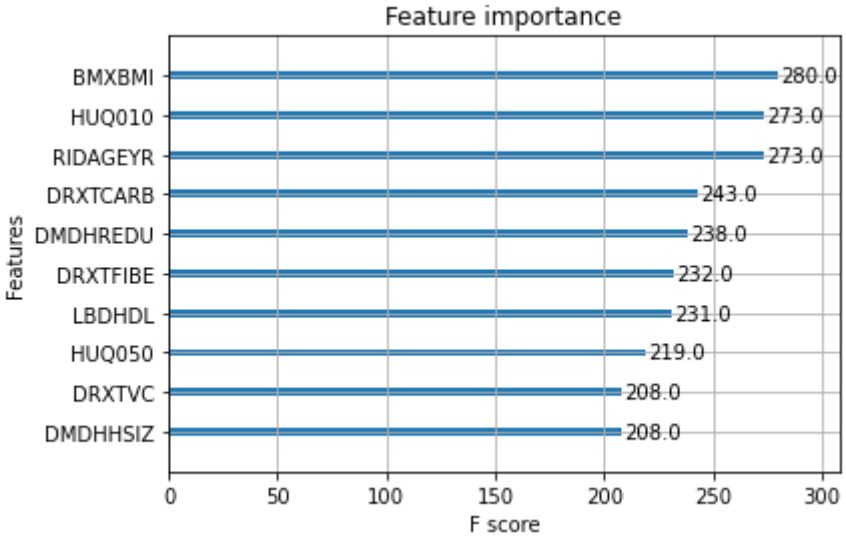
	Feature_Importance	Label
0	0.114660	RIDAGEYR
44	0.108025	BPQ020_2_0
34	0.065473	HUQ010
35	0.041907	HUQ050
38	0.029651	RIDRETH1_3_0

	Feature_Importance	Label
41	0.028990	DMDBORN4_2_0
29	0.028125	BMXBMI
31	0.023802	LBDHDL
58	0.022741	HUQ020_3_0
48	0.019666	PAQ665_2_0

Upsample:

In [ ]:
```
plot_importance(xg_clf_u, max_num_features = 10)
```

Out[ ]:
```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='Features'>
```



In [ ]:
```
xgu_fs = xg_clf_u.get_booster().get_score(importance_type='weight')
xgu_fs = pd.DataFrame(xgu_fs.items(), columns = ['Label', 'Feature_Importance'])
xgu_fs = xgu_fs.nlargest(10, 'Feature_Importance')
xgu_fs
```

Out[ ]:

	Label	Feature_Importance
29	BMXBMI	280.0
0	RIDAGEYR	273.0
34	HUQ010	273.0
9	DRXTCARB	243.0
3	DMDHREDU	238.0
12	DRXTFIBE	232.0
31	LBDHDL	231.0
35	HUQ050	219.0
1	DMDHHSIZ	208.0

	Label	Feature_Importance
25	DRXTVC	208.0

In [ ]:
```
xgu_fi = pd.DataFrame(data = {'Feature_Importance':xg_clf_u.feature_importances_
xgu_fi = xgu_fi.nlargest(10,'Feature_Importance')
xgu_fi
```

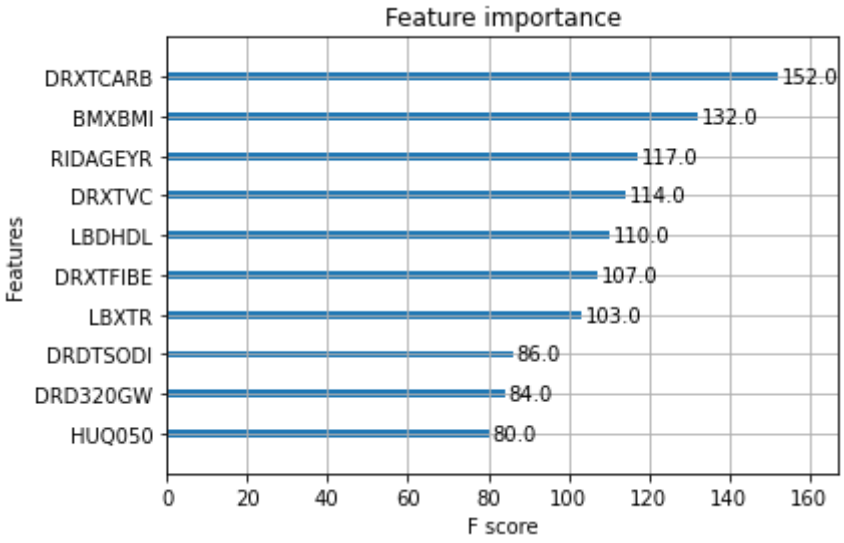
Out[ ]:

