In [1]:
```python
#importing different in-built python libraries
import numpy as np     #used for mathematical programming
import pandas as pd    #data restructuring
import matplotlib.pyplot as plt  #data visualization
import seaborn as sns
```

In [2]:
```python
application_record = pd.read_csv('application_record.csv')    #storing the given data in var
iables
credit_record = pd.read_csv('credit_record.csv')
```

In [3]:
```python
application_record.head()   #displaying data
```

Out[3]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_T
0	5008804	M	Y	Y	0	427500.0	Wor
1	5008805	M	Y	Y	0	427500.0	Wor
2	5008806	M	Y	Y	0	112500.0	Wor
3	5008808	F	N	Y	0	270000.0	Commercial asso
4	5008809	F	N	Y	0	270000.0	Commercial asso

In [4]:
```python
application_record.shape  #this displays the (number of rows in our dataset, number of colum
n in dataset)
```

Out[4]: (438557, 18)

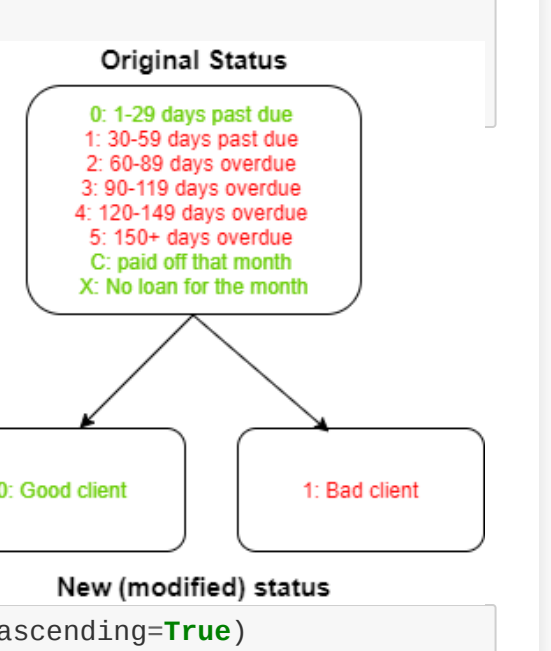
In [5]:
```python
application_record['ID'].nunique()  #the number of unique IDs
```

Out[5]: 438510

In [6]:
```python
credit_record.head()
#       0: 1-29 days past due
#1: 30-59 days past due
#2: 60-89 days overdue
#3: 90-119 days overdue
#4: 120-149 days overdue
#5: Overdue or bad debts, write-offs for more than 150 days
#C: paid off that month X: No loan for the month
```

Out[6]:

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	X
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	C



In [7]:
```python
credit_record=credit_record.sort_values(['ID','MONTHS_BALANCE'],ascending=True)
credit_record['STATUS'].replace({'C': 0, 'X' : 0}, inplace=True)
credit_record['STATUS'] = credit_record['STATUS'].astype('int')
credit_record['STATUS'] = credit_record['STATUS'].apply(lambda x:1 if x >= 1 else 0)
```

In [8]:
```python
credit_record.head()
```

Out[8]:

	ID	MONTHS_BALANCE	STATUS
3	5001711	-3	0
2	5001711	-2	0
1	5001711	-1	0
0	5001711	0	0
22	5001712	-18	0

grouping our data by ID, and considering only the maximum value of the Status column for each ID

In [9]:
```python
creditrecord=credit_record.loc[credit_record.groupby(['ID'])['STATUS'].idxmax()]
creditrecord.head()
#grouping our data by ID, and considering only the maximum value of the Status column for ea
ch ID
```

Out[9]:

	ID	MONTHS_BALANCE	STATUS
3	5001711	-3	0
22	5001712	-18	0
44	5001713	-21	0
59	5001714	-14	0
119	5001715	-59	0

In [10]:
```python
creditrecord.shape  #(number of rows, number of columns)
```

Out[10]: (45985, 3)

In [11]:
```python
application_record.info()
#There are many null values in occupation_type so we drop it
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438557 entries, 0 to 438556
Data columns (total 18 columns):
ID                     438557 non-null int64
CODE_GENDER            438557 non-null object
FLAG_OWN_CAR           438557 non-null object
FLAG_OWN_REALTY        438557 non-null object
CNT_CHILDREN           438557 non-null int64
AMT_INCOME_TOTAL       438557 non-null float64
NAME_INCOME_TYPE       438557 non-null object
NAME_EDUCATION_TYPE    438557 non-null object
NAME_FAMILY_STATUS     438557 non-null object
NAME_HOUSING_TYPE      438557 non-null object
DAYS_BIRTH             438557 non-null int64
DAYS_EMPLOYED          438557 non-null int64
FLAG_MOBIL             438557 non-null int64
FLAG_WORK_PHONE        438557 non-null int64
FLAG_PHONE             438557 non-null int64
FLAG_EMAIL             438557 non-null int64
OCCUPATION_TYPE        304354 non-null object
CNT_FAM_MEMBERS        438557 non-null float64
dtypes: float64(2), int64(8), object(8)
memory usage: 60.2+ MB
```

----------> High number of N/A values

Label encoder used to convert categorical data to numerical data

In [12]:
```python
application_record=application_record.drop_duplicates('ID')   #dropping duplicate IDs
application_record=application_record.drop('OCCUPATION_TYPE',axis=1)   #drop because of NA V
alues
from sklearn.preprocessing import LabelEncoder
#Label encoder used to convert categorical data to numerical data
le=LabelEncoder()
application_record.loc[:,['CODE_GENDER','FLAG_OWN_CAR','FLAG_OWN_REALTY','NAME_INCOME_TYPE',
'NAME_FAMILY_STATUS','NAME_HOUSING_TYPE']] = \
application_record.loc[:,['CODE_GENDER','FLAG_OWN_CAR','FLAG_OWN_REALTY','NAME_INCOME_TYPE',
'NAME_FAMILY_STATUS','NAME_HOUSING_TYPE']].apply(le.fit_transform)
application_record.DAYS_BIRTH=application_record.DAYS_BIRTH*-1
application_record.DAYS_EMPLOYED=application_record.DAYS_EMPLOYED*-1
application_record['NAME_EDUCATION_TYPE'].replace({'Lower secondary':0,'Secondary / secondar
y special':1,'Incomplete higher':2,'Higher education':3,'Academic degree':4},inplace=True)
application_record['NAME_EDUCATION_TYPE']=application_record['NAME_EDUCATION_TYPE'].astype(
'int')
application_record=application_record.drop('FLAG_MOBIL',axis=1)
application_record['DAYS_EMPLOYED'] = application_record['DAYS_EMPLOYED'].apply(lambda x:0 i
f x<0 else x)
application_record.head()
```

Out[12]:

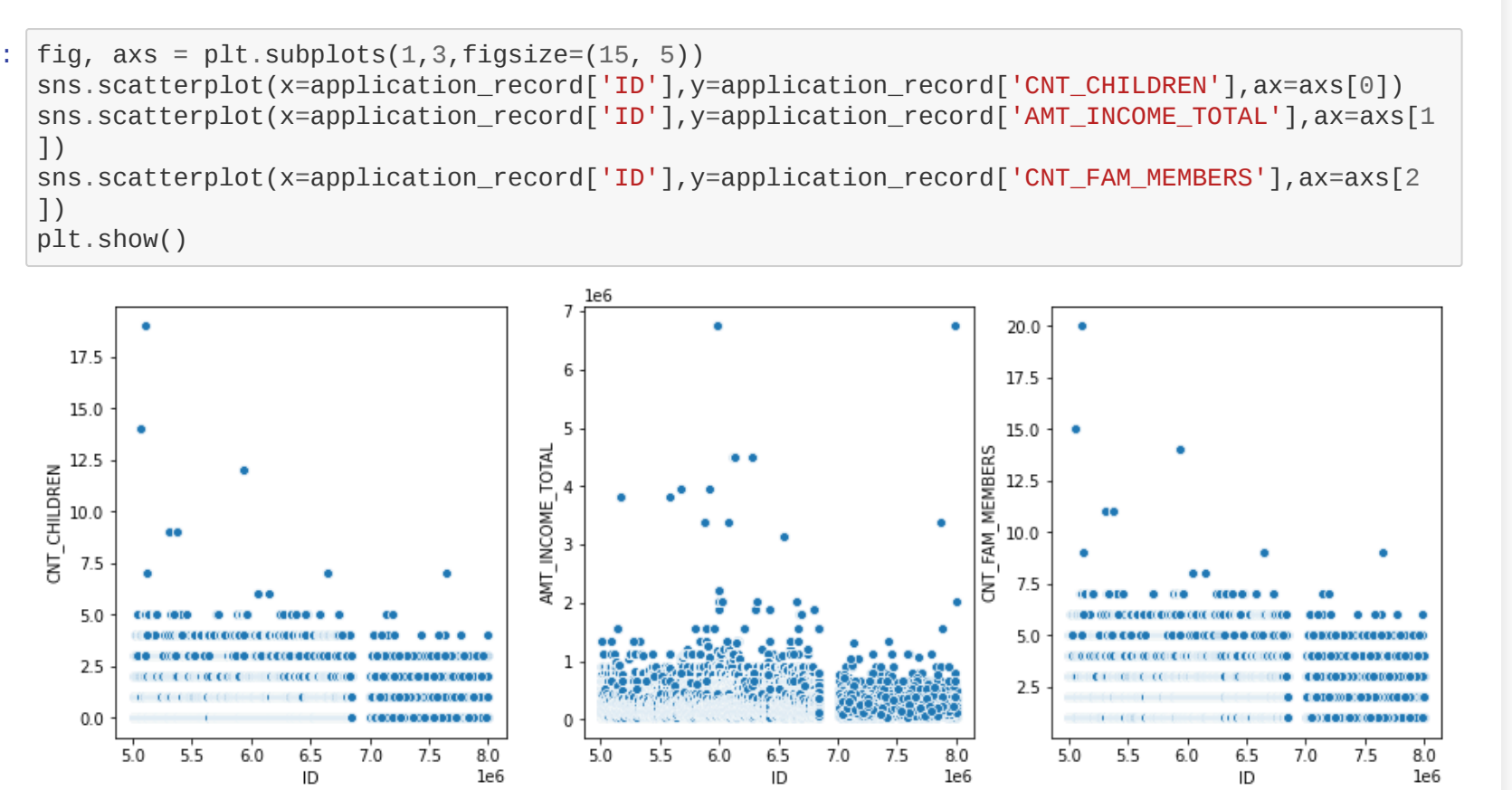
	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_T
0	5008804	1	1	1	0	427500.0	
1	5008805	1	1	1	0	427500.0	
2	5008806	1	1	1	0	112500.0	
3	5008808	0	0	1	0	270000.0	
4	5008809	0	0	1	0	270000.0	

In [13]:
```python
application_record.shape
```

Out[13]: (438510, 16)

Outlier detection

In [14]:
```python
fig, axs = plt.subplots(1,3,figsize=(15, 5))
sns.scatterplot(x=application_record['ID'],y=application_record['CNT_CHILDREN'],ax=axs[0])
sns.scatterplot(x=application_record['ID'],y=application_record['AMT_INCOME_TOTAL'],ax=axs[1
])
sns.scatterplot(x=application_record['ID'],y=application_record['CNT_FAM_MEMBERS'],ax=axs[2
])
plt.show()
```



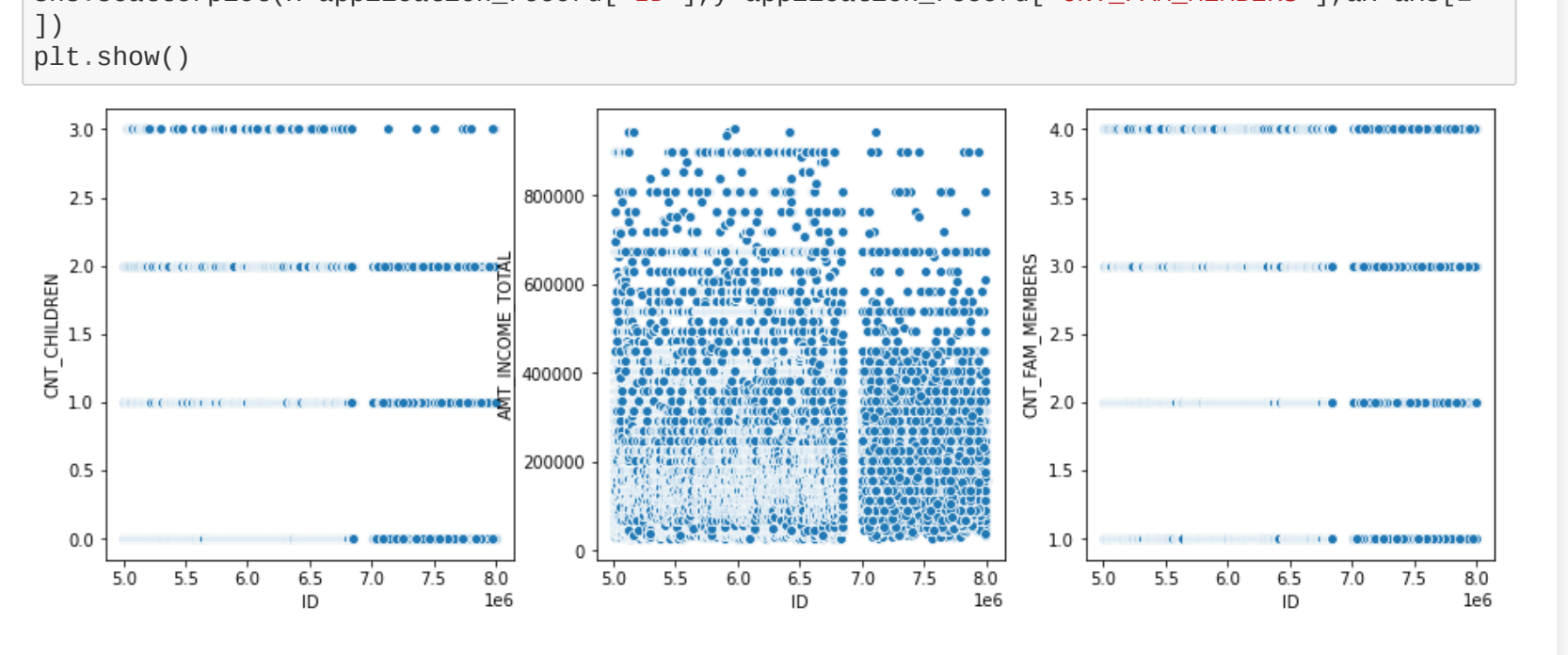
Outlier Removal

In [15]:
```python
# We remove the top 0.1% values as we consider them to be outliers for following 3 columns
upper_limit=application_record['CNT_CHILDREN'].quantile(0.999)
lower_limit=application_record['CNT_CHILDREN'].quantile(0.000)
application_record = application_record[(application_record['CNT_CHILDREN']>=lower_limit) &
(application_record['CNT_CHILDREN']<upper_limit)]
```

In [16]:
```python
upper_limit=application_record['AMT_INCOME_TOTAL'].quantile(0.999)
lower_limit=application_record['AMT_INCOME_TOTAL'].quantile(0.000)
application_record = application_record[(application_record['AMT_INCOME_TOTAL']>=lower_limit
) & (application_record['AMT_INCOME_TOTAL']<upper_limit)]
```

In [17]:
```python
upper_limit=application_record['CNT_FAM_MEMBERS'].quantile(0.999)
lower_limit=application_record['CNT_FAM_MEMBERS'].quantile(0.000)
application_record = application_record[(application_record['CNT_FAM_MEMBERS']>=lower_limit)
& (application_record['CNT_FAM_MEMBERS']<upper_limit)]
```

In [18]:
```python
fig, axs = plt.subplots(1,3,figsize=(15, 5))
sns.scatterplot(x=application_record['ID'],y=application_record['CNT_CHILDREN'],ax=axs[0])
sns.scatterplot(x=application_record['ID'],y=application_record['AMT_INCOME_TOTAL'],ax=axs[1
])
sns.scatterplot(x=application_record['ID'],y=application_record['CNT_FAM_MEMBERS'],ax=axs[2
])
plt.show()
```



In [19]:
```python
application_record.shape
```

Out[19]: (432377, 16)

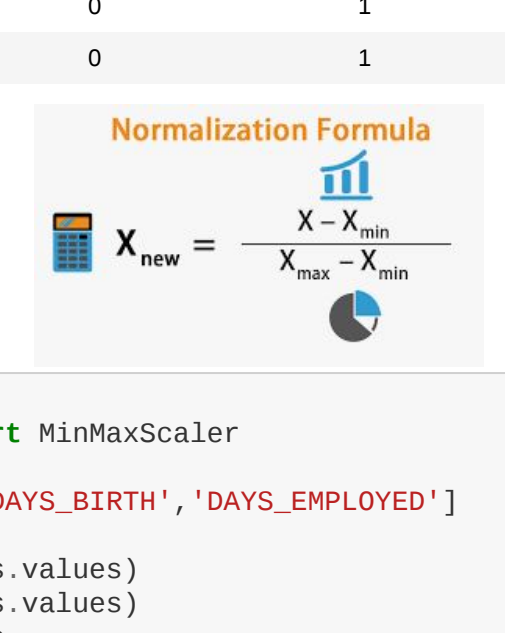
In [20]:
```python
new_dataset=pd.merge(application_record,creditrecord,on='ID')
new_dataset
```

Out[20]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOM
0	5008804	1	1	1	0	427500.0	
1	5008805	1	1	1	0	427500.0	
2	5008806	1	1	1	0	112500.0	
3	5008808	0	0	1	0	270000.0	
4	5008809	0	0	1	0	270000.0	
...	...	...	...	...	...	...	
35951	5149828	1	1	1	0	315000.0	
35952	5149834	0	0	1	0	157500.0	
35953	5149838	0	0	1	0	157500.0	
35954	5150049	0	0	1	0	283500.0	
35955	5150337	1	0	1	0	112500.0	

35956 rows × 18 columns

NORMALIZATION!



$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

In [21]:
```python
#Normalization
from sklearn.preprocessing import MinMaxScaler
norm_dataset=new_dataset.copy()
col_names=['AMT_INCOME_TOTAL','DAYS_BIRTH','DAYS_EMPLOYED']
features=norm_dataset[col_names]
norm=MinMaxScaler().fit(features.values)
features=norm.transform(features.values)
norm_dataset[col_names]=features
norm_dataset=norm_dataset.drop(['MONTHS_BALANCE'],axis=1)
norm_dataset
```

Out[21]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOM
0	5008804	1	1	1	0	0.436275	
1	5008805	1	1	1	0	0.436275	
2	5008806	1	1	1	0	0.093137	
3	5008808	0	0	1	0	0.264706	
4	5008809	0	0	1	0	0.264706	
...	...	...	...	...	...	...	
35951	5149828	1	1	1	0	0.313725	
35952	5149834	0	0	1	0	0.142157	
35953	5149838	0	0	1	0	0.142157	
35954	5150049	0	0	1	0	0.279412	
35955	5150337	1	0	1	0	0.093137	

35956 rows × 17 columns

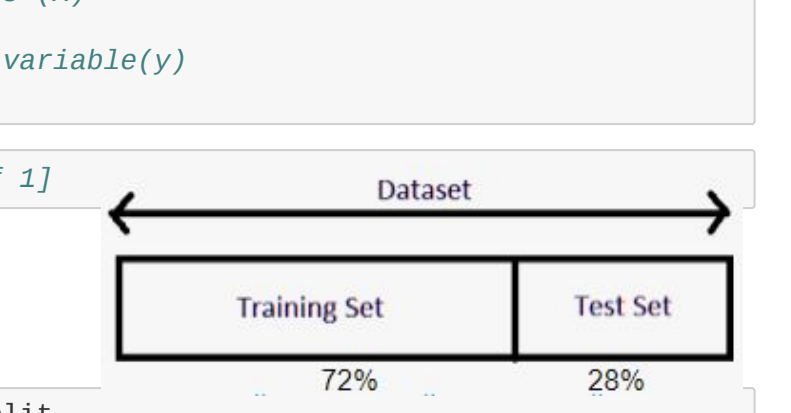
Values converted between 0 and 1

In [22]:
```python
#dropping ID and Status column to get our features (X)
X_norm=norm_dataset.drop(['ID','STATUS'],axis=1)
#Selecting the Status column to get our response variable(y)
y=norm_dataset['STATUS']
```

In [23]:
```python
np.bincount(y)  #displays [number of 0, number of 1]
```

Out[23]: array([31730, 4226], dtype=int64)

High imbalance in the data



In [24]:
```python
from sklearn.model_selection import train_test_split
X_train_norm, X_test_norm, y_train, y_test = train_test_split(X_norm, y, test_size=0.28, ran
dom_state=10)
```

SMOTE oversampling

In [25]:
```python
#SMOTE
from imblearn.over_sampling import SMOTE
smote=SMOTE()
X_train_smote,y_train_smote=smote.fit_resample(X_train_norm,y_train)
```

```
Using TensorFlow backend.
```

In [26]:
```python
np.bincount(y_train_smote) # y values after oversampling
#(number of zeros, number of one)
```

Out[26]: array([22855, 22855], dtype=int64)

Imbalance removed from the training data using oversampling

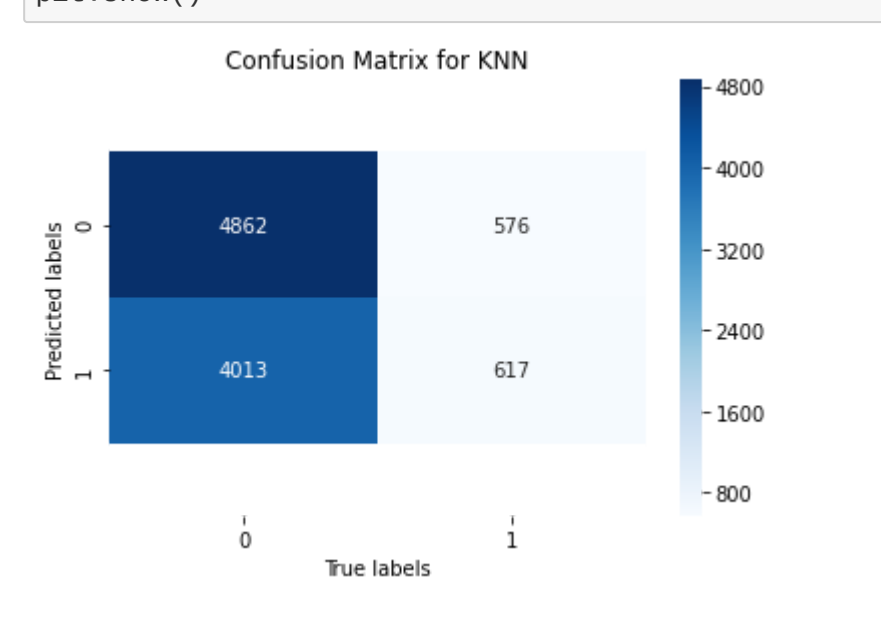
Logistic Regression

In [28]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
clf=LogisticRegression()
clf.fit(X_train_smote,y_train_smote)
predictions=clf.predict(X_test_norm)
print(accuracy_score(predictions,y_test))
```

```
C:\Users\Shravan Tawri\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: Futu
reWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence thi
s warning.
  FutureWarning)
```

0.5441994437822805

In [29]:
```python
cm=confusion_matrix(predictions,y_test)
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax,cmap='Blues',fmt='g')
ax.set_xlabel('True labels')
ax.set_ylabel('Predicted labels')
ax.set_title('Confusion Matrix for KNN')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show()
```

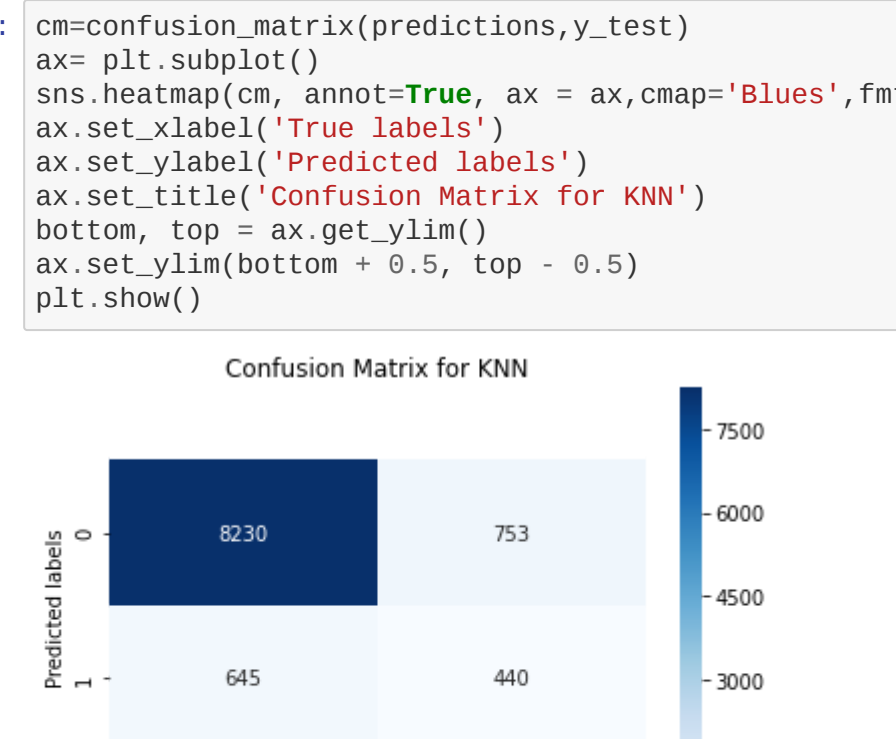


KNN Algorithm

In [30]:
```python
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=5,weights='distance')
neigh.fit(X_train_smote, y_train_smote)
predictions=neigh.predict(X_test_norm)
print(accuracy_score(predictions,y_test))
```

0.8611442193087009

In [31]:
```python
cm=confusion_matrix(predictions,y_test)
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax,cmap='Blues',fmt='g')
ax.set_xlabel('True labels')
ax.set_ylabel('Predicted labels')
ax.set_title('Confusion Matrix for KNN')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show()
```

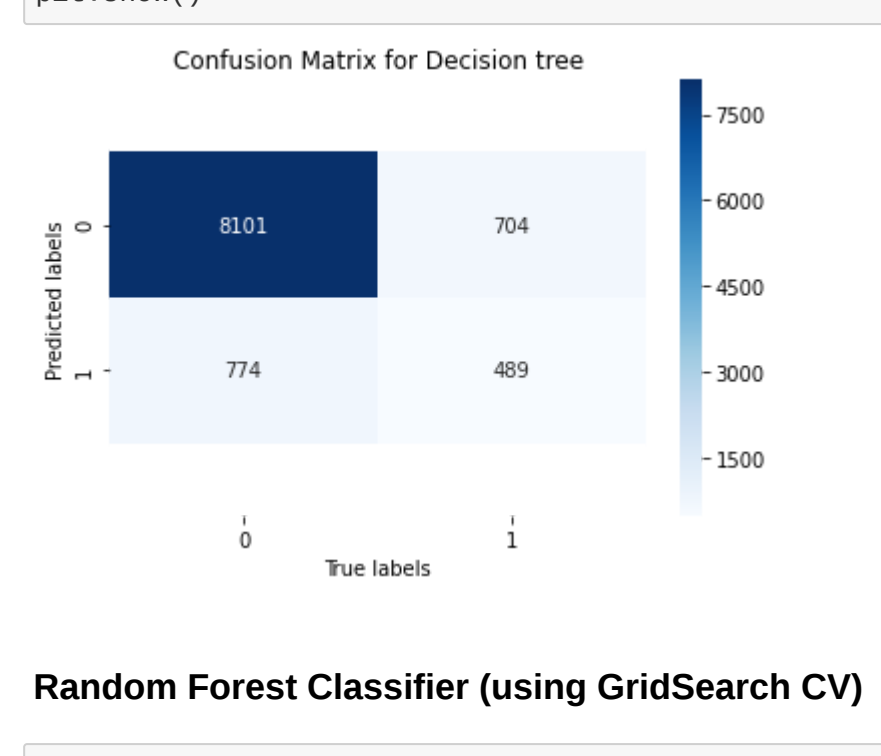


Decision Tree Classifier

In [32]:
```python
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X_train_smote,y_train_smote)
predictions=clf.predict(X_test_norm)
print(accuracy_score(predictions,y_test))
```

0.8531982518871672

In [33]:
```python
cm=confusion_matrix(predictions,y_test)
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax,cmap='Blues',fmt='g')
ax.set_xlabel('True labels')
ax.set_ylabel('Predicted labels')
ax.set_title('Confusion Matrix for Decision tree')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show()
```



Random Forest Classifier (using GridSearch CV)

In [34]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
clf = RandomForestClassifier()
parameters=[{'n_estimators':[5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100],'c
riterion':['gini', 'entropy'], 'class_weight':['balanced']}]
grid_search = GridSearchCV(estimator=clf,param_grid=parameters,scoring='f1',cv=7)
grid_search.fit(X_train_smote,y_train_smote)
grid_search.best_score_
```

Out[34]: 0.9057837734858251

In [35]:
```python
grid_search.best_params_
```

Out[35]: {'class_weight': 'balanced', 'criterion': 'entropy', 'n_estimators': 100}

In [36]:
```python
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
clf = RandomForestClassifier(n_estimators=100,class_weight='balanced',criterion='entropy')
clf.fit(X_train_smote,y_train_smote)
predictions=clf.predict(X_test_norm)
print(accuracy_score(predictions,y_test))
confusion_matrix(predictions,y_test)
```

0.8578665077473182

Out[36]:
```
array([[8121,  677],
       [ 754,  516]], dtype=int64)
```

In [37]:
```python
data = [['Logistic Regression', 0.517,0.13,54.41,0.207], ['KNN',0.368,0.405,86.11,0.385], [
'DecisionTree',0.41,0.387,85.31,0.398],['Random Forest',0.432,0.40,85.54,0.42]]
summary=pd.DataFrame(data, columns = ['Model Algorithm', 'Recall','Precision','Accuracy','F
1'])
summary
```

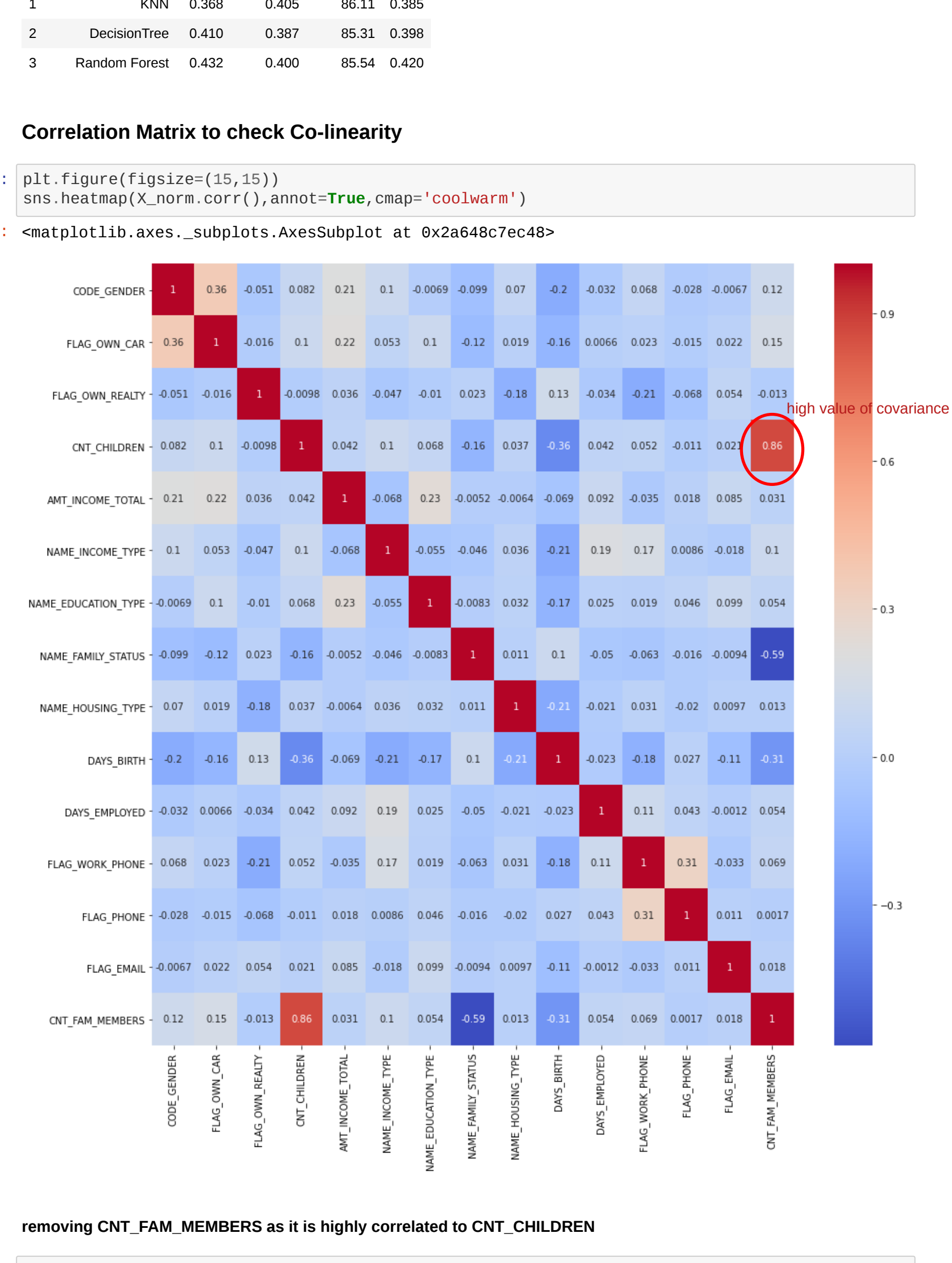
Out[37]:

	Model Algorithm	Recall	Precision	Accuracy	F1
0	Logistic Regression	0.517	0.130	54.41	0.207
1	KNN	0.368	0.405	86.11	0.385
2	DecisionTree	0.410	0.387	85.31	0.398
3	Random Forest	0.432	0.400	85.54	0.420

