Peer-graded Assignment: Course Final Project:

Supervised Machine Learning: Classification

About Dataset:

Context:

The dataset includes data from a random sample of 20,000 digital and 20,000 film-screen mammograms received by women age 60-89 years within the Breast Cancer Surveillance Consortium (BCSC) between January 2005 and December 2008. Some women contribute multiple examinations to the dataset. Data is useful in teaching about data analysis, epidemiological study designs, or statistical methods for binary outcomes or correlated data.

Content:

Features:

```
Feature Name:                             Type:     Description:


Age_At_The_Time_Of_Mammography          number  Patient's age in years at time of mammogram

Radiologists_Assessment                string   Radiologist's assessment based on the BI-RADS
scale

Comparison_Mammogram_From_Mammography   string  Comparison mammogram from prior mammography
examination available

Patients_BI_RADS_Breast_Density        string   Patient's BI-RADS breast density as recorded at
time of mammogram

Family_History_Of_Breast_Cancer        string   Family history of breast cancer in a first degree
relative

Current_Use_Of_Hormone_Therapy          string  Current use of hormone therapy at time of
mammogram

Binary_Indicator                          string    Binary indicator of whether the woman had
ever received a prior
                                                    mammogram

History_Of_Breast_Biopsy                  string    Prior history of breast biopsy

Is_Film_Or_Digital_Mammogra               boolean   Film or digital mammogram
                                                    (true=Digital mammogram, false=Film

mammogram)
```

Target:

```
Is_Binary_Indicator_Of_Cancer_Diagnosis   boolean   Binary indicator of cancer diagnosis within
one year of
                                                    screening mammogram
                                                    (false= No cancer diagnosis, true= Cancer

diagnosis)
```

Acknowledgement:

https://www.kaggle.com/haithemhermessi/breast-cancer-screening-data-set

Acknowledgement to Breast Cancer Surveillance Consortium (BCSC) for making this data set available for research purposes.

Objective of Analysis: Prediction of Cancer Diagnosis

In [1]:
```
#Importing necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#Importing the Data

df=pd.read_csv('data.csv',index_col='Patients_Study_ID')
```

In [3]:
```
df.head()
```

Out[3]:

	Age_At_The_Time_Of_Mammography	Radiologists_Assessment	Is_Binary_Indicator_Of_Cancer_Diagnosis	Comparison_Mammo
Patients_Study_ID				
1	62	Negative	False	
2	65	Negative	False	
3	69	Needs additional imaging	False	
4	64	Benign findings	False	
5	63	Probably benign	False	

In [4]:
```
#List of Features

features_list=['Age_At_The_Time_Of_Mammography','Radiologists_Assessment','Is_Binary_Indicator_Of_Cancer_Diagnosi
```

Data Cleaning and Feature Engineering:

In the given dataset, the missing values have been listed as 'Missing', hence we need to convert them to NaN values in order to impute them using central tendencies

In [5]:
```
for x in features_list:
    if 'Missing' in df[x].unique():
        df[x].replace({'Missing':np.nan},inplace=True)
```

Count of missing values for each attribute is given below:

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
Age_At_The_Time_Of_Mammography              0
Radiologists_Assessment                     0
Is_Binary_Indicator_Of_Cancer_Diagnosis     0
Comparison_Mammogram_From_Mammography    4680
Patients_BI_RADS_Breast_Density             0
Family_History_Of_Breast_Cancer           228
Current_Use_Of_Hormone_Therapy           1772
Binary_Indicator                          578
History_Of_Breast_Biopsy                  815
Is_Film_Or_Digital_Mammogram                0
Body_Mass_Index                         23208
dtype: int64
```

More than 50% values are missing for the atttribute 'Body_Mass_Index', hence it can be dropped

In [7]:
```
#Dropping 'Body_Mass_Index' column from dataset

df.drop(['Body_Mass_Index'],axis=1,inplace=True)
features_list.remove('Body_Mass_Index')
```

Info. about the dataset after dropping 'Body_Mass_Index':

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 39998 entries, 1 to 36714
Data columns (total 10 columns):
 #   Column                                   Non-Null Count  Dtype
---  ------                                   --------------  -----
 0   Age_At_The_Time_Of_Mammography           39998 non-null  int64
 1   Radiologists_Assessment                  39998 non-null  object
 2   Is_Binary_Indicator_Of_Cancer_Diagnosis  39998 non-null  bool
 3   Comparison_Mammogram_From_Mammography    35318 non-null  object
 4   Patients_BI_RADS_Breast_Density          39998 non-null  object
 5   Family_History_Of_Breast_Cancer          39770 non-null  object
 6   Current_Use_Of_Hormone_Therapy           38226 non-null  object
```

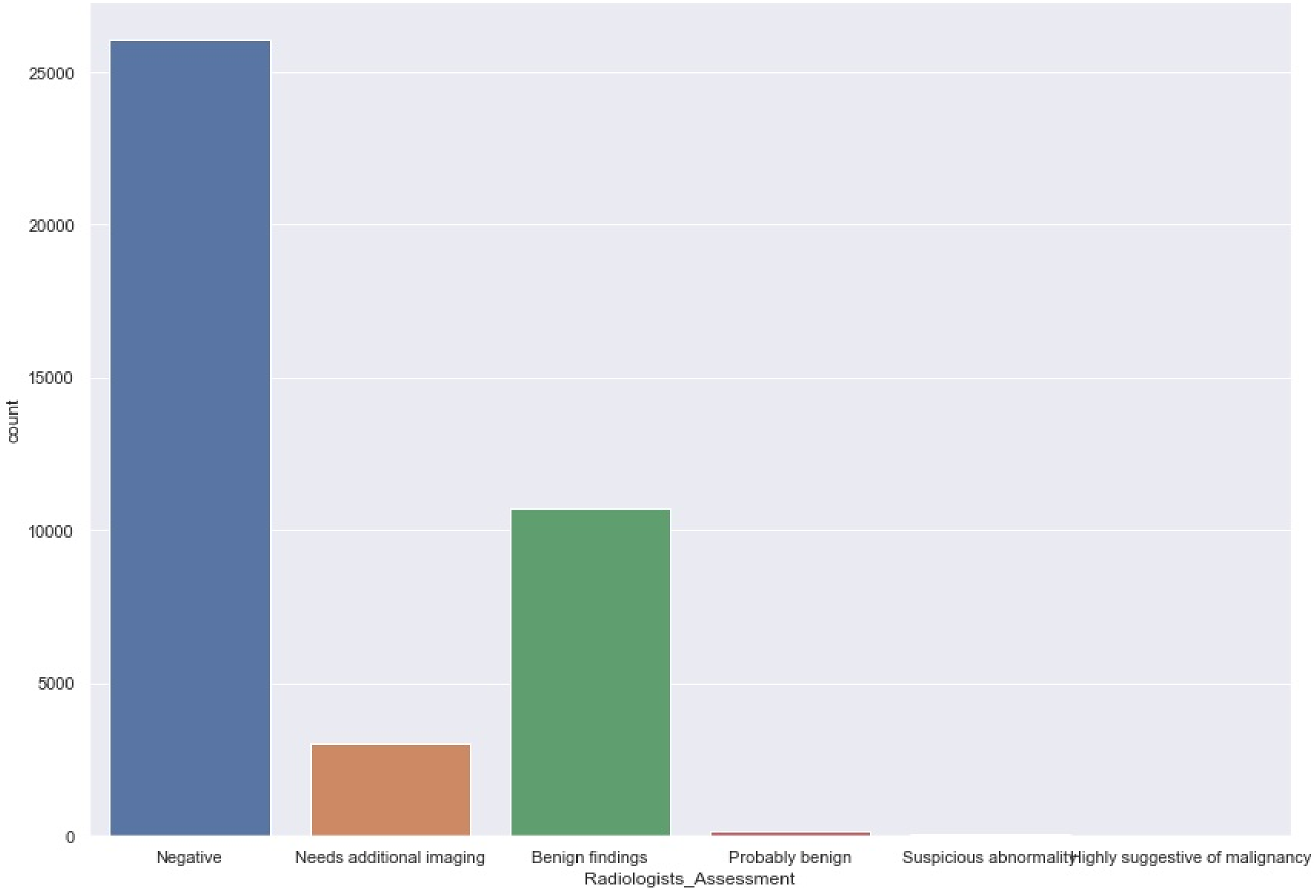
```
 7   Binary_Indicator                       39420 non-null  object
 8   History_Of_Breast_Biopsy               39183 non-null  object
 9   Is_Film_Or_Digital_Mammogram           39998 non-null  bool
dtypes: bool(2), int64(1), object(7)
memory usage: 2.8+ MB
```

