In [208]:
```
import pandas as pd
import numpy as np


# Use a csv file created in an earlier week, based on the original data set, but in which some problems have been resolved incl. categorical entries
# the health_NoCats now represents the corrected file with categorical data translated into numerical format
health = pd.read_csv('health_NoCats.csv', header = 0)  #this is the updated file with the more useable Age_Category which is ordered (unlike the original)
```

In [209]:
```
#define the list of features (columns) to be used as predictors
#note this subset is constructed to create greater independence between columns and to eliminate unhelpful features based on bar charts of those features vs. drug persistency

features = ['Gluco_Record_During_Rx', 'Dexa_During_Rx', 'Frag_Frac_During_Rx', 'Risk_Segment_During_Rx', 'Adherent_Flag', 'Idn_Indicator',
            'Injectable_Experience_During_Rx', 'Comorb_Encounter_For_Screening_For_Malignant_Neoplasms', 'Comorb_Encounter_For_Immunization',
            'Comorb_Encntr_For_General_Exam_W_O_Complaint_Susp_Or_Reprtd_Dx', 'Comorb_Vitamin_D_Deficiency',
            'Comorb_Other_Joint_Disorder_Not_Elsewhere_Classified', 'Comorb_Encntr_For_Oth_Sp_Exam_W_O_Complaint_Suspected_Or_Reprtd_Dx',
            'Comorb_Long_Term_Current_Drug_Therapy', 'Comorb_Dorsalgia', 'Comorb_Personal_History_Of_Other_Diseases_And_Conditions',
            'Comorb_Other_Disorders_Of_Bone_Density_And_Structure', 'Comorb_Disorders_of_lipoprotein_metabolism_and_other_lipidemias',
            'Comorb_Osteoporosis_without_current_pathological_fracture', 'Comorb_Personal_history_of_malignant_neoplasm',
            'Comorb_Gastro_esophageal_reflux_disease', 'Concom_Cholesterol_And_Triglyceride_Regulating_Preparations', 'Concom_Narcotics',
            'Concom_Systemic_Corticosteroids_Plain', 'Concom_Anti_Depressants_And_Mood_Stabilisers', 'Concom_Fluoroquinolones',
            'Concom_Cephalosporins', 'Concom_Macrolides_And_Similar_Types', 'Concom_Broad_Spectrum_Penicillins', 'Concom_Anaesthetics_General',
            'Concom_Viral_Vaccines', 'Risk_Rheumatoid_Arthritis', 'Risk_Untreated_Chronic_Hyperthyroidism', 'Risk_Untreated_Chronic_Hypogonadism',
            'Risk_Smoking_Tobacco', 'Risk_Chronic_Malnutrition_Or_Malabsorption', 'Risk_Chronic_Liver_Disease',  'Risk_Low_Calcium_Intake',
            'Risk_Vitamin_D_Insufficiency', 'Risk_Poor_Health_Frailty', 'Risk_Excessive_Thinness', 'Risk_Estrogen_Deficiency', 'Risk_Immobilization',
            'Dexa_Freq_During_Rx_Bucket_Flag','Change_RiskSeg_Worsened','Change_RiskSeg_Improved', 'Change_RiskSeg_Unk' , 'ChangedTScore_Worsened',
            'ChangedTScore_Improved', 'ChangedTScore_Unk', 'AsianRace_Flag',  'Midwest_Flag', 'Ntm_Speciality_MyBuckets1', 'Ntm_Speciality_MyBuckets2' ]
```

In [210]:
```
# import the sklearn package for use in log regression, import confusion matrix
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

# instantiate the model
logreg = LogisticRegression()
```

In [211]:
```
X = health[features]
y = health.Persistency_Flag


# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
```

In [212]:
```
# fit the model with data - cannot run it yet, until variables are transformed
logreg.fit(X_train,y_train)

#
y_pred=logreg.predict(X_test)

confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[487  56]
 [ 98 215]]

/Users/jen/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to
silence this warning.
  FutureWarning)
```