Correlation Matrix to check Co-linearity

In [38]:
```python
plt.figure(figsize=(15,15))
sns.heatmap(X_norm.corr(),annot=True,cmap='coolwarm')
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x2a648c7ec48>



removing CNT_FAM_MEMBERS as it is highly correlated to CNT_CHILDREN

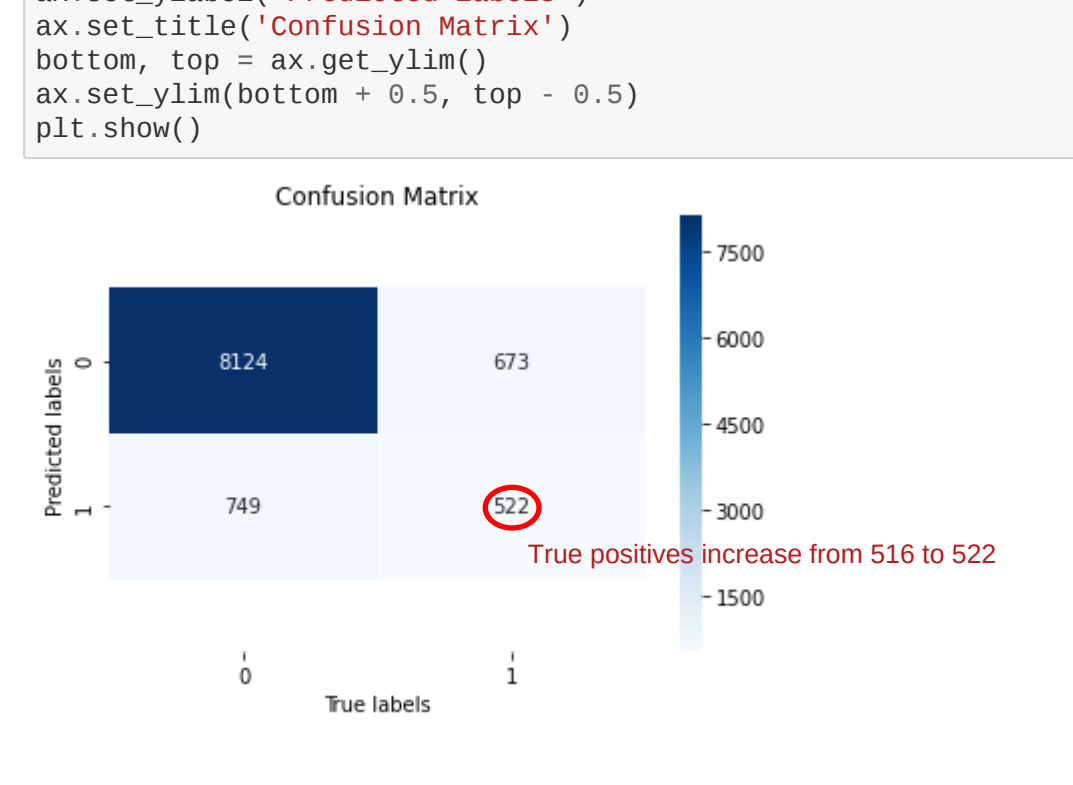
In [64]:
```python
X_norm_new=norm_dataset.drop(['ID','STATUS'],axis=1)
X_train_norm_new, X_test_norm_new, y_train_new, y_test_new = train_test_split(X_norm_new, y,
test_size=0.28, random_state=42)
X_train_smote_new,y_train_smote_new=smote.fit_resample(X_train_norm_new,y_train_new)
```

In [66]:
```python
clf_new = RandomForestClassifier(n_estimators=100,class_weight='balanced',criterion='entrop
y')
clf_new.fit(X_train_smote_new,y_train_smote_new)
predictions=clf_new.predict(X_test_norm_new)
print(accuracy_score(predictions,y_test_new))
confusion_matrix(predictions,y_test_new)
```

0.8587604290822408

Out[66]:
```
array([[8124,  673],
       [ 749,  522]], dtype=int64)
```

In [68]:
```python
cm=confusion_matrix(predictions,y_test_new)
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax,cmap='Blues',fmt='g')
ax.set_xlabel('True labels')
ax.set_ylabel('Predicted labels')
ax.set_title('Confusion Matrix')
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show()
```



Check the importance of each feature

In [71]:
```python
importance=clf_new.feature_importances_
```

In [73]:
```python
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
plt.bar([X_train_norm_new.columns[x] for x in range(len(importance))], importance)
plt.xticks(rotation = 90)
plt.show()
```

```
Feature: 0, Score: 0.03248
Feature: 1, Score: 0.03251
Feature: 2, Score: 0.03409
Feature: 3, Score: 0.02111
Feature: 4, Score: 0.22235
Feature: 5, Score: 0.03653
Feature: 6, Score: 0.03884
Feature: 7, Score: 0.04864
Feature: 8, Score: 0.02537
Feature: 9, Score: 0.22439
Feature: 10, Score: 0.16940
Feature: 11, Score: 0.02794
Feature: 12, Score: 0.03312
Feature: 13, Score: 0.02181
Feature: 14, Score: 0.03142
```