In [9]:
```
sns.set(rc={'figure.figsize':(14,10)})
```

Performing Feature-Engineering:

In [10]:
```
#Count-plot for 'Radiologists_Assessment' attribute

print(df['Radiologists_Assessment'].value_counts())
sns.countplot(df['Radiologists_Assessment'])

#Numerical Encoding of Categorical Values:
#'Negative':0
#'Probably benign':1
#'Benign findings':2
#'Needs additional imaging':3
#'Suspicious abnormality':4
#'Highly suggestive of Maglignancy':5

df['Radiologists_Assessment'].replace({'Negative':0,'Probably benign':1,'Benign findings':2,'Needs additional ima
```

```
Negative                            26031
Benign findings                     10717
Needs additional imaging             3049
Probably benign                       139
Suspicious abnormality                 57
Highly suggestive of malignancy         5
Name: Radiologists_Assessment, dtype: int64
```



In [11]:
```
#Value Counts for other attributes:

print(df['Family_History_Of_Breast_Cancer'].value_counts())

print('')

print(df['Current_Use_Of_Hormone_Therapy'].value_counts())

print('')
```

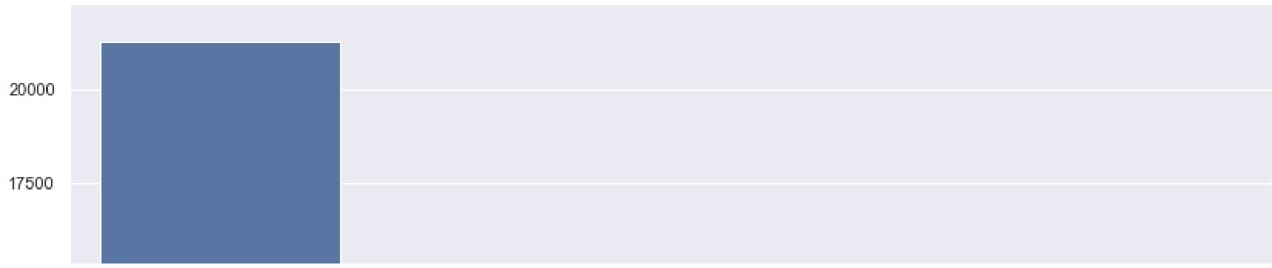
```
print(df['Binary_Indicator'].value_counts())

print('')

print(df['History_Of_Breast_Biopsy'].value_counts())

print('')

print(df['Comparison_Mammogram_From_Mammography'].value_counts())

print('')

print(df['Is_Binary_Indicator_Of_Cancer_Diagnosis'].value_counts())

print('')

print(df['Is_Film_Or_Digital_Mammogram'].value_counts())

#Binary Encoding:

df['Family_History_Of_Breast_Cancer'].replace({'Yes':1,'No':0},inplace=True)
df['Current_Use_Of_Hormone_Therapy'].replace({'Yes':1,'No':0},inplace=True)
df['Binary_Indicator'].replace({'Yes':1,'No':0},inplace=True)
df['History_Of_Breast_Biopsy'].replace({'Yes':1,'No':0},inplace=True)
df['Comparison_Mammogram_From_Mammography'].replace({'Yes':1,'No':0},inplace=True)
df['Is_Binary_Indicator_Of_Cancer_Diagnosis'].replace({False:0,True:1},inplace=True)
df['Is_Film_Or_Digital_Mammogram'].replace({False:0,True:1},inplace=True)
```

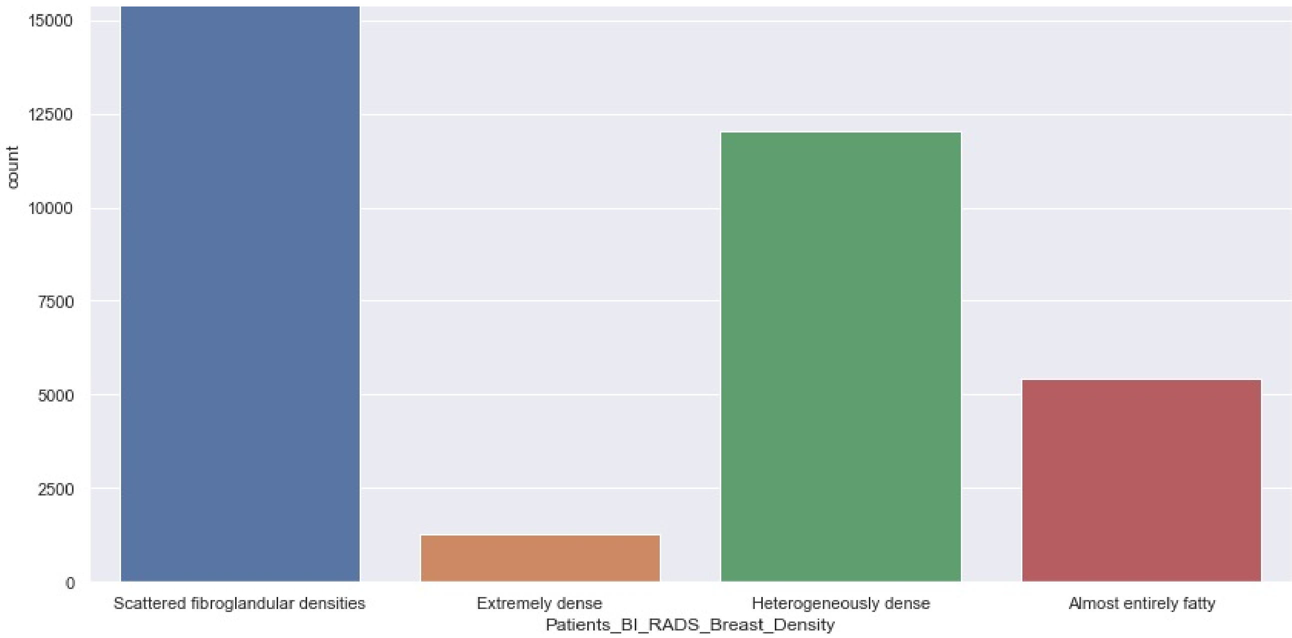
```
No     33027
Yes     6743
Name: Family_History_Of_Breast_Cancer, dtype: int64

No     33977
Yes     4249
Name: Current_Use_Of_Hormone_Therapy, dtype: int64

Yes    39124
No       296
Name: Binary_Indicator, dtype: int64

No     28733
Yes    10450
Name: History_Of_Breast_Biopsy, dtype: int64

Yes    34016
No      1302
Name: Comparison_Mammogram_From_Mammography, dtype: int64

False    39739
True       259
Name: Is_Binary_Indicator_Of_Cancer_Diagnosis, dtype: int64

True     20000
False    19998
Name: Is_Film_Or_Digital_Mammogram, dtype: int64
```

In [12]:

```
#Count-plot for 'Patients_BI_RADS_Breast_Density' attribute

print(df['Patients_BI_RADS_Breast_Density'].value_counts())
sns.countplot(df['Patients_BI_RADS_Breast_Density'])

#Numerical Encoding of Categorical Values:
#'Scattered fibroglandular densities':0
#'Heterogeneously dense':1
#'Almost entirely fatty':2
#'Extremely dense':3

df['Patients_BI_RADS_Breast_Density'].replace({'Scattered fibroglandular densities':0,'Heterogeneously dense':1,'
```

```
Scattered fibroglandular densities    21246
Heterogeneously dense                 12028
Almost entirely fatty                  5429
Extremely dense                        1295
Name: Patients_BI_RADS_Breast_Density, dtype: int64
```





In [13]:
```
features_list=df.columns.tolist()
features_list.remove('Is_Binary_Indicator_Of_Cancer_Diagnosis')
```