	Feature_Importance	Label
44	0.241901	BPQ020_2_0
35	0.082921	HUQ050
34	0.059202	HUQ010
0	0.043736	RIDAGEYR
59	0.039718	HUQ030_2_0
36	0.032349	RIAGENDR_2_0
51	0.031842	SMD410_1_0
38	0.025693	RIDRETH1_3_0
57	0.023903	HUQ020_2_0
39	0.022682	RIDRETH1_4_0

Downsample:

In [ ]:
```
plot_importance(xg_clf_d, max_num_features = 10)
```

Out[ ]:
```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='Fe
atures'>
```



In [ ]:
```
xgd_fs = xg_clf_d.get_booster().get_score(importance_type='weight')
xgd_fs = pd.DataFrame(xgd_fs.items(), columns = ['Label', 'Feature_Importance'])
xgd_fs = xgd_fs.nlargest(10, 'Feature_Importance')
xgd_fs
```

Out[ ]:

	Label	Feature_Importance
9	DRXTCARB	152.0
29	BMXBMI	132.0
0	RIDAGEYR	117.0
25	DRXTVC	114.0
31	LBDHDL	110.0
12	DRXTFIBE	107.0
32	LBXTR	103.0
5	DRDTSODI	86.0
4	DRD320GW	84.0
35	HUQ050	80.0

In [ ]:
```
xgd_fi = pd.DataFrame(data = {'Feature_Importance':xg_clf_d.feature_importances_
xgd_fi = xgd_fi.nlargest(10,'Feature_Importance')
xgd_fi
```

Out[ ]:

	Feature_Importance	Label
0	0.121709	RIDAGEYR
44	0.097705	BPQ020_2_0
34	0.067642	HUQ010
35	0.044495	HUQ050
38	0.034083	RIDRETH1_3_0
29	0.026940	BMXBMI
31	0.023834	LBDHDL
41	0.021730	DMDBORN4_2_0
58	0.021192	HUQ020_3_0
59	0.019643	HUQ030_2_0

Rankings:

Weights for each ranking:

In [ ]:
```
wts = {'chi2_fi': 0.025, 'rf_fi': .15, 'xg_fs': .10, 'xg_fi': .075}
```

In [ ]:
```
#Suffix names to join on
jl = ['chi2_fi', 'rfr_fi', 'rfu_fi', 'rfd_fi', 'xgr_fs', 'xgu_fs', 'xgd_fs', 'xg
```

In [ ]:
```
#Outer join function
def outerjoin_df(dfs_list):
```

```
    n = 1
    df_join = dfs_list[0]
    for d in dfs_list[1:]:
        df_join = df_join.merge(d, how='outer', suffixes=('_'+jl[n-1], '_'+jl[n]
        n = n+1
    return df_join
```

In [ ]:
```
#List of top 10 feature importances
join_list = [chi2_fi, rfr_fi, rfu_fi, rfd_fi, xgr_fs, xgu_fs, xgd_fs, xgr_fi, xg
```

In [ ]:
```
#Joined list of top 10 feature importances
joined = outerjoin_df(join_list)
```

In [ ]:
```
joined.head()
```

Out[ ]:

	Label	Feature_Importance_chi2_fi	Feature_Importance_rfr_fi	Feature_Importance_rfu_fi
0	DRXTKCAL	71856.306275	0.02105	NaN
1	DRXTPOTA	22334.886274	NaN	NaN
2	DRDTSODI	19060.586205	NaN	NaN
3	DRXTCALC	18891.120855	NaN	NaN
4	DRXTPHOS	16480.268606	NaN	NaN

In [ ]:
```
#Get the percentage importance of top ten for each feature importance
pct_list = pd.DataFrame()
pct_list['Label'] = joined['Label']
for i, j in enumerate(jl):
    pct_list[j] = joined['Feature_Importance_'+jl[i]]/joined['Feature_Importance
```

In [ ]:
```
#Fill in NaNs with 0
pct_list = pct_list.fillna(0)
pct_list.head()
```

Out[ ]:

	Label	chi2_fi	rfr_fi	rfu_fi	rfd_fi	xgr_fs	xgu_fs	xgd_fs	xgr_fi	xgu_fi
0	DRXTKCAL	0.389243	0.033487	0.0	0.034408	0.000000	0.0	0.000000	0.0	0.0
1	DRXTPOTA	0.120987	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.0
2	DRDTSODI	0.103250	0.000000	0.0	0.000000	0.084172	0.0	0.079263	0.0	0.0
3	DRXTCALC	0.102332	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.0
4	DRXTPHOS	0.089273	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.0

Column for weighted overall risk factor importance ranking

In [ ]:
```
#Weighted overall ranking
pct_list['Importance'] = pct_list[jl[0]] * wts['chi2_fi'] + pct_list[jl[1]] * wt
```

In [ ]:
```
pct_list.head()
```

Out[ ]:

	Label	chi2_fi	rfr_fi	rfu_fi	rfd_fi	xgr_fs	xgu_fs	xgd_fs	xgr_fi	xgu_fi
0	DRXTKCAL	0.389243	0.033487	0.0	0.034408	0.000000	0.0	0.000000	0.0	0.0
1	DRXTPOTA	0.120987	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.0
2	DRDTSODI	0.103250	0.000000	0.0	0.000000	0.084172	0.0	0.079263	0.0	0.0
3	DRXTCALC	0.102332	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.0
4	DRXTPHOS	0.089273	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.0

Identified Top 10 Risk Factors:

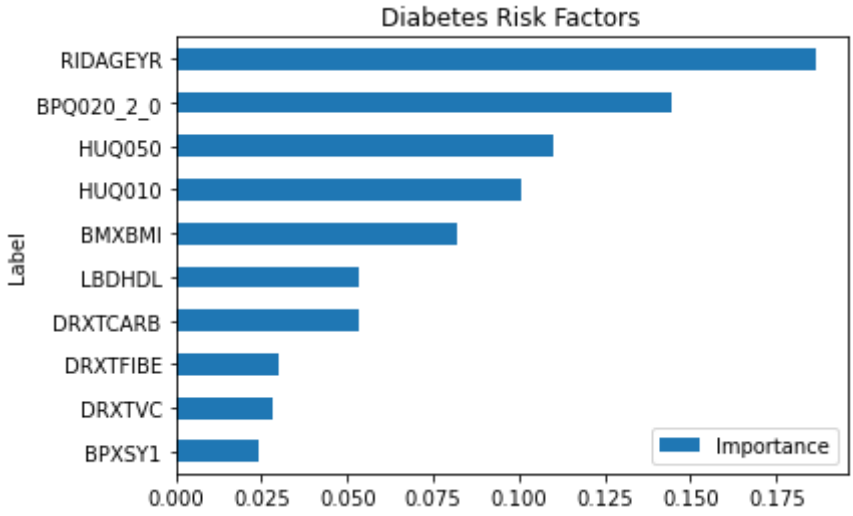
In [ ]:
```
Risk_Factors = pct_list[['Label','Importance']].copy()
```

In [ ]:
```
Risk_Factors_10 = Risk_Factors.nlargest(10, 'Importance')
Risk_Factors_10
```

Out[ ]:

	Label	Importance
7	RIDAGEYR	0.186267
10	BPQ020_2_0	0.144253
12	HUQ050	0.109887
11	HUQ010	0.100704
13	BMXBMI	0.082241
15	LBDHDL	0.053214
5	DRXTCARB	0.053045
20	DRXTFIBE	0.029847
9	DRXTVC	0.028460
14	BPXSY1	0.024011

In [ ]:
```
Risk_Factors_10 = Risk_Factors_10.sort_values(by=['Importance'])
Risk_Factors_Plot = Risk_Factors_10.plot.barh(y='Importance', x='Label', title='
```


Diabetes Risk Factors
RIDAGEYR
BPQ020_2_0
HUQ050
HUQ010
BMXBMI
LBDHDL
DRXTCARB
DRXTFIBE
DRXTVC
BPXSY1
Label
Importance
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175