Predictive Equipment Failures

1. Introduction

Organization will use Stripper Well Equipment's to produce the oils at the well level. Due to low operation cost maintenance on equipment's, Organization will get a prominent profit on oil production. However, Equipment's will subject to fail (breakdown) due to Stripper Rubber shrinks, oil colour change, Feed gas temperature increase/decrease etc.,

Any unplanned downtime of equipment would degrade or interrupt a company's core business, potentially resulting in significant penalties and unmeasurable reputation loss.

1.1 Business Problem

Predict the asset failures upfront. Such that, based on the prediction, Organization will send crew to Physical Locations to fix the equipment or send a replacement equipment before fail which will rapidly decrease the downtime of equipment's and also helps to reduce the operational cost drastically and increase the revenue to the Organization.

1.2 Business Objectives and Constraints

- No Low Latency requirement. But latency should not be more than 5 minutes
- Errors can be very costly

1.3 Sources / Useful Links

- Some of the useful research papers related to Machine Learning approach for Predictive Maintenance are as follows

  1.https://www.researchgate.net/publication/327334242_Machine_Learning_approach_for_Predi (https://www.researchgate.net/publication/327334242_Machine_Learning_approach_for_Predict

  2.https://www.mdpi.com/2071-1050/12/19/8211 (https://www.mdpi.com/2071-1050/12/19/8211)

  3.https://www.sciencedirect.com/science/article/pii/S2212827118312988 (https://www.sciencedirect.com/science/article/pii/S2212827118312988)

1.4 About Dataset Analysis

Below are the provided datasets

Dataset	Description
`equip_failures_training_set.csv`	The training dataset. This document contains history of asset failures. This dataset contains “Target” feature information. “Target” feature represents whether the equipment failure happened in downhole or surface.
`equip_failures_test_set.csv`	The test set. This dataset doesn’t contain Target feature information. We must apply ML algorithms and predict the target.

Below are the features across datasets

Feature	Description
`Id`	Row Identifier
`Target`	Refers to downhole or surface equipment failure • `Target value 1 refers to downhole failures` • `Target value 0 refers to surface failures`
`Measure Columns`	Single Measurement Sensors data. Data is captured from 100 sensors. There are 100 columns information available in the dataset
`Histogram Bin Columns`	Refers to bins of a sensor that show its distribution over time. There are 70 columns belongs to Histogram bin

2. Mapping problem in to Supervised Learning Problem:

In [1]:
```
# import supported libraries
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pandas import DataFrame
import operator
from tqdm import tqdm
from sklearn import preprocessing
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, chi2
from scipy.stats import boxcox
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
```

2.1 Reading Data

In [2]:
```
EquipFail_train_dataset = pd.read_csv('equip_failures_training_set.csv')
print ("Total number of Training dataset records are (Rows X Columns):", EquipFai
```

```
Total number of Training dataset records are (Rows X Columns): (60000, 172)
```

In [3]:
```
#EquipFail_test_dataset = pd.read_csv('equip_failures_test_set.csv')
#print ("Total number of Test dataset records are (Rows X Columns):", EquipFail_t
```

Observation:

Due to "Target" feature doesn't exists in Test data set, total number of columns are shown as 171.

In [4]:
```
EquipFail_train_dataset.head(5)
```

Out[4]:

	id	target	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_m
0	1	0	76698	na	2130706438	280	
1	2	0	33058	na	0	na	
2	3	0	41040	na	228	100	
3	4	0	12	0	70	66	
4	5	0	60874	na	1368	458	

5 rows × 172 columns

Observation:

Each Row / Data point is a failure event. For each failure event, data has been collected from 107 sensors that collect a variety of physical information both on the surface and below the ground.

Noticed that, there are features which contains 'na' values. We will fill 'na' values with other values to predict the model better

In [5]:
```
print("Below are the list of columns in the Train dataset:")
EquipFail_train_dataset.columns
```

```
Below are the list of columns in the Train dataset:
```

Out[5]:
```
Index(['id', 'target', 'sensor1_measure', 'sensor2_measure', 'sensor3_measure',
       'sensor4_measure', 'sensor5_measure', 'sensor6_measure',
       'sensor7_histogram_bin0', 'sensor7_histogram_bin1',
       ...
       'sensor105_histogram_bin2', 'sensor105_histogram_bin3',
       'sensor105_histogram_bin4', 'sensor105_histogram_bin5',
       'sensor105_histogram_bin6', 'sensor105_histogram_bin7',
       'sensor105_histogram_bin8', 'sensor105_histogram_bin9',
       'sensor106_measure', 'sensor107_measure'],
      dtype='object', length=172)
```

3. EDA Analysis

In [6]:
```
EquipFail_train_dataset.describe(include=[np.number])
```

Out[6]:

	id	target	sensor1_measure
count	60000.000000	60000.000000	6.000000e+04
mean	30000.500000	0.016667	5.933650e+04
std	17320.652413	0.128020	1.454301e+05
min	1.000000	0.000000	0.000000e+00
25%	15000.750000	0.000000	8.340000e+02
50%	30000.500000	0.000000	3.077600e+04
75%	45000.250000	0.000000	4.866800e+04
max	60000.000000	1.000000	2.746564e+06

In [7]:
```
EquipFail_train_dataset.describe(include=[np.object])
```

Out[7]:

	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure
count	60000	60000	60000	60000	60000
unique	30	2062	1887	334	419
top	na	0	na	0	0
freq	46329	8752	14861	55543	55476

4 rows × 169 columns

Observation:

As we can see that, sensor2_measure feature contains more 'na' records than the sensor integer information. We can remove such sensors information as it will not much useful for our modelling. We will find such sensors going further and will remove if required.

3.1 UniVariate Analysis

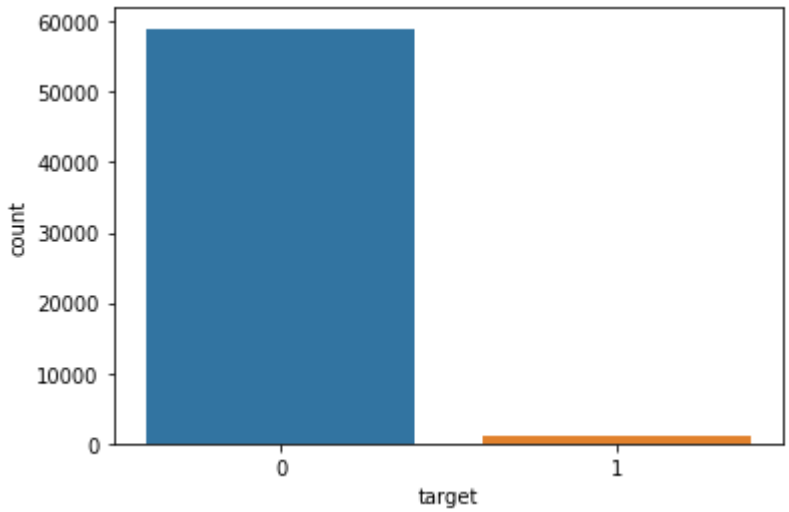
In [8]:
```
EquipFail_train_dataset.index
```

Out[8]:
```
RangeIndex(start=0, stop=60000, step=1)
```

In [9]:
```
EquipFail_train_dataset_valuecounts = EquipFail_train_dataset['target'].value_cou
EquipFail_train_dataset_valuecounts
```

Out[9]:
```
0    59000
1     1000
Name: target, dtype: int64
```

In [10]:
```
#https://seaborn.pydata.org/generated/seaborn.countplot.html
sns.countplot(x =EquipFail_train_dataset["target"], data = EquipFail_train_datase
plt.show()

print ("Percentage of Target data points equal to 0 are: ",(EquipFail_train_datas
print ("Percentage of Target data points equal to 1 are: ",(EquipFail_train_datas
```




```
Percentage of Target data points equal to 0 are:  98.33333333333333 %
Percentage of Target data points equal to 1 are:  1.6666666666666667 %
```

Observation:

1) As per above analysis, there are 98% of failures which are recorded against Surface related failures and 1.6% of failures are belongs to Downhole related failures.

2) We can understand that, Data is imbalanced. We will apply following methods to support imbalanced dataset.

1) Upsampling or Downsampling
2) Drop Class Priors
3) Modify the model to account class imbalance

3.1.1 Univariate Analysis Through Line Plot

In [11]:
```
#replace 'na' strings to zero's
EquipFail_train_dataset=EquipFail_train_dataset.replace('na',np.NaN)
```

In [12]:
```
#convert columns of object datatype to float datatype to perform EDA analysis
for columnname in EquipFail_train_dataset:
    #print (i)
    EquipFail_train_dataset[columnname]=EquipFail_train_dataset[columnname].astyp
```

In [13]:
```
EquipFail_train_dataset.head()
```

Out[13]:

	id	target	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_n
0	1.0	0.0	76698.0	NaN	2.130706e+09	280.0	
1	2.0	0.0	33058.0	NaN	0.000000e+00	NaN	
2	3.0	0.0	41040.0	NaN	2.280000e+02	100.0	
3	4.0	0.0	12.0	0.0	7.000000e+01	66.0	
4	5.0	0.0	60874.0	NaN	1.368000e+03	458.0	

5 rows × 172 columns

In [14]:
```
### find out the count of NaN information in all sensors and will remove feature


NaN_Values_count =[]

NaN_Values_count = [[featuress,EquipFail_train_dataset.shape[0],EquipFail_train_d
NaN_Values_count

#lets convert list to datafram for easy understanding
#https://datatofish.com/list-to-dataframe/

NaN_Values_count_df = DataFrame (NaN_Values_count,columns=['Column_Name','Number


NaN_Values_count_df.sort_values(by=['Percentage of NaN'], inplace=True,ascending=
NaN_Values_count_df.head(16)
```

Out[14]:

	Column_Name	Number of Records (All)	Number of records (NaN)	Percentage of NaN
80	sensor43_measure	60000	49264	82.106667
79	sensor42_measure	60000	48722	81.203333
78	sensor41_measure	60000	47740	79.566667
77	sensor40_measure	60000	46333	77.221667
114	sensor68_measure	60000	46329	77.215000
3	sensor2_measure	60000	46329	77.215000
76	sensor39_measure	60000	44009	73.348333
75	sensor38_measure	60000	39549	65.915000
74	sensor37_measure	60000	27277	45.461667
73	sensor36_measure	60000	23034	38.390000
93	sensor56_measure	60000	14861	24.768333
94	sensor57_measure	60000	14861	24.768333

Observation

- sensor43_measure, sensor42_measure, sensor41_measure, sensor40_measure, sensor68_measure, sensor2_measure , sensor39_measure, sensor38_measure contains more number of NaN values compared with other sensor features

In [15]:
```
#Lets drop histogram sensors

#for histcolumnname in EquipFail_train_dataset:
#    #print (i)
#    if "histogram" in histcolumnname:
#        EquipFail_train_dataset.drop([histcolumnname], axis=1, inplace=True)
```

In [16]:
```
EquipFail_train_dataset.head()
```

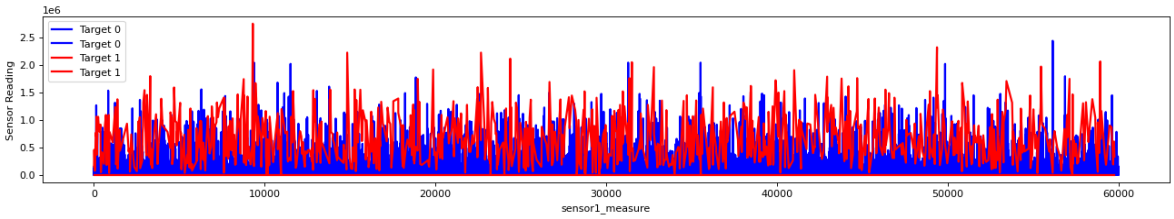
Out[16]:

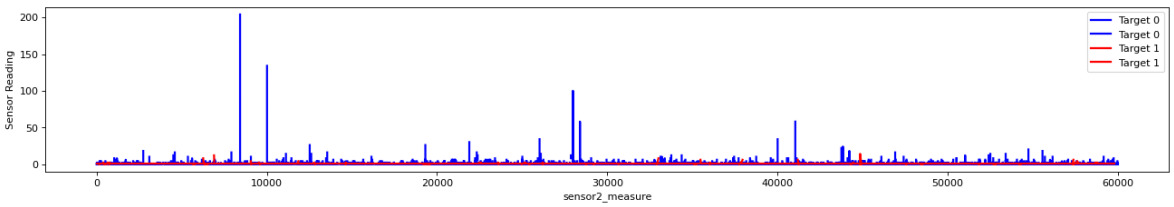
	id	target	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_n
0	1.0	0.0	76698.0	NaN	2.130706e+09	280.0	
1	2.0	0.0	33058.0	NaN	0.000000e+00	NaN	
2	3.0	0.0	41040.0	NaN	2.280000e+02	100.0	
3	4.0	0.0	12.0	0.0	7.000000e+01	66.0	
4	5.0	0.0	60874.0	NaN	1.368000e+03	458.0	

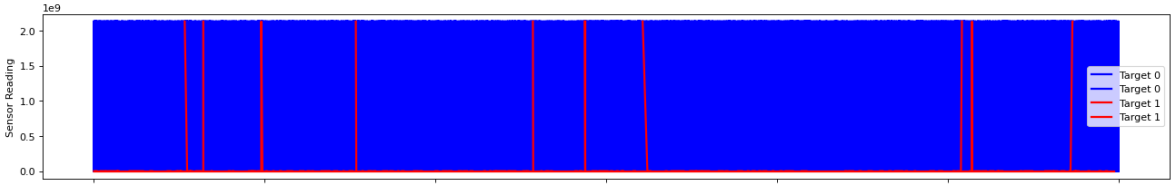
5 rows × 172 columns

In [17]:
```
#Lets remove column id
EquipFail_train_dataset.drop(['id'], axis=1, inplace=True)
```

In [18]:
```
for top15columns_plot in EquipFail_train_dataset.columns[1:]:
    target_0_1=EquipFail_train_dataset[["target",top15columns_plot]]
    target_0=target_0_1[target_0_1["target"]==0]
    target_1=target_0_1[target_0_1["target"]==1]
    plt.figure(num=None, figsize=(20, 3), dpi=80, facecolor='w', edgecolor='k')
    plt.plot(target_0, color='blue', linewidth = 2,  label = 'Target 0')
    plt.plot(target_1, color='red', linewidth = 2,  label = 'Target 1')
    plt.xlabel(top15columns_plot)
    plt.ylabel('Sensor Reading')
    plt.legend()
    plt.show()
```







Observation

Below are few observations:

- Sensor92_measure contains very less trends for Target 0 (Surface failure) and 1 (downhole failure). Noticed that, most of the sensor92_measure values are either zero or null values.
- Sensor102_measure contains high peaks for surface (Target 0) failures.
- Most of the sensor 3 data points are very high for Target 0 and 1. There is a high chance that, if Sensor 3 reaches to high value then Equipment might fail. Noticed that, there are very less spikes for Target 0 when compared with Target 1
- Sensor 36, 37 and 38 has similar distribution. Target 0 (Surface failure) data points has wide spread
- Sensor 54 graph distribution is looks odd. Lets see more information on sensor 54 later

3.1.2 Univariate Analysis Through PDF

In [19]:
```
#Lets plot PDF against each column towards target variable to understand the data

for top15columns in EquipFail_train_dataset.columns[1:]:
    sns.FacetGrid(EquipFail_train_dataset, hue="target", size=5).map(sns.distplot
    plt.show()
```

```
c:\program files\python36\lib\site-packages\seaborn\axisgrid.py:316: UserWar
ning: The `size` parameter has been renamed to `height`; please update your
code.
  warnings.warn(msg, UserWarning)
c:\program files\python36\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a f
uture version. Please adapt your code to use either `displot` (a figure-leve
l function with similar flexibility) or `histplot` (an axes-level function f
or histograms).
  warnings.warn(msg, FutureWarning)
c:\program files\python36\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a f
uture version. Please adapt your code to use either `displot` (a figure-leve
l function with similar flexibility) or `histplot` (an axes-level function f
or histograms).
  warnings.warn(msg, FutureWarning)
```

Observation :

Below are some of the observations:

- sensor92_measure plot has too much of overlapping between target variable 0 and 1. PDF shows that, senssor92_measure has a right skweness (outliers).

- Below sensors PDF are similar and the data is overlapped for a target variable 0 and 1.

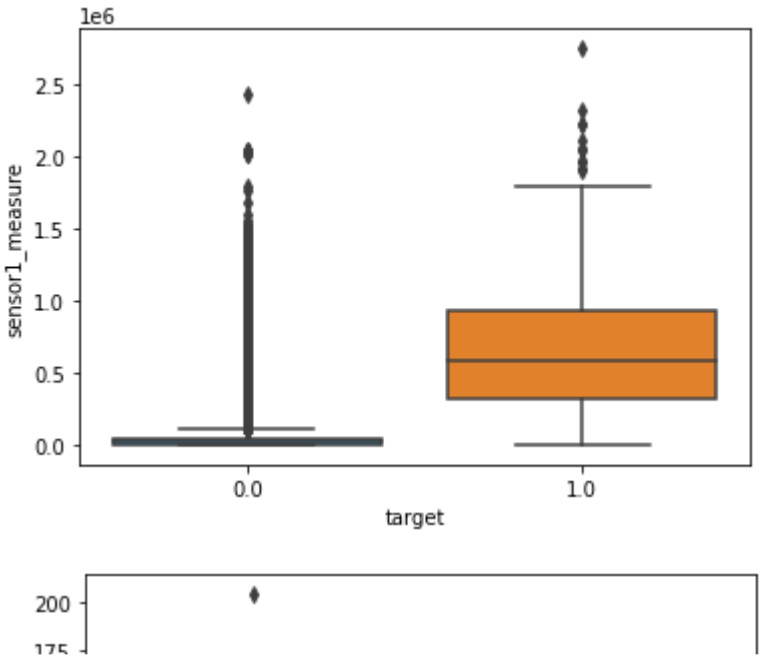
  sensor27_measure, sensor47_measure, sensor46_measure, sensor67_measure, sensor48 measure, sensor53 measure, sensor59 measure

- Noticed that, there are outliers for sensor3_measure after value 2 in X-axis
- Below sensors PDF are similar and the data is overlapped for a target variable 0 and 1.

  sensor15_measure, sensor8_measure, sensor32_measure, sensor16_measure
- All sensors features has a skwness problem and contains outliers as well

3.1.3 Univariate Analysis Through BOX Plot

In [20]:
```
for top15columns_boxplot in EquipFail_train_dataset.columns[1:]:
    sns.boxplot(x='target',y=top15columns_boxplot, data=EquipFail_train_dataset)
    plt.show()
```



Observation :

- sensor92_measure: IQR range for target 0 and 1 is near to zero. It's because most of the values are zero. Target 0 has many outliers when compared with Target 1
- Sensor27_measure, Sensor47_Measure, Sensor46_Measure, Sensor67_Measure, Sensor48_Measure, Sensor53_Measure,, Sensor59_Measure, Sensor14_Measure, Sensor15_Measure, Sensor8_Measure, Sensor32_Measure:

```
    1) Observed that, 75 percentile of Surface Failures (Target 0)
is less than 25th percentile of Downhole failures (Target 1). Which
 means, downhole failure trend is higher than surface failures
    2) Median line of a box plot between target 0 and 1 lies outsi
de of the box, which means there is a difference between two target
 1 and 0
    3) None of the sensors has a normal distribution but contains
 a right skewed with outliers
```

3.2 Pre Processing

In [18]:
```
#Lets remove the features which contains more than 70% of NaN values

#EquipFail_train_dataset.drop(['sensor43_measure','sensor42_measure','sensor41_me
EquipFail_train_dataset.drop(['sensor43_measure','sensor42_measure','sensor41_mea

EquipFail_train_dataset
```

Out[18]:

	target	sensor1_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_m
0	0.0	76698.0	2.130706e+09	280.0	0.0	
1	0.0	33058.0	0.000000e+00	NaN	0.0	
2	0.0	41040.0	2.280000e+02	100.0	0.0	
3	0.0	12.0	7.000000e+01	66.0	0.0	
4	0.0	60874.0	1.368000e+03	458.0	0.0	
...	...	...	...	...	...	
59995	0.0	153002.0	6.640000e+02	186.0	0.0	
59996	0.0	2286.0	2.130707e+09	224.0	0.0	
59997	0.0	112.0	2.130706e+09	18.0	0.0	
59998	0.0	80292.0	2.130706e+09	494.0	0.0	
59999	0.0	40222.0	6.980000e+02	628.0	0.0	

60000 rows × 162 columns

3.2.1 Replace Np.Nan with Median values

There are high chances that Mean, Std Varaince and Variance values could be wrong if one outlier found.

Median values will provide correct results even though if data have outliers.

As we can observe during EDA that, there are multiple outliers are visible across the dataset. Hence, its better to replace 'Np.Nan' with Median to overcome outliers when compared with Mean, std variance etc.,

In [19]:
```
#Apply for loop to replace Nan with median
for equipfailcolumns in EquipFail_train_dataset.columns[1:]:
    EquipFail_train_dataset[equipfailcolumns].fillna(EquipFail_train_dataset[equi
```

In [20]:
```
X_before_Normalization = EquipFail_train_dataset.copy()
```

Conclusion on EDA Analysis:

Performed EDA analysis on the sensor features as follows:

1) Performed Univariate analysis on class label (Target) and found out that, the dataset is imbalanced. Due to the data is imbalanced, F1score or AUC score or Confusion matrix is best metrics to check the ML model performance.

2) Analysed the features by applying Line plot. Observed that,

```
- Sensor92_measure contains very less trends for Target 0 (Surface failu
re) and 1 (downhole failure)
- Sensor102_measure contains high peaks for surface (Target 0) failures.
- Most of the sensor 3 data points are very high for Target 0 and 1. The
re is a high chance that, if Sensor 3 reaches to high value then Equipme
nt might fail. Noticed that, there are very less spikes for Target 0 whe
n compared with Target 1
- Sensor 36, 37 and 38 has similar distribution. Target 0 (Surface failu
re) data points has wide spread
- Sensor 54 graph distribution is looks odd. Lets see more information o
n sensor 54 later
```

3) Plotted PDF plot and found that, sensor15_measure, sensor8_measure, sensor32_measure, sensor16_measure has a similar distribution

4) Plotted Box plot against each feature and observed a few things on following sensors.

- Sensor27_measure, Sensor47_Measure, Sensor46_Measure, Sensor67_Measure, Sensor48_Measure, Sensor53_Measure,, Sensor59_Measure, Sensor14_Measure, Sensor15_Measure, Sensor8_Measure, Sensor32_Measure:

```
1) 75 percentile of Surface Failures (Target 0) is less than 25th pe
rcentile of Downhole failures (Target 1). Which means, downhole fail
ure trend is higher than surface failures
2) Median line of a box plot between target 0 and 1 lies outside of
 the box, which means there is a difference between two target 1 and
0
3) None of the sensors has a normal distribution but contains a righ
t skewed with outliers
```

5) Performed Preprocessing on Dataset

```
- Dropped unused columns which contains more than 70% of data has NaN va
lues
- Imputed NaN values with median
```

4. Apply ML Models

Below are the choosen ML models for the given highly imbalanced dataset

1) Logistic Regression

2) Random Forest Classifier

3) LightGBM Classifier

3) XGBoost Classifier

Have tried other ML models, but above ML models are performing better than others.

4.1 Approach 1

Lets apply the following functinoalities in the dataset and will proceed with applying ML models to predict the performance metric

- Remove unused features (>70% of data contains null values)
- Impute median for the missing values
- Apply Normlization on the dataset
- Apply ML Models

This approach will be considered as a base model and will help to provide the least performance metric scores. Following by, will apply more techniques (like feature enigerring, reduce outliers etc.,) to improve the metrics better than Approach 1 metric scores.

Since we have already done the preprocessing on the dataset in previous section, lets apply ML Models to it.

Splitting data into Train and cross validation(or test): Stratified Sampling

In [21]:
```
y = X_before_Normalization['target'].values
X_before_Normalization.drop(['target'], axis=1, inplace=True)
X = X_before_Normalization
X.head(1)
```

Out[21]:

	sensor1_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sen
0	76698.0	2.130706e+09	280.0	0.0	0.0	

1 rows × 161 columns

In [22]:
```
# train test split
from sklearn.model_selection import train_test_split
#splitting data in to train and test with 20 percentage as test data
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.2, stra
```

Apply Normalization

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.

In [23]:
```
#normalize dataframe https://stackoverflow.com/questions/26414913/normalize-colum

#X_norm = EquipFail_train_dataset.copy() #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler().fit(X_train1)
X_train1 = pd.DataFrame(min_max_scaler.transform(X_train1),columns=X_train1.colum
X_test1 = pd.DataFrame(min_max_scaler.transform(X_test1),columns=X_test1.columns)
#X_normalized = min_max_scaler.fit_transform(X_norm)
#X_norm_final = pd.DataFrame(min_max_scaler.fit_transform(X_norm),columns=X_norm.
X_train1
```

Out[23]:

	sensor1_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure
0	0.394478	2.909833e-07	0.000647	0.0	0.0
1	0.014155	4.421068e-07	0.004619	0.0	0.0
2	0.000226	9.999998e-01	0.000092	0.0	0.0
3	0.024482	9.999998e-01	0.003438	0.0	0.0
4	0.016437	1.013748e-07	0.000965	0.0	0.0
...	...	...	...	...	...
47995	0.000711	7.133783e-08	0.000647	0.0	0.0
47996	0.013839	3.510572e-07	0.003212	0.0	0.0
47997	0.000307	1.220252e-08	0.000133	0.0	0.0
47998	0.170086	7.133783e-08	0.000647	0.0	0.0
47999	0.018058	1.342278e-07	0.001334	0.0	0.0

48000 rows × 161 columns

Define KFold CV

In [24]:
```
from sklearn.model_selection import KFold
cv=KFold(n_splits=5,random_state=None, shuffle=False)
```

find best threshold and plot confusion matrix

In [25]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold",
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

print("="*100)
```

```
====================================================================================
====================
```

4.1.1 Logistic Regression

In [26]:
```
# Copy splitted data
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [27]:
```
#Lets take hyper parameter between 10**-4 to 10**4.
#Have used Class_weight parameter to take care of imbalance dataset

from sklearn.linear_model import LogisticRegression
parametersK_LR = {'C' : [(10**x) for x in range(-4,4)] }
LogRegression = GridSearchCV(LogisticRegression(class_weight='balanced'),paramete
LogRegression.fit(X_train,y_train)
LogRegression.cv_results_
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

Out[27]:
```
{'mean_fit_time': array([0.46797147, 0.54627533, 0.83533592, 1.35096889, 3.1
8867364,
        3.5429729 , 3.07030773, 2.52654486]),
 'std_fit_time': array([0.06615615, 0.06606958, 0.10481043, 0.19969954, 0.32
94432 ,
        0.43249576, 0.18053304, 0.20702107]),
 'mean_score_time': array([0.01328449, 0.01836286, 0.01698995, 0.03437371,
0.0150373 ,
        0.01444511, 0.0125278 , 0.01093082]),
 'std_score_time': array([0.00115966, 0.01039464, 0.00811805, 0.01864344, 0.
00228023,
        0.0021712 , 0.00184993, 0.00189355]),
 'param_C': masked_array(data=[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
              mask=[False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.0001},
  {'C': 0.001},
  {'C': 0.01},
  {'C': 0.1},
  {'C': 1},
  {'C': 10},
  {'C': 100},
  {'C': 1000}],
 'split0_test_score': array([0.97302083, 0.96875   , 0.96458333, 0.96572917,
0.97083333,
        0.97614583, 0.9765625 , 0.97645833]),
 'split1_test_score': array([0.97635417, 0.97072917, 0.96510417, 0.96645833,
0.97114583,
        0.97395833, 0.97302083, 0.97270833]),
 'split2_test_score': array([0.975     , 0.97114583, 0.96625   , 0.96677083,
0.9715625 ,
        0.975     , 0.9746875 , 0.97489583]),
 'split3_test_score': array([0.97427083, 0.96958333, 0.9640625 , 0.96520833,
0.96958333,
        0.97270833, 0.97354167, 0.9734375 ]),
 'split4_test_score': array([0.97020833, 0.96604167, 0.96239583, 0.961875  ,
0.9640625 ,
        0.96895833, 0.9690625 , 0.969375  ]),
 'mean_test_score': array([0.97377083, 0.96925   , 0.96447917, 0.96520833,
0.9694375 ,
        0.97335417, 0.973375  , 0.973375  ]),
 'std_test_score': array([0.00208271, 0.0018131 , 0.00126895, 0.00175421, 0.
00276746,
```

```
       0.00247417, 0.00247522, 0.00237774]),
 'rank_test_score': array([1, 6, 8, 7, 5, 4, 2, 2])}
```

In [28]:
```
BestLogRegParam = LogRegression.best_params_
BestLogRegParam
```

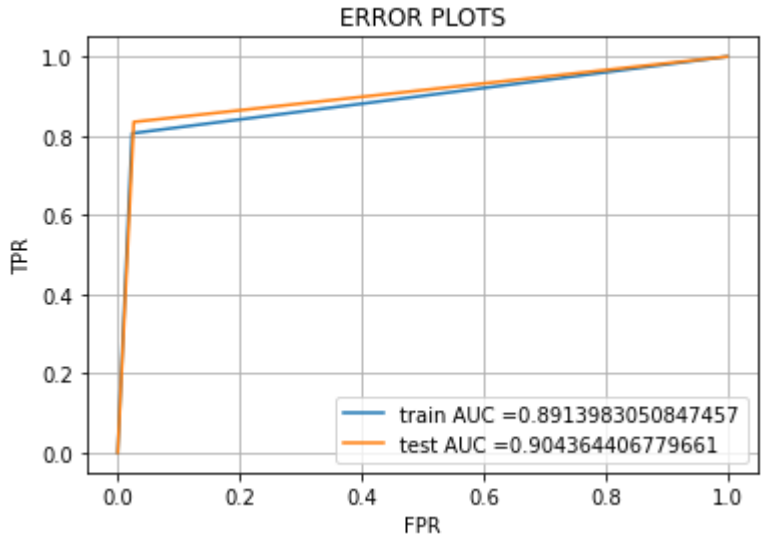
Out[28]:
```
{'C': 0.0001}
```

In [29]:
```
BestLogRegParam = BestLogRegParam.get('C')
```

Observation

- 0.0001 is the Best hyperparamter provided by GridsearchCV. Let apply the ML model with best hyper paramter to find out the metrics.

In [30]:
```
#Apply Logistic regression with best hyper parameter to predict the metrics

LogRegressionClass = LogisticRegression(C=BestLogRegParam, class_weight='balanced
LogRegressionClass.fit(X_train, y_train)

y_train_LogReg_pred = LogRegressionClass.predict(X_train)
y_test_LogReg_pred = LogRegressionClass.predict(X_test)

train_LogReg_fpr, train_LogReg_tpr, train_LogReg_thresholds = roc_curve(y_train,
test_LogReg_fpr, test_LogReg_tpr, test_LogReg_thresholds = roc_curve(y_test, y_te
#plot AUC
plt.plot(train_LogReg_fpr, train_LogReg_tpr, label="train AUC ="+str(auc(train_Lo
plt.plot(test_LogReg_fpr, test_LogReg_tpr, label="test AUC ="+str(auc(test_LogReg
AUC_LogReg_Train = str(auc(train_LogReg_fpr, train_LogReg_tpr))
AUC_LogReg_Test = str(auc(test_LogReg_fpr, test_LogReg_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