In [213]:
```
print(round(100*(98+56)/(487+56+98+215),1), '%')

print("This is the percentage incorrectly classified within the test set")
print("Further refinements will be applied in an effort to improve this error rate")
```

```
18.0 %
This is the percentage incorrectly classified within the test set
Further refinements will be applied in an effort to improve this error rate
```

In [214]:
```
#statsmodels provides more information by way of summary()
#we can use this to refine the columns further (eliminating some)
#we are looking to keep those columns (predictor variables) with low values in the 'P>[t]'' column of the summary, or the ones where 'coef' has larger absolute value

import statsmodels.api as sm
X_train = sm.add_constant(X_train)
lm_2 = sm.OLS(y_train, X_train).fit()
lm_2.summary()
```

```
/Users/jen/opt/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2495: FutureWarning: Method .ptp is deprecated and will be removed in a future version. Use nump
y.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

Out[214]:

OLS Regression Results

Dep. Variable:	Persistency_Flag	**R-squared:**	0.457
Model:	OLS	**Adj. R-squared:**	0.445
Method:	Least Squares	**F-statistic:**	39.85
Date:	Tue, 07 Dec 2021	**Prob (F-statistic):**	4.47e-289
Time:	20:07:49	**Log-Likelihood:**	-1004.7
No. Observations:	2568	**AIC:**	2117.
Df Residuals:	2514	**BIC:**	2433.
Df Model:	53		
Covariance Type:	nonrobust		

In [215]:
```
#now redefine features eliminating all those with importance vals from the list above less than 0.01
features = ['Gluco_Record_During_Rx', 'Dexa_During_Rx', 'Frag_Frac_During_Rx', 'Risk_Segment_During_Rx', 'Adherent_Flag', 'Idn_Indicator',
            'Injectable_Experience_During_Rx', 'Comorb_Encounter_For_Screening_For_Malignant_Neoplasms', 'Comorb_Encounter_For_Immunization',
            'Comorb_Encntr_For_General_Exam_W_O_Complaint_Susp_Or_Reprtd_Dx', 'Comorb_Vitamin_D_Deficiency',
            'Comorb_Other_Joint_Disorder_Not_Elsewhere_Classified', 'Comorb_Encntr_For_Oth_Sp_Exam_W_O_Complaint_Suspected_Or_Reprtd_Dx',
            'Comorb_Long_Term_Current_Drug_Therapy', 'Comorb_Dorsalgia', 'Comorb_Personal_History_Of_Other_Diseases_And_Conditions',
            'Comorb_Other_Disorders_Of_Bone_Density_And_Structure', 'Comorb_Disorders_of_lipoprotein_metabolism_and_other_lipidemias',
            'Comorb_Osteoporosis_without_current_pathological_fracture', 'Comorb_Personal_history_of_malignant_neoplasm',
            'Comorb_Gastro_esophageal_reflux_disease', 'Concom_Cholesterol_And_Triglyceride_Regulating_Preparations',
            'Concom_Narcotics', 'Concom_Systemic_Corticosteroids_Plain', 'Concom_Anti_Depressants_And_Mood_Stabilisers', 'Concom_Fluoroquinolones',
            'Concom_Cephalosporins', 'Concom_Macrolides_And_Similar_Types', 'Concom_Broad_Spectrum_Penicillins', 'Concom_Anaesthetics_General',
            'Concom_Viral_Vaccines', 'Risk_Smoking_Tobacco', 'Risk_Chronic_Malnutrition_Or_Malabsorption','Risk_Vitamin_D_Insufficiency',
            'Dexa_Freq_During_Rx_Bucket_Flag', 'Change_RiskSeg_Unk',
            'ChangedTScore_Unk', 'Midwest_Flag', 'Ntm_Speciality_MyBuckets1' ]

#rebuild the test and training sets with this new reduced set of features and without the extra column used by stats models
X = health[features]
y = health.Persistency_Flag

# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
```

In [216]:
```
from sklearn.ensemble import RandomForestClassifier



#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

# prediction on test set
y_pred=clf.predict(X_test)

# Import scikit-learn metrics module for accuracy calculation
# from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8095794392523364
```

In [217]:
```

