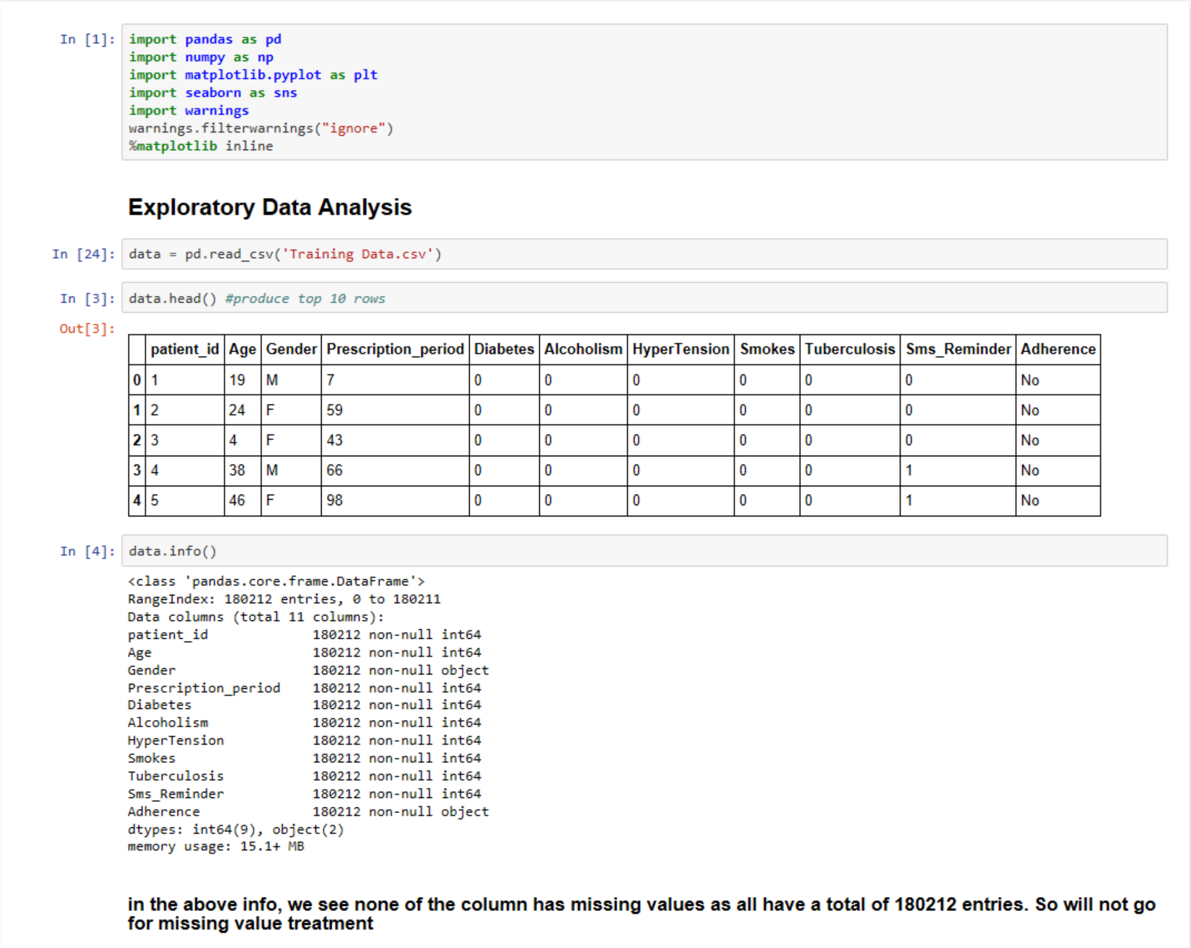


In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

Exploratory Data Analysis

In [24]:
```
data = pd.read_csv('Training Data.csv')
```

In [3]:
```
data.head() #produce top 10 rows
```

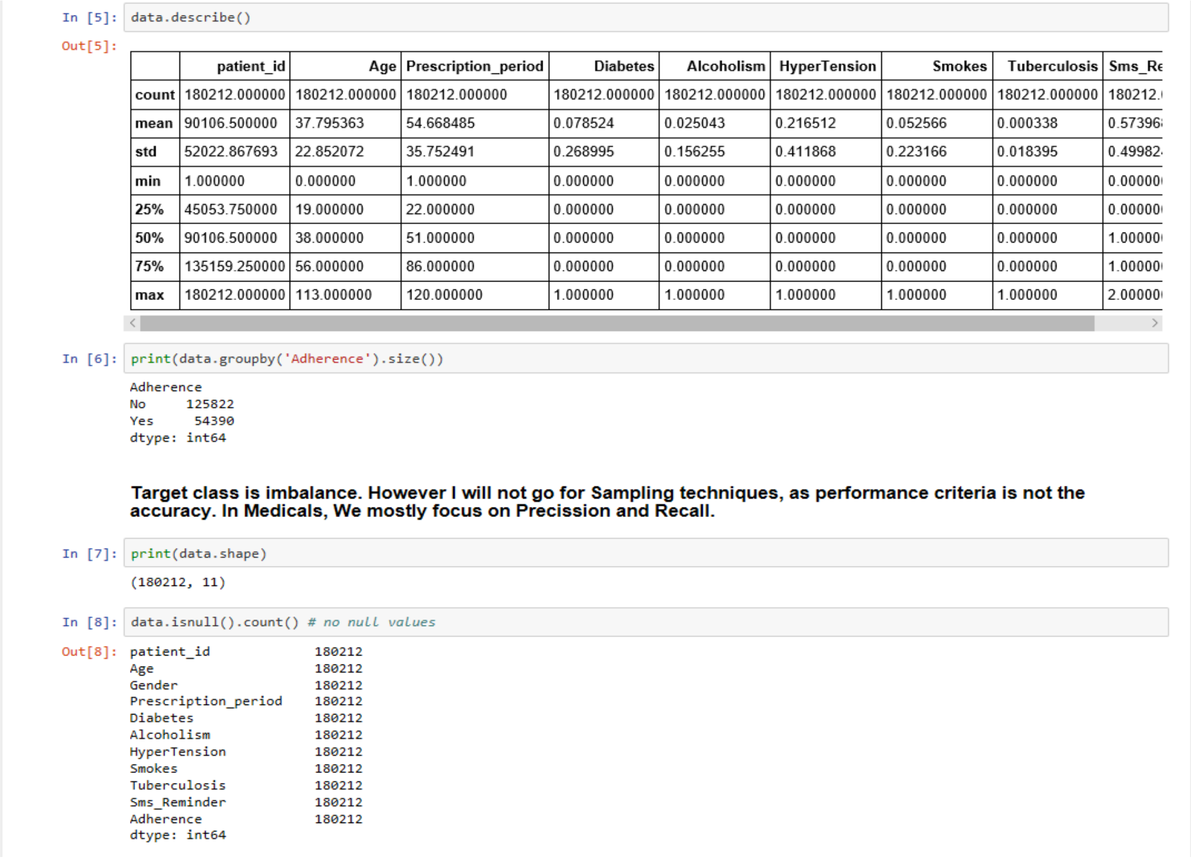
Out[3]:

	patient_id	Age	Gender	Prescription_period	Diabetes	Alcoholism	HyperTension	Smokes	Tuberculosis	Sms_Reminder	Adherence
0	1	19	M	7	0	0	0	0	0	0	No
1	2	24	F	59	0	0	0	0	0	0	No
2	3	4	F	43	0	0	0	0	0	0	No
3	4	38	M	66	0	0	0	0	0	1	No
4	5	46	F	98	0	0	0	0	0	1	No

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180212 entries, 0 to 180211
Data columns (total 11 columns):
patient_id             180212 non-null int64
Age                    180212 non-null int64
Gender                 180212 non-null object
Prescription_period    180212 non-null int64
Diabetes               180212 non-null int64
Alcoholism             180212 non-null int64
HyperTension           180212 non-null int64
Smokes                 180212 non-null int64
Tuberculosis           180212 non-null int64
Sms_Reminder           180212 non-null int64
Adherence              180212 non-null object
dtypes: int64(9), object(2)
memory usage: 15.1+ MB
```

in the above info, we see none of the column has missing values as all have a total of 180212 entries. So will not go for missing value treatment



In [5]:
```
data.describe()
```

Out[5]:

	patient_id	Age	Prescription_period	Diabetes	Alcoholism	HyperTension	Smokes	Tuberculosis	Sms_Re
count	180212.000000	180212.000000	180212.000000	180212.000000	180212.000000	180212.000000	180212.000000	180212.000000	180212.
mean	90106.500000	37.795363	54.668485	0.078524	0.025043	0.216512	0.052566	0.000338	0.57396
std	52022.867693	22.852072	35.752491	0.268995	0.156255	0.411868	0.223166	0.018395	0.49982
min	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	45053.750000	19.000000	22.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
50%	90106.500000	38.000000	51.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.00000
75%	135159.250000	56.000000	86.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.00000
max	180212.000000	113.000000	120.000000	1.000000	1.000000	1.000000	1.000000	1.000000	2.00000

In [6]:
```
print(data.groupby('Adherence').size())
```

```
Adherence
No     125822
Yes     54390
dtype: int64
```

Target class is imbalance. However I will not go for Sampling techniques, as performance criteria is not the accuracy. In Medicals, We mostly focus on Precission and Recall.

In [7]:
```
print(data.shape)
```

```
(180212, 11)
```

In [8]:
```
data.isnull().count() # no null values
```

Out[8]:
```
patient_id             180212
Age                    180212
Gender                 180212
Prescription_period    180212
Diabetes               180212
Alcoholism             180212
HyperTension           180212
Smokes                 180212
Tuberculosis           180212
Sms_Reminder           180212
Adherence              180212
dtype: int64
```

In [9]:
```
data.isna().count() # no missing values
```

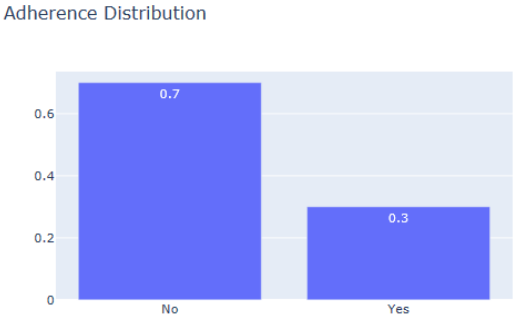
Out[9]:
```
patient_id             180212
Age                    180212
Gender                 180212
Prescription_period    180212
Diabetes               180212
Alcoholism             180212
HyperTension           180212
Smokes                 180212
Tuberculosis           180212
Sms_Reminder           180212
Adherence              180212
dtype: int64
```

Data Visualization

In [31]:
```
#!pip install plotly
```

In [10]:
```
import plotly
import plotly.offline as pyoff
import plotly.figure_factory as ff
from plotly.offline import init_notebook_mode, iplot, plot
import plotly.graph_objs as go
```

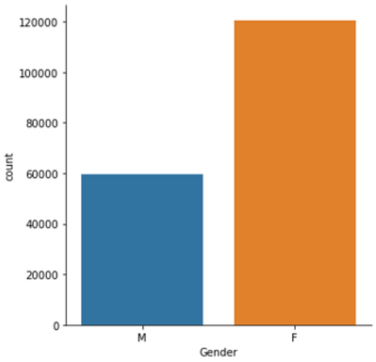
In [11]:
```
temp = data.Adherence.value_counts()
trace = go.Bar(x=temp.index,
               y= np.round(temp.astype(float)/temp.values.sum(),2),
               text = np.round(temp.astype(float)/temp.values.sum(),2),
               textposition = 'auto',
               name = 'Adherence')
data1 = [trace]
layout = go.Layout(
    autosize=False,
    width=600,
    height=400,title = "Adherence Distribution"
)

fig = go.Figure(data=data1, layout=layout)
iplot(fig)
del temp
```



In [12]:
```
print(data.groupby('Gender').size())
print('-------------------------------------------------')
sns.factorplot('Gender',data=data,kind='count')
```

```
Gender
F    120438
M     59774
dtype: int64
-------------------------------------------------
```

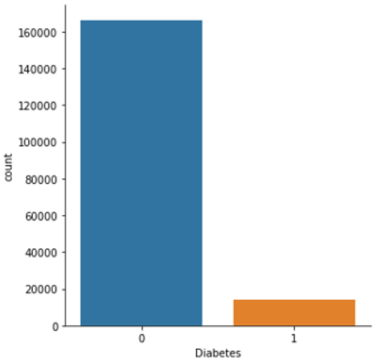
Out[12]:
```
<seaborn.axisgrid.FacetGrid at 0x168788b66d8>
```



```
In [13]: print(data.groupby('Diabetes').size())
         print('--------------------------------------------')
         sns.factorplot('Diabetes',data=data,kind='count')
```

```
Diabetes
0    166061
1     14151
dtype: int64
--------------------------------------------
```

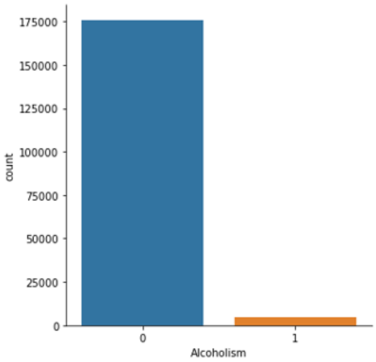
```
Out[13]: <seaborn.axisgrid.FacetGrid at 0x1680e600b70>
```



```
In [14]: print(data.groupby('Alcoholism').size())
         print('---------------------------------------------')
         sns.factorplot('Alcoholism',data=data,kind='count')
```

```
Alcoholism
0    175699
1      4513
dtype: int64
---------------------------------------------
```

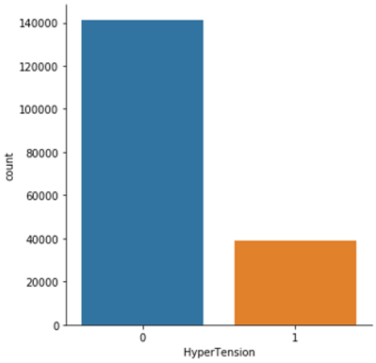
```
Out[14]: <seaborn.axisgrid.FacetGrid at 0x1680c770cc0>
```

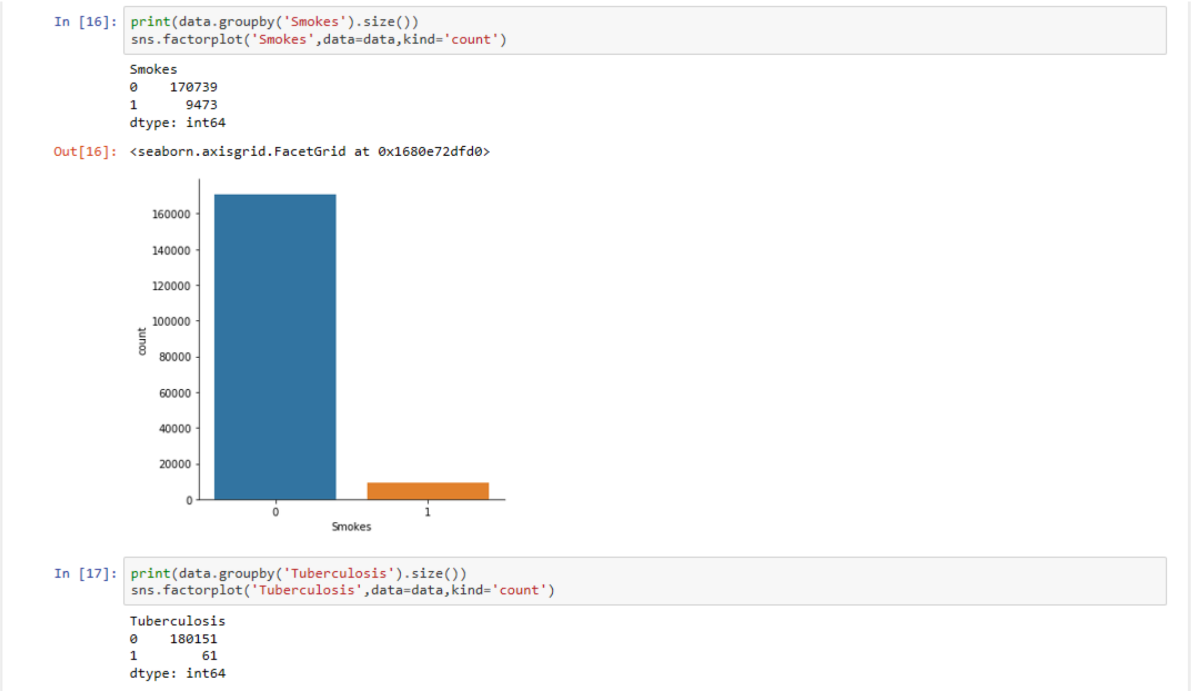


```
In [15]: print(data.groupby('HyperTension').size())

         sns.factorplot('HyperTension',data=data,kind='count')
```

```
HyperTension
0    141194
1     39018
dtype: int64
```

```
Out[15]: <seaborn.axisgrid.FacetGrid at 0x1680e6bf8d0>
```





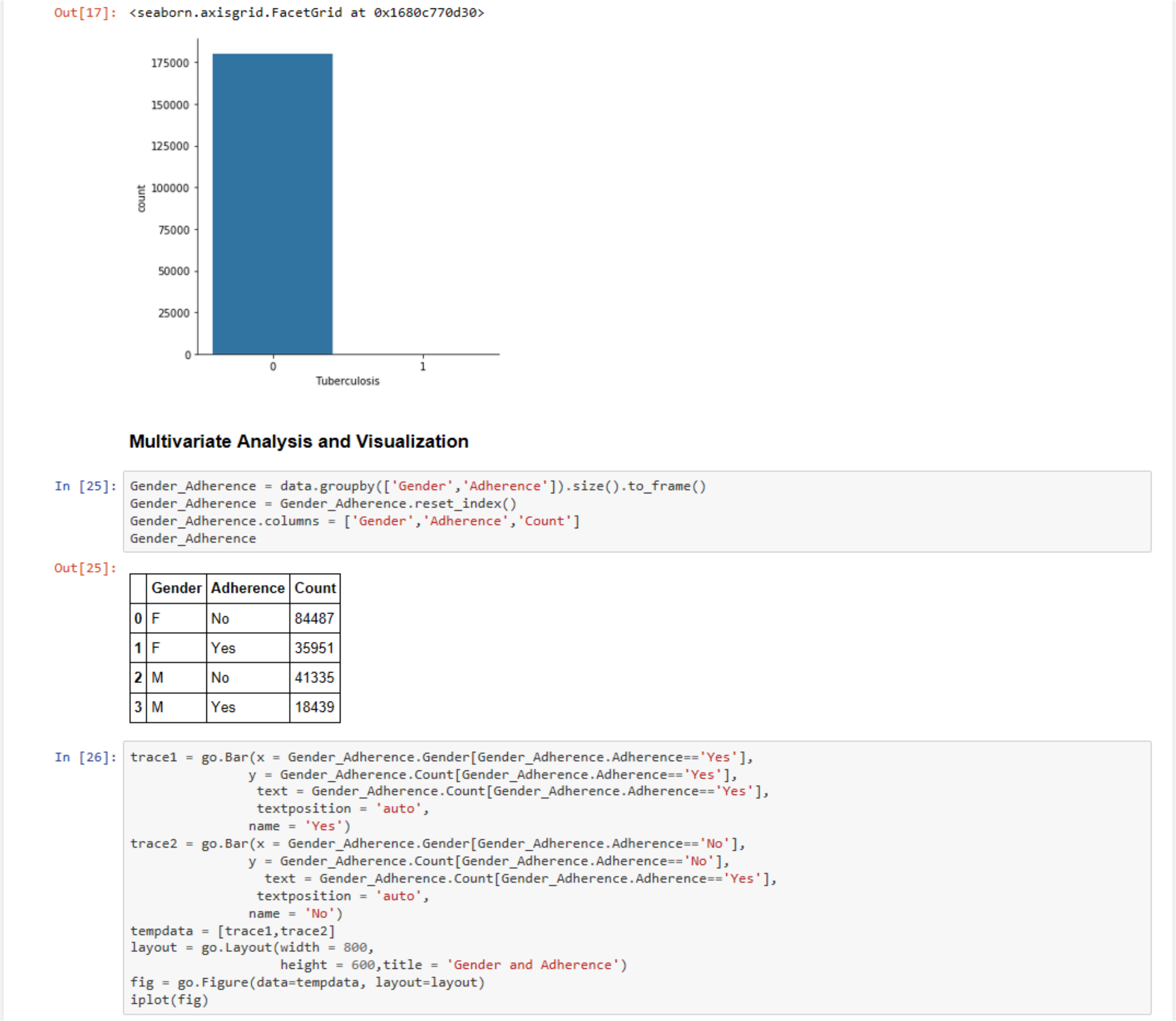
```
In [16]: print(data.groupby('Smokes').size())
         sns.factorplot('Smokes',data=data,kind='count')
```

```
Smokes
0    170739
1      9473
dtype: int64
```

```
Out[16]: <seaborn.axisgrid.FacetGrid at 0x1680e72dfd0>
```

```
In [17]: print(data.groupby('Tuberculosis').size())
         sns.factorplot('Tuberculosis',data=data,kind='count')
```

```
Tuberculosis
0    180151
1        61
dtype: int64
```



```
Out[17]: <seaborn.axisgrid.FacetGrid at 0x1680c770d30>
```

Multivariate Analysis and Visualization

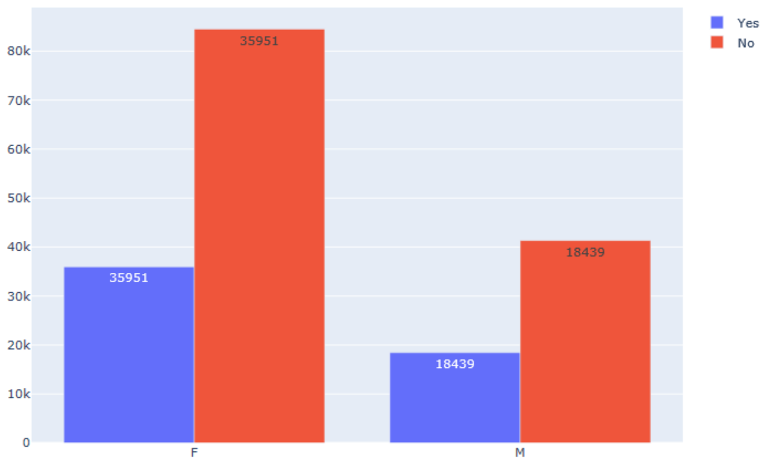
```
In [25]: Gender_Adherence = data.groupby(['Gender','Adherence']).size().to_frame()
         Gender_Adherence = Gender_Adherence.reset_index()
         Gender_Adherence.columns = ['Gender','Adherence','Count']
         Gender_Adherence
```

Out[25]:

	Gender	Adherence	Count
0	F	No	84487
1	F	Yes	35951
2	M	No	41335
3	M	Yes	18439

```
In [26]: trace1 = go.Bar(x = Gender_Adherence.Gender[Gender_Adherence.Adherence=='Yes'],
                        y = Gender_Adherence.Count[Gender_Adherence.Adherence=='Yes'],
                         text = Gender_Adherence.Count[Gender_Adherence.Adherence=='Yes'],
                         textposition = 'auto',
                        name = 'Yes')
         trace2 = go.Bar(x = Gender_Adherence.Gender[Gender_Adherence.Adherence=='No'],
                        y = Gender_Adherence.Count[Gender_Adherence.Adherence=='No'],
                          text = Gender_Adherence.Count[Gender_Adherence.Adherence=='Yes'],
                         textposition = 'auto',
                        name = 'No')
         tempdata = [trace1,trace2]
         layout = go.Layout(width = 800,
                            height = 600,title = 'Gender and Adherence')
         fig = go.Figure(data=tempdata, layout=layout)
         iplot(fig)
```

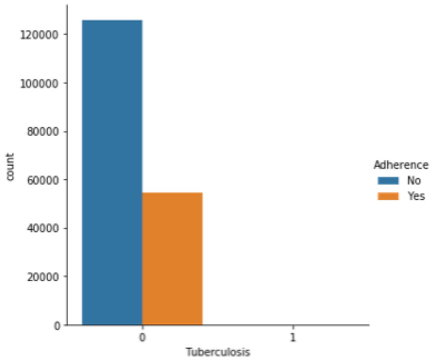
Gender and Adherence



```
In [27]: print(data.groupby(['Tuberculosis','Adherence']).size())
         sns.factorplot('Tuberculosis',data=data,hue='Adherence',kind='count')
```

```
Tuberculosis  Adherence
0             No           125783
              Yes           54368
1             No               39
              Yes              22
dtype: int64
```

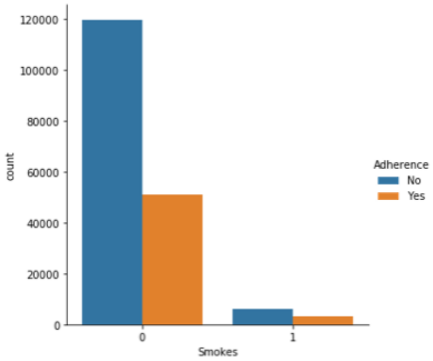
```
Out[27]: <seaborn.axisgrid.FacetGrid at 0x168106f00b8>
```

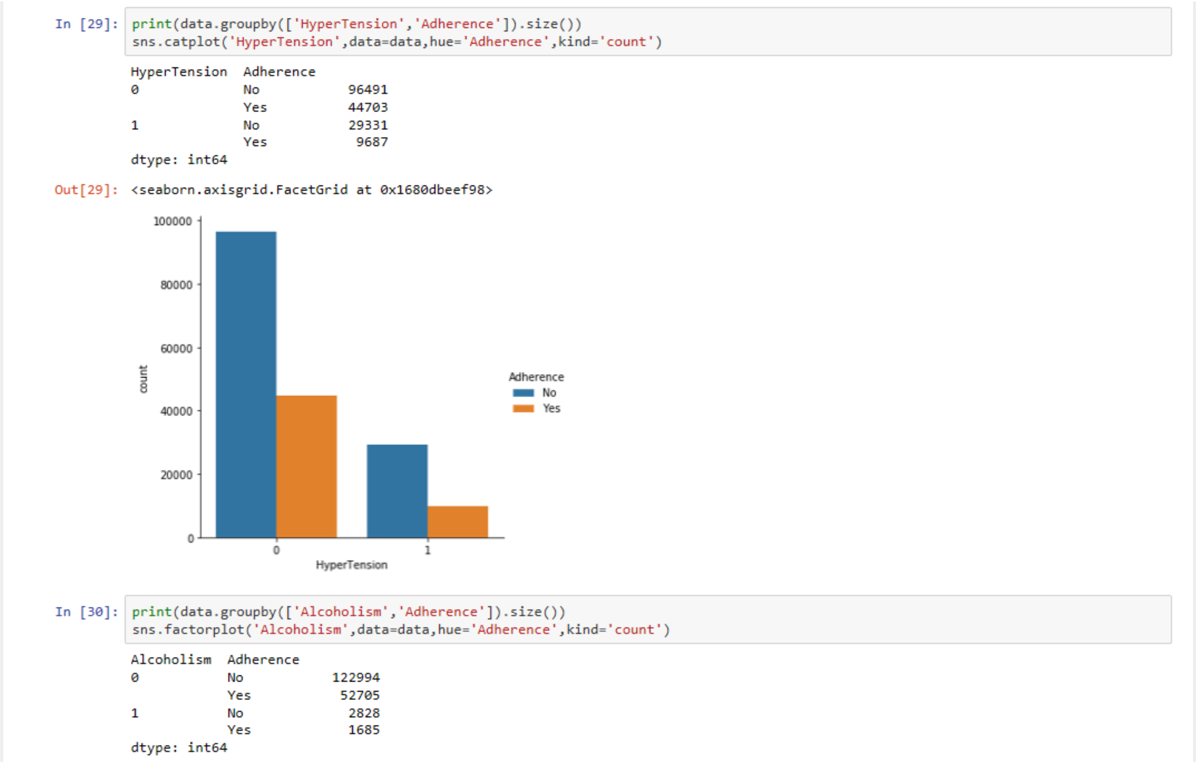


```
In [28]: print(data.groupby(['Smokes','Adherence']).size())
         sns.factorplot('Smokes',data=data,hue='Adherence',kind='count')
```

```
Smokes  Adherence
0       No           119651
        Yes           51088
1       No             6171
        Yes            3302
dtype: int64
```

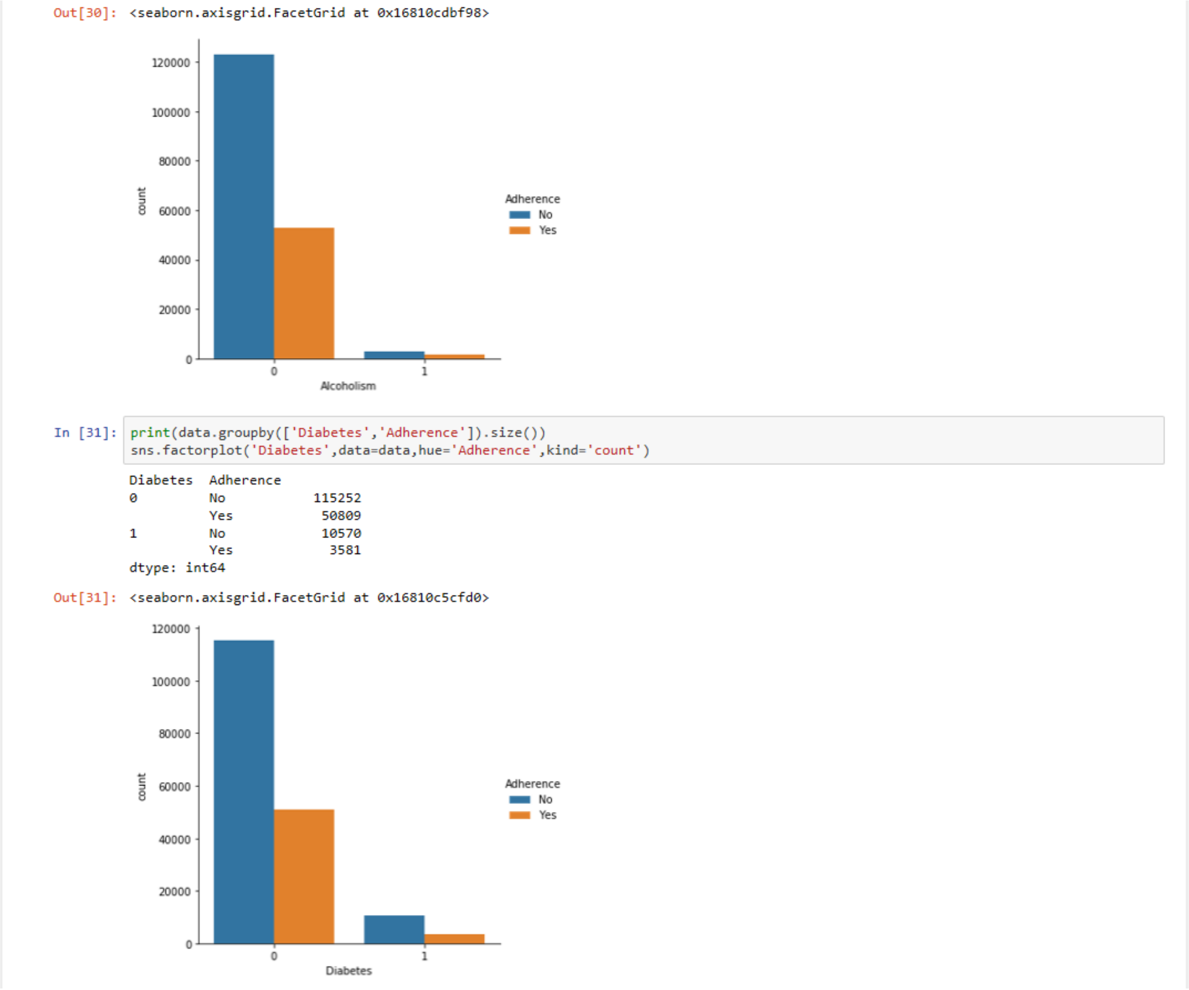
```
Out[28]: <seaborn.axisgrid.FacetGrid at 0x1680e6eeb38>
```





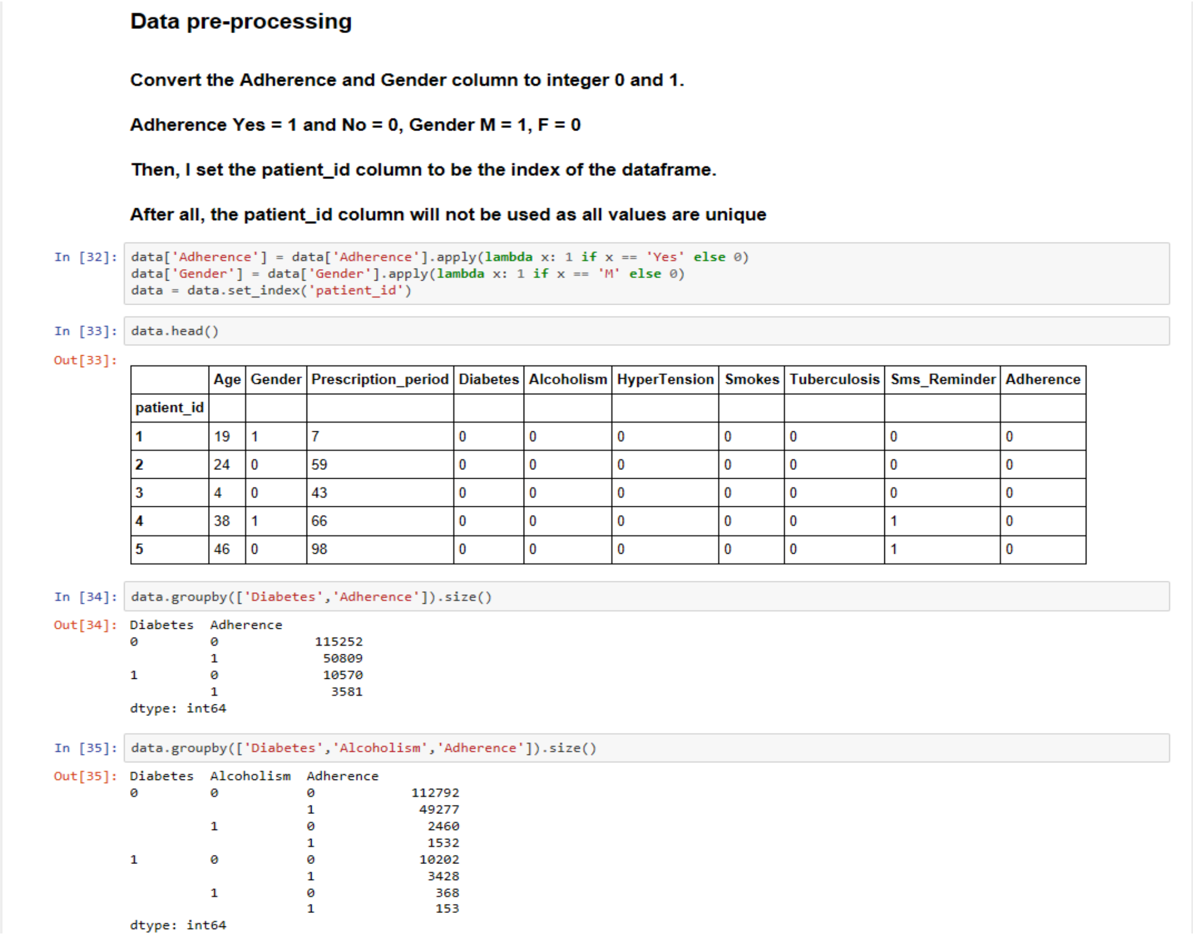
```
In [29]: print(data.groupby(['HyperTension','Adherence']).size())
         sns.catplot('HyperTension',data=data,hue='Adherence',kind='count')

         HyperTension  Adherence
         0             No           96491
                       Yes          44703
         1             No           29331
                       Yes           9687
         dtype: int64

Out[29]: <seaborn.axisgrid.FacetGrid at 0x1680dbeef98>
```

```
In [30]: print(data.groupby(['Alcoholism','Adherence']).size())
         sns.factorplot('Alcoholism',data=data,hue='Adherence',kind='count')

         Alcoholism  Adherence
         0           No           122994
                     Yes           52705
         1           No             2828
                     Yes            1685
         dtype: int64
```



```
Out[30]: <seaborn.axisgrid.FacetGrid at 0x16810cdbf98>
```

```
In [31]: print(data.groupby(['Diabetes','Adherence']).size())
         sns.factorplot('Diabetes',data=data,hue='Adherence',kind='count')

         Diabetes  Adherence
         0         No           115252
                   Yes           50809
         1         No            10570
                   Yes            3581
         dtype: int64

Out[31]: <seaborn.axisgrid.FacetGrid at 0x16810c5cfd0>
```



Data pre-processing

Convert the Adherence and Gender column to integer 0 and 1.

Adherence Yes = 1 and No = 0, Gender M = 1, F = 0

Then, I set the patient_id column to be the index of the dataframe.

After all, the patient_id column will not be used as all values are unique

In [32]:

```
data['Adherence'] = data['Adherence'].apply(lambda x: 1 if x == 'Yes' else 0)
data['Gender'] = data['Gender'].apply(lambda x: 1 if x == 'M' else 0)
data = data.set_index('patient_id')
```

In [33]:

```
data.head()
```

Out[33]:

	Age	Gender	Prescription_period	Diabetes	Alcoholism	HyperTension	Smokes	Tuberculosis	Sms_Reminder	Adherence
patient_id										
1	19	1	7	0	0	0	0	0	0	0
2	24	0	59	0	0	0	0	0	0	0
3	4	0	43	0	0	0	0	0	0	0
4	38	1	66	0	0	0	0	0	1	0
5	46	0	98	0	0	0	0	0	1	0

In [34]:

```
data.groupby(['Diabetes','Adherence']).size()
```

Out[34]:

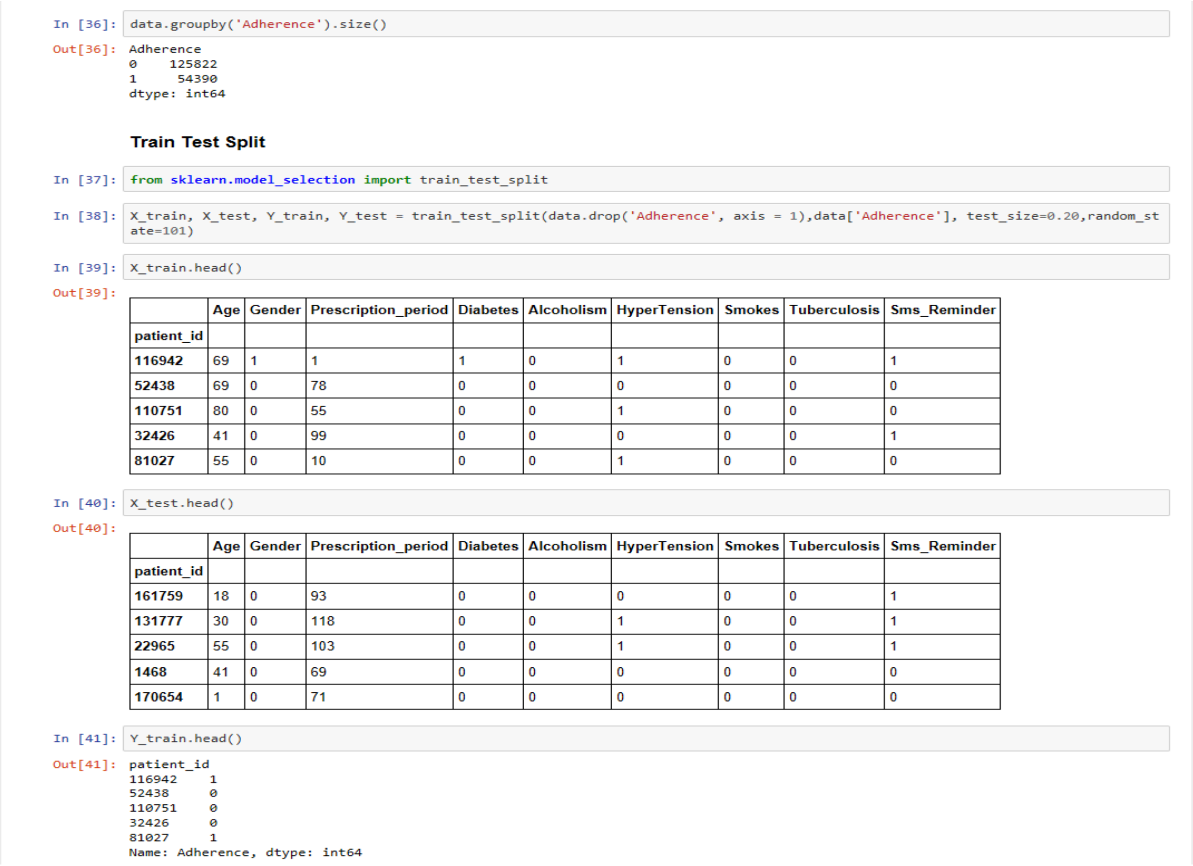
```
Diabetes  Adherence
0         0            115252
          1             50809
1         0             10570
          1              3581
dtype: int64
```

In [35]:

```
data.groupby(['Diabetes','Alcoholism','Adherence']).size()
```

Out[35]:

```
Diabetes  Alcoholism  Adherence
0         0           0            112792
                      1             49277
          1           0              2460
                      1              1532
1         0           0             10202
                      1              3428
          1           0               368
                      1               153
dtype: int64
```



In [36]:

```
data.groupby('Adherence').size()
```

Out[36]:

```
Adherence
0    125822
1     54390
dtype: int64
```

Train Test Split

In [37]:

```
from sklearn.model_selection import train_test_split
```

In [38]:

```
X_train, X_test, Y_train, Y_test = train_test_split(data.drop('Adherence', axis = 1),data['Adherence'], test_size=0.20,random_st
ate=101)
```

In [39]:

```
X_train.head()
```

Out[39]:

	Age	Gender	Prescription_period	Diabetes	Alcoholism	HyperTension	Smokes	Tuberculosis	Sms_Reminder
patient_id									
116942	69	1	1	1	0	1	0	0	1
52438	69	0	78	0	0	0	0	0	0
110751	80	0	55	0	0	1	0	0	0
32426	41	0	99	0	0	0	0	0	1
81027	55	0	10	0	0	1	0	0	0

In [40]:

```
X_test.head()
```

Out[40]:

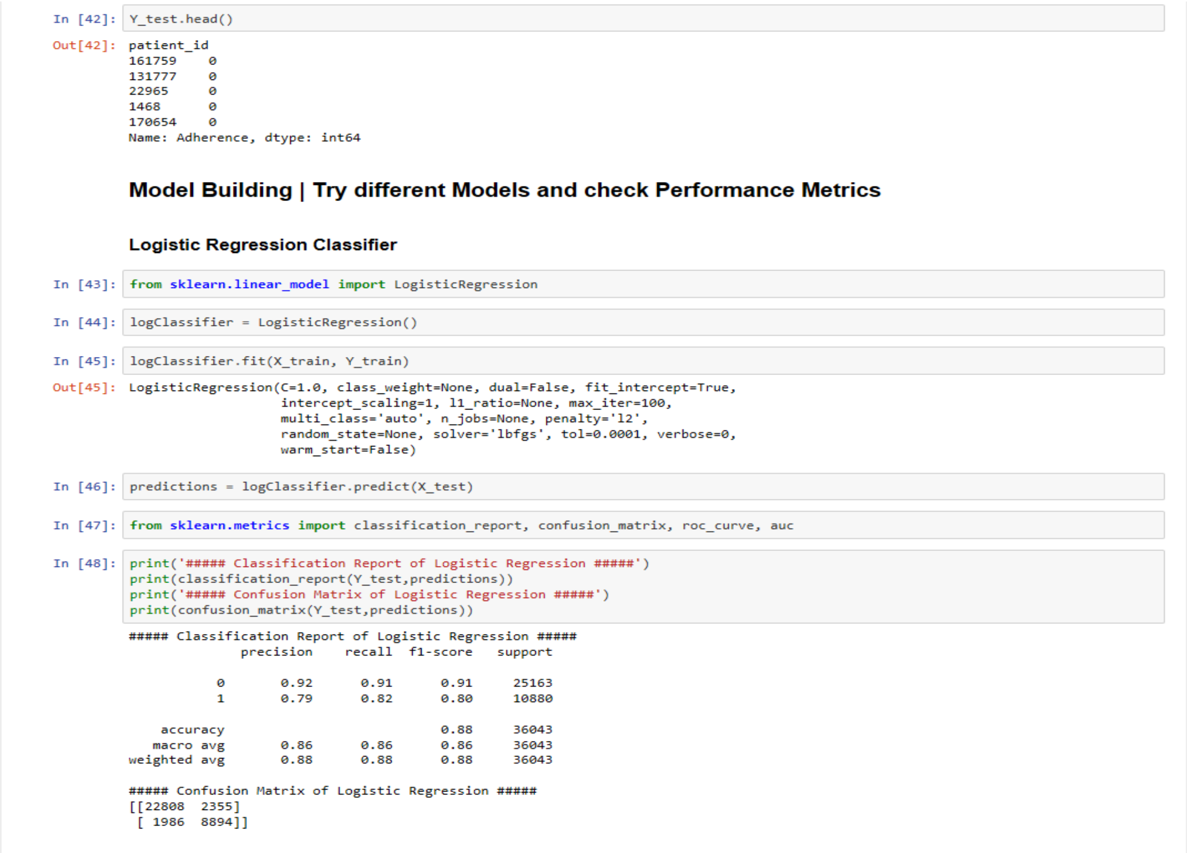
	Age	Gender	Prescription_period	Diabetes	Alcoholism	HyperTension	Smokes	Tuberculosis	Sms_Reminder
patient_id									
161759	18	0	93	0	0	0	0	0	1
131777	30	0	118	0	0	1	0	0	1
22965	55	0	103	0	0	1	0	0	1
1468	41	0	69	0	0	0	0	0	0
170654	1	0	71	0	0	0	0	0	0

In [41]:

```
Y_train.head()
```

Out[41]:

```
patient_id
116942    1
52438     0
110751    0
32426     0
81027     1
Name: Adherence, dtype: int64
```

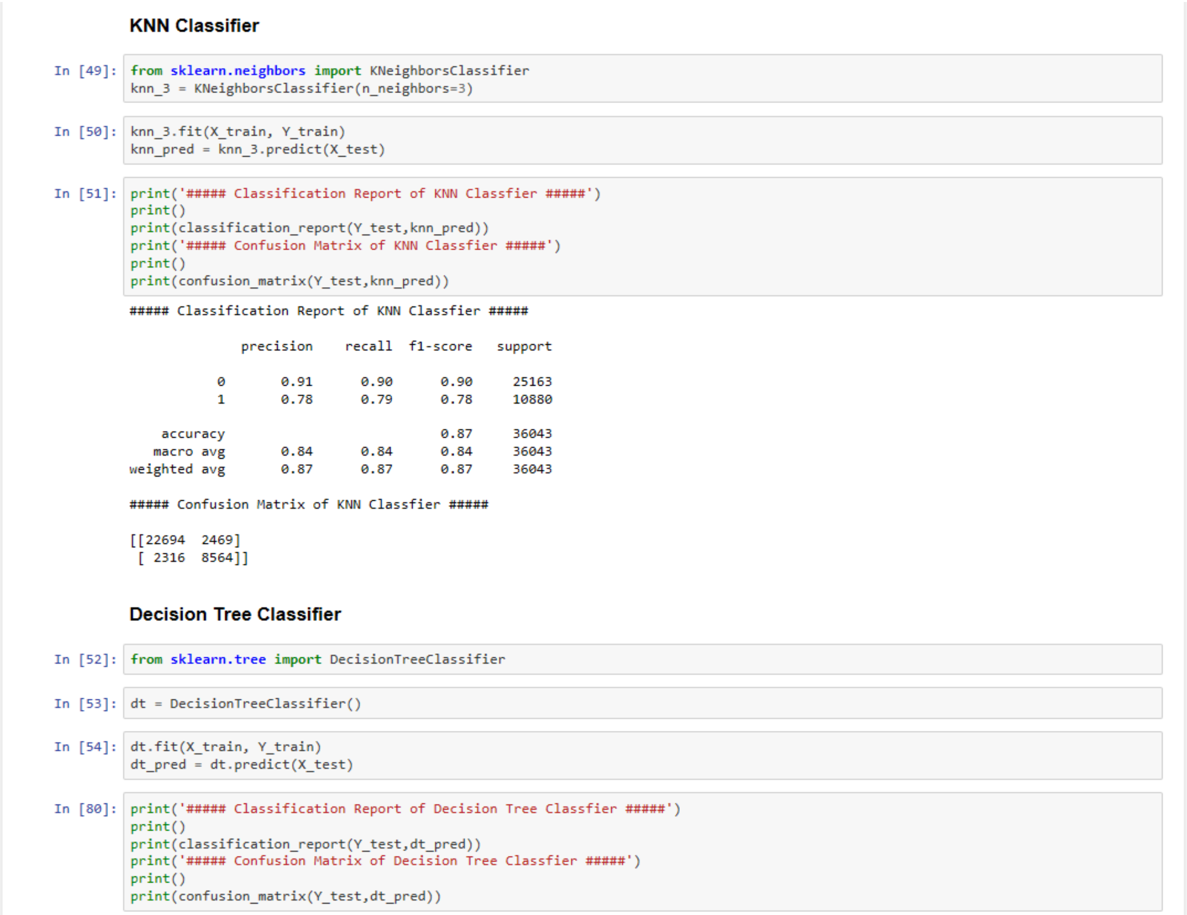


```
In [42]: Y_test.head()

Out[42]: patient_id
         161759    0
         131777    0
         22965     0
         1468      0
         170654    0
         Name: Adherence, dtype: int64
```

Model Building | Try different Models and check Performance Metrics

Logistic Regression Classifier

```
In [43]: from sklearn.linear_model import LogisticRegression

In [44]: logClassifier = LogisticRegression()

In [45]: logClassifier.fit(X_train, Y_train)

Out[45]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=1, l1_ratio=None, max_iter=100,
                            multi_class='auto', n_jobs=None, penalty='l2',
                            random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                            warm_start=False)

In [46]: predictions = logClassifier.predict(X_test)

In [47]: from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc

In [48]: print('##### Classification Report of Logistic Regression #####')
         print(classification_report(Y_test,predictions))
         print('##### Confusion Matrix of Logistic Regression #####')
         print(confusion_matrix(Y_test,predictions))

         ##### Classification Report of Logistic Regression #####
                       precision    recall  f1-score   support

                    0       0.92      0.91      0.91     25163
                    1       0.79      0.82      0.80     10880

             accuracy                           0.88     36043
            macro avg       0.86      0.86      0.86     36043
         weighted avg       0.88      0.88      0.88     36043

         ##### Confusion Matrix of Logistic Regression #####
         [[22808  2355]
          [ 1986  8894]]
```



KNN Classifier

```
In [49]: from sklearn.neighbors import KNeighborsClassifier
         knn_3 = KNeighborsClassifier(n_neighbors=3)

In [50]: knn_3.fit(X_train, Y_train)
         knn_pred = knn_3.predict(X_test)

In [51]: print('##### Classification Report of KNN Classfier #####')
         print()
         print(classification_report(Y_test,knn_pred))
         print('##### Confusion Matrix of KNN Classfier #####')
         print()
         print(confusion_matrix(Y_test,knn_pred))

         ##### Classification Report of KNN Classfier #####

                       precision    recall  f1-score   support

                    0       0.91      0.90      0.90     25163
                    1       0.78      0.79      0.78     10880

             accuracy                           0.87     36043
            macro avg       0.84      0.84      0.84     36043
         weighted avg       0.87      0.87      0.87     36043

         ##### Confusion Matrix of KNN Classfier #####

         [[22694  2469]
          [ 2316  8564]]
```

Decision Tree Classifier

```
In [52]: from sklearn.tree import DecisionTreeClassifier

In [53]: dt = DecisionTreeClassifier()

In [54]: dt.fit(X_train, Y_train)
         dt_pred = dt.predict(X_test)

In [80]: print('##### Classification Report of Decision Tree Classfier #####')
         print()
         print(classification_report(Y_test,dt_pred))
         print('##### Confusion Matrix of Decision Tree Classfier #####')
         print()
         print(confusion_matrix(Y_test,dt_pred))
```

```
##### Classification Report of Decision Tree Classfier #####

              precision    recall  f1-score   support

           0       0.89      0.90      0.90     25163
           1       0.76      0.75      0.76     10880

    accuracy                           0.85     36043
   macro avg       0.83      0.83      0.83     36043
weighted avg       0.85      0.85      0.85     36043

##### Confusion Matrix of Decision Tree Classfier #####

[[22568  2595]
 [ 2684  8196]]
```

Ensemble Learning | Bagging with Random Feature Subspaces that is Random Forest Classifier

```
In [55]: from sklearn.ensemble import RandomForestClassifier
```

```
In [56]: rf = RandomForestClassifier()
```

```
In [57]: rf.fit(X_train, Y_train)
         rf_pred = rf.predict(X_test)
```

```
In [58]: print('##### Classification Report of Random Forest Classfier #####')
         print()
         print(classification_report(Y_test,rf_pred))
         print('##### Confusion Matrix of Random Forest Classfier #####')
         print()
         print(confusion_matrix(Y_test,rf_pred))
```

```
##### Classification Report of Random Forest Classfier #####

              precision    recall  f1-score   support

           0       0.92      0.90      0.91     25163
           1       0.77      0.81      0.79     10880

    accuracy                           0.87     36043
   macro avg       0.84      0.85      0.85     36043
weighted avg       0.87      0.87      0.87     36043

##### Confusion Matrix of Random Forest Classfier #####

[[22534  2629]
 [ 2075  8805]]
```

Ensemble Learning | Improving Week Learners with Boosting

Adaboost Classifier

```
In [59]: from sklearn.ensemble import AdaBoostClassifier
```

```
In [60]: classifier = AdaBoostClassifier(
             DecisionTreeClassifier(max_depth=1),
             n_estimators=200
         )
```

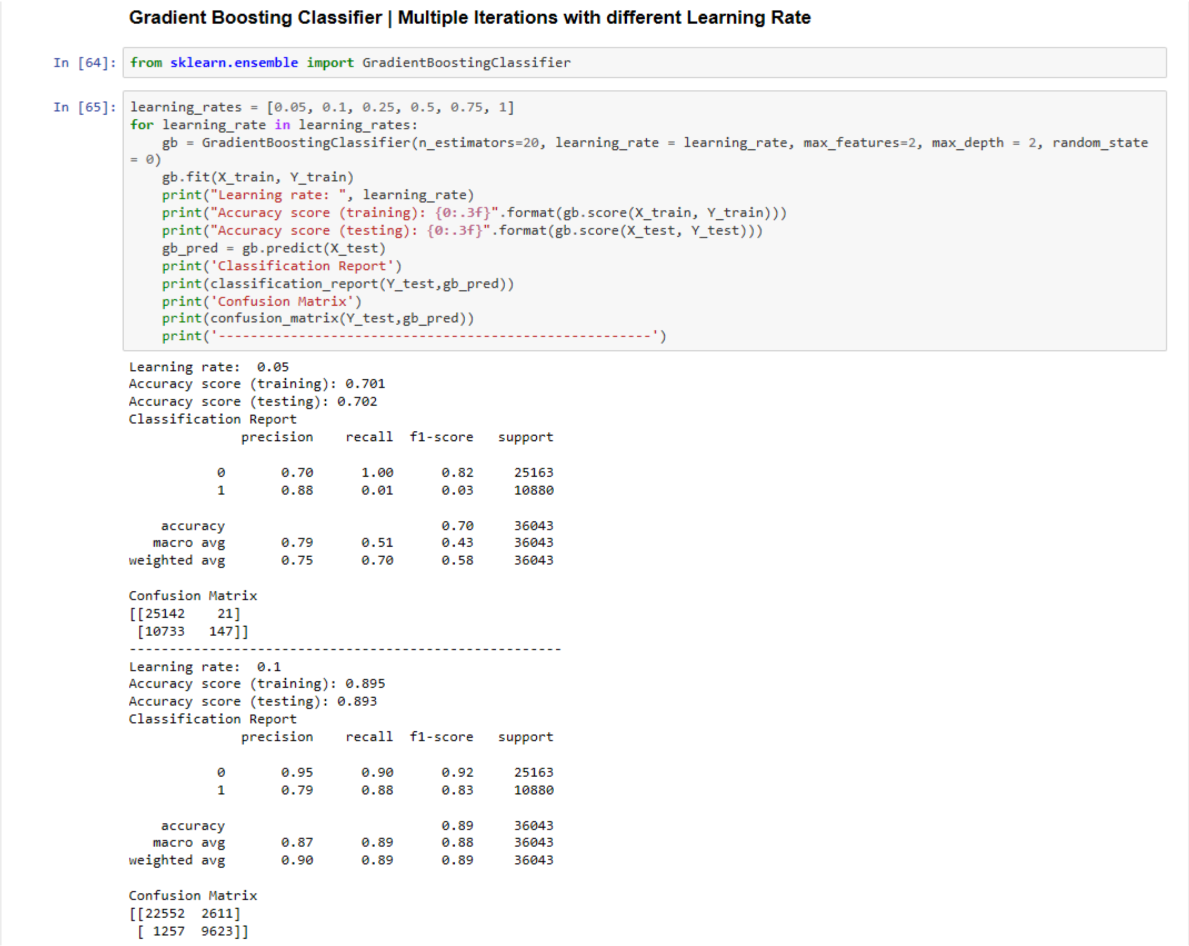
```
In [61]: classifier.fit(X_train, Y_train)
```

```
Out[61]: AdaBoostClassifier(algorithm='SAMME.R',
                            base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                                  class_weight=None,
                                                                  criterion='gini',
                                                                  max_depth=1,
                                                                  max_features=None,
                                                                  max_leaf_nodes=None,
                                                                  min_impurity_decrease=0.0,
                                                                  min_impurity_split=None,
                                                                  min_samples_leaf=1,
                                                                  min_samples_split=2,
                                                                  min_weight_fraction_leaf=0.0,
                                                                  presort='deprecated',
                                                                  random_state=None,
                                                                  splitter='best'),
                            learning_rate=1.0, n_estimators=200, random_state=None)
```

```
In [62]: adaboost_pred = classifier.predict(X_test)
```

```
In [63]: print('##### Classification Report of Adaboost Classfier #####')
         print()
         print(classification_report(Y_test,adaboost_pred))
         print('##### Confusion Matrix of Adaboost Classfier #####')
         print()
         print(confusion_matrix(Y_test,adaboost_pred))
```

```
##### Classification Report of Adaboost Classfier #####

              precision    recall  f1-score   support

           0       0.95      0.89      0.92     25163
           1       0.79      0.89      0.83     10880

    accuracy                           0.89     36043
   macro avg       0.87      0.89      0.88     36043
weighted avg       0.90      0.89      0.89     36043

##### Confusion Matrix of Adaboost Classfier #####

[[22517  2646]
 [ 1217  9663]]
```

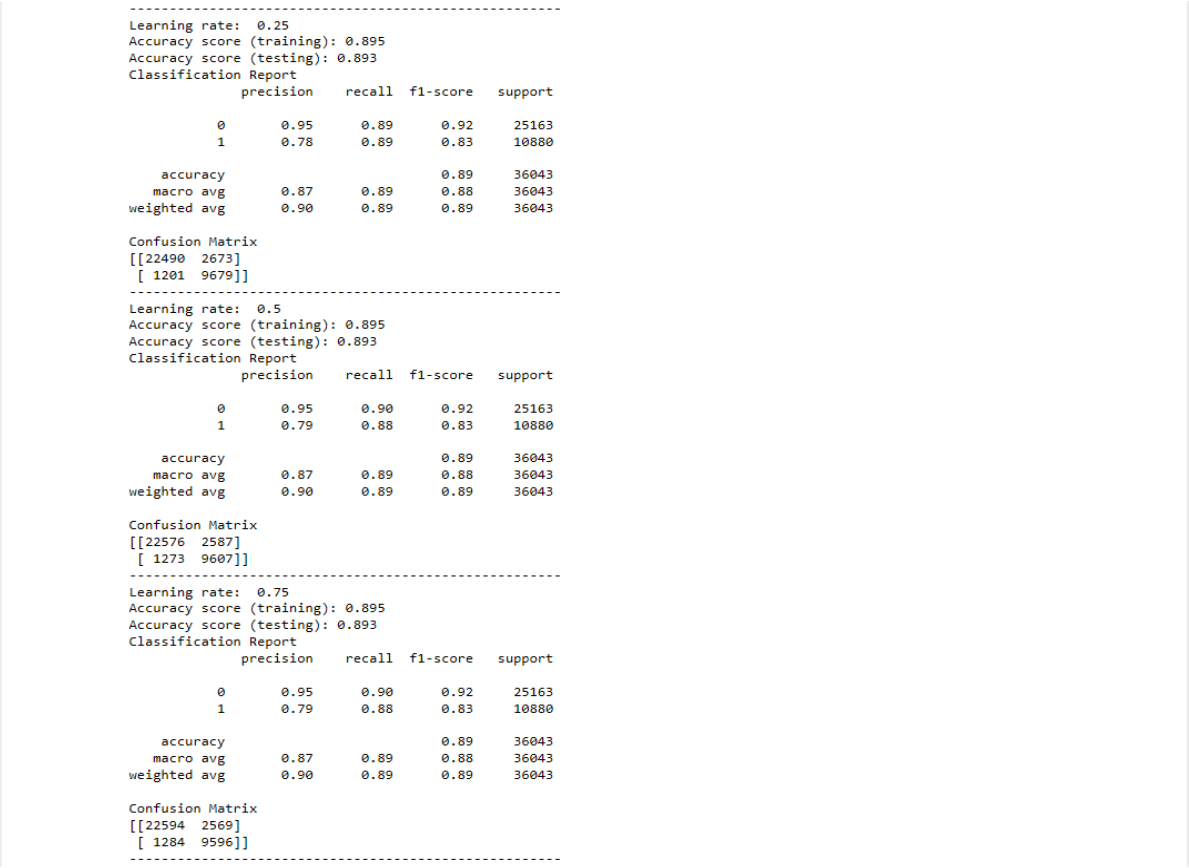


Gradient Boosting Classifier | Multiple Iterations with different Learning Rate

In [64]:
```python
from sklearn.ensemble import GradientBoostingClassifier
```

In [65]:
```python
learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in learning_rates:
    gb = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2, max_depth = 2, random_state
= 0)
    gb.fit(X_train, Y_train)
    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb.score(X_train, Y_train)))
    print("Accuracy score (testing): {0:.3f}".format(gb.score(X_test, Y_test)))
    gb_pred = gb.predict(X_test)
    print('Classification Report')
    print(classification_report(Y_test,gb_pred))
    print('Confusion Matrix')
    print(confusion_matrix(Y_test,gb_pred))
    print('-----------------------------------------------------')
```

```
Learning rate:  0.05
Accuracy score (training): 0.701
Accuracy score (testing): 0.702
Classification Report
              precision    recall  f1-score   support

           0       0.70      1.00      0.82     25163
           1       0.88      0.01      0.03     10880

    accuracy                           0.70     36043
   macro avg       0.79      0.51      0.43     36043
weighted avg       0.75      0.70      0.58     36043

Confusion Matrix
[[25142    21]
 [10733   147]]
-----------------------------------------------------
Learning rate:  0.1
Accuracy score (training): 0.895
Accuracy score (testing): 0.893
Classification Report
              precision    recall  f1-score   support

           0       0.95      0.90      0.92     25163
           1       0.79      0.88      0.83     10880

    accuracy                           0.89     36043
   macro avg       0.87      0.89      0.88     36043
weighted avg       0.90      0.89      0.89     36043

Confusion Matrix
[[22552  2611]
 [ 1257  9623]]
```



```
-----------------------------------------------------
Learning rate:  0.25
Accuracy score (training): 0.895
Accuracy score (testing): 0.893
Classification Report
              precision    recall  f1-score   support

           0       0.95      0.89      0.92     25163
           1       0.78      0.89      0.83     10880

    accuracy                           0.89     36043
   macro avg       0.87      0.89      0.88     36043
weighted avg       0.90      0.89      0.89     36043

Confusion Matrix
[[22490  2673]
 [ 1201  9679]]
-----------------------------------------------------
Learning rate:  0.5
Accuracy score (training): 0.895
Accuracy score (testing): 0.893
Classification Report
              precision    recall  f1-score   support

           0       0.95      0.90      0.92     25163
           1       0.79      0.88      0.83     10880

    accuracy                           0.89     36043
   macro avg       0.87      0.89      0.88     36043
weighted avg       0.90      0.89      0.89     36043

Confusion Matrix
[[22576  2587]
 [ 1273  9607]]
-----------------------------------------------------
Learning rate:  0.75
Accuracy score (training): 0.895
Accuracy score (testing): 0.893
Classification Report
              precision    recall  f1-score   support

           0       0.95      0.90      0.92     25163
           1       0.79      0.88      0.83     10880

    accuracy                           0.89     36043
   macro avg       0.87      0.89      0.88     36043
weighted avg       0.90      0.89      0.89     36043

Confusion Matrix
[[22594  2569]
 [ 1284  9596]]
-----------------------------------------------------
```

```
Learning rate:  1
Accuracy score (training): 0.895
Accuracy score (testing): 0.893
Classification Report
              precision    recall  f1-score   support

           0       0.95      0.89      0.92     25163
           1       0.78      0.89      0.83     10880

    accuracy                           0.89     36043
   macro avg       0.87      0.89      0.88     36043
weighted avg       0.90      0.89      0.89     36043

Confusion Matrix
[[22507  2656]
 [ 1206  9674]]
------------------------------------------------------
```

Final Model with Gradient Boosting Classifier with Hyperparameter tuned with Learning Rate = 0.75

In [66]:

```
learning_rate = 0.75
gb = GradientBoostingClassifier(n_estimators=20, learning_rate = learning_rate, max_features=2, max_depth = 2, random_state = 0)
gb.fit(X_train, Y_train)
print("Learning rate: ", learning_rate)
print("Accuracy score (training): {0:.3f}".format(gb.score(X_train, Y_train)))
print("Accuracy score (testing): {0:.3f}".format(gb.score(X_test, Y_test)))
gb_pred = gb.predict(X_test)
print('##### Classification Report #####')
print(classification_report(Y_test,gb_pred))
print('##### Confusion Matrix #####')
print(confusion_matrix(Y_test,gb_pred))
```

```
Learning rate:  0.75
Accuracy score (training): 0.895
Accuracy score (testing): 0.893
##### Classification Report #####
              precision    recall  f1-score   support

           0       0.95      0.90      0.92     25163
           1       0.79      0.88      0.83     10880

    accuracy                           0.89     36043
   macro avg       0.87      0.89      0.88     36043
weighted avg       0.90      0.89      0.89     36043

##### Confusion Matrix #####
[[22594  2569]
 [ 1284  9596]]
```

In [67]:

```
# ROC curve and Area-Under-Curve (AUC)
y_scores_gb = gb.decision_function(X_test)
fpr_gb, tpr_gb, _ = roc_curve(Y_test, y_scores_gb)
roc_auc_gb = auc(fpr_gb, tpr_gb)

print("Area under ROC curve = {:0.2f}".format(roc_auc_gb))
```

```
Area under ROC curve = 0.91
```

Prediction on the Test Data Set Provided | Generating final Result.csv in given format

In [68]:

