SCHOOL OF APPLIED TECHNOLOGY

Illinois Institute of Technology

CODE AND OUTPUT SNAPSHOTS

Group Number-297: Diabetic Medication & Patient Re-admission Prediction using different Classification Algorithms

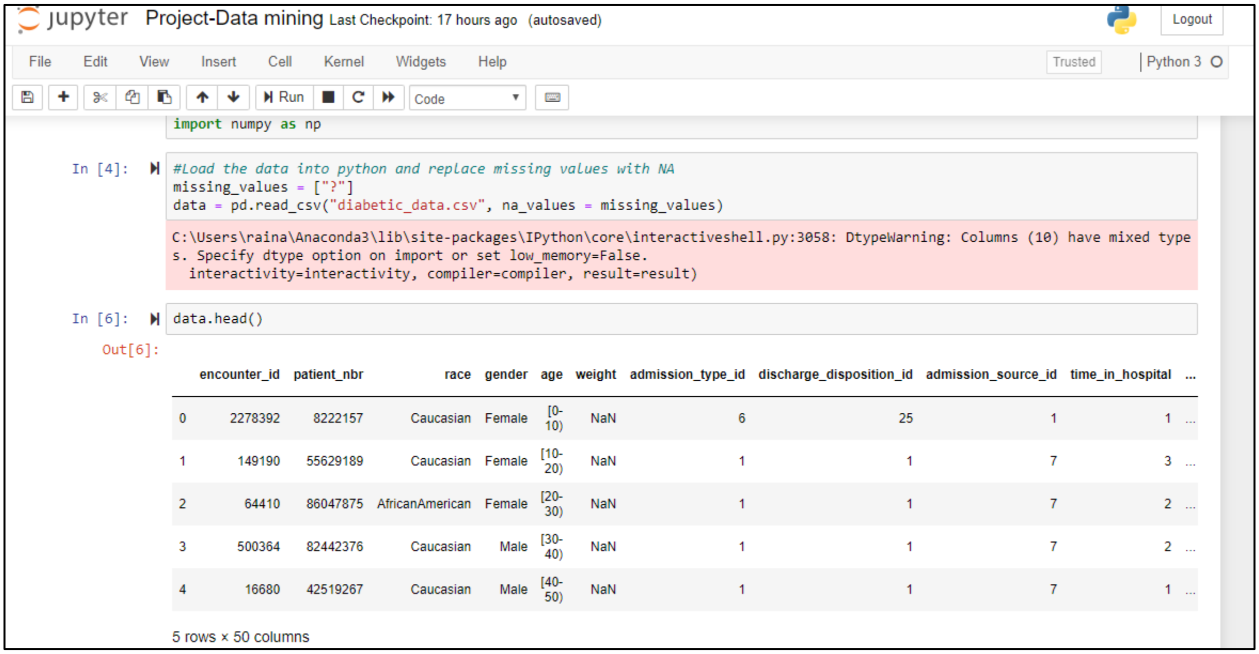
First Name	Last Name	Email address
Prashant	Raina	praina@hawk.iit.edu
Vaidehi	Rathkanthiwar	vrathkanthiwar@hawk.iit.edu
Utkarsha	Vidhale	uvidhale@hawk.iit.edu

ITMD 525

Topics in Data Management: Data Mining

Python Code and output snapshots of the Project : -

Snapshot while loading the dataset :



```
import numpy as np
```

In [4]:

```
#Load the data into python and replace missing values with NA
missing_values = ["?"]
data = pd.read_csv("diabetic_data.csv", na_values = missing_values)
```

```
C:\Users\raina\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3058: DtypeWarning: Columns (10) have mixed type
s. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [6]:

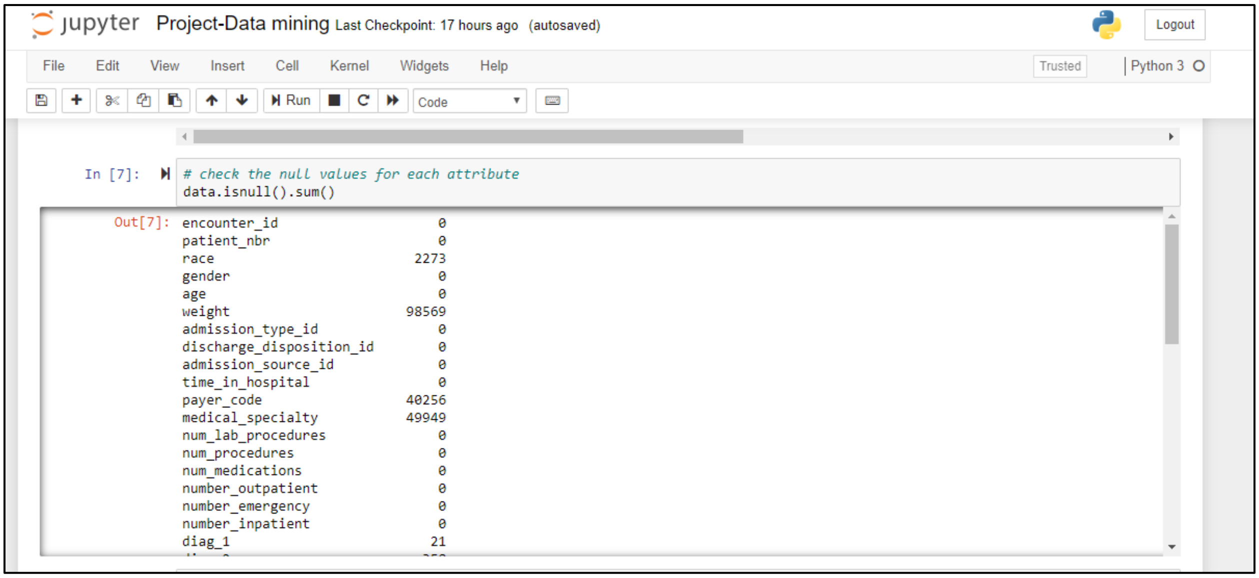
```
data.head()
```

Out[6]:

	encounter_id	patient_nbr	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time_in_hospital	...
0	2278392	8222157	Caucasian	Female	[0-10)	NaN	6	25	1	1	...
1	149190	55629189	Caucasian	Female	[10-20)	NaN	1	1	7	3	...
2	64410	86047875	AfricanAmerican	Female	[20-30)	NaN	1	1	7	2	...
3	500364	82442376	Caucasian	Male	[30-40)	NaN	1	1	7	2	...
4	16680	42519267	Caucasian	Male	[40-50)	NaN	1	1	7	1	...

5 rows × 50 columns

Check for missing values in the dataset :



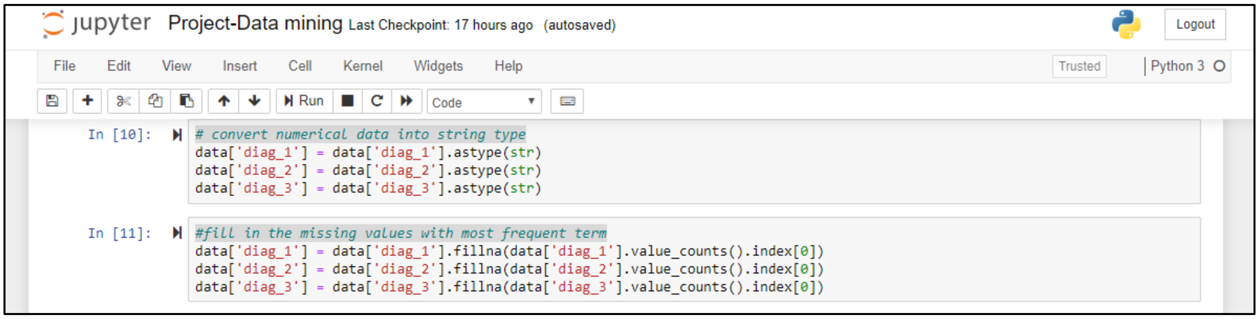
In [7]:

```
# check the null values for each attribute
data.isnull().sum()
```

Out[7]:

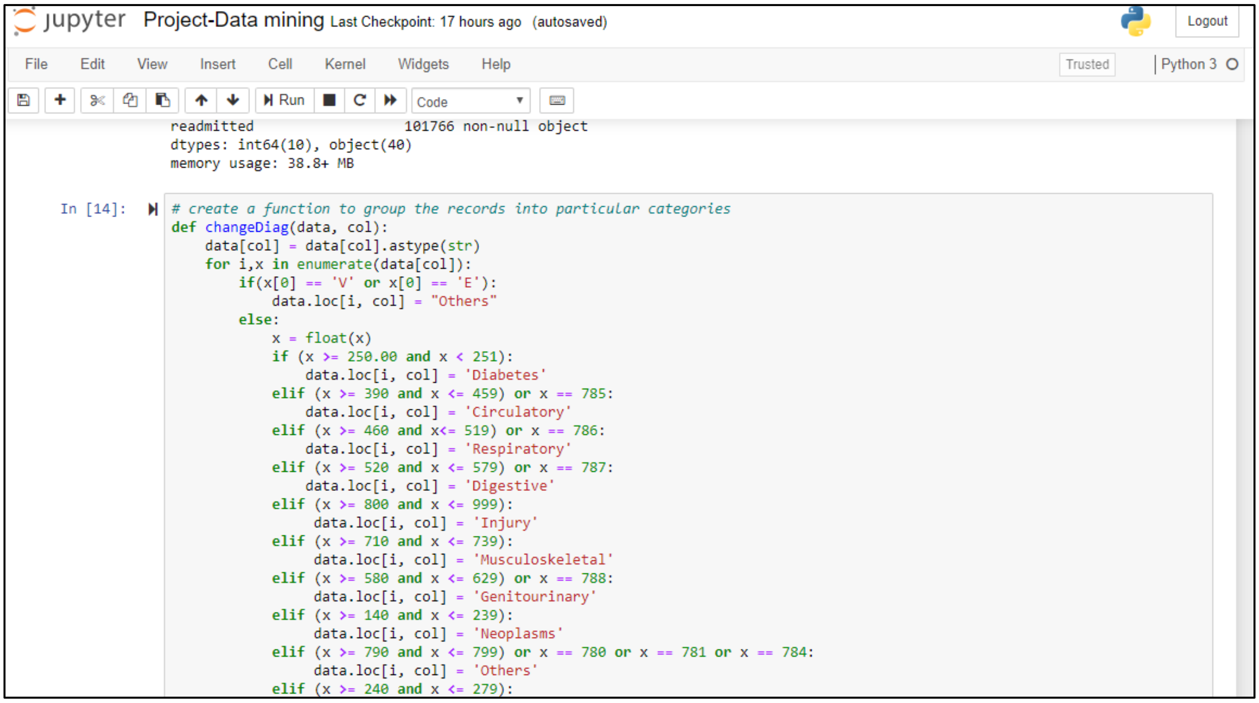
```
encounter_id                    0
patient_nbr                     0
race                         2273
gender                          0
age                             0
weight                      98569
admission_type_id               0
discharge_disposition_id        0
admission_source_id             0
time_in_hospital                0
payer_code                  40256
medical_specialty           49949
num_lab_procedures              0
num_procedures                  0
num_medications                 0
number_outpatient               0
number_emergency                0
number_inpatient                0
diag_1                         21
```

Fill in missing values in the data set with the most frequent value :



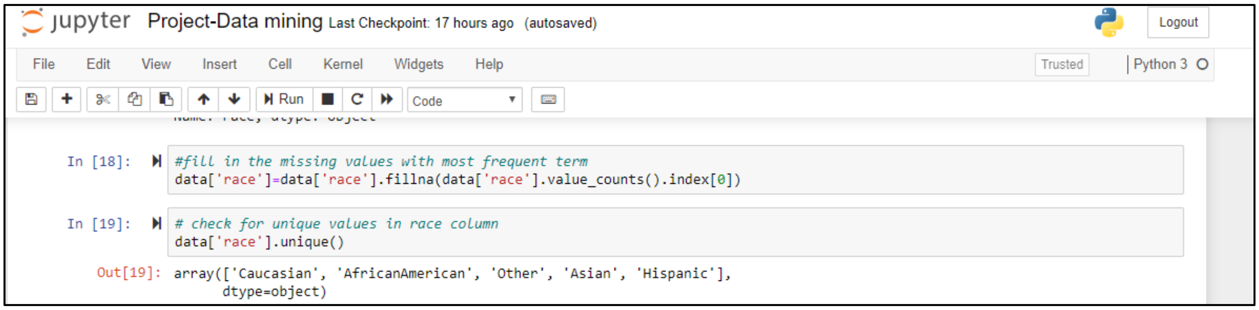
```
In [10]:  # convert numerical data into string type
          data['diag_1'] = data['diag_1'].astype(str)
          data['diag_2'] = data['diag_2'].astype(str)
          data['diag_3'] = data['diag_3'].astype(str)

In [11]:  #fill in the missing values with most frequent term
          data['diag_1'] = data['diag_1'].fillna(data['diag_1'].value_counts().index[0])
          data['diag_2'] = data['diag_2'].fillna(data['diag_2'].value_counts().index[0])
          data['diag_3'] = data['diag_3'].fillna(data['diag_3'].value_counts().index[0])
```

Grouping the records of columns diag_1, diag_2, diag_3 into particular categories:



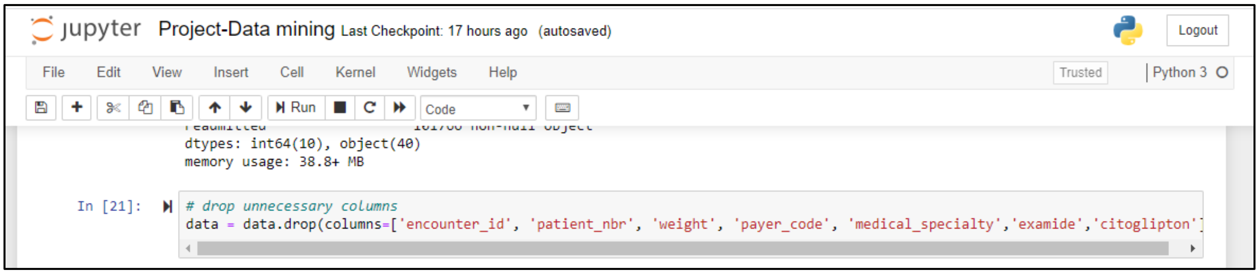
```
readmitted                  101766 non-null object
dtypes: int64(10), object(40)
memory usage: 38.8+ MB

In [14]:  # create a function to group the records into particular categories
          def changeDiag(data, col):
              data[col] = data[col].astype(str)
              for i,x in enumerate(data[col]):
                  if(x[0] == 'V' or x[0] == 'E'):
                      data.loc[i, col] = "Others"
                  else:
                      x = float(x)
                      if (x >= 250.00 and x < 251):
                          data.loc[i, col] = 'Diabetes'
                      elif (x >= 390 and x <= 459) or x == 785:
                          data.loc[i, col] = 'Circulatory'
                      elif (x >= 460 and x<= 519) or x == 786:
                          data.loc[i, col] = 'Respiratory'
                      elif (x >= 520 and x <= 579) or x == 787:
                          data.loc[i, col] = 'Digestive'
                      elif (x >= 800 and x <= 999):
                           data.loc[i, col] = 'Injury'
                      elif (x >= 710 and x <= 739):
                           data.loc[i, col] = 'Musculoskeletal'
                      elif (x >= 580 and x <= 629) or x == 788:
                           data.loc[i, col] = 'Genitourinary'
                      elif (x >= 140 and x <= 239):
                           data.loc[i, col] = 'Neoplasms'
                      elif (x >= 790 and x <= 799) or x == 780 or x == 781 or x == 784:
                           data.loc[i, col] = 'Others'
                      elif (x >= 240 and x <= 279):
```

Fill in missing values for Race with most frequent value :


jupyter Project-Data mining Last Checkpoint: 17 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 ○
Run
Code

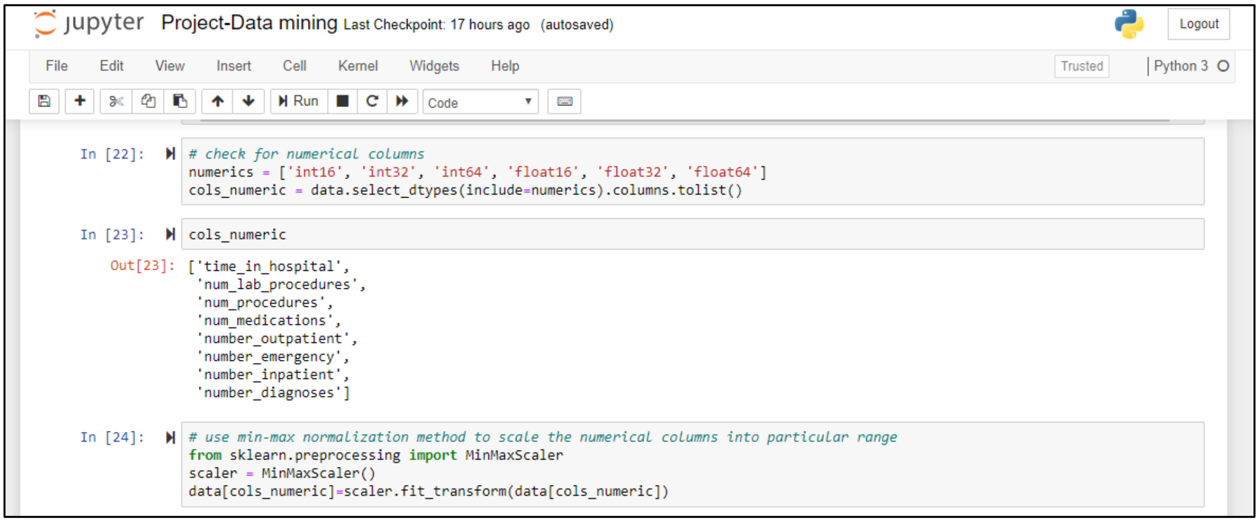
```
Name: race, dtype: object

In [18]:  #fill in the missing values with most frequent term
          data['race']=data['race'].fillna(data['race'].value_counts().index[0])

In [19]:  # check for unique values in race column
          data['race'].unique()

Out[19]:  array(['Caucasian', 'AfricanAmerican', 'Other', 'Asian', 'Hispanic'],
                dtype=object)
```

Drop unnecessary columns from the dataset :


jupyter Project-Data mining Last Checkpoint: 17 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 ○
Run
Code

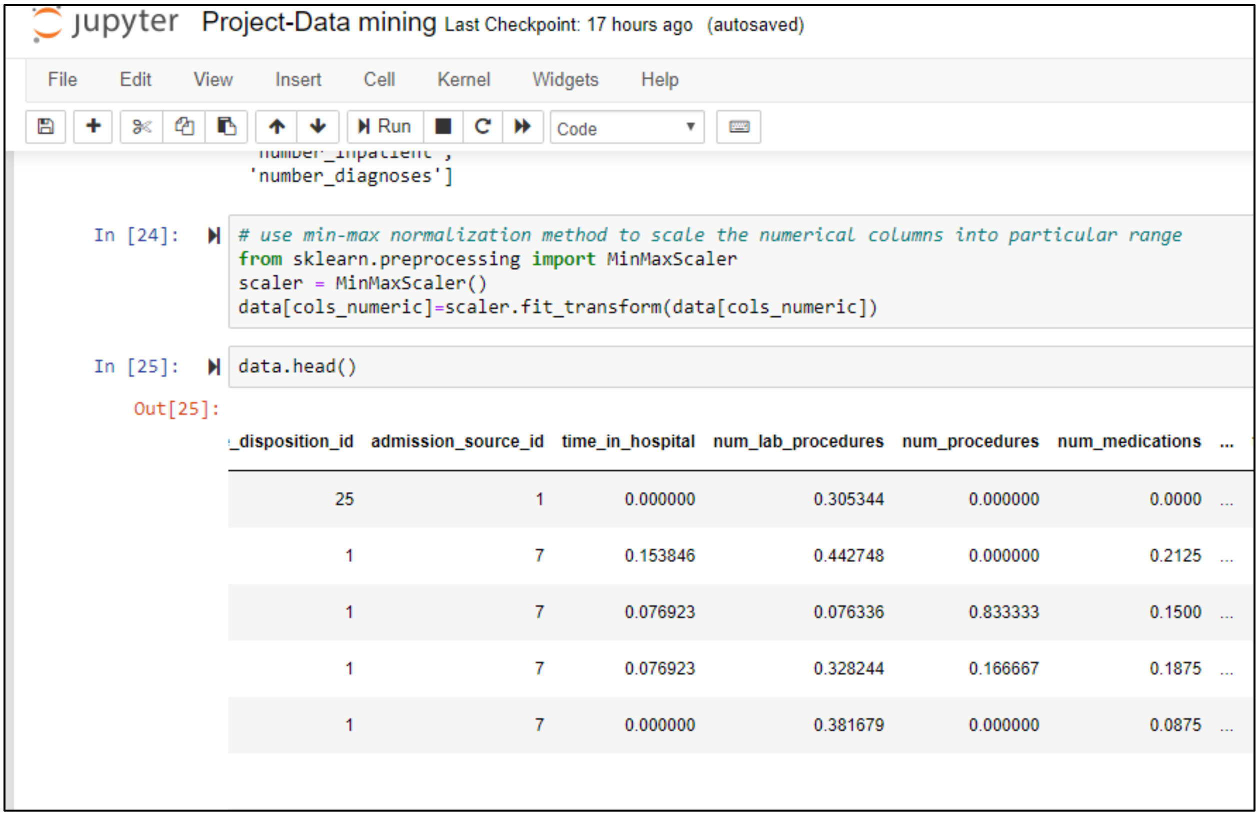
```
readmitted                101766 non-null object
dtypes: int64(10), object(40)
memory usage: 38.8+ MB

In [21]:  # drop unnecessary columns
          data = data.drop(columns=['encounter_id', 'patient_nbr', 'weight', 'payer_code', 'medical_specialty','examide','citoglipton']
```

Check for numerical data and perform Min Max normalization :


jupyter Project-Data mining Last Checkpoint: 17 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 ○
Run
Code

```
In [22]:  # check for numerical columns
          numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
          cols_numeric = data.select_dtypes(include=numerics).columns.tolist()

In [23]:  cols_numeric

Out[23]:  ['time_in_hospital',
           'num_lab_procedures',
           'num_procedures',
           'num_medications',
           'number_outpatient',
           'number_emergency',
           'number_inpatient',
           'number_diagnoses']

In [24]:  # use min-max normalization method to scale the numerical columns into particular range
          from sklearn.preprocessing import MinMaxScaler
          scaler = MinMaxScaler()
          data[cols_numeric]=scaler.fit_transform(data[cols_numeric])
```

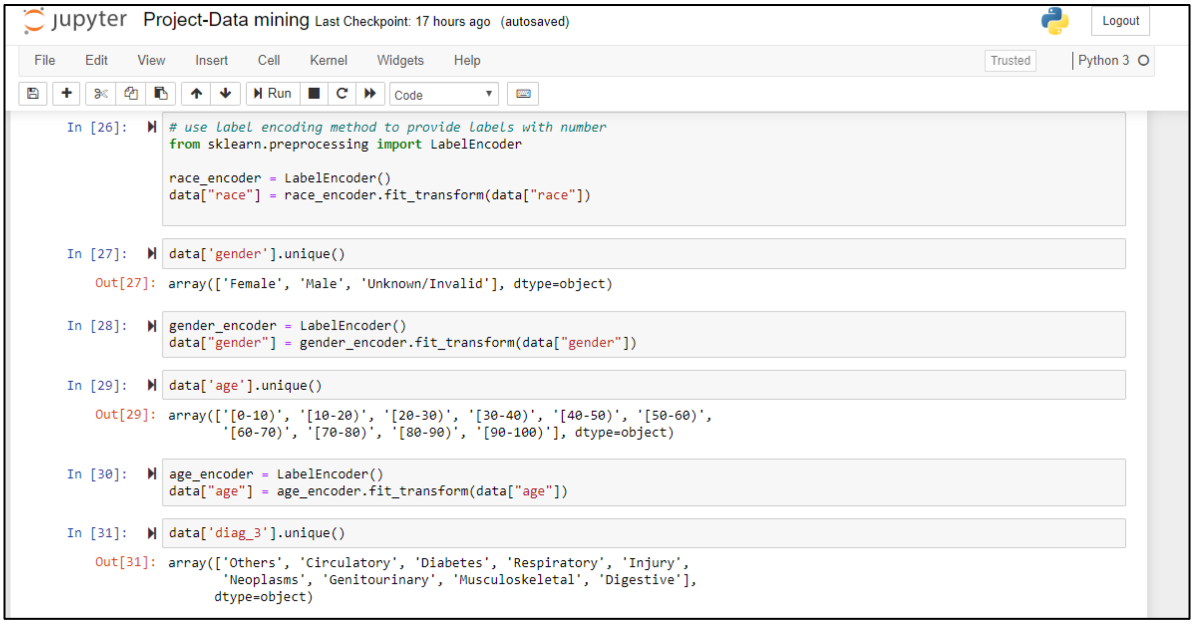
Data after performing Min Max normalization :


jupyter
Project-Data mining Last Checkpoint: 17 hours ago (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code

```
           'number_inpatient',
           'number_diagnoses']

In [24]:   # use min-max normalization method to scale the numerical columns into particular range
           from sklearn.preprocessing import MinMaxScaler
           scaler = MinMaxScaler()
           data[cols_numeric]=scaler.fit_transform(data[cols_numeric])

In [25]:   data.head()

Out[25]:
```

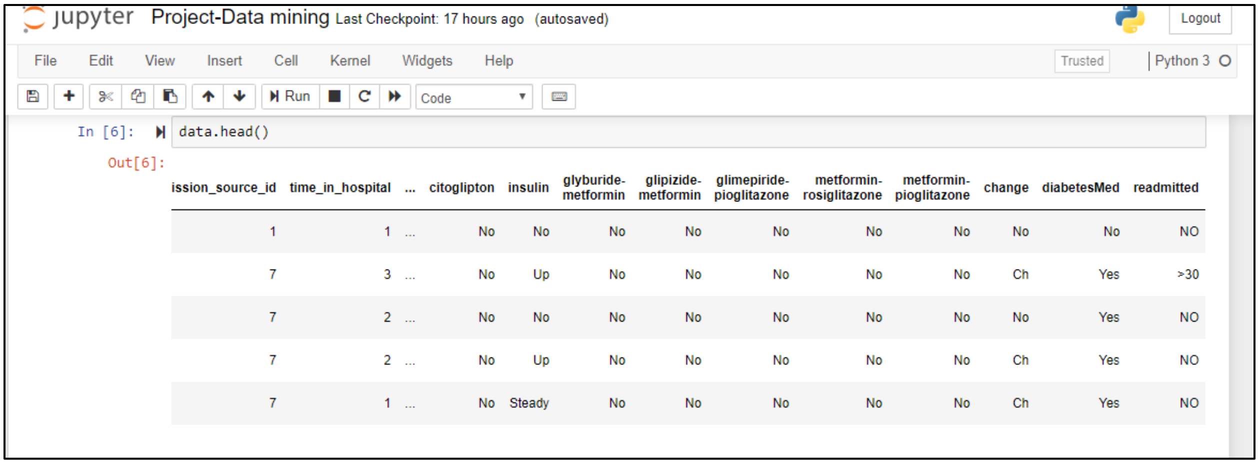
_disposition_id	admission_source_id	time_in_hospital	num_lab_procedures	num_procedures	num_medications	...
25	1	0.000000	0.305344	0.000000	0.0000	...
1	7	0.153846	0.442748	0.000000	0.2125	...
1	7	0.076923	0.076336	0.833333	0.1500	...
1	7	0.076923	0.328244	0.166667	0.1875	...
1	7	0.000000	0.381679	0.000000	0.0875	...

Label encoding for giving labels with number :



```
In [26]:   # use label encoding method to provide labels with number
           from sklearn.preprocessing import LabelEncoder

           race_encoder = LabelEncoder()
           data["race"] = race_encoder.fit_transform(data["race"])

In [27]:   data['gender'].unique()

Out[27]:   array(['Female', 'Male', 'Unknown/Invalid'], dtype=object)

In [28]:   gender_encoder = LabelEncoder()
           data["gender"] = gender_encoder.fit_transform(data["gender"])

In [29]:   data['age'].unique()

Out[29]:   array(['[0-10)', '[10-20)', '[20-30)', '[30-40)', '[40-50)', '[50-60)',
                  '[60-70)', '[70-80)', '[80-90)', '[90-100)'], dtype=object)

In [30]:   age_encoder = LabelEncoder()
           data["age"] = age_encoder.fit_transform(data["age"])

In [31]:   data['diag_3'].unique()

Out[31]:   array(['Others', 'Circulatory', 'Diabetes', 'Respiratory', 'Injury',
                  'Neoplasms', 'Genitourinary', 'Musculoskeletal', 'Digestive'],
                 dtype=object)
```

Data before and after label encoding :

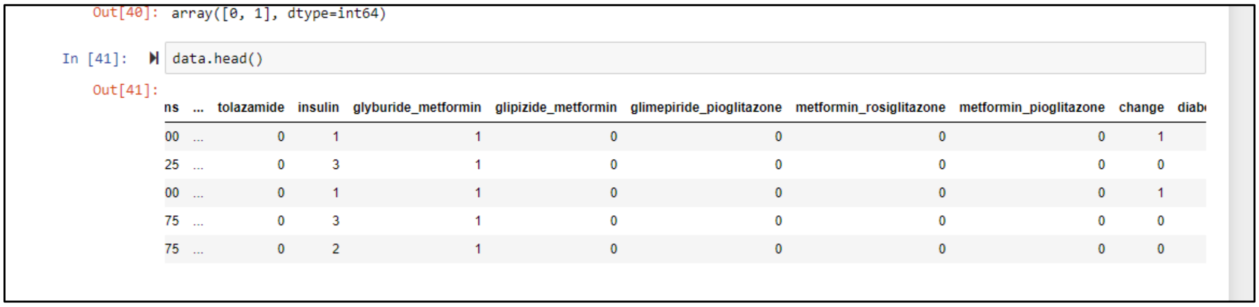

jupyter Project-Data mining Last Checkpoint: 17 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

In [6]:

```
data.head()
```

Out[6]:

ission_source_id	time_in_hospital	...	citoglipton	insulin	glyburide-metformin	glipizide-metformin	glimepiride-pioglitazone	metformin-rosiglitazone	metformin-pioglitazone	change	diabetesMed	readmitted
1	1	...	No	No	No	No	No	No	No	No	No	NO
7	3	...	No	Up	No	No	No	No	No	Ch	Yes	>30
7	2	...	No	No	No	No	No	No	No	No	Yes	NO
7	2	...	No	Up	No	No	No	No	No	Ch	Yes	NO
7	1	...	No	Steady	No	No	No	No	No	Ch	Yes	NO



```
Out[40]: array([0, 1], dtype=int64)
```

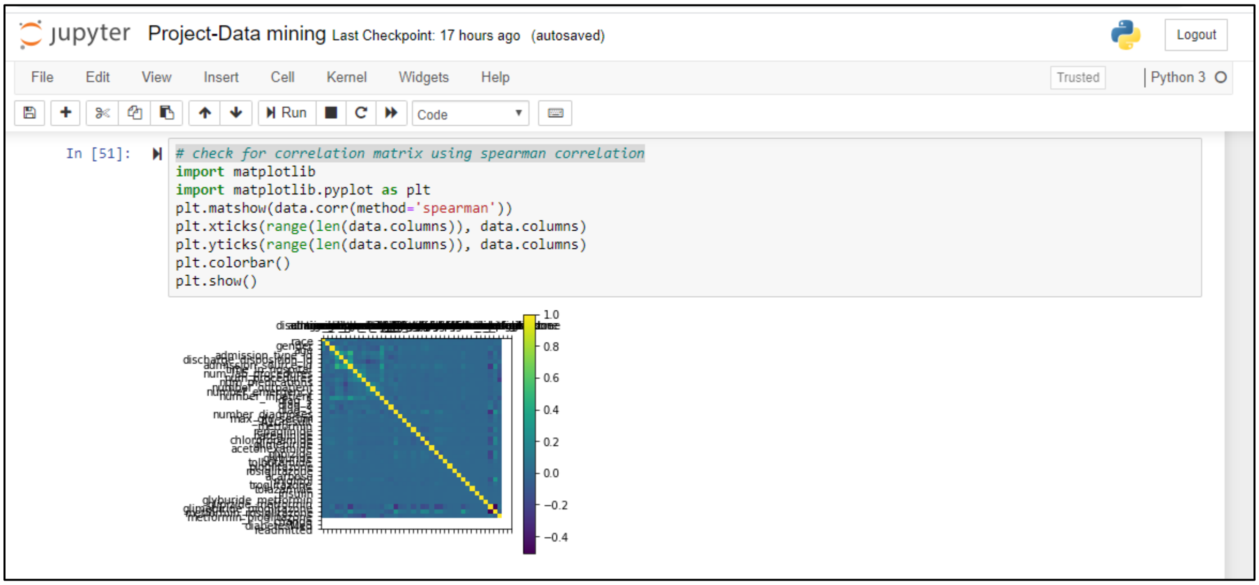
In [41]:

```
data.head()
```

Out[41]:

ns	...	tolazamide	insulin	glyburide_metformin	glipizide_metformin	glimepiride_pioglitazone	metformin_rosiglitazone	metformin_pioglitazone	change	diab
00	...	0	1	1	0	0	0	0	1	
25	...	0	3	1	0	0	0	0	0	
00	...	0	1	1	0	0	0	0	1	
75	...	0	3	1	0	0	0	0	0	
75	...	0	2	1	0	0	0	0	0	

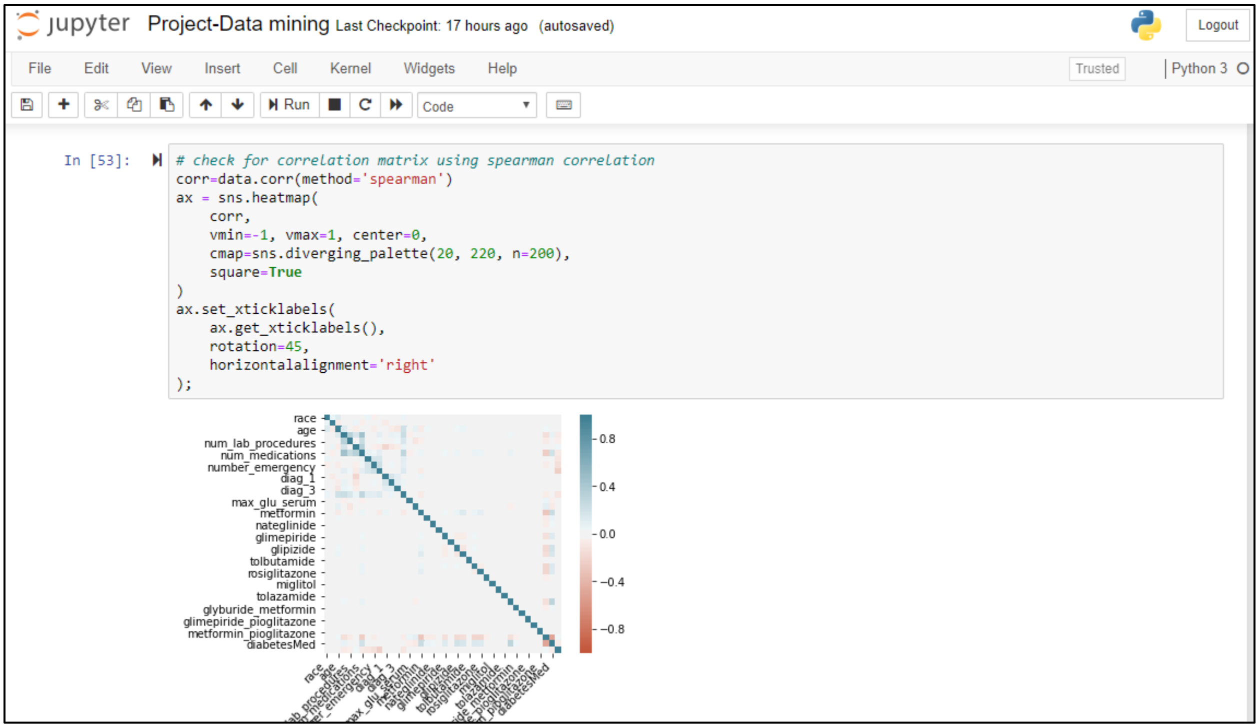
Correlation matrix – spearman method



In [51]:

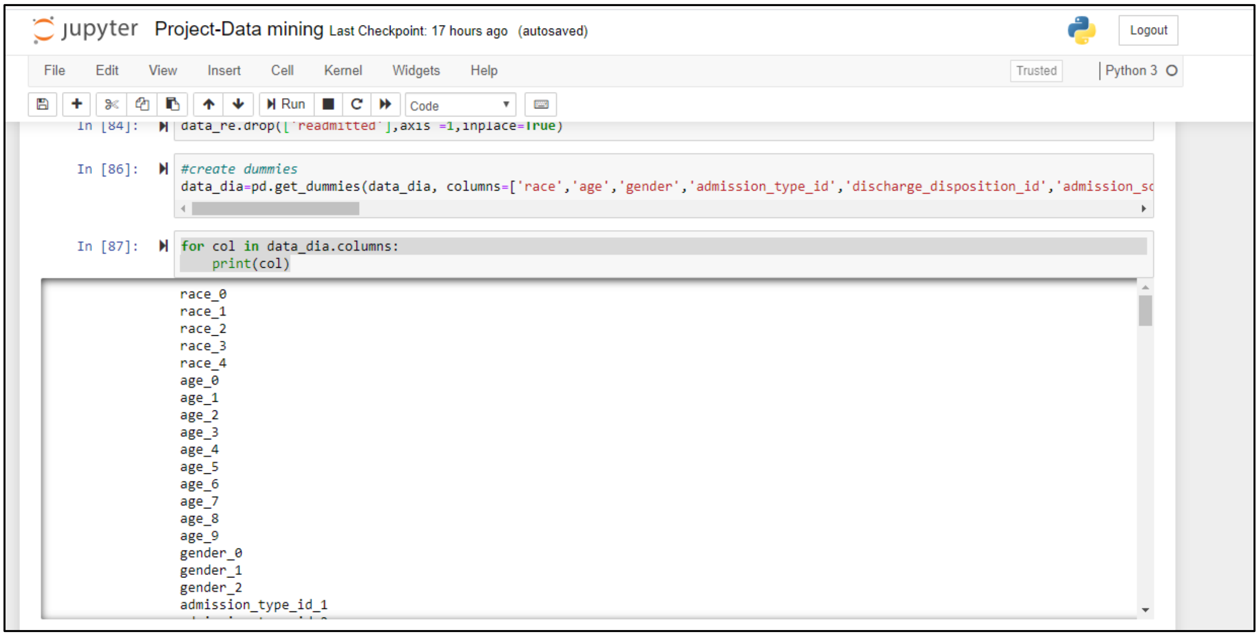
```
# check for correlation matrix using spearman correlation
import matplotlib
import matplotlib.pyplot as plt
plt.matshow(data.corr(method='spearman'))
plt.xticks(range(len(data.columns)), data.columns)
plt.yticks(range(len(data.columns)), data.columns)
plt.colorbar()
plt.show()
```

Heatmap correlation



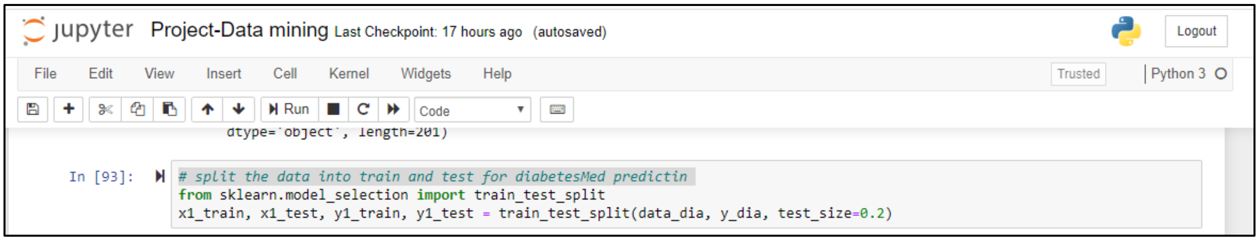
```
In [53]: # check for correlation matrix using spearman correlation
         corr=data.corr(method='spearman')
         ax = sns.heatmap(
             corr,
             vmin=-1, vmax=1, center=0,
             cmap=sns.diverging_palette(20, 220, n=200),
             square=True
         )
         ax.set_xticklabels(
             ax.get_xticklabels(),
             rotation=45,
             horizontalalignment='right'
         );
```

Creating dummies for the columns :



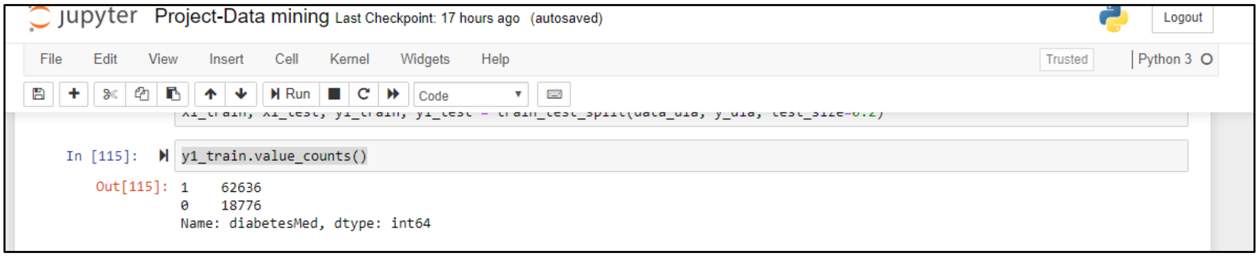
```
In [84]: data_re.drop(['readmitted'],axis =1,inplace=True)

In [86]: #create dummies
         data_dia=pd.get_dummies(data_dia, columns=['race','age','gender','admission_type_id','discharge_disposition_id','admission_sc

In [87]: for col in data_dia.columns:
             print(col)

         race_0
         race_1
         race_2
         race_3
         race_4
         age_0
         age_1
         age_2
         age_3
         age_4
         age_5
         age_6
         age_7
         age_8
         age_9
         gender_0
         gender_1
         gender_2
         admission_type_id_1
```

Split data before SMOTE process :


jupyter Project-Data mining Last Checkpoint: 17 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

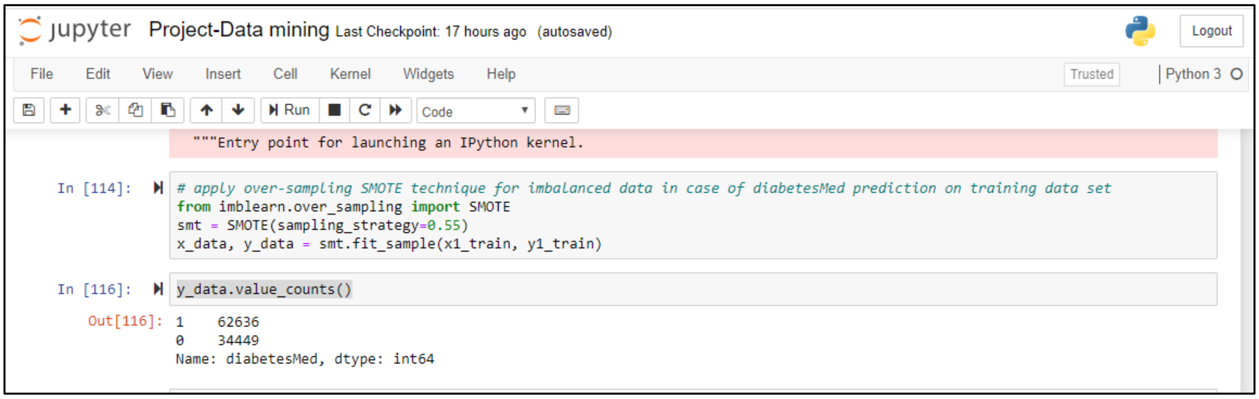
```
        dtype='object', length=201)

In [93]:  # split the data into train and test for diabetesMed predictin
          from sklearn.model_selection import train_test_split
          x1_train, x1_test, y1_train, y1_test = train_test_split(data_dia, y_dia, test_size=0.2)
```

Imbalance data for diabetesMed variable :


jupyter Project-Data mining Last Checkpoint: 17 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

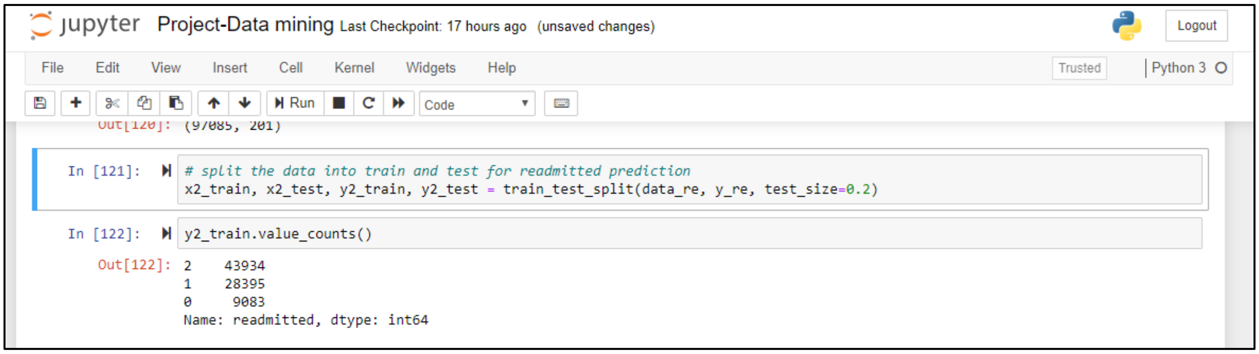
```
           x1_train, x1_test, y1_train, y1_test = train_test_split(data_dia, y_dia, test_size=0.2)

In [115]:  y1_train.value_counts()

Out[115]:  1    62636
           0    18776
           Name: diabetesMed, dtype: int64
```

While applying SMOTE technique and balancing the data on training dataset


jupyter Project-Data mining Last Checkpoint: 17 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

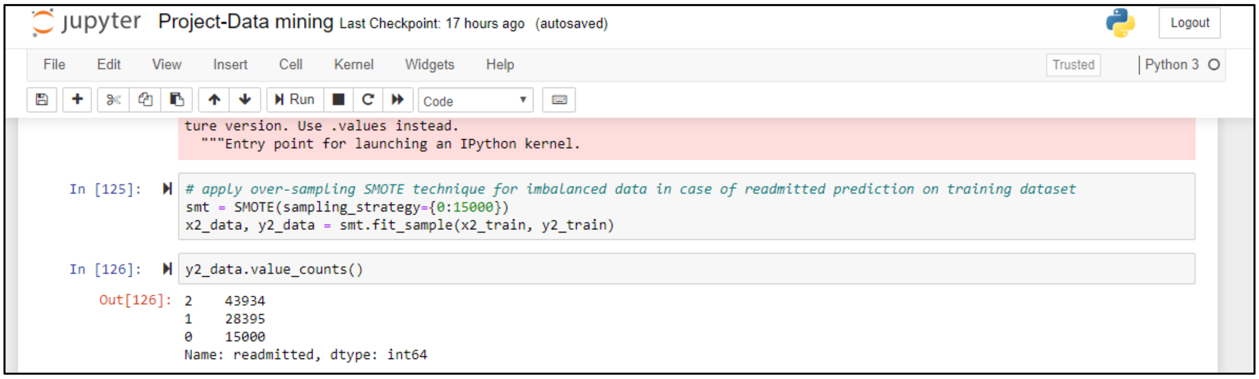
```
             """Entry point for launching an IPython kernel.

In [114]:  # apply over-sampling SMOTE technique for imbalanced data in case of diabetesMed prediction on training data set
           from imblearn.over_sampling import SMOTE
           smt = SMOTE(sampling_strategy=0.55)
           x_data, y_data = smt.fit_sample(x1_train, y1_train)

In [116]:  y_data.value_counts()

Out[116]:  1    62636
           0    34449
           Name: diabetesMed, dtype: int64
```

Split the data before SMOTE for readmitted variable


jupyter Project-Data mining Last Checkpoint: 17 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

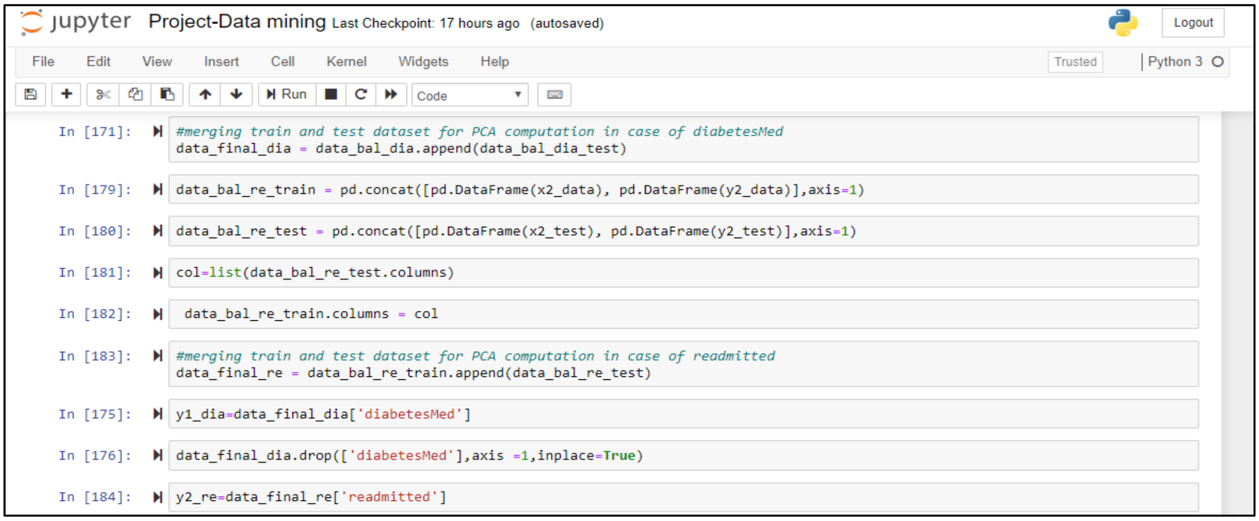
```
Out[120]: (97085, 201)

In [121]:  # split the data into train and test for readmitted prediction
           x2_train, x2_test, y2_train, y2_test = train_test_split(data_re, y_re, test_size=0.2)

In [122]:  y2_train.value_counts()

Out[122]: 2    43934
          1    28395
          0     9083
          Name: readmitted, dtype: int64
```

While applying SMOTE technique and balancing the data on training dataset


jupyter Project-Data mining Last Checkpoint: 17 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
ture version. Use .values instead.
  """Entry point for launching an IPython kernel.

In [125]:  # apply over-sampling SMOTE technique for imbalanced data in case of readmitted prediction on training dataset
           smt = SMOTE(sampling_strategy={0:15000})
           x2_data, y2_data = smt.fit_sample(x2_train, y2_train)

In [126]:  y2_data.value_counts()

Out[126]: 2    43934
          1    28395
          0    15000
          Name: readmitted, dtype: int64
```

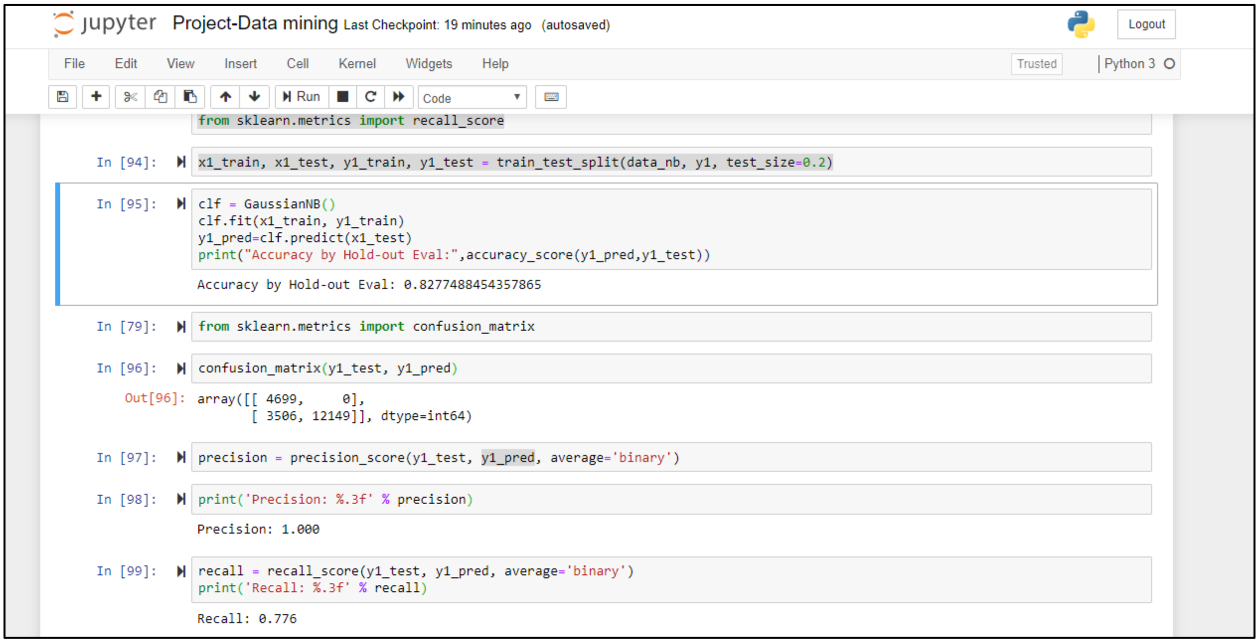
Merging datasets for PCA calculation


jupyter Project-Data mining Last Checkpoint: 17 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [171]:  #merging train and test dataset for PCA computation in case of diabetesMed
           data_final_dia = data_bal_dia.append(data_bal_dia_test)

In [179]:  data_bal_re_train = pd.concat([pd.DataFrame(x2_data), pd.DataFrame(y2_data)],axis=1)

In [180]:  data_bal_re_test = pd.concat([pd.DataFrame(x2_test), pd.DataFrame(y2_test)],axis=1)

In [181]:  col=list(data_bal_re_test.columns)

In [182]:   data_bal_re_train.columns = col

In [183]:  #merging train and test dataset for PCA computation in case of readmitted
           data_final_re = data_bal_re_train.append(data_bal_re_test)

In [175]:  y1_dia=data_final_dia['diabetesMed']

In [176]:  data_final_dia.drop(['diabetesMed'],axis =1,inplace=True)

In [184]:  y2_re=data_final_re['readmitted']
```

diabetesMed Prediction

Naïve Bayes results for diabetesMed before balancing data :


jupyter Project-Data mining Last Checkpoint: 19 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
from sklearn.metrics import recall_score
```

```
In [94]:  x1_train, x1_test, y1_train, y1_test = train_test_split(data_nb, y1, test_size=0.2)
```

```
In [95]:  clf = GaussianNB()
          clf.fit(x1_train, y1_train)
          y1_pred=clf.predict(x1_test)
          print("Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))

          Accuracy by Hold-out Eval: 0.8277488454357865
```

```
In [79]:  from sklearn.metrics import confusion_matrix
```

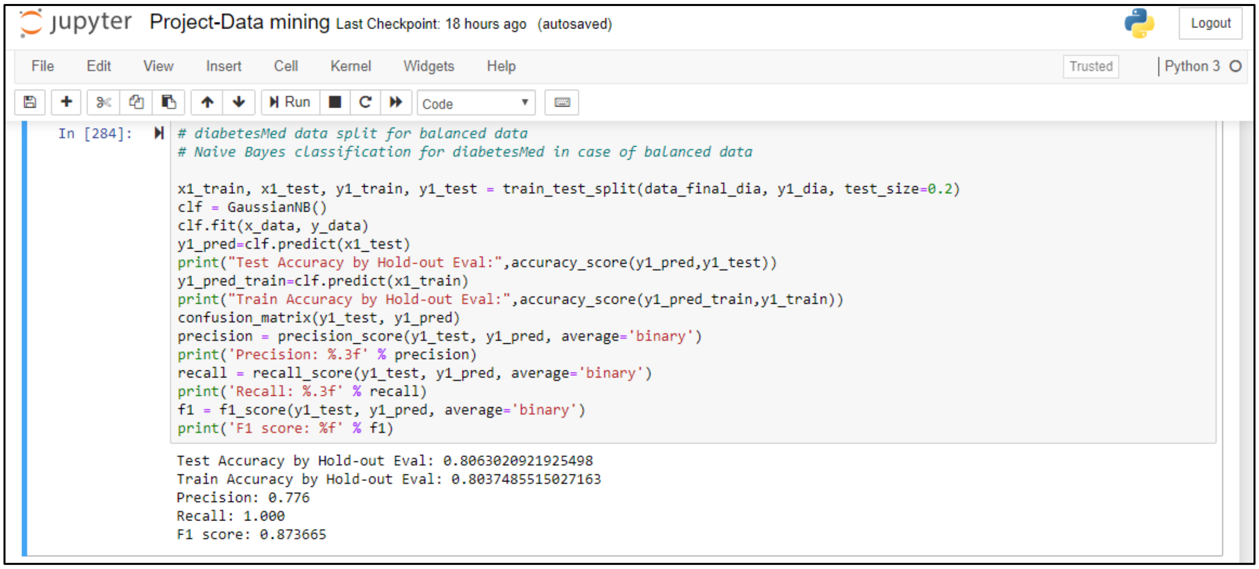
```
In [96]:  confusion_matrix(y1_test, y1_pred)

Out[96]:  array([[ 4699,     0],
                 [ 3506, 12149]], dtype=int64)
```

```
In [97]:  precision = precision_score(y1_test, y1_pred, average='binary')
```

```
In [98]:  print('Precision: %.3f' % precision)

          Precision: 1.000
```

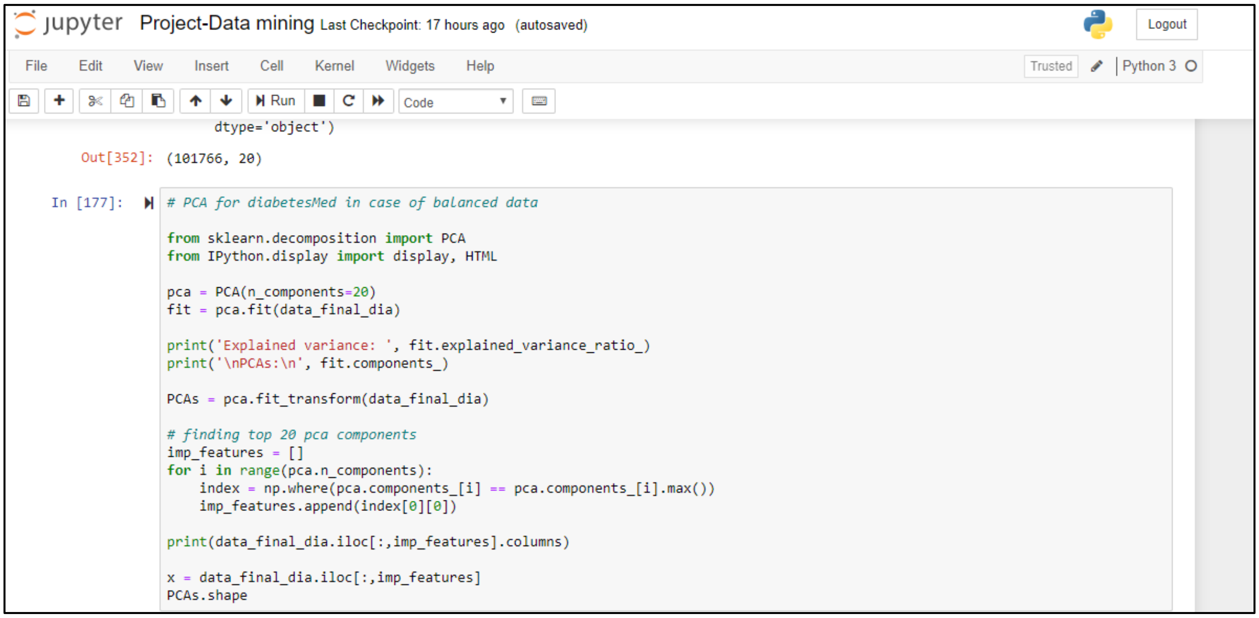
```
In [99]:  recall = recall_score(y1_test, y1_pred, average='binary')
          print('Recall: %.3f' % recall)

          Recall: 0.776
```

After applying SMOTE technique on training dataset :

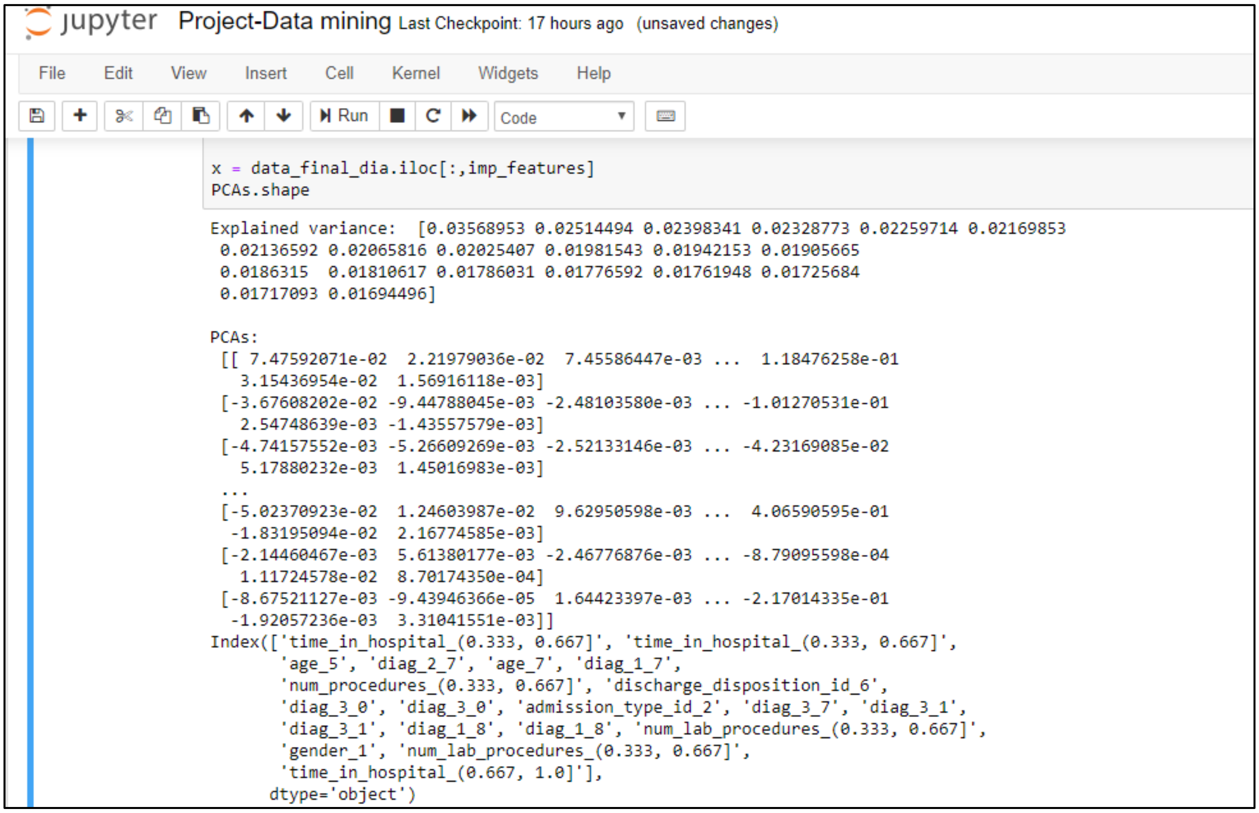

jupyter Project-Data mining Last Checkpoint: 18 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [284]:  # diabetesMed data split for balanced data
           # Naive Bayes classification for diabetesMed in case of balanced data

           x1_train, x1_test, y1_train, y1_test = train_test_split(data_final_dia, y1_dia, test_size=0.2)
           clf = GaussianNB()
           clf.fit(x_data, y_data)
           y1_pred=clf.predict(x1_test)
           print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
           y1_pred_train=clf.predict(x1_train)
           print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
           confusion_matrix(y1_test, y1_pred)
           precision = precision_score(y1_test, y1_pred, average='binary')
           print('Precision: %.3f' % precision)
           recall = recall_score(y1_test, y1_pred, average='binary')
           print('Recall: %.3f' % recall)
           f1 = f1_score(y1_test, y1_pred, average='binary')
           print('F1 score: %f' % f1)

           Test Accuracy by Hold-out Eval: 0.8063020921925498
           Train Accuracy by Hold-out Eval: 0.8037485515027163
           Precision: 0.776
           Recall: 1.000
           F1 score: 0.873665
```

PCA with the Top-20 variables :


jupyter Project-Data mining Last Checkpoint: 17 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

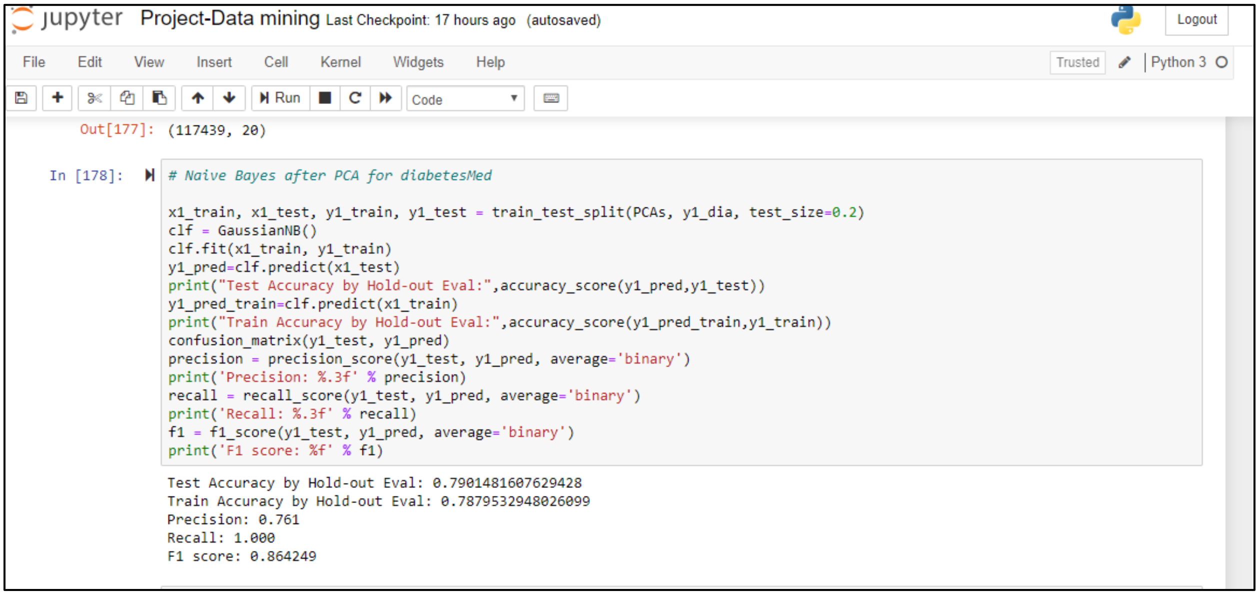
```
                dtype='object')

      Out[352]: (101766, 20)

      In [177]:  # PCA for diabetesMed in case of balanced data

                 from sklearn.decomposition import PCA
                 from IPython.display import display, HTML

                 pca = PCA(n_components=20)
                 fit = pca.fit(data_final_dia)

                 print('Explained variance: ', fit.explained_variance_ratio_)
                 print('\nPCAs:\n', fit.components_)

                 PCAs = pca.fit_transform(data_final_dia)

                 # finding top 20 pca components
                 imp_features = []
                 for i in range(pca.n_components):
                     index = np.where(pca.components_[i] == pca.components_[i].max())
                     imp_features.append(index[0][0])

                 print(data_final_dia.iloc[:,imp_features].columns)

                 x = data_final_dia.iloc[:,imp_features]
                 PCAs.shape
```


jupyter Project-Data mining Last Checkpoint: 17 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
x = data_final_dia.iloc[:,imp_features]
PCAs.shape
```

```
Explained variance:  [0.03568953 0.02514494 0.02398341 0.02328773 0.02259714 0.02169853
 0.02136592 0.02065816 0.02025407 0.01981543 0.01942153 0.01905665
 0.0186315  0.01810617 0.01786031 0.01776592 0.01761948 0.01725684
 0.01717093 0.01694496]

PCAs:
 [[ 7.47592071e-02  2.21979036e-02  7.45586447e-03 ...  1.18476258e-01
   3.15436954e-02  1.56916118e-03]
 [-3.67608202e-02 -9.44788045e-03 -2.48103580e-03 ... -1.01270531e-01
   2.54748639e-03 -1.43557579e-03]
 [-4.74157552e-03 -5.26609269e-03 -2.52133146e-03 ... -4.23169085e-02
   5.17880232e-03  1.45016983e-03]
 ...
 [-5.02370923e-02  1.24603987e-02  9.62950598e-03 ...  4.06590595e-01
  -1.83195094e-02  2.16774585e-03]
 [-2.14460467e-03  5.61380177e-03 -2.46776876e-03 ... -8.79095598e-04
   1.11724578e-02  8.70174350e-04]
 [-8.67521127e-03 -9.43946366e-05  1.64423397e-03 ... -2.17014335e-01
  -1.92057236e-03  3.31041551e-03]]
Index(['time_in_hospital_(0.333, 0.667]', 'time_in_hospital_(0.333, 0.667]',
       'age_5', 'diag_2_7', 'age_7', 'diag_1_7',
       'num_procedures_(0.333, 0.667]', 'discharge_disposition_id_6',
       'diag_3_0', 'diag_3_0', 'admission_type_id_2', 'diag_3_7', 'diag_3_1',
       'diag_3_1', 'diag_1_8', 'diag_1_8', 'num_lab_procedures_(0.333, 0.667]',
       'gender_1', 'num_lab_procedures_(0.333, 0.667]',
       'time_in_hospital_(0.667, 1.0]'],
      dtype='object')
```

Naïve Bayes after applying PCA

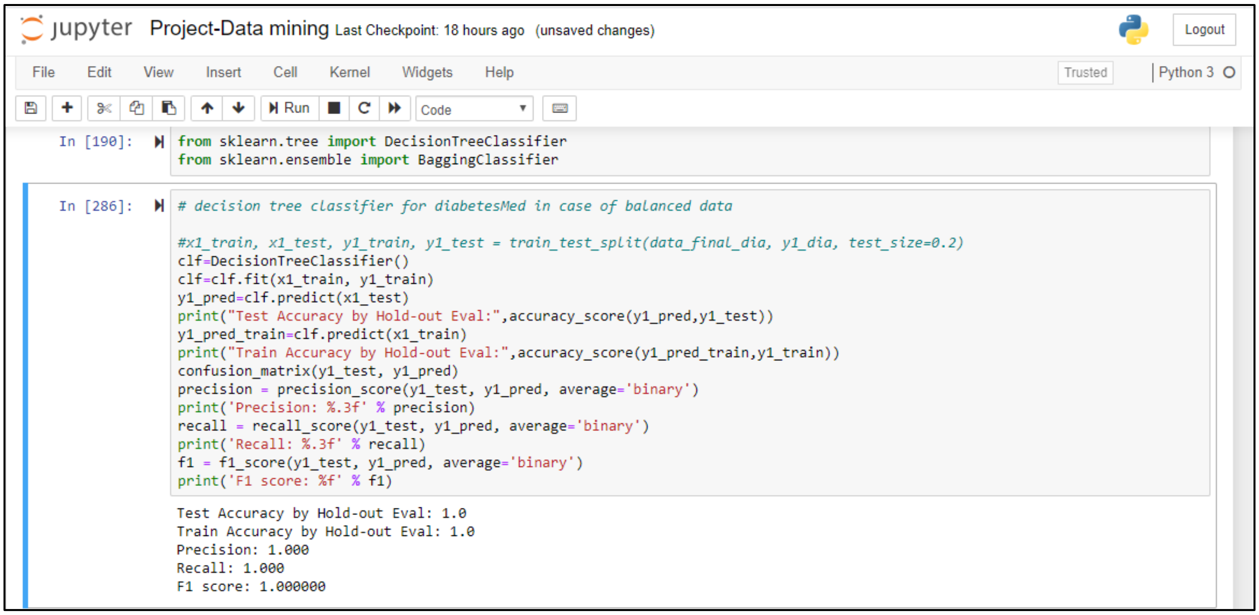


```
Out[177]: (117439, 20)
```

```
In [178]:
# Naive Bayes after PCA for diabetesMed

x1_train, x1_test, y1_train, y1_test = train_test_split(PCAs, y1_dia, test_size=0.2)
clf = GaussianNB()
clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
y1_pred_train=clf.predict(x1_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)
```

```
Test Accuracy by Hold-out Eval: 0.7901481607629428
Train Accuracy by Hold-out Eval: 0.7879532948026099
Precision: 0.761
Recall: 1.000
F1 score: 0.864249
```

Decision tree without applying PCA :

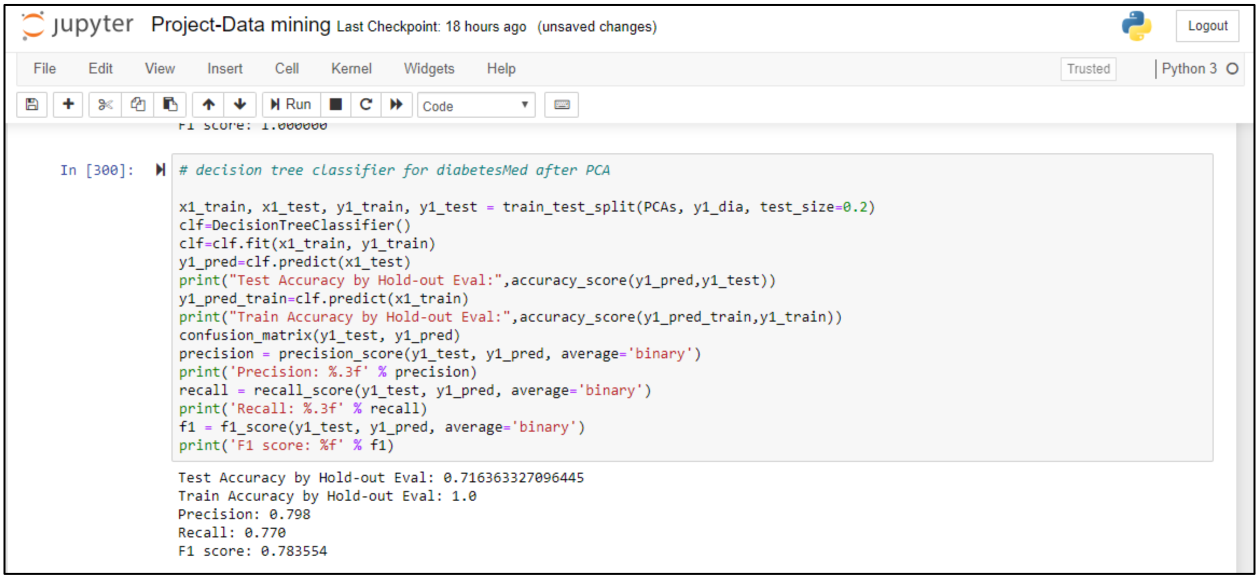


```
In [190]:
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
```

```
In [286]:
# decision tree classifier for diabetesMed in case of balanced data

#x1_train, x1_test, y1_train, y1_test = train_test_split(data_final_dia, y1_dia, test_size=0.2)
clf=DecisionTreeClassifier()
clf=clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
y1_pred_train=clf.predict(x1_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)
```

```
Test Accuracy by Hold-out Eval: 1.0
Train Accuracy by Hold-out Eval: 1.0
Precision: 1.000
Recall: 1.000
F1 score: 1.000000
```

Decision tree after using PCA



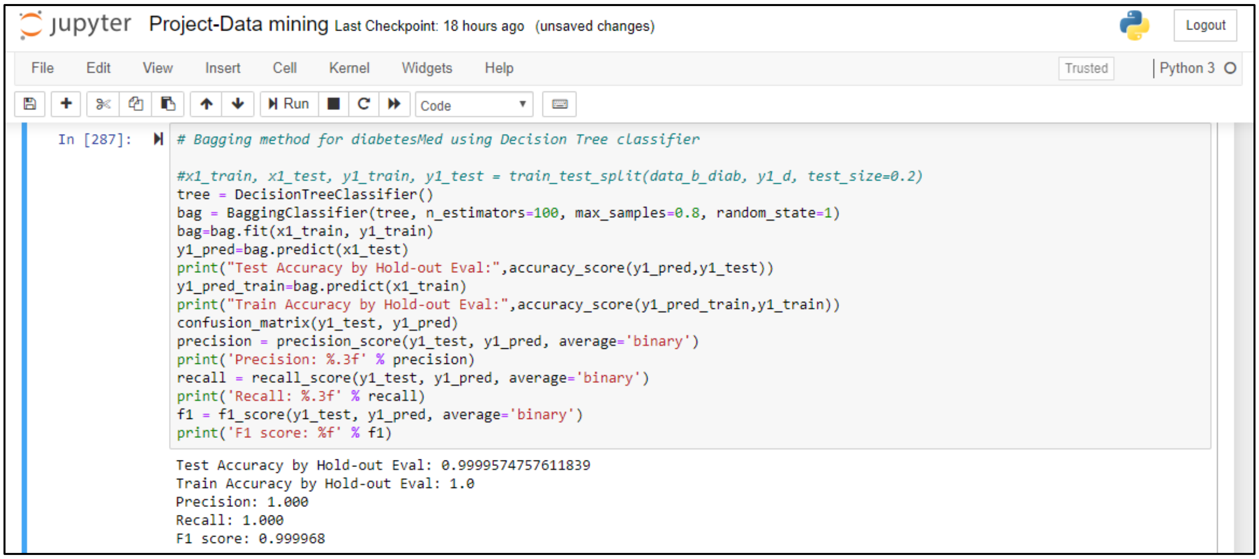
```
F1 score: 1.000000
```

In [300]:

```
# decision tree classifier for diabetesMed after PCA

x1_train, x1_test, y1_train, y1_test = train_test_split(PCAs, y1_dia, test_size=0.2)
clf=DecisionTreeClassifier()
clf=clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
y1_pred_train=clf.predict(x1_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)
```

```
Test Accuracy by Hold-out Eval: 0.716363327096445
Train Accuracy by Hold-out Eval: 1.0
Precision: 0.798
Recall: 0.770
F1 score: 0.783554
```

Bagging method using decision tree classifier without applying PCA :

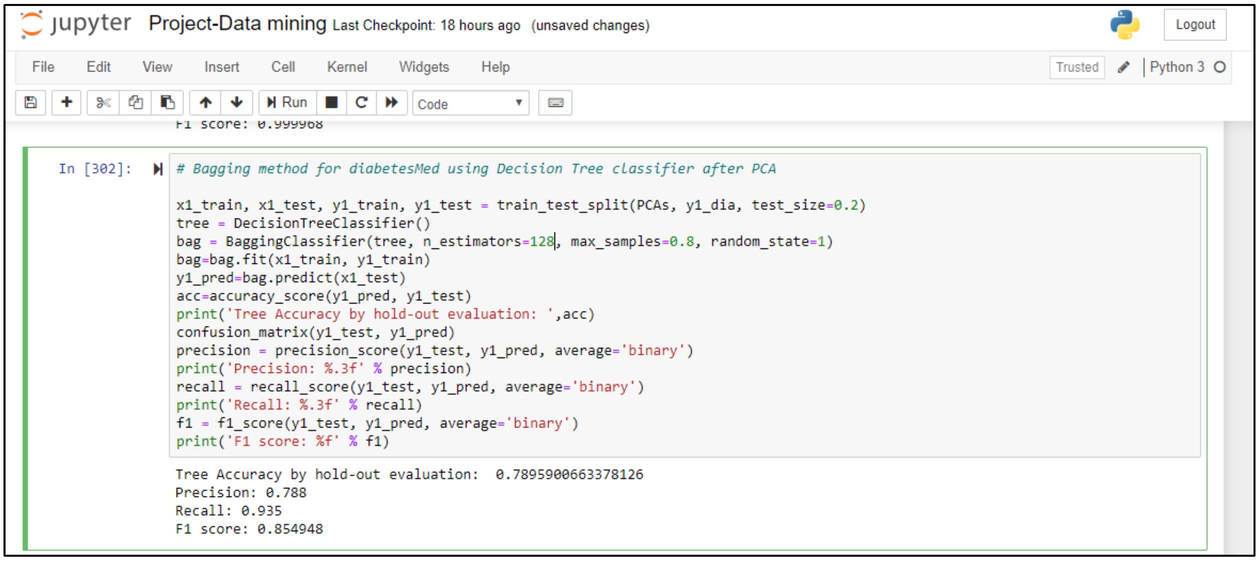


In [287]:

```
# Bagging method for diabetesMed using Decision Tree classifier

#x1_train, x1_test, y1_train, y1_test = train_test_split(data_b_diab, y1_d, test_size=0.2)
tree = DecisionTreeClassifier()
bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8, random_state=1)
bag=bag.fit(x1_train, y1_train)
y1_pred=bag.predict(x1_test)
print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
y1_pred_train=bag.predict(x1_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)
```

```
Test Accuracy by Hold-out Eval: 0.9999574757611839
Train Accuracy by Hold-out Eval: 1.0
Precision: 1.000
Recall: 1.000
F1 score: 0.999968
```

Bagging method using Decision tree classifier after PCA

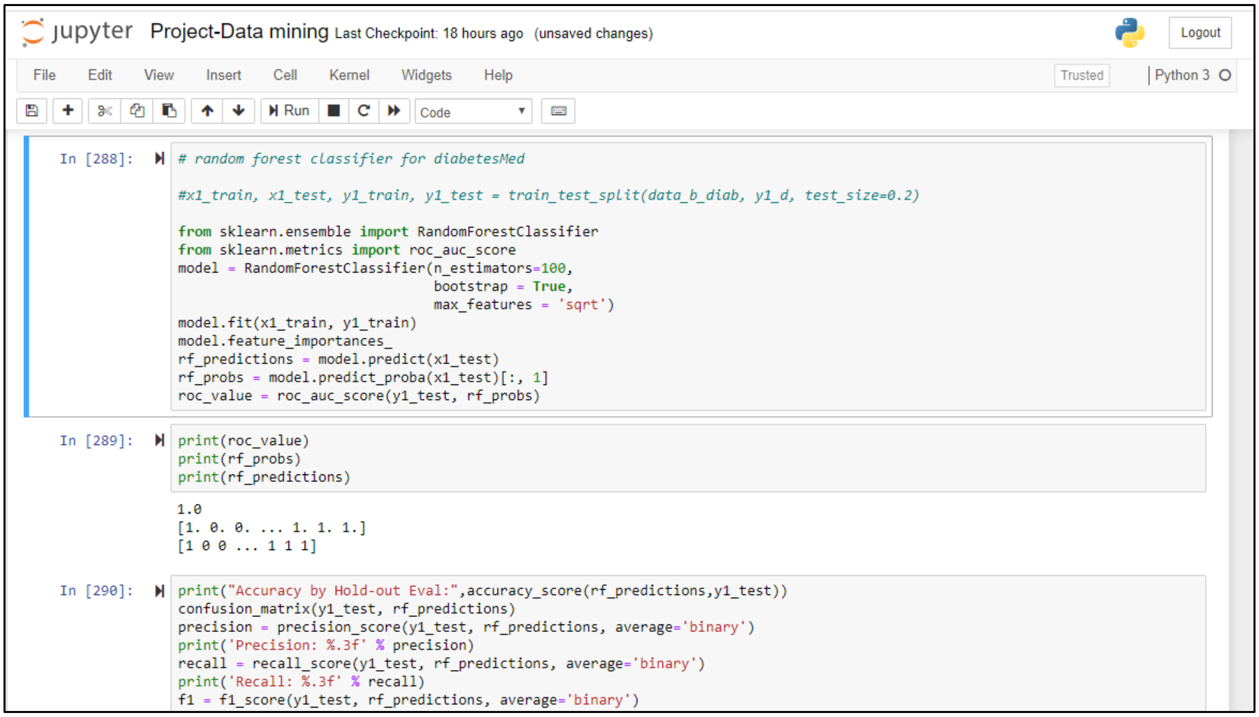

jupyter Project-Data mining Last Checkpoint: 18 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
F1 score: 0.999968
```

In [302]:
```
# Bagging method for diabetesMed using Decision Tree classifier after PCA

x1_train, x1_test, y1_train, y1_test = train_test_split(PCAs, y1_dia, test_size=0.2)
tree = DecisionTreeClassifier()
bag = BaggingClassifier(tree, n_estimators=128, max_samples=0.8, random_state=1)
bag=bag.fit(x1_train, y1_train)
y1_pred=bag.predict(x1_test)
acc=accuracy_score(y1_pred, y1_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)
```

```
Tree Accuracy by hold-out evaluation:  0.7895900663378126
Precision: 0.788
Recall: 0.935
F1 score: 0.854948
```

Random forest without applying PCA :

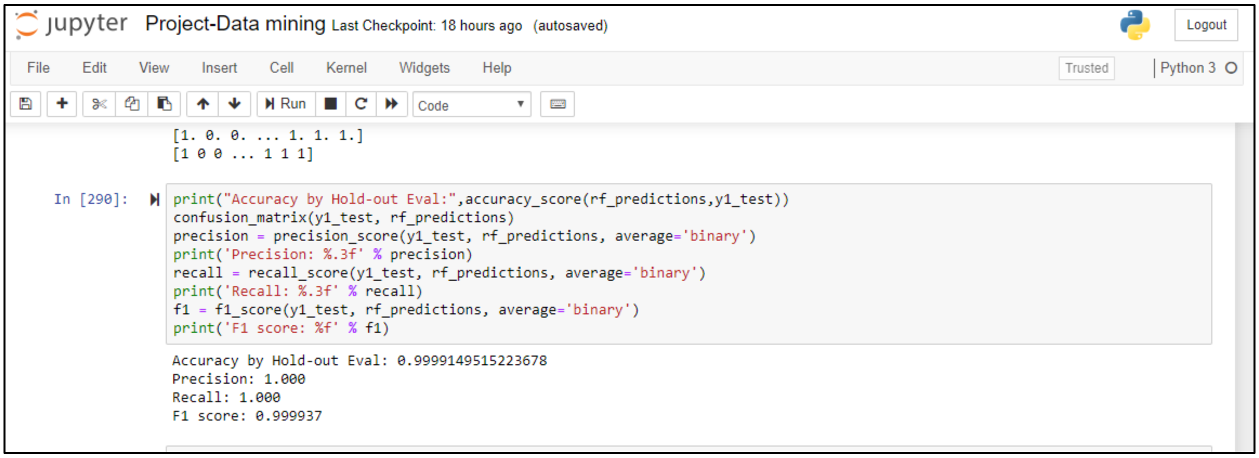

jupyter Project-Data mining Last Checkpoint: 18 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

In [288]:
```
# random forest classifier for diabetesMed

#x1_train, x1_test, y1_train, y1_test = train_test_split(data_b_diab, y1_d, test_size=0.2)

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
model = RandomForestClassifier(n_estimators=100,
                               bootstrap = True,
                               max_features = 'sqrt')
model.fit(x1_train, y1_train)
model.feature_importances_
rf_predictions = model.predict(x1_test)
rf_probs = model.predict_proba(x1_test)[:, 1]
roc_value = roc_auc_score(y1_test, rf_probs)
```

In [289]:
```
print(roc_value)
print(rf_probs)
print(rf_predictions)
```

```
1.0
[1. 0. 0. ... 1. 1. 1.]
[1 0 0 ... 1 1 1]
```

In [290]:
```
print("Accuracy by Hold-out Eval:",accuracy_score(rf_predictions,y1_test))
confusion_matrix(y1_test, rf_predictions)
precision = precision_score(y1_test, rf_predictions, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, rf_predictions, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, rf_predictions, average='binary')
```

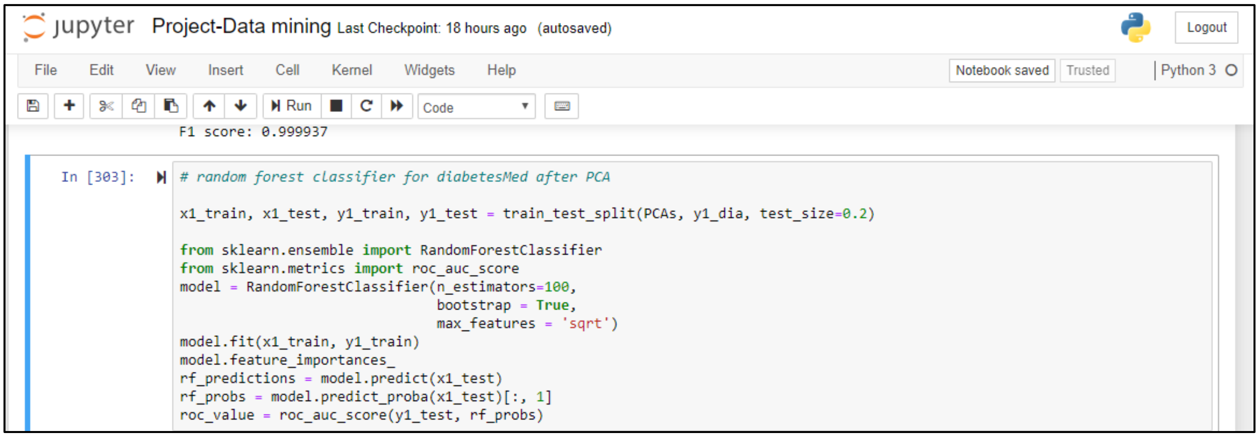

jupyter Project-Data mining Last Checkpoint: 18 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
[1. 0. 0. ... 1. 1. 1.]
[1 0 0 ... 1 1 1]
```

```
In [290]:  print("Accuracy by Hold-out Eval:",accuracy_score(rf_predictions,y1_test))
           confusion_matrix(y1_test, rf_predictions)
           precision = precision_score(y1_test, rf_predictions, average='binary')
           print('Precision: %.3f' % precision)
           recall = recall_score(y1_test, rf_predictions, average='binary')
           print('Recall: %.3f' % recall)
           f1 = f1_score(y1_test, rf_predictions, average='binary')
           print('F1 score: %f' % f1)
```

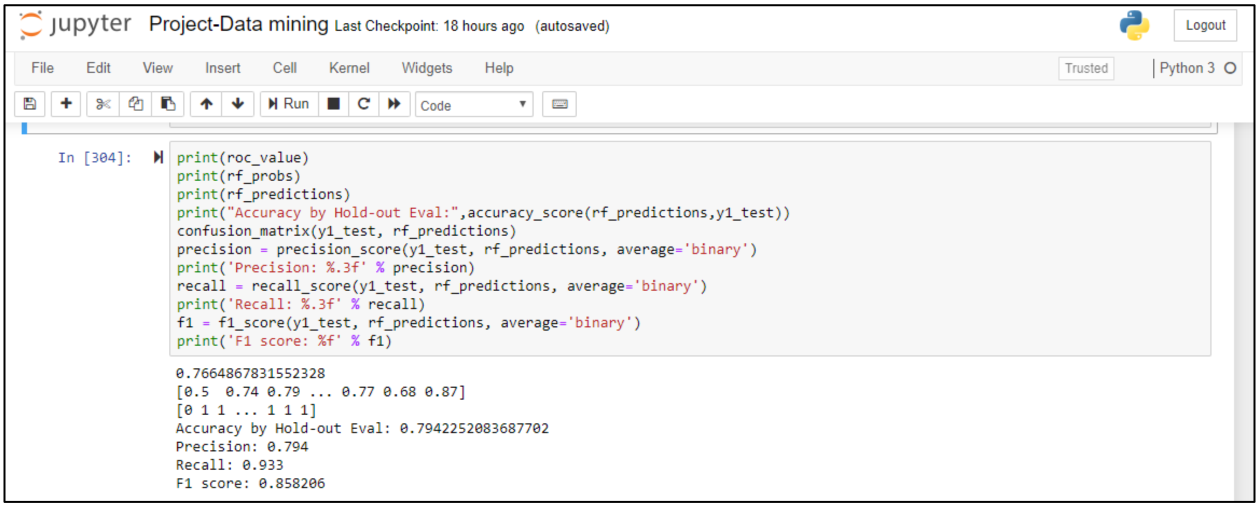
```
Accuracy by Hold-out Eval: 0.9999149515223678
Precision: 1.000
Recall: 1.000
F1 score: 0.999937
```

Random forest after PCA


jupyter Project-Data mining Last Checkpoint: 18 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Notebook saved
Trusted
Python 3
Run
Code

```
F1 score: 0.999937
```

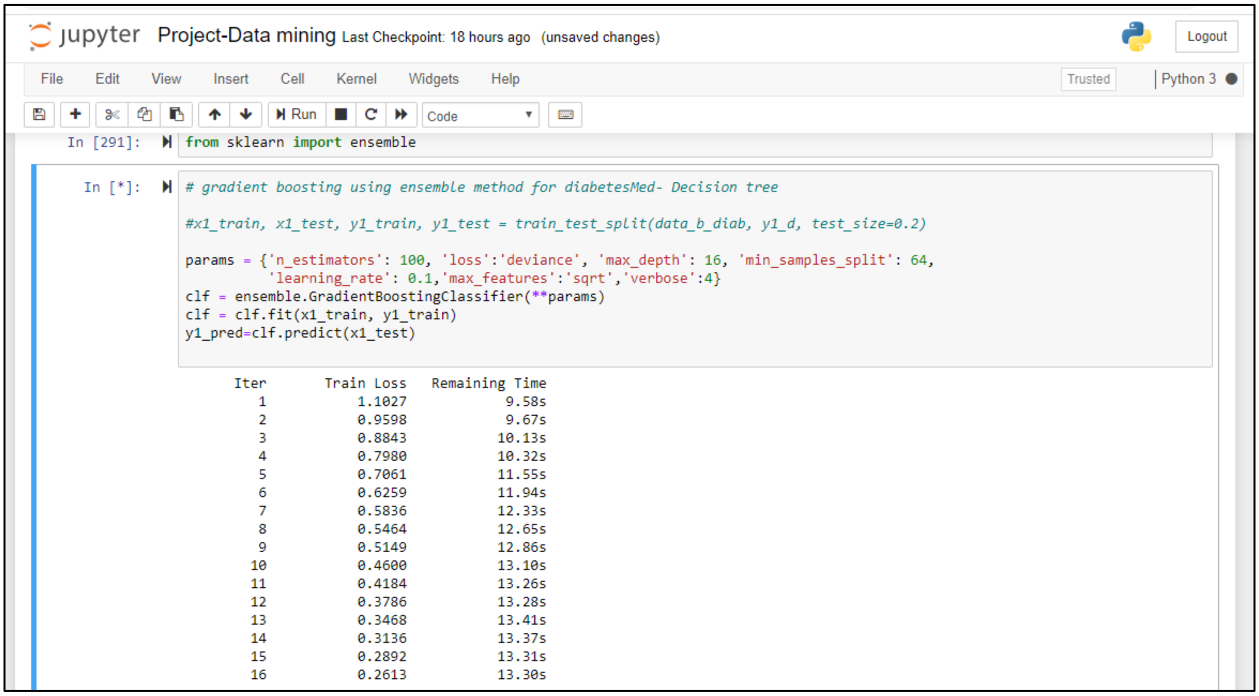
```
In [303]:  # random forest classifier for diabetesMed after PCA

           x1_train, x1_test, y1_train, y1_test = train_test_split(PCAs, y1_dia, test_size=0.2)

           from sklearn.ensemble import RandomForestClassifier
           from sklearn.metrics import roc_auc_score
           model = RandomForestClassifier(n_estimators=100,
                                          bootstrap = True,
                                          max_features = 'sqrt')
           model.fit(x1_train, y1_train)
           model.feature_importances_
           rf_predictions = model.predict(x1_test)
           rf_probs = model.predict_proba(x1_test)[:, 1]
           roc_value = roc_auc_score(y1_test, rf_probs)
```


jupyter Project-Data mining Last Checkpoint: 18 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [304]:  print(roc_value)
           print(rf_probs)
           print(rf_predictions)
           print("Accuracy by Hold-out Eval:",accuracy_score(rf_predictions,y1_test))
           confusion_matrix(y1_test, rf_predictions)
           precision = precision_score(y1_test, rf_predictions, average='binary')
           print('Precision: %.3f' % precision)
           recall = recall_score(y1_test, rf_predictions, average='binary')
           print('Recall: %.3f' % recall)
           f1 = f1_score(y1_test, rf_predictions, average='binary')
           print('F1 score: %f' % f1)
```

```
0.7664867831552328
[0.5  0.74 0.79 ... 0.77 0.68 0.87]
[0 1 1 ... 1 1 1]
Accuracy by Hold-out Eval: 0.7942252083687702
Precision: 0.794
Recall: 0.933
F1 score: 0.858206
```

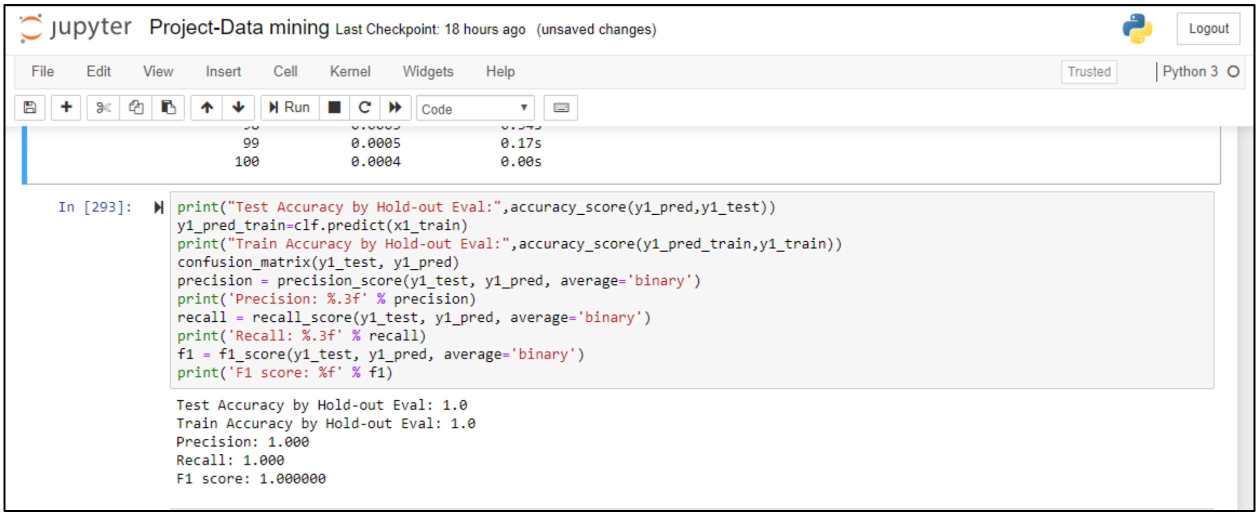
Gradient boosting using ensemble method – Decision tree classifier without applying PCA :



```
In [291]:  from sklearn import ensemble
```

```
In [*]:  # gradient boosting using ensemble method for diabetesMed- Decision tree

         #x1_train, x1_test, y1_train, y1_test = train_test_split(data_b_diab, y1_d, test_size=0.2)

         params = {'n_estimators': 100, 'loss':'deviance', 'max_depth': 16, 'min_samples_split': 64,
                   'learning_rate': 0.1,'max_features':'sqrt','verbose':4}
         clf = ensemble.GradientBoostingClassifier(**params)
         clf = clf.fit(x1_train, y1_train)
         y1_pred=clf.predict(x1_test)
```

```
      Iter       Train Loss   Remaining Time 
         1           1.1027            9.58s
         2           0.9598            9.67s
         3           0.8843           10.13s
         4           0.7980           10.32s
         5           0.7061           11.55s
         6           0.6259           11.94s
         7           0.5836           12.33s
         8           0.5464           12.65s
         9           0.5149           12.86s
        10           0.4600           13.10s
        11           0.4184           13.26s
        12           0.3786           13.28s
        13           0.3468           13.41s
        14           0.3136           13.37s
        15           0.2892           13.31s
        16           0.2613           13.30s
```

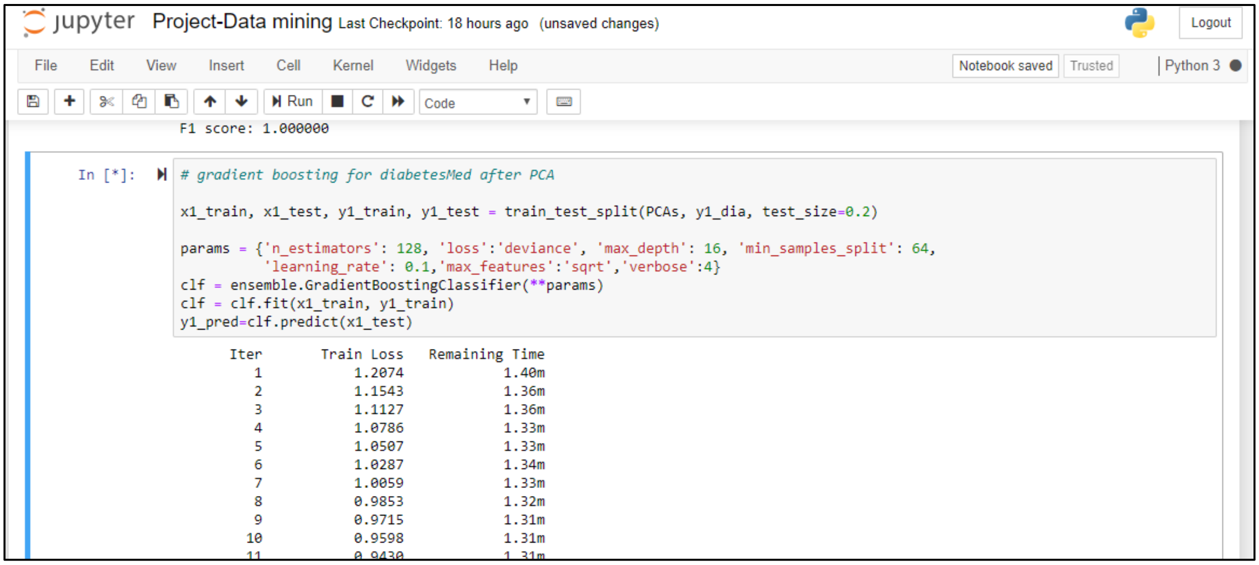


```
        99           0.0005            0.17s
       100           0.0004            0.00s
```

```
In [293]:  print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
           y1_pred_train=clf.predict(x1_train)
           print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
           confusion_matrix(y1_test, y1_pred)
           precision = precision_score(y1_test, y1_pred, average='binary')
           print('Precision: %.3f' % precision)
           recall = recall_score(y1_test, y1_pred, average='binary')
           print('Recall: %.3f' % recall)
           f1 = f1_score(y1_test, y1_pred, average='binary')
           print('F1 score: %f' % f1)
```

```
Test Accuracy by Hold-out Eval: 1.0
Train Accuracy by Hold-out Eval: 1.0
Precision: 1.000
Recall: 1.000
F1 score: 1.000000
```

Gradient Boosting using ensemble method – Decision tree classifier after PCA

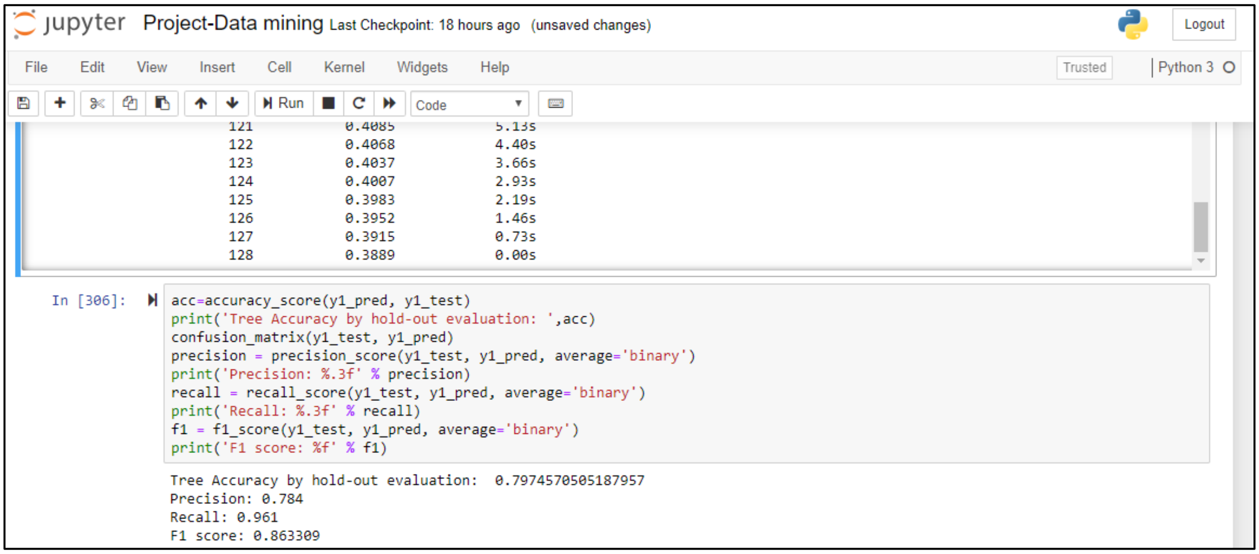


```
F1 score: 1.000000
```

In [*]:

```python
# gradient boosting for diabetesMed after PCA

x1_train, x1_test, y1_train, y1_test = train_test_split(PCAs, y1_dia, test_size=0.2)

params = {'n_estimators': 128, 'loss':'deviance', 'max_depth': 16, 'min_samples_split': 64,
          'learning_rate': 0.1,'max_features':'sqrt','verbose':4}
clf = ensemble.GradientBoostingClassifier(**params)
clf = clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
```

```
      Iter       Train Loss   Remaining Time
         1           1.2074            1.40m
         2           1.1543            1.36m
         3           1.1127            1.36m
         4           1.0786            1.33m
         5           1.0507            1.33m
         6           1.0287            1.34m
         7           1.0059            1.33m
         8           0.9853            1.32m
         9           0.9715            1.31m
        10           0.9598            1.31m
        11           0.9430            1.31m
```



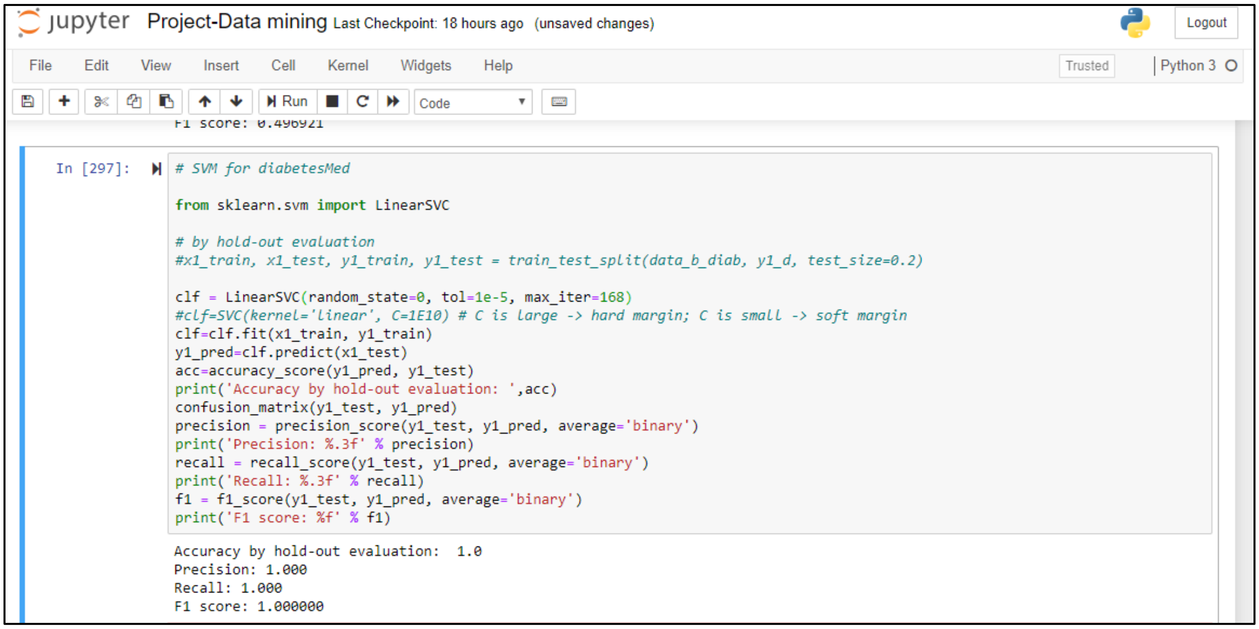
```
       121           0.4085            5.13s
       122           0.4068            4.40s
       123           0.4037            3.66s
       124           0.4007            2.93s
       125           0.3983            2.19s
       126           0.3952            1.46s
       127           0.3915            0.73s
       128           0.3889            0.00s
```

In [306]:

```python
acc=accuracy_score(y1_pred, y1_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)
```

```
Tree Accuracy by hold-out evaluation:  0.7974570505187957
Precision: 0.784
Recall: 0.961
F1 score: 0.863309
```

SVM (Support Vector Machines) without applying PCA :


jupyter
Project-Data mining Last Checkpoint: 18 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

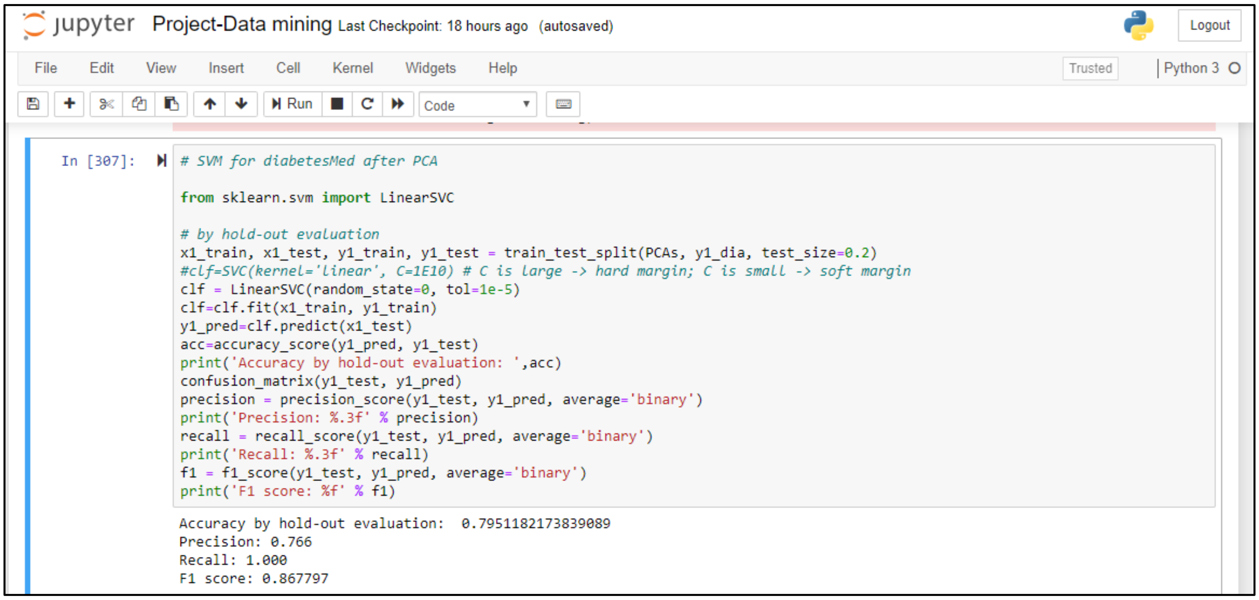
```
F1 score: 0.496921
```

In [297]:

```
# SVM for diabetesMed

from sklearn.svm import LinearSVC

# by hold-out evaluation
#x1_train, x1_test, y1_train, y1_test = train_test_split(data_b_diab, y1_d, test_size=0.2)

clf = LinearSVC(random_state=0, tol=1e-5, max_iter=168)
#clf=SVC(kernel='linear', C=1E10) # C is large -> hard margin; C is small -> soft margin
clf=clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
acc=accuracy_score(y1_pred, y1_test)
print('Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)
```

```
Accuracy by hold-out evaluation:  1.0
Precision: 1.000
Recall: 1.000
F1 score: 1.000000
```

SVM (Support Vector Machines) after using PCA


jupyter
Project-Data mining Last Checkpoint: 18 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

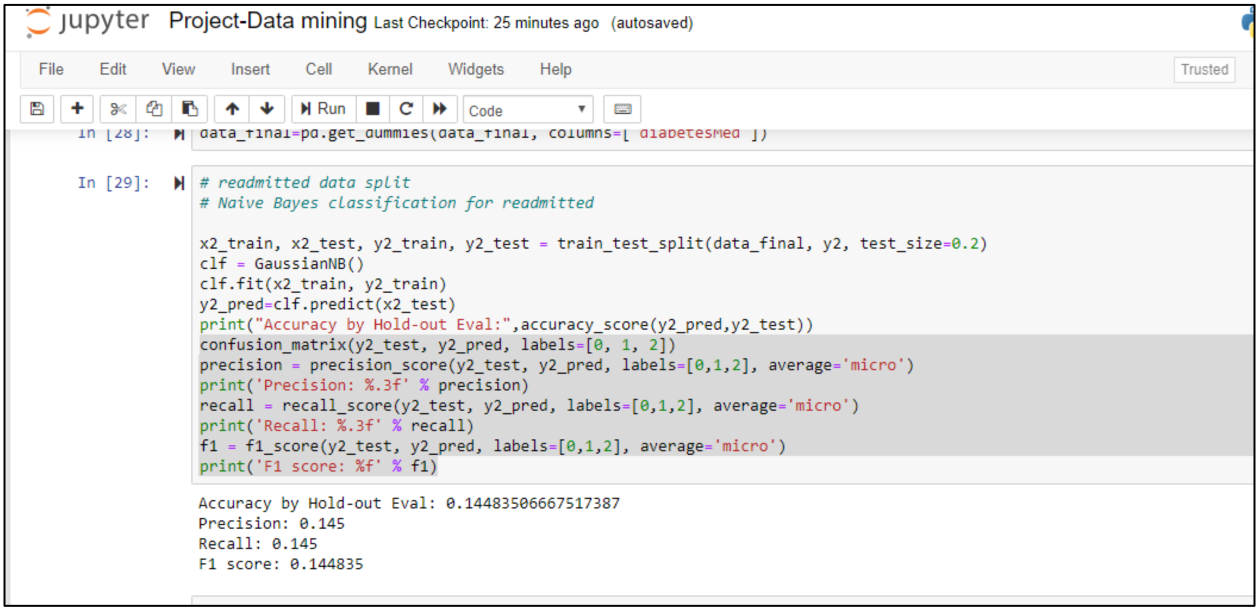
In [307]:

```
# SVM for diabetesMed after PCA

from sklearn.svm import LinearSVC

# by hold-out evaluation
x1_train, x1_test, y1_train, y1_test = train_test_split(PCAs, y1_dia, test_size=0.2)
#clf=SVC(kernel='linear', C=1E10) # C is large -> hard margin; C is small -> soft margin
clf = LinearSVC(random_state=0, tol=1e-5)
clf=clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
acc=accuracy_score(y1_pred, y1_test)
print('Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)
```

```
Accuracy by hold-out evaluation:  0.7951182173839089
Precision: 0.766
Recall: 1.000
F1 score: 0.867797
```

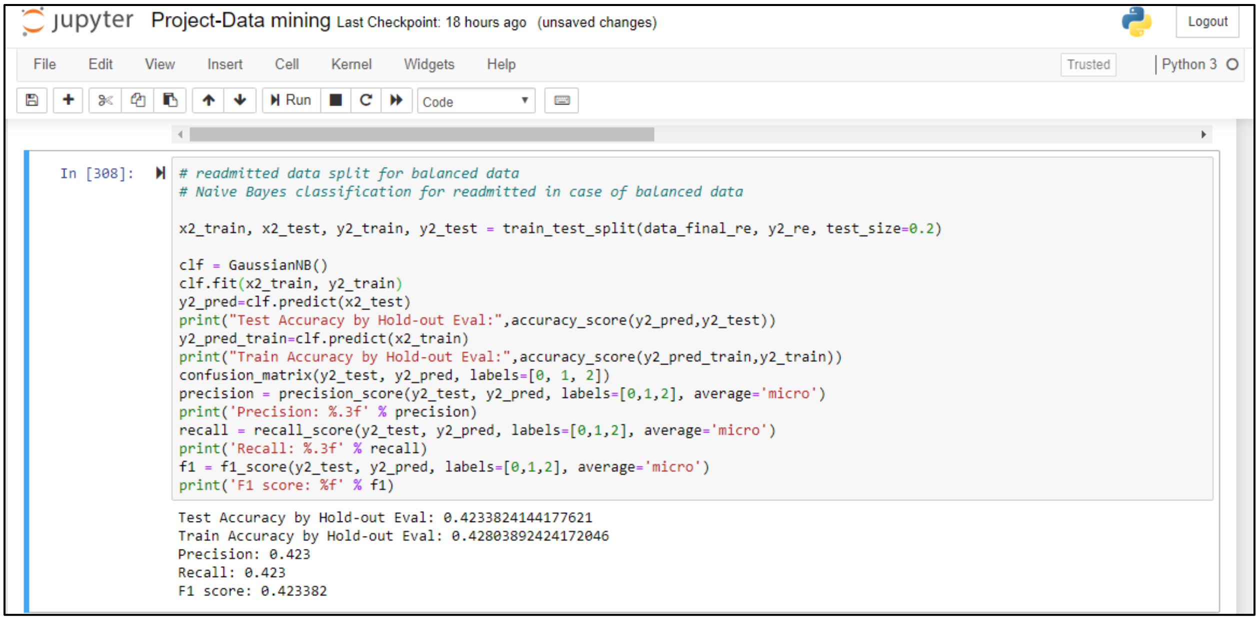
Readmitted variable

Naive Bayes for readmitted in case of imbalanced data


jupyter
Project-Data mining Last Checkpoint: 25 minutes ago (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Run
Code

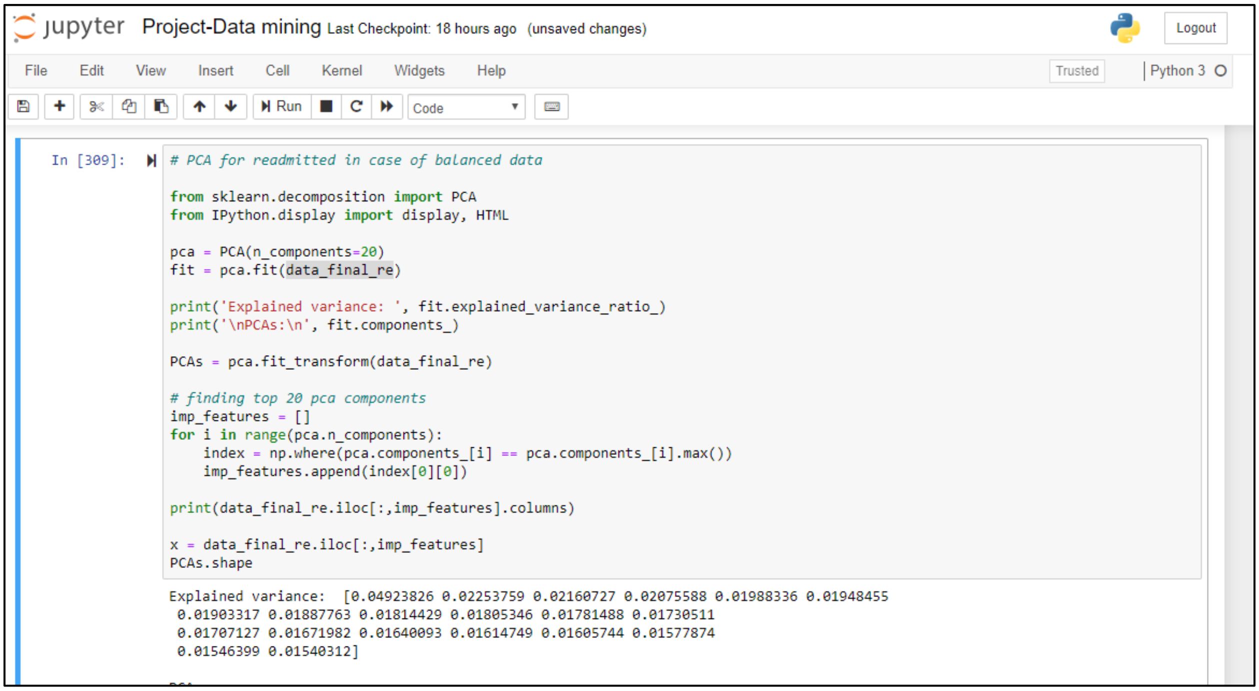
```
In [28]: data_final=pd.get_dummies(data_final, columns=['diabetesMed'])

In [29]: # readmitted data split
         # Naive Bayes classification for readmitted

         x2_train, x2_test, y2_train, y2_test = train_test_split(data_final, y2, test_size=0.2)
         clf = GaussianNB()
         clf.fit(x2_train, y2_train)
         y2_pred=clf.predict(x2_test)
         print("Accuracy by Hold-out Eval:",accuracy_score(y2_pred,y2_test))
         confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
         precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
         print('Precision: %.3f' % precision)
         recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
         print('Recall: %.3f' % recall)
         f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
         print('F1 score: %f' % f1)

         Accuracy by Hold-out Eval: 0.14483506667517387
         Precision: 0.145
         Recall: 0.145
         F1 score: 0.144835
```

Naïve Bayes on balanced data



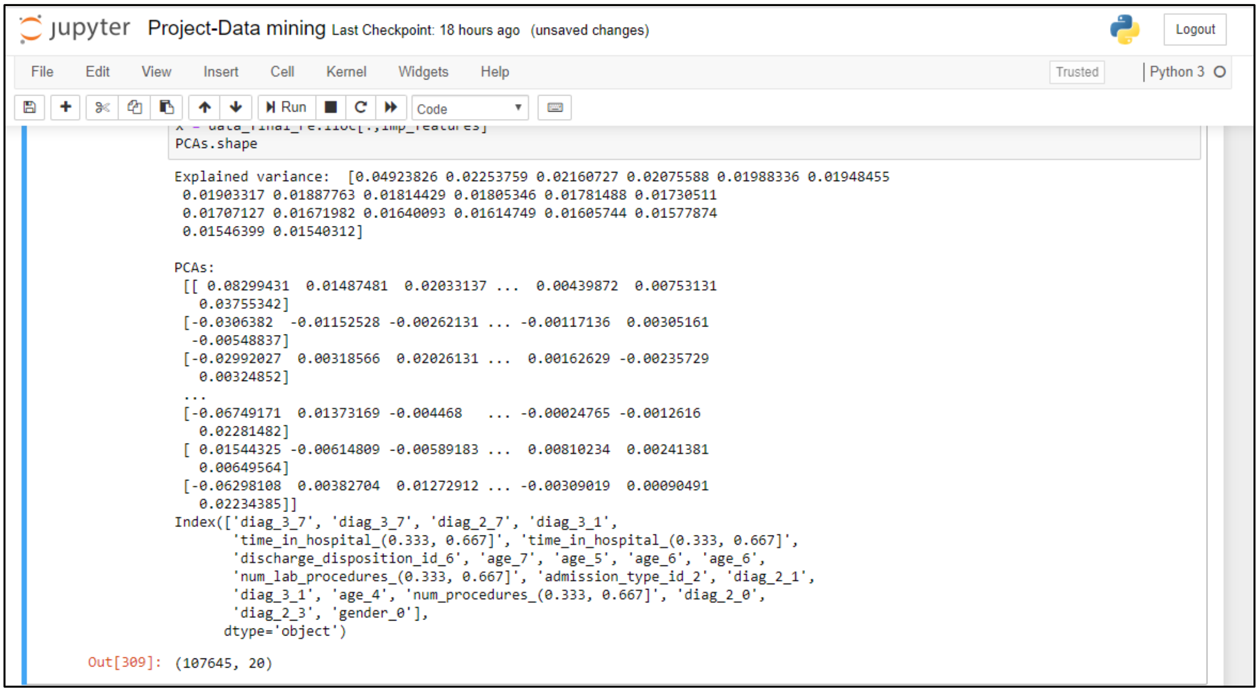
```
In [308]: # readmitted data split for balanced data
          # Naive Bayes classification for readmitted in case of balanced data

          x2_train, x2_test, y2_train, y2_test = train_test_split(data_final_re, y2_re, test_size=0.2)

          clf = GaussianNB()
          clf.fit(x2_train, y2_train)
          y2_pred=clf.predict(x2_test)
          print("Test Accuracy by Hold-out Eval:",accuracy_score(y2_pred,y2_test))
          y2_pred_train=clf.predict(x2_train)
          print("Train Accuracy by Hold-out Eval:",accuracy_score(y2_pred_train,y2_train))
          confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
          precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
          print('Precision: %.3f' % precision)
          recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
          print('Recall: %.3f' % recall)
          f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
          print('F1 score: %f' % f1)

          Test Accuracy by Hold-out Eval: 0.4233824144177621
          Train Accuracy by Hold-out Eval: 0.42803892424172046
          Precision: 0.423
          Recall: 0.423
          F1 score: 0.423382
```

PCA Top 20 features


jupyter
Project-Data mining Last Checkpoint: 18 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [309]:
# PCA for readmitted in case of balanced data

from sklearn.decomposition import PCA
from IPython.display import display, HTML

pca = PCA(n_components=20)
fit = pca.fit(data_final_re)

print('Explained variance: ', fit.explained_variance_ratio_)
print('\nPCAs:\n', fit.components_)

PCAs = pca.fit_transform(data_final_re)

# finding top 20 pca components
imp_features = []
for i in range(pca.n_components):
    index = np.where(pca.components_[i] == pca.components_[i].max())
    imp_features.append(index[0][0])

print(data_final_re.iloc[:,imp_features].columns)

x = data_final_re.iloc[:,imp_features]
PCAs.shape
```

```
Explained variance:  [0.04923826 0.02253759 0.02160727 0.02075588 0.01988336 0.01948455
 0.01903317 0.01887763 0.01814429 0.01805346 0.01781488 0.01730511
 0.01707127 0.01671982 0.01640093 0.01614749 0.01605744 0.01577874
 0.01546399 0.01540312]
```

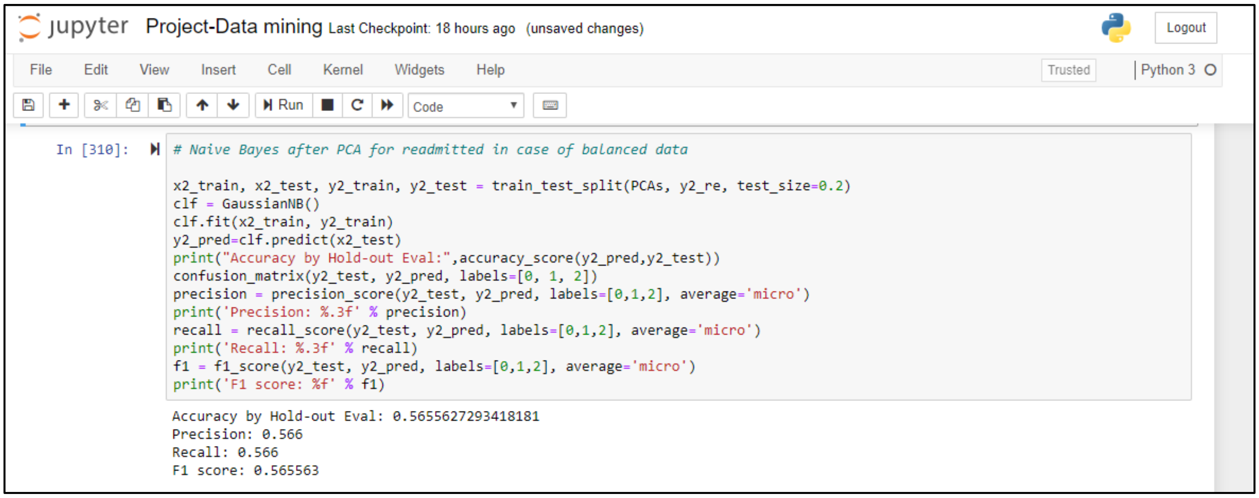

jupyter
Project-Data mining Last Checkpoint: 18 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
PCAs.shape
```

```
Explained variance:  [0.04923826 0.02253759 0.02160727 0.02075588 0.01988336 0.01948455
 0.01903317 0.01887763 0.01814429 0.01805346 0.01781488 0.01730511
 0.01707127 0.01671982 0.01640093 0.01614749 0.01605744 0.01577874
 0.01546399 0.01540312]
```

```
PCAs:
 [[ 0.08299431  0.01487481  0.02033137 ...  0.00439872  0.00753131
   0.03755342]
 [-0.0306382  -0.01152528 -0.00262131 ... -0.00117136  0.00305161
  -0.00548837]
 [-0.02992027  0.00318566  0.02026131 ...  0.00162629 -0.00235729
   0.00324852]
 ...
 [-0.06749171  0.01373169 -0.004468   ... -0.00024765 -0.0012616
   0.02281482]
 [ 0.01544325 -0.00614809 -0.00589183 ...  0.00810234  0.00241381
   0.00649564]
 [-0.06298108  0.00382704  0.01272912 ... -0.00309019  0.00090491
   0.02234385]]
Index(['diag_3_7', 'diag_3_7', 'diag_2_7', 'diag_3_1',
       'time_in_hospital_(0.333, 0.667]', 'time_in_hospital_(0.333, 0.667]',
       'discharge_disposition_id_6', 'age_7', 'age_5', 'age_6', 'age_6',
       'num_lab_procedures_(0.333, 0.667]', 'admission_type_id_2', 'diag_2_1',
       'diag_3_1', 'age_4', 'num_procedures_(0.333, 0.667]', 'diag_2_0',
       'diag_2_3', 'gender_0'],
      dtype='object')

Out[309]: (107645, 20)
```

Naïve Bayes After using PCA:

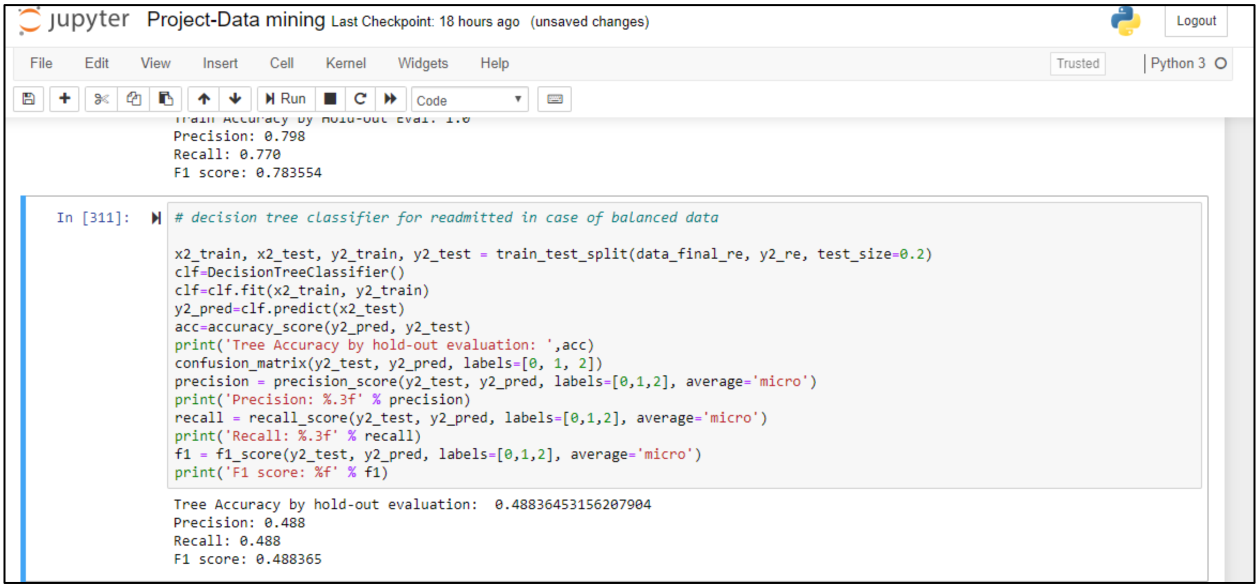

jupyter Project-Data mining Last Checkpoint: 18 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

In [310]:

```
# Naive Bayes after PCA for readmitted in case of balanced data

x2_train, x2_test, y2_train, y2_test = train_test_split(PCAs, y2_re, test_size=0.2)
clf = GaussianNB()
clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
print("Accuracy by Hold-out Eval:",accuracy_score(y2_pred,y2_test))
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)
```

```
Accuracy by Hold-out Eval: 0.5655627293418181
Precision: 0.566
Recall: 0.566
F1 score: 0.565563
```

Decision tree classifier without PCA:


jupyter Project-Data mining Last Checkpoint: 18 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

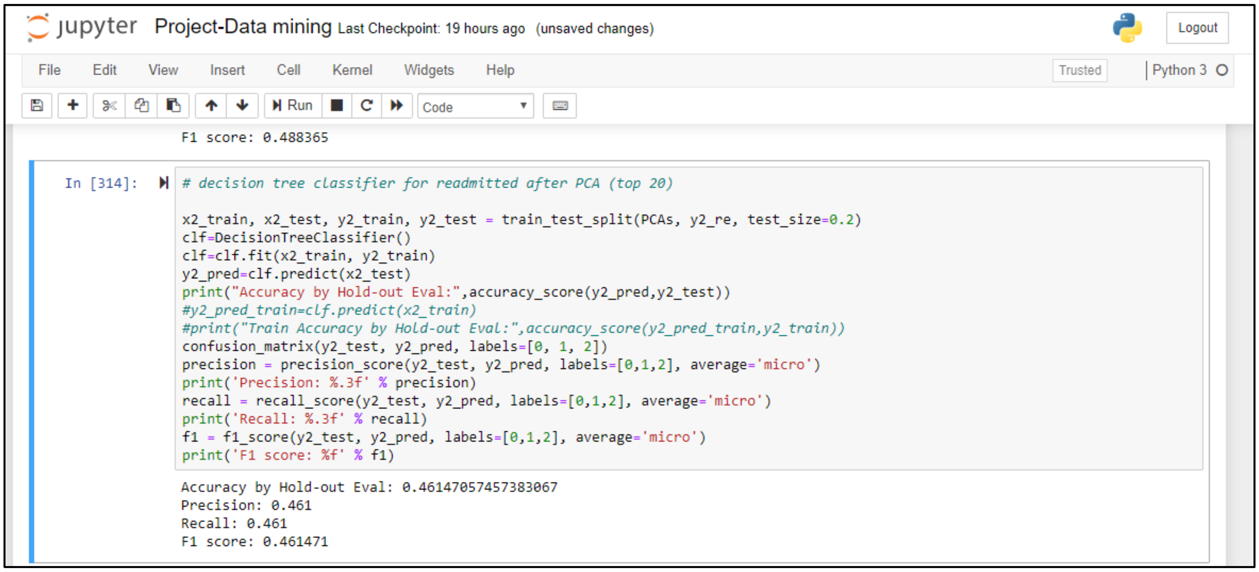
```
Train Accuracy by Hold-out Eval: 1.0
Precision: 0.798
Recall: 0.770
F1 score: 0.783554
```

In [311]:

```
# decision tree classifier for readmitted in case of balanced data

x2_train, x2_test, y2_train, y2_test = train_test_split(data_final_re, y2_re, test_size=0.2)
clf=DecisionTreeClassifier()
clf=clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)
```

```
Tree Accuracy by hold-out evaluation:  0.48836453156207904
Precision: 0.488
Recall: 0.488
F1 score: 0.488365
```

Decision tree after PCA

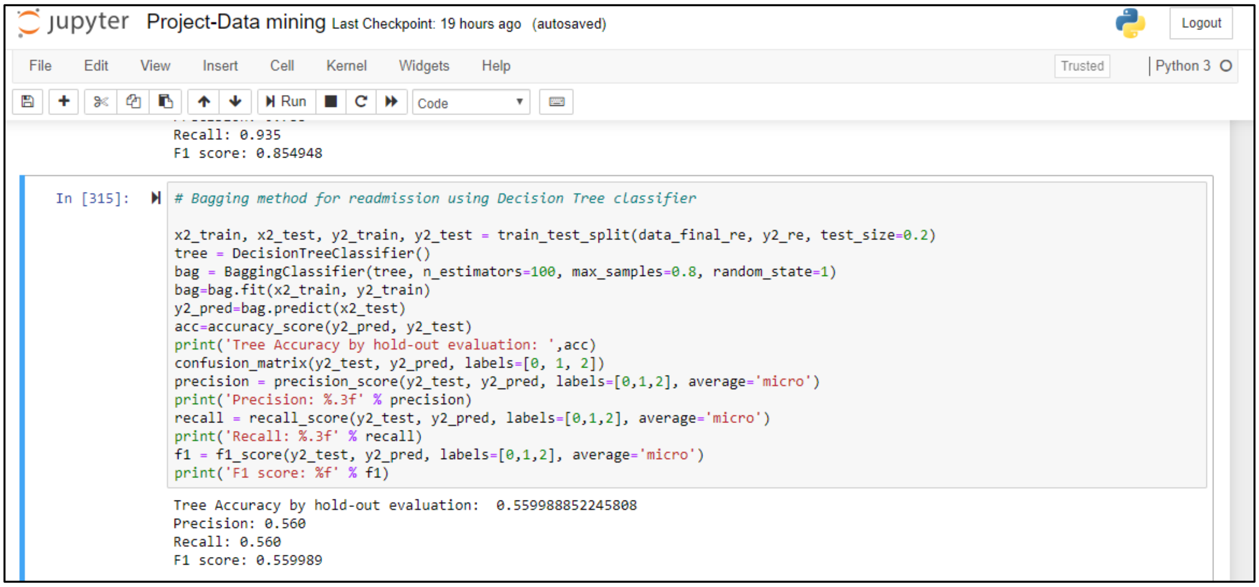


```
F1 score: 0.488365
```

```python
# decision tree classifier for readmitted after PCA (top 20)

x2_train, x2_test, y2_train, y2_test = train_test_split(PCAs, y2_re, test_size=0.2)
clf=DecisionTreeClassifier()
clf=clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
print("Accuracy by Hold-out Eval:",accuracy_score(y2_pred,y2_test))
#y2_pred_train=clf.predict(x2_train)
#print("Train Accuracy by Hold-out Eval:",accuracy_score(y2_pred_train,y2_train))
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)
```

```
Accuracy by Hold-out Eval: 0.46147057457383067
Precision: 0.461
Recall: 0.461
F1 score: 0.461471
```

Bagging method using decision tree classifier without PCA :

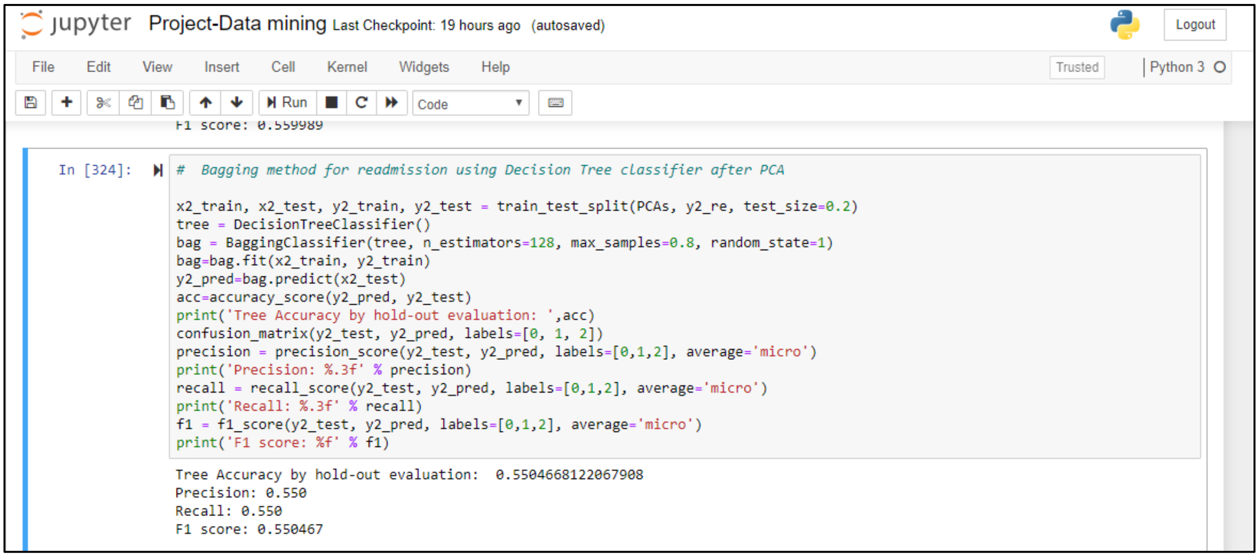


```
Recall: 0.935
F1 score: 0.854948
```

```python
# Bagging method for readmission using Decision Tree classifier

x2_train, x2_test, y2_train, y2_test = train_test_split(data_final_re, y2_re, test_size=0.2)
tree = DecisionTreeClassifier()
bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8, random_state=1)
bag=bag.fit(x2_train, y2_train)
y2_pred=bag.predict(x2_test)
acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)
```

```
Tree Accuracy by hold-out evaluation:  0.559988852245808
Precision: 0.560
Recall: 0.560
F1 score: 0.559989
```

Bagging method using decision tree classifier after PCA



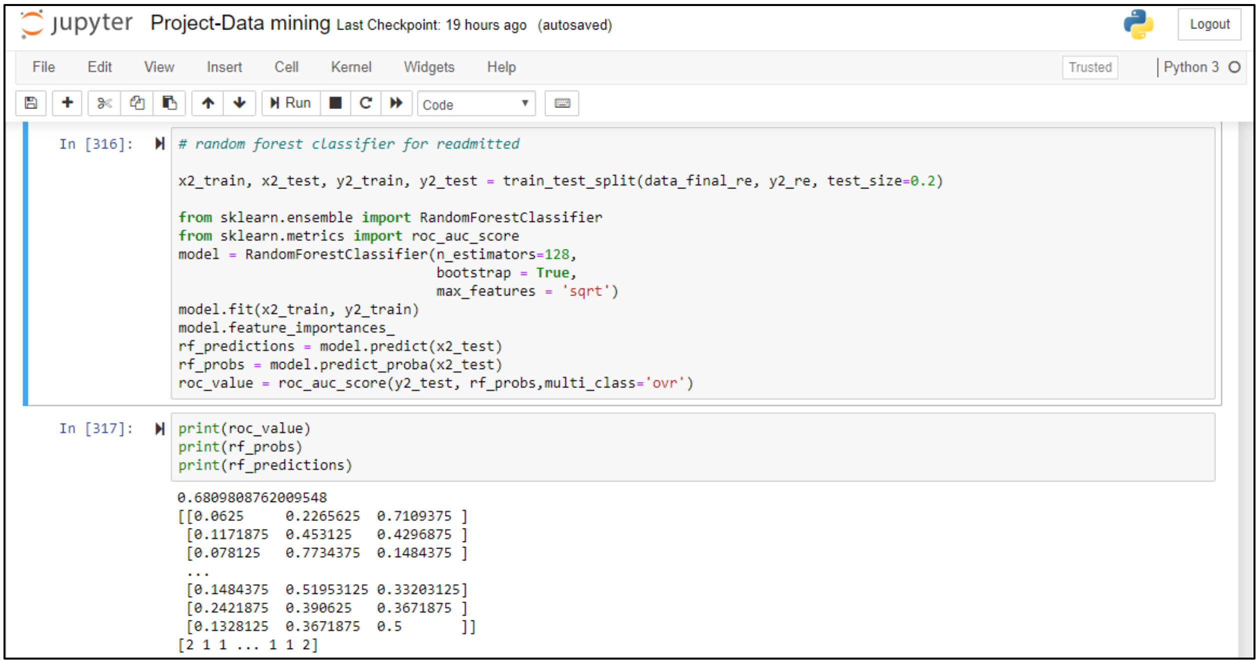
```
F1 score: 0.559989
```

In [324]:

```
#  Bagging method for readmission using Decision Tree classifier after PCA

x2_train, x2_test, y2_train, y2_test = train_test_split(PCAs, y2_re, test_size=0.2)
tree = DecisionTreeClassifier()
bag = BaggingClassifier(tree, n_estimators=128, max_samples=0.8, random_state=1)
bag=bag.fit(x2_train, y2_train)
y2_pred=bag.predict(x2_test)
acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)
```

```
Tree Accuracy by hold-out evaluation:  0.5504668122067908
Precision: 0.550
Recall: 0.550
F1 score: 0.550467
```

Random forest without applying PCA :



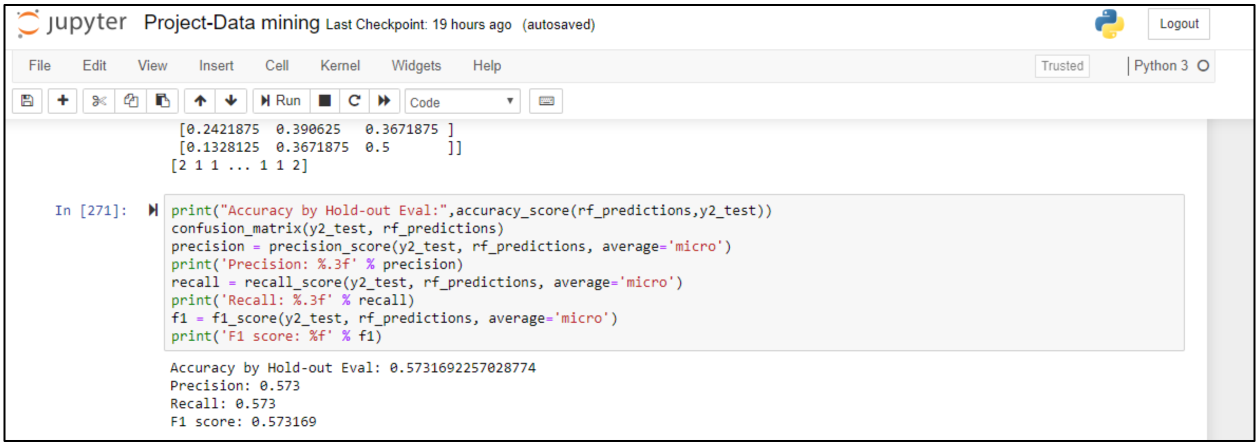
In [316]:

```
# random forest classifier for readmitted

x2_train, x2_test, y2_train, y2_test = train_test_split(data_final_re, y2_re, test_size=0.2)

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
model = RandomForestClassifier(n_estimators=128,
                               bootstrap = True,
                               max_features = 'sqrt')
model.fit(x2_train, y2_train)
model.feature_importances_
rf_predictions = model.predict(x2_test)
rf_probs = model.predict_proba(x2_test)
roc_value = roc_auc_score(y2_test, rf_probs,multi_class='ovr')
```

In [317]:

```
print(roc_value)
print(rf_probs)
print(rf_predictions)
```

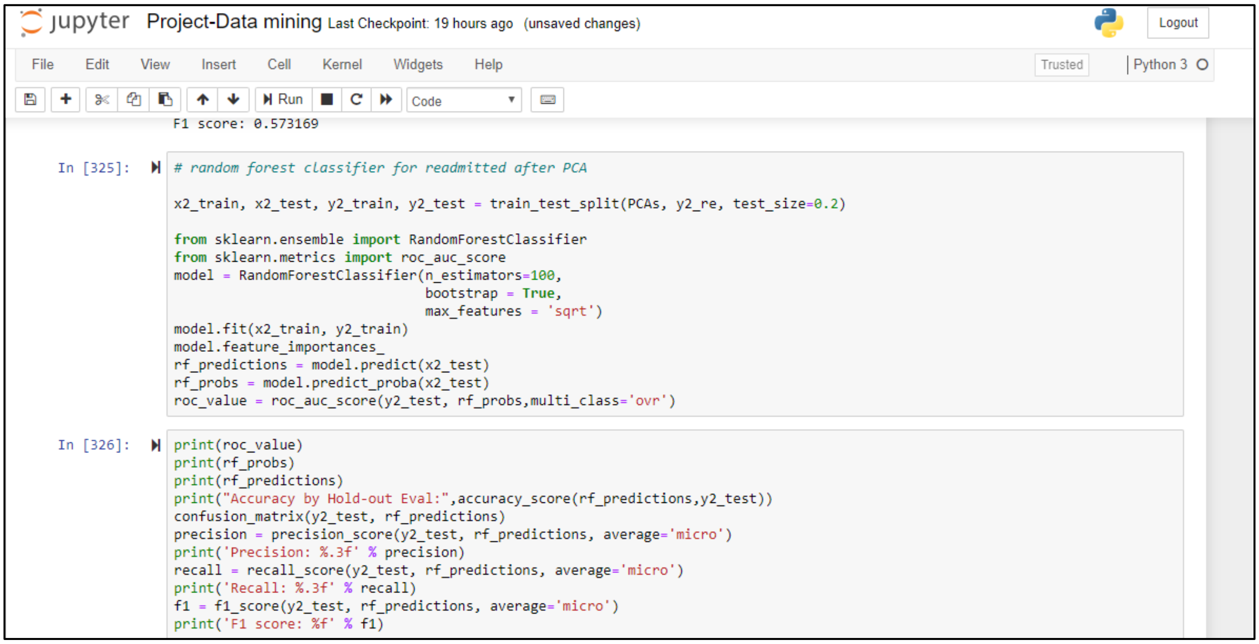
```
0.6809808762009548
[[0.0625     0.2265625  0.7109375 ]
 [0.1171875  0.453125   0.4296875 ]
 [0.078125   0.7734375  0.1484375 ]
 ...
 [0.1484375  0.51953125 0.33203125]
 [0.2421875  0.390625   0.3671875 ]
 [0.1328125  0.3671875  0.5       ]]
[2 1 1 ... 1 1 2]
```



```
 [0.2421875  0.390625   0.3671875 ]
 [0.1328125  0.3671875  0.5       ]]
[2 1 1 ... 1 1 2]
```

```
In [271]:  print("Accuracy by Hold-out Eval:",accuracy_score(rf_predictions,y2_test))
           confusion_matrix(y2_test, rf_predictions)
           precision = precision_score(y2_test, rf_predictions, average='micro')
           print('Precision: %.3f' % precision)
           recall = recall_score(y2_test, rf_predictions, average='micro')
           print('Recall: %.3f' % recall)
           f1 = f1_score(y2_test, rf_predictions, average='micro')
           print('F1 score: %f' % f1)

           Accuracy by Hold-out Eval: 0.5731692257028774
           Precision: 0.573
           Recall: 0.573
           F1 score: 0.573169
```

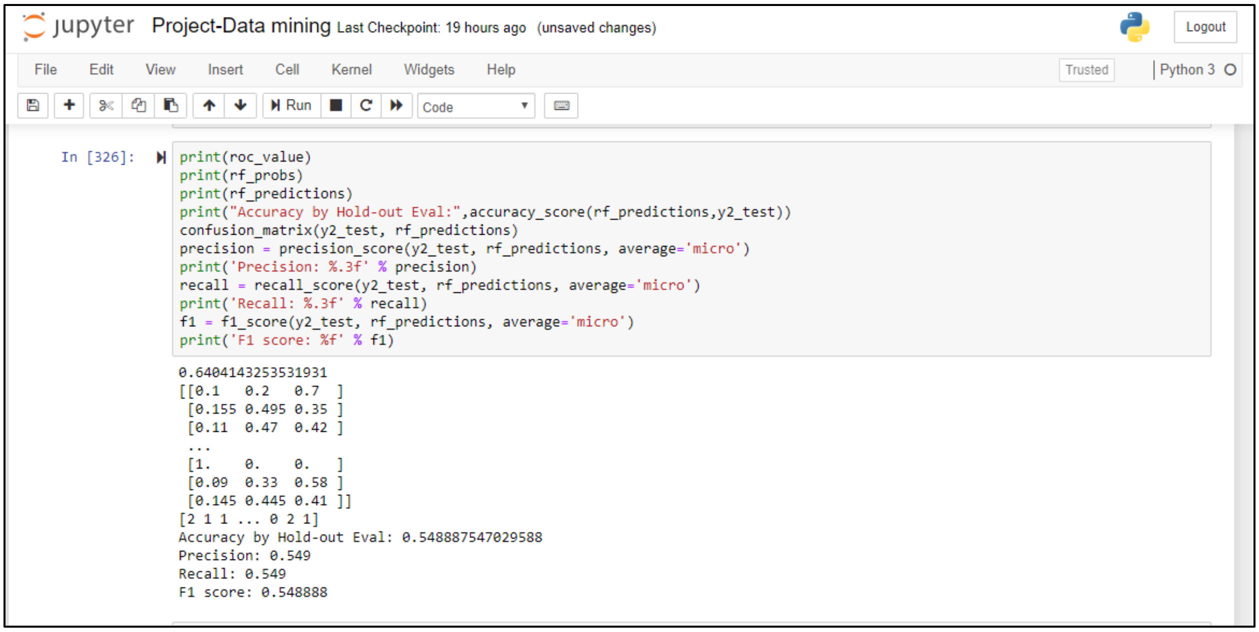
Random forest after PCA



```
F1 score: 0.573169
```

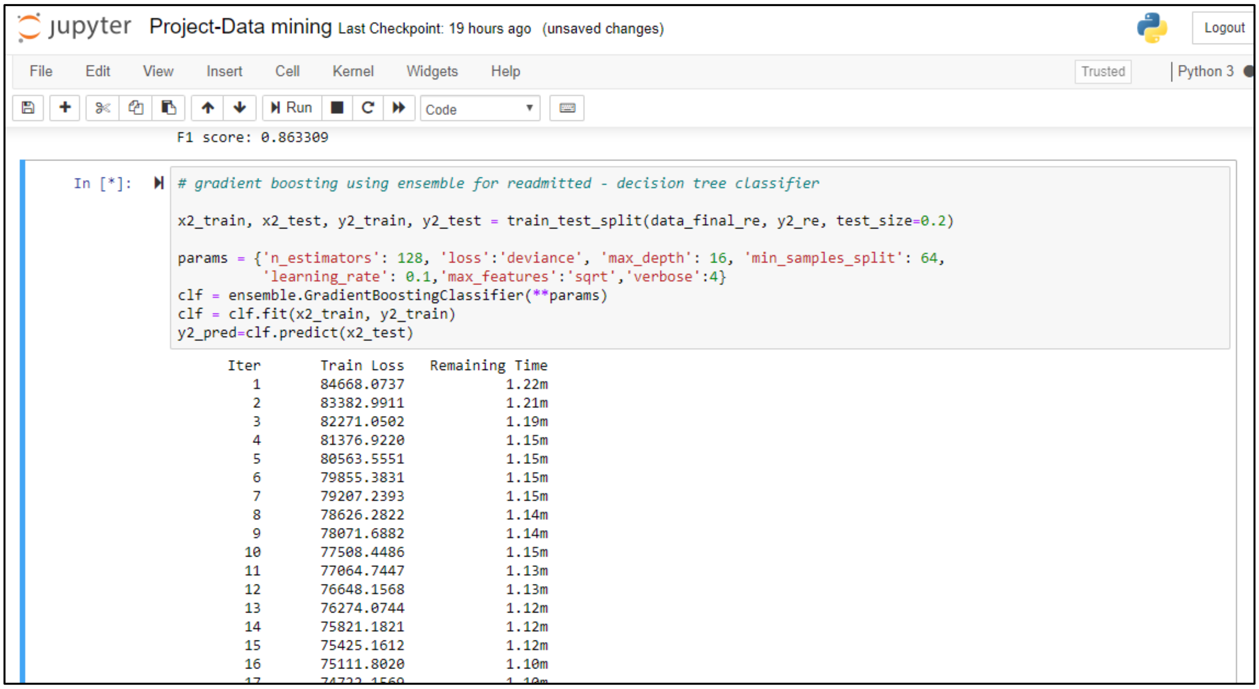
```
In [325]:  # random forest classifier for readmitted after PCA

           x2_train, x2_test, y2_train, y2_test = train_test_split(PCAs, y2_re, test_size=0.2)

           from sklearn.ensemble import RandomForestClassifier
           from sklearn.metrics import roc_auc_score
           model = RandomForestClassifier(n_estimators=100,
                                          bootstrap = True,
                                          max_features = 'sqrt')
           model.fit(x2_train, y2_train)
           model.feature_importances_
           rf_predictions = model.predict(x2_test)
           rf_probs = model.predict_proba(x2_test)
           roc_value = roc_auc_score(y2_test, rf_probs,multi_class='ovr')
```

```
In [326]:  print(roc_value)
           print(rf_probs)
           print(rf_predictions)
           print("Accuracy by Hold-out Eval:",accuracy_score(rf_predictions,y2_test))
           confusion_matrix(y2_test, rf_predictions)
           precision = precision_score(y2_test, rf_predictions, average='micro')
           print('Precision: %.3f' % precision)
           recall = recall_score(y2_test, rf_predictions, average='micro')
           print('Recall: %.3f' % recall)
           f1 = f1_score(y2_test, rf_predictions, average='micro')
           print('F1 score: %f' % f1)
```

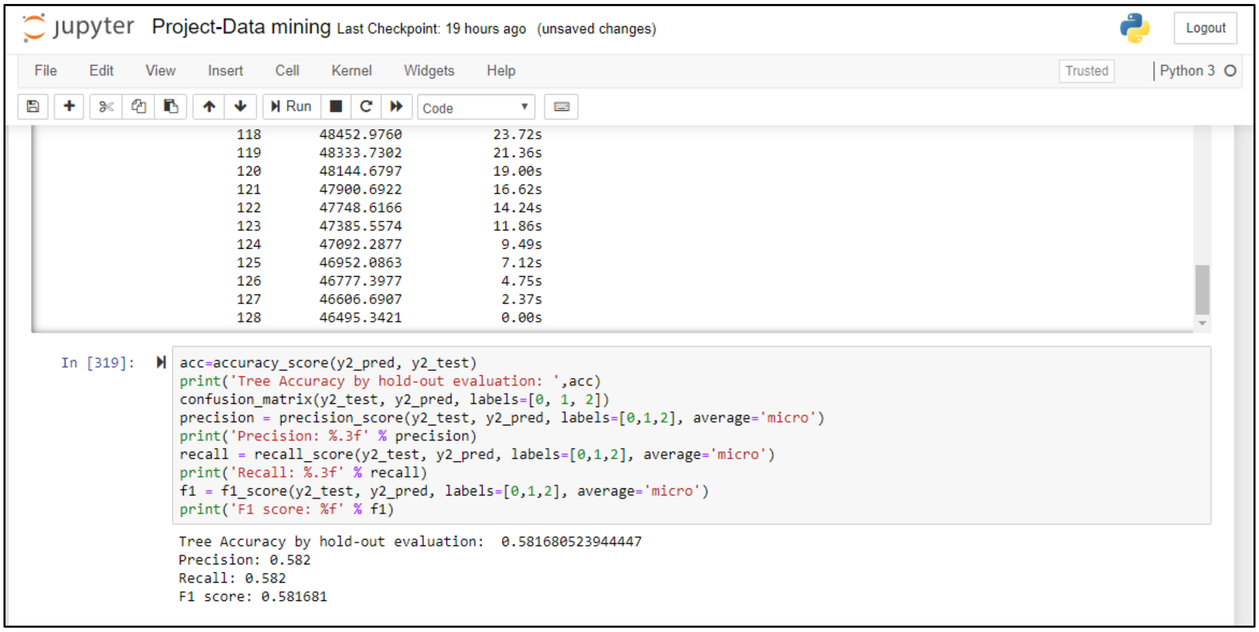

jupyter Project-Data mining Last Checkpoint: 19 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [326]:  print(roc_value)
           print(rf_probs)
           print(rf_predictions)
           print("Accuracy by Hold-out Eval:",accuracy_score(rf_predictions,y2_test))
           confusion_matrix(y2_test, rf_predictions)
           precision = precision_score(y2_test, rf_predictions, average='micro')
           print('Precision: %.3f' % precision)
           recall = recall_score(y2_test, rf_predictions, average='micro')
           print('Recall: %.3f' % recall)
           f1 = f1_score(y2_test, rf_predictions, average='micro')
           print('F1 score: %f' % f1)

           0.6404143253531931
           [[0.1   0.2   0.7  ]
            [0.155 0.495 0.35 ]
            [0.11  0.47  0.42 ]
            ...
            [1.    0.    0.   ]
            [0.09  0.33  0.58 ]
            [0.145 0.445 0.41 ]]
           [2 1 1 ... 0 2 1]
           Accuracy by Hold-out Eval: 0.548887547029588
           Precision: 0.549
           Recall: 0.549
           F1 score: 0.548888
```

Gradient boosting-ensemble method using decision tree without applying PCA :


jupyter Project-Data mining Last Checkpoint: 19 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
           F1 score: 0.863309

In [*]:    # gradient boosting using ensemble for readmitted - decision tree classifier

           x2_train, x2_test, y2_train, y2_test = train_test_split(data_final_re, y2_re, test_size=0.2)

           params = {'n_estimators': 128, 'loss':'deviance', 'max_depth': 16, 'min_samples_split': 64,
                     'learning_rate': 0.1,'max_features':'sqrt','verbose':4}
           clf = ensemble.GradientBoostingClassifier(**params)
           clf = clf.fit(x2_train, y2_train)
           y2_pred=clf.predict(x2_test)

                 Iter       Train Loss   Remaining Time
                    1       84668.0737            1.22m
                    2       83382.9911            1.21m
                    3       82271.0502            1.19m
                    4       81376.9220            1.15m
                    5       80563.5551            1.15m
                    6       79855.3831            1.15m
                    7       79207.2393            1.15m
                    8       78626.2822            1.14m
                    9       78071.6882            1.14m
                   10       77508.4486            1.15m
                   11       77064.7447            1.13m
                   12       76648.1568            1.13m
                   13       76274.0744            1.12m
                   14       75821.1821            1.12m
                   15       75425.1612            1.12m
                   16       75111.8020            1.10m
                   17       74722.1560            1.10m
```



jupyter Project-Data mining Last Checkpoint: 19 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

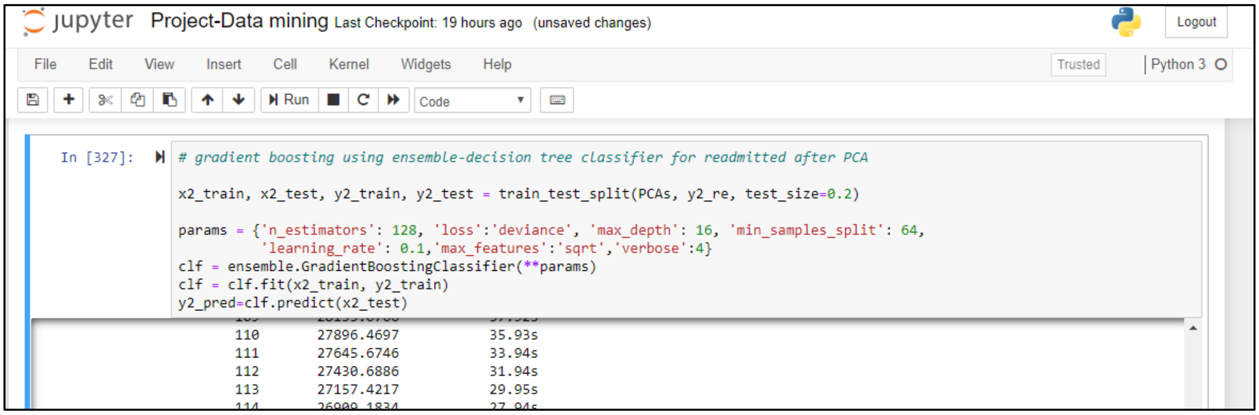
```
        118         48452.9760          23.72s
        119         48333.7302          21.36s
        120         48144.6797          19.00s
        121         47900.6922          16.62s
        122         47748.6166          14.24s
        123         47385.5574          11.86s
        124         47092.2877           9.49s
        125         46952.0863           7.12s
        126         46777.3977           4.75s
        127         46606.6907           2.37s
        128         46495.3421           0.00s
```

In [319]:

```
acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)
```

```
Tree Accuracy by hold-out evaluation:  0.581680523944447
Precision: 0.582
Recall: 0.582
F1 score: 0.581681
```

Gradient boosting ensemble method using decision tree after PCA

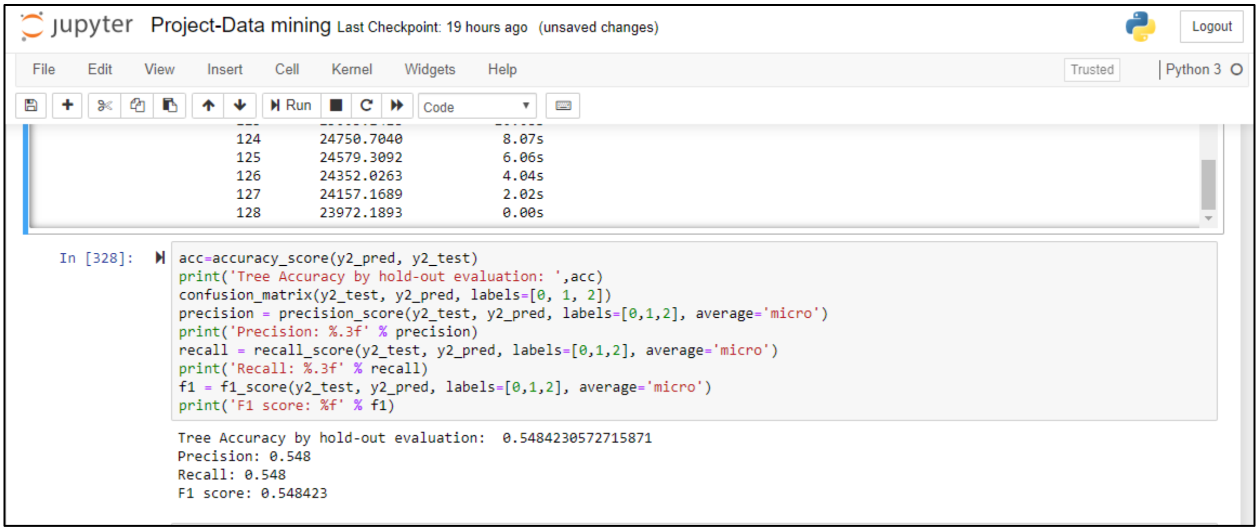


jupyter Project-Data mining Last Checkpoint: 19 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

In [327]:

```
# gradient boosting using ensemble-decision tree classifier for readmitted after PCA

x2_train, x2_test, y2_train, y2_test = train_test_split(PCAs, y2_re, test_size=0.2)

params = {'n_estimators': 128, 'loss':'deviance', 'max_depth': 16, 'min_samples_split': 64,
          'learning_rate': 0.1,'max_features':'sqrt','verbose':4}
clf = ensemble.GradientBoostingClassifier(**params)
clf = clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
```

```
        110         27896.4697          35.93s
        111         27645.6746          33.94s
        112         27430.6886          31.94s
        113         27157.4217          29.95s
        114         26909.1834          27.94s
```


jupyter Project-Data mining Last Checkpoint: 19 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

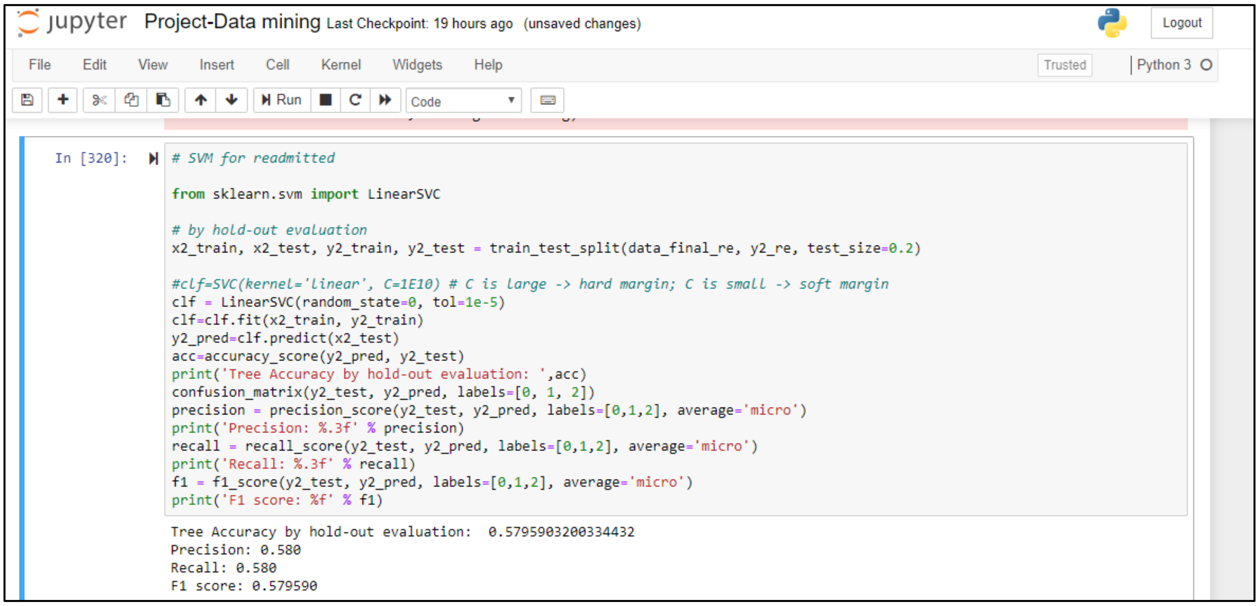
```
        124        24750.7040        8.07s
        125        24579.3092        6.06s
        126        24352.0263        4.04s
        127        24157.1689        2.02s
        128        23972.1893        0.00s
```

In [328]:

```
acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)
```

```
Tree Accuracy by hold-out evaluation:  0.5484230572715871
Precision: 0.548
Recall: 0.548
F1 score: 0.548423
```

SVM (Support Vector Machines) without applying PCA :

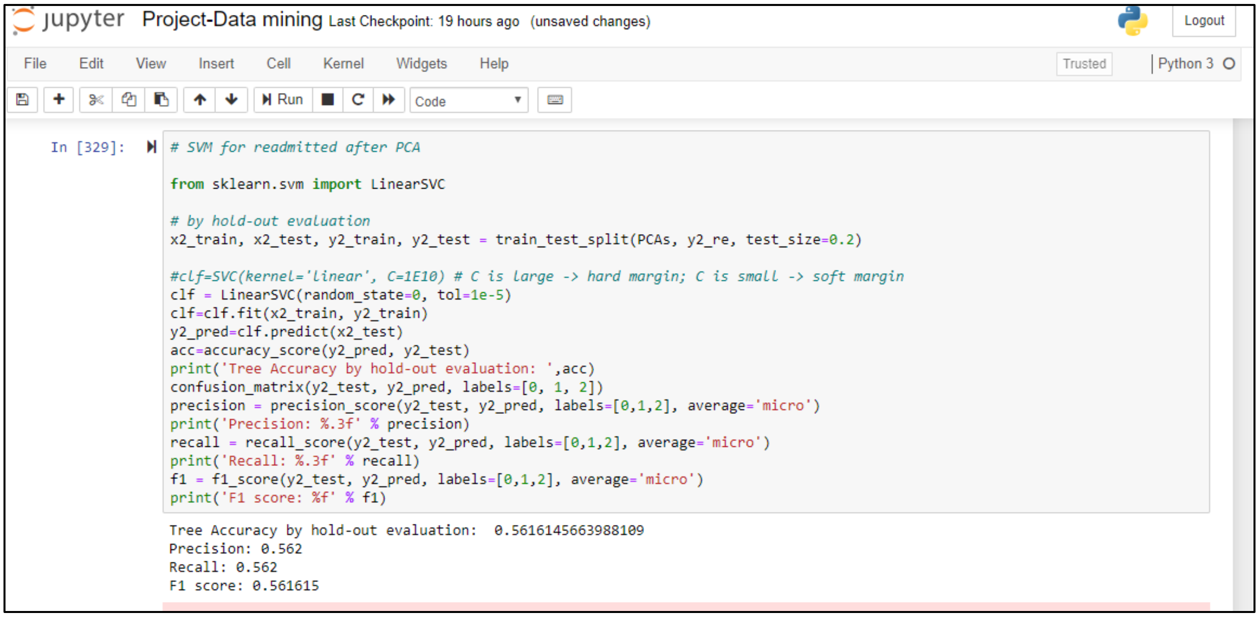

jupyter Project-Data mining Last Checkpoint: 19 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

In [320]:

```
# SVM for readmitted

from sklearn.svm import LinearSVC

# by hold-out evaluation
x2_train, x2_test, y2_train, y2_test = train_test_split(data_final_re, y2_re, test_size=0.2)

#clf=SVC(kernel='linear', C=1E10) # C is large -> hard margin; C is small -> soft margin
clf = LinearSVC(random_state=0, tol=1e-5)
clf=clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)
```

```
Tree Accuracy by hold-out evaluation:  0.5795903200334432
Precision: 0.580
Recall: 0.580
F1 score: 0.579590
```

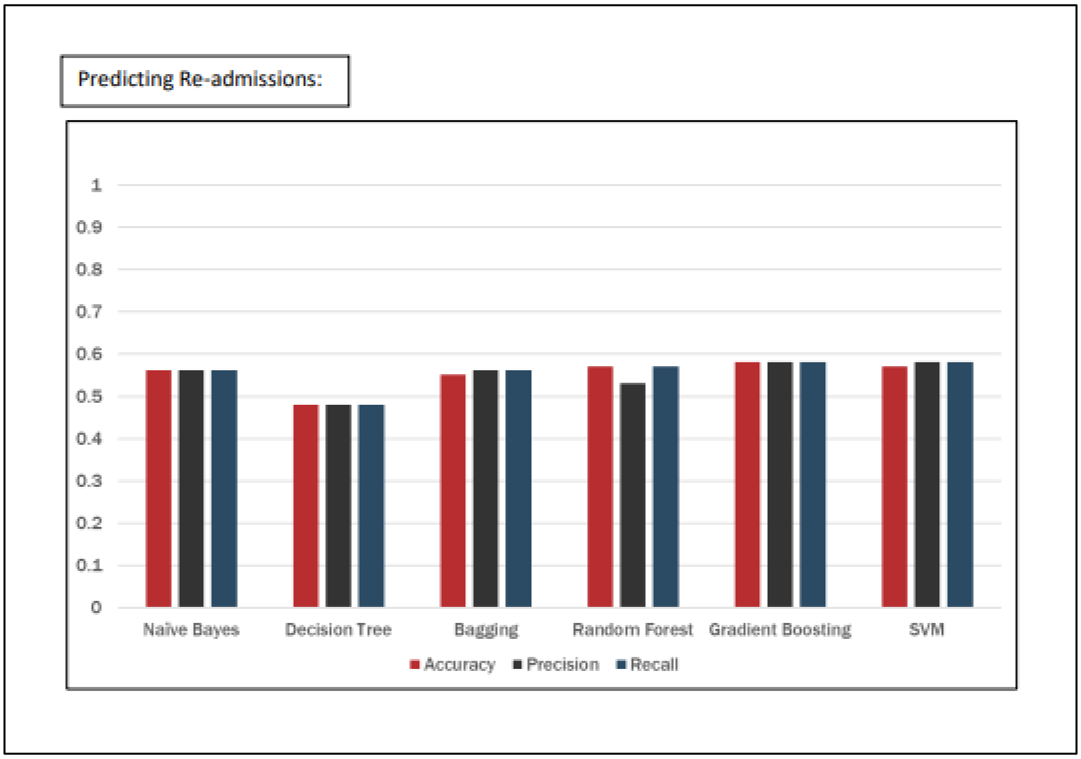
SVM (Support Vector Machines) after PCA


jupyter Project-Data mining Last Checkpoint: 19 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [329]:

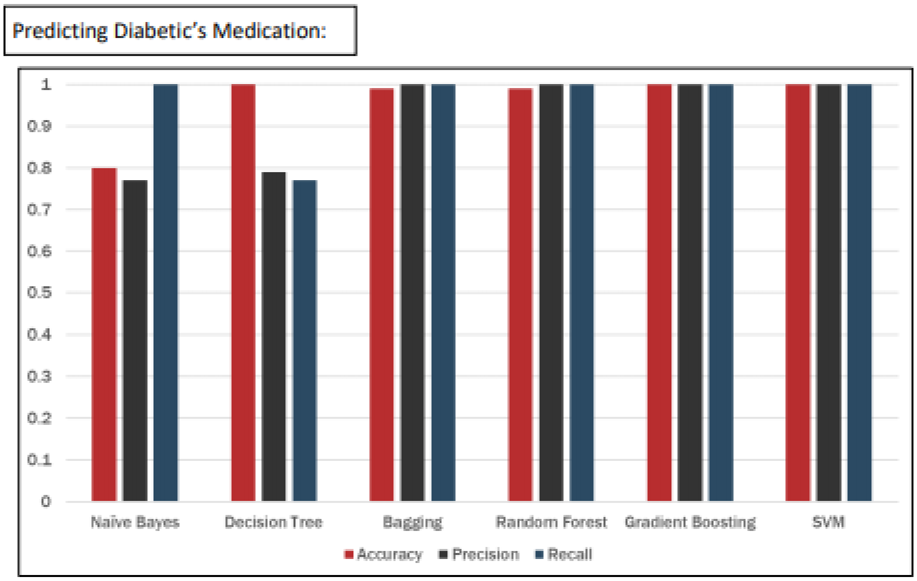
```
# SVM for readmitted after PCA

from sklearn.svm import LinearSVC

# by hold-out evaluation
x2_train, x2_test, y2_train, y2_test = train_test_split(PCAs, y2_re, test_size=0.2)

#clf=SVC(kernel='linear', C=1E10) # C is large -> hard margin; C is small -> soft margin
clf = LinearSVC(random_state=0, tol=1e-5)
clf=clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)
```

```
Tree Accuracy by hold-out evaluation:  0.5616145663988109
Precision: 0.562
Recall: 0.562
F1 score: 0.561615
```

Graph comparison between different methods used based on accuracy, precision and recall while predicting Re-admissions :



Graph comparison between different methods used based on accuracy, precision and recall while predicting Diabetic's Medication :


Predicting Diabetic's Medication:
Naïve Bayes
Decision Tree
Bagging
Random Forest
Gradient Boosting
SVM
Accuracy
Precision
Recall