# the above classifier also affords the opportunity to rank the predictor variables by importance (according to this classification method)

feature_imp = pd.Series(clf.feature_importances_,index=features).sort_values(ascending=False)
feature_imp
```

Out[217]:

```
Dexa_Freq_During_Rx_Bucket_Flag                                  0.112446
Dexa_During_Rx                                                   0.072713
Comorb_Long_Term_Current_Drug_Therapy                            0.055298
Comorb_Encounter_For_Screening_For_Malignant_Neoplasms           0.048567
```

In [218]:
```
#now with the original, log reg on smaller set of features (predictors)
# fit the model with data
# instantiate the model
logreg = LogisticRegression()
logreg.fit(X_train,y_train)

#
y_pred=logreg.predict(X_test)

#confusion matrix
from sklearn.metrics import confusion_matrix

confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[481  62]
 [101 212]]

/Users/jen/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to
silence this warning.
  FutureWarning)
```

In [219]:
```
#again the accuracy is very close, within 1% of what it was with more predictors
print('error rate with log reg and fewer predictors')
print(round(100*(62+101)/(481+62+101+212),2))
```

```
error rate with log reg and fewer predictors
19.04
```

In [220]:
```
# Note - the accuracy with random forest on fewer predictors was also very close to what it was with the larger set of predictors (within 2 %)
```

In [221]:
```
from sklearn.ensemble import AdaBoostClassifier #For Classification
from sklearn.ensemble import AdaBoostRegressor #For Regression
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
cl = AdaBoostClassifier(n_estimators=100, base_estimator=dtree,learning_rate=1)
cl.fit(X_train,y_train)

# prediction on test set
y_pred=cl.predict(X_test)


# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7663551401869159
```

In [222]:
```
from sklearn.ensemble import GradientBoostingClassifier #For Classification
from sklearn.ensemble import GradientBoostingRegressor #For Regression
cl = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1)
cl.fit(X_train, y_train)

# prediction on test set
y_pred=cl.predict(X_test)


# Model Accuracy, how often is the classifier correct?
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8084112149532711
```

In [223]:

```
# to run the next one I had to install a new package
# install -c anaconda py-xgboost
```

In [224]:

```

from xgboost import XGBClassifier
xgbc = XGBClassifier()

xgbc.fit(X_train, y_train)


# prediction on test set
y_pred=xgbc.predict(X_test)


# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8212616822429907
```

In [225]:

```
# in order to use the next classifiers, that take into account imbalanced classes, I had to install imbalanced-learn

#conda install -c conda-forge imbalanced-learn
```

In [226]:

```
# This classifier addresses the imbalanced classes (unequal persistent vs. non-persistent) producing substantially superior results to earlier efforts
# that ignored the class imbalance

# bagged decision trees with random undersampling for imbalanced classification
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from imblearn.ensemble import BalancedBaggingClassifier

# define model
model = BalancedBaggingClassifier()



# generate dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=4)
# define model
model = BalancedBaggingClassifier()
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.2f' % mean(scores))

scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# summarize performance
print('Mean Accuracy: %.2f' % mean(scores))
```

```
Mean ROC AUC: 0.97
Mean Accuracy: 0.95
```

In [227]:

```
# Random  Forest with random undersampling for Imbalanced Classif.
#This is another classifier that addresses the imbalanced classes (unequal persistent vs. non-persistent)
```

```python
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from imblearn.ensemble import BalancedRandomForestClassifier


# define model
model = BalancedRandomForestClassifier(n_estimators=10)


# generate dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=4)
# define model
model = BalancedRandomForestClassifier(n_estimators=10)
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.2f' % mean(scores))


# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# summarize performance
print('Mean accuracy: %.2f' % mean(scores))
```

```
Mean ROC AUC: 0.97
Mean accuracy: 0.93
```

In [228]:
```python
print("Best classifier on this data appears to be the BalancedBaggingClassifier for imbalanced classes from imblearn.ensemble ")
```

```
Best classifier on this data appears to be the BalancedBaggingClassifier for imbalanced classes from imblearn.ensemble
```

In [ ]:

In [ ]: