

Exploratory Data Analysis Equipment Failure Prediction.

Overview

This case study is regarding Conocophillips , multinational energy firm , that funds multiple energy projects in the US. According to them 80% of oil wells in the US are Stripper Wells(oil or gas well that is nearing the end of its economically useful life). These wells produce less volume but at an aggregate level are responsible for significant amount of oil production.

They have low operational costs and low capital intensity - ultimately providing a source of steady cash flow to fund operations that require more funds to get off the ground. Meaning less investment and relatively better outcomes.

The company requires these low cost wells to remain well maintained so that the cash flow remains steady .

But even mechanical and electronic equipment in any field have their shelf life and break down with time. It takes a lot investment of money and resources to get the repairs/replacement done and results in lost oil production .

The aim is to predict this equipment failure depending upon the data given from the sensors so that teams are pre prepared to handle failures as they occur.

Business Problem in machine learning terms:

Given the data points with their respective features, use classification to find out whether the data points belong to surface failure or downhole failure.

Metric to be used:

The formula for the standard FBeta-score is the Harmonic Mean

(

Harmonic Mean of a and b = $((a^{-1}) + (b^{-1})) / (\text{number of elements that is } 2)^{-1}$

)

of the precision and recall. A perfect model has an F-score of 1.

Precision = $\text{TP}/(\text{TP}+\text{FP})$

Recall = $\text{TP}/(\text{TP}+\text{FN})$

Formula :

$$\text{Fbeta} = ((1 + \text{beta}^2) * \text{Precision} * \text{Recall}) / ((\text{beta}^2 * \text{Precision}) + \text{Recall})$$

What is the difference between f.5 and f2

a) $f.5 = (1+.5^2) * (. (precision * recall) / (.5^2 * precision)+recall))$:

Means here the weight of the precision is cut to a quarter of original value. Meaning that if we divide the numerator by this lessened denominator ($.25 * \text{precision}$) then we get more overall value than the value we get if we divide the numerator by $1 * \text{Recall}$. We get more value for the f.5 score by dividing by precision that means **more weightage given to the precision (the component containing the false positive) here**.

The only difference here is between precision and recall is that of the false positive and false negative respectively.

b) $f2 = (1+2^2) * (. (precision * recall) / (2^2 * precision)+recall))$:

Means here the weight of the precision is more than quadrupled.

Meaning that if we divide the numerator by this increased denominator ($4 * \text{precision}$) then we get less overall value than the value we get if we divide the numerator by $1 * \text{Recall}$. We get more value for f2 score by dividing by recall that means **more weightage given to the recall (the component containing the false negative) here**.

So if I consider downhole failures as my positive class then I do not want that I should mistake a downhole failure for a surface failure. Meaning that here, I should not have any False Negative. Meaning I should not mistake downhole for a surface failure, so I want to reduce false negative, I will consider f2 score respectively.

In this case my priority is the prevention of downhole failures.

(They are more expensive, more impactful on failure, hazardous and less accessible for repair when they occur, difficult to handle and very less in numbers).

This means I should not confuse a downhole failure for a surface failure.

Accordingly, as the data is imbalanced , I will use the f2 to give the downhole failure , my priority.

```
In [1]: import os  
os.cpu_count()
```

```
Out[1]: 2
```

```
In [2]: !nvidia-smi
```

```
Mon Nov 2 05:25:13 2020  
+-----+  
| NVIDIA-SMI 455.32.00     Driver Version: 418.67      CUDA Version: 10.1 |  
+-----+-----+-----+  
| GPU Name      Persistence-M| Bus-Id      Disp.A  | Volatile Uncorr. ECC | |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |  
|                   |                           |          | MIG M. |  
+-----+-----+-----+-----+  
| 0  Tesla T4           Off | 00000000:00:04.0 Off |            0 | |
| N/A 44C   P8    9W / 70W |        0MiB / 15079MiB |     0%  Default |  
|                           |                           |            | ERR! |  
+-----+-----+-----+  
  
+-----+  
| Processes:  
| GPU GI CI      PID  Type  Process name          GPU Memory |  
| ID  ID          ID   ID   |                         Usage |  
+-----+  
| No running processes found  
+-----+
```

```
In [3]: !cat /usr/local/cuda/version.txt
```

```
CUDA Version 10.1.243
```

```
In [4]: !python -V
```

```
Python 3.6.9
```

For faster tSNE we use the rapids API to run tSNE and PCA on a GPU.

They directly use the GPU cores provided by both Kaggle and google colab instead of the cpus cores.

Huge increase in runtime speed.

Better not to run in kaggle else you will get stuck after a little while.

Instead run in google colab.

<https://rapids.ai/> (<https://rapids.ai/>)

```
In [5]: #For Google Colab
import pynvml
pynvml.nvmlInit()
handle = pynvml.nvmlDeviceGetHandleByIndex(0)
device_name = pynvml.nvmlDeviceGetName(handle)
if (device_name != b'Tesla T4') and (device_name != b'Tesla P4') and (device_name != b'Tesla P100-PCIE-16GB'):
    raise Exception("""
        Unfortunately this instance does not have a T4, P4 or P100 GPU.

        Please make sure you've configured Colab to request a GPU instance type.

        Sometimes Colab allocates a Tesla K80 instead of a T4, P4 or P100. Resetting the instance.

        If you get a K80 GPU, try Runtime -> Reset all runtimes...
    """)
else:
    print('Woo! You got the right kind of GPU!')

Woo! You got the right kind of GPU!
```

```
In [6]: %capture
#Use this to stop the huge output messages that come while installing.
!git clone https://github.com/rapidsai/rapidsai-csp-utils.git
!bash rapidsai-csp-utils/colab/rapids-colab.sh stable

import sys, os

dist_package_index = sys.path.index('/usr/local/lib/python3.6/dist-packages')
sys.path = sys.path[:dist_package_index] + ['/usr/local/lib/python3.6/site-packages'] + sys.path[dist_package_index:]
exec(open('rapidsai-csp-utils/colab/update_modules.py').read(), globals())

%%time #For Kaggle import sys !cp .. /input/rapids/rapids.0.15.0 /opt/conda/envs/rapids.tar.gz !cd /opt/conda/envs/ && tar -xzvf rapids.tar.gz >
```

```
/dev/null sys.path = ["/opt/conda/envs/rapids/lib/python3.7/site-packages"] + sys.path sys.path = ["/opt/conda/envs/rapids/lib/python3.7"] +  
sys.path sys.path = ["/opt/conda/envs/rapids/lib"] + sys.path !cp /opt/conda/envs/rapids/lib/libxgboost.so /opt/conda/lib/
```

```
In [7]: import pandas.testing as tm  
import cudf, cuml  
from cuml import PCA  
from cuml.decomposition import PCA  
from cuml.manifold import TSNE  
import seaborn as sns  
import matplotlib.pyplot as plt  
%matplotlib inline
```

```
In [8]: import seaborn as sns  
import matplotlib.pyplot as plt  
%matplotlib inline
```

```
In [9]: from google.colab import drive  
drive.mount("/content/gdrive")
```

Mounted at /content/gdrive

```
In [10]: import numpy as np  
import pandas as pd
```

```
In [ ]: #train_n_test=pd.read_csv("equip_failures_training_set.csv")  
#train_n_test=pd.read_csv("/content/gdrive/My Drive/equip_failures_training_set.csv")  
#train_n_test=pd.read_csv("../input/yellow/equip_failures_training_set.csv")  
#train_n_test=pd.read_csv("equip_failures_training_set.csv")
```

Find the duplicates . Duplicate data points are of no value.

```
In [ ]: train_n_test.drop("id",axis=1,inplace=True)  
train_n_test.drop_duplicates(keep='first', inplace=True)
```

```
In [ ]: train_n_test.columns.values[0:5]
```

```
Out[9]: array(['target', 'sensor1_measure', 'sensor2_measure', 'sensor3_measure',
   'sensor4_measure'], dtype=object)
```

```
In [ ]: len(train_n_test.columns.values)
```

```
Out[10]: 171
```

```
In [ ]: train_n_test.target.value_counts()
```

```
Out[11]: 0    59000
1     1000
Name: target, dtype: int64
```

The data is highly imbalanced.

Meaning most of the failures are surfaces related failures.

```
In [ ]: train_n_test.head(5)
```

```
Out[12]:
```

	target	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	
0	0	76698	na	2130706438	280	0	0	0	0
1	0	33058	na	0	na	0	0	0	0
2	0	41040	na	228	100	0	0	0	0
3	0	12	0	70	66	0	10	0	0
4	0	60874	na	1368	458	0	0	0	0

5 rows × 171 columns

Is there np.Nan at all or all the cells have been marked as na strings?

```
In [ ]: for column in list(train_n_test.columns.values):
    if np.NaN in train_n_test[column]:
        print(column, " : ", np.NaN in train_n_test[column])
```

```
In [ ]: counter=0
for ele in train_n_test.isnull().any(): # train.isnull().any() returns a list of whether any value is null.
    if ele==True:
        counter=counter+1
print(counter,"null Values we have.")
```

0 null Values we have.

All cells have been marked as na strings only.

Find out the different data types in the table columns given.

```
In [ ]: xx={}
for column in list(train_n_test.columns.values):
    x={}
    for ele in train_n_test[column]:
        if type(ele) in x:
            x.update({type(ele): x[type(ele)]+1})
        else :
            x.update({type(ele): int(1)})
    xx.update({column: x})
    del x
list(xx.items())[0:8]
```

```
Out[15]: [('target', {int: 60000}),
('sensor1_measure', {int: 60000}),
('sensor2_measure', {str: 60000}),
('sensor3_measure', {str: 60000}),
('sensor4_measure', {str: 60000}),
('sensor5_measure', {str: 60000}),
('sensor6_measure', {str: 60000}),
('sensor7_histogram_bin0', {str: 60000})]
```

Observation : Except the first 2 features rest of them are in a string format even if they are the float values.

What is the meaning of na values in this case ?

na here means not available.

Following are the reasons :

- 1) The sensor did not get activated on a particular stimulus as it responds to a different stimulus.Hence the reading na.
- 2) Sensor damage due to over stimulus.

There does not seem to be np.NaN used here, instead it is na as in not available in string format.

All the features that are in a string format (<class 'str'>) Have the value of "na" in them.

Why to use np.NaN in the case that there is a string given as "na" ?

Because in case we want to add multiply etc take average the string na will not help.

But if we convert the string nan to np.NaN we can use numpy to add the other values and ignore this np.NaN
np.sum() takes a np.NaN value as 0 and adds while taking the sum.

```
In [ ]: # Replace the string that is "na" in the features by np.NaN
for column in list(train_n_test.columns.values):
    train_n_test[column]=train_n_test[column].replace('na', np.NaN)
```

```
In [ ]: # Convert the rest of the non numeric elements in the format of a string to a numerical feature.
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_numeric.html
train_n_test=train_n_test.apply(pd.to_numeric,downcast='float')
```

```
In [ ]: train_n_test.dtypes[0:5]
```

```
Out[18]: target      float32
sensor1_measure   float32
sensor2_measure   float32
sensor3_measure   float32
sensor4_measure   float32
dtype: object
```

Dividing the train data set into train and test . I need more data to feed into my model.I will not take the cross validation data as I will perform randomsearchcv and gridseachcv

```
In [ ]: y_train_n_test=train_n_test.pop("target")
```

```
In [ ]: from sklearn.model_selection import train_test_split
train, test, y_train, y_test = train_test_split(train_n_test, y_train_n_test, test_size=0.20, stratify=y_trai
In [ ]: def reset_index(x,y):
    x["target"] = y
    x.reset_index(drop=True, inplace=True)
    y = x.pop("target")
    return x,y
In [ ]: # ignoring SettingWithCopyWarning
pd.options.mode.chained_assignment = None
train, y_train = reset_index(train, y_train)
test, y_test = reset_index(test, y_test)
In [ ]: #train.to_pickle("train.pickle")
#y_train.to_pickle("y_train.pickle")
#test.to_pickle("test.pickle")
#y_test.to_pickle("y_test.pickle")
In [ ]: #train=pd.read_pickle("/content/gdrive/My Drive/train.pickle")
#y_train=pd.read_pickle("/content/gdrive/My Drive/y_train.pickle")
#test=pd.read_pickle("/content/gdrive/My Drive/test.pickle")
#y_test=pd.read_pickle("/content/gdrive/My Drive/y_test.pickle")
```

```
In [ ]: xx={}
for column in list(train.columns.values):
    x={}
    for ele in train[column]:
        if type(ele) in x:
            x.update({type(ele): x[type(ele)]+1})
        else :
            x.update({type(ele): int(1)})
    xx.update({column: x})
    del x
list(xx.items())[0:8]
```

```
Out[25]: [('sensor1_measure', {float: 48000}),
('sensor2_measure', {float: 48000}),
('sensor3_measure', {float: 48000}),
('sensor4_measure', {float: 48000}),
('sensor5_measure', {float: 48000}),
('sensor6_measure', {float: 48000}),
('sensor7_histogram_bin0', {float: 48000}),
('sensor7_histogram_bin1', {float: 48000})]
```

All the values that you see as <class 'float'> are np.NaNs although na stands for Not Available. For our ease of use we converted them into np.NaN

Find out for every data point if it is np.NaN or not. If np.NaN then 1 else 0

```
In [ ]: counter=0
for ele in train["sensor2_measure"]:
    if np.isnan(ele):
        counter=counter+1
counter
```

```
Out[26]: 37156
```

Here I create a nan dataframe that holds the value for the particular feature in our dataframe.
**if nan then 1
if not nan then 0**

```
In [ ]: %%time
d={}
for column in train.columns.values:
    temp_list=[]
    for row in train.index.values:
        if np.isnan(train[column][row]):
            temp_list.append(1.0)
        else:
            temp_list.append(0.0)
    d.update({column:temp_list})
nan_or_not=pd.DataFrame.from_dict(d,dtype="float32")
```

CPU times: user 1min 33s, sys: 11.5 ms, total: 1min 33s
Wall time: 1min 33s

```
In [ ]: #nan_or_not.to_pickle("nan_or_not.pickle")
#nan_or_not=pd.read_pickle("/content/gdrive/My Drive/nan_or_not.pickle")
#nan_or_not=pd.read_pickle("//content/gdrive/My Drive//content/gdrive/My Drive/nan_or_not.pickle")
```

1) Get an idea as to the readings could be from what all sensors.

2) Start the basic EDA .

3) Try to figure out if sensors have NaN values in both the target values or unique to one target value only.