Imputation of Missing Values:

In [14]:
```
df.isnull().sum()
```

Out[14]:
```
Age_At_The_Time_Of_Mammography              0
Radiologists_Assessment                     0
Is_Binary_Indicator_Of_Cancer_Diagnosis     0
Comparison_Mammogram_From_Mammography    4680
Patients_BI_RADS_Breast_Density             0
Family_History_Of_Breast_Cancer           228
Current_Use_Of_Hormone_Therapy           1772
Binary_Indicator                          578
History_Of_Breast_Biopsy                  815
Is_Film_Or_Digital_Mammogram                0
dtype: int64
```

In [15]:
```
for x in features_list:
    df[x].fillna(df[x].median(),inplace=True)
```

Missing Values for all the attributes have been imputed by the corresponding Median values

In [16]:
```
features=df[features_list]
target=df['Is_Binary_Indicator_Of_Cancer_Diagnosis']
```

In [17]:
```
df.isnull().sum()
```

Out[17]:
```
Age_At_The_Time_Of_Mammography           0
Radiologists_Assessment                  0
Is_Binary_Indicator_Of_Cancer_Diagnosis  0
Comparison_Mammogram_From_Mammography    0
Patients_BI_RADS_Breast_Density          0
Family_History_Of_Breast_Cancer          0
Current_Use_Of_Hormone_Therapy           0
Binary_Indicator                         0
History_Of_Breast_Biopsy                 0
Is_Film_Or_Digital_Mammogram             0
dtype: int64
```

In [18]:
```
df.describe()
```

Out[18]:

	Age_At_The_Time_Of_Mammography	Radiologists_Assessment	Is_Binary_Indicator_Of_Cancer_Diagnosis	Comparison_Mammogram_From_I
count	39998.000000	39998.000000	39998.000000	
mean	69.555703	0.774364	0.006475	

std	7.202581	1.089773	0.080209
min	60.000000	0.000000	0.000000
25%	63.000000	0.000000	0.000000
50%	68.000000	0.000000	0.000000
75%	75.000000	2.000000	0.000000
max	89.000000	5.000000	1.000000

Clearly there are no outliers in the given dataset as majority of the attributes are categorical in nature

Summary of the dataset after performing Data Cleaning and Feature Engineering Techniques:

In [19]:
```
df.head()
```

Out[19]:

	Age_At_The_Time_Of_Mammography	Radiologists_Assessment	Is_Binary_Indicator_Of_Cancer_Diagnosis	Comparison_Mammo
Patients_Study_ID				
1	62	0	0	
2	65	0	0	
3	69	3	0	
4	64	2	0	
5	63	1	0	