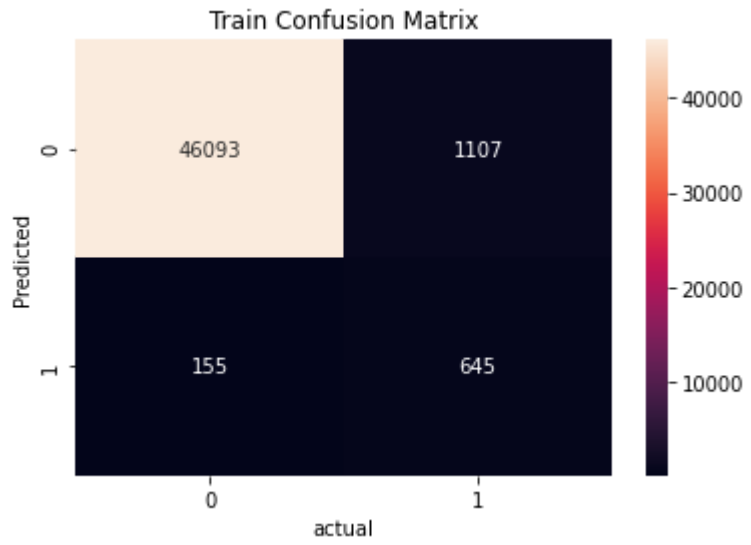
Observations:

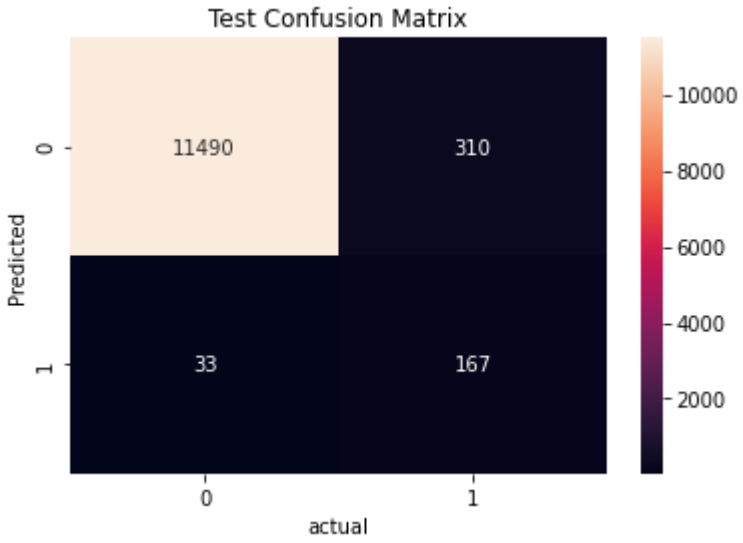
Train Accuray with Hyper Parameter 0.0001 is 89 %

Test Accuray with Hyper Parameter 0.0001 is 90 %

In [31]:
```
#heatmap confusion matrix https://stackoverflow.com/questions/35572000/how-can-i-
best_LogReg_t = find_best_threshold(train_LogReg_thresholds, train_LogReg_fpr, tr
LogReg_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_train_Log
df_cm1=pd.DataFrame(LogReg_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

#heatmap confusion matrix https://stackoverflow.com/questions/35572000/how-can-i-
LogReg_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_LogReg
df_cm1=pd.DataFrame(LogReg_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")


plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.7873407044491526 for threshold 1.0
```





Observations:

Logistic Regression model with hyper parameter 0.0001 is not a sensible model as TPR rates are very low when compared with FNR for Train and Test confusion matrix. Which means, Logistic regression model is wrongly predicting positive (Target 1) class as Negative (Target 0) class

In [32]:
```
from sklearn.metrics import f1_score
F1_Score_LogReg_Train=f1_score(y_train,y_train_LogReg_pred)
F1_Score_LogReg_Test=f1_score(y_test,y_test_LogReg_pred)
print('Train f1 score',f1_score(y_train,y_train_LogReg_pred))
print('Test f1 score',f1_score(y_test,y_test_LogReg_pred))
```

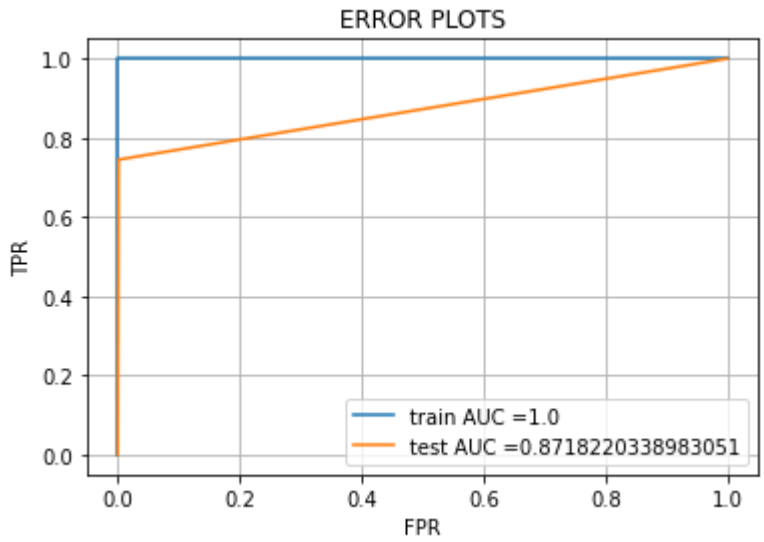
```
Train f1 score 0.5054858934169278
Test f1 score 0.4933530280649926
```

Observations:

Logistic Regression F1 Scores are pretty bad

4.1.2 Random Forest

In [33]:
```
from sklearn.ensemble import RandomForestClassifier

RFClass_Final = RandomForestClassifier(n_estimators=1000,n_jobs=-1,random_state=0
RFClass_Final.fit(X_train, y_train)

y_train_RFClass_pred = RFClass_Final.predict(X_train)
y_test_RFClass_pred = RFClass_Final.predict(X_test)

train_RFClass_fpr, train_RFClass_tpr, train_RFClass_thresholds = roc_curve(y_trai
test_RFClass_fpr, test_RFClass_tpr, test_RFClass_thresholds = roc_curve(y_test, y

plt.plot(train_RFClass_fpr, train_RFClass_tpr, label="train AUC ="+str(auc(train_
plt.plot(test_RFClass_fpr, test_RFClass_tpr, label="test AUC ="+str(auc(test_RFCl
AUC_RFClass_Train = str(auc(train_RFClass_fpr, train_RFClass_tpr))
AUC_RFClass_Test = str(auc(test_RFClass_fpr, test_RFClass_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



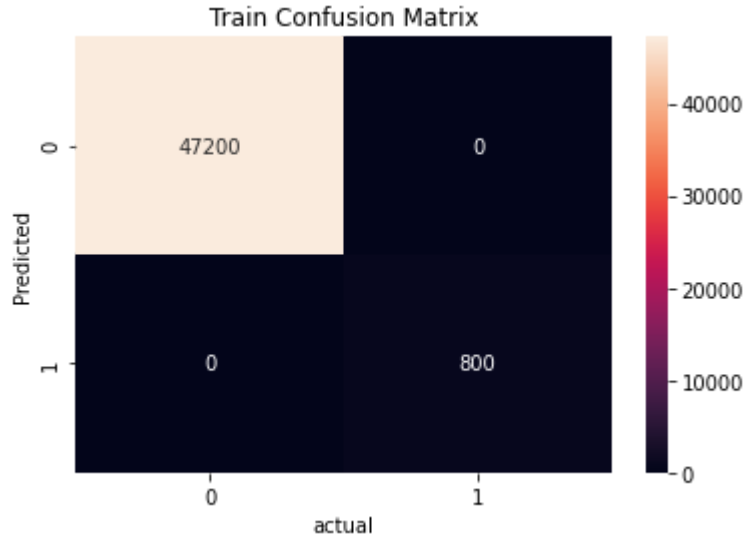
Observations:

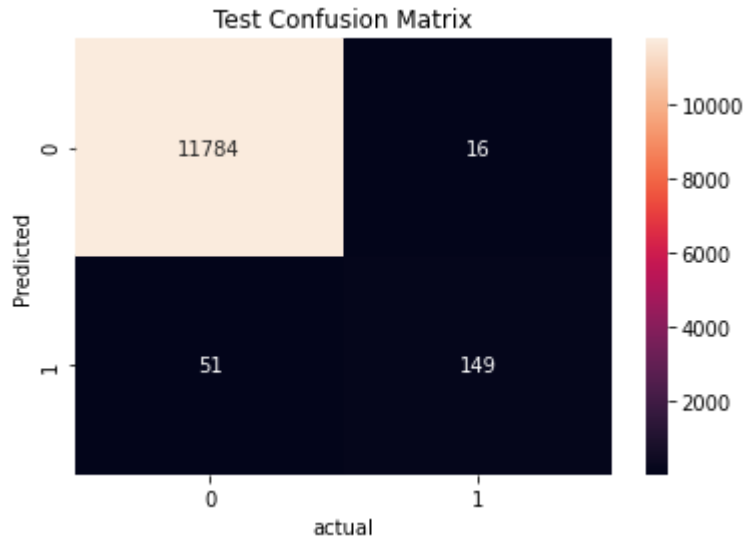
Train Accuray with Hyper Parameter ('n_estimators': 1000) is 100 %

Test Accuray with Hyper Parameter ('n_estimators': 10000) is 87 %

In [34]:
```
best_RFClass_t = find_best_threshold(train_RFClass_thresholds, train_RFClass_fpr,
RFClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_train_RF
df_cm1=pd.DataFrame(RFClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

#heatmap confusion matrix https://stackoverflow.com/questions/35572000/how-can-i-
RFClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_RFCla
df_cm1=pd.DataFrame(RFClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





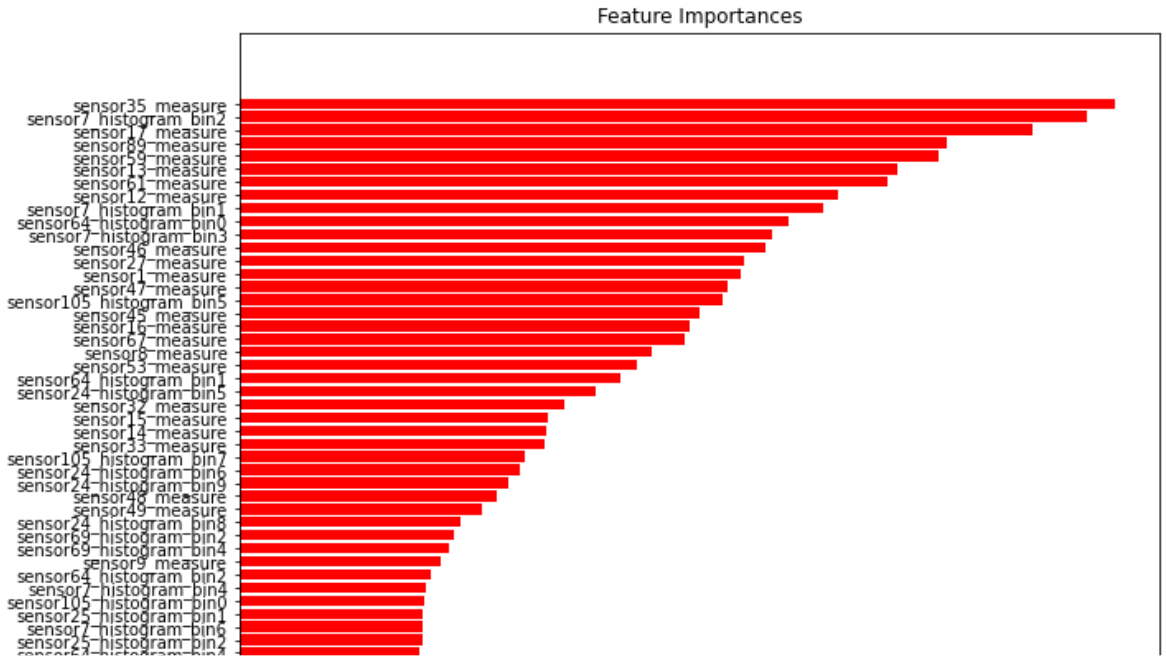
Observations:

Random Forest is sensible model as TPR and TNR rates are high when compared with FNR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [35]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_RFClass_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_RFClass_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00       800

    accuracy                           1.00     48000
   macro avg       1.00      1.00      1.00     48000
weighted avg       1.00      1.00      1.00     48000

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.90      0.74      0.82       200

    accuracy                           0.99     12000
   macro avg       0.95      0.87      0.91     12000
weighted avg       0.99      0.99      0.99     12000
```

In [36]:
```
RF_Train_features = X_train.columns
RF_importances = RFClass_Final.feature_importances_
RF_indices = (np.argsort(RF_importances))[-100:]
plt.figure(figsize=(10,16))
plt.title('Feature Importances')
plt.barh(range(len(RF_indices)), RF_importances[RF_indices], color='r', align='ce
plt.yticks(range(len(RF_indices)), [RF_Train_features[i] for i in RF_indices])
plt.xlabel('Relative Importance')
plt.show()
```



Observations:

Observed that, Sensor 17, 13, 12, 45, 59, 61, 89, 1, 27 and 45 are top 10 most useful and important features

4.1.3 LGBM Classifier

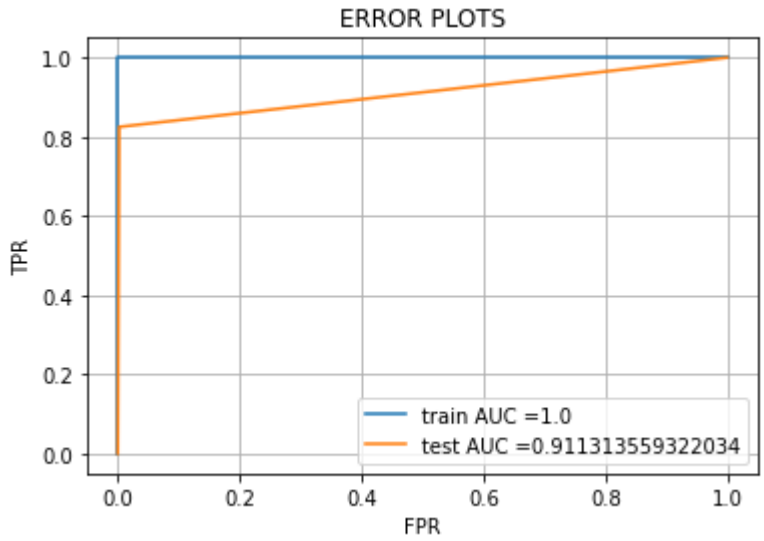
Apply ML Model with best parameters

In [37]:
```
#Hyper parameters are tunned manually and below are best hyper parameters
import lightgbm as lgb
clf1 = lgb.LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=25
               n_estimators=200, num_leaves=900, objective='binary',
               random_state=0, scale_pos_weight=59, silent=False)
clf1.fit(X_train, y_train)
```

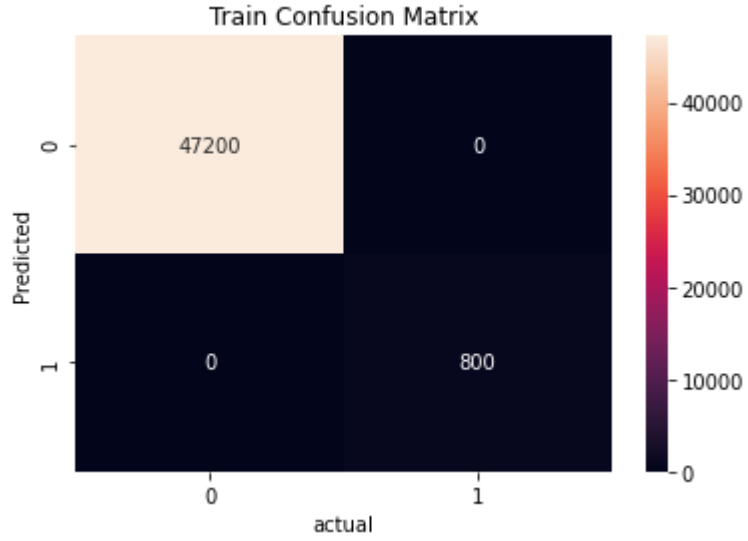
```
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignore
d. Current value: boosting=gbdt
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignore
d. Current value: boosting=gbdt
[LightGBM] [Info] Number of positive: 800, number of negative: 47200
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.074089 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 37203
[LightGBM] [Info] Number of data points in the train set: 48000, number of u
sed features: 159
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

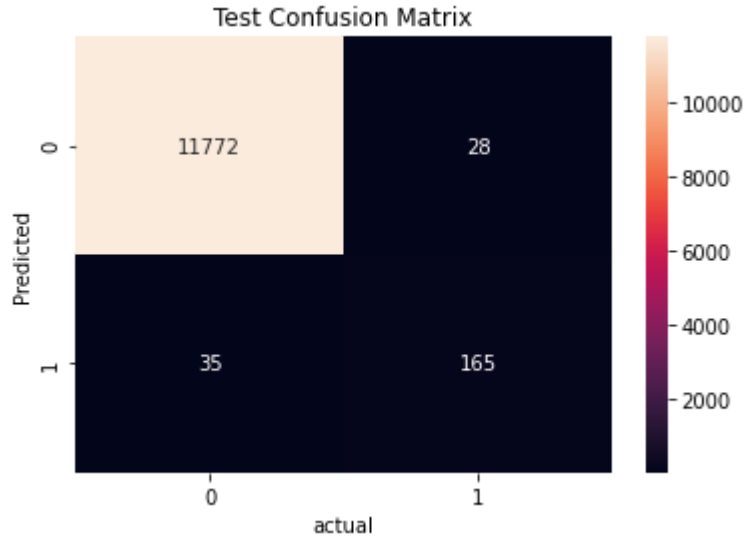
Plot AUC, Confusion Matrix and F1 Scores

In [38]:
```
y_train_pred = clf1.predict(X_train)
y_test_pred = clf1.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





```
Train f1 score 1.0
Test f1 score 0.8396946564885495
```

Observations:

Train Accuray with Hyper Parameter (max_depth=25, n_estimators=200) is 100 %

Test Accuray with Hyper Parameter (max_depth=25, n_estimators=200) is 91 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

F1 Score improved compared with previous ML models

In [39]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00       800

    accuracy                           1.00     48000
   macro avg       1.00      1.00      1.00     48000
weighted avg       1.00      1.00      1.00     48000

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.85      0.82      0.84       200

    accuracy                           0.99     12000
   macro avg       0.93      0.91      0.92     12000
weighted avg       0.99      0.99      0.99     12000
```

Feature Importance

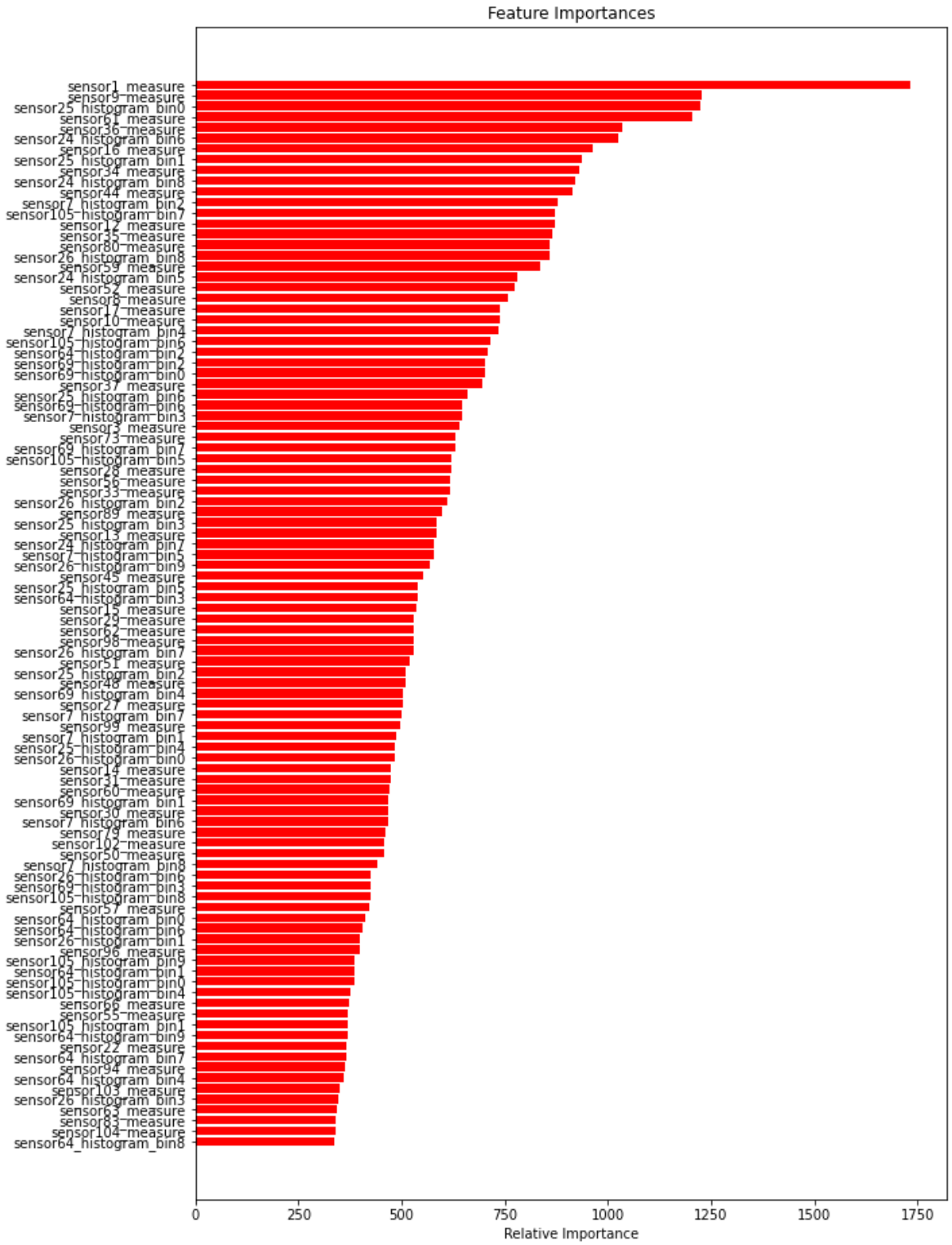
In [40]:
```
# Feature importance

clf1.booster_.feature_importance()

# importance of each attribute
fea_imp_ = pd.DataFrame({'cols':X_train.columns, 'fea_imp':clf1.feature_importanc
fea_imp_.loc[fea_imp_.fea_imp >=0].sort_values(by=['cols'], ascending = True)
```

Out[40]:

	cols	fea_imp
144	sensor100_measure	3
145	sensor101_measure	1
146	sensor102_measure	459
147	sensor103_measure	349
148	sensor104_measure	340
...	...	...
140	sensor96_measure	400
141	sensor97_measure	268
142	sensor98_measure	529
143	sensor99_measure	497
16	sensor9_measure	1226

161 rows × 2 columns

In [41]:
```
RF_Train_features = X_train.columns
RF_importances = clf1.feature_importances_
RF_indices = (np.argsort(RF_importances))[-100:]
plt.figure(figsize=(10,16))
plt.title('Feature Importances')
plt.barh(range(len(RF_indices)), RF_importances[RF_indices], color='r', align='ce
plt.yticks(range(len(RF_indices)), [RF_Train_features[i] for i in RF_indices])
plt.xlabel('Relative Importance')
plt.show()
```

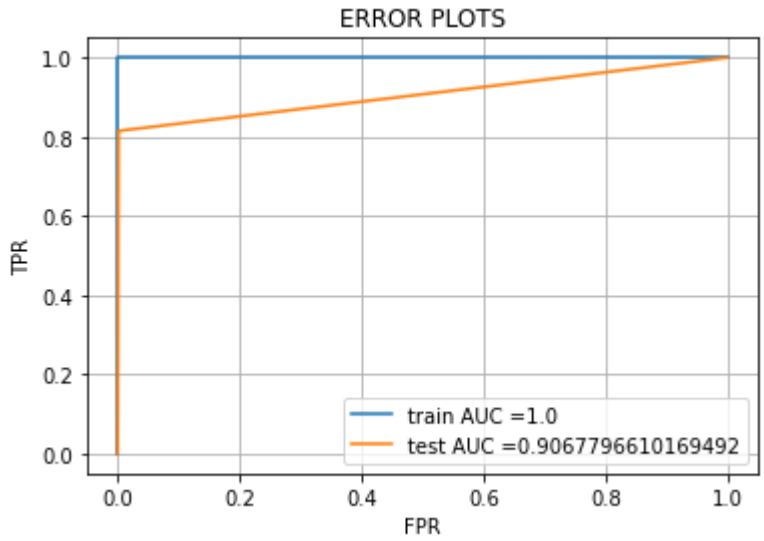


4.1.4 XGBoost

Apply ML Model with best parameters

In [42]:
```
#Hyper parameters are tunned manually and below are best hyper parameters
from xgboost import XGBClassifier
XGBoostClass_Final = XGBClassifier(learning_rate =0.1,n_estimators=1000,max_depth
XGBoostClass_Final.fit(X_train, y_train)

y_train_XGBoostClass_pred = XGBoostClass_Final.predict(X_train)
y_test_XGBoostClass_pred = XGBoostClass_Final.predict(X_test)

train_XGBoostClass_fpr, train_XGBoostClass_tpr, train_XGBoostClass_thresholds = r
test_XGBoostClass_fpr, test_XGBoostClass_tpr, test_XGBoostClass_thresholds = roc_

plt.plot(train_XGBoostClass_fpr, train_XGBoostClass_tpr, label="train AUC ="+str(
plt.plot(test_XGBoostClass_fpr, test_XGBoostClass_tpr, label="test AUC ="+str(auc
AUC_XGBoostClass_Train = str(auc(train_XGBoostClass_fpr, train_XGBoostClass_tpr))
AUC_XGBoostClass_Test = str(auc(test_XGBoostClass_fpr, test_XGBoostClass_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
c:\program files\python36\lib\site-packages\xgboost\sklearn.py:892: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[16:44:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```



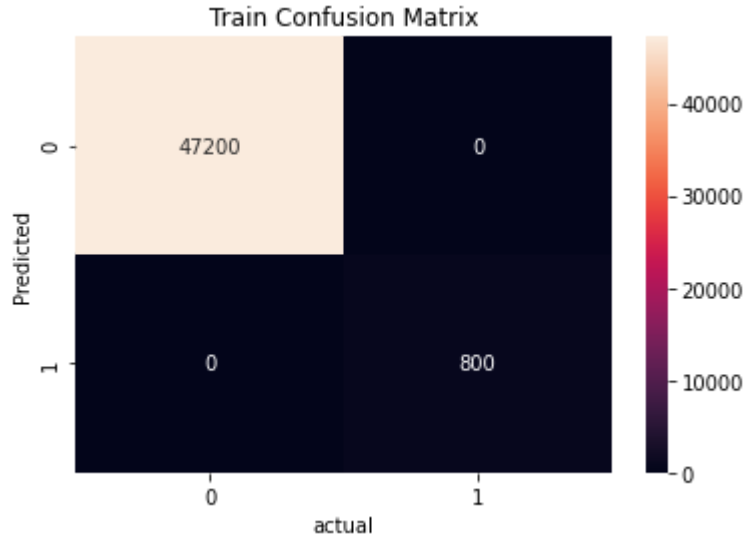
Observations:

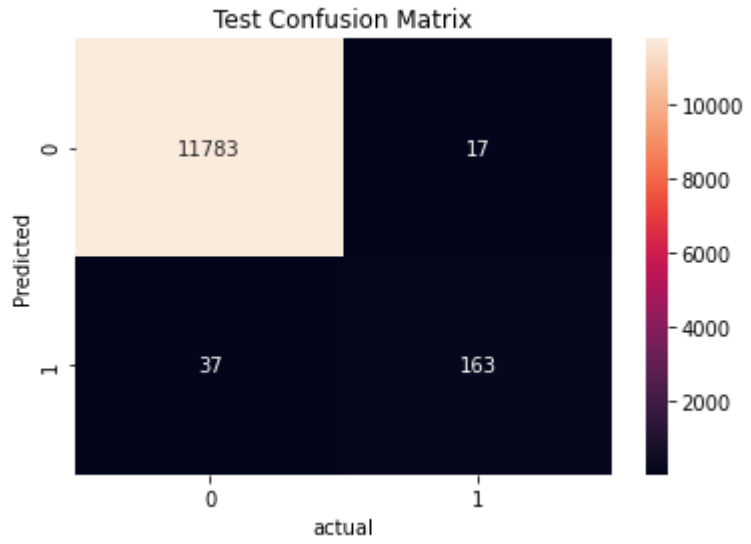
Train Accuray with Hyper Parameter (max_depth=5, n_estimators=1000) is 100 %

Test Accuray with Hyper Parameter (max_depth=5, n_estimators=1000) is 91 %

In [43]:
```
best_XGBoostClass_t = find_best_threshold(train_XGBoostClass_thresholds, train_XG
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





Observation

- XGBoostClassifier is a sensible model due to TNR and TPR rates are high when compared with FPR, FNR. However, there are fewer points which are wrongly classfied.

In [44]:
```
F1_Score_XGBoostClass_Train=f1_score(y_train,y_train_XGBoostClass_pred)
F1_Score_XGBoostClass_Test=f1_score(y_test,y_test_XGBoostClass_pred)
print('Train f1 score',f1_score(y_train,y_train_XGBoostClass_pred))
print('Test f1 score',f1_score(y_test,y_test_XGBoostClass_pred))
```

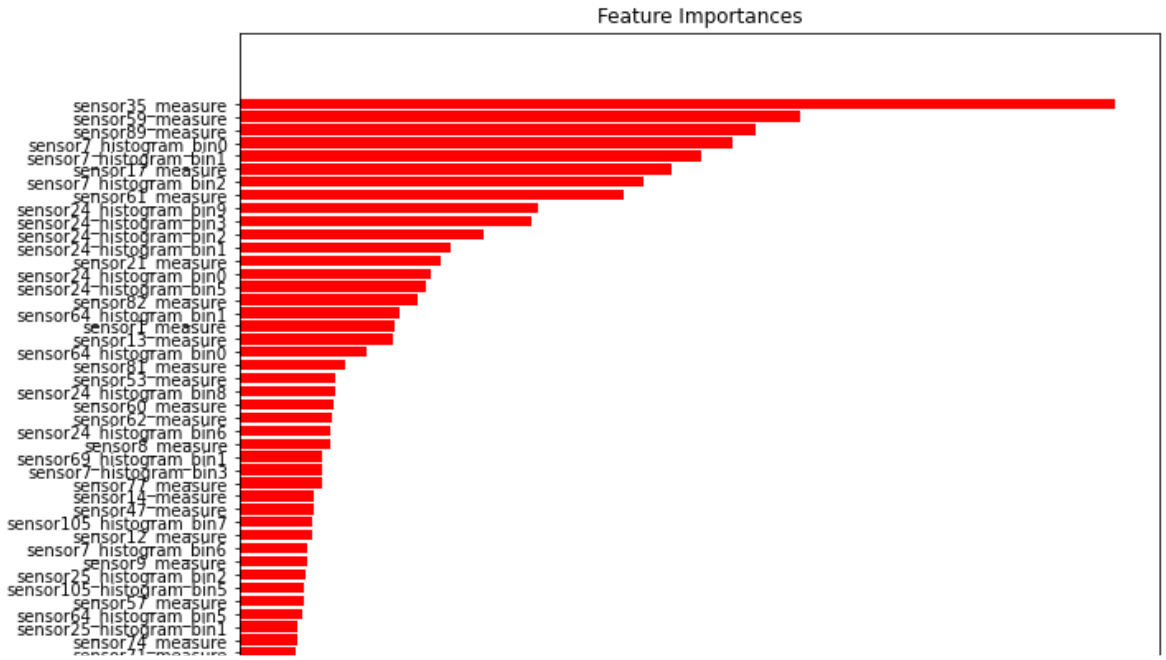
```
Train f1 score 1.0
Test f1 score 0.8578947368421053
```

In [45]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_XGBoostClass_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_XGBoostClass_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00       800

    accuracy                           1.00     48000
   macro avg       1.00      1.00      1.00     48000
weighted avg       1.00      1.00      1.00     48000

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.91      0.81      0.86       200

    accuracy                           1.00     12000
   macro avg       0.95      0.91      0.93     12000
weighted avg       1.00      1.00      1.00     12000
```

Feature Importance

In [46]:
```
XGBoost_Train_features = X_train.columns
XGBoost_importances = XGBoostClass_Final.feature_importances_
XGBoost_indices = (np.argsort(XGBoost_importances))[-100:]
plt.figure(figsize=(10,16))
plt.title('Feature Importances')
plt.barh(range(len(XGBoost_indices)), XGBoost_importances[XGBoost_indices], color
plt.yticks(range(len(XGBoost_indices)), [XGBoost_Train_features[i] for i in XGBoo
plt.xlabel('Relative Importance')
plt.show()
```



4.1.5 Approach 1: Conclusion

In [1]:
```
from prettytable import PrettyTable
import math

Finaloutput = PrettyTable()
Finaloutput1 = PrettyTable()

Finaloutput.field_names = ["Model", "Train AUC Score", "Test AUC Score","Train F1

Finaloutput.add_row(["Logistic Regression",89,90,51,49,50])
Finaloutput.add_row(["Random Forest",100,87,100,82,91])
Finaloutput.add_row(["LGBMClassifier",100,91,100,84,92])
Finaloutput.add_row(["XGBoost Classifier",100,91,100,86,93])
print(Finaloutput)
```

```
+---------------------+-----------------+---------------+---------------+----
----------+---------------------+
|        Model        | Train AUC Score | Test AUC Score | Train F1 Score | Tes
t F1 Score | Test F1 Macro Score |
+---------------------+-----------------+---------------+---------------+----
----------+---------------------+
| Logistic Regression |        89       |       90      |       51      |
49      |          50         |
|    Random Forest    |       100       |       87      |      100      |
82      |          91         |
|    LGBMClassifier   |       100       |       91      |      100      |
84      |          92         |
|  XGBoost Classifier |       100       |       91      |      100      |
86      |          93         |
+---------------------+-----------------+---------------+---------------+----
----------+---------------------+
```

Observations:

LGBMClassifier and XGBoostClassifer provided best F1 Macro score when compared with other ML models. However, XGBoostClassifer has better AUC score and F1 scores when compared with LGBMClassifer. Hence XGBoostClassifer is best ML model

4.2 Approach 2

Lets apply the following functinoalities in the dataset and will proceed with applying ML models to predict the performance metric

- Remove unused features (>70% of data contains null values)
- Impute median for the missing values
- Apply Feature Engineering Techniques
- Remove one of the Highly Correlated features
- Apply Normlization on the dataset
- Apply ML Models

Since we have already done the first two funtionalities on the dataset in preprocessing section, lets proceed with remaining functionalities

Apply Feature Engineering

Adding mean, Median, Std as a new features to the dataset

In [48]:
```
X_before_Normalization = EquipFail_train_dataset.copy()
```

In [49]:
```
#Lets add mean, median, standard deviation  to the dataset
X_before_Normalization['mean'] = X_before_Normalization[X_before_Normalization.cc
X_before_Normalization['median'] = X_before_Normalization[X_before_Normalization.
#X_before_Normalization['std'] = X_before_Normalization[X_before_Normalization.cc
X_before_Normalization
```

Out[49]:

	target	sensor1_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_m
0	0.0	76698.0	2.130706e+09	280.0	0.0	
1	0.0	33058.0	0.000000e+00	126.0	0.0	
2	0.0	41040.0	2.280000e+02	100.0	0.0	
3	0.0	12.0	7.000000e+01	66.0	0.0	
4	0.0	60874.0	1.368000e+03	458.0	0.0	
...	...	...	...	...	...	
59995	0.0	153002.0	6.640000e+02	186.0	0.0	
59996	0.0	2286.0	2.130707e+09	224.0	0.0	
59997	0.0	112.0	2.130706e+09	18.0	0.0	
59998	0.0	80292.0	2.130706e+09	494.0	0.0	
59999	0.0	40222.0	6.980000e+02	628.0	0.0	

60000 rows × 164 columns

In [50]:
```
X_before_Normalization.head()
```

Out[50]:

	target	sensor1_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measu
0	0.0	76698.0	2.130706e+09	280.0	0.0	(
1	0.0	33058.0	0.000000e+00	126.0	0.0	(
2	0.0	41040.0	2.280000e+02	100.0	0.0	(
3	0.0	12.0	7.000000e+01	66.0	0.0	1(
4	0.0	60874.0	1.368000e+03	458.0	0.0	(

5 rows × 164 columns

Consider Highly Correlated features and add new features

In [51]:
```
X_before_Normalization["sensor32_measure'-'sensor8_measure"]=X_before_Normalizati
X_before_Normalization["sensor27_measure'-'sensor67_measure"]=X_before_Normalizat
X_before_Normalization["sensor27_measure'-'sensor47_measure"]=X_before_Normalizat
X_before_Normalization["sensor27_measure'-'sensor46_measure"]=X_before_Normalizat
X_before_Normalization["sensor92_measure'-'sensor93_measure"]=X_before_Normalizat
X_before_Normalization["sensor14_measure'-'sensor15_measure"]=X_before_Normalizat
X_before_Normalization["sensor72_measure'-'sensor78_measure"]=X_before_Normalizat
X_before_Normalization["sensor95_measure'-'sensor94_measure"]=X_before_Normalizat
X_before_Normalization["sensor12_measure'-'sensor13_measure"]=X_before_Normalizat
X_before_Normalization["sensor89_measure'-'sensor33_measure"]=X_before_Normalizat
X_before_Normalization["sensor90_measure'-'sensor91_measure"]=X_before_Normalizat
X_before_Normalization["sensor27_measure'-'sensor14_measure"]=X_before_Normalizat
X_before_Normalization["sensor14_measure'-'sensor46_measure"]=X_before_Normalizat
X_before_Normalization["sensor14_measure'-'sensor67_measure"]=X_before_Normalizat
X_before_Normalization["sensor14_measure'-'sensor47_measure"]=X_before_Normalizat
X_before_Normalization["sensor32_measure'-'sensor14_measure"]=X_before_Normalizat
X_before_Normalization["sensor8_measure'-'sensor14_measure"]=X_before_Normalizati
X_before_Normalization["sensor97_measure'-'sensor96_measure"]=X_before_Normalizat
X_before_Normalization["sensor32_measure'-'sensor33_measure"]=X_before_Normalizat
X_before_Normalization["sensor33_measure'-'sensor8_measure"]=X_before_Normalizati
X_before_Normalization["sensor1_measure'-'sensor45_measure"]=X_before_Normalizati
X_before_Normalization["sensor89_measure'-'sensor35_measure"]=X_before_Normalizat
X_before_Normalization["sensor14_measure'-'sensor33_measure"]=X_before_Normalizat
X_before_Normalization["sensor33_measure'-'sensor27_measure"]=X_before_Normalizat
X_before_Normalization["sensor8_measure'-'sensor27_measure"]=X_before_Normalizati
X_before_Normalization["sensor33_measure'-'sensor17_measure"]=X_before_Normalizat
X_before_Normalization["sensor15_measure'-'sensor27_measure"]=X_before_Normalizat
X_before_Normalization["sensor32_measure'-'sensor27_measure"]=X_before_Normalizat
X_before_Normalization["sensor67_measure'-'sensor33_measure"]=X_before_Normalizat
X_before_Normalization["sensor33_measure'-'sensor47_measure"]=X_before_Normalizat
X_before_Normalization["sensor46_measure'-'sensor33_measure"]=X_before_Normalizat
X_before_Normalization["sensor8_measure'-'sensor46_measure"]=X_before_Normalizati
X_before_Normalization["sensor8_measure'-'sensor67_measure"]=X_before_Normalizati
X_before_Normalization["sensor8_measure'-'sensor47_measure"]=X_before_Normalizati
X_before_Normalization["sensor46_measure'-'sensor15_measure"]=X_before_Normalizat
X_before_Normalization["sensor15_measure'-'sensor67_measure"]=X_before_Normalizat
X_before_Normalization["sensor47_measure'-'sensor15_measure"]=X_before_Normalizat
X_before_Normalization["sensor46_measure'-'sensor89_measure"]=X_before_Normalizat
X_before_Normalization["sensor67_measure'-'sensor89_measure"]=X_before_Normalizat
X_before_Normalization["sensor47_measure'-'sensor89_measure"]=X_before_Normalizat
X_before_Normalization["sensor46_measure'-'sensor32_measure"]=X_before_Normalizat
X_before_Normalization["sensor67_measure'-'sensor32_measure"]=X_before_Normalizat
X_before_Normalization["sensor47_measure'-'sensor32_measure"]=X_before_Normalizat
X_before_Normalization["sensor34_measure'-'sensor16_measure"]=X_before_Normalizat
X_before_Normalization["sensor32_measure'-'sensor15_measure"]=X_before_Normalizat
X_before_Normalization["sensor27_measure'-'sensor89_measure"]=X_before_Normalizat
X_before_Normalization["sensor8_measure'-'sensor15_measure"]=X_before_Normalizati
X_before_Normalization["sensor6_measure'-'sensor5_measure"]=X_before_Normalizatio
X_before_Normalization["sensor35_measure'-'sensor33_measure"]=X_before_Normalizat
X_before_Normalization["sensor35_measure'-'sensor17_measure"]=X_before_Normalizat
```

Based on the Approach 1, listed out top 8 least features from all ML MOdels and will remove such features since these features doesn't useful for ML modelling

In [52]:
```
#rwemove least features

X_before_Normalization.drop(['sensor101_measure','sensor106_measure','sensor107_m
```

Lets find out highly correlated features each other and will keep one feature and remove another feature to predict the model better.

In [54]:
```
#https://www.dezyre.com/recipes/drop-out-highly-correlated-features-in-python

# Creating correlation matrix
cor_matrix = X_before_Normalization.corr().abs()


# Selecting upper triangle of correlation matrix
upper_tri = cor_matrix.where(np.triu(np.ones(cor_matrix.shape),
                                     k=1).astype(np.bool))


# Finding index of feature columns with correlation greater than 0.95
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.95
print(); print(to_drop)

#n.drop(X_before_Normalization.columns[to_drop], axis=1)
#print(); print(df1.head())
```

```
['sensor13_measure', 'sensor14_measure', 'sensor15_measure', 'sensor26_histogra
m_bin3', 'sensor26_histogram_bin4', 'sensor27_measure', 'sensor32_measure', 'se
nsor33_measure', 'sensor35_measure', 'sensor45_measure', 'sensor46_measure', 's
ensor47_measure', 'sensor53_measure', 'sensor56_measure', 'sensor59_measure',
'sensor64_histogram_bin5', 'sensor65_measure', 'sensor67_measure', 'sensor89_me
asure', 'sensor95_measure', 'sensor105_histogram_bin4', "sensor27_measure'-'sen
sor47_measure", "sensor27_measure'-'sensor46_measure", "sensor92_measure'-'sens
or93_measure", "sensor95_measure'-'sensor94_measure", "sensor12_measure'-'senso
r13_measure", "sensor90_measure'-'sensor91_measure", "sensor27_measure'-'sensor
14_measure", "sensor14_measure'-'sensor46_measure", "sensor14_measure'-'sensor6
7_measure", "sensor14_measure'-'sensor47_measure", "sensor32_measure'-'sensor14
_measure", "sensor8_measure'-'sensor14_measure", "sensor97_measure'-'sensor96_m
easure", "sensor32_measure'-'sensor33_measure", "sensor33_measure'-'sensor8_mea
sure", "sensor89_measure'-'sensor35_measure", "sensor14_measure'-'sensor33_meas
ure", "sensor33_measure'-'sensor27_measure", "sensor8_measure'-'sensor27_measur
e", "sensor33_measure'-'sensor17_measure", "sensor15_measure'-'sensor27_measur
e", "sensor32_measure'-'sensor27_measure", "sensor67_measure'-'sensor33_measur
e", "sensor33_measure'-'sensor47_measure", "sensor46_measure'-'sensor33_measur
e", "sensor8_measure'-'sensor46_measure", "sensor8_measure'-'sensor67_measure",
"sensor8_measure'-'sensor47_measure", "sensor46_measure'-'sensor15_measure", "s
ensor15_measure'-'sensor67_measure", "sensor47_measure'-'sensor15_measure", "se
nsor46_measure'-'sensor89_measure", "sensor67_measure'-'sensor89_measure", "sen
sor47_measure'-'sensor89_measure", "sensor46_measure'-'sensor32_measure", "sens
or67_measure'-'sensor32_measure", "sensor47_measure'-'sensor32_measure", "senso
r34_measure'-'sensor16_measure", "sensor27_measure'-'sensor89_measure", "sensor
8_measure'-'sensor15_measure", "sensor35_measure'-'sensor33_measure"]
```

Above are the highly correlated features. Lets remove such features

In [55]:
```
# Droping Marked Features

X_before_Normalization.drop(['sensor13_measure', 'sensor14_measure', 'sensor15_me
```

Splitting data into Train and cross validation(or test): Stratified Sampling

In [56]:
```
# extract data column project_is_approved from total_project_data and add it to v
# remove project_is_approved from total_project_data and store the data in to var
y = X_before_Normalization['target'].values
X_before_Normalization.drop(['target'], axis=1, inplace=True)
X = X_before_Normalization
X.head(1)
```

Out[56]:

	sensor1_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sen
0	76698.0	2.130706e+09	280.0	0.0	0.0	

1 rows × 148 columns

In [57]:
```
#split the data in to train and cross validation and test before performing BOW,
# train test split
from sklearn.model_selection import train_test_split
#splitting data in to train and test with 20 percentage as test data
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.2, stra
#splitting train data in to train and cv with 33 percentage as cv data
```

Apply Normalization

In [58]:
```
#normalize dataframe https://stackoverflow.com/questions/26414913/normalize-colum

#X_norm = EquipFail_train_dataset.copy() #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler().fit(X_train1)
X_train1 = pd.DataFrame(min_max_scaler.transform(X_train1),columns=X_train1.colu
X_test1 = pd.DataFrame(min_max_scaler.transform(X_test1),columns=X_test1.columns)
#X_normalized = min_max_scaler.fit_transform(X_norm)
#X_norm_final = pd.DataFrame(min_max_scaler.fit_transform(X_norm),columns=X_norm.

#save the variables and export pickle
#https://www.codegrepper.com/code-examples/prolog/how+to+create+pickle+file+in+py

import pickle
filename= "Train_norm.pkl"
with open(filename,'wb') as Norm:
    pickle.dump(min_max_scaler,Norm)

X_train1
```

Out[58]:

	sensor1_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure
0	0.394478	2.909833e-07	0.000647	0.0	0.0
1	0.014155	4.421068e-07	0.004619	0.0	0.0
2	0.000226	9.999998e-01	0.000092	0.0	0.0
3	0.024482	9.999998e-01	0.003438	0.0	0.0
4	0.016437	1.013748e-07	0.000965	0.0	0.0
...	...	...	...	...	...
47995	0.000711	7.133783e-08	0.000647	0.0	0.0
47996	0.013839	3.510572e-07	0.003212	0.0	0.0
47997	0.000307	1.220252e-08	0.000133	0.0	0.0
47998	0.170086	7.133783e-08	0.000647	0.0	0.0
47999	0.018058	1.342278e-07	0.001334	0.0	0.0

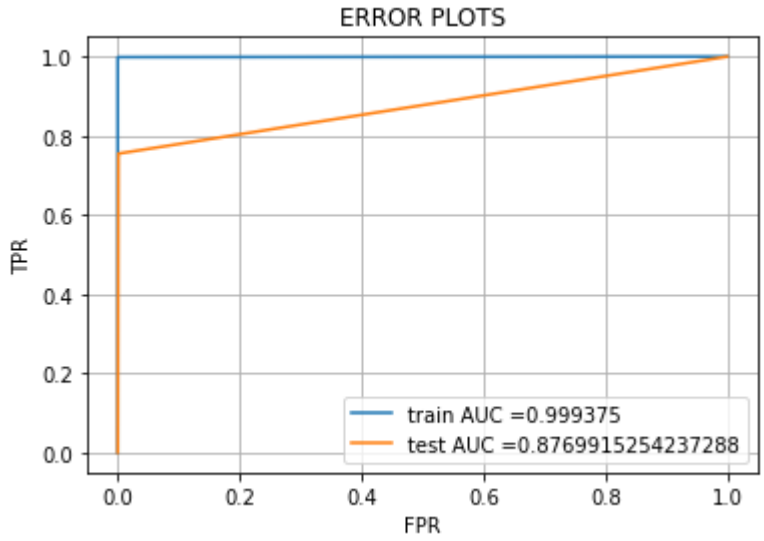
48000 rows × 148 columns

In [59]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

4.2.1 Random Forest With No Sampling Techique

Apply ML Model with best parameters

In [60]:
```
RFClass_Final = RFClass_Final = RandomForestClassifier(max_depth=200,n_estimators
RFClass_Final.fit(X_train, y_train)

y_train_RFClass_pred = RFClass_Final.predict(X_train)
y_test_RFClass_pred = RFClass_Final.predict(X_test)

train_RFClass_fpr, train_RFClass_tpr, train_RFClass_thresholds = roc_curve(y_trai
test_RFClass_fpr, test_RFClass_tpr, test_RFClass_thresholds = roc_curve(y_test, y

plt.plot(train_RFClass_fpr, train_RFClass_tpr, label="train AUC ="+str(auc(train_
plt.plot(test_RFClass_fpr, test_RFClass_tpr, label="test AUC ="+str(auc(test_RFCl
AUC_RFClass_Train = str(auc(train_RFClass_fpr, train_RFClass_tpr))
AUC_RFClass_Test = str(auc(test_RFClass_fpr, test_RFClass_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Observations:

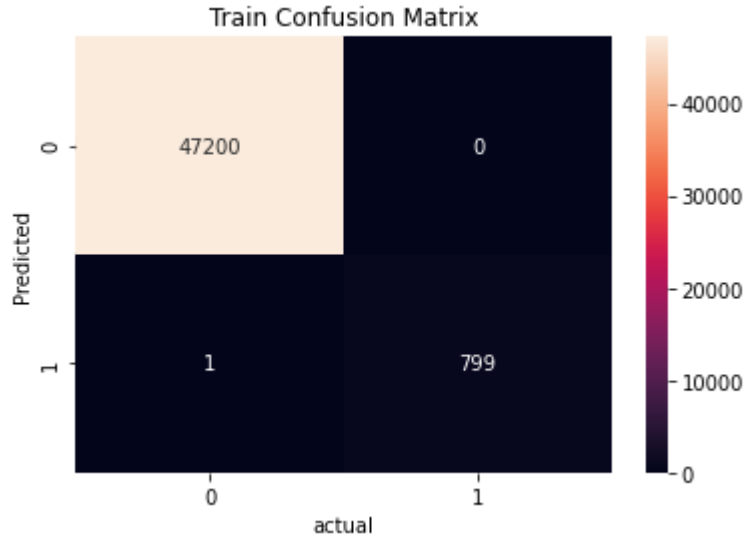
Train Accuray with Hyper Parameter ({'max_depth': 200, 'n_estimators': 1000}) is 100 %

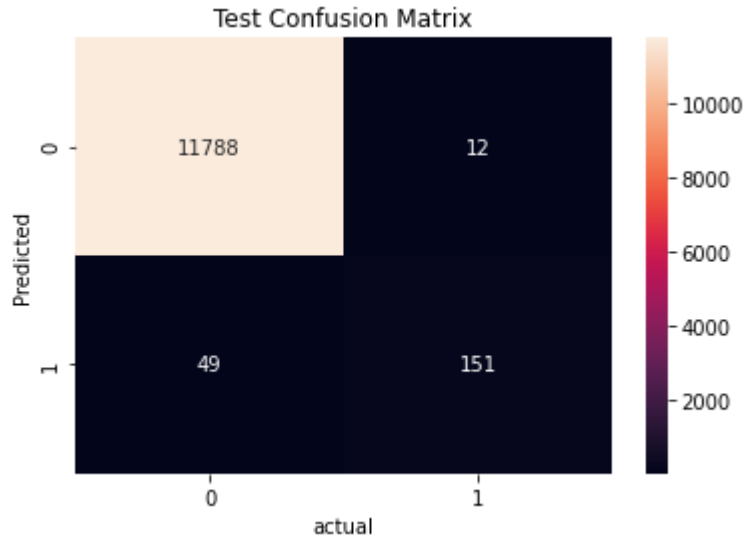
Test Accuray with Hyper Parameter ({'max_depth': 200, 'n_estimators': 1000}) is 88 %

Plot ROC and Confusion Matrix

In [61]:
```
best_RFClass_t = find_best_threshold(train_RFClass_thresholds, train_RFClass_fpr,
RFClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_train_RF
df_cm1=pd.DataFrame(RFClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

#heatmap confusion matrix https://stackoverflow.com/questions/35572000/how-can-i-
RFClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_RFCla
df_cm1=pd.DataFrame(RFClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.99875 for threshold 1.0
```





Observations:

Random Forest is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [62]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_RFClass_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_RFClass_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00       800

    accuracy                           1.00     48000
   macro avg       1.00      1.00      1.00     48000
weighted avg       1.00      1.00      1.00     48000

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.93      0.76      0.83       200

    accuracy                           0.99     12000
   macro avg       0.96      0.88      0.91     12000
weighted avg       0.99      0.99      0.99     12000
```

In [63]:
```
F1_Score_RFClass_Train=f1_score(y_train,y_train_RFClass_pred)
F1_Score_RFClass_Test=f1_score(y_test,y_test_RFClass_pred)
print('Train f1 score',f1_score(y_train,y_train_RFClass_pred))
print('Test f1 score',f1_score(y_test,y_test_RFClass_pred))
```

```
Train f1 score 0.9993746091307067
Test f1 score 0.8319559228650139
```

Feature Importance

In [64]:
```
RF_Train_features = X_train.columns
RF_importances = RFClass_Final.feature_importances_
RF_indices = (np.argsort(RF_importances))[-100:]
plt.figure(figsize=(10,16))
plt.title('Feature Importances')
plt.barh(range(len(RF_indices)), RF_importances[RF_indices], color='r', align='ce
plt.yticks(range(len(RF_indices)), [RF_Train_features[i] for i in RF_indices])
plt.xlabel('Relative Importance')
plt.show()
```

4.2.2 RandomForest with RandomOverSampler

In [65]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [66]:
```
#randomoversampler sklearn example https://imbalanced-learn.readthedocs.io/en/sta
#Perform Resampling only on Train data
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_train, y_train = ros.fit_resample(X_train, y_train)
from collections import Counter
print(sorted(Counter(y_train).items()))
```

```
[(0.0, 47200), (1.0, 47200)]
```

In [67]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47200
47200
```

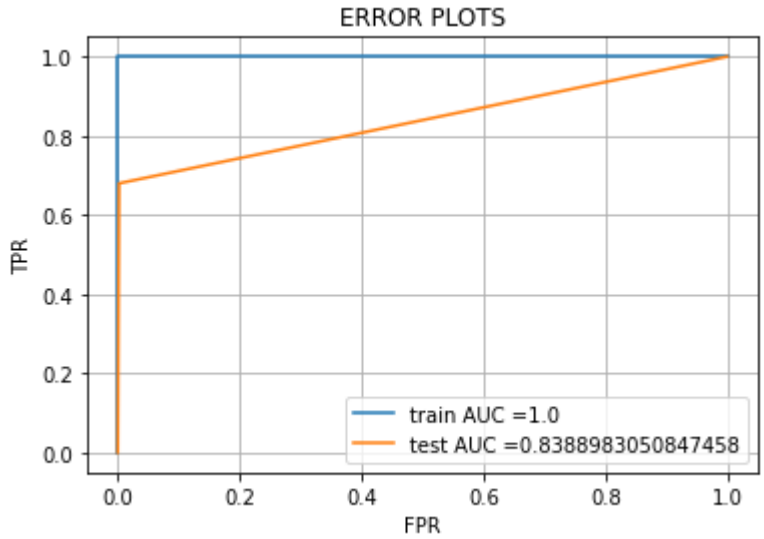
Apply ML Model with best parameters

In [68]:
```
from sklearn.ensemble import RandomForestClassifier

RFClass_Final = RandomForestClassifier(n_estimators=1000,n_jobs=-1)
#class_weight='balanced',max_depth=BestRFClassParam.get('max_depth'),n_estimators
RFClass_Final.fit(X_train, y_train)

y_train_RFClass_pred = RFClass_Final.predict(X_train)
y_test_RFClass_pred = RFClass_Final.predict(X_test)


#RFClass_Final = RandomForestClassifier(n_estimators=1000,n_jobs=-1)
#0.80
```

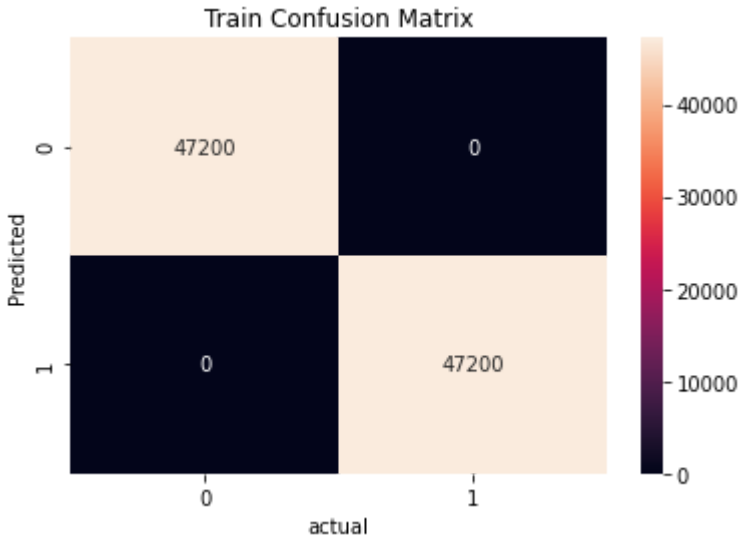
Plot AUC, Confusion Matrix and F1 Scores

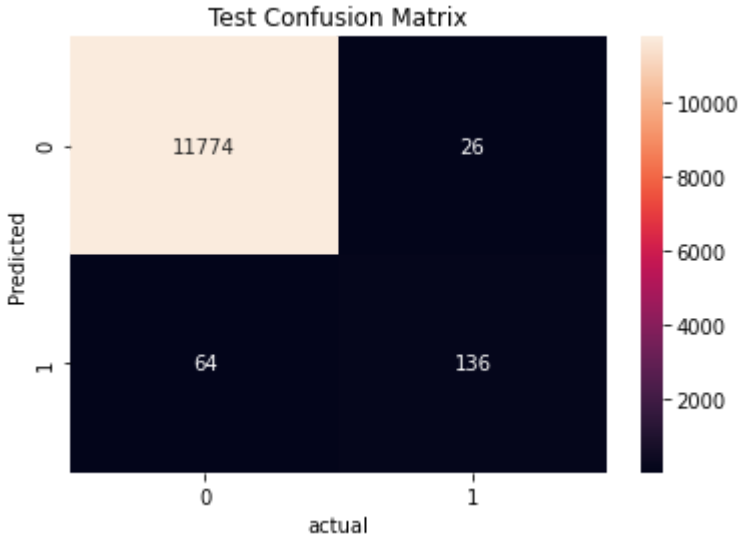
In [69]:

```
train_RFClass_fpr, train_RFClass_tpr, train_RFClass_thresholds = roc_curve(y_trai
test_RFClass_fpr, test_RFClass_tpr, test_RFClass_thresholds = roc_curve(y_test, y

plt.plot(train_RFClass_fpr, train_RFClass_tpr, label="train AUC ="+str(auc(train_
plt.plot(test_RFClass_fpr, test_RFClass_tpr, label="test AUC ="+str(auc(test_RFCl
AUC_RFClass_Train = str(auc(train_RFClass_fpr, train_RFClass_tpr))
AUC_RFClass_Test = str(auc(test_RFClass_fpr, test_RFClass_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
best_RFClass_t = find_best_threshold(train_RFClass_thresholds, train_RFClass_fpr,
RFClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_train_RF
df_cm1=pd.DataFrame(RFClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

#heatmap confusion matrix https://stackoverflow.com/questions/35572000/how-can-i-
RFClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_RFCla
df_cm1=pd.DataFrame(RFClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





Observations:

Train Accuray with Hyper Parameter (n_estimators=1000) is 100 %

Test Accuray with Hyper Parameter (n_estimators=1000) is 84 %

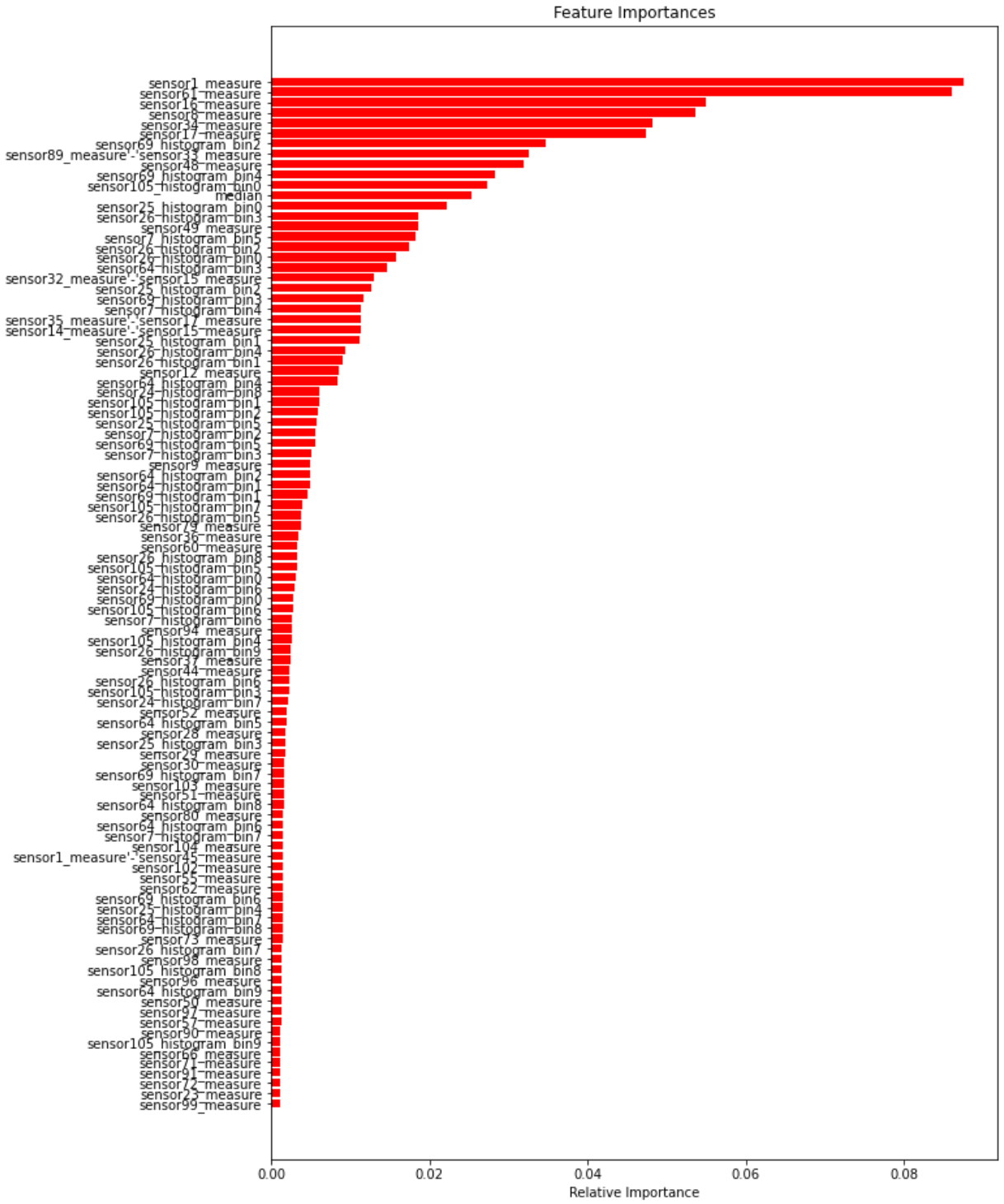
Random Forest is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [70]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_RFClass_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_RFClass_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00     47200

    accuracy                           1.00     94400
   macro avg       1.00      1.00      1.00     94400
weighted avg       1.00      1.00      1.00     94400

Test Classification Report
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00     11800
         1.0       0.84      0.68      0.75       200

    accuracy                           0.99     12000
   macro avg       0.92      0.84      0.87     12000
weighted avg       0.99      0.99      0.99     12000
```

Feature Importance

In [71]:
```
RF_Train_features = X_train.columns
RF_importances = RFClass_Final.feature_importances_
RF_indices = (np.argsort(RF_importances))[-100:]
plt.figure(figsize=(10,16))
plt.title('Feature Importances')
plt.barh(range(len(RF_indices)), RF_importances[RF_indices], color='r', align='ce
plt.yticks(range(len(RF_indices)), [RF_Train_features[i] for i in RF_indices])
plt.xlabel('Relative Importance')
plt.show()
```



4.2.3 LIGHT GBM Classifer with No Sampling

Apply ML Model with best parameters

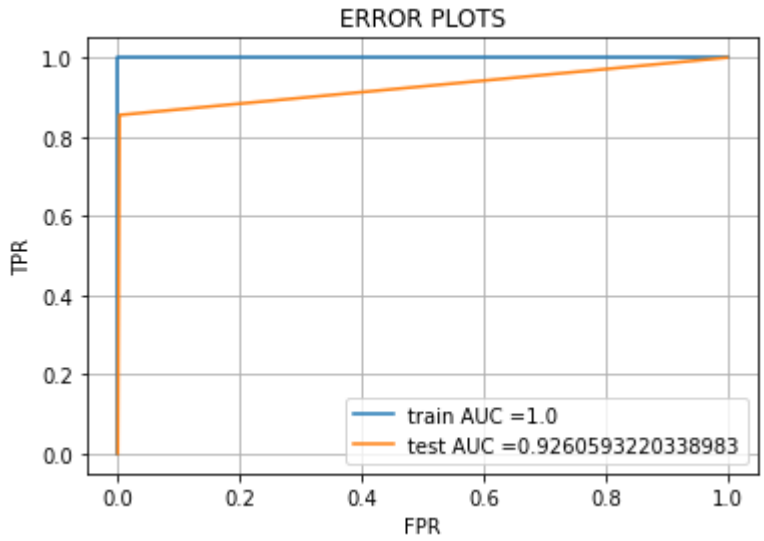
In [72]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [73]:
```
import lightgbm as lgb
clf1=lgb.LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=25,
               n_estimators=200, num_leaves=900, objective='binary',
               random_state=0, scale_pos_weight=59, silent=False,reg_lambda=0.28)
clf1.fit(X_train, y_train)
```

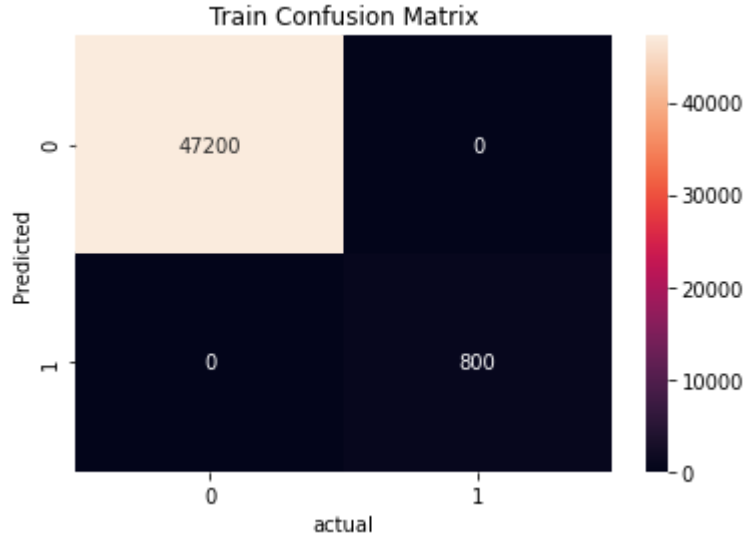
```
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignore
d. Current value: boosting=gbdt
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignore
d. Current value: boosting=gbdt
[LightGBM] [Info] Number of positive: 800, number of negative: 47200
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.067390 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 35046
[LightGBM] [Info] Number of data points in the train set: 48000, number of u
sed features: 146
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

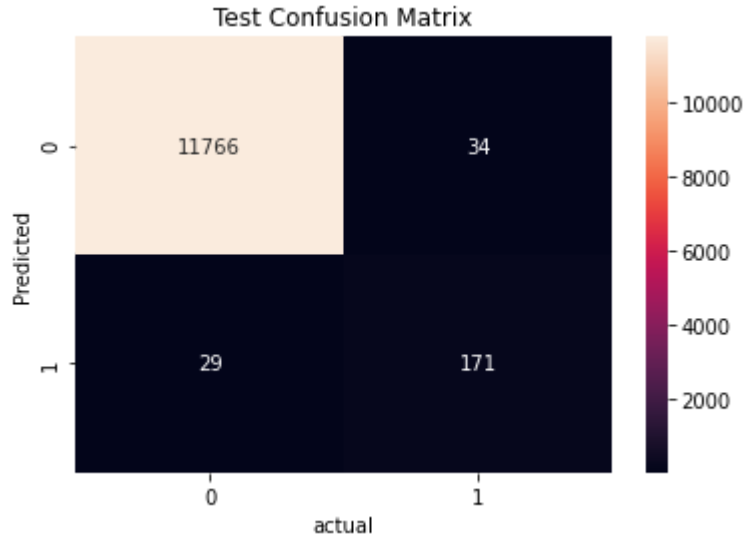
Plot AUC, Confusion Matrix and F1 Scores

In [74]:
```
y_train_pred = clf1.predict(X_train)
y_test_pred = clf1.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





```
Train f1 score 1.0
Test f1 score 0.8444444444444444
```

Observations:

Train Accuray with Hyper Parameter (max_depth=25, n_estimators=200) is 100 %

Test Accuray with Hyper Parameter (max_depth=25, n_estimators=200) is 93 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

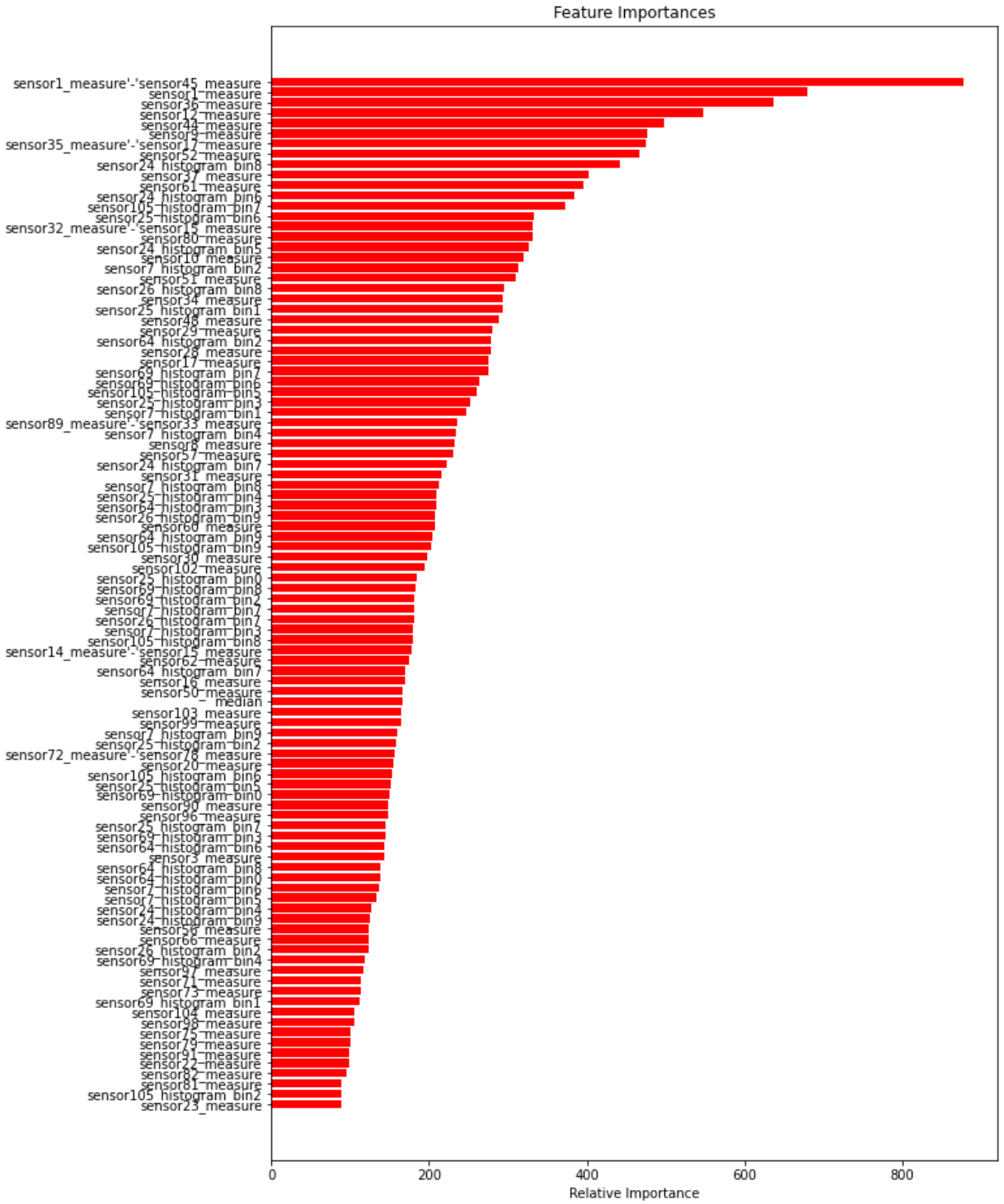
F1 Score improved compared with previous ML models

In [75]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00       800

    accuracy                           1.00     48000
   macro avg       1.00      1.00      1.00     48000
weighted avg       1.00      1.00      1.00     48000

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.83      0.85      0.84       200

    accuracy                           0.99     12000
   macro avg       0.92      0.93      0.92     12000
weighted avg       0.99      0.99      0.99     12000
```

Feature Importance

In [76]:
```
RF_Train_features = X_train.columns
RF_importances = clf1.feature_importances_
RF_indices = (np.argsort(RF_importances))[-100:]
plt.figure(figsize=(10,16))
plt.title('Feature Importances')
plt.barh(range(len(RF_indices)), RF_importances[RF_indices], color='r', align='ce
plt.yticks(range(len(RF_indices)), [RF_Train_features[i] for i in RF_indices])
plt.xlabel('Relative Importance')
plt.show()
```



4.2.4 LIGHT GBM Classifer with RandomOverSampler

In [77]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [78]:
```
#randomoversampler sklearn example https://imbalanced-learn.readthedocs.io/en/sta
#Perform Resampling only on Train data
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_train, y_train = ros.fit_resample(X_train, y_train)
from collections import Counter
print(sorted(Counter(y_train).items()))
```

```
[(0.0, 47200), (1.0, 47200)]
```

In [79]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47200
47200
```

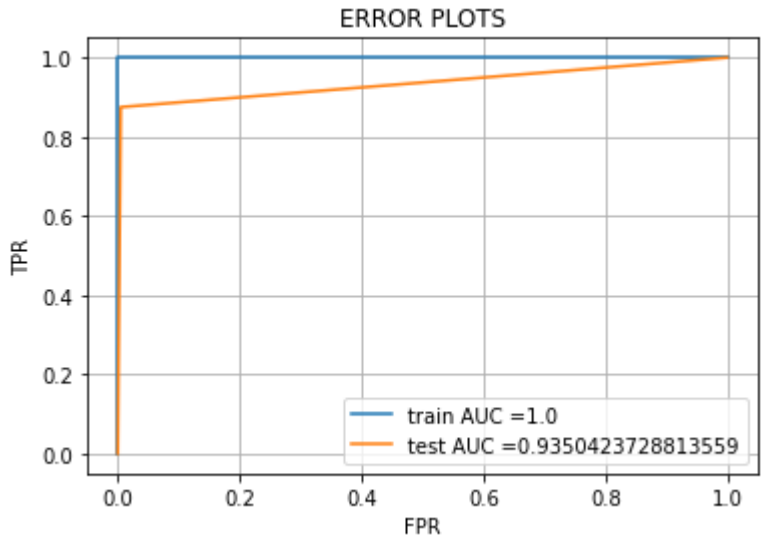
Apply ML Model with best parameters

In [80]:
```
import lightgbm as lgb
clf1=lgb.LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=25,
               n_estimators=200, num_leaves=900, objective='binary',
               random_state=0, scale_pos_weight=59, silent=False,reg_lambda=0.28)
clf1.fit(X_train, y_train)
```

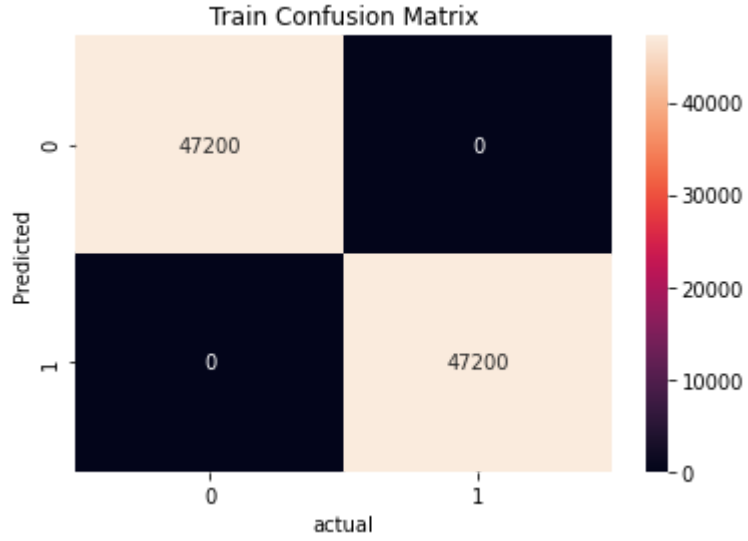
```
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignore
d. Current value: boosting=gbdt
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignore
d. Current value: boosting=gbdt
[LightGBM] [Info] Number of positive: 47200, number of negative: 47200
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.136794 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 34827
[LightGBM] [Info] Number of data points in the train set: 94400, number of u
sed features: 147
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

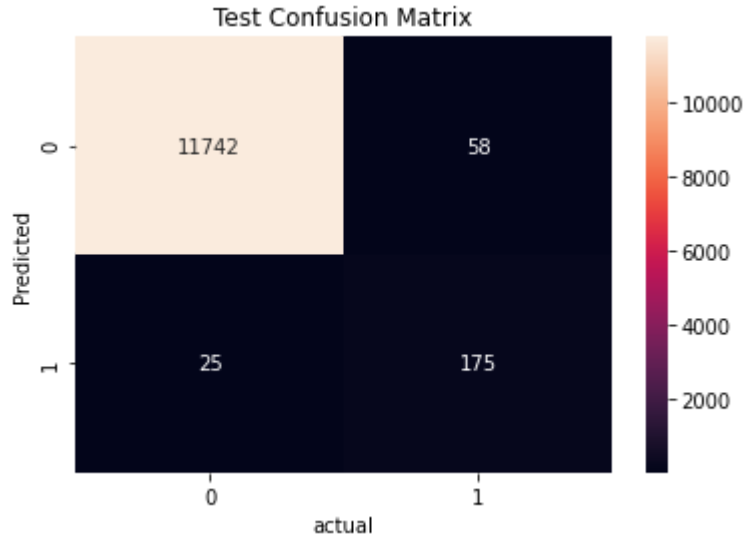
Plot AUC, Confusion Matrix and F1 Scores

In [81]:
```
y_train_pred1 = clf1.predict(X_train)
y_test_pred1 = clf1.predict(X_test)



train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred1)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred1)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





Observations:

Train Accuray with Hyper Parameter (max_depth=25, n_estimators=200) is 100 %

Test Accuray with Hyper Parameter (max_depth=25, n_estimators=200) is 94 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [82]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred1))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred1))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00     47200

    accuracy                           1.00     94400
   macro avg       1.00      1.00      1.00     94400
weighted avg       1.00      1.00      1.00     94400

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.75      0.88      0.81       200

    accuracy                           0.99     12000
   macro avg       0.87      0.94      0.90     12000
weighted avg       0.99      0.99      0.99     12000
```

4.2.5 LIGHT GBM Classifer with SMOTE

In [83]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [84]:
```
#https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-mi
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 0)
X_train, y_train = sm.fit_sample(X_train, y_train)

print('After OverSampling, the shape of train_X: {}'.format(X_train.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train == 0)))
```

```
After OverSampling, the shape of train_X: (94400, 148)
After OverSampling, the shape of train_y: (94400,)

After OverSampling, counts of label '1': 47200
After OverSampling, counts of label '0': 47200
```

In [85]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47200
47200
```

Apply ML Model with best parameters

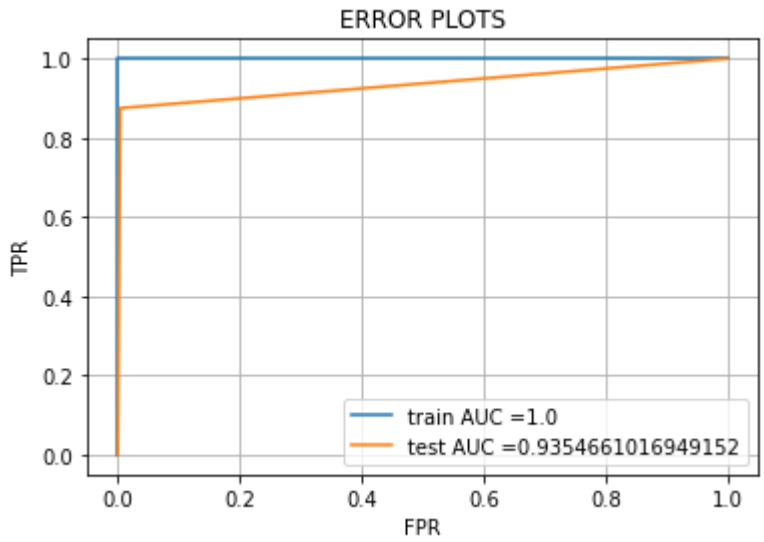
In [86]:
```
import lightgbm as lgb
clf1 = lgb.LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=10
              n_estimators=200, num_leaves=300, objective='binary',
               random_state=0, reg_lambda=0.28)
clf1.fit(X_train, y_train)
```

```
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignored.
Current value: boosting=gbdt
```

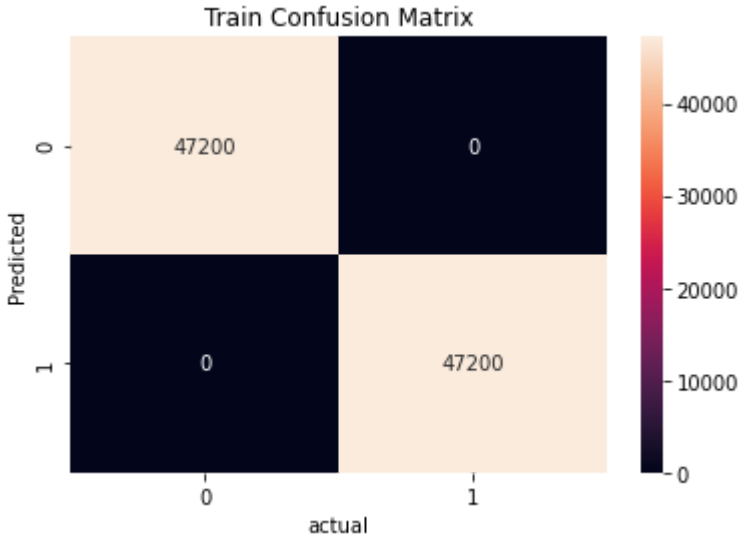
Out[86]:
```
LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=10,
               n_estimators=200, num_leaves=300, objective='binary',
               random_state=0, reg_lambda=0.28)
```

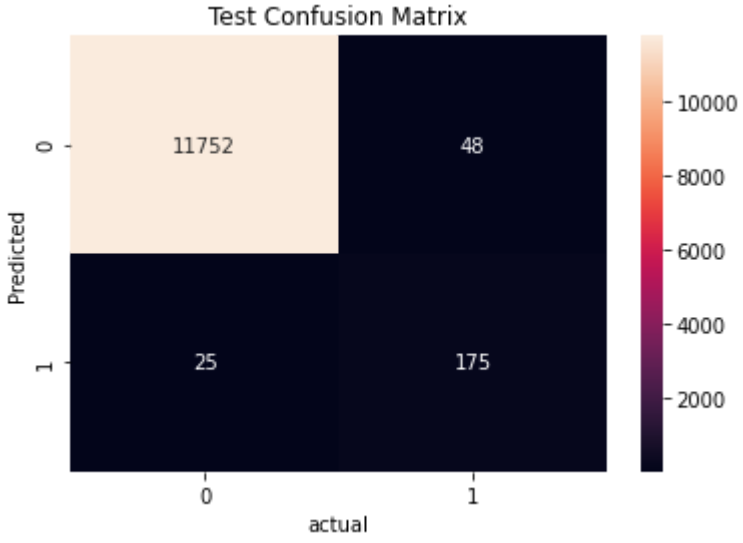
Plot AUC, Confusion Matrix and F1 Scores

In [87]:
```
y_train_pred1 = clf1.predict(X_train)
y_test_pred1 = clf1.predict(X_test)


train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred1)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred1)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```



```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





Observations:

Train Accuray with Hyper Parameter (max_depth=10, n_estimators=200) is 100 %

Test Accuray with Hyper Parameter (max_depth=10, n_estimators=200) is 94 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [88]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred1))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred1))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00     47200

    accuracy                           1.00     94400
   macro avg       1.00      1.00      1.00     94400
weighted avg       1.00      1.00      1.00     94400

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.78      0.88      0.83       200

    accuracy                           0.99     12000
   macro avg       0.89      0.94      0.91     12000
weighted avg       0.99      0.99      0.99     12000
```

In [ ]:
```
```

4.2.6 LIGHT GBM Classifer with SMOTE TOMEK

In [89]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [90]:
```
#https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-mi
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import TomekLinks
sm = SMOTETomek(random_state = 0)
X_train, y_train = sm.fit_sample(X_train, y_train)

print('After OverSampling, the shape of train_X: {}'.format(X_train.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train == 0)))
```

```
After OverSampling, the shape of train_X: (94394, 148)
After OverSampling, the shape of train_y: (94394,)

After OverSampling, counts of label '1': 47197
After OverSampling, counts of label '0': 47197
```

In [91]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47197
47197
```

Apply ML Model with best parameters

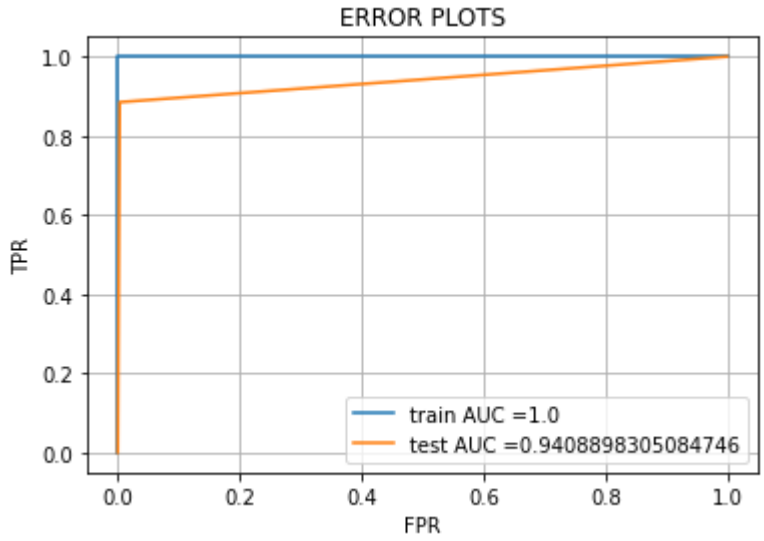
In [92]:
```
import lightgbm as lgb
clf1 = lgb.LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=10
              n_estimators=200, num_leaves=300, objective='binary',
               random_state=0)
clf1.fit(X_train, y_train)

#https://www.codegrepper.com/code-examples/prolog/how+to+create+pickle+file+in+py
#save the variables and export pickle
import pickle
filename= "Trained_Model.pkl"
with open(filename,'wb') as XGBoostFile:
    pickle.dump(clf1,XGBoostFile)
```

```
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignored.
Current value: boosting=gbdt
```

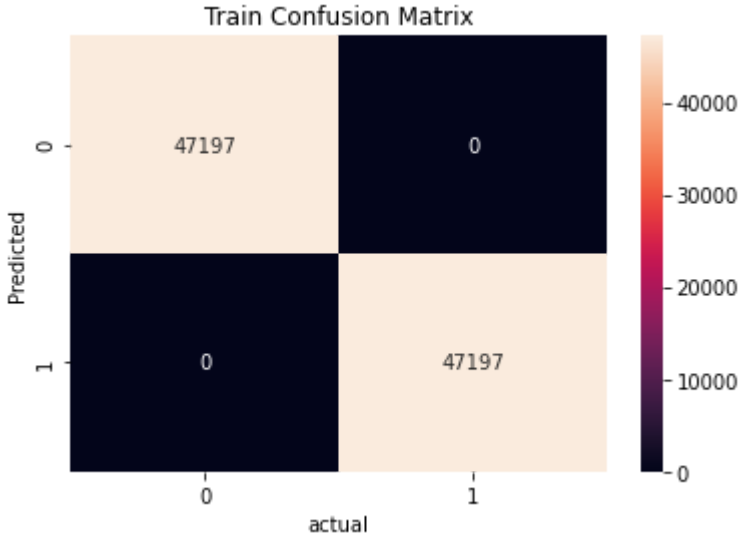
Out[92]:
```
LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=10,
               n_estimators=200, num_leaves=300, objective='binary',
               random_state=0)
```

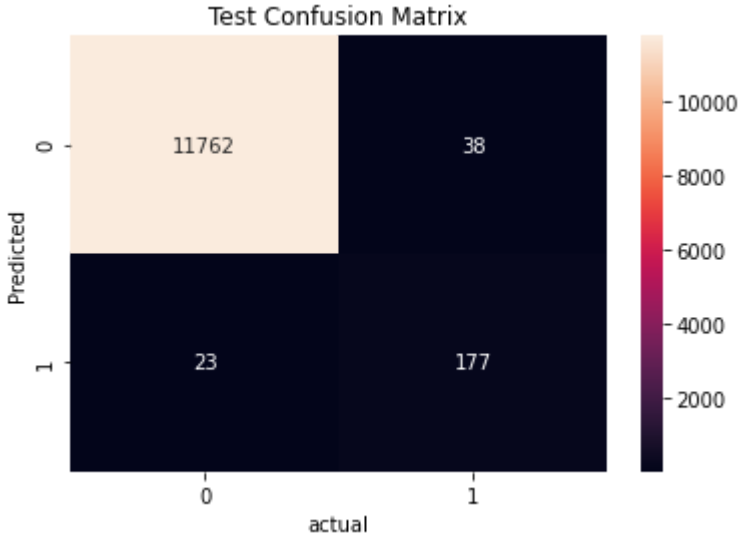
Plot AUC, Confusion Matrix and F1 Scores

In [93]:
```
y_train_pred1 = clf1.predict(X_train)
y_test_pred1 = clf1.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred1)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred1)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```



```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





Observations:

Train Accuray with Hyper Parameter (max_depth=10, n_estimators=200) is 100 %

Test Accuray with Hyper Parameter (max_depth=10, n_estimators=200) is 94 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [94]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred1))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred1))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47197
         1.0       1.00      1.00      1.00     47197

    accuracy                           1.00     94394
   macro avg       1.00      1.00      1.00     94394
weighted avg       1.00      1.00      1.00     94394

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.82      0.89      0.85       200

    accuracy                           0.99     12000
   macro avg       0.91      0.94      0.93     12000
weighted avg       1.00      0.99      1.00     12000
```

4.2.7 XGBOOST With No Sampling

Apply ML Model with best parameters

In [95]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [96]:
```
from xgboost import XGBClassifier
XGClassifier_final = XGBClassifier(learning_rate =0.1,n_estimators=1000,max_depth
```

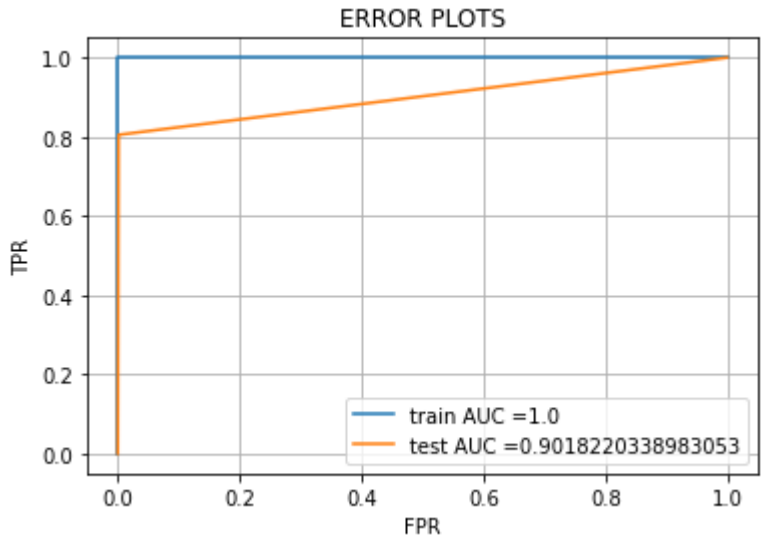
In [97]:
```
XGClassifier_final.fit(X_train, y_train )
                       #eval_metric=eval_metric, eval_set=eval_set,
                       #verbose=True)
```

```
c:\program files\python36\lib\site-packages\xgboost\sklearn.py:892: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[16:59:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

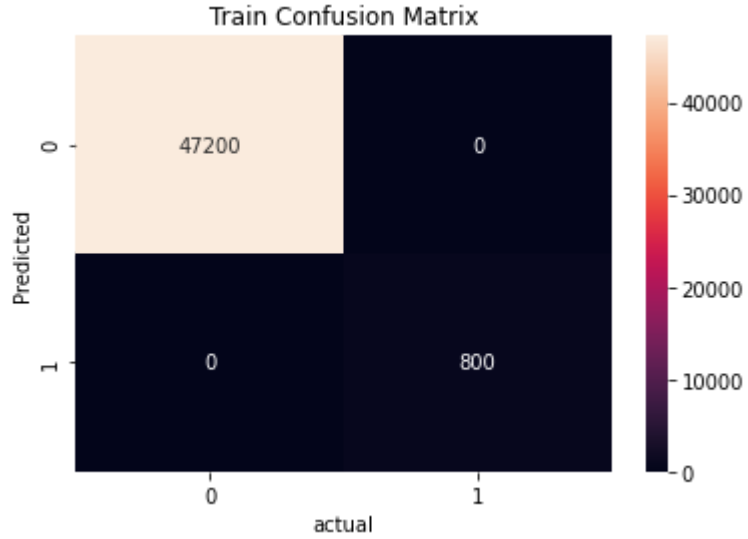
Out[97]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=1000, n_jobs=4, nthread=4, num_parallel_tree=1,
              random_state=27, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              seed=27, subsample=0.8, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

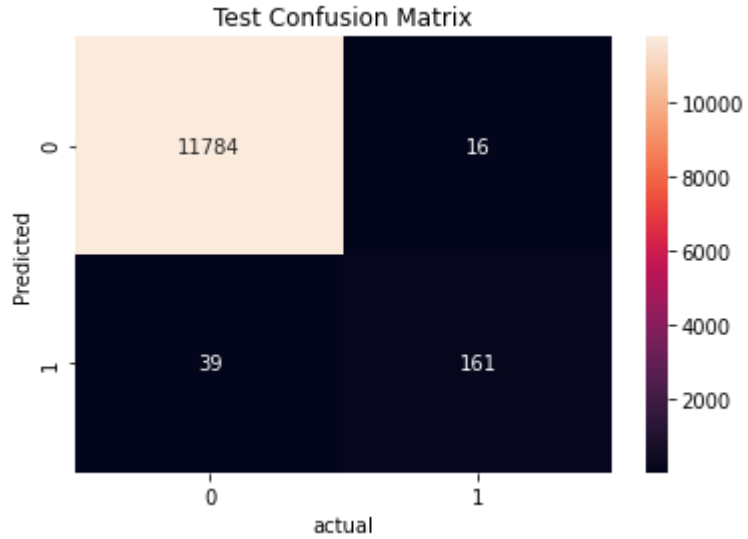
Plot AUC, Confusion Matrix and F1 Scores

In [98]:
```
y_train_pred = XGClassifier_final.predict(X_train)
y_test_pred = XGClassifier_final.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```






```
Train f1 score 1.0
Test f1 score 0.8541114058355438
```

Observations:

Train Accuray with Hyper Parameter (max_depth=5, n_estimators=1000) is 100 %

Test Accuray with Hyper Parameter (max_depth=5, n_estimators=1000) is 90 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

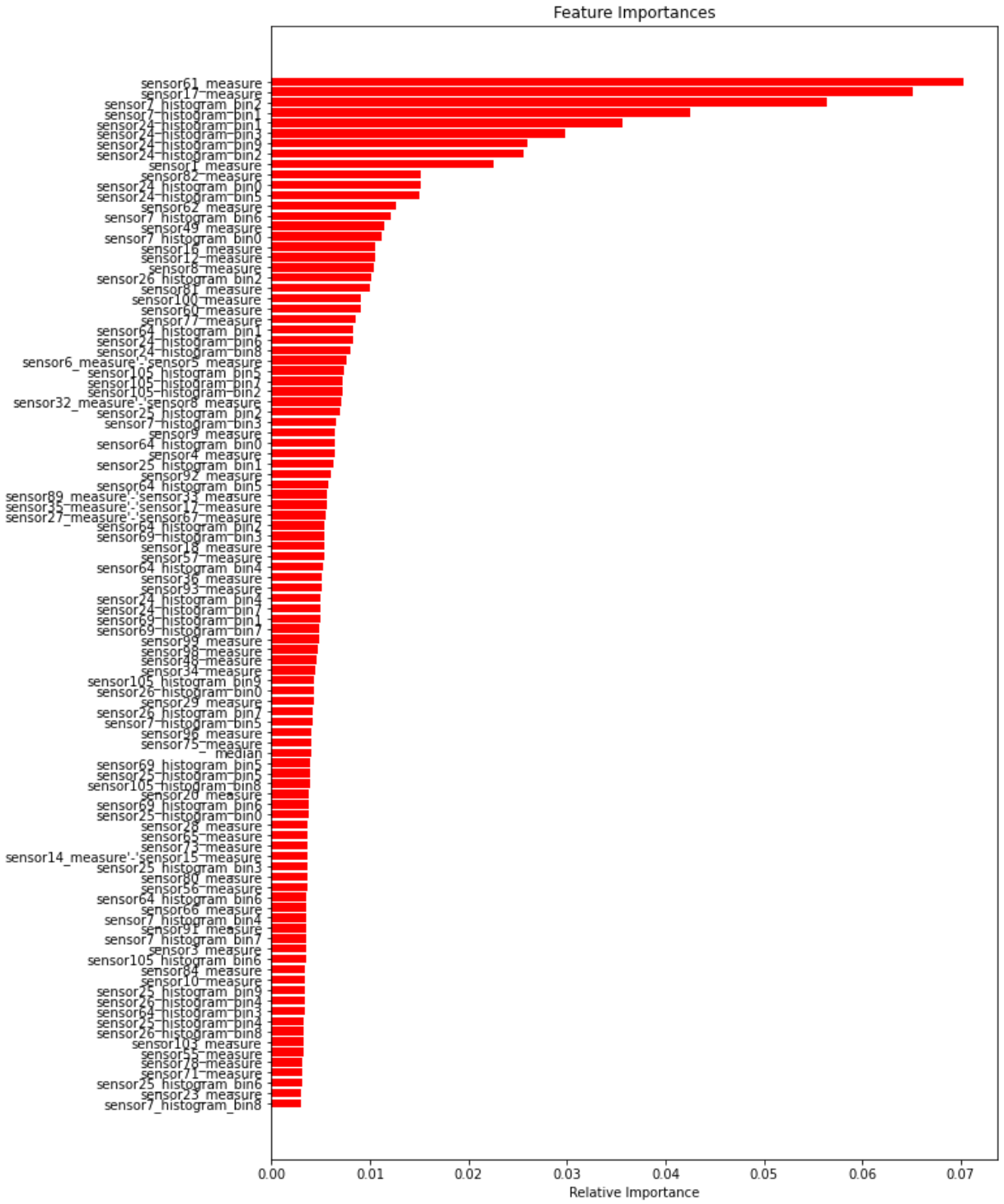
F1 Score improved compared with previous ML models

In [99]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00       800

    accuracy                           1.00     48000
   macro avg       1.00      1.00      1.00     48000
weighted avg       1.00      1.00      1.00     48000

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.91      0.81      0.85       200

    accuracy                           1.00     12000
   macro avg       0.95      0.90      0.93     12000
weighted avg       1.00      1.00      1.00     12000
```

Feature Importance

In [100]:
```
RF_Train_features = X_train.columns
RF_importances = XGClassifier_final.feature_importances_
RF_indices = (np.argsort(RF_importances))[-100:]
plt.figure(figsize=(10,16))
plt.title('Feature Importances')
plt.barh(range(len(RF_indices)), RF_importances[RF_indices], color='r', align='ce
plt.yticks(range(len(RF_indices)), [RF_Train_features[i] for i in RF_indices])
plt.xlabel('Relative Importance')
plt.show()
```



In [101]:
```
# Feature importance

#lightGBM model fit
#gbm = lgb.LGBMClassifier()
#gbm.fit(train, target)
XGClassifier_final.feature_importances_

# importance of each attribute
fea_imp_ = pd.DataFrame({'cols':X_train.columns, 'fea_imp':XGClassifier_final.fea
fea_imp_.loc[fea_imp_.fea_imp <0.004].sort_values(by=['cols'], ascending = True)
```

Out[101]:

	cols	fea_imp
125	sensor102_measure	0.001910
126	sensor103_measure	0.003231
127	sensor104_measure	0.002426
128	sensor105_histogram_bin0	0.002827
129	sensor105_histogram_bin1	0.002427
...	...	...
114	sensor84_measure	0.003443
115	sensor90_measure	0.002915
116	sensor91_measure	0.003544
119	sensor94_measure	0.002219
121	sensor97_measure	0.001400

82 rows × 2 columns

In [ ]:
```
```

4.2.8 XGBOOST with RandomOverSampler

In [102]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [103]:
```
#randomoversampler sklearn example https://imbalanced-learn.readthedocs.io/en/sta
#Perform Resampling only on Train data
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_train, y_train = ros.fit_resample(X_train, y_train)
from collections import Counter
print(sorted(Counter(y_train).items()))
```

```
[(0.0, 47200), (1.0, 47200)]
```

In [104]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47200
47200
```

Apply ML Model with best parameters

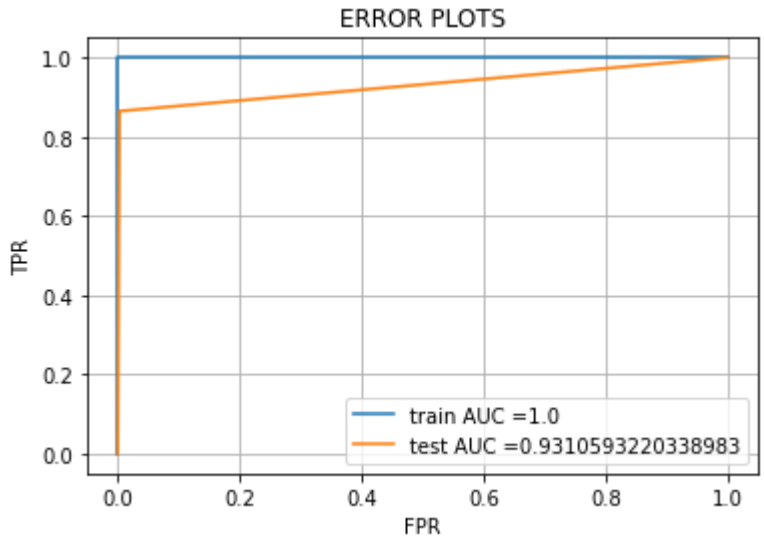
In [105]:
```
from xgboost import XGBClassifier
XGClassifier_final = XGBClassifier(n_estimators=1000,max_depth=7,
                                   objective= 'binary:logistic',n_jobs=-1)


XGClassifier_final.fit(X_train, y_train)
```

```
c:\program files\python36\lib\site-packages\xgboost\sklearn.py:892: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[17:01:47] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

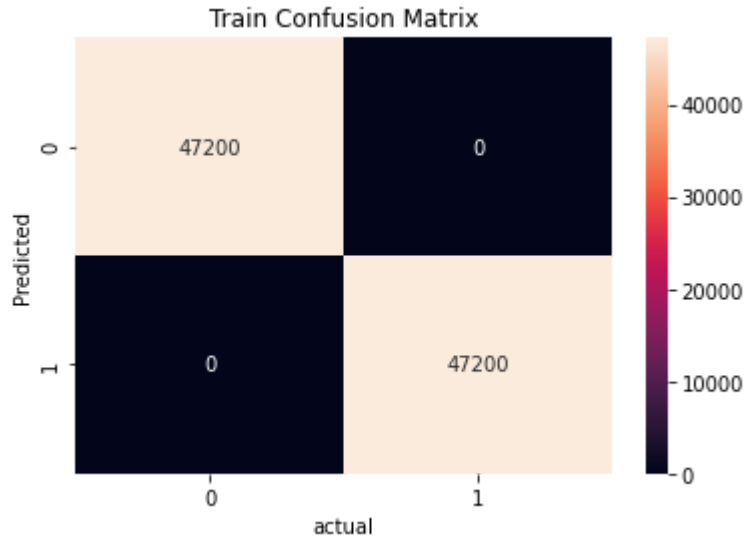
Out[105]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=1000, n_jobs=-1, num_parallel_tree=1, random_state=
0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

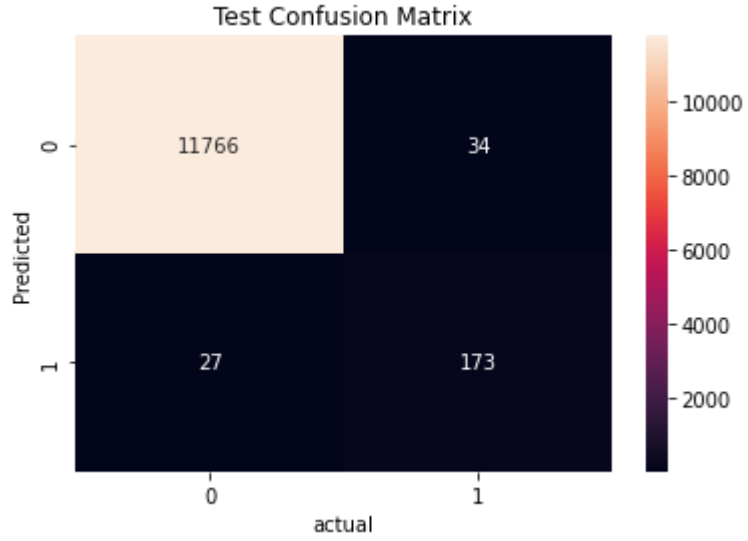
Plot AUC, Confusion Matrix and F1 Scores

In [106]:
```
y_train_pred = XGClassifier_final.predict(X_train)
y_test_pred = XGClassifier_final.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```






```
Train f1 score 1.0
Test f1 score 0.8501228501228502
```

Observations:

Train Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 100 %

Test Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 93 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [107]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00     47200

    accuracy                           1.00     94400
   macro avg       1.00      1.00      1.00     94400
weighted avg       1.00      1.00      1.00     94400

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.84      0.86      0.85       200

    accuracy                           0.99     12000
   macro avg       0.92      0.93      0.92     12000
weighted avg       1.00      0.99      0.99     12000
```

4.2.9 XGBoost with SMOTE

In [108]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [109]:
```
#https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-mi
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 0)
X_train, y_train = sm.fit_sample(X_train, y_train)

print('After OverSampling, the shape of train_X: {}'.format(X_train.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train == 0)))
```

```
After OverSampling, the shape of train_X: (94400, 148)
After OverSampling, the shape of train_y: (94400,)

After OverSampling, counts of label '1': 47200
After OverSampling, counts of label '0': 47200
```

In [110]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47200
47200
```

Apply ML Model with best parameters

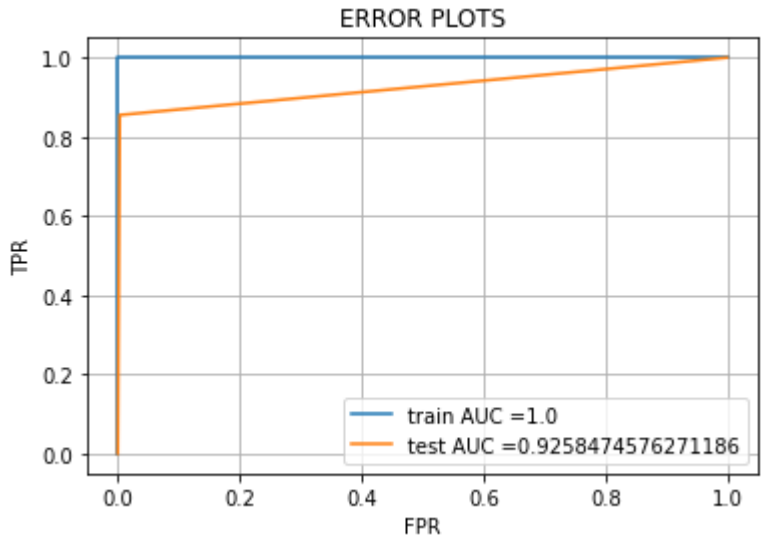
In [111]:
```
from xgboost import XGBClassifier
XGClassifier_final = XGBClassifier(n_estimators=1000,max_depth=7,
                                   objective= 'binary:logistic',n_jobs=-1,reg_lambda


XGClassifier_final.fit(X_train, y_train)
```

```
c:\program files\python36\lib\site-packages\xgboost\sklearn.py:892: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[17:04:23] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

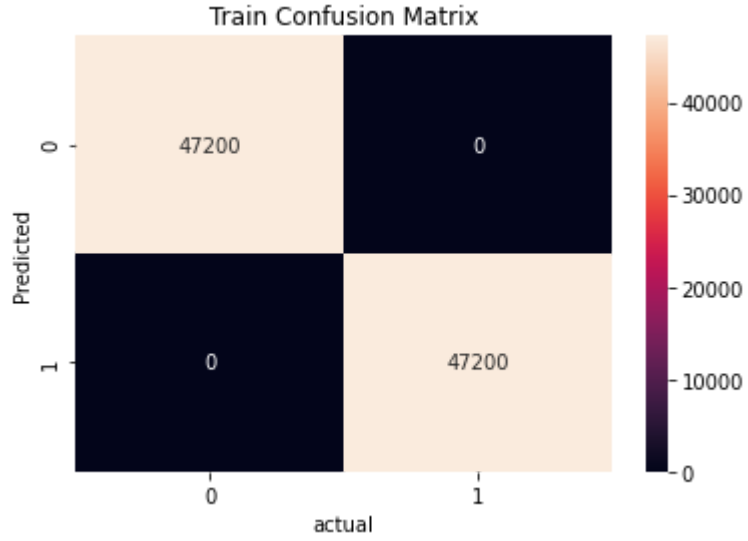
Out[111]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=1000, n_jobs=-1, num_parallel_tree=1, random_state=
0,
              reg_alpha=0, reg_lambda=2.5, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

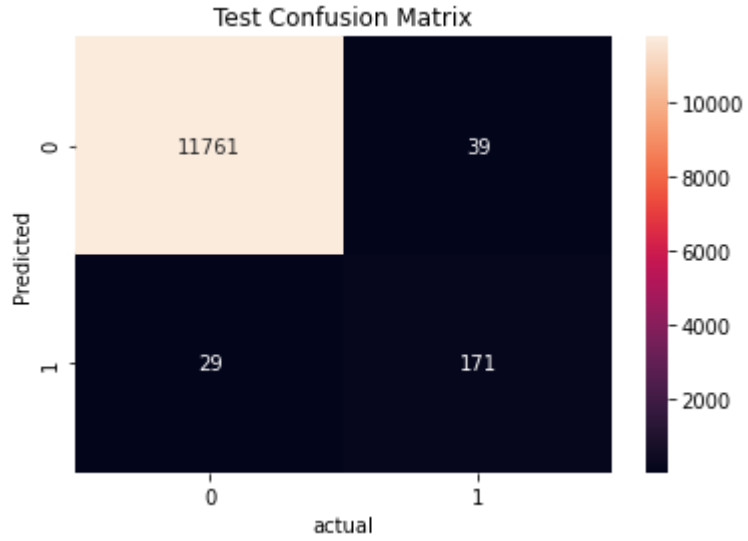
Plot AUC, Confusion Matrix and F1 Scores

In [112]:
```
y_train_pred = XGClassifier_final.predict(X_train)
y_test_pred = XGClassifier_final.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```






```
Train f1 score 1.0
Test f1 score 0.8341463414634146
```

Observations:

Train Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 100 %

Test Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 93 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [113]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00     47200

    accuracy                           1.00     94400
   macro avg       1.00      1.00      1.00     94400
weighted avg       1.00      1.00      1.00     94400

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.81      0.85      0.83       200

    accuracy                           0.99     12000
   macro avg       0.91      0.93      0.92     12000
weighted avg       0.99      0.99      0.99     12000
```

4.2.10 XGBoost with SMOTE TOMEK

In [114]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [115]:
```
#https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-mi
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import TomekLinks
sm = SMOTETomek(random_state = 0)
X_train, y_train = sm.fit_sample(X_train, y_train)

print('After OverSampling, the shape of train_X: {}'.format(X_train.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train == 0)))
```

```
After OverSampling, the shape of train_X: (94394, 148)
After OverSampling, the shape of train_y: (94394,)

After OverSampling, counts of label '1': 47197
After OverSampling, counts of label '0': 47197
```

In [116]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47197
47197
```

Apply ML Model with best parameters

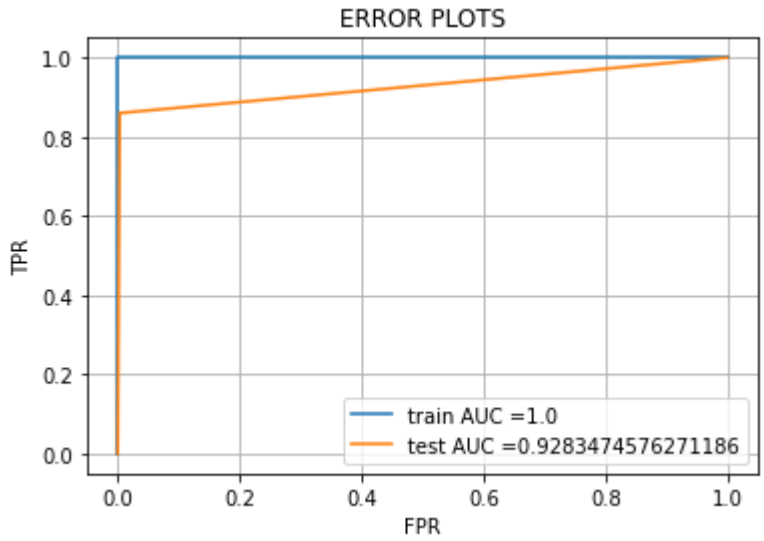
In [117]:
```
from xgboost import XGBClassifier
XGClassifier_final = XGBClassifier(n_estimators=1000,max_depth=7,
                                   objective= 'binary:logistic',n_jobs=-1,reg_alpha=


XGClassifier_final.fit(X_train, y_train)
```

```
c:\program files\python36\lib\site-packages\xgboost\sklearn.py:892: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[17:12:43] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

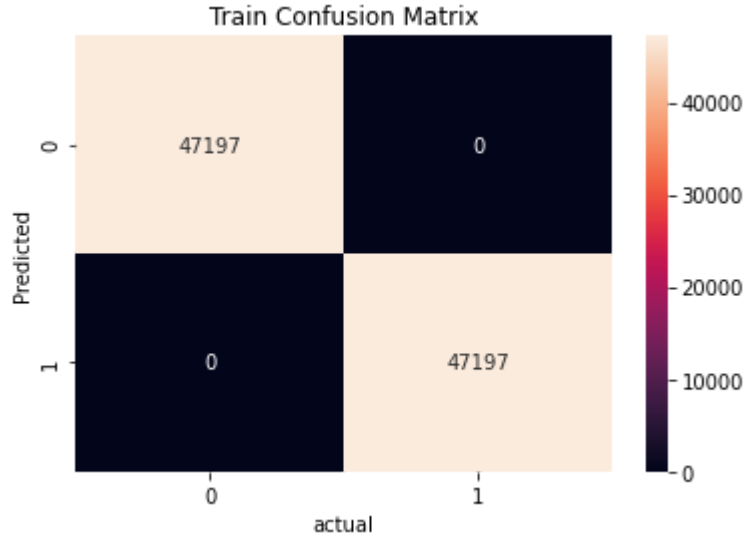
Out[117]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=1000, n_jobs=-1, num_parallel_tree=1, random_state=
0,
              reg_alpha=1e-05, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

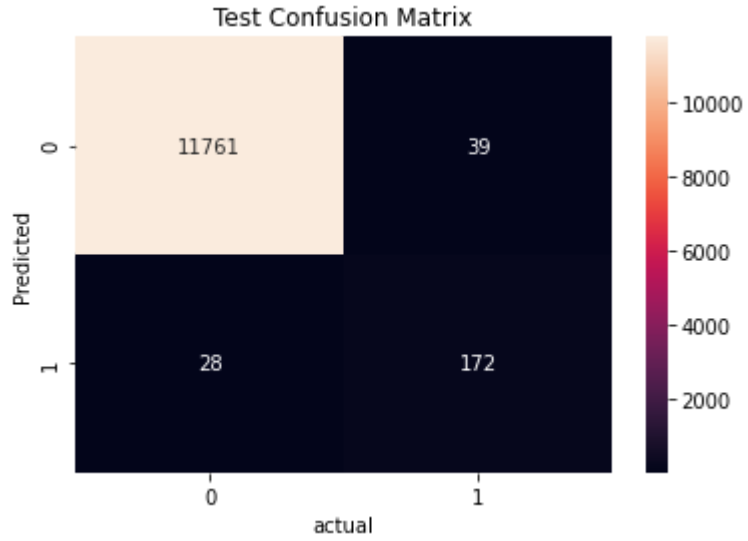
Plot AUC, Confusion Matrix and F1 Scores

In [118]:
```
y_train_pred = XGClassifier_final.predict(X_train)
y_test_pred = XGClassifier_final.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```






```
Train f1 score 1.0
Test f1 score 0.8369829683698298
```

Observations:

Train Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 100 %

Test Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 93 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [119]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47197
         1.0       1.00      1.00      1.00     47197

    accuracy                           1.00     94394
   macro avg       1.00      1.00      1.00     94394
weighted avg       1.00      1.00      1.00     94394

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.82      0.86      0.84       200

    accuracy                           0.99     12000
   macro avg       0.91      0.93      0.92     12000
weighted avg       0.99      0.99      0.99     12000
```

In [ ]:

In [ ]:

In [ ]:

4.2.11 Approach 2: Conclusion

In [2]:
```
from prettytable import PrettyTable
import math

Finaloutput = PrettyTable()
Finaloutput1 = PrettyTable()

Finaloutput.field_names = ["Model", "Sampling", "Train AUC Score", "Test AUC Scor

Finaloutput.add_row(["Random Forest","No Sampling",100,88,100,83,91])
Finaloutput.add_row(["Random Forest","ROS",100,84,99,75,87])
Finaloutput.add_row(["LGBMClassifier","No Sampling",100,93,100,84,92])
Finaloutput.add_row(["LGBMClassifier","ROS",100,94,100,81,90])
Finaloutput.add_row(["LGBMClassifier","SMOTE Over Sampling",100,94,100,83,91])
Finaloutput.add_row(["LGBMClassifier","SMOTE TOMEK Over Sampling",100,94,100,85,9
Finaloutput.add_row(["XGBoost Classifier","No Sampling",100,90,100,85,93])
Finaloutput.add_row(["XGBoost Classifier","ROS",100,93,100,85,92])
Finaloutput.add_row(["XGBoost Classifier","SMOTE Over Sampling",100,93,100,83,92]
Finaloutput.add_row(["XGBoost Classifier","SMOTE TOMEK Over Sampling",100,93,100,



print(Finaloutput)
```

```
+--------------------+---------------------------+-----------------+-----------
-----+----------------+---------------+---------------------+
|       Model        |          Sampling         | Train AUC Score | Test AUC S
core | Train F1 Score | Test F1 Score | Test F1 Macro Score |
+--------------------+---------------------------+-----------------+-----------
-----+----------------+---------------+---------------------+
|   Random Forest    |        No Sampling        |       100       |       88
|      100       |       83      |          91         |
|   Random Forest    |            ROS            |       100       |       84
|       99       |       75      |          87         |
|   LGBMClassifier   |        No Sampling        |       100       |       93
|      100       |       84      |          92         |
|   LGBMClassifier   |            ROS            |       100       |       94
|      100       |       81      |          90         |
|   LGBMClassifier   |    SMOTE Over Sampling    |       100       |       94
|      100       |       83      |          91         |
|   LGBMClassifier   | SMOTE TOMEK Over Sampling |       100       |       94
|      100       |       85      |          93         |
| XGBoost Classifier |        No Sampling        |       100       |       90
|      100       |       85      |          93         |
| XGBoost Classifier |            ROS            |       100       |       93
|      100       |       85      |          92         |
| XGBoost Classifier |    SMOTE Over Sampling    |       100       |       93
|      100       |       83      |          92         |
| XGBoost Classifier | SMOTE TOMEK Over Sampling |       100       |       93
|      100       |       84      |          92         |
+--------------------+---------------------------+-----------------+-----------
-----+----------------+---------------+---------------------+
```

Observations:

XGBOOST with No Sampling and LGBMClassifier with SMOTE TOMEK Over Sampling has provided a best F1 Scores when compared with other ML models.

LGBMClassifier with SMOTE TOMEK Over Sampling has provided best AUC scores.

4.3 Approach 3

Lets apply the following functionalities in the dataset and will proceed with applying ML models to predict the performance metric

```
1) Remove unused features (>70% of data contains null values)
2) Impute median for the missing values
3) Apply Feature Engineering Techniques
4) Drop Top 8 Least Features
5) Remove columns which as high Skewness >50
6) Remove once of the highly correlated features
7) Split the data in to train and test and apply following sampling
techniques to balance the Class Labels across Train and Test dataset.
    a) No Sampling with Normlization
    b) Random Over Sampler with Normlization
    c) SMOTE Oversampling  with Normlization
    d) SMOTE TOMEK with Normlization
8) Apply ML Models for all sampling techniques
```

Have already implemented the functionalities from point 1 to point 2 in Approach 1. Lets consider the same dataset and implement from point 3.

Apply Feature Engineering

Adding mean, Median, Std as a new features to the dataset

In [122]:
```
### Apply Feature Engineering

#Adding mean, Median, Std as a new features to the dataset
X_before_Normalization = EquipFail_train_dataset.copy()
```

Consider Highly Correlated features and add new features

In [123]:
```
#Lets add mean, median, standard deviation  to the dataset
X_before_Normalization['mean'] = X_before_Normalization[X_before_Normalization.co
X_before_Normalization['median'] = X_before_Normalization[X_before_Normalization.
#EquipFail_train_dataset['std'] = EquipFail_train_dataset[EquipFail_train_dataset
X_before_Normalization["sensor32_measure'-'sensor8_measure"]=X_before_Normalizati
X_before_Normalization["sensor27_measure'-'sensor67_measure"]=X_before_Normalizat
X_before_Normalization["sensor27_measure'-'sensor47_measure"]=X_before_Normalizat
X_before_Normalization["sensor27_measure'-'sensor46_measure"]=X_before_Normalizat
X_before_Normalization["sensor92_measure'-'sensor93_measure"]=X_before_Normalizat
X_before_Normalization["sensor14_measure'-'sensor15_measure"]=X_before_Normalizat
X_before_Normalization["sensor72_measure'-'sensor78_measure"]=X_before_Normalizat
X_before_Normalization["sensor95_measure'-'sensor94_measure"]=X_before_Normalizat
X_before_Normalization["sensor12_measure'-'sensor13_measure"]=X_before_Normalizat
X_before_Normalization["sensor89_measure'-'sensor33_measure"]=X_before_Normalizat
X_before_Normalization["sensor90_measure'-'sensor91_measure"]=X_before_Normalizat
X_before_Normalization["sensor27_measure'-'sensor14_measure"]=X_before_Normalizat
X_before_Normalization["sensor14_measure'-'sensor46_measure"]=X_before_Normalizat
X_before_Normalization["sensor14_measure'-'sensor67_measure"]=X_before_Normalizat
X_before_Normalization["sensor14_measure'-'sensor47_measure"]=X_before_Normalizat
X_before_Normalization["sensor32_measure'-'sensor14_measure"]=X_before_Normalizat
X_before_Normalization["sensor8_measure'-'sensor14_measure"]=X_before_Normalizati
X_before_Normalization["sensor97_measure'-'sensor96_measure"]=X_before_Normalizat
X_before_Normalization["sensor32_measure'-'sensor33_measure"]=X_before_Normalizat
X_before_Normalization["sensor33_measure'-'sensor8_measure"]=X_before_Normalizati
X_before_Normalization["sensor1_measure'-'sensor45_measure"]=X_before_Normalizati
X_before_Normalization["sensor89_measure'-'sensor35_measure"]=X_before_Normalizat
X_before_Normalization["sensor14_measure'-'sensor33_measure"]=X_before_Normalizat
X_before_Normalization["sensor33_measure'-'sensor27_measure"]=X_before_Normalizat
X_before_Normalization["sensor8_measure'-'sensor27_measure"]=X_before_Normalizati
X_before_Normalization["sensor33_measure'-'sensor17_measure"]=X_before_Normalizat
X_before_Normalization["sensor15_measure'-'sensor27_measure"]=X_before_Normalizat
X_before_Normalization["sensor32_measure'-'sensor27_measure"]=X_before_Normalizat
X_before_Normalization["sensor67_measure'-'sensor33_measure"]=X_before_Normalizat
X_before_Normalization["sensor33_measure'-'sensor47_measure"]=X_before_Normalizat
X_before_Normalization["sensor46_measure'-'sensor33_measure"]=X_before_Normalizat
X_before_Normalization["sensor8_measure'-'sensor46_measure"]=X_before_Normalizati
X_before_Normalization["sensor8_measure'-'sensor67_measure"]=X_before_Normalizati
X_before_Normalization["sensor8_measure'-'sensor47_measure"]=X_before_Normalizati
X_before_Normalization["sensor46_measure'-'sensor15_measure"]=X_before_Normalizat
X_before_Normalization["sensor15_measure'-'sensor67_measure"]=X_before_Normalizat
X_before_Normalization["sensor47_measure'-'sensor15_measure"]=X_before_Normalizat
X_before_Normalization["sensor46_measure'-'sensor89_measure"]=X_before_Normalizat
X_before_Normalization["sensor67_measure'-'sensor89_measure"]=X_before_Normalizat
X_before_Normalization["sensor47_measure'-'sensor89_measure"]=X_before_Normalizat
X_before_Normalization["sensor46_measure'-'sensor32_measure"]=X_before_Normalizat
X_before_Normalization["sensor67_measure'-'sensor32_measure"]=X_before_Normalizat
X_before_Normalization["sensor47_measure'-'sensor32_measure"]=X_before_Normalizat
X_before_Normalization["sensor34_measure'-'sensor16_measure"]=X_before_Normalizat
X_before_Normalization["sensor32_measure'-'sensor15_measure"]=X_before_Normalizat
X_before_Normalization["sensor27_measure'-'sensor89_measure"]=X_before_Normalizat
X_before_Normalization["sensor8_measure'-'sensor15_measure"]=X_before_Normalizati
X_before_Normalization["sensor6_measure'-'sensor5_measure"]=X_before_Normalizatio
X_before_Normalization["sensor35_measure'-'sensor33_measure"]=X_before_Normalizat
X_before_Normalization["sensor35_measure'-'sensor17_measure"]=X_before_Normalizat
```

Based on the Approach 1, listed out top 8 least features from all ML MOdels and will remove such features since these features doesn't useful for ML modelling

In [124]:
```
#rwemove least features from lgbt
#X_before_Normalization.drop(['sensor19_measure','sensor47_measure','sensor58_mea


#least features XGBOOST and lgbt
X_before_Normalization.drop(['sensor101_measure','sensor106_measure','sensor107_m

#'sensor101_measure','sensor106_measure','sensor107_measure','sensor11_measure',
```

Remove columns which has more than 50 skewness

In [125]:
```
#Remove skewness >50

X_before_Normalization.drop(['sensor56_measure','sensor65_measure','sensor4_measu
```

Lets find out highly correlated features each other and will keep one feature and remove another feature to predict the model better.

In [126]:
```
# Creating correlation matrix
cor_matrix = X_before_Normalization.corr().abs()
#print(); print(cor_matrix)

# Selecting upper triangle of correlation matrix
upper_tri = cor_matrix.where(np.triu(np.ones(cor_matrix.shape),
                                     k=1).astype(np.bool))
#print(); print(upper_tri)

# Finding index of feature columns with correlation greater than 0.95
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.95
print(); print(to_drop)

# Droping Marked Features
#df1 = X_before_Normalization.drop(X_before_Normalization.columns[to_drop], axis=
#print(); print(df1.head())
```

```
['sensor13_measure', 'sensor14_measure', 'sensor15_measure', 'sensor26_histogra
m_bin3', 'sensor26_histogram_bin4', 'sensor27_measure', 'sensor32_measure', 'se
nsor33_measure', 'sensor35_measure', 'sensor45_measure', 'sensor46_measure', 's
ensor47_measure', 'sensor53_measure', 'sensor59_measure', 'sensor64_histogram_b
in5', 'sensor67_measure', 'sensor89_measure', 'sensor95_measure', 'sensor105_hi
stogram_bin4', "sensor27_measure'-'sensor47_measure", "sensor27_measure'-'senso
r46_measure", "sensor92_measure'-'sensor93_measure", "sensor95_measure'-'sensor
94_measure", "sensor12_measure'-'sensor13_measure", "sensor90_measure'-'sensor9
1_measure", "sensor27_measure'-'sensor14_measure", "sensor14_measure'-'sensor46
_measure", "sensor14_measure'-'sensor67_measure", "sensor14_measure'-'sensor47_
measure", "sensor32_measure'-'sensor14_measure", "sensor8_measure'-'sensor14_me
asure", "sensor97_measure'-'sensor96_measure", "sensor32_measure'-'sensor33_mea
sure", "sensor33_measure'-'sensor8_measure", "sensor89_measure'-'sensor35_measu
re", "sensor14_measure'-'sensor33_measure", "sensor33_measure'-'sensor27_measur
e", "sensor8_measure'-'sensor27_measure", "sensor33_measure'-'sensor17_measur
e", "sensor15_measure'-'sensor27_measure", "sensor32_measure'-'sensor27_measur
e", "sensor67_measure'-'sensor33_measure", "sensor33_measure'-'sensor47_measur
e", "sensor46_measure'-'sensor33_measure", "sensor8_measure'-'sensor46_measur
e", "sensor8_measure'-'sensor67_measure", "sensor8_measure'-'sensor47_measure",
"sensor46_measure'-'sensor15_measure", "sensor15_measure'-'sensor67_measure",
"sensor47_measure'-'sensor15_measure", "sensor46_measure'-'sensor89_measure",
"sensor67_measure'-'sensor89_measure", "sensor47_measure'-'sensor89_measure",
"sensor46_measure'-'sensor32_measure", "sensor67_measure'-'sensor32_measure",
"sensor47_measure'-'sensor32_measure", "sensor34_measure'-'sensor16_measure",
"sensor27_measure'-'sensor89_measure", "sensor8_measure'-'sensor15_measure", "s
ensor35_measure'-'sensor33_measure"]
```

Above are the highly correlated features. Lets remove such features

In [127]:
```
# Droping Marked Features lgbt no features added

#X_before_Normalization.drop(['sensor13_measure', 'sensor14_measure', 'sensor15_m

#Drping feagtures after added new features and later removed highly correlated fe

#X_before_Normalization.drop(['sensor33_measure', 'sensor35_measure', 'sensor59_m

#applying everyting
X_before_Normalization.drop(['sensor13_measure', 'sensor14_measure', 'sensor15_me
```

Splitting data into Train and cross validation(or test): Stratified Sampling

In [128]:
```
# extract data column project_is_approved from total_project_data and add it to v
# remove project_is_approved from total_project_data and store the data in to var
y = X_before_Normalization['target'].values
X_before_Normalization.drop(['target'], axis=1, inplace=True)
X = X_before_Normalization
X.head(1)
```

Out[128]:

	sensor1_measure	sensor3_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin
0	76698.0	2.130706e+09	0.0	0.0	0.

1 rows × 135 columns

In [129]:
```
#split the data in to train and cross validation and test before performing BOW,
# train test split
from sklearn.model_selection import train_test_split
#splitting data in to train and test with 33 percentage as test data
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.2, stra
#splitting train data in to train and cv with 33 percentage as cv data
```

Apply Normalization

In [130]:
```
#normalize dataframe https://stackoverflow.com/questions/26414913/normalize-colum

#X_norm = EquipFail_train_dataset.copy() #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler().fit(X_train1)
X_train1 = pd.DataFrame(min_max_scaler.transform(X_train1),columns=X_train1.colu
X_test1 = pd.DataFrame(min_max_scaler.transform(X_test1),columns=X_test1.columns)
#X_normalized = min_max_scaler.fit_transform(X_norm)
#X_norm_final = pd.DataFrame(min_max_scaler.fit_transform(X_norm),columns=X_norm.
X_train1
```

Out[130]:

	sensor1_measure	sensor3_measure	sensor5_measure	sensor6_measure	sensor7_histogram
0	0.394478	2.909833e-07	0.0	0.0	
1	0.014155	4.421068e-07	0.0	0.0	
2	0.000226	9.999998e-01	0.0	0.0	
3	0.024482	9.999998e-01	0.0	0.0	
4	0.016437	1.013748e-07	0.0	0.0	
...	...	...	...	...	
47995	0.000711	7.133783e-08	0.0	0.0	
47996	0.013839	3.510572e-07	0.0	0.0	
47997	0.000307	1.220252e-08	0.0	0.0	
47998	0.170086	7.133783e-08	0.0	0.0	
47999	0.018058	1.342278e-07	0.0	0.0	

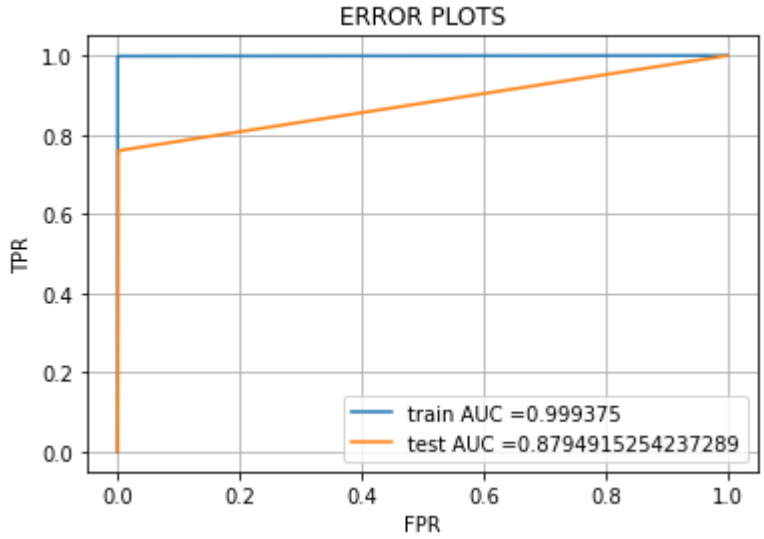
48000 rows × 135 columns

In [131]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

4.3.1 Random Forest With No Sampling

Apply ML Model with best parameters

In [132]:
```
RFClass_Final = RandomForestClassifier(max_depth=200,n_estimators=1000,n_jobs=-1,
RFClass_Final.fit(X_train, y_train)

y_train_RFClass_pred = RFClass_Final.predict(X_train)
y_test_RFClass_pred = RFClass_Final.predict(X_test)

train_RFClass_fpr, train_RFClass_tpr, train_RFClass_thresholds = roc_curve(y_trai
test_RFClass_fpr, test_RFClass_tpr, test_RFClass_thresholds = roc_curve(y_test, y

plt.plot(train_RFClass_fpr, train_RFClass_tpr, label="train AUC ="+str(auc(train_
plt.plot(test_RFClass_fpr, test_RFClass_tpr, label="test AUC ="+str(auc(test_RFCl
AUC_RFClass_Train = str(auc(train_RFClass_fpr, train_RFClass_tpr))
AUC_RFClass_Test = str(auc(test_RFClass_fpr, test_RFClass_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Observations:

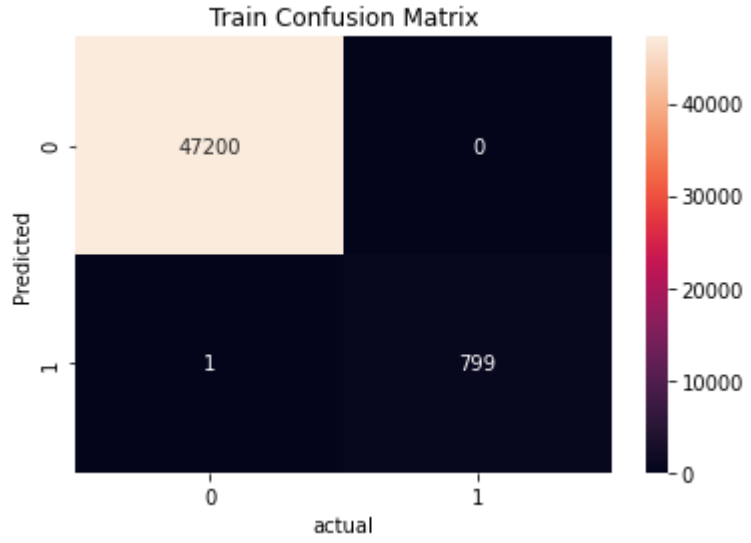
Train Accuray with Hyper Parameter ({'max_depth': 200, 'n_estimators': 1000}) is 100 %

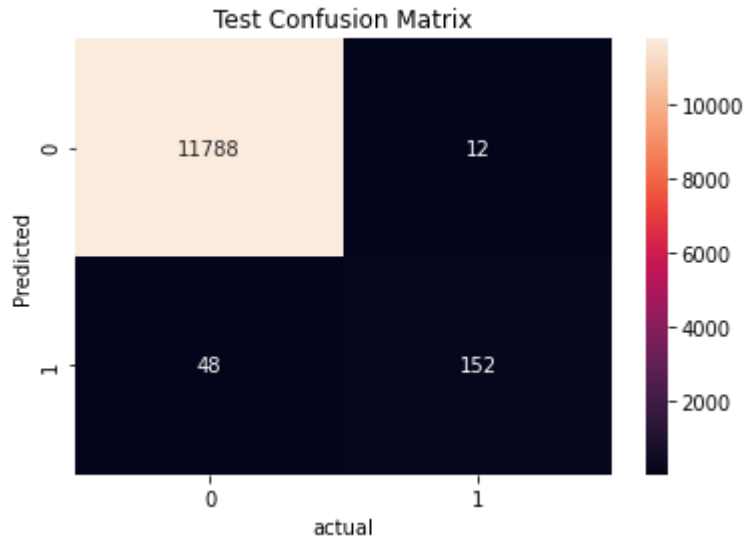
Test Accuray with Hyper Parameter ({'max_depth': 200, 'n_estimators': 1000}) is 88 %

Plot AUC, Confusion Matrix and F1 Scores

In [133]:
```
best_RFClass_t = find_best_threshold(train_RFClass_thresholds, train_RFClass_fpr,
RFClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_train_RF
df_cm1=pd.DataFrame(RFClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

#heatmap confusion matrix https://stackoverflow.com/questions/35572000/how-can-i-
RFClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_RFCla
df_cm1=pd.DataFrame(RFClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.99875 for threshold 1.0
```





Observations:

Random Forest is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [134]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_RFClass_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_RFClass_pred))
```

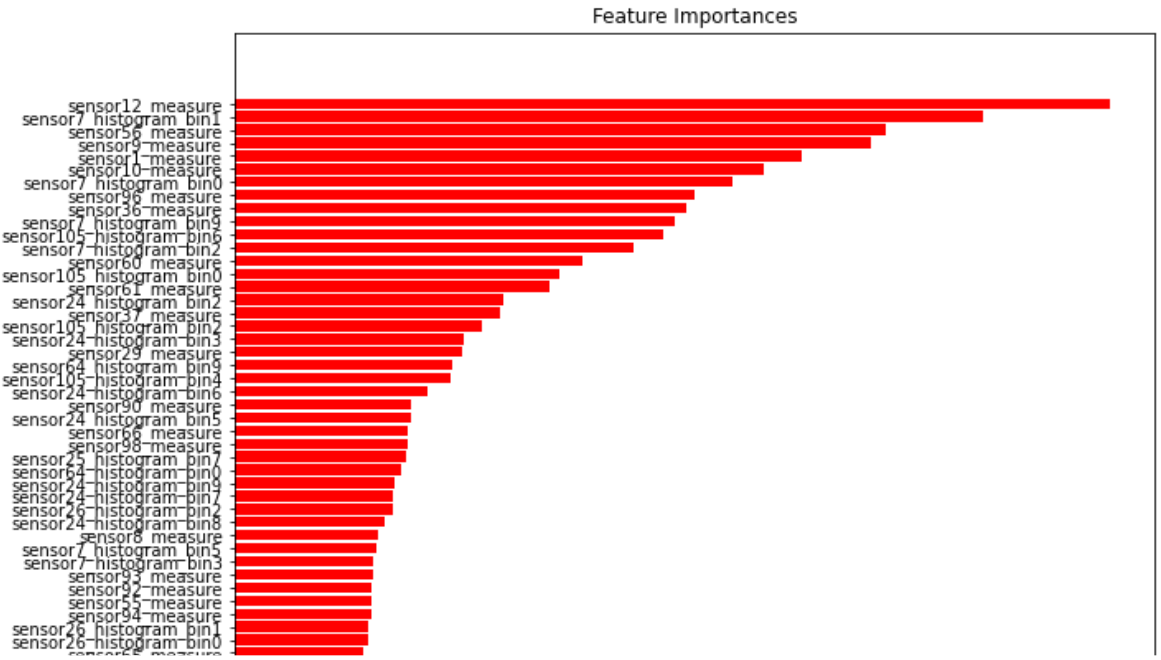
```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00       800

    accuracy                           1.00     48000
   macro avg       1.00      1.00      1.00     48000
weighted avg       1.00      1.00      1.00     48000

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.93      0.76      0.84       200

    accuracy                           0.99     12000
   macro avg       0.96      0.88      0.92     12000
weighted avg       0.99      0.99      0.99     12000
```

In [135]:
```
F1_Score_RFClass_Train=f1_score(y_train,y_train_RFClass_pred)
F1_Score_RFClass_Test=f1_score(y_test,y_test_RFClass_pred)
print('Train f1 score',f1_score(y_train,y_train_RFClass_pred))
print('Test f1 score',f1_score(y_test,y_test_RFClass_pred))
```

```
Train f1 score 0.9993746091307067
Test f1 score 0.8351648351648352
```

Feature Importance

In [136]:
```
### Feature ImportanceRF_Train_features = X_train.columns
RF_importances = RFClass_Final.feature_importances_
RF_indices = (np.argsort(RF_importances))[-100:]
plt.figure(figsize=(10,16))
plt.title('Feature Importances')
plt.barh(range(len(RF_indices)), RF_importances[RF_indices], color='r', align='ce
plt.yticks(range(len(RF_indices)), [RF_Train_features[i] for i in RF_indices])
plt.xlabel('Relative Importance')
plt.show()
```



4.3.2 RandomForest with RandomOverSampler

In [137]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [138]:
```
#randomoversampler sklearn example https://imbalanced-learn.readthedocs.io/en/sta
#Perform Resampling only on Train data
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_train, y_train = ros.fit_resample(X_train, y_train)
from collections import Counter
print(sorted(Counter(y_train).items()))
```

```
[(0.0, 47200), (1.0, 47200)]
```

In [139]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47200
47200
```

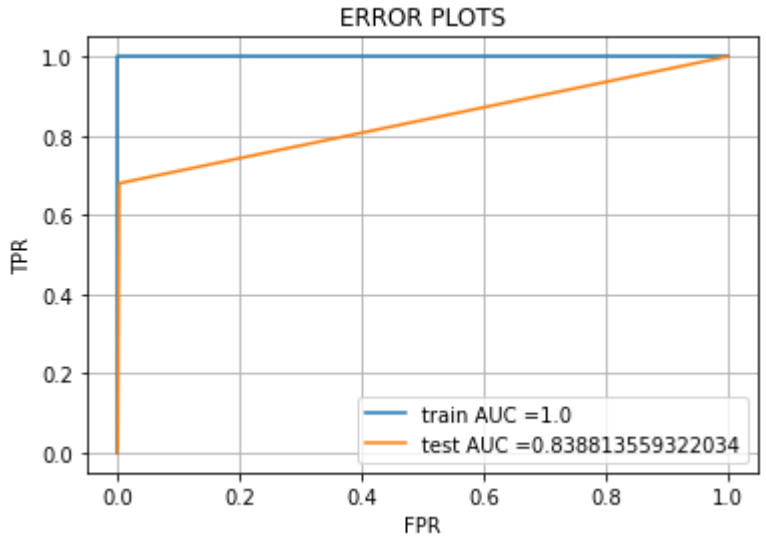
Apply ML Model with best parameters

In [140]:
```
from sklearn.ensemble import RandomForestClassifier

RFClass_Final = RandomForestClassifier(n_estimators=1000,n_jobs=-1)
#class_weight='balanced',max_depth=BestRFClassParam.get('max_depth'),n_estimators
RFClass_Final.fit(X_train, y_train)

y_train_RFClass_pred = RFClass_Final.predict(X_train)
y_test_RFClass_pred = RFClass_Final.predict(X_test)


#RFClass_Final = RandomForestClassifier(n_estimators=1000,n_jobs=-1)
#0.80
```

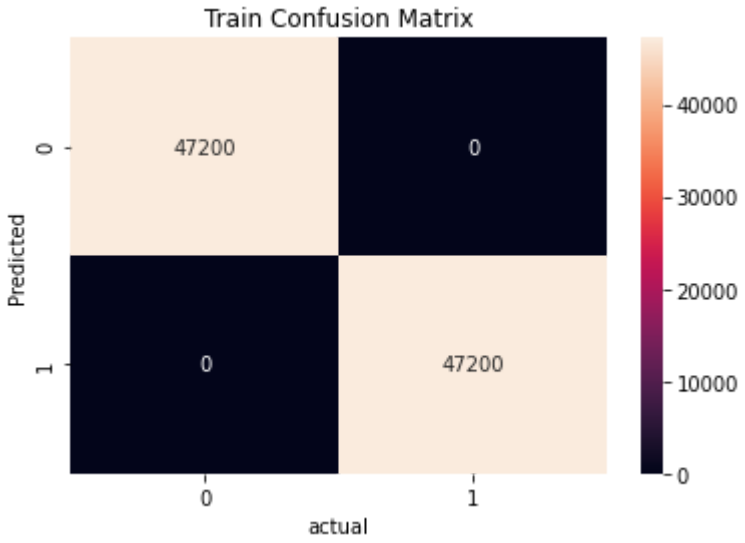
Plot AUC, Confusion Matrix and F1 Scores

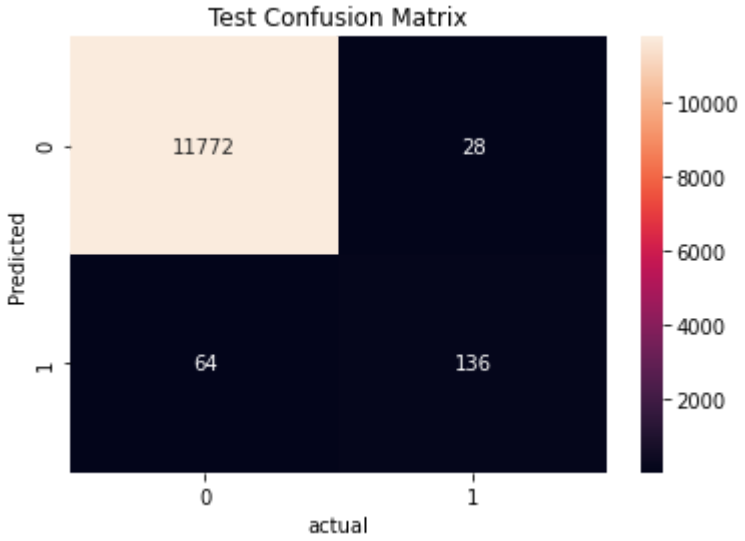
In [141]:

```
train_RFClass_fpr, train_RFClass_tpr, train_RFClass_thresholds = roc_curve(y_trai
test_RFClass_fpr, test_RFClass_tpr, test_RFClass_thresholds = roc_curve(y_test, y

plt.plot(train_RFClass_fpr, train_RFClass_tpr, label="train AUC ="+str(auc(train_
plt.plot(test_RFClass_fpr, test_RFClass_tpr, label="test AUC ="+str(auc(test_RFCl
AUC_RFClass_Train = str(auc(train_RFClass_fpr, train_RFClass_tpr))
AUC_RFClass_Test = str(auc(test_RFClass_fpr, test_RFClass_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
best_RFClass_t = find_best_threshold(train_RFClass_thresholds, train_RFClass_fpr,
RFClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_train_RF
df_cm1=pd.DataFrame(RFClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

#heatmap confusion matrix https://stackoverflow.com/questions/35572000/how-can-i-
RFClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_RFCla
df_cm1=pd.DataFrame(RFClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





Observations:

Train Accuray with Hyper Parameter (n_estimators=1000) is 100 %

Test Accuray with Hyper Parameter (n_estimators=1000) is 84 %

Random Forest is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [142]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_RFClass_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_RFClass_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00     47200

    accuracy                           1.00     94400
   macro avg       1.00      1.00      1.00     94400
weighted avg       1.00      1.00      1.00     94400

Test Classification Report
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00     11800
         1.0       0.83      0.68      0.75       200

    accuracy                           0.99     12000
   macro avg       0.91      0.84      0.87     12000
weighted avg       0.99      0.99      0.99     12000
```

4.3.3 LIGHT GBM Classifer with No Sampler

Apply ML Model with best parameters

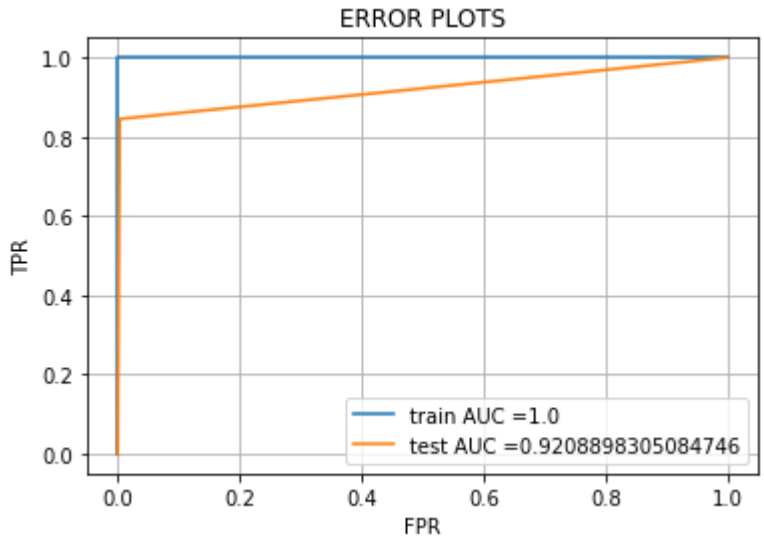
In [143]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [144]:
```
import lightgbm as lgb
clf1=lgb.LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=25,
               n_estimators=200, num_leaves=900, objective='binary',
               random_state=0, scale_pos_weight=59, silent=False,reg_lambda=0.28)
clf1.fit(X_train, y_train)
```

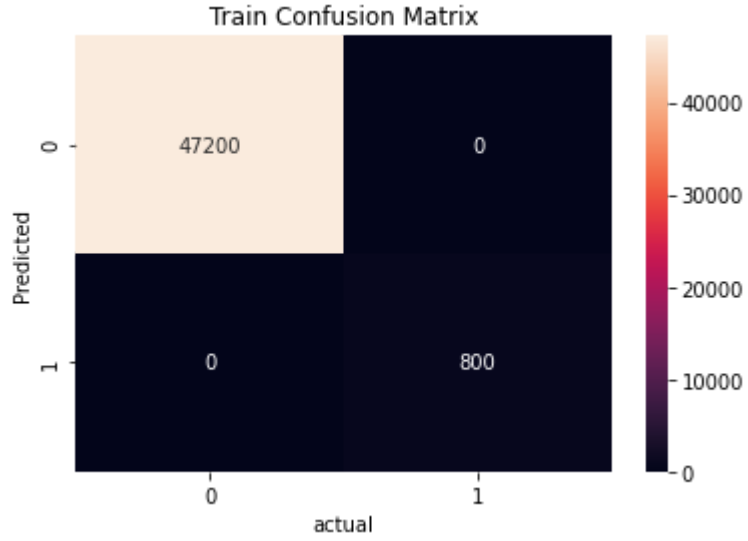
```
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignore
d. Current value: boosting=gbdt
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignore
d. Current value: boosting=gbdt
[LightGBM] [Info] Number of positive: 800, number of negative: 47200
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.044247 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 32035
[LightGBM] [Info] Number of data points in the train set: 48000, number of u
sed features: 133
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

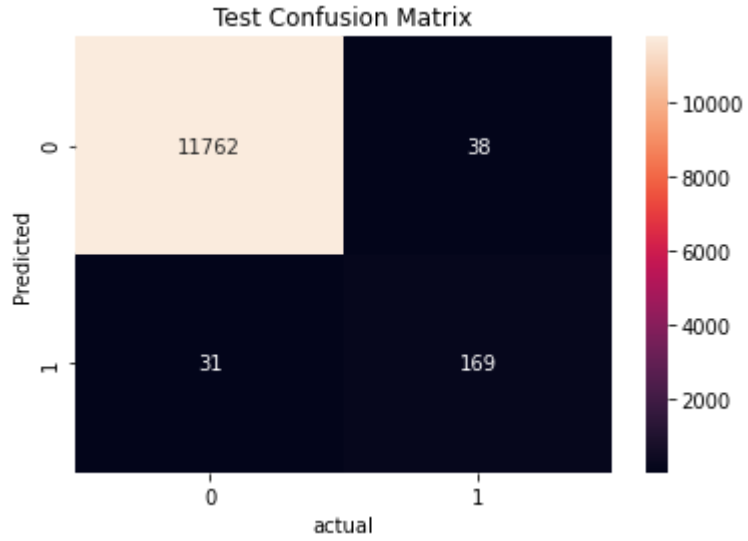
Plot AUC, Confusion Matrix and F1 Scores

In [145]:
```
y_train_pred = clf1.predict(X_train)
y_test_pred = clf1.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





```
Train f1 score 1.0
Test f1 score 0.8304668304668306
```

Observations:

Train Accuray with Hyper Parameter (max_depth=25, n_estimators=200) is 100 %

Test Accuray with Hyper Parameter (max_depth=25, n_estimators=200) is 93 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [146]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00       800

    accuracy                           1.00     48000
   macro avg       1.00      1.00      1.00     48000
weighted avg       1.00      1.00      1.00     48000

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.82      0.84      0.83       200

    accuracy                           0.99     12000
   macro avg       0.91      0.92      0.91     12000
weighted avg       0.99      0.99      0.99     12000
```

4.3.4 LIGHT GBM With RandomOverSampler

In [147]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [148]:
```
#randomoversampler sklearn example https://imbalanced-learn.readthedocs.io/en/sta
#Perform Resampling only on Train data
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_train, y_train = ros.fit_resample(X_train, y_train)
from collections import Counter
print(sorted(Counter(y_train).items()))
```

```
[(0.0, 47200), (1.0, 47200)]
```

In [149]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47200
47200
```

Apply ML Model with best parameters

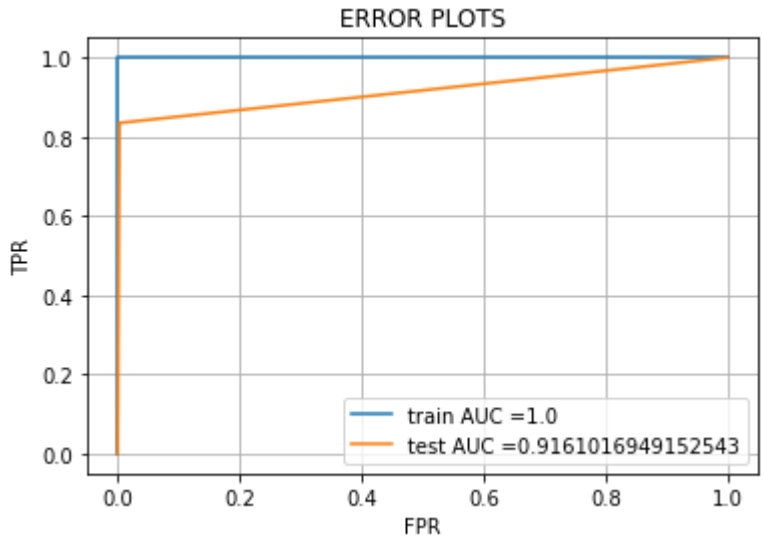
In [150]:
```
import lightgbm as lgb
clf1 = lgb.LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=10
              n_estimators=200, num_leaves=300, objective='binary',
               random_state=0, reg_lambda=0.28)
clf1.fit(X_train, y_train)
```

```
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignored.
Current value: boosting=gbdt
```

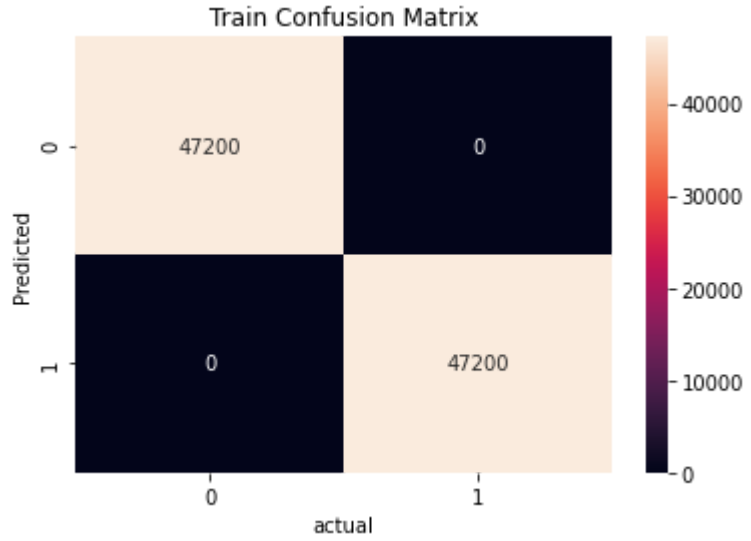
Out[150]:
```
LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=10,
               n_estimators=200, num_leaves=300, objective='binary',
               random_state=0, reg_lambda=0.28)
```

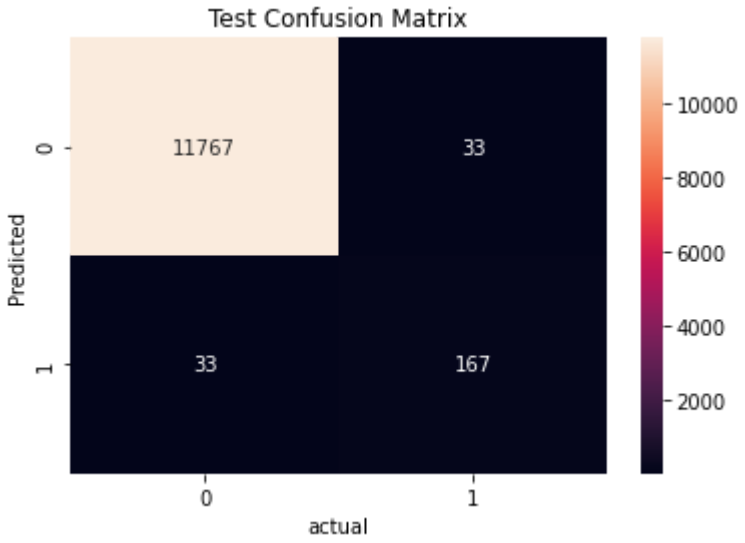
Plot AUC, Confusion Matrix and F1 Scores

In [151]:
```
y_train_pred1 = clf1.predict(X_train)
y_test_pred1 = clf1.predict(X_test)



train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred1)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred1)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





Observations:

Train Accuray with Hyper Parameter (max_depth=10, n_estimators=200) is 100 %

Test Accuray with Hyper Parameter (max_depth=10, n_estimators=200) is 92 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [152]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred1))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred1))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00     47200

    accuracy                           1.00     94400
   macro avg       1.00      1.00      1.00     94400
weighted avg       1.00      1.00      1.00     94400

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.83      0.83      0.83       200

    accuracy                           0.99     12000
   macro avg       0.92      0.92      0.92     12000
weighted avg       0.99      0.99      0.99     12000
```

4.3.5 LIGHT GBM With SMOTE

In [153]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [154]:
```
#https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-mi
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 0)
X_train, y_train = sm.fit_sample(X_train, y_train)

print('After OverSampling, the shape of train_X: {}'.format(X_train.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train == 0)))
```

```
After OverSampling, the shape of train_X: (94400, 135)
After OverSampling, the shape of train_y: (94400,)

After OverSampling, counts of label '1': 47200
After OverSampling, counts of label '0': 47200
```

In [155]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47200
47200
```

Apply ML Model with best parameters

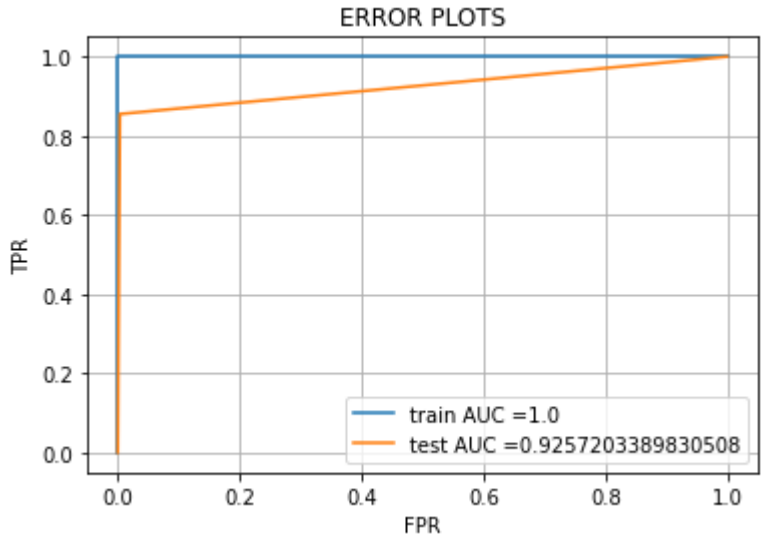
In [156]:
```
import lightgbm as lgb
clf1 = lgb.LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=10
              n_estimators=200, num_leaves=300, objective='binary',
               random_state=0, reg_lambda=0.28)
clf1.fit(X_train, y_train)
```

```
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignored.
Current value: boosting=gbdt
```

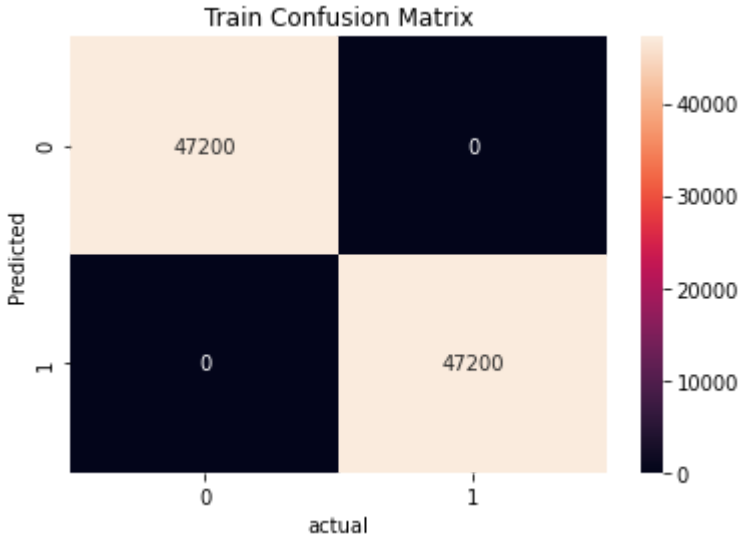
Out[156]:
```
LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=10,
               n_estimators=200, num_leaves=300, objective='binary',
               random_state=0, reg_lambda=0.28)
```

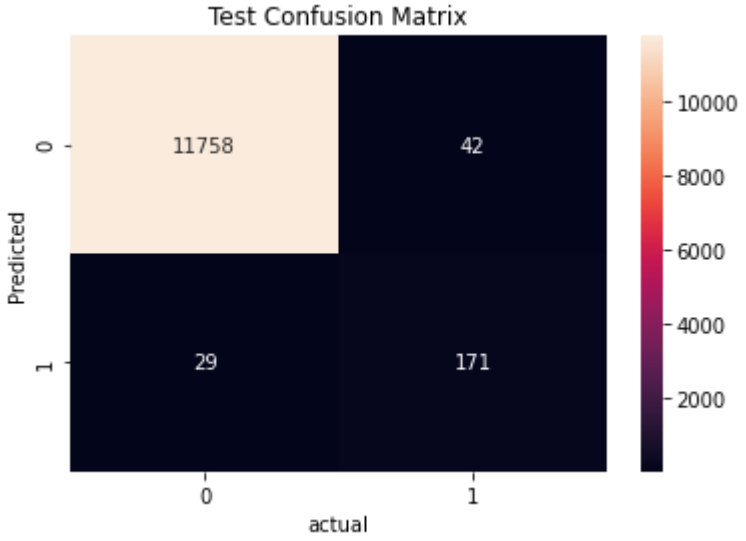
Plot AUC, Confusion Matrix and F1 Scores

In [157]:
```
y_train_pred1 = clf1.predict(X_train)
y_test_pred1 = clf1.predict(X_test)


train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred1)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred1)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```



```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





Observations:

Train Accuray with Hyper Parameter (max_depth=10, n_estimators=200) is 100 %

Test Accuray with Hyper Parameter (max_depth=10, n_estimators=200) is 93 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [158]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred1))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred1))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00     47200

    accuracy                           1.00     94400
   macro avg       1.00      1.00      1.00     94400
weighted avg       1.00      1.00      1.00     94400

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.80      0.85      0.83       200

    accuracy                           0.99     12000
   macro avg       0.90      0.93      0.91     12000
weighted avg       0.99      0.99      0.99     12000
```

In [ ]:
```
```

4.3.6 LIGHT GBM With SMOTE TOMEK

In [159]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [160]:
```
#https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-mi
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import TomekLinks
sm = SMOTETomek(random_state = 0)
X_train, y_train = sm.fit_sample(X_train, y_train)

print('After OverSampling, the shape of train_X: {}'.format(X_train.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train == 0)))
```

```
After OverSampling, the shape of train_X: (94394, 135)
After OverSampling, the shape of train_y: (94394,)

After OverSampling, counts of label '1': 47197
After OverSampling, counts of label '0': 47197
```

In [161]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47197
47197
```

Apply ML Model with best parameters

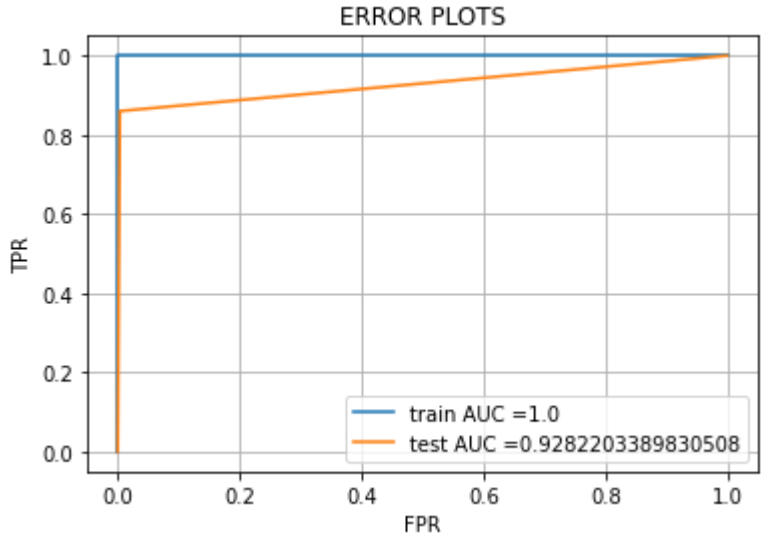
In [162]:
```
import lightgbm as lgb
clf1 = lgb.LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=10
              n_estimators=200, num_leaves=300, objective='binary',
               random_state=0)
clf1.fit(X_train, y_train)
```

```
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignored.
Current value: boosting=gbdt
```

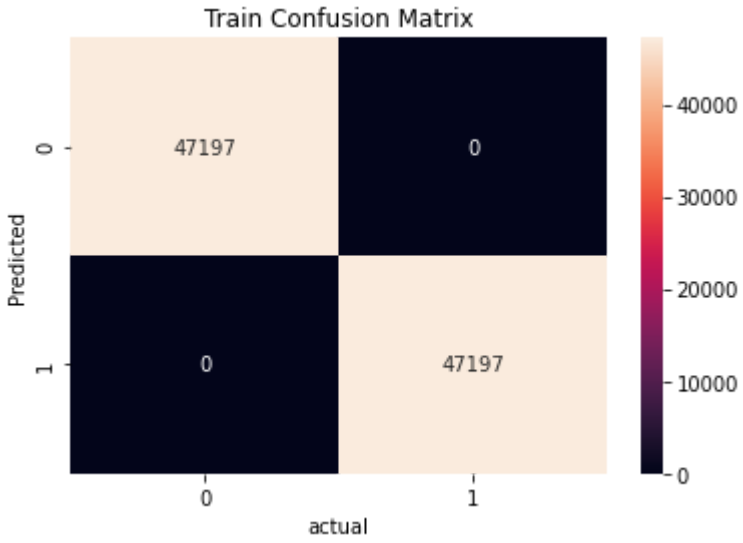
Out[162]:
```
LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=10,
               n_estimators=200, num_leaves=300, objective='binary',
               random_state=0)
```

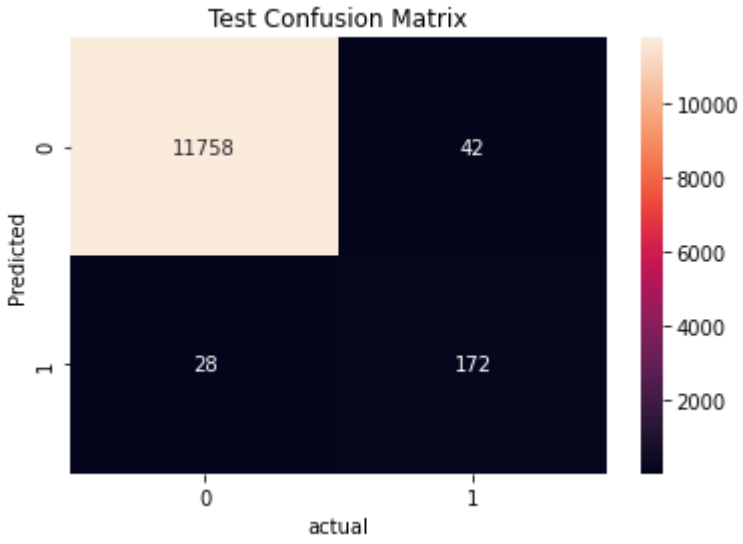
Plot AUC, Confusion Matrix and F1 Scores

In [163]:
```
y_train_pred1 = clf1.predict(X_train)
y_test_pred1 = clf1.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred1)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred1)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()
```



```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





Observations:

Train Accuray with Hyper Parameter (max_depth=10, n_estimators=200) is 100 %

Test Accuray with Hyper Parameter (max_depth=10, n_estimators=200) is 93 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [164]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred1))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred1))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47197
         1.0       1.00      1.00      1.00     47197

    accuracy                           1.00     94394
   macro avg       1.00      1.00      1.00     94394
weighted avg       1.00      1.00      1.00     94394

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.80      0.86      0.83       200

    accuracy                           0.99     12000
   macro avg       0.90      0.93      0.91     12000
weighted avg       0.99      0.99      0.99     12000
```

4.3.7 XGBOOST with No Sampling

In [165]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

Apply ML Model with best parameters

In [166]:
```
from xgboost import XGBClassifier
XGClassifier_final = XGBClassifier(learning_rate =0.1,n_estimators=1000,max_depth
```

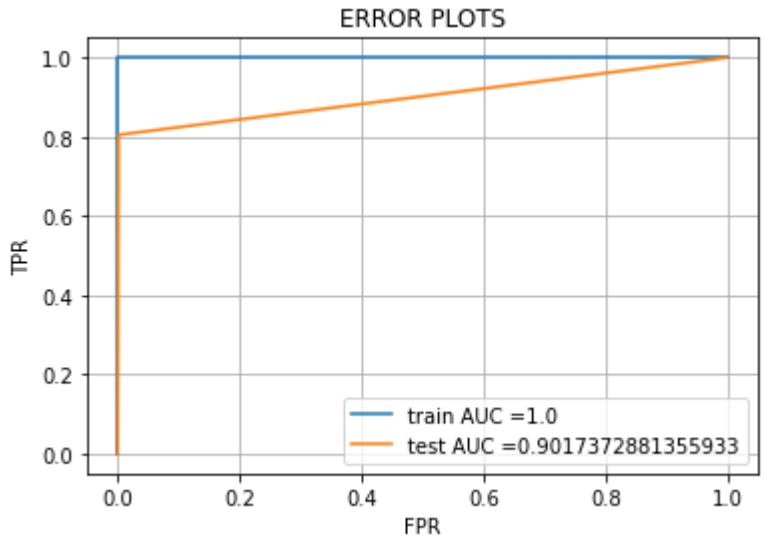
In [167]:
```
XGClassifier_final.fit(X_train, y_train )
                       #eval_metric=eval_metric, eval_set=eval_set,
                       #verbose=True)
```

```
c:\program files\python36\lib\site-packages\xgboost\sklearn.py:892: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[17:25:52] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

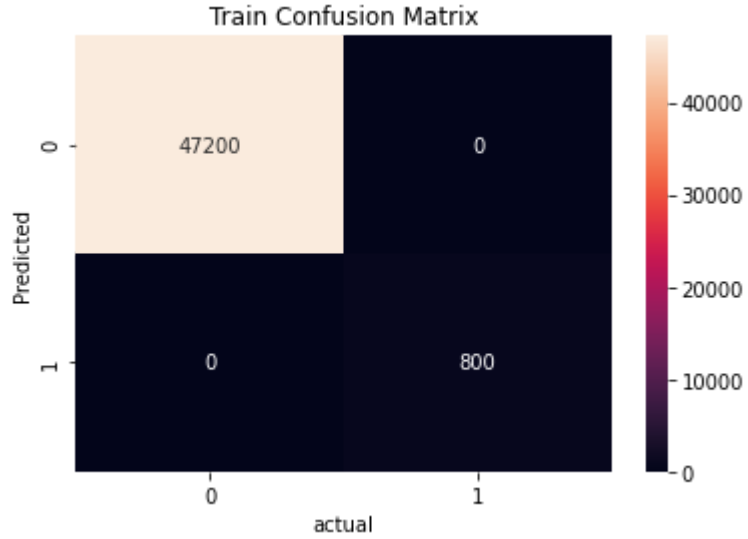
Out[167]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=1000, n_jobs=4, nthread=4, num_parallel_tree=1,
              random_state=27, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              seed=27, subsample=0.8, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

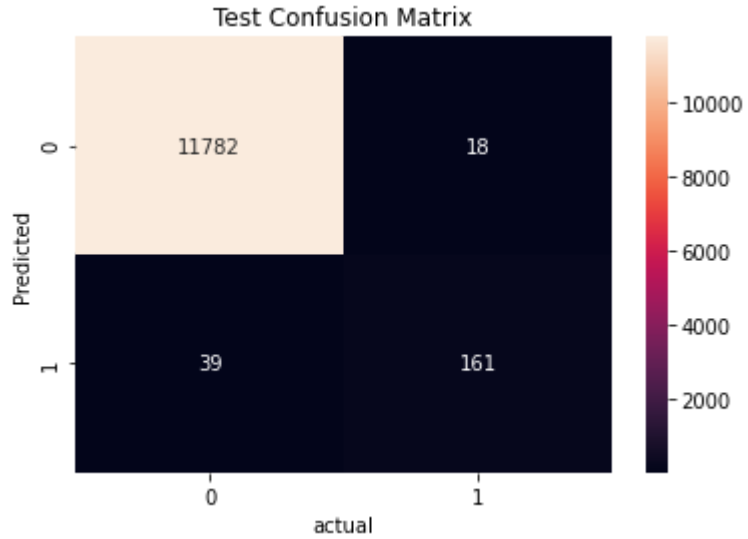
Plot AUC, Confusion Matrix and F1 Scores

In [168]:
```
y_train_pred = XGClassifier_final.predict(X_train)
y_test_pred = XGClassifier_final.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```






```
Train f1 score 1.0
Test f1 score 0.8496042216358839
```

Observations:

Train Accuray with Hyper Parameter (max_depth=5, n_estimators=1000) is 100 %

Test Accuray with Hyper Parameter (max_depth=5, n_estimators=1000) is 90 %

XGBoost Classifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [169]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00       800

    accuracy                           1.00     48000
   macro avg       1.00      1.00      1.00     48000
weighted avg       1.00      1.00      1.00     48000

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.90      0.81      0.85       200

    accuracy                           1.00     12000
   macro avg       0.95      0.90      0.92     12000
weighted avg       1.00      1.00      1.00     12000
```

Feature Importance

In [170]:
```
# Feature importance

#lightGBM model fit
#gbm = lgb.LGBMClassifier()
#gbm.fit(train, target)
XGClassifier_final.feature_importances_

# importance of each attribute
fea_imp_ = pd.DataFrame({'cols':X_train.columns, 'fea_imp':XGClassifier_final.fea
fea_imp_.loc[fea_imp_.fea_imp <0.004].sort_values(by=['cols'], ascending = True)
```

Out[170]:

	cols	fea_imp
112	sensor102_measure	0.002236
113	sensor103_measure	0.003404
114	sensor104_measure	0.002641
115	sensor105_histogram_bin0	0.002976
116	sensor105_histogram_bin1	0.003158
...	...	...
100	sensor80_measure	0.002863
102	sensor90_measure	0.003908
105	sensor93_measure	0.002120
106	sensor94_measure	0.003338
108	sensor97_measure	0.001626

64 rows × 2 columns

In [ ]:
```

```

4.3.8 XGBOOST with RandomOverSampler

In [171]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [172]:
```
#randomoversampler sklearn example https://imbalanced-learn.readthedocs.io/en/sta
#Perform Resampling only on Train data
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_train, y_train = ros.fit_resample(X_train, y_train)
from collections import Counter
print(sorted(Counter(y_train).items()))
```

```
[(0.0, 47200), (1.0, 47200)]
```

In [173]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47200
47200
```

Apply ML Model with best parameters

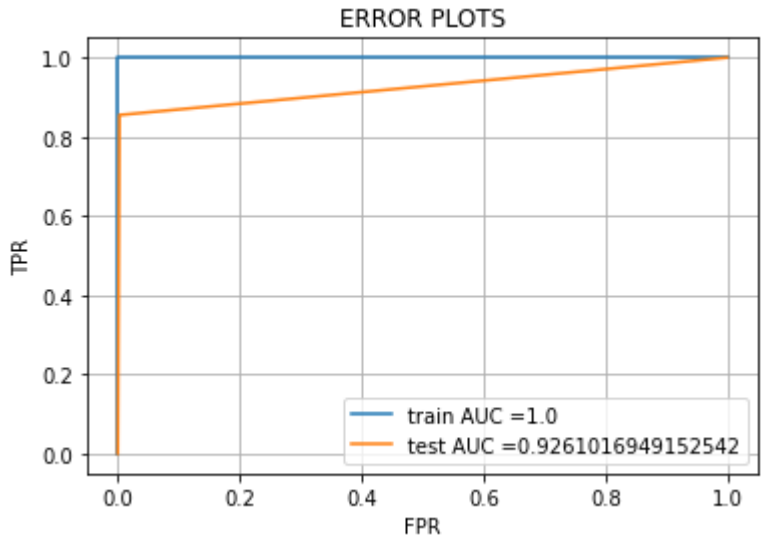
In [174]:
```
from xgboost import XGBClassifier
XGClassifier_final = XGBClassifier(n_estimators=1000,max_depth=7,random_state=0,
                                   objective= 'binary:logistic',n_jobs=-1,reg_lambda


XGClassifier_final.fit(X_train, y_train)
```

```
c:\program files\python36\lib\site-packages\xgboost\sklearn.py:892: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[17:27:54] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

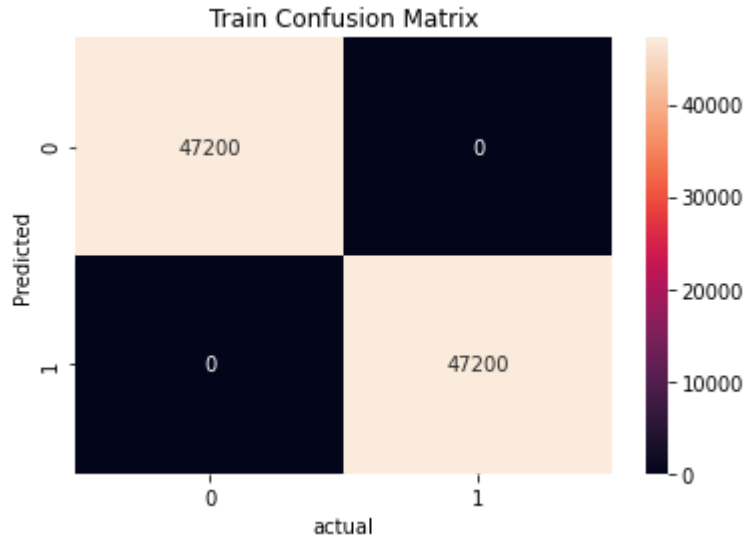
Out[174]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=1000, n_jobs=-1, num_parallel_tree=1, random_state=
0,
              reg_alpha=0, reg_lambda=2.5, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

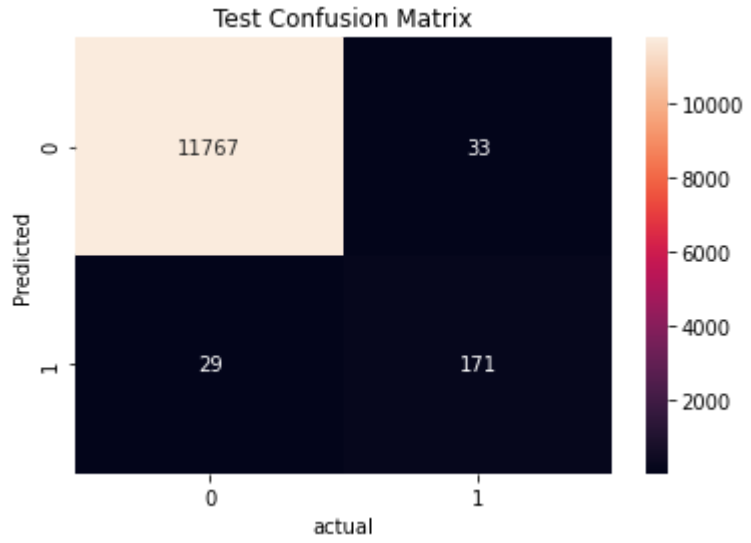
Plot AUC, Confusion Matrix and F1 Scores

In [175]:
```
y_train_pred = XGClassifier_final.predict(X_train)
y_test_pred = XGClassifier_final.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```






```
Train f1 score 1.0
Test f1 score 0.8465346534653465
```

Observations:

Train Accuray with Hyper Parameter (max_depth=7 n_estimators=1000) is 100 %

Test Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 93 %

XGBoost Classifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [176]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00     47200

    accuracy                           1.00     94400
   macro avg       1.00      1.00      1.00     94400
weighted avg       1.00      1.00      1.00     94400

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.84      0.85      0.85       200

    accuracy                           0.99     12000
   macro avg       0.92      0.93      0.92     12000
weighted avg       0.99      0.99      0.99     12000
```

4.3.9 XGBoost with SMOTE

In [177]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [178]:
```
#https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-mi
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 0)
X_train, y_train = sm.fit_sample(X_train, y_train)

print('After OverSampling, the shape of train_X: {}'.format(X_train.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train == 0)))
```

```
After OverSampling, the shape of train_X: (94400, 135)
After OverSampling, the shape of train_y: (94400,)

After OverSampling, counts of label '1': 47200
After OverSampling, counts of label '0': 47200
```

In [179]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47200
47200
```

Apply ML Model with best parameters

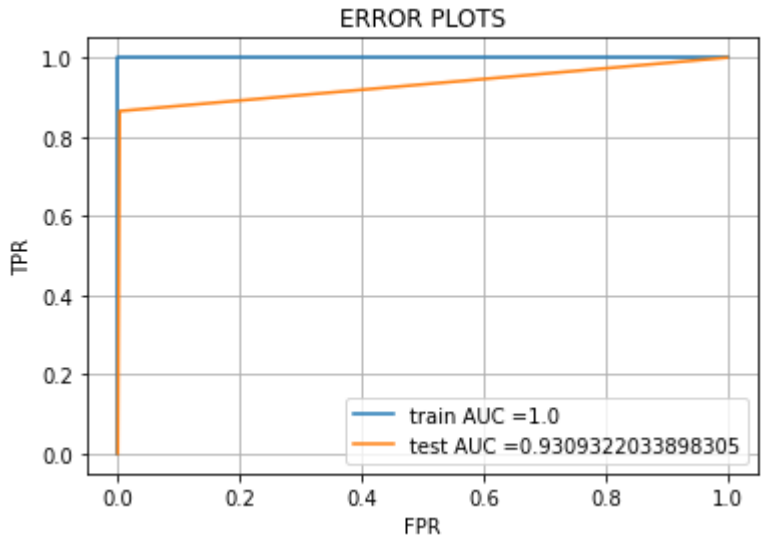
In [180]:
```
from xgboost import XGBClassifier
XGClassifier_final = XGBClassifier(n_estimators=1000,max_depth=7,random_state=0,
                                   objective= 'binary:logistic',n_jobs=-1,reg_lambda


XGClassifier_final.fit(X_train, y_train)
```

```
c:\program files\python36\lib\site-packages\xgboost\sklearn.py:892: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[17:30:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

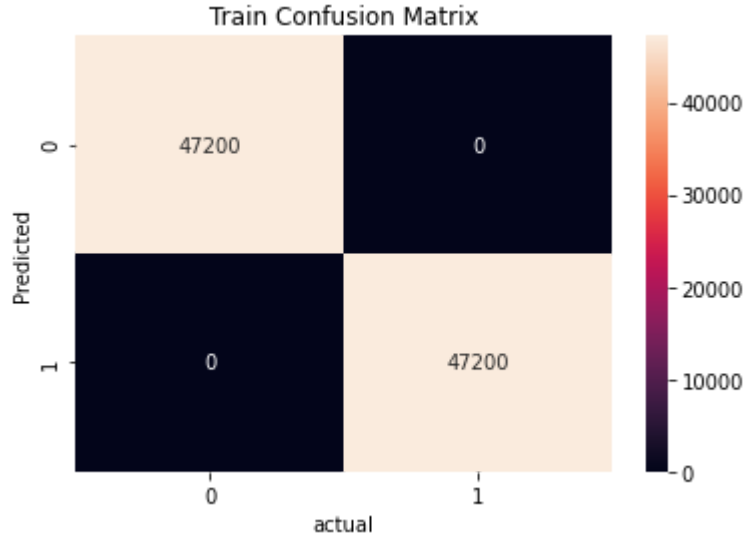
Out[180]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=1000, n_jobs=-1, num_parallel_tree=1, random_state=
0,
              reg_alpha=0, reg_lambda=2.5, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

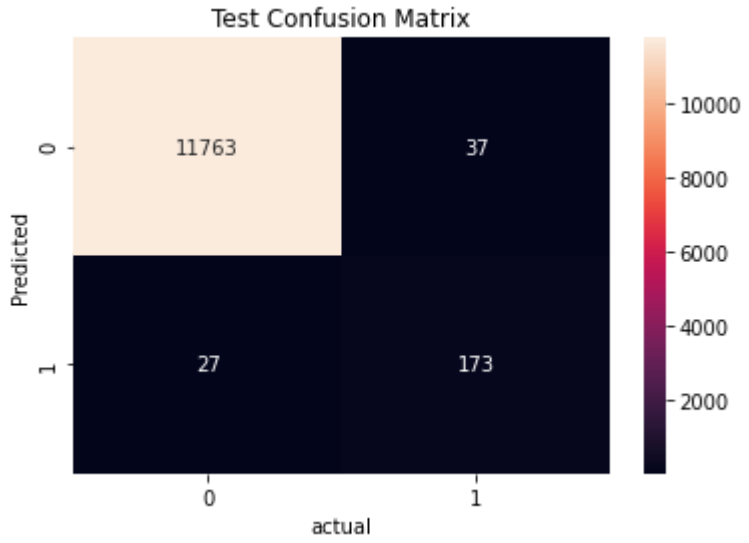
Plot AUC, Confusion Matrix and F1 Scores

In [181]:
```
y_train_pred = XGClassifier_final.predict(X_train)
y_test_pred = XGClassifier_final.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```






```
Train f1 score 1.0
Test f1 score 0.8439024390243902
```

Observations:

Train Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 100 %

Test Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 93 %

XGBoost Classifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [182]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00     47200

    accuracy                           1.00     94400
   macro avg       1.00      1.00      1.00     94400
weighted avg       1.00      1.00      1.00     94400

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.82      0.86      0.84       200

    accuracy                           0.99     12000
   macro avg       0.91      0.93      0.92     12000
weighted avg       0.99      0.99      0.99     12000
```

4.3.10 XGBoost with SMOTE TOMEK

In [183]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [184]:
```
#https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-mi
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import TomekLinks
sm = SMOTETomek(random_state = 0)
X_train, y_train = sm.fit_sample(X_train, y_train)

print('After OverSampling, the shape of train_X: {}'.format(X_train.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train == 0)))
```

```
After OverSampling, the shape of train_X: (94394, 135)
After OverSampling, the shape of train_y: (94394,)

After OverSampling, counts of label '1': 47197
After OverSampling, counts of label '0': 47197
```

In [185]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47197
47197
```

Apply ML Model with best parameters

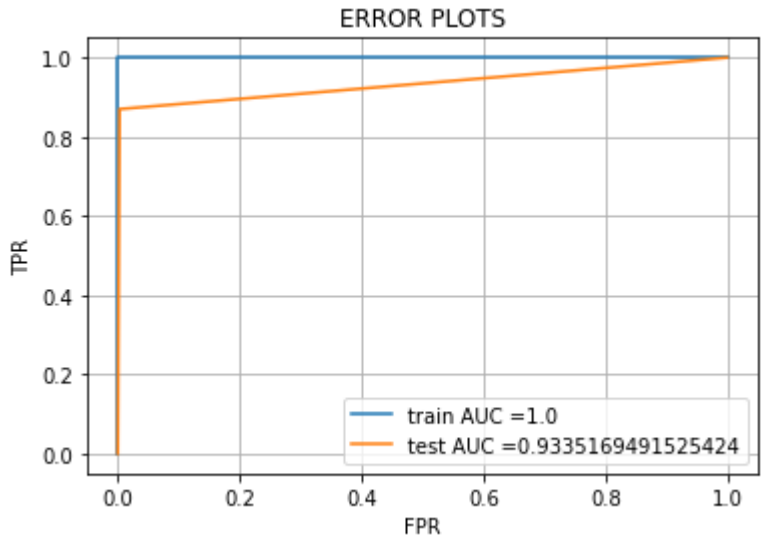
In [186]:
```
from xgboost import XGBClassifier
XGClassifier_final = XGBClassifier(n_estimators=1000,max_depth=7,
                                   objective= 'binary:logistic',n_jobs=-1,reg_alpha=


XGClassifier_final.fit(X_train, y_train)
```

```
c:\program files\python36\lib\site-packages\xgboost\sklearn.py:892: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[17:38:36] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

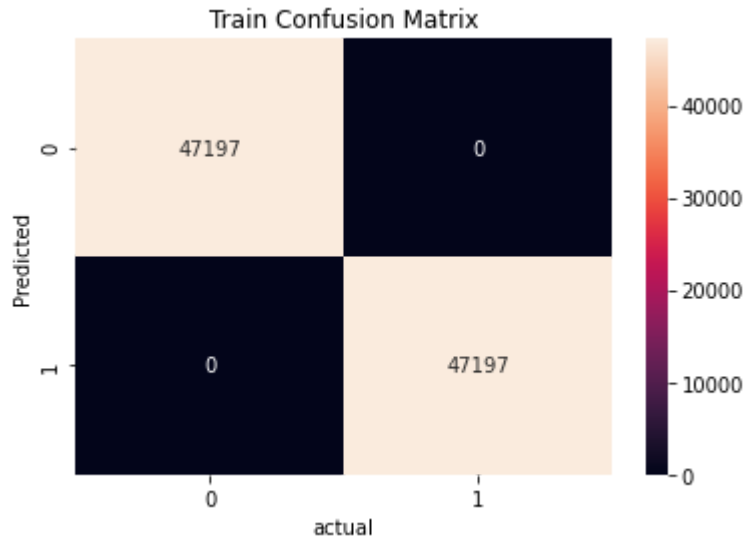
Out[186]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=1000, n_jobs=-1, num_parallel_tree=1, random_state=
0,
              reg_alpha=1e-05, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

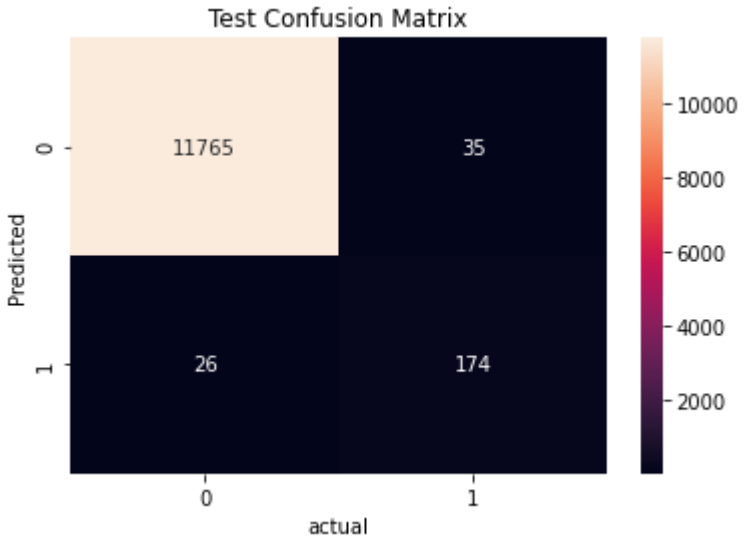
Plot AUC, Confusion Matrix and F1 Scores

In [187]:
```
y_train_pred = XGClassifier_final.predict(X_train)
y_test_pred = XGClassifier_final.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```






```
Train f1 score 1.0
Test f1 score 0.8508557457212714
```

Observations:

Train Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 100 %

Test Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 93 %

XGBoost Classifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [188]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47197
         1.0       1.00      1.00      1.00     47197

    accuracy                           1.00     94394
   macro avg       1.00      1.00      1.00     94394
weighted avg       1.00      1.00      1.00     94394

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.83      0.87      0.85       200

    accuracy                           0.99     12000
   macro avg       0.92      0.93      0.92     12000
weighted avg       1.00      0.99      0.99     12000
```

4.3.11 Approach 3: Conclusion

In [3]:
```
from prettytable import PrettyTable
import math

Finaloutput = PrettyTable()
Finaloutput1 = PrettyTable()

Finaloutput.field_names = ["Model", "Sampling", "Train AUC Score", "Test AUC Scor

Finaloutput.add_row(["Random Forest","No Sampling",100,88,100,84,92])
Finaloutput.add_row(["Random Forest","ROS",100,84,100,75,87])
Finaloutput.add_row(["LGBMClassifier","No Sampling",100,92,100,83,91])
Finaloutput.add_row(["LGBMClassifier","ROS",100,92,100,83,92])
Finaloutput.add_row(["LGBMClassifier","SMOTE Over Sampling",100,93,100,83,91])
Finaloutput.add_row(["LGBMClassifier","SMOTE TOMEK Over Sampling",100,93,100,83,9
Finaloutput.add_row(["XGBoost Classifier","No Sampling",100,90,100,85,92])
Finaloutput.add_row(["XGBoost Classifier","ROS",100,93,100,85,92])
Finaloutput.add_row(["XGBoost Classifier","SMOTE Over Sampling",100,93,100,84,92]
Finaloutput.add_row(["XGBoost Classifier","SMOTE TOMEK Over Sampling",100,93,100,



print(Finaloutput)
```

```
+--------------------+---------------------------+-----------------+-----------
-----+----------------+---------------+---------------------+
|       Model        |          Sampling         | Train AUC Score | Test AUC S
core | Train F1 Score | Test F1 Score | Test F1 Macro Score |
+--------------------+---------------------------+-----------------+-----------
-----+----------------+---------------+---------------------+
|   Random Forest    |        No Sampling        |       100       |       88
|      100       |       84      |          92         |
|   Random Forest    |            ROS            |       100       |       84
|      100       |       75      |          87         |
|   LGBMClassifier   |        No Sampling        |       100       |       92
|      100       |       83      |          91         |
|   LGBMClassifier   |            ROS            |       100       |       92
|      100       |       83      |          92         |
|   LGBMClassifier   |    SMOTE Over Sampling    |       100       |       93
|      100       |       83      |          91         |
|   LGBMClassifier   | SMOTE TOMEK Over Sampling |       100       |       93
|      100       |       83      |          91         |
| XGBoost Classifier |        No Sampling        |       100       |       90
|      100       |       85      |          92         |
| XGBoost Classifier |            ROS            |       100       |       93
|      100       |       85      |          92         |
| XGBoost Classifier |    SMOTE Over Sampling    |       100       |       93
|      100       |       84      |          92         |
| XGBoost Classifier | SMOTE TOMEK Over Sampling |       100       |       93
|      100       |       85      |          92         |
+--------------------+---------------------------+-----------------+-----------
-----+----------------+---------------+---------------------+
```

Observations:

XGBoost with ROS Sampling and SMOTE TOMEK Over Sampling has provided best AUC (0.93) and F1 (0.85) scores when compared with other ML moelds.

In [ ]:

4.4 Approach 4

Lets apply the following functionalities in the dataset and will proceed with applying ML models to predict the performance metric

1) Remove unused features (>70% of data contains null values)
2) Impute median for the missing values
3) Remove Top 8 least features and Apply Log Transformation
4) Apply Feature Engineering Techniques
5) Remove one of the highly correlated features between each other
6) Apply Normlization on the dataset
7) Apply ML Models

Have already implemented the functionalities of point 1 and 2 in Approach 1. Lets consider the same dataset and implement from point 3.

Remove Top 8 least features

In [191]:
```
X_before_Normalization = EquipFail_train_dataset.copy()
```

In [192]:
```
#rwemove least features
X_before_Normalization.drop(['sensor101_measure','sensor106_measure','sensor107_m
```

Log Normal Transformation

In [193]:
```
# Lets apply Log Normal distribution to reduce the skewness
#target_01_skew_old=[]
target_01_skew_old= pd.DataFrame()

target_01_skew_old=X_before_Normalization.iloc[:,1:].skew()

target_01_skew_old
```

Out[193]:
```
sensor1_measure              6.115752
sensor3_measure              1.877413
sensor4_measure            244.948972
sensor5_measure             92.531591
sensor6_measure             52.780674
                               ...
sensor105_histogram_bin5    14.635798
sensor105_histogram_bin6    10.353473
sensor105_histogram_bin7    15.803504
sensor105_histogram_bin8    13.491407
sensor105_histogram_bin9    29.369380
Length: 153, dtype: float64
```

In [194]:
```
# Lets apply Log Normal distribution to reduce the skewness
#target_01_skew_log=[]
target_01_skew_log= pd.DataFrame()
for skew_01_log_column in X_before_Normalization.columns[1:]:
    target_01_skew_log[skew_01_log_column]=np.log(X_before_Normalization[skew_01_
#target_01_skew_log["target"]=1
target_01_skew_log1=target_01_skew_log.skew()
```

In [195]:
```
target_01_skew_log1
```

Out[195]:
```
sensor1_measure            -0.835772
sensor3_measure             1.441372
sensor4_measure            -0.555298
sensor5_measure             7.531393
sensor6_measure             7.443812
                              ...
sensor105_histogram_bin5   -1.153176
sensor105_histogram_bin6   -0.835094
sensor105_histogram_bin7   -0.731840
sensor105_histogram_bin8   -0.250353
sensor105_histogram_bin9    0.642720
Length: 153, dtype: float64
```

Compare skewness with original dataset columns and with Log Transformation columns. Add better skewness columns in to list

In [196]:
```
target_01_skew_log_list =[]
for skew_01_column in X_before_Normalization.columns[1:]:
    print ("new",target_01_skew_log1.loc[skew_01_column])
    print("old",target_01_skew_old.loc[skew_01_column])
    #print(min(target_01_skew_log1.loc[skew_01_column], target_01_skew_old.loc[sk
    if (min(target_01_skew_log1.loc[skew_01_column], target_01_skew_old.loc[skew_
        target_01_skew_log_list.append(skew_01_column)
        print("took new one")
target_01_skew_log_list
```

```
new -0.8357719762061686
old 6.115752483633122
took new one
new 1.4413721280351979
old 1.8774126527812818
took new one
new -0.5552979014791289
old 244.94897171343118
took new one
new 7.531392719766963
old 92.53159100748171
took new one
new 7.443812280536036
old 52.780673569245266
took new one
new 20.929485200604102
old 154.25252677874337
took new one
new 9.993442075019527
```

Lets Apply Log transformation to the dataset as per above columns to reduce the skewness

In [197]:
```
for commonfeaturelistlength in range (len(target_01_skew_log_list)):
    X_before_Normalization[target_01_skew_log_list[commonfeaturelistlength]]=np.l
    print ("log transformation applied to column ",target_01_skew_log_list[commor
```

```
log transformation applied to column  sensor1_measure
log transformation applied to column  sensor3_measure
log transformation applied to column  sensor4_measure
log transformation applied to column  sensor5_measure
log transformation applied to column  sensor6_measure
log transformation applied to column  sensor7_histogram_bin0
log transformation applied to column  sensor7_histogram_bin1
log transformation applied to column  sensor7_histogram_bin2
log transformation applied to column  sensor7_histogram_bin3
log transformation applied to column  sensor7_histogram_bin4
log transformation applied to column  sensor7_histogram_bin5
log transformation applied to column  sensor7_histogram_bin6
log transformation applied to column  sensor7_histogram_bin7
log transformation applied to column  sensor7_histogram_bin8
log transformation applied to column  sensor7_histogram_bin9
log transformation applied to column  sensor8_measure
log transformation applied to column  sensor9_measure
log transformation applied to column  sensor10_measure
log transformation applied to column  sensor12_measure
```

Splitting data into Train and cross validation(or test): Stratified Sampling

In [198]:
```
# extract data column project_is_approved from total_project_data and add it to v
# remove project_is_approved from total_project_data and store the data in to var
y = X_before_Normalization['target'].values
X_before_Normalization.drop(['target'], axis=1, inplace=True)
X = X_before_Normalization
X.head(1)
```

Out[198]:

	sensor1_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sen
0	11.247644	21.479719	5.638355	0.0	0.0	

1 rows × 153 columns

In [199]:
```
#split the data in to train and cross validation and test before performing BOW,
# train test split
from sklearn.model_selection import train_test_split
#splitting data in to train and test with 33 percentage as test data
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.2, stra
#splitting train data in to train and cv with 33 percentage as cv data
```

Apply Normalization

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.

In [200]:
```
#normalize dataframe https://stackoverflow.com/questions/26414913/normalize-colum

#X_norm = EquipFail_train_dataset.copy() #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler().fit(X_train1)
X_train1 = pd.DataFrame(min_max_scaler.transform(X_train1),columns=X_train1.colu
X_test1 = pd.DataFrame(min_max_scaler.transform(X_test1),columns=X_test1.columns)
#X_normalized = min_max_scaler.fit_transform(X_norm)
#X_norm_final = pd.DataFrame(min_max_scaler.fit_transform(X_norm),columns=X_norm.
X_train1
```

Out[200]:

	sensor1_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure
0	0.937259	0.299414	0.397714	0.0	0.0
1	0.712823	0.318862	0.558576	0.0	0.0
2	0.434008	1.000000	0.241742	0.0	0.0
3	0.749776	1.000000	0.534378	0.0	0.0
4	0.722901	0.250464	0.430354	0.0	0.0
...	...	...	...	...	...
47995	0.511075	0.234195	0.397714	0.0	0.0
47996	0.711300	0.308139	0.528810	0.0	0.0
47997	0.454566	0.153439	0.270592	0.0	0.0
47998	0.880516	0.234195	0.397714	0.0	0.0
47999	0.729248	0.263480	0.456854	0.0	0.0

48000 rows × 153 columns

In [201]:
```
from sklearn.model_selection import KFold
cv=KFold(n_splits=5,random_state=None, shuffle=False)
```

4.4.1 XGBOOST With Random Over Sampling

In [202]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

In [203]:
```
#randomoversampler sklearn example https://imbalanced-learn.readthedocs.io/en/sta
#Perform Resampling only on Train data
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X_train, y_train = ros.fit_resample(X_train, y_train)
from collections import Counter
print(sorted(Counter(y_train).items()))
```

```
[(0.0, 47200), (1.0, 47200)]
```

In [204]:
```
i_class0_sampled = np.where(y_train == 0)[0]
i_class1_sampled = np.where(y_train == 1)[0]
print (len(i_class0_sampled))
print (len(i_class1_sampled))
```

```
47200
47200
```

Apply ML Model with best parameters

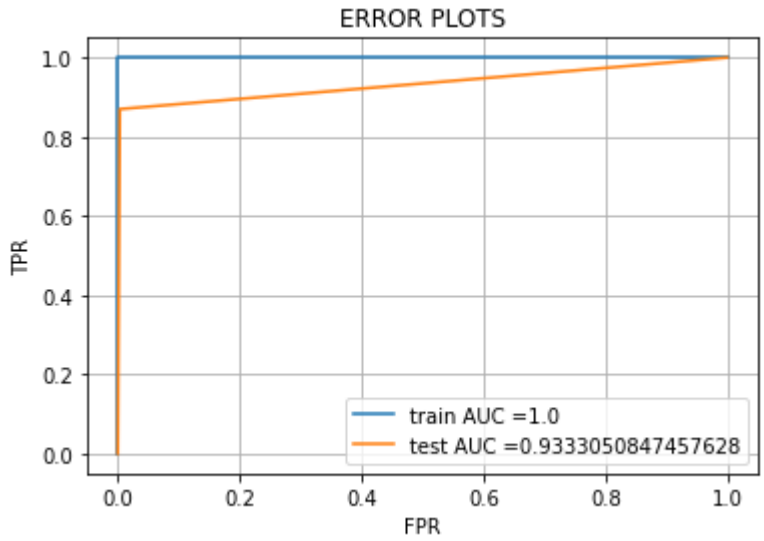
In [205]:
```
from xgboost import XGBClassifier
#XGClassifier_final = XGBClassifier(n_estimators=1000,reg_alpha=2,
#                                   objective= 'binary:logistic',n_jobs=-1,random_st
XGClassifier_final = XGBClassifier(n_estimators=1000,max_depth=7,
                                   objective= 'binary:logistic',n_jobs=-1,reg_alpha=

XGClassifier_final.fit(X_train, y_train)
```

```
c:\program files\python36\lib\site-packages\xgboost\sklearn.py:892: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[17:48:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

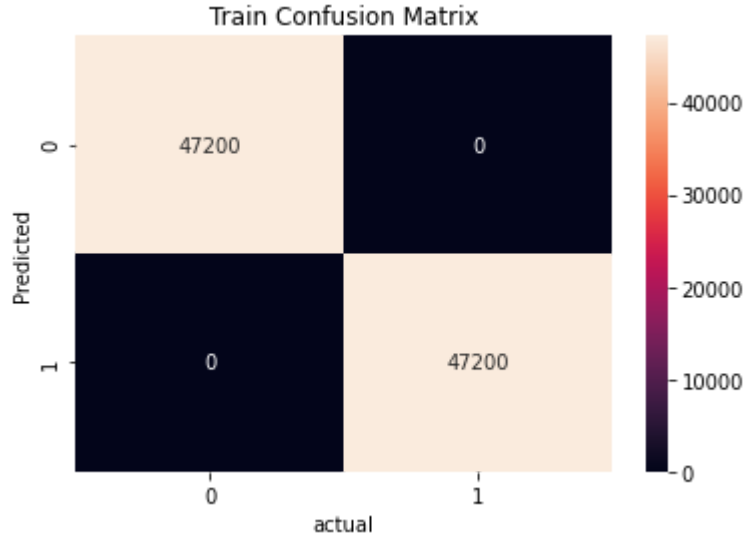
Out[205]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=1000, n_jobs=-1, num_parallel_tree=1, random_state=
0,
              reg_alpha=1e-05, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

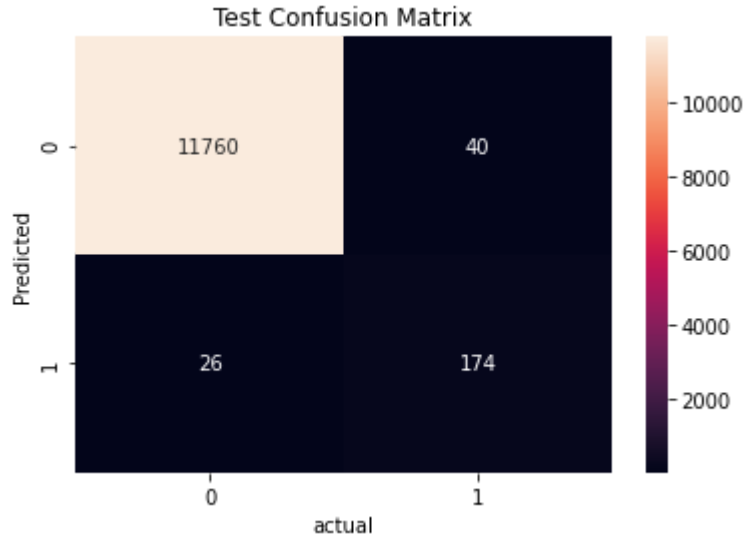
Plot AUC, Confusion Matrix and F1 Scores

In [206]:
```
y_train_pred = XGClassifier_final.predict(X_train)
y_test_pred = XGClassifier_final.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```






```
Train f1 score 1.0
Test f1 score 0.8405797101449274
```

Observations:

Train Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 100 %

Test Accuray with Hyper Parameter (max_depth=7, n_estimators=1000) is 93 %

XGBoost Classifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [207]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00     47200

    accuracy                           1.00     94400
   macro avg       1.00      1.00      1.00     94400
weighted avg       1.00      1.00      1.00     94400

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.81      0.87      0.84       200

    accuracy                           0.99     12000
   macro avg       0.91      0.93      0.92     12000
weighted avg       0.99      0.99      0.99     12000
```

In [ ]:
```
```

4.4.2 LIGHT GBM With No Sampling

In [208]:
```
X_train, X_test, y_train, y_test=X_train1.copy(), X_test1.copy(), y_train1.copy()
```

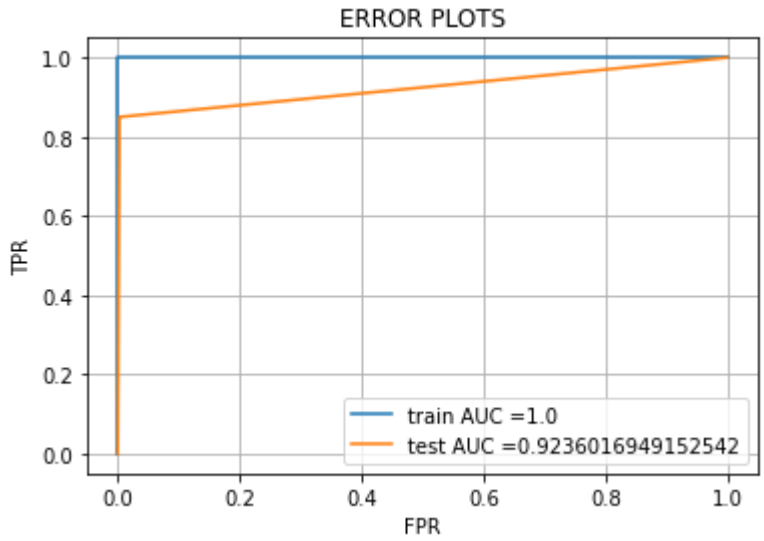
Apply ML Model with best parameters

In [209]:
```
import lightgbm as lgb
clf1=lgb.LGBMClassifier(boost_from_average=False, boosting='gbdt', max_depth=25,
               n_estimators=200, num_leaves=900, objective='binary',
               random_state=0, scale_pos_weight=59, silent=False,reg_lambda=0.28)
clf1.fit(X_train, y_train)
```

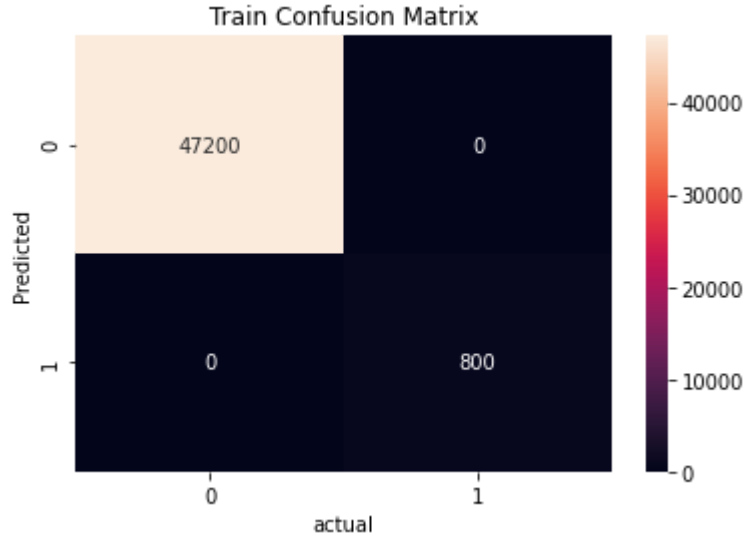
```
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignore
d. Current value: boosting=gbdt
[LightGBM] [Warning] boosting is set=gbdt, boosting_type=gbdt will be ignore
d. Current value: boosting=gbdt
[LightGBM] [Info] Number of positive: 800, number of negative: 47200
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.053795 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 36796
[LightGBM] [Info] Number of data points in the train set: 48000, number of u
sed features: 151
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

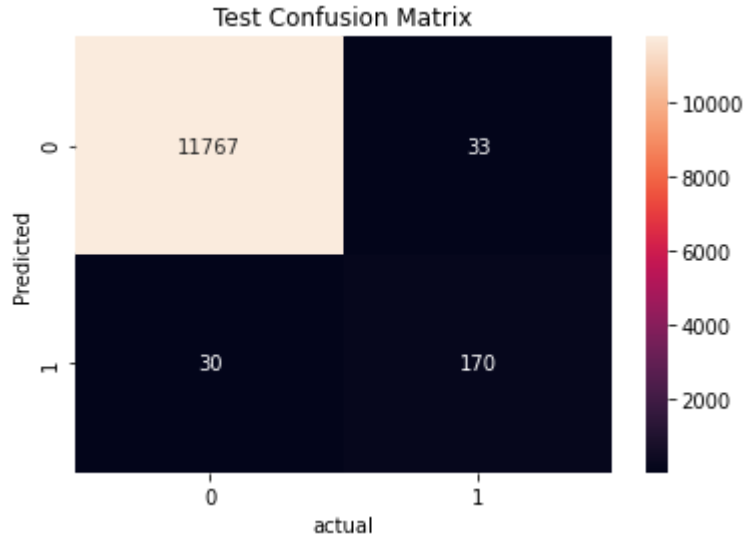
Plot AUC, Confusion Matrix and F1 Scores

In [210]:
```
y_train_pred = clf1.predict(X_train)
y_test_pred = clf1.predict(X_test)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
AUC_Train = str(auc(train_fpr, train_tpr))
AUC_Test = str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()


best_t = find_best_threshold(train_thresholds, train_fpr, train_tpr)
XGBoostClass_trainconfusion = confusion_matrix(y_train, predict_with_best_t(y_tra
df_cm1=pd.DataFrame(XGBoostClass_trainconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Train Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()


XGBoostClass_Testconfusion = confusion_matrix(y_test, predict_with_best_t(y_test_
df_cm1=pd.DataFrame(XGBoostClass_Testconfusion)
sns.heatmap(df_cm1,annot=True,fmt="d")

plt.title ("Test Confusion Matrix")
plt.ylabel("Predicted")
plt.xlabel("actual")
plt.show()

from sklearn.metrics import f1_score

F1_Score_Train=f1_score(y_train,y_train_pred)
F1_Score_Test=f1_score(y_test,y_test_pred)
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```




```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
```





```
Train f1 score 1.0
Test f1 score 0.8436724565756824
```

Observations:

Train Accuray with Hyper Parameter (max_depth=25, n_estimators=200) is 100 %

Test Accuray with Hyper Parameter (max_depth=25, n_estimators=200) is 92 %

LGBMClassifier is sensible model as TPR and TNR rates are high when compared with FNR and FPR for Train and Test confusion matrix. However there are few points which are wrongly classified

In [211]:
```
from sklearn.metrics import classification_report
print ("Train Classification Report")
print(classification_report(y_train, y_train_pred))
print ("Test Classification Report")
print(classification_report(y_test, y_test_pred))
```

```
Train Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     47200
         1.0       1.00      1.00      1.00       800

    accuracy                           1.00     48000
   macro avg       1.00      1.00      1.00     48000
weighted avg       1.00      1.00      1.00     48000

Test Classification Report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     11800
         1.0       0.84      0.85      0.84       200

    accuracy                           0.99     12000
   macro avg       0.92      0.92      0.92     12000
weighted avg       0.99      0.99      0.99     12000
```

4.4.3 Approach 4: Conclusion

In [4]:
```
from prettytable import PrettyTable
import math

Finaloutput = PrettyTable()
Finaloutput1 = PrettyTable()

Finaloutput.field_names = ["Model", "Sampling", "Train AUC Score", "Test AUC Scor

Finaloutput.add_row(["LGBMClassifier","No Sampling",100,93,100,84,92])
Finaloutput.add_row(["XGBoost Classifier","ROS",100,92,100,84,92])



print(Finaloutput)
```

```
+--------------------+-------------+-----------------+----------------+--------
--------+--------------+---------------------+
|       Model        |   Sampling  | Train AUC Score | Test AUC Score | Train F
1 Score | Test F1 Score | Test F1 Macro Score |
+--------------------+-------------+-----------------+----------------+--------
--------+--------------+---------------------+
|   LGBMClassifier   | No Sampling |       100       |       93       |      10
0       |       84      |          92         |
| XGBoost Classifier |     ROS     |       100       |       92       |      10
0       |       84      |          92         |
+--------------------+-------------+-----------------+----------------+--------
--------+--------------+---------------------+
```

5. Overall Conclusion on F1 Score

Among 4 approaches, Second approach with LGBMClassifier (SMOTE TOMEK Over Sampling) has provided best Test F1Score (0.85%), F1 Macro Score (0.93), best Test AUC Score (0.94) when compared with other approaches and ML models.

Approach 3 of XGBOOSTClassifier with ROS & SMOTE TOMEK Over Sampling has provided next best results with Test F1 Score (0.85) and F1 Macro Score (0.92) when compared with other ML models.