Overall np.NaN values feature wise

```
In [ ]: #np.isnan(dataframe).sum()
#dataframe.isnull().sum().sum()
#np.isinf(dataframe)
#dataframe.dropna()
#dataframe.dropna().isnull().sum().sum()
#dataframe.dropna().to_numpy().shape
```

```
In [ ]: train["target"]=y_train
```

Compare the target 0 and target 1 np.NaN values feature wise

```
In [ ]: column_list=[]
nan_ratio=[]
one_zero=[]

for column in list(train.columns.values):
    if column!="target":
        column_list.append(column)
        one_zero.append(0)
        nan_ratio.append(train[column][train.target==0].isnull().sum()/len(train[column][train.target==0]))
        column_list.append(column)
        one_zero.append(1)
        nan_ratio.append(train[column][train.target==1].isnull().sum()/len(train[column][train.target==1]))
```

Compare the features with the highest na to total ratio descending sorted by class 0

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2==0])
d={i:nan_ratio[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

```
In [ ]: new_column_list=[]
new_nan_ratio=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i])
    new_nan_ratio.append(nan_ratio[i])
    new_one_zero.append(one_zero[i])
    new_column_list.append(column_list[i+1])
    new_nan_ratio.append(nan_ratio[i+1])
    new_one_zero.append(one_zero[i+1])
```

```
In [ ]: #https://seaborn.pydata.org/generated/seaborn.barplot.html
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
rng = np.arange(0, 1.1, .1)

for i in range(0,len(column_list),40):
    plt.figure(figsize=(15,5))
    my_plot=sns.barplot(x=new_column_list[i:i+40], y=new_nan_ratio[i:i+40], hue=new_one_zero[i:i+40])
    my_plot.set_yticks(rng)

    for p in my_plot.patches:
        width = p.get_width()
        height = p.get_height()
        x, y = p.get_xy()
        my_plot.annotate('{:.2%}'.format(height), (p.get_x() + .5*width, p.get_y() + height + 0.01), va = 'bottom')

    plt.title("Percent of NaN in a target value descending sorted by class 0\n", size=14)
    plt.xticks(rotation=90)
    plt.ylabel("% of Nan in a target", size=13)
    plt.xlabel("Feature", size=13)
    plt.show()
```

Percent of NaN in a target value descending sorted by class 0

Compare the features with the highest na to total ratio sorted by class 1

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:nan_ratio[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

```
In [ ]: new_column_list=[]
new_nan_ratio=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_nan_ratio.append(nan_ratio[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_nan_ratio.append(nan_ratio[i])
    new_one_zero.append(one_zero[i])
```

```
In [ ]: #https://seaborn.pydata.org/generated/seaborn.barplot.html
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
rng = np.arange(0, 1.1, .1)

for i in range(0,len(column_list),40):
    plt.figure(figsize=(15,5))
    my_plot=sns.barplot(x=new_column_list[i:i+40], y=new_nan_ratio[i:i+40], hue=new_one_zero[i:i+40])
    my_plot.set_yticks(rng)

    for p in my_plot.patches:
        width = p.get_width()
        height = p.get_height()
        x, y = p.get_xy()
        my_plot.annotate('{:.2%}'.format(height), (p.get_x() + .5*width, p.get_y() + height + 0.01), va = 'bottom')

    plt.title("Percent of NaN in a target value descending sorted by class 1\n", size=14)
    plt.xticks(rotation=90)
    plt.ylabel("% of Nan in a target", size=13)
    plt.xlabel("Feature", size=13)
    plt.show()
```

Percent of NaN in a target value descending sorted by class 1

Observation :

- 1) Observe that sensors like sensor 58,65,4,56,57 have more np.NaN values for downhole failures and less nans for surface related failures that there is a chance that these sensors are more related to a particular types of surface failures.
- 2) Whereas sensors like 43,42,41,40,39,38,37 have more np.NaN values for surface failures and lesser nans for downhole failures meaning that there is a chance that these sensors are more related to a particular types of downhole failures.

Which target has the greatest number of NaNs ?

```
In [ ]: counter={}
i=j=0
for column in list(train.columns.values):
    if (train[column][train.target==1].isnull().sum()/len(train[column][train.target==1]))>=train[column][train.target==0].isnull().sum()/len(train[column][train.target==0])):
        i=i+1
        counter.update({"Downhole Failure": i})
    elif (train[column][train.target==0].isnull().sum()/len(train[column][train.target==0]))>=train[column][train.target==1].isnull().sum()/len(train[column][train.target==1])):
        j=j+1
        counter.update({"Surface Failure": j})
```

```
In [ ]: counter
```

```
Out[40]: {'Downhole Failure': 88, 'Surface Failure': 83}
```

Observation : We have more sensors with np.Nans for Downhole Features. Meaning 88 sensors have more np.Nans for Downhole Failures and 83 sensors have more np.Nans for Surface Failures.

Target with greatest number of NaNs for only the measure sensors after removing the histogram features

```
In [ ]: counter={}
i=j=0
for column in list(train.columns.values):
    if column.find("measure")!=-1:
        if (train[column][train.target==1].isnull().sum()/len(train[column][train.target==1]))
        >=train[column][train.target==0].isnull().sum()/len(train[column][train.target==0])):
            i=i+1
            counter.update({"Downhole Failure": i})
        elif (train[column][train.target==0].isnull().sum()/len(train[column][train.target==0]))
        >=train[column][train.target==1].isnull().sum()/len(train[column][train.target==1])):
            j=j+1
            counter.update({"Surface Failure": j})
```

```
In [ ]: counter
```

```
Out[42]: {'Downhole Failure': 87, 'Surface Failure': 13}
```

Observation : If we consider only the measure sensors, then a lot more np.NaNs are present for Downhole Features that is 87 sensors have more np.NaN for downhole failures whereas 13 sensors have more np.NaN for surface failures. Meaning the measure sensors maybe favouring the Surface Failures more.

Target with greatest number of NaNs for only the histogram sensors after removing the measure features

```
In [ ]: counter={}
i=j=0
for column in list(train.columns.values):
    if column.find("histogram")!=-1:
        if (train[column][train.target==1].isnull().sum()/len(train[column][train.target==1]))
        >=train[column][train.target==0].isnull().sum()/len(train[column][train.target==0])):
            i=i+1
            counter.update({"Downhole Failure": i})
        elif (train[column][train.target==0].isnull().sum()/len(train[column][train.target==0]))
        >=train[column][train.target==1].isnull().sum()/len(train[column][train.target==1])):
            j=j+1
            counter.update({"Surface Failure": j})
```

```
In [ ]: counter
```

```
Out[44]: {'Surface Failure': 70}
```

Observation : If we consider only the histogram sensors, then all of the histogram sensors have more np.NaN values for Surface Failure. Meaning , there is a chance that histogram features favour the Downhole Failures more.

Using histograms, plots find out the nature of means for features for both the classes and thereby, the outliers.

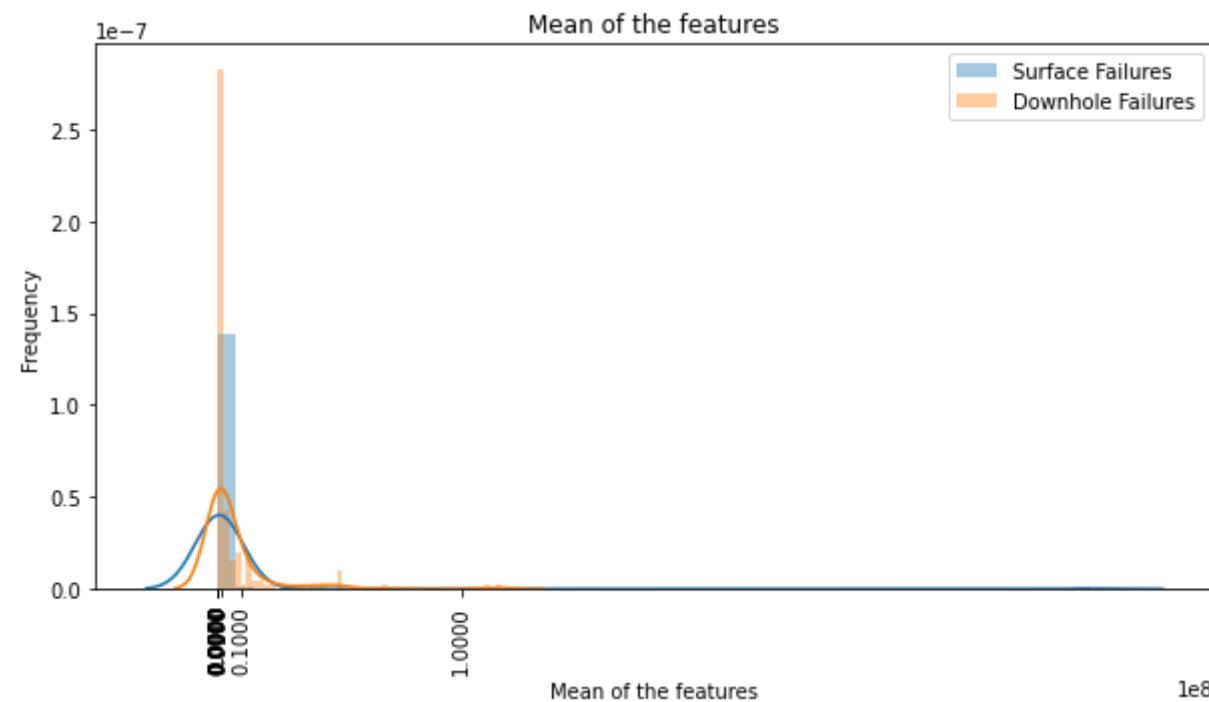
```
In [ ]: #Find the mean.  
sample=train.copy()
```

```
In [ ]: mean_0=[]  
mean_1=[]  
for column in list(sample.columns.values):  
    mean_0.append(train[train.target==0][column].mean())  
    mean_1.append(train[train.target==1][column].mean())
```

```
In [ ]: median_0=[]  
median_1=[]  
for column in list(sample.columns.values):  
    median_0.append(train[train.target==0][column].median())  
    median_1.append(train[train.target==1][column].median())
```

10) Try and plot it's histogram for both the target values.

```
In [ ]: import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Mean of the features')
plt.ylabel('Frequency')
plt.xlabel('Mean of the features')
plt.xticks([10**e for e in range(0,9)], rotation=90)
plt.legend()
plt.show()
```

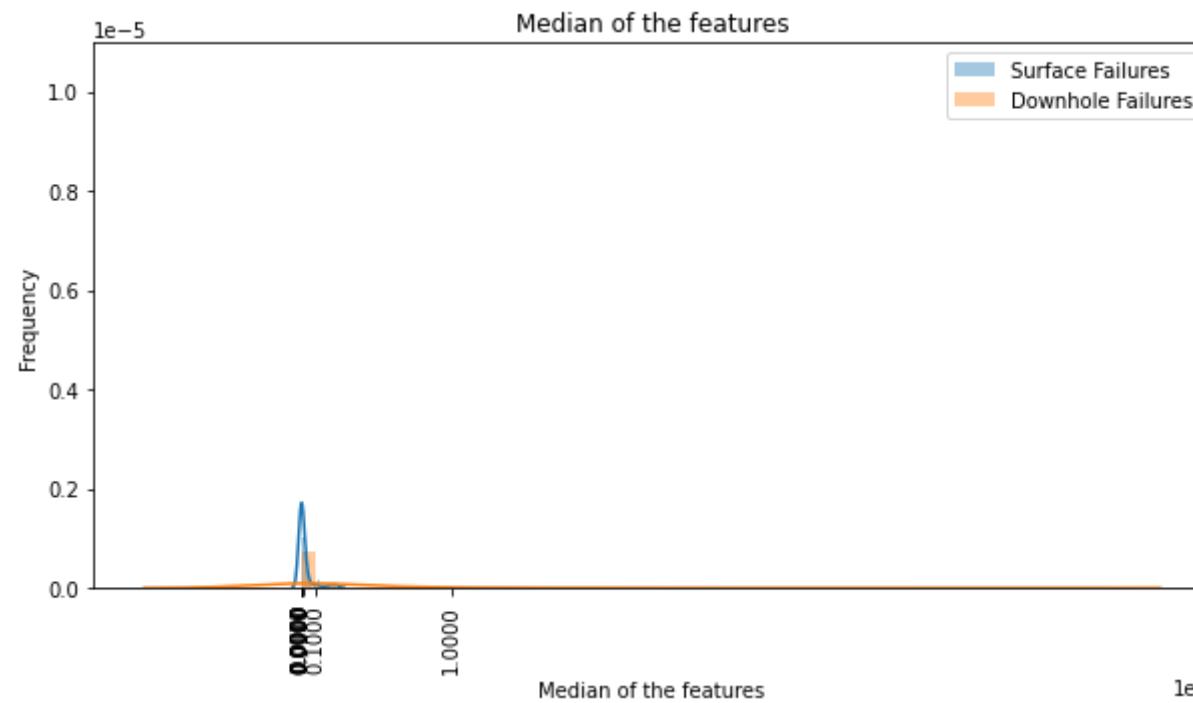


Observations :

Majority of both the surface and downhole failures have mean sensor readings 1000 and we see that only small values get a mean till 1e8 that also surface level failures .

As the mean value is susceptible to outliers , we are getting a high value but we cannot be sure of this.
So we will try the median value as well.

```
In [ ]: import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('Median of the features')
plt.ylabel('Frequency')
plt.xlabel('Median of the features')
plt.xticks([10**e for e in range(0,8)]), rotation=90
plt.legend()
plt.show()
```

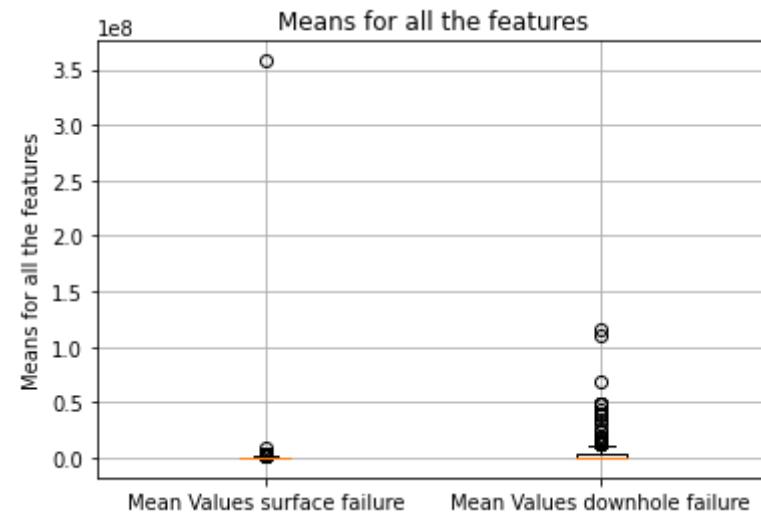


Observations :

Looking at the median values it can be said that the majority of the surface related failures are in the range of 0-1000 but we can see that the downhole failure values have median values till e^7 which is huge whereas this does not happen for the surface related failures.

11) Try box plot / violin plot to visualize the outliers.

```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Means for all the features')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the features')  
plt.grid()  
plt.show()
```

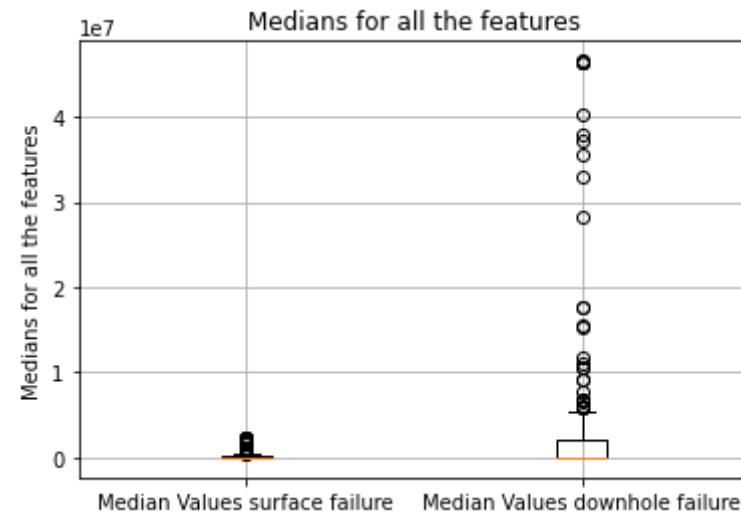


Observations :

The mean values for the downhole failures are more extreme than that of the surface level failures.

More detail can be revealed when we plot a bar plot for both the classes feature wise.

```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('Medians for all the features')  
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))  
plt.ylabel('Medians for all the features')  
plt.grid()  
plt.show()
```



Observations :

The mean values can be affected by the high values or outliers that is why we try the median values as well.
Above are the median values for all the features.

The above diagram shows that there is more extreme or high value data for downhole failures.

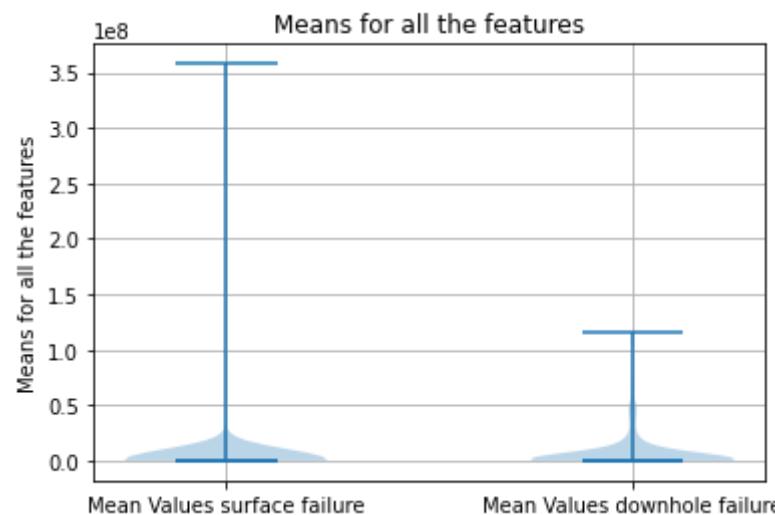
These are not outliers as these are the medians for each feature. It means that the original values for the features are high themselves.

Meaning whenever the downhole failures occur, the above sensor data (which we will find out later) is high and such high occurrences do not occur when it is a surface related failure.

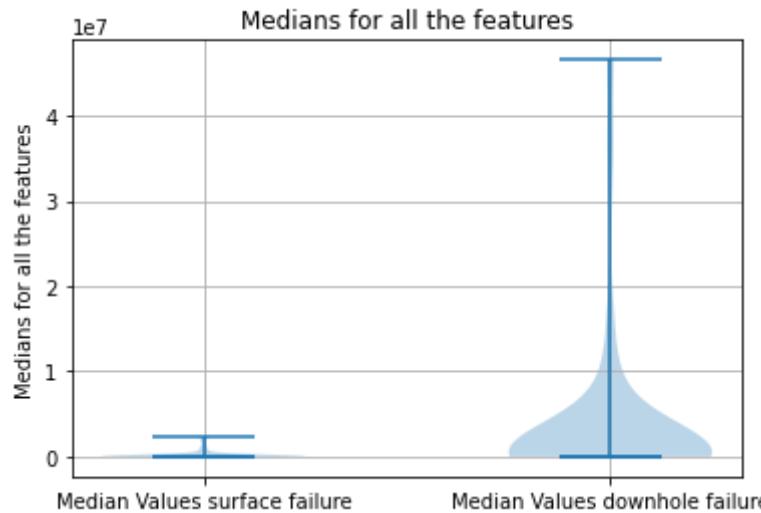
More detail can be revealed when we plot a bar plot for both the classes feature wise.

We can see a similar state in the violin plots as well.

```
In [ ]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Means for all the features')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the features')
plt.grid()
plt.show()
```



```
In [ ]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('Medians for all the features')
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))
plt.ylabel('Medians for all the features')
plt.grid()
plt.show()
```



Compare the median of the available features without imputing, sorted by class 0

Without imputing any value, find out the mean and median values for the features for each class and observe which all features can separate both the classes because we assume some features to have different mean and median values for the two different classes given here.

```
In [ ]: y_train=train.pop("target")
```

```
In [ ]: from sklearn.preprocessing import StandardScaler
train_standardized=pd.DataFrame()
for column in list(train.columns.values):
    scalar = StandardScaler()
    #Convert into column vector that is many rows but only one column.
    scalar.fit(train[column].values.reshape(-1,1))
    train_standardized[column] = scalar.transform(train[column].values.reshape(-1,1)).flatten()
del scalar
```

```
In [ ]: train_standardized["target"] = y_train
```

```
In [ ]: train_standardized.head()
```

Out[58]:

	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor
0	-0.209185	NaN	-0.447114	NaN	-0.058784	-0.057656		-0.010516
1	0.226385	NaN	2.236571	0.220177	-0.058784	-0.057656		-0.010516
2	-0.411601	-0.19623	-0.447113	-0.114751	-0.058784	-0.057656		-0.010516
3	0.027280	NaN	-0.447112	0.243406	-0.058784	-0.057656		-0.010516
4	-0.132515	NaN	-0.447113	0.006796	-0.058784	-0.057656		-0.010516

5 rows × 171 columns

I have been getting the mean and the median in negatives if I standardize. Here I am not taking the absolute value as the mean of the absolute values is different than that of the values that are not so this will create a problem later on.

Find the features where the difference between the means of the two classes is more than .15
sorted by class 1

```
In [ ]: column_list=[]
mean_list=[]
one_zero=[]
for column in list(train_standardized.columns.values):

    if column!="target":

        column_list.append(column)
        one_zero.append(0)
        mean_list.append(train_standardized[train_standardized.target==0][column].mean())

        column_list.append(column)
        one_zero.append(1)
        mean_list.append(train_standardized[train_standardized.target==1][column].mean())
```

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:mean_list[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

```
In [ ]: counter=0
mean_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.
    if (np.abs(np.abs(mean_list[i])-np.abs(mean_list[i-1]))>=.15):
        mean_higher_reading.append(mean_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

        mean_higher_reading.append(mean_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between median readings of surface and downhole failures.")
```

So we have approximate 159 sensor that show significant difference (> .15) between median readings of surface and downhole failures.

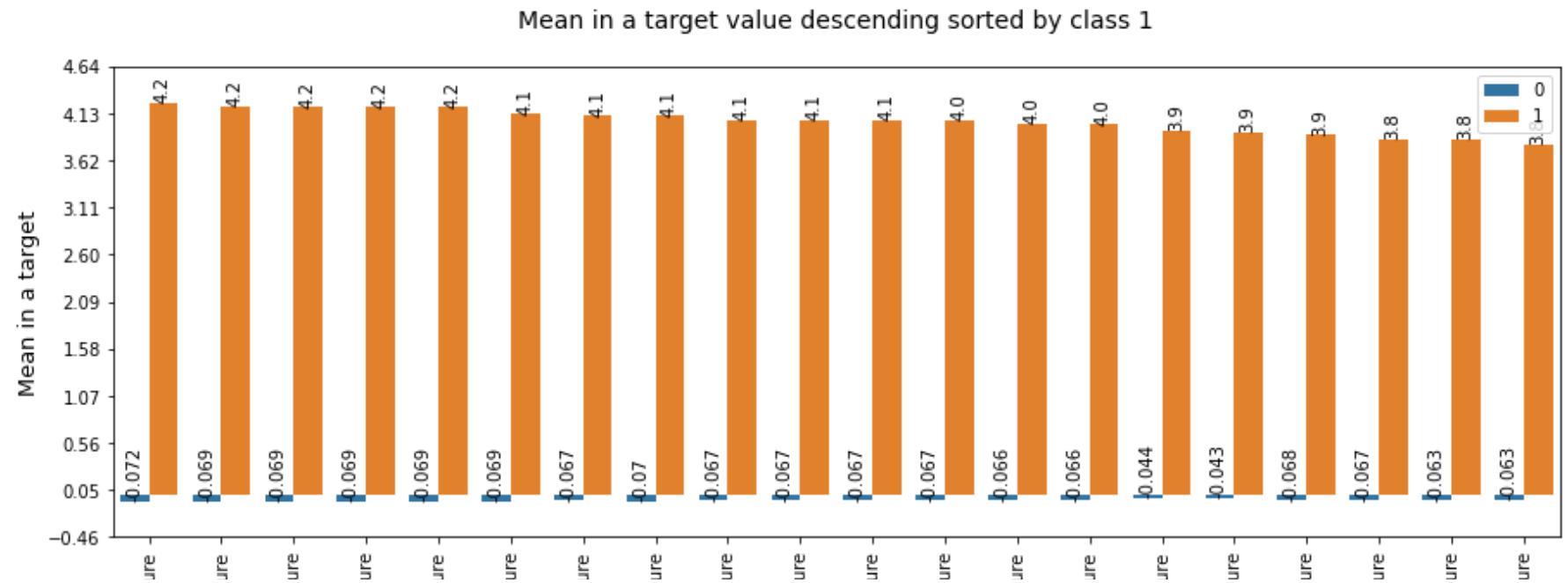
```
In [36]: def plot_diagram(column_list,content_list,one_zero_list,mean_or_median,zero_one):
    #https://seaborn.pydata.org/generated/seaborn.barplot.html
    import seaborn as sns
    import matplotlib.pyplot as plt
    %matplotlib inline
    #rng = np.arange(0, max(content_list)+.5, .5)
    try:
        rng = np.arange(min(content_list), max(content_list)+(max(content_list)/5),
                        round((max(content_list)+(max(content_list)/5))/10,2))
    except ZeroDivisionError:
        try:
            rng = np.arange(min(content_list), max(content_list)+(max(content_list)/5),
                            10*(round((max(content_list)+(max(content_list)/5))/10,3)))
        except ZeroDivisionError:
            rng = np.arange(min(new_median_list), max(new_median_list)+(max(new_median_list)/5),
                            1*(round(((max(new_median_list)+(max(new_median_list)/5))-min(new_median_list))/5,
                                     1)))
    for i in range(0,len(column_list),40):
        plt.figure(figsize=(15,5))
        my_plot=sns.barplot(x=column_list[i:i+40], y=content_list[i:i+40], hue=one_zero_list[i:i+40])
        my_plot.set_yticks(rng)

        for p in my_plot.patches:
            width = p.get_width()
            height = p.get_height()
            x, y = p.get_xy()
            my_plot.annotate('{:.2}'.format(height), (p.get_x()+.5*width, p.get_y() + height + 0.01), va = 'bottom',
                            ha = 'center')

            title=mean_or_median+" in a target value descending sorted by class "+str(zero_one)+"\n"
            x_label="Feature"
            y_label=mean_or_median+" in a target"

            plt.title(title, size=14)
            plt.xticks(rotation=90)
            plt.ylabel(y_label, size=13)
            plt.xlabel(x_label, size=13)
            plt.show()
```

```
In [ ]: plot_diagram(column_higher_reading,mean_higher_reading,one_zero_higher_reading,"Mean",1)
```



Observation : If I do not perform any imputation then I get measure sensors like 59,27,47 etc whose mean for the Downhole failures is a lot more than that of surface failures.

So if my data point is not an outlier then this means that based on the mean value of the sensor I can find out if my data point belongs to a surface failure or a downhole failure.

Compare the median of the available features without imputing, sorted by class 1

```
In [ ]: column_list=[]
median_list=[]
one_zero=[]
for column in list(train_standardized.columns.values):

    if column!="target":

        column_list.append(column)
        one_zero.append(0)

        median_list.append(train_standardized[train_standardized.target==0][column].median())

        column_list.append(column)
        one_zero.append(1)
        median_list.append(train_standardized[train_standardized.target==1][column].median())
```

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:median_list[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

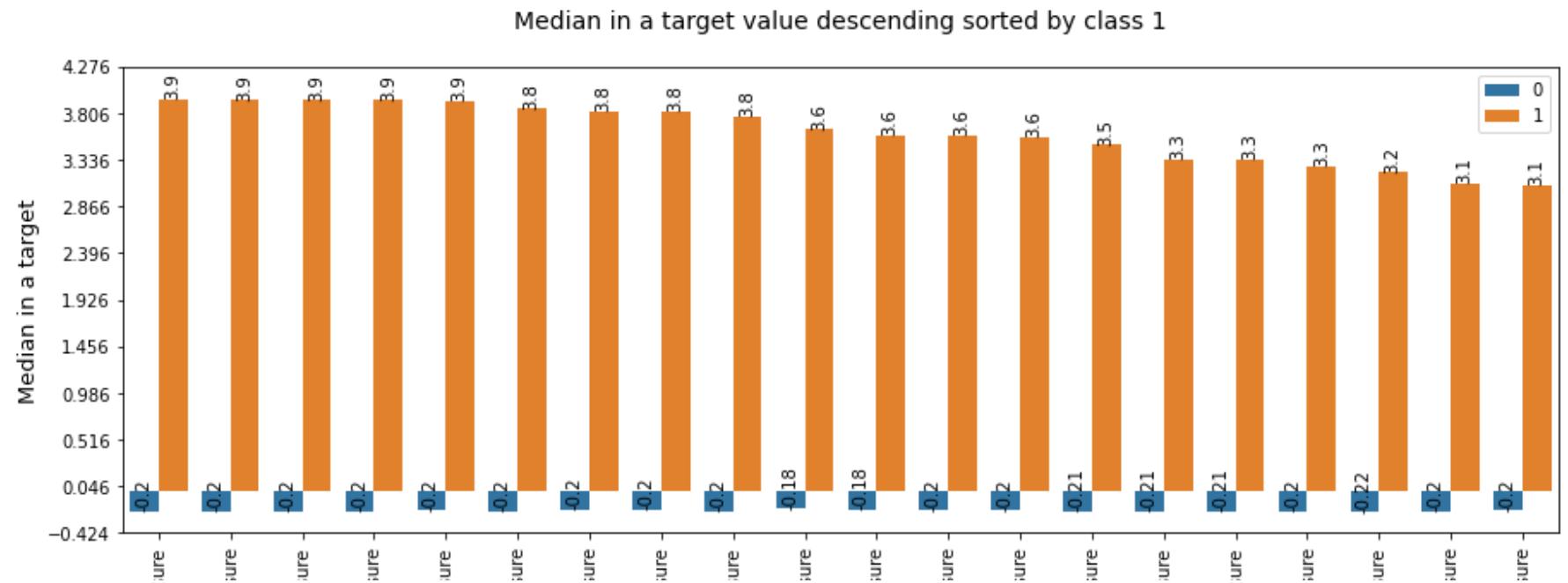
```
In [ ]: counter=0
median_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.
    if (np.abs(np.abs(median_list[i])-np.abs(median_list[i-1]))>=.15):
        median_higher_reading.append(median_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

        median_higher_reading.append(median_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between median readings of surface and downhole failures.")
```

So we have approximate 86 sensor that show significant difference (> .15) between median readings of surface and downhole failures.

In []: `plot_diagram(column_higher_reading,median_higher_reading,one_zero_higher_reading,"Median",1)`



Observations :

The median values of the downhole failures are way higher than that of the surface value failures. From this , we can also figure out actually which features contribute to the downhole failures and which features contribute to the surface ones.

Given a point is not an outlier then we can also estimate whether a data point will belong to the surface failure or the downhole failure based on the median value.

Perform PCA find which feature gives the highest variance . Try the same with tSNE.

1) Impute np.NaN with 0 values with all features.

In []: `data_type="0_impute_data"`

```
In [ ]: train_0_all_features=train.copy()  
y_train_0_all_features=y_train.copy()  
test_0_all_features=test.copy()  
y_test_0_all_features=y_test.copy()
```

```
In [ ]: train_0_all_features.shape
```

```
Out[84]: (48000, 170)
```

```
In [ ]: y_train_0_all_features.shape
```

```
Out[85]: (48000,)
```

```
In [ ]: test_0_all_features.shape
```

```
Out[86]: (12000, 170)
```

```
In [ ]: y_test_0_all_features.shape
```

```
Out[87]: (12000,)
```

```
In [ ]: for i,j in zip(train_0_all_features.columns.values,test_0_all_features.columns.values):  
    if (i!=j):  
        print(i," : ",j)
```

Filling the np.NaN values with 0 this time.

```
In [ ]: train_0_all_features=train_0_all_features.fillna(0)
train_0_all_features.head(5)
```

Out[89]:

	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor
0	28906.0	0.0	0.000000e+00	0.0	0.0	0.0	0.0	0.0
1	91000.0	0.0	2.130706e+09	1268.0	0.0	0.0	0.0	0.0
2	50.0	0.0	4.200000e+01	28.0	0.0	0.0	0.0	0.0
3	62616.0	0.0	1.488000e+03	1354.0	0.0	0.0	0.0	0.0
4	39836.0	0.0	5.020000e+02	478.0	0.0	0.0	0.0	0.0

5 rows × 170 columns

```
In [ ]: test_0_all_features=test_0_all_features.fillna(0)
test_0_all_features.head(5)
```

Out[90]:

	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor
0	63284.0	0.0	0.000000e+00	0.0	0.0	0.0	0.0	0.0
1	120196.0	0.0	2.130706e+09	814.0	0.0	0.0	0.0	0.0
2	8.0	0.0	6.000000e+00	6.0	0.0	0.0	0.0	0.0
3	86.0	0.0	2.130706e+09	8.0	0.0	0.0	0.0	0.0
4	61396.0	0.0	2.130706e+09	512.0	0.0	0.0	0.0	0.0

5 rows × 170 columns

**Reset Indices for both the train and x's and y's.
Do this always before standardizing and mean/median imputing**

```
In [ ]: train_0_all_features,y_train_0_all_features=reset_index(train_0_all_features,y_train_0_all_features)
test_0_all_features,y_test_0_all_features=reset_index(test_0_all_features,y_test_0_all_features)
```

Standardise the dataframe

```
In [ ]: import pickle
from sklearn.preprocessing import StandardScaler
def standardize_test_n_train_wrt_train(train_df,test_df,data_type):
    train_dataframe=pd.DataFrame()
    test_dataframe=pd.DataFrame()

    scalar_dict={}

    for column in list(train_df.columns.values):

        scalar = StandardScaler()

        #Convert into column vector that is many rows but only one column.
        scalar.fit(train_df[column].values.reshape(-1,1))

        scalar_dict.update({column:scalar})

        train_dataframe[column] = scalar.transform(train_df[column].values.reshape(-1,1)).flatten()
        test_dataframe[column] = scalar.transform(test_df[column].values.reshape(-1,1)).flatten()
        del scalar

    pickle.dump(scalar_dict, open("scalar_dict_"+str(data_type)+".pkl",'wb'))

    return train_dataframe,test_dataframe
```

```
In [ ]: standard_train_0_all_features_df,standard_test_0_all_features_df=standardize_test_n_train_wrt_train(train_0_
```

```
In [ ]: standard_train_0_all_features_df.shape
```

```
Out[94]: (48000, 170)
```

```
In [ ]: standard_test_0_all_features_df.shape
```

```
Out[95]: (12000, 170)
```

Truncated SVD to get the best features

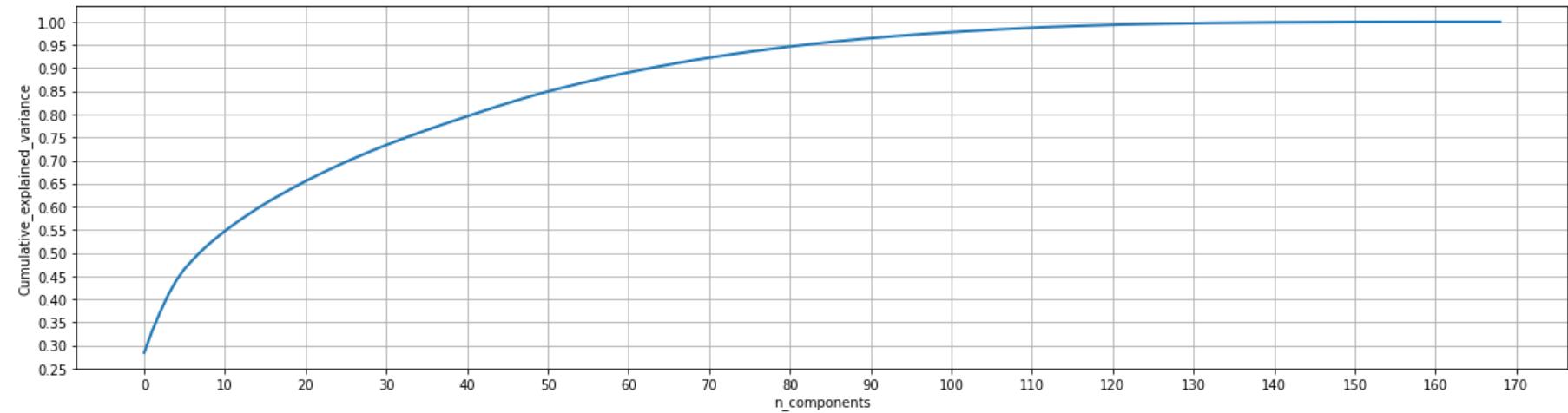
```
In [ ]: #https://stackoverflow.com/questions/50796024/feature-variable-importance-after-a-pca-analysis
```

```
In [ ]: %%time
import time
from sklearn.decomposition import TruncatedSVD
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
def truncated_svd(input_dataframe,n_components):
    svd = TruncatedSVD(n_components=n_components)
    svd.fit(input_dataframe)
    percentage_var_explained = svd.explained_variance_ / np.sum(svd.explained_variance_)
    cum_var_explained = np.cumsum(percentage_var_explained)

    # Plot the SVD spectrum
    plt.figure(1, figsize=(20, 5))
    plt.clf()
    plt.plot(cum_var_explained, linewidth=2)
    plt.axis('tight')
    plt.grid()
    plt.yticks(np.arange(0.25, 1.01, .05))
    plt.xticks(np.arange(0, 172, 10))
    plt.xlabel('n_components')
    plt.ylabel('Cumulative_explained_variance')
    plt.show()
    return svd
```

```
CPU times: user 14.6 ms, sys: 11.6 ms, total: 26.3 ms
Wall time: 25.1 ms
```

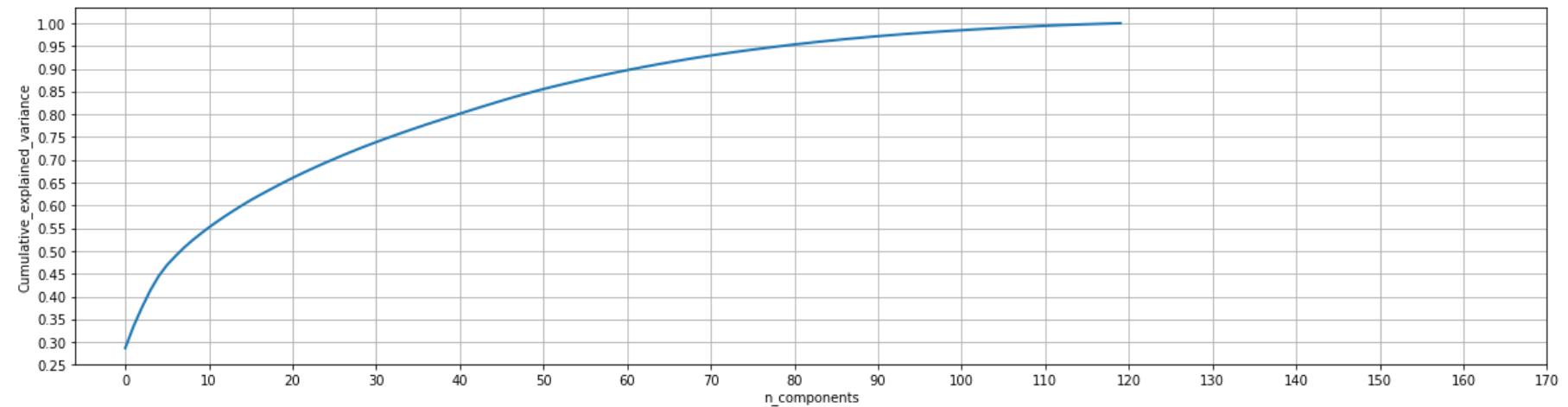
In []: `truncated_svd(standard_train_0_all_features_df, 169)`



Out[98]: `TruncatedSVD(n_components=169)`

I choose 120 components to describe my data as 120 components explain atleast 99% variance of the total data.

```
In [ ]: svd=truncated_svd(standard_train_0_all_features_df,120)
```



For the 120 principal components using svd.explained_variance_ratio_ find the variance distribution.

```
In [ ]: print(svd.explained_variance_ratio_)
```

#Here the highest ratio is of the Principal Component 1 and the lowest value is of Principal Component 120.
#this is the sorted list of all the principal components.

```
[0.28472978 0.04780687 0.04091736 0.0371289 0.03108652 0.02423595  
 0.01922317 0.01806965 0.01594909 0.01458631 0.01395989 0.01301183  
 0.01240924 0.01175223 0.01140193 0.01093418 0.01020836 0.00980257  
 0.0096912 0.00950576 0.00917567 0.00882468 0.00853742 0.00836943  
 0.00804615 0.0077958 0.00775106 0.0074695 0.00739996 0.00710472  
 0.00686645 0.00672584 0.00656871 0.0064729 0.00628121 0.00619268  
 0.00613079 0.00595993 0.00591603 0.00586467 0.00583638 0.00581102  
 0.0057254 0.00562058 0.00560362 0.00553332 0.00539706 0.00521059  
 0.00510985 0.00503851 0.00471029 0.0045352 0.00442628 0.00429436  
 0.00425464 0.00419091 0.00412103 0.00395315 0.00387172 0.00376928  
 0.00368745 0.00357553 0.00355151 0.00342599 0.00332717 0.00319809  
 0.00312569 0.00310408 0.00305175 0.00282526 0.00277869 0.00269064  
 0.00266907 0.00254603 0.00252256 0.00240416 0.00231023 0.00227994  
 0.00225883 0.00222558 0.00215774 0.00206164 0.00202298 0.00193276  
 0.00188686 0.00184892 0.00175645 0.00173155 0.00164004 0.00162618  
 0.00156647 0.00154083 0.00145946 0.00139649 0.00136515 0.00132101  
 0.00129135 0.00126299 0.00117782 0.00113794 0.0010851 0.00106854  
 0.0010531 0.00102521 0.00100454 0.00098676 0.00095396 0.00090743  
 0.00089738 0.00088396 0.00081966 0.00077167 0.00073413 0.00066698  
 0.00064432 0.00063143 0.00059212 0.00058496 0.00057145 0.00053722]
```

```
In [ ]: print(len(svd.explained_variance_ratio_))
```

120

```
In [ ]: np.sum(svd.explained_variance_ratio_)
```

Out[102]: 0.9930403

Now understand that if I transform my dataframe with the help of fitted svd object using svd.transform(dataframe), it does not mean that my original features are getting reduced.

It means that my original features are getting replaced by newer features or newer components which are lesser than my original number of features. That is what they mean by dimensionality reduction. Not reducing the already existing features , but replacing them by different features that are lesser in number.

So if I want to know what all features are important with the most variance and what all features should be removed I cannot find that by transforming .

Meaning no interpretability.

So I can find out the most important features that contribute to my principal components with the help of the 170 values present in each component.The more the value in component for a feature , that important that feature is to a component.

```
In [ ]: #main matrix of the components (descending sorted as per the variance) and their contributing features.  
svd.components_.shape
```

```
Out[103]: (120, 170)
```

```
In [ ]: #number of features  
svd.components_.shape[1]
```

```
Out[104]: 170
```

```
In [ ]: #number of components  
svd.components_.shape[0]
```

```
Out[105]: 120
```

```
In [ ]: set_number_of_features=120
```

Trying Forward Feature selection

Here, in forward feature selection, out of 170 features even if I choose to select 10 features only , still it will take a lot of time.

Therefore I go with recursive feature elimination.

Using Recursive Feature Elimination, I can get the features that I need.

```
In [ ]: #!pip install joblib  
#!pip install mlxtend
```

```
In [ ]: import warnings  
from pandas.core.common import SettingWithCopyWarning  
warnings.simplefilter(action="ignore", category=SettingWithCopyWarning)  
import numpy as np  
import pandas as pd  
from joblib import dump, load  
from sklearn import metrics  
from sklearn.model_selection import train_test_split  
from sklearn.feature_selection import RFECV  
from sklearn.ensemble import RandomForestClassifier  
  
def recursive_feature_elimination(dataframe, y ,number_of_features):  
    clf = RandomForestClassifier(n_estimators=100, n_jobs=-1)  
    selector = RFECV(clf, step=5, cv=5,min_features_to_select=number_of_features,scoring="f1",n_jobs=-1)  
    selector = selector.fit(dataframe, y)  
    print("Rankings : ",selector.ranking_)  
    print("Selected indices :\n",pd.Series(selector.ranking_).value_counts())  
    list_of_index=[]  
    for i in list(range(len(selector.ranking_))):  
        if selector.ranking_[i]==1:  
            list_of_index.append(i)  
    return dataframe[dataframe.columns[list_of_index]]
```

```
In [ ]: new standard train 0 all features df.head()
```

Out[110]:

	sensor1_measure	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_hist
0	-0.209185	-0.432133	-0.105982	-0.028788	-0.057798	-0.115322	
1	0.226385	2.314107	0.287838	-0.028788	-0.057798	-0.115322	
2	-0.411601	-0.432133	-0.097285	-0.028788	-0.057798	-0.114347	
3	0.027280	-0.432131	0.314548	-0.028788	-0.057798	-0.115322	
4	-0.132515	-0.432132	0.042477	-0.028788	-0.057798	-0.115322	

5 rows × 150 columns

Try spearman corelation coefficient to find out co relation. (Finding and keeping correlated features is good for cause and effect analysis but it might not be of much use when we find that our features are dependent on other features, we expect our features to be

independent of each other.).

```
In [ ]: def find_the_total_features(mid_dataframe,my_range):
    set_for_features=set()
    for row in list(mid_dataframe.index.values):
        for column in list(mid_dataframe.columns.values):
            if (mid_dataframe[column][row]>=my_range or mid_dataframe[column][row]<=-my_range) and column!=row:
                set_for_features.update([column])
                set_for_features.update([row])
    return set_for_features,len(list(mid_dataframe.index.values))-len(list(set_for_features))

def select_spearman(input_dataframe,number_of_features):
    #spearman correlation.
    mid_dataframe=input_dataframe.corr(method ='spearman')
    feature_dict={}
    feature_dict.update({.1:find_the_total_features(mid_dataframe,.1)[1]}) 
    feature_dict.update({.90:find_the_total_features(mid_dataframe,.90)[1]}) 
    feature_dict.update({.95:find_the_total_features(mid_dataframe,.95)[1]}) 
    feature_dict.update({.97:find_the_total_features(mid_dataframe,.97)[1]}) 
    feature_dict.update({.98:find_the_total_features(mid_dataframe,.98)[1]}) 
    feature_dict.update({.99:find_the_total_features(mid_dataframe,.99)[1]}) 
    feature_dict.update({.999:find_the_total_features(mid_dataframe,.999)[1]}) 

    print("If I take the data having correlation between only -.1 and .1 range then I get ",
          feature_dict[.1]," features.")
    print("If I take the data having correlation between only -.90 and .90 range then I get ",
          feature_dict[.90]," features.")
    print("If I take the data having correlation between only -.95 and .95 range then I get ",
          feature_dict[.95]," features.")
    print("If I take the data having correlation between only -.97 and .97 range then I get ",
          feature_dict[.97]," features.")
    print("If I take the data having correlation between only -.98 and .98 range then I get ",
          feature_dict[.98]," features.")
    print("If I take the data having correlation between only -.99 and .99 range then I get ",
          feature_dict[.99]," features.")
    print("If I take the data having correlation between only -.999 and .999 range then I get ",
          feature_dict[.999]," features.")
    return mid_dataframe

def perform_spearman(input_dataframe,mid_dataframe,selected_range):
    #most of the data is less is correlated and if I accept only -10% to 10% then there are very less features
    for column in find_the_total_features(mid_dataframe,selected_range)[0]:
```

```

try:
    input_dataframe.drop(column, axis=1, inplace=True)
except KeyError:
    pass

return input_dataframe

```

In []: %time

```

spearman=select_spearman(new_standard_train_0_all_features_df,int(set_number_of_features+np.floor((len(train
If I take the data having correlation between only -.1 and .1 range then I get 0 features.
If I take the data having correlation between only -.90 and .90 range then I get 63 features.
If I take the data having correlation between only -.95 and .95 range then I get 88 features.
If I take the data having correlation between only -.97 and .97 range then I get 102 features.
If I take the data having correlation between only -.98 and .98 range then I get 117 features.
If I take the data having correlation between only -.99 and .99 range then I get 132 features.
If I take the data having correlation between only -.999 and .999 range then I get 145 features.
CPU times: user 15.3 s, sys: 96.5 ms, total: 15.4 s
Wall time: 15.3 s

```

I need enough data points here, so I will select the range accordingly.

In []: new_standard_train_0_all_features_df=perform_spearman(new_standard_train_0_all_features_df,spearman,.99)

In []: new_standard_train_0_all_features_df.head()

Out[114]:

	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_histogram_bin4	sensc
0	-0.432133	-0.105982	-0.028788	-0.057798	-0.115322	-0.179659	
1	2.314107	0.287838	-0.028788	-0.057798	-0.115322	-0.174906	
2	-0.432133	-0.097285	-0.028788	-0.057798	-0.114347	-0.176379	
3	-0.432131	0.314548	-0.028788	-0.057798	-0.115322	-0.180076	
4	-0.432132	0.042477	-0.028788	-0.057798	-0.115322	-0.180233	

5 rows × 132 columns

```
In [ ]: new_standard_train_0_all_features_df.shape
```

```
Out[115]: (48000, 132)
```

```
In [ ]: new_standard_test_0_all_features_df=new_standard_train_0_all_features_df[new_standard_train_0_all_features_df.col  
new_standard_test_0_all_features_df.head()
```

```
Out[116]:
```

	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_histogram_bin4	sens
0	-0.432133	-0.105982	-0.028788	-0.057798	-0.115322	-0.180082	
1	2.314107	0.146833	-0.028788	-0.057798	-0.115322	-0.180023	
2	-0.432133	-0.104118	-0.028788	-0.057798	-0.115322	-0.180334	
3	2.314107	-0.103497	-0.028788	-0.057798	-0.115322	-0.180149	
4	2.314107	0.053037	-0.028788	-0.057798	-0.115322	-0.180242	

5 rows × 132 columns

```
In [ ]: #new_standard_train_0_all_features_df.to_pickle("new_standard_train_0_all_features_df.pickle")  
#y_train_0_all_features.to_pickle("y_train_0_all_features.pickle")  
#new_standard_test_0_all_features_df.to_pickle("new_standard_test_0_all_features_df.pickle")  
#y_test_0_all_features.to_pickle("y_test_0_all_features.pickle")
```

```
In [ ]: #new_standard_train_0_all_features_df=pd.read_pickle("/content/gdrive/My Drive/new_standard_train_0_all_fea  
#y_train_0_all_features=pd.read_pickle("/content/gdrive/My Drive/y_train_0_all_features.pickle")  
  
#new_standard_test_0_all_features_df=pd.read_pickle("/content/gdrive/My Drive/new_standard_test_0_all_featu  
#y_test_0_all_features=pd.read_pickle("/content/gdrive/My Drive/y_test_0_all_features.pickle")
```

```
In [ ]: np.save('new_standard_train_0_all_features_df_columns.npy',new_standard_train_0_all_features_df.value
```

```
In [ ]: new_standard_train_0_all_features_df["target"] = y_train_0_all_features
new_standard_train_0_all_features_df.head()
```

Out[17]:

	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_histogram_bin4	sens
0	-0.432133	-0.105982	-0.028788	-0.057798	-0.115322	-0.179659	
1	2.314107	0.287838	-0.028788	-0.057798	-0.115322	-0.174906	
2	-0.432133	-0.097285	-0.028788	-0.057798	-0.114347	-0.176379	
3	-0.432131	0.314548	-0.028788	-0.057798	-0.115322	-0.180076	
4	-0.432132	0.042477	-0.028788	-0.057798	-0.115322	-0.180233	

5 rows × 133 columns

Compare the target 0 and target 1 mean values feature wise after 0 value imputing

```
In [ ]: column_list = []
mean_list = []
one_zero = []
for column in list(new_standard_train_0_all_features_df.columns.values):

    if column != "target":

        column_list.append(column)
        one_zero.append(0)
        mean_list.append(new_standard_train_0_all_features_df[column][new_standard_train_0_all_features_df.t

        column_list.append(column)
        one_zero.append(1)
        mean_list.append(new_standard_train_0_all_features_df[column][new_standard_train_0_all_features_df.t
```

Find the features where the difference between the means of the two classes is more than .15
descending sorted by class 1

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:mean_list[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

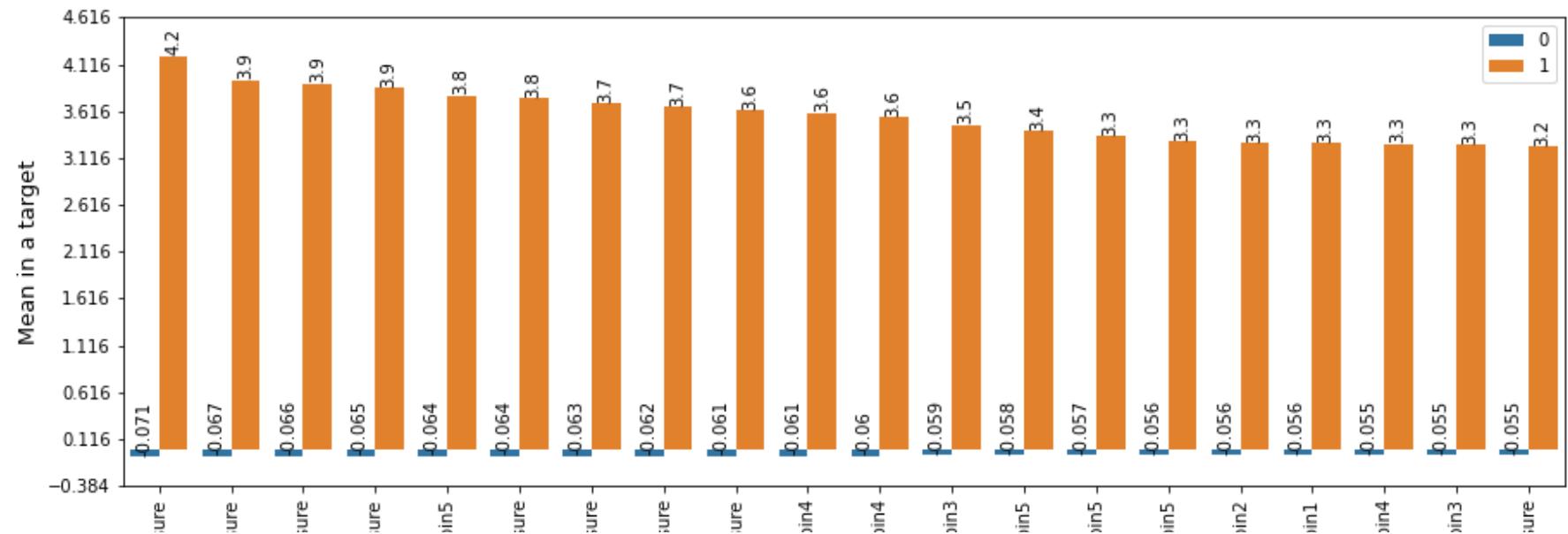
```
In [ ]: counter=0
new_column_list=[]
new_mean_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_mean_list.append(mean_list[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_mean_list.append(mean_list[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 132 sensors.

In []:

```
try:  
    plot_diagram(new_column_list,new_mean_list,new_one_zero,"Mean",1)  
except ValueError:  
    pass
```

Mean in a target value descending sorted by class 1



Taking the mean values class wise and feature wise after 0 imputation, there is a considerable difference between the means of the downhole failure and surface failure.

The mean for the downhole failure for most of the given sensors is more than the mean of the surface level failures.

```
In [ ]: counter=0
mean_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.

    if (np.abs(np.abs(mean_list[i])-np.abs(mean_list[i-1]))>=.15):
        mean_higher_reading.append(mean_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

        mean_higher_reading.append(mean_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

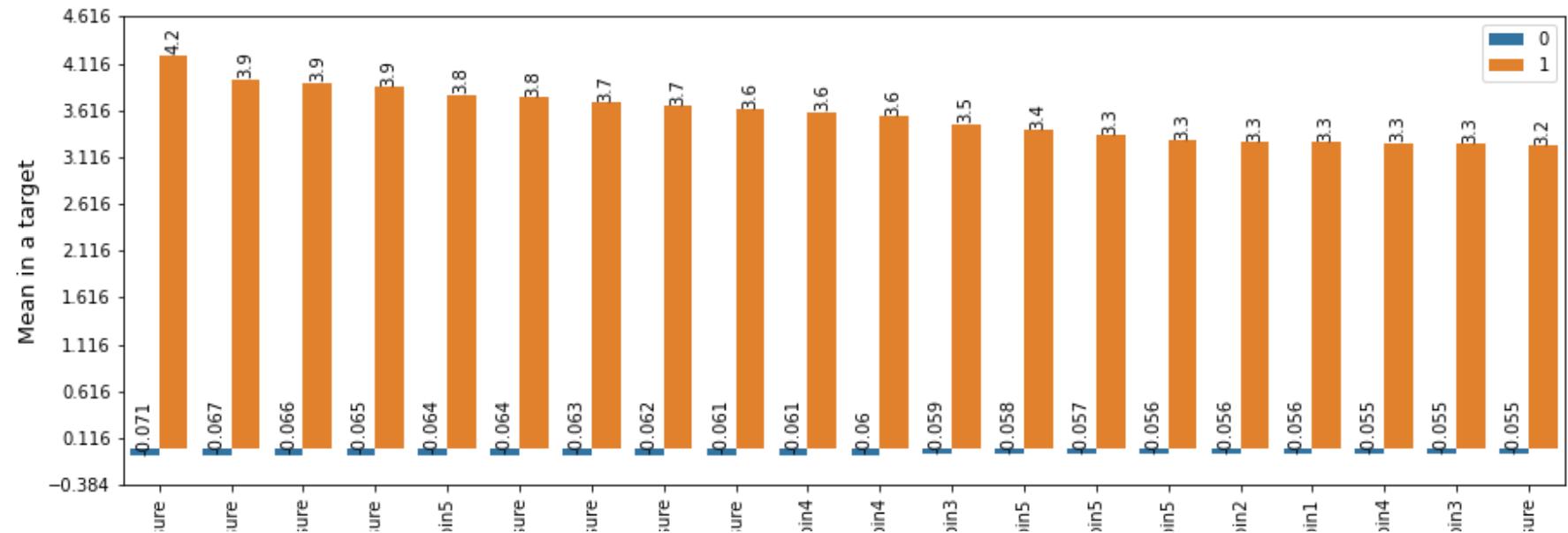
    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between median readings of
```

So we have approximate 129 sensor that show significant difference (> .15) between median readings of surface and downhole failures.

In []: **try:**

```
plot_diagram(column_higher_reading,mean_higher_reading,one_zero_higher_reading,"Mean",1)
except ValueError:
    pass
```

Mean in a target value descending sorted by class 1

**Observations regarding this**

- 1) Above are the sensors with most difference between the means of the surface failures as well as the downhole failures.
- 2) Here the mean for the downhole failures is a lot more than that of the surface failures.
- 3) Note that all the values which were missing , we filled them with 0 values.

Compare the target 0 and target 1 median values feature wise after 0 value imputing

```
In [ ]: column_list=[]
median_list=[]
one_zero=[]

for column in list(new_standard_train_0_all_features_df.columns.values):
    if column!="target":

        column_list.append(column)
        one_zero.append(0)
        median_list.append(new_standard_train_0_all_features_df[column][new_standard_train_0_all_features_df
```

**Find the features where the difference between the medians of the two classes is more than .15
descending sorted by class 1**

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:median_list[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

```
In [ ]: counter=0
new_column_list=[]
new_median_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_median_list.append(median_list[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_median_list.append(median_list[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 132 sensors.

```
In [ ]: try:
    plot_diagram(new_column_list,new_median_list,new_one_zero,"Median",1)
except ValueError:
    pass
```

Observations:

- 1) In the above graph there are sensors for which the mean of downhole failures is more than that of the surface failures. For example sensor measure 59,16,49.
- 2) For a few sensors the value is the same. Eg: sensor measure 68,81,100
- 3) Whereas for a few sensors, the mean of the surface failure is more than that of the surface. Eg: sensor histogram 105 bin 8,sensor 29, 28 measure.

```
In [ ]: counter=0
median_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.

    if (np.abs(np.abs(median_list[i])-np.abs(median_list[i-1]))>=.15):
        median_higher_reading.append(median_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

        median_higher_reading.append(median_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

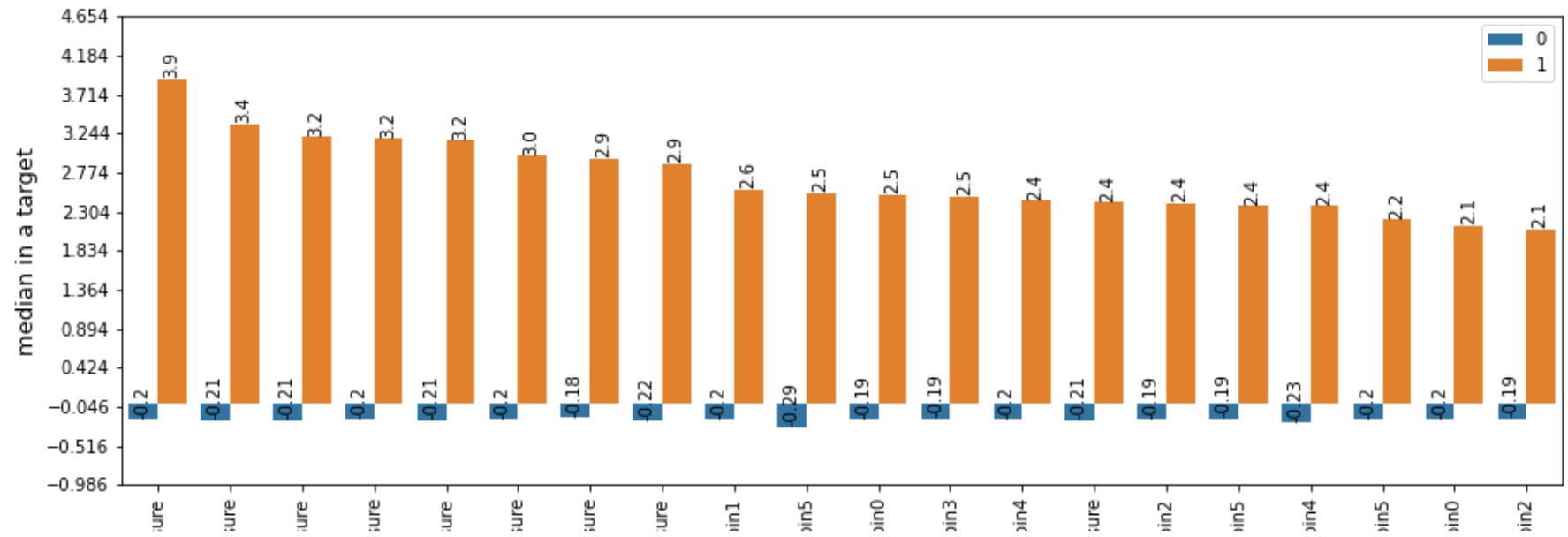
    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between median readings of surface and downhole failures.")
```

So we have approximate 63 sensor that show significant difference (> .15) between median readings of surface and downhole failures.

In []:

```
try:  
    plot_diagram(column_higher_reading,median_higher_reading,one_zero_higher_reading,"median",1)  
except ValueError:  
    pass
```

median in a target value descending sorted by class 1



```
In [51]: def dummy_classifier_based_on_median(data_point,c_list,m_list):
    feature_win=[0,0]
    for i in range(0,len(c_list),2):
        # In this dummy classifier, I will take the value that is closer to a particular median,
        # If my feature value is closer to the median value of the class 0 i.e surface failure,
        # I increase my point for surface failure.
        point=data_point[c_list[i]]
        if (np.abs(np.abs(point)-np.abs(m_list[i])))>(np.abs(np.abs(point)-np.abs(m_list[i+1]))):
            feature_win[1]=feature_win[1]+1
        # If my feature value is closer to the median value of the class 1 i.e downhole failure,
        # I increase my point for downhole failure.
        elif (np.abs(np.abs(np.abs(point)-np.abs(m_list[i+1])))>(np.abs(point)-np.abs(m_list[i]))):
            feature_win[0]=feature_win[0]+1
        # If my feature value is closer to both the classes,
        # I take the majority into consideration and increase a point for class 0.
        elif (np.abs(np.abs(np.abs(point)-np.abs(m_list[i+1])))==(np.abs(point)-np.abs(m_list[i]))):
            feature_win[0]=feature_win[0]+1
        if feature_win[0]>=feature_win[1]:
            print("Predicted surface failure.")
        if feature_win[1]>feature_win[0]:
            print("Predicted downhole failure.")
    return feature_win
```

Basic Classifier

```
In [ ]: new_standard_train_0_all_features_df.drop("target",axis=1,inplace=True)
```

```
In [ ]: print(dummy_classifier_based_on_median(new_standard_train_0_all_features_df.loc[0],
                                              column_list,
                                              median_list))
```

Predicted surface failure.
[108, 24]

```
In [ ]: print(dummy_classifier_based_on_median(new_standard_test_0_all_features_df.loc[529],  
                                              column_list,  
                                              median_list))
```

Predicted surface failure.
[116, 16]

```
In [ ]: new_standard_train_0_all_features_df["target"] = y_train_0_all_features
```

EDA for the dataset

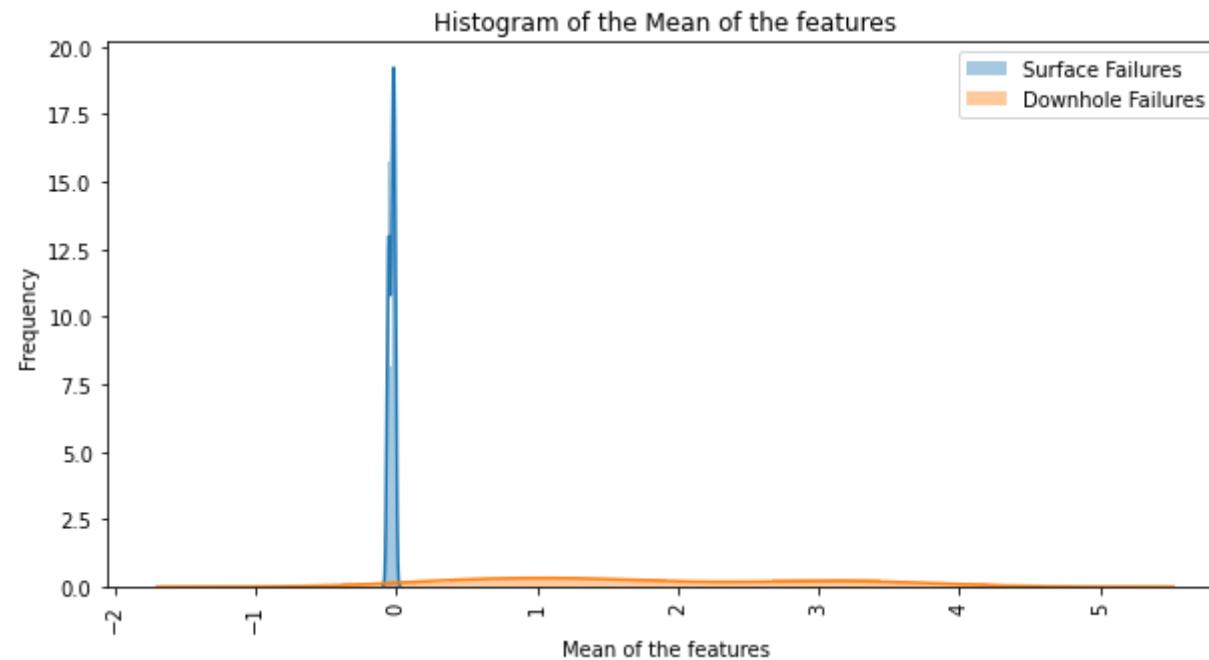
Using histograms, plots find out the nature of means for features for both the classes and thereby, the outliers.

```
In [ ]: mean_0=[]  
mean_1=[]  
for column in list(new_standard_train_0_all_features_df.columns.values):  
    if column!="target":  
        mean_0.append(new_standard_train_0_all_features_df[new_standard_train_0_all_features_df.target==0][c]  
        mean_1.append(new_standard_train_0_all_features_df[new_standard_train_0_all_features_df.target==1][c])
```

```
In [ ]: median_0=[]  
median_1=[]  
for column in list(new_standard_train_0_all_features_df.columns.values):  
    if column!="target":  
        median_0.append(new_standard_train_0_all_features_df[new_standard_train_0_all_features_df.target==0][c])  
        median_1.append(new_standard_train_0_all_features_df[new_standard_train_0_all_features_df.target==1][c])
```

10) Try and plot it's histogram for both the target values.

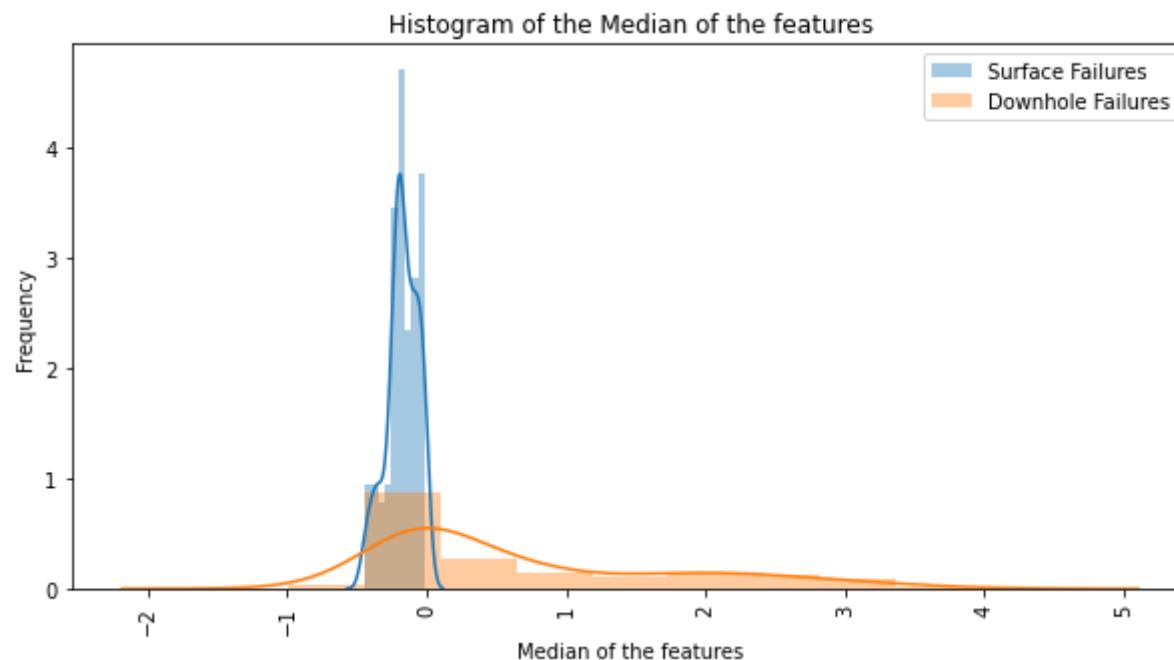
```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Histogram of the Mean of the features')
plt.ylabel('Frequency')
plt.xlabel('Mean of the features')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations :

- 1) The mean values for surface failures after imputing zero is around the range of 0 only.
- 2) But we can see that the mean values for the downhole features have values more in magnitude.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('Histogram of the Median of the features')
plt.ylabel('Frequency')
plt.xlabel('Median of the features')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```

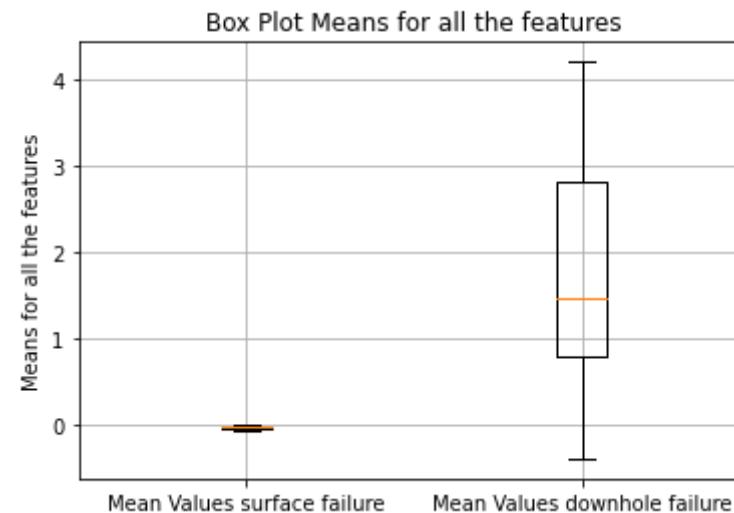


Observations :

After 0 value imputation , Similar to the mean values , more variance is seen the medians of the downhole failure rather than that of the surface failures.

11) Try box plot / violin plot to visualize the outliers.

```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Box Plot Means for all the features')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the features')  
plt.grid()  
plt.show()
```

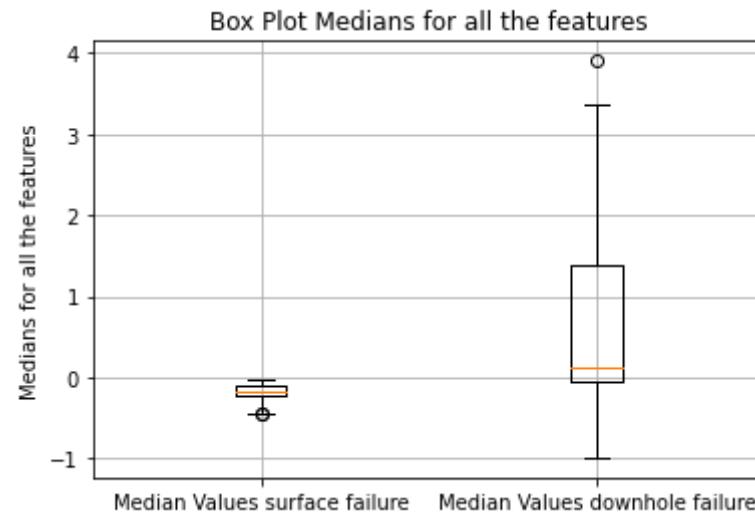


Observations :

The mean values for the downhole failures show more variance than that of the surface level failures.

More detail can be revealed when we plot a bar plot for both the classes feature wise.

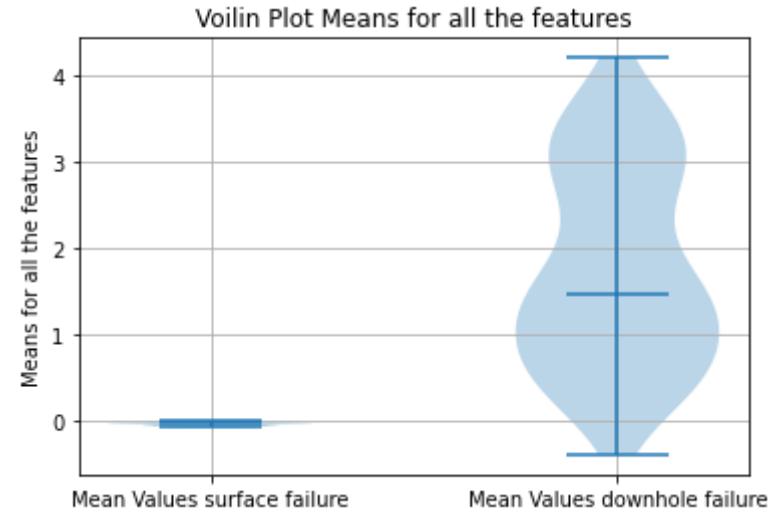
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('Box Plot Medians for all the features')  
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))  
plt.ylabel('Medians for all the features')  
plt.grid()  
plt.show()
```



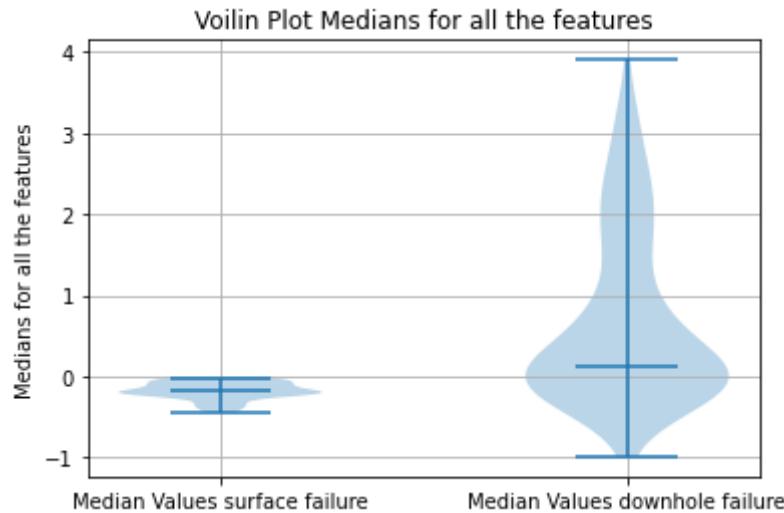
Observations :

Apart from the 0 to -5 range , we can easily make out the difference between the medians of both the surface as well as downhole failures.

```
In [ ]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Violin Plot Means for all the features')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the features')
plt.grid()
plt.show()
```



```
In [ ]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('Violin Plot Medians for all the features')
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))
plt.ylabel('Medians for all the features')
plt.grid()
plt.show()
```



Observation :

From 0 to .-.5 higher chance of finding the value point as surface failure.

But exceed this particular and we have a higher chance of finding the value point as a downhole failure.

[Why is there such a huge difference between mean/median of the surface failures ?](#)

[Remember I have 59000 surface failures where if majority is np.NaN then on imputing all those points with 0, we get mean and median as 0.,](#)

[whereas for the 1000 downhole features,if we impute 0 to these 1000 data points, if we have less values to impute and we have a lesser chance to get mean/median as a 0.](#)

[Meaning imputing 0, we will always have a more chance of a near zero mean and median for a surface failure given more np.NaNs here.](#)

[But here from the previous graphs without imputing we can see that still the mean and median for the surface value is near 0 only.](#)

PCA

In [13]:

```
def run_pca(dataframe,target_value):

    pca = PCA(n_components = 2)

    pca_data = pca.fit_transform(dataframe)

    #The shape of anything that you want to hstack must of a column vector that is multiple rows * 1 column
    for_pca = np.hstack((pca_data[:,0].reshape(-1,1),pca_data[:,1].reshape(-1,1), target_value.to_numpy().r

    print(for_pca)

    for_pca_df = pd.DataFrame(data=for_pca, columns=['Dimension_1_X','Dimension_2_Y','target'])

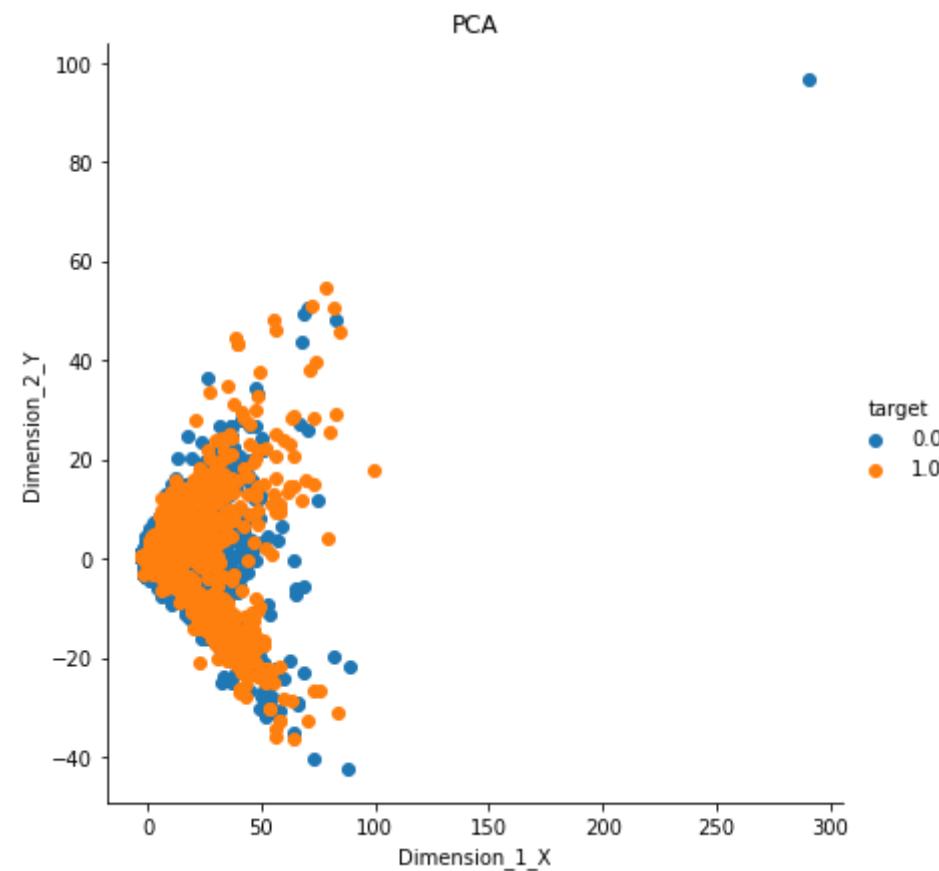
    title="PCA"
    sns.FacetGrid(for_pca_df, hue="target", height=6).map(plt.scatter, 'Dimension_1_X', 'Dimension_2_Y').add
    plt.title(title)
    plt.show()
del pca,pca_data,for_pca,for_pca_df
```

In []: new_standard_train_0_all_features_df.drop("target",axis=1,inplace=True)

In []:

```
%%time  
run_pca(new_standard_train_0_all_features_df,y_train_0_all_features)
```

```
[[ -1.5879772 -0.26651788 0.          ]  
 [ 2.442555   -0.16708013 0.          ]  
 [-2.822596    0.3773239 0.          ]  
 ...  
 [-0.5035219 -0.0782191 0.          ]  
 [-2.837994   0.3715543 0.          ]  
 [-1.8105145 -3.1082456 0.          ]]]
```



```
CPU times: user 1.95 s, sys: 936 ms, total: 2.89 s
Wall time: 6.5 s
```

tsne

```
In [ ]: def run_tsne(dataframe,target_value,a,b):
    t_SNE = TSNE(n_components=2,perplexity=a, n_iter=b)
    t_SNE_data = t_SNE.fit_transform(dataframe)

    #The shape of anything that you want to hstack must of a column vector that is multiple rows * 1 column
    for_tsne = np.hstack((t_SNE_data[:,0].reshape(-1,1),t_SNE_data[:,1].reshape(-1,1), target_value.to_numpy))

    for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_1_X','Dimension_2_Y','target'])

    title="TSNE for perplexity : "+str(a)+" and iterations : "+str(b)
    sns.FacetGrid(for_tsne_df, hue="target", height=6).map(plt.scatter, 'Dimension_1_X', 'Dimension_2_Y').add_titles(title)
    plt.title(title)
    plt.show()
    del t_SNE,t_SNE_data,for_tsne,for_tsne_df
```

Do not put these values in a loop as we do not get all the output graphs

In []: new_standard_train_0_all_features_df.head()

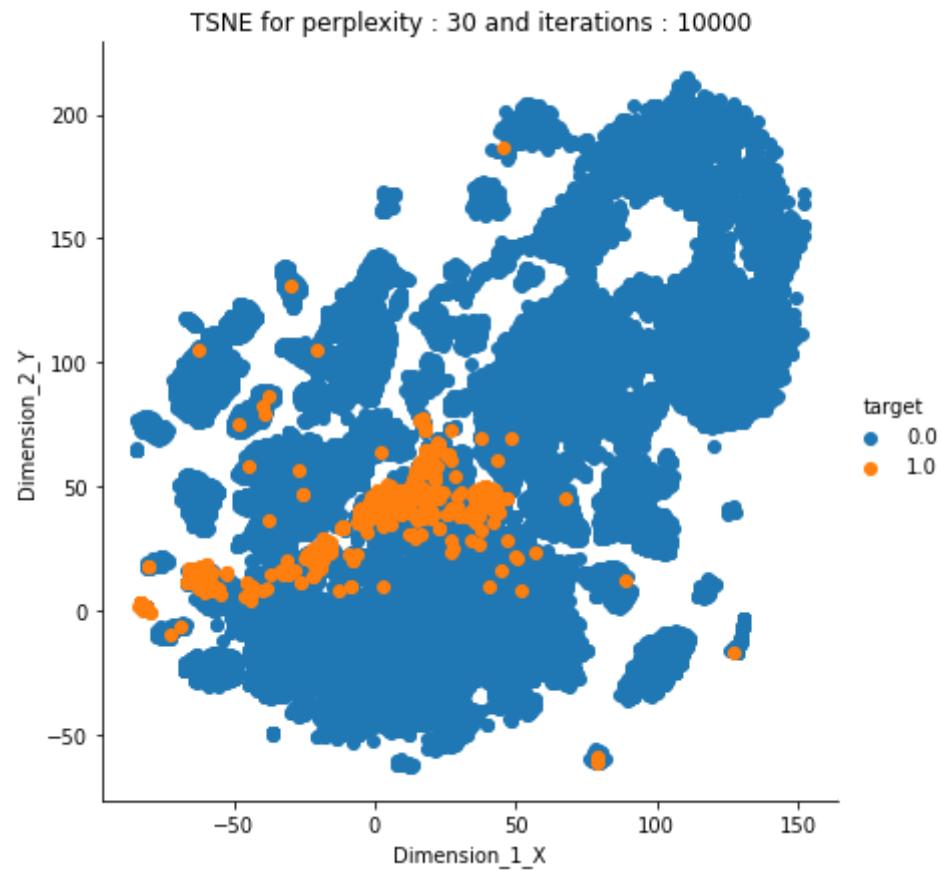
Out[22]:

	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_histogram_bin4	sensc
0	-0.432133	-0.105982	-0.028788	-0.057798	-0.115322	-0.179659	
1	2.314107	0.287838	-0.028788	-0.057798	-0.115322	-0.174906	
2	-0.432133	-0.097285	-0.028788	-0.057798	-0.114347	-0.176379	
3	-0.432131	0.314548	-0.028788	-0.057798	-0.115322	-0.180076	
4	-0.432132	0.042477	-0.028788	-0.057798	-0.115322	-0.180233	

5 rows × 132 columns

In []:

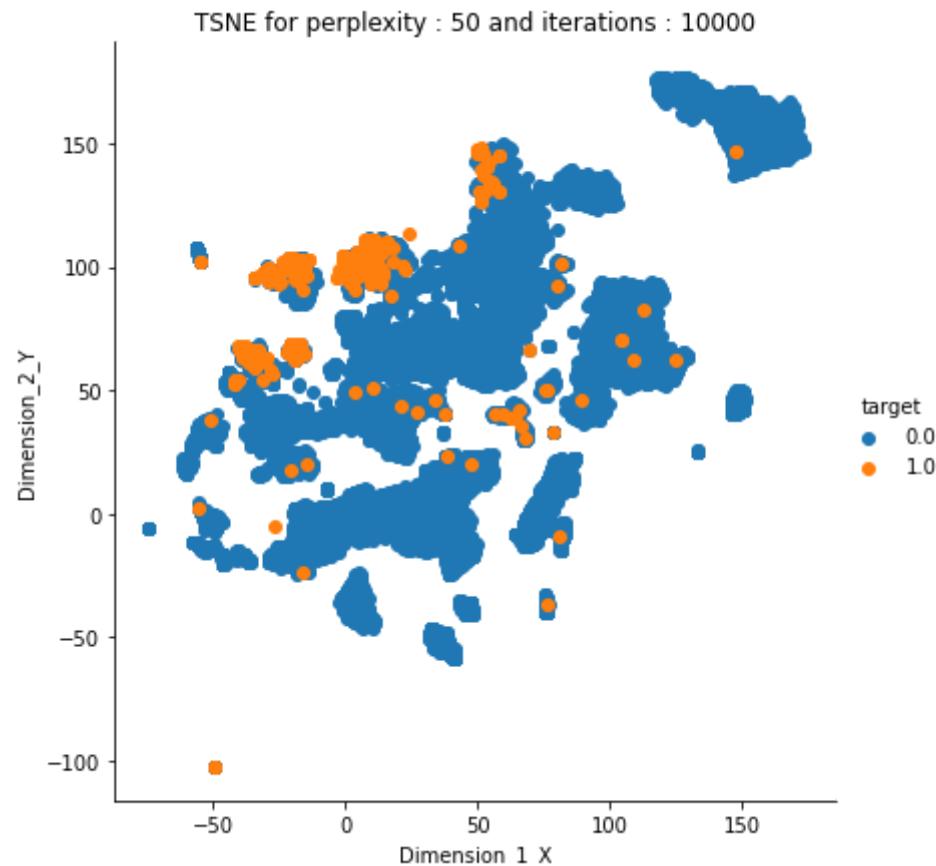
```
%%time  
run_tsne(new_standard_train_0_all_features_df,y_train_0_all_features,30,10000)
```



```
CPU times: user 12.5 s, sys: 13.6 s, total: 26.1 s
Wall time: 25.9 s
```

In []:

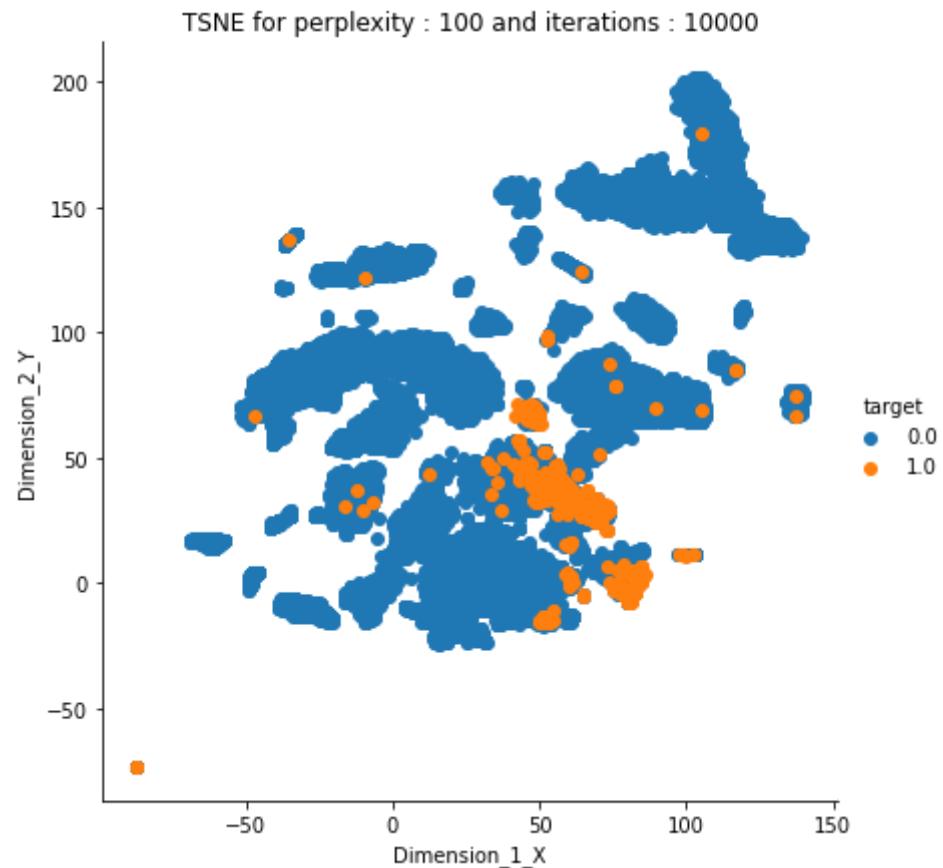
```
%%time  
run_tsne(new_standard_train_0_all_features_df,y_train_0_all_features,50,10000)
```



CPU times: user 13.1 s, sys: 13.6 s, total: 26.7 s
Wall time: 26.6 s

In []:

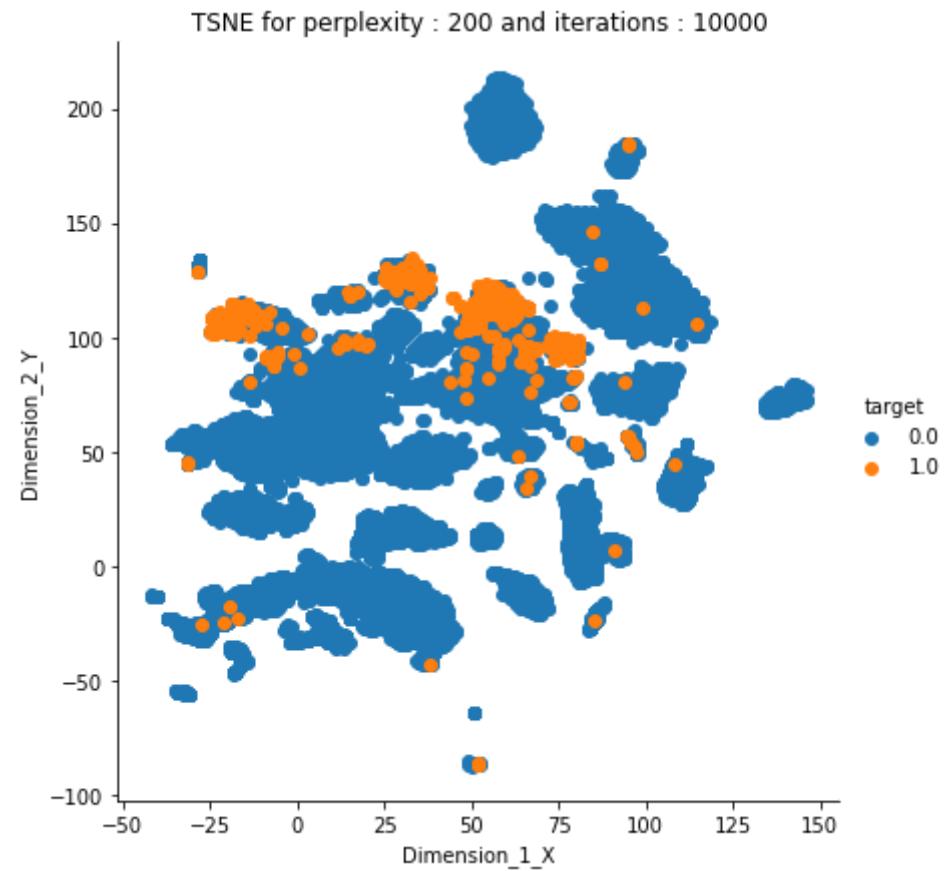
```
%%time  
run_tsne(new_standard_train_0_all_features_df,y_train_0_all_features,100,10000)
```



CPU times: user 13.7 s, sys: 13.4 s, total: 27.1 s
Wall time: 27 s

In []:

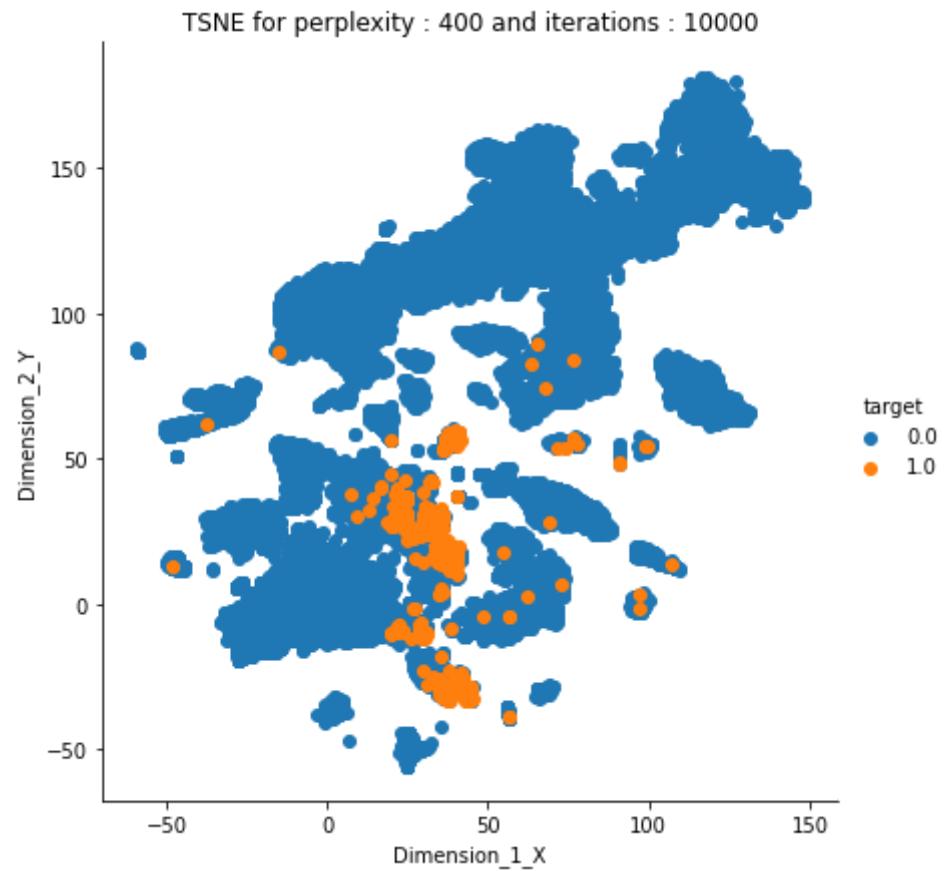
```
%%time  
run_tsne(new_standard_train_0_all_features_df,y_train_0_all_features,200,10000)
```