In [20]:
```
df.info()
```

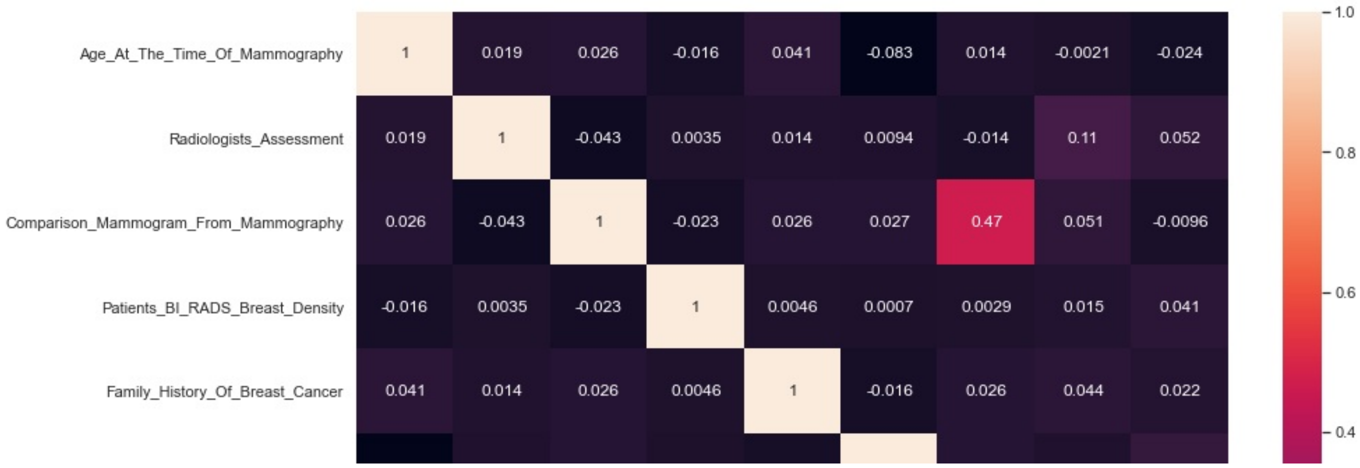
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 39998 entries, 1 to 36714
Data columns (total 10 columns):
 #   Column                                   Non-Null Count  Dtype  
---  ------                                   --------------  -----  
 0   Age_At_The_Time_Of_Mammography           39998 non-null  int64  
 1   Radiologists_Assessment                  39998 non-null  int64  
 2   Is_Binary_Indicator_Of_Cancer_Diagnosis  39998 non-null  int64  
 3   Comparison_Mammogram_From_Mammography    39998 non-null  float64
 4   Patients_BI_RADS_Breast_Density          39998 non-null  int64  
 5   Family_History_Of_Breast_Cancer          39998 non-null  float64
 6   Current_Use_Of_Hormone_Therapy           39998 non-null  float64
 7   Binary_Indicator                         39998 non-null  float64
 8   History_Of_Breast_Biopsy                 39998 non-null  float64
 9   Is_Film_Or_Digital_Mammogram             39998 non-null  int64  
dtypes: float64(5), int64(5)
memory usage: 4.6 MB
```

In [21]:
```
X1=features
y=target
```

In [22]:
```
column_names=features_list
```

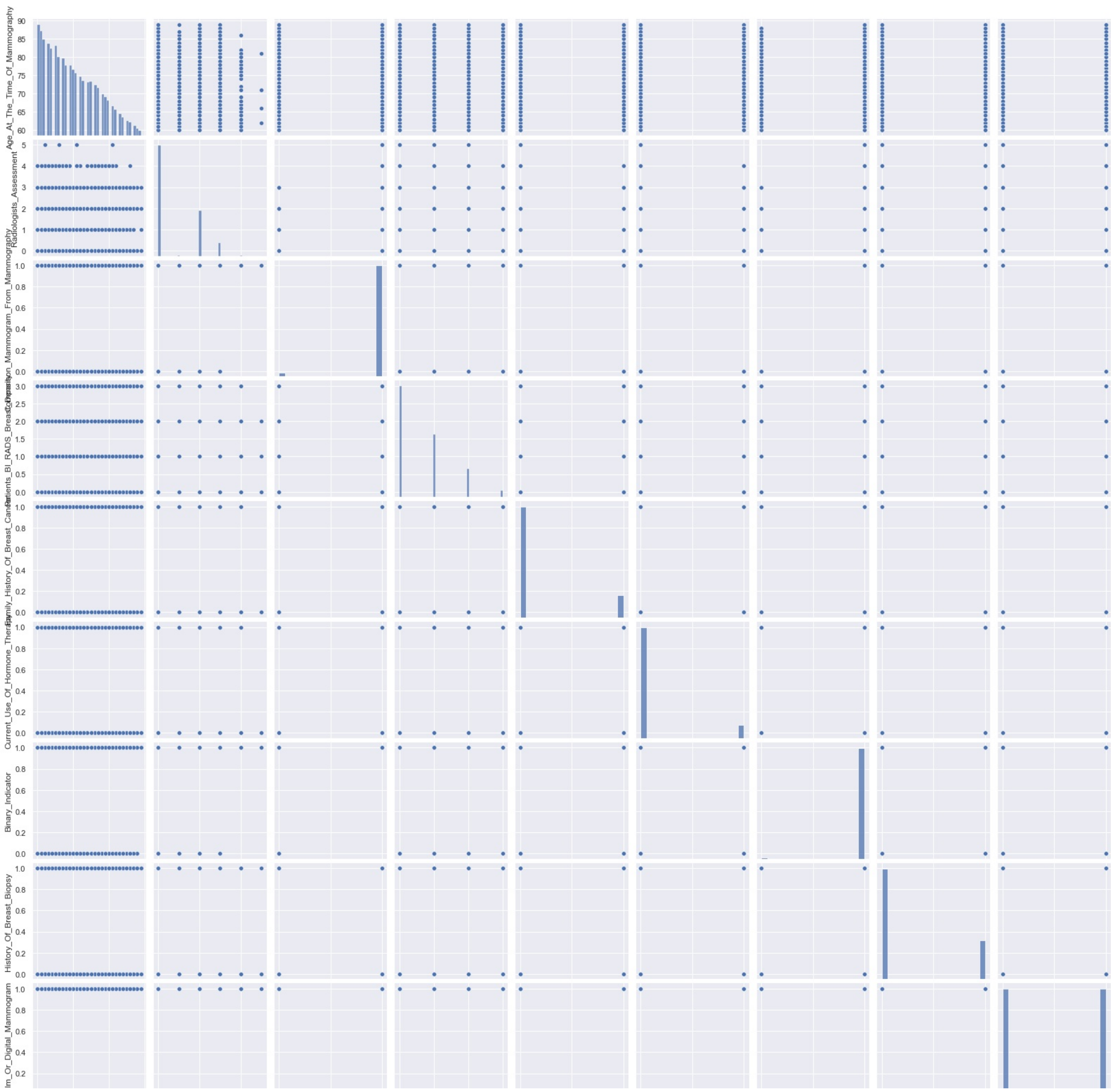
In [23]:
```
sns.set(rc={'figure.figsize':(14,10)})
sns.heatmap(X1.corr(),annot=True)
```

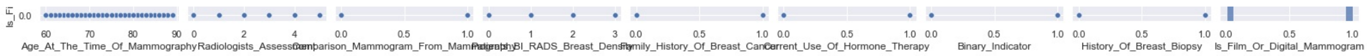
Out[23]: <AxesSubplot:>



In [24]:
```
sns.pairplot(X1)
```

Out[24]: <seaborn.axisgrid.PairGrid at 0x288e5fdb910>





Analysis:

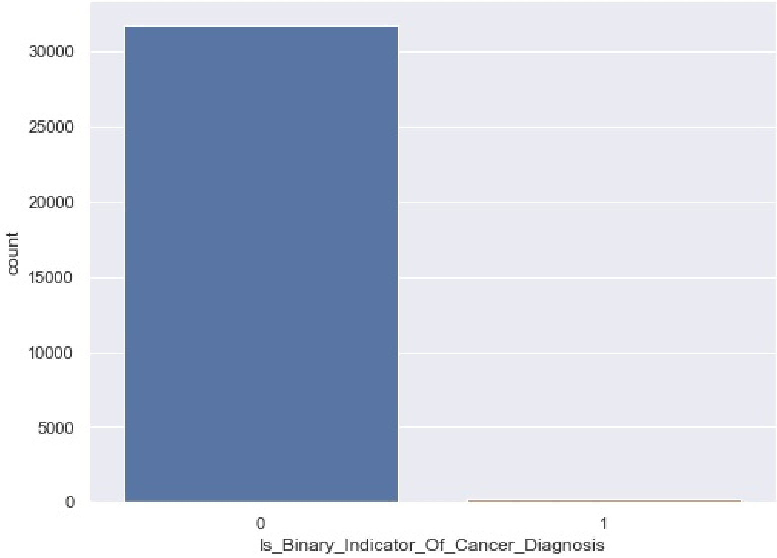
Performing Stratified Train - Test Split:

In [25]:
```
from sklearn.model_selection import train_test_split
```

In [26]:
```
X1_train,X_test,y1_train,y_test=train_test_split(X1,y,test_size=0.20,random_state=42,stratify=y)
```

In [27]:
```
sns.set(rc={'figure.figsize':(8,6)})
sns.countplot(y1_train)
```

Out[27]:
```
<AxesSubplot:xlabel='Is_Binary_Indicator_Of_Cancer_Diagnosis', ylabel='count'>
```



In [28]:
```
y1_train.value_counts()
```

Out[28]:
```
0    31791
1      207
Name: Is_Binary_Indicator_Of_Cancer_Diagnosis, dtype: int64
```

It can be seen that the Target Classes are heavily imbalanced, such that No Cancer Diagnosis (0) class accounts for about >99% of the cases.

This can lead to biasing of ML models towards majority class, hence we need to either Oversample the minority class or Undersample the majority class.

Performing Oversampling of minority class using ADASYN algorithm (Adaptive Synthetic Sampling Approach):

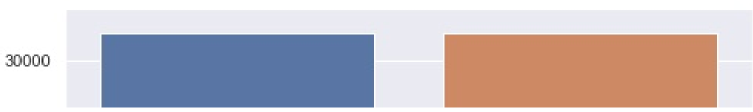
It expands on the procedure of SMOTE, by shifting the importance of the classification boundary to those minority classes which are difficult.

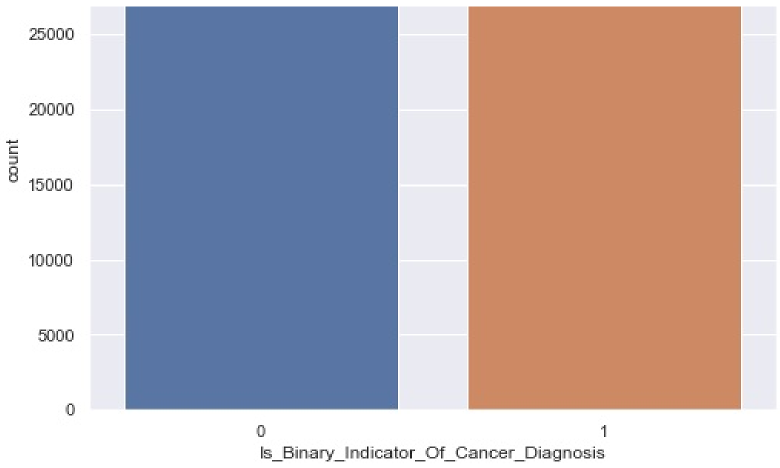
In [29]:
```
from imblearn.over_sampling import ADASYN
```

In [30]:
```
ada=ADASYN(random_state=42)
X2_train, y2_train = ada.fit_resample(X1_train, y1_train)
```

In [31]:
```
sns.set(rc={'figure.figsize':(8,6)})
sns.countplot(y2_train)
```

Out[31]:
```
<AxesSubplot:xlabel='Is_Binary_Indicator_Of_Cancer_Diagnosis', ylabel='count'>
```





In [32]:
```
y2_train.value_counts()
```

Out[32]:
```
1    31840
0    31791
Name: Is_Binary_Indicator_Of_Cancer_Diagnosis, dtype: int64
```

Clearly both the classes have been balanced using the ADASYN technique.

However, it must be noted since we have over sampled the minority class from about ~250 cases to ~31k cases for the training data, hence we are bound to face some degree of irreducible error due to generation of such large amount of data from very small amount of data.

Note: We will be using the original data as well as oversampled data for comparison and analysis of models.

In [33]:
```
sns.set(rc={'figure.figsize':(10,8)})
```

Scaling of Values for distance based algorithms:

In [34]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [35]:
```
scaler1=MinMaxScaler()
scaler2=MinMaxScaler()
X1_train=scaler1.fit_transform(X1_train)
X1_test=scaler1.transform(X_test)

X2_train=scaler2.fit_transform(X2_train)
X2_test=scaler2.transform(X_test)
```

Metrics to Analyse ML Models:

1. Recall Score for 'Cancer Diagnosis (1)' Class:

2. ROC Curve

We want to predict True Positive 'Cancer Diagnosis (1)' Cases, hence we want to minimise Type II Error. This can be achieved by maximising the Recall Score and AUC.

In [36]:
```
from sklearn.metrics import classification_report,recall_score,roc_curve,confusion_matrix
```

In [37]:
```
def report(X_train,X_test,y_train,y_test,y_train_predict,y_test_predict):
    print("Training Report:")
    rep1=classification_report(y_train,y_train_predict)
    print(rep1)
    print('Recall Score:',end=' ')
    print(recall_score(y_train,y_train_predict,average=None))
    print('')
    print('Testing Report:')
    rep2=classification_report(y_test,y_test_predict)
    print(rep2)
    print('Recall Score:',end=' ')
    print(recall_score(y_test,y_test_predict,average=None))
    print('')
    print('Confusion Matrix:')
    print(pd.DataFrame(confusion_matrix(y_test,  y_test_predict)))
```

```
    fpr, tpr, _ = roc_curve(y_test,  y_test_predict)
    #create ROC curve
    plt.plot(fpr,tpr)
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

Logistic Regression:

1. No Regularisation:

In [38]:
```
from sklearn.linear_model import LogisticRegression
```

In [39]:
```
lr_model=LogisticRegression(solver='liblinear',random_state=42,n_jobs=-1)
```

Original Data:

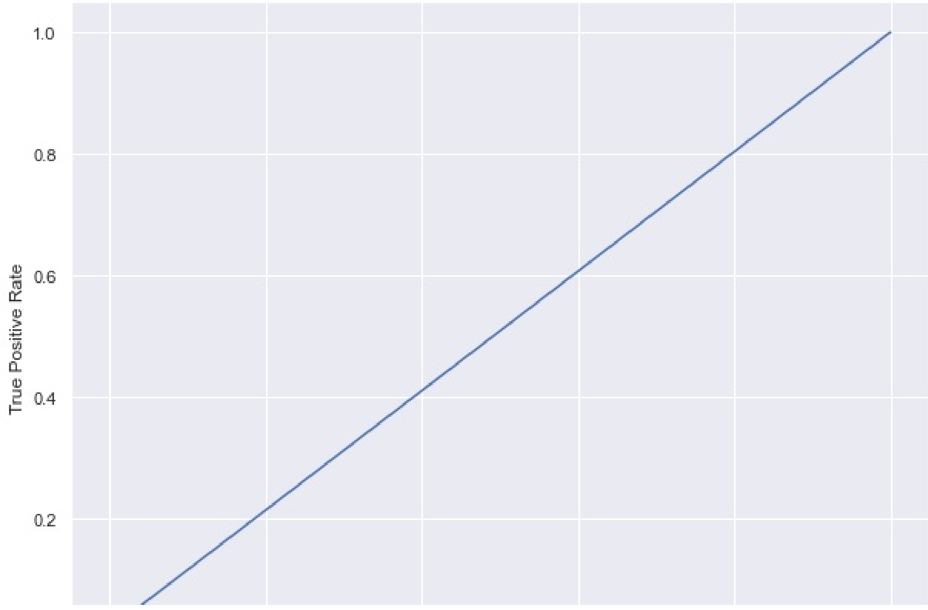
In [40]:
```
lr_model.fit(X1_train,y1_train)
```

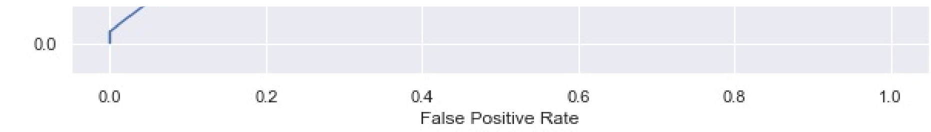
Out[40]:
```
LogisticRegression(n_jobs=-1, random_state=42, solver='liblinear')
```

In [41]:
```
y1_train_predict=lr_model.predict(X1_train)
y1_test_predict=lr_model.predict(X1_test)
```

In [42]:
```
report(X1_train,X1_test,y1_train,y_test,y1_train_predict,y1_test_predict)
```

```
Training Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     31791
           1       0.75      0.01      0.03       207

    accuracy                           0.99     31998
   macro avg       0.87      0.51      0.51     31998
weighted avg       0.99      0.99      0.99     31998

Recall Score: [0.99996854 0.01449275]

Testing Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      7948
           1       1.00      0.02      0.04        52

    accuracy                           0.99      8000
   macro avg       1.00      0.51      0.52      8000
weighted avg       0.99      0.99      0.99      8000

Recall Score: [1.         0.01923077]

Confusion Matrix:
      0  1
0  7948  0
1    51  1
```





Poor and unnacceptable metrics: Model is able to predict 'No Cancer Diagnosis (0)' with high precision and recall as it is the majority class and thus gets biased towards it. However we wish to achieve high recall for 'Cancer Diagnosis (1)' Class which is nearly 0 for this model and hence this model is not acceptable at all.

Oversampled Data:

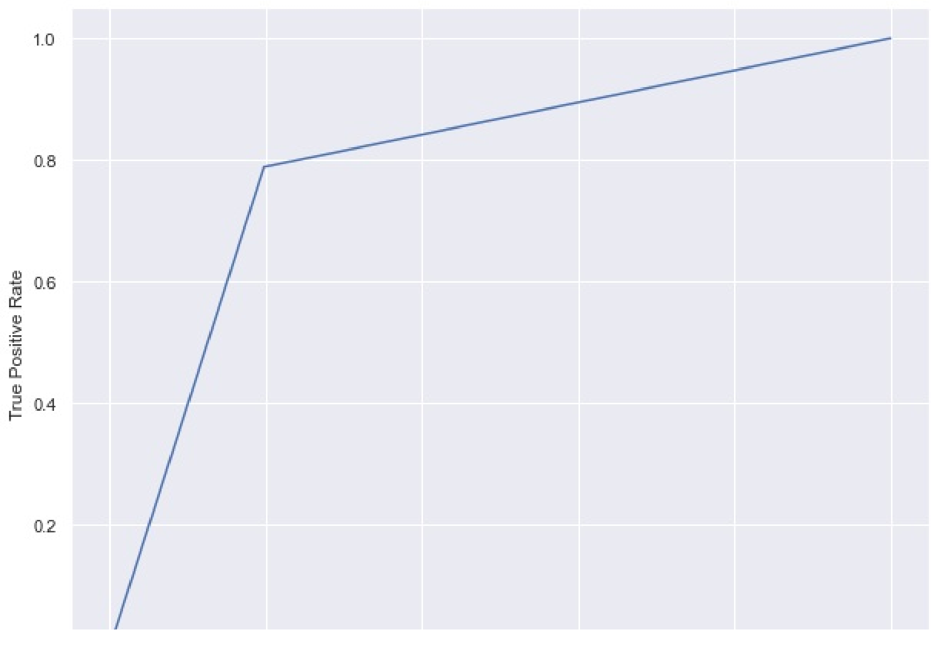
In [43]:
```
lr_model.fit(X2_train,y2_train)
```

Out[43]:
```
LogisticRegression(n_jobs=-1, random_state=42, solver='liblinear')
```

In [44]:
```
y2_train_predict=lr_model.predict(X2_train)
y2_test_predict=lr_model.predict(X2_test)
```

In [45]:
```
report(X2_train,X2_test,y2_train,y_test,y2_train_predict,y2_test_predict)
```

```
Training Report:
              precision    recall  f1-score   support

           0       0.91      0.81      0.85     31791
           1       0.83      0.92      0.87     31840

    accuracy                           0.86     63631
   macro avg       0.87      0.86      0.86     63631
weighted avg       0.87      0.86      0.86     63631

Recall Score: [0.80758705 0.91815327]

Testing Report:
              precision    recall  f1-score   support

           0       1.00      0.80      0.89      7948
           1       0.03      0.79      0.05        52

    accuracy                           0.80      8000
   macro avg       0.51      0.80      0.47      8000
weighted avg       0.99      0.80      0.88      8000

Recall Score: [0.80259185 0.78846154]

Confusion Matrix:
      0     1
0  6379  1569
1    11    41
```



Decent metrics: ~79% Recall is decent and is a great improvement over the recall score corresponding the model for original data. Recall for the training data is ~92% which is also quite promising.

In [46]:
```
from sklearn.linear_model import LogisticRegressionCV
```

2. L1 Regularisation:

In [47]:
```
lr_l1=LogisticRegressionCV(Cs=30,penalty='l1',cv=6,solver='liblinear',random_state=42,scoring='recall')
```

Oversampled Data:

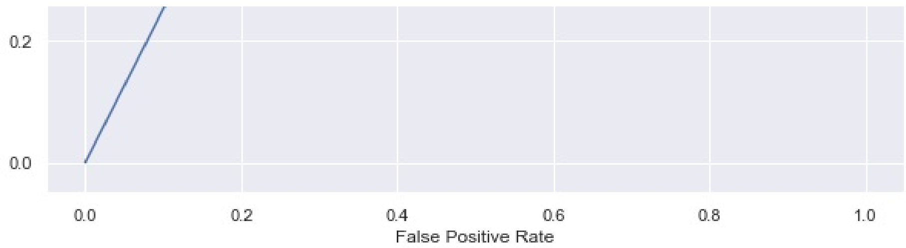
In [48]:
```
lr_l1.fit(X2_train,y2_train)
```

Out[48]:
```
LogisticRegressionCV(Cs=30, cv=6, penalty='l1', random_state=42,
                     scoring='recall', solver='liblinear')
```

In [49]:
```
y2_train_predict=lr_l1.predict(X2_train)
y2_test_predict=lr_l1.predict(X2_test)
```

In [50]:
```
report(X2_train,X2_test,y2_train,y_test,y2_train_predict,y2_test_predict)
```

```
Training Report:
              precision    recall  f1-score   support

           0       0.90      0.66      0.76     31791
           1       0.73      0.93      0.82     31840

    accuracy                           0.79     63631
   macro avg       0.82      0.79      0.79     63631
weighted avg       0.82      0.79      0.79     63631

Recall Score: [0.65779623 0.93040201]

Testing Report:
              precision    recall  f1-score   support

           0       1.00      0.66      0.79      7948
           1       0.02      0.87      0.03        52

    accuracy                           0.66      8000
   macro avg       0.51      0.76      0.41      8000
weighted avg       0.99      0.66      0.79      8000

Recall Score: [0.65752391 0.86538462]

Confusion Matrix:
      0     1
0  5226  2722
1     7    45
```



Good and Acceptable metrics: ~87% Recall is good and is an improvement over the recall score corresponding to unregualrised model for oversampled data. Recall score for the training data is ~93% which is also quite promising.

3. L2 Regularisation:

In [51]:
```
lr_l2=LogisticRegressionCV(Cs=30,penalty='l2',cv=6,solver='liblinear',random_state=42,scoring='recall')
```

Oversampled Data:

In [52]:
```
lr_l2.fit(X2_train,y2_train)
```

Out[52]:
```
LogisticRegressionCV(Cs=30, cv=6, random_state=42, scoring='recall',
                     solver='liblinear')
```

In [53]:
```
y2_train_predict=lr_l2.predict(X2_train)
y2_test_predict=lr_l2.predict(X2_test)
```

In [54]:
```
report(X2_train,X2_test,y2_train,y_test,y2_train_predict,y2_test_predict)
```

```
Training Report:
              precision    recall  f1-score   support

           0       0.91      0.77      0.84     31791
           1       0.80      0.93      0.86     31840

    accuracy                           0.85     63631
   macro avg       0.86      0.85      0.85     63631
weighted avg       0.86      0.85      0.85     63631

Recall Score: [0.77477903 0.92578518]

Testing Report:
              precision    recall  f1-score   support

           0       1.00      0.77      0.87      7948
           1       0.02      0.77      0.04        52

    accuracy                           0.77      8000
   macro avg       0.51      0.77      0.46      8000
weighted avg       0.99      0.77      0.86      8000

Recall Score: [0.76962758 0.76923077]

Confusion Matrix:
      0     1
0  6117  1831
1    12    40
```

Decent metrics: ~77% Recall is decent, however not an improvement over the recall score corresponding to the model for original data.

K-Nearest Neighbors:

In [55]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [56]:
```
from sklearn.model_selection import GridSearchCV,StratifiedKFold
```

In [57]:
```
##Using GridSearchCV to perform CV over range of parameters and determine the best set of parameters

ss = StratifiedKFold(n_splits=6, random_state=42,shuffle=True)
n_neighbors_list=np.arange(1,50,2)
parameters = {'n_neighbors':n_neighbors_list,'p':[1,2]}
knn=KNeighborsClassifier()
clf = GridSearchCV(knn, parameters,cv=ss,scoring='recall')
clf.fit(X2_train, y2_train)
clf.best_params_
```

Out[57]:
```
{'n_neighbors': 33, 'p': 2}
```

In [58]:
```
knn_model=KNeighborsClassifier(n_neighbors=clf.best_params_['n_neighbors'],p=clf.best_params_['p'])
```

Original Data:

In [59]:
```
knn_model.fit(X1_train,y1_train)
```

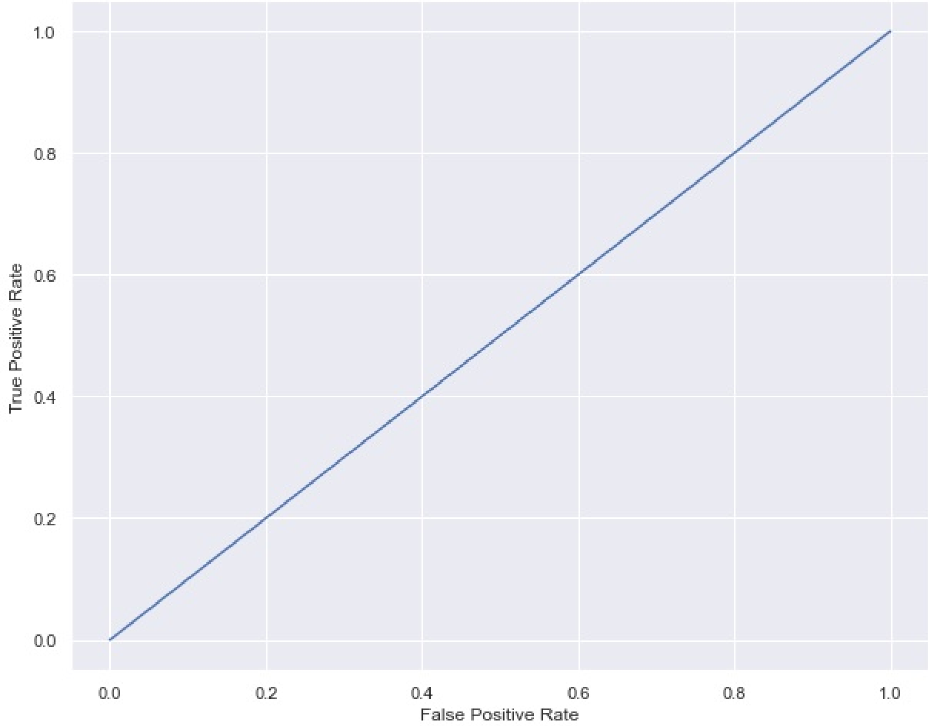
Out[59]:
```
KNeighborsClassifier(n_neighbors=33)
```

In [60]:
```
y1_train_predict=knn_model.predict(X1_train)
y1_test_predict=knn_model.predict(X1_test)
```

In [61]:
```
report(X1_train,X1_test,y1_train,y_test,y1_train_predict,y1_test_predict)
```

```
Training Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     31791
           1       0.00      0.00      0.00       207

    accuracy                           0.99     31998
   macro avg       0.50      0.50      0.50     31998
weighted avg       0.99      0.99      0.99     31998

Recall Score: [1. 0.]

Testing Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      7948
           1       0.00      0.00      0.00        52

    accuracy                           0.99      8000
   macro avg       0.50      0.50      0.50      8000
weighted avg       0.99      0.99      0.99      8000

Recall Score: [1. 0.]

Confusion Matrix:
```

```
      0  1
0  7948  0
1    52  0
```



Poor and un-acceptable metrics: Zero recall score for 'Cancer Diagnosis (1)' Class.

Oversampled Data:

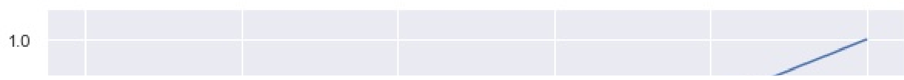
In [62]:
```
knn_model.fit(X2_train,y2_train)
```

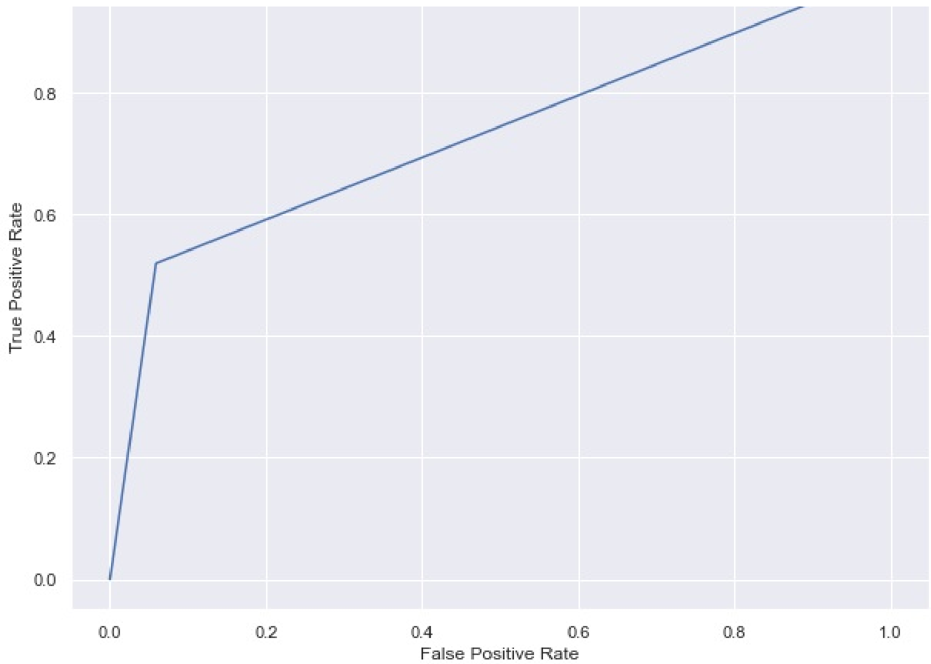
Out[62]:
```
KNeighborsClassifier(n_neighbors=33)
```

In [63]:
```
y2_train_predict=knn_model.predict(X2_train)
y2_test_predict=knn_model.predict(X2_test)
```

In [64]:
```
report(X2_train,X2_test,y2_train,y_test,y2_train_predict,y2_test_predict)
```

```
Training Report:
              precision    recall  f1-score   support

           0       0.96      0.95      0.95     31791
           1       0.95      0.96      0.95     31840

    accuracy                           0.95     63631
   macro avg       0.95      0.95      0.95     63631
weighted avg       0.95      0.95      0.95     63631

Recall Score: [0.94511025 0.96092965]

Testing Report:
              precision    recall  f1-score   support

           0       1.00      0.94      0.97      7948
           1       0.05      0.52      0.10        52

    accuracy                           0.94      8000
   macro avg       0.53      0.73      0.53      8000
weighted avg       0.99      0.94      0.96      8000

Recall Score: [0.94111726 0.51923077]

Confusion Matrix:
      0    1
0  7480  468
1    25   27
```





Unnacceptable metrics: There is an improvement in Recall score as compared to model trained with original data, however it is still not acceptable.

Linear SVM:

In [65]:
```
from sklearn.svm import LinearSVC
from sklearn import svm
```

In [66]:
```
LSVC=LinearSVC()
```

Original Data:

In [67]:
```
LSVC.fit(X1_train,y1_train)
```

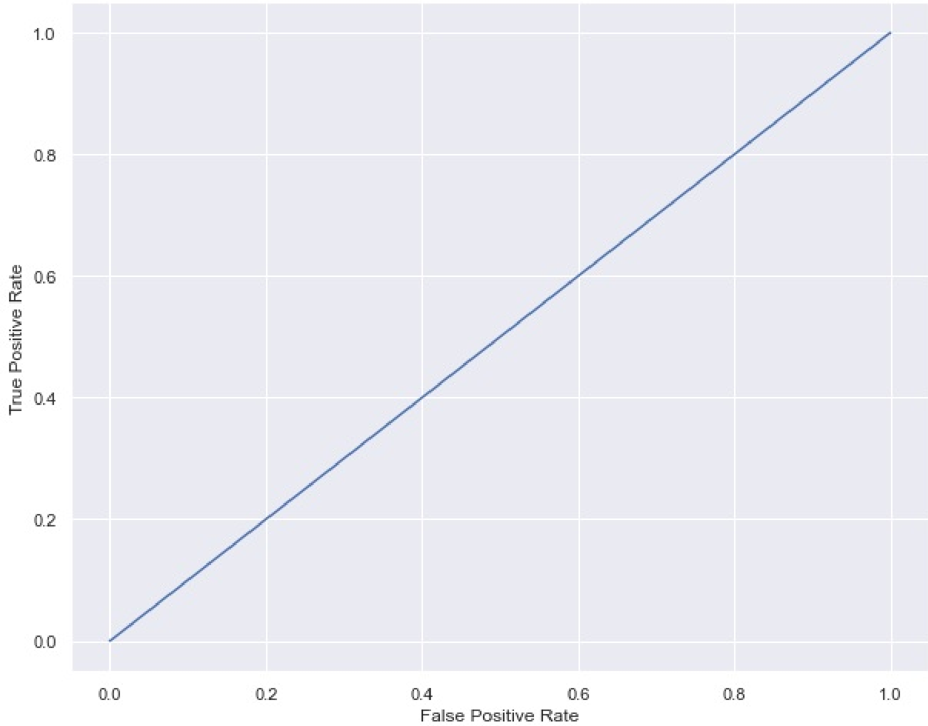
Out[67]:
```
LinearSVC()
```

In [68]:
```
y1_train_predict=LSVC.predict(X1_train)
y1_test_predict=LSVC.predict(X1_test)
```

In [69]:
```
report(X1_train,X1_test,y1_train,y_test,y1_train_predict,y1_test_predict)
```

```
Training Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     31791
           1       0.00      0.00      0.00       207

    accuracy                           0.99     31998
   macro avg       0.50      0.50      0.50     31998
weighted avg       0.99      0.99      0.99     31998

Recall Score: [1. 0.]

Testing Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      7948
           1       0.00      0.00      0.00        52

    accuracy                           0.99      8000
   macro avg       0.50      0.50      0.50      8000
weighted avg       0.99      0.99      0.99      8000

Recall Score: [1. 0.]

Confusion Matrix:
      0  1
```

```
0  7948  0
1    52  0
```



Poor and un-acceptable metrics: Zero recall score for 'Cancer Diagnosis (1)' Class.

Oversampled Data:

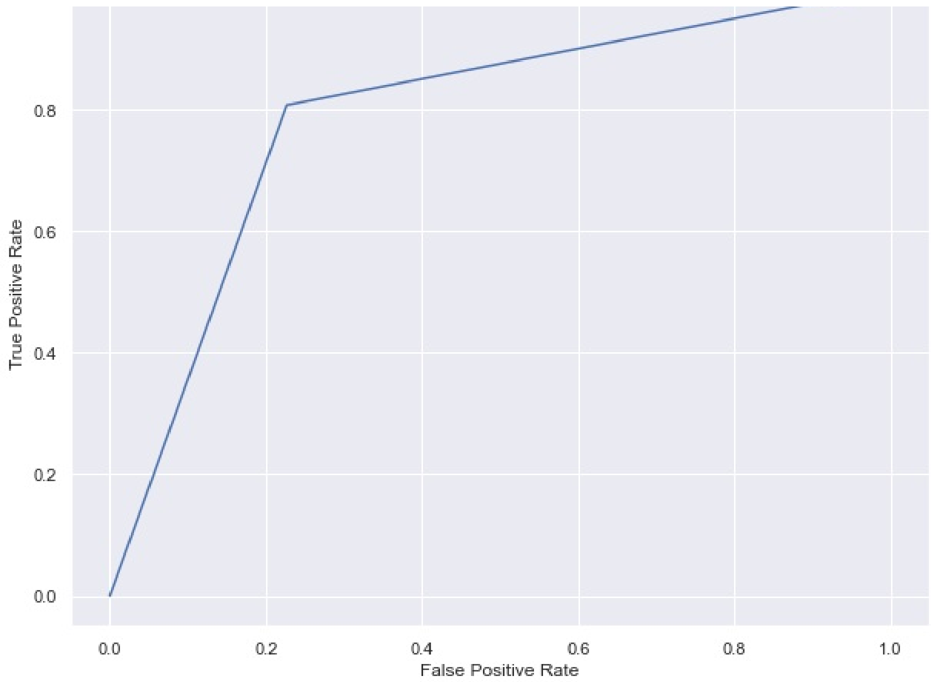
In [70]:
```
LSVC.fit(X2_train,y2_train)
```

Out[70]:
```
LinearSVC()
```

In [71]:
```
y2_train_predict=LSVC.predict(X2_train)
y2_test_predict=LSVC.predict(X2_test)
```

In [72]:
```
report(X2_train,X2_test,y2_train,y_test,y2_train_predict,y2_test_predict)
```

```
Training Report:
              precision    recall  f1-score   support

           0       0.91      0.78      0.84     31791
           1       0.81      0.92      0.86     31840

    accuracy                           0.85     63631
   macro avg       0.86      0.85      0.85     63631
weighted avg       0.86      0.85      0.85     63631

Recall Score: [0.77751565 0.92201633]

Testing Report:
              precision    recall  f1-score   support

           0       1.00      0.77      0.87      7948
           1       0.02      0.81      0.04        52

    accuracy                           0.77      8000
   macro avg       0.51      0.79      0.46      8000
weighted avg       0.99      0.77      0.87      8000

Recall Score: [0.7732763  0.80769231]

Confusion Matrix:
      0     1
0  6146  1802
1    10    42
```



Decent metrics: ~81% Recall is decent and is a great improvement over the recall score corresponding the model for original data. Recall for the training data is ~92% which is also quite promising.

Stacking: Voting Classifier

In [73]:
```
from sklearn.ensemble import VotingClassifier
```

Using Logistic Regression Classifer and Linear SVM via hard voting

In [74]:
```
estimators=[('lr_l1',lr_l1),('linear_svm',LSVC)]
VC=VotingClassifier(estimators,voting='hard')
```

Oversampled Data:

In [75]:
```
VC.fit(X2_train,y2_train)
```

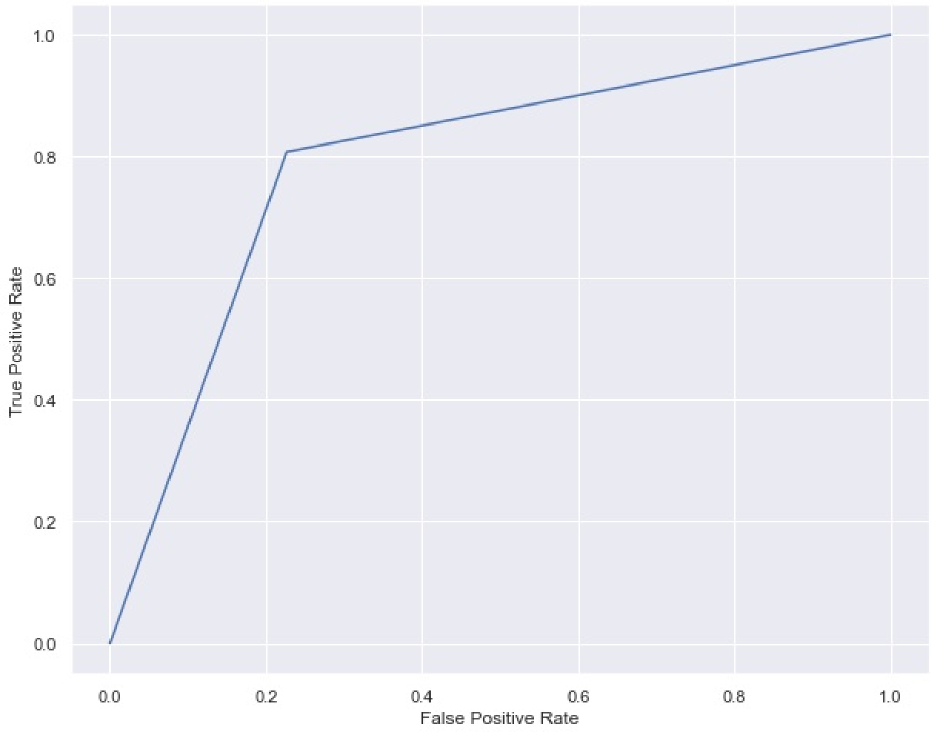
Out[75]:
```
VotingClassifier(estimators=[('lr_l1',
                              LogisticRegressionCV(Cs=30, cv=6, penalty='l1',
                                                   random_state=42,
                                                   scoring='recall',
                                                   solver='liblinear')),
                             ('linear_svm', LinearSVC())])
```

In [76]:
```
y2_train_predict=VC.predict(X2_train)
y2_test_predict=VC.predict(X2_test)
```

In [77]:
```
report(X2_train,X2_test,y2_train,y_test,y2_train_predict,y2_test_predict)
```

```
Training Report:
              precision    recall  f1-score   support

           0       0.91      0.78      0.84     31791
           1       0.81      0.92      0.86     31840

    accuracy                           0.85     63631
   macro avg       0.86      0.85      0.85     63631
weighted avg       0.86      0.85      0.85     63631

Recall Score: [0.77751565 0.92201633]

Testing Report:
              precision    recall  f1-score   support

           0       1.00      0.77      0.87      7948
           1       0.02      0.81      0.04        52
```

```
    accuracy                           0.77      8000
   macro avg       0.51      0.79      0.46      8000
weighted avg       0.99      0.77      0.87      8000

Recall Score: [0.7732763  0.80769231]

Confusion Matrix:
      0     1
0  6146  1802
1    10    42
```



Decent metrics: ~81% Recall is decent. ~92% Recall for the training data is also quite promising.

Synopsis:

Objective: Prediction of Cancer Diagnosis

Here, our objective was to predict whether a person is Diagnosed with Cancer or not. We may make some mistakes in predicting a healthy person as diagnosed with cancer, however we want to minimise the error of predicting a person diagnosed with Cancer as healthy, i.e. We focused on reducing the Type II Error and on maximising the Recall Score.

Data Cleaning and Feature Engineering Techniques Used:

1.Imputation of Missing Data

2.Numerical Encoding of Categorical Data

3.No Outliers as almost entire dataset is categorical in nature, similarly no transformation needed

4.Scaling of Dataset using MinMax Scaler

Models used for Training Data:

1.Logistic Regression:No Regularisation (LR)

2.Logistic Regression:L1 Regularisation (LR_L1)

3.Logistic Regression:L2 Regularisation (LR_L2)

4. K-Nearest Neighbors (KNN)

5. Linear Support Vector Machine (LSVM)

6. Stacking: Voting Classifier (VC)

The Recall Scores corresponding the models are given in the below cell:

In [78]:
```
print('Recall Score for Oversampled Data:')
data = {'Recall Score':['79%','87%','77%','52%','81%','81%']}
labels=['LR','LR_L1','LR_L2','KNN','LSVM','VC']
print(pd.DataFrame(data, index =labels))
```

```
Recall Score for Oversampled Data:
      Recall Score
LR             79%
LR_L1          87%
LR_L2          77%
KNN            52%
LSVM           81%
VC             81%
```

Clearly Logistic Regression with L1 Regularisation is offering best Recall Score (~87%) for the target class. Hence it is most acceptable.

Note: The Classes were severely imbalanced in the original dataset, hence Oversampling techniques have been used so as to prevent biasing. Oversampling techniques do not introduce much variance, hence we are bound to face some irreducible error in our recall score.

Such models could be useful in diagnosing a patient in the absence or un-availability of a Doctor. By training upon more quality data, the model could be improved enough upon so as to be effective enough for commercial use.

PS:

The Analysis can be revisited by conducting more research and improving upon the quality of dataset.

Domain Experts can contacted to execute the Data Cleaning and Feature Engineering tasks more effectively.

Also, the dataset could be trained upon Tree-based models to check for any improvements in Recall Score.

Analysis Conducted by-Aarohan Verma

Third Year Student at TIET, Patiala