```
test_data = pd.read_csv('Test Data.csv')
```

In [69]:

```
test_data['Gender'] = test_data['Gender'].apply(lambda x: 1 if x == 'M' else 0)
```

In [70]:

```
test_data = test_data.set_index('patient_id')
```

In [71]:

```
test_data.head()
```

Out[71]:

	Age	Gender	Prescription_period	Diabetes	Alcoholism	HyperTension	Smokes	Tuberculosis	Sms_Reminder
patient_id									
1	5	1	28	0	0	0	0	0	1
2	62	0	9	1	0	1	0	0	0
3	4	0	73	0	0	0	0	0	1
4	33	1	117	0	0	0	0	0	0
5	38	1	8	0	0	0	0	0	1

In [72]:

```
predictions_test = gb.predict(test_data)
```

In [73]:

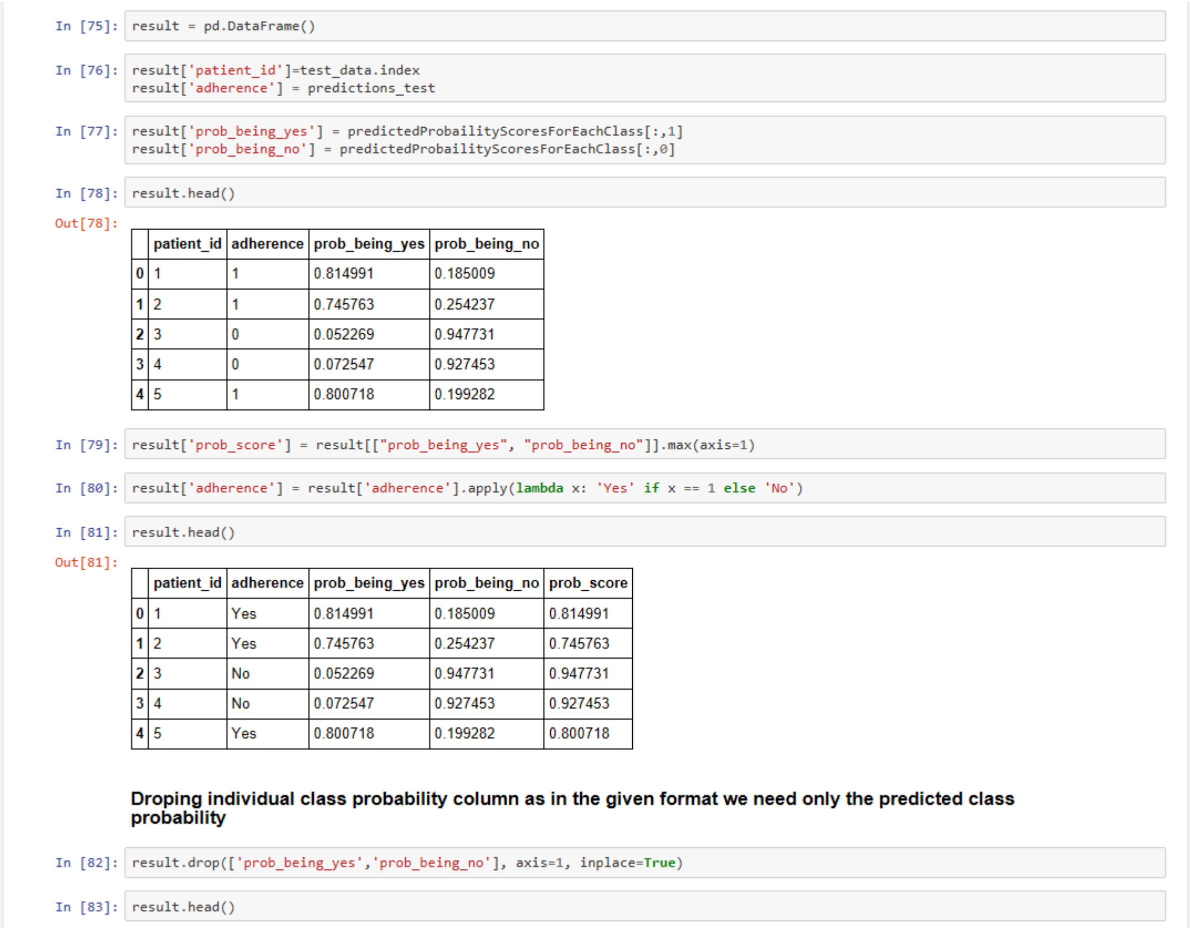
```
predictedProbailityScoresForEachClass = gb.predict_proba(test_data)
```

In [74]:

```
prob = pd.DataFrame(predictedProbailityScoresForEachClass)
prob.head()
```

Out[74]:

	0	1
0	0.185009	0.814991
1	0.254237	0.745763
2	0.947731	0.052269
3	0.927453	0.072547
4	0.199282	0.800718



```
In [75]: result = pd.DataFrame()
```

```
In [76]: result['patient_id']=test_data.index
         result['adherence'] = predictions_test
```

```
In [77]: result['prob_being_yes'] = predictedProbailityScoresForEachClass[:,1]
         result['prob_being_no'] = predictedProbailityScoresForEachClass[:,0]
```

```
In [78]: result.head()
```

Out[78]:

	patient_id	adherence	prob_being_yes	prob_being_no
0	1	1	0.814991	0.185009
1	2	1	0.745763	0.254237
2	3	0	0.052269	0.947731
3	4	0	0.072547	0.927453
4	5	1	0.800718	0.199282

```
In [79]: result['prob_score'] = result[["prob_being_yes", "prob_being_no"]].max(axis=1)
```

```
In [80]: result['adherence'] = result['adherence'].apply(lambda x: 'Yes' if x == 1 else 'No')
```

```
In [81]: result.head()
```

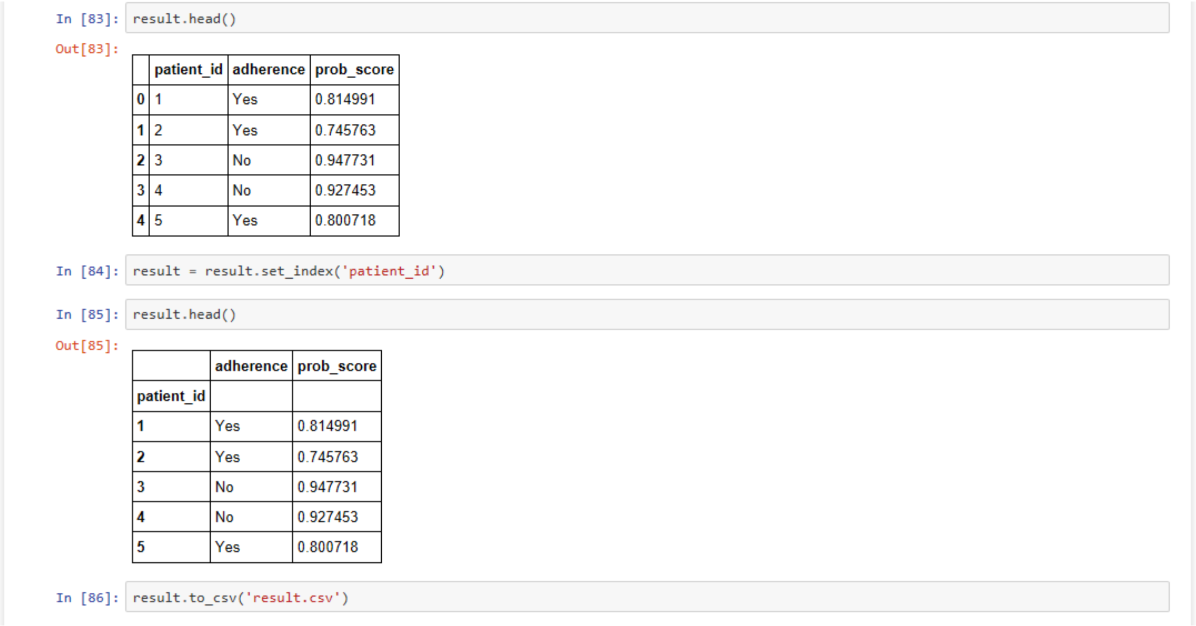
Out[81]:

	patient_id	adherence	prob_being_yes	prob_being_no	prob_score
0	1	Yes	0.814991	0.185009	0.814991
1	2	Yes	0.745763	0.254237	0.745763
2	3	No	0.052269	0.947731	0.947731
3	4	No	0.072547	0.927453	0.927453
4	5	Yes	0.800718	0.199282	0.800718

Droping individual class probability column as in the given format we need only the predicted class probability

```
In [82]: result.drop(['prob_being_yes','prob_being_no'], axis=1, inplace=True)
```

```
In [83]: result.head()
```



```
In [83]: result.head()
```

Out[83]:

	patient_id	adherence	prob_score
0	1	Yes	0.814991
1	2	Yes	0.745763
2	3	No	0.947731
3	4	No	0.927453
4	5	Yes	0.800718

```
In [84]: result = result.set_index('patient_id')
```

```
In [85]: result.head()
```

Out[85]:

	adherence	prob_score
patient_id		
1	Yes	0.814991
2	Yes	0.745763
3	No	0.947731
4	No	0.927453
5	Yes	0.800718

```
In [86]: result.to_csv('result.csv')
```

Thank You

For any help and support in Data Science, Machine Learning and Artificial Intelligence,

Please connect with me on

Website: https://enetwork.ai/ or https://enetwork.ai/contact-us

LinkedIn: https://www.linkedin.com/in/ashutoshtripathi1/

Instagram: https://www.instagram.com/ashutosh_ai/

Medium Articles: https://medium.com/@ashutosh.optimistic