CPU times: user 14.3 s, sys: 13.6 s, total: 27.9 s
Wall time: 27.7 s

In []:

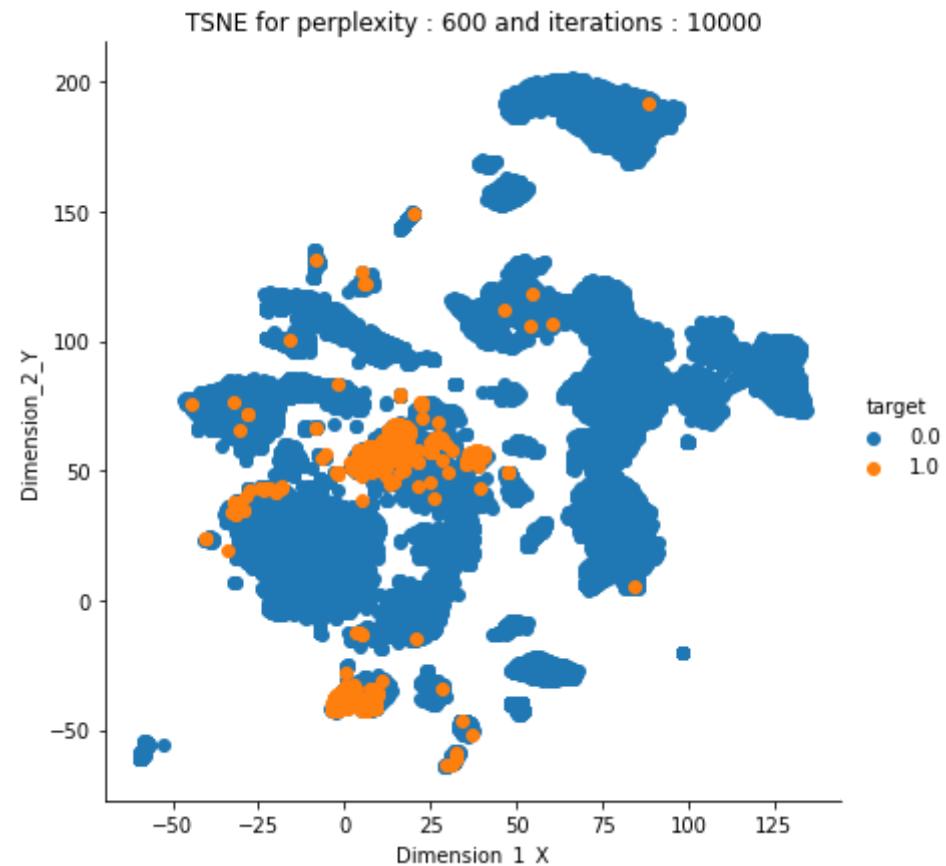
```
%%time  
run_tsne(new_standard_train_0_all_features_df,y_train_0_all_features,400,10000)
```



CPU times: user 14.2 s, sys: 13.2 s, total: 27.4 s
Wall time: 27.2 s

In []:

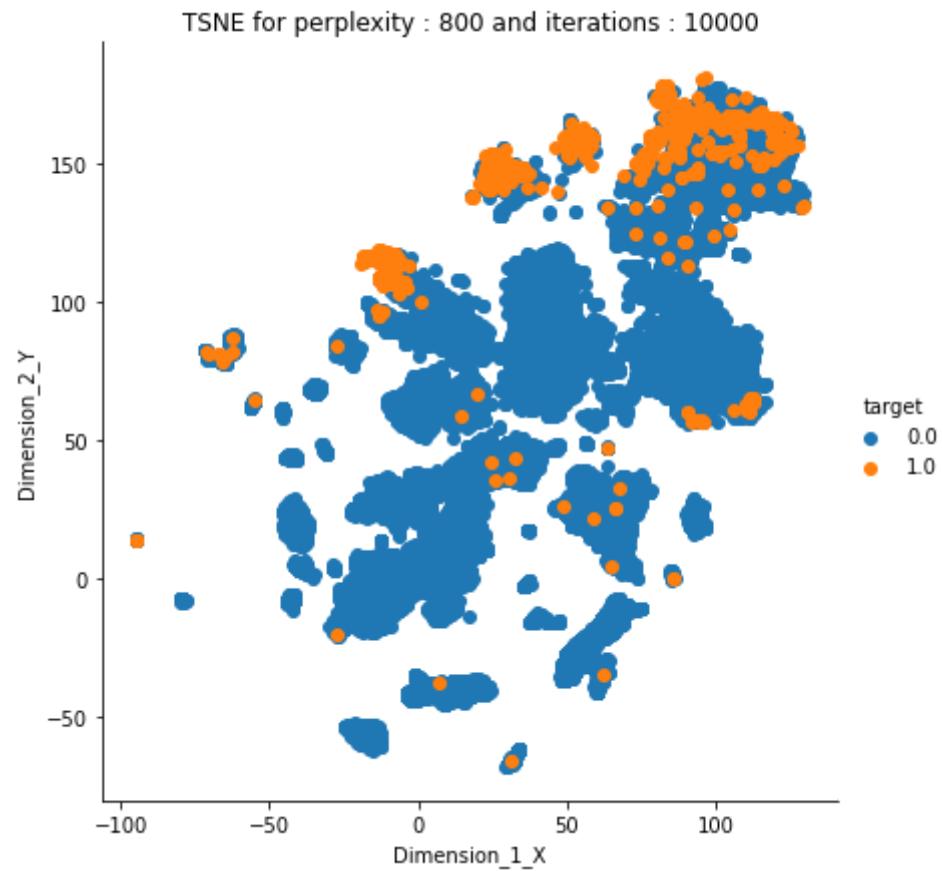
```
%%time  
run_tsne(new_standard_train_0_all_features_df,y_train_0_all_features,600,10000)
```



CPU times: user 14 s, sys: 13.3 s, total: 27.3 s
Wall time: 27.1 s

In []:

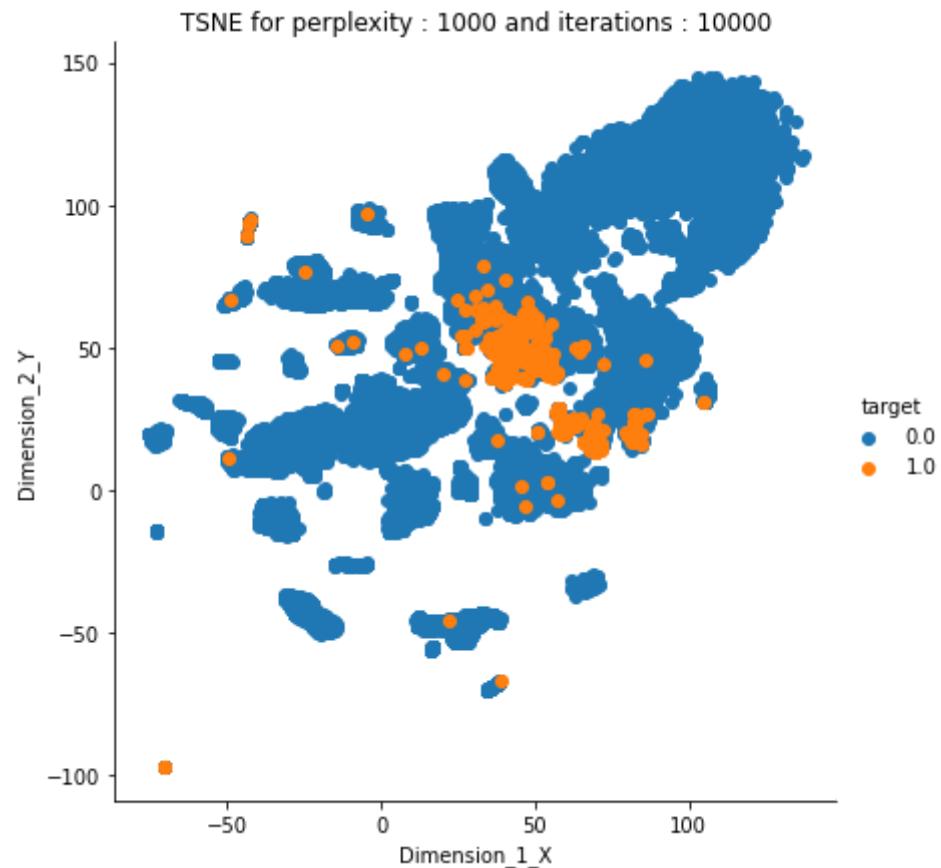
```
%%time  
run_tsne(new_standard_train_0_all_features_df,y_train_0_all_features,800,10000)
```



CPU times: user 14.2 s, sys: 13.5 s, total: 27.7 s
Wall time: 27.5 s

In []:

```
%%time  
run_tsne(new_standard_train_0_all_features_df,y_train_0_all_features,1000,10000)
```



CPU times: user 14.2 s, sys: 13.6 s, total: 27.8 s
Wall time: 27.6 s

Observation :

- 1) Observe that different perplexities tSNE is able to differentiate all the data points into different groups, but it is not able to differentiate properly the surface failures from the downhole failures. If the surface and downhole failures are coming under the same category then they are shown in the same cluster.
- 2) Not all the the surface and downhole features are coming under the same category.

Perform a similar EDA on the outliers.

Here we take the outliers from the features that have been selected.

For each sensor we find the outliers that also classwise.

The issue is that there maybe high outliers here , but there maybe lower outliers here as well.

<https://www.geeksforgeeks.org/interquartile-range-and-quartile-deviation-using-numpy-and-scipy/> (<https://www.geeksforgeeks.org/interquartile-range-and-quartile-deviation-using-numpy-and-scipy/>)

Higher outlier = more than $1.5 \cdot \text{IQR}$ above the third quartile.

Lower outlier = less than $1.5 \cdot \text{IQR}$ below the first quartile.

The IQR is the difference between Q3 and Q1.

```
In [ ]: new_standard_train_0_all_features_df["target"] = y_train_0_all_features
```

```
In [ ]: counter=0
main_dictionary=dict()
outliers=[]
list_of_outliers_features_classes=[]
for i in range(len(column_list)):
    d=dict()
    new_list=new_standard_train_0_all_features_df[new_standard_train_0_all_features_df.target==one_zero[i]]\[
        column_list[i]
    new_index=list(new_list.reset_index()["index"])
    QR1=np.percentile(new_list, 25)
    QR3=np.percentile(new_list, 75)
    IQR=QR3-QR1
    for index,value in zip(new_index,new_list):
        if (value>(QR3+(IQR*1.5))) or (value<(QR1-(IQR*1.5))):
            counter=counter+1
            d.update({index:value})
            if index not in main_dictionary:
                main_dictionary.update({index:list(new_standard_train_0_all_features_df.loc[index])})
    list_of_outliers_features_classes.append(d)
del d
outliers.append(counter)
counter=0
```

```
In [ ]: print("Total number of data points that are outliers in atleast one or more than one feature : ",len(np.unique
```

```
Total number of data points that are outliers in atleast one or more than one feature : 44483
```

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:outliers[i] for i in index}

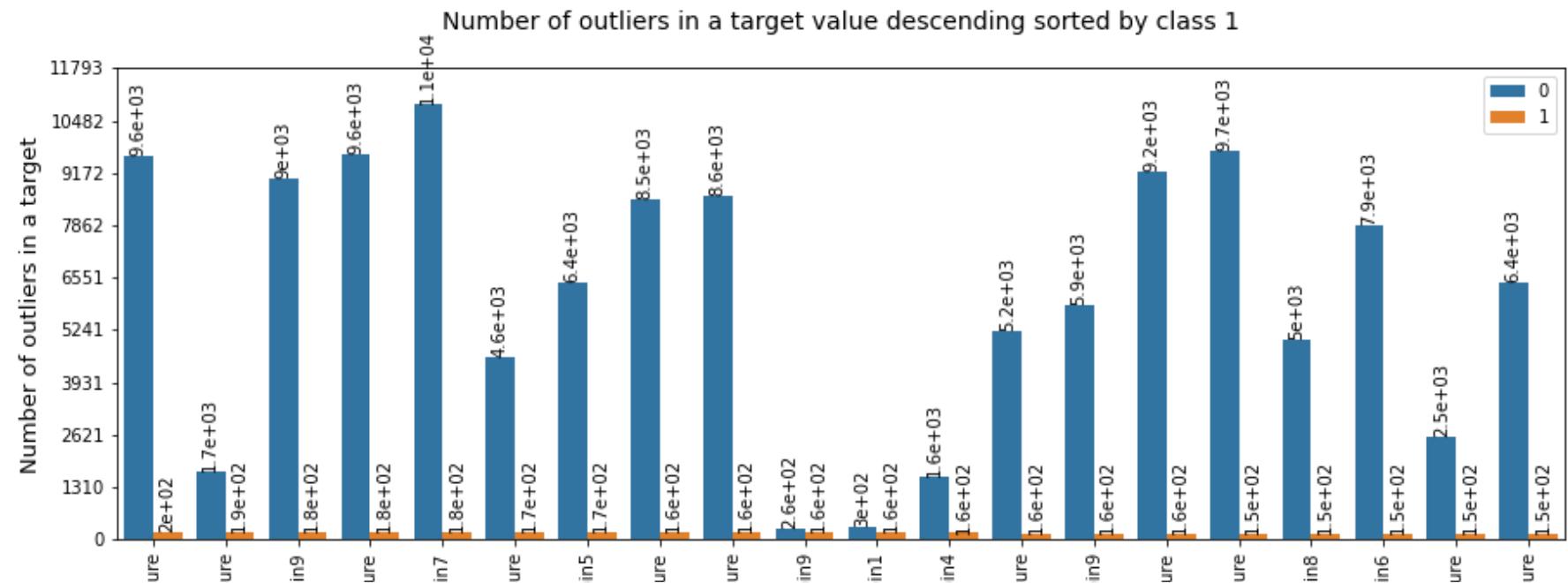
d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])

counter=0
new_column_list=[]
new_outliers_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_outliers_list.append(outliers[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_outliers_list.append(outliers[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 132 sensors.

In []: **try:**

```
plot_diagram(new_column_list,new_outliers_list,new_one_zero,"Number of outliers",1)
except ValueError:
    pass
```

**Observation :**

Maximum number of outliers can be seen in the surface level failures , compared to it, there are way less numbers of outliers in the downhole failures.

Meaning, here originally and during imputation , there is chance that we make the median or mean as near to 0 for when we have imputed enough data points.

So these values , then become the outliers. Hence more outliers for the surface related failures.

For every column feature, get the mean for the outliers for both class 0 and class 1.

```
In [ ]: mean_0=[]
mean_1=[]
for i in range(0,len(list_of_outliers_features_classes),2):
    class_0=[ele for ele in list_of_outliers_features_classes[i].values()]
    class_1=[ele for ele in list_of_outliers_features_classes[i+1].values()]
    if (len(class_0)==0) :
        mean_0.append(0.01)
    else :
        mean_0.append(np.mean(class_0))

    if (len(class_1)==0):
        mean_1.append(0.01)
    else :
        mean_1.append(np.mean(class_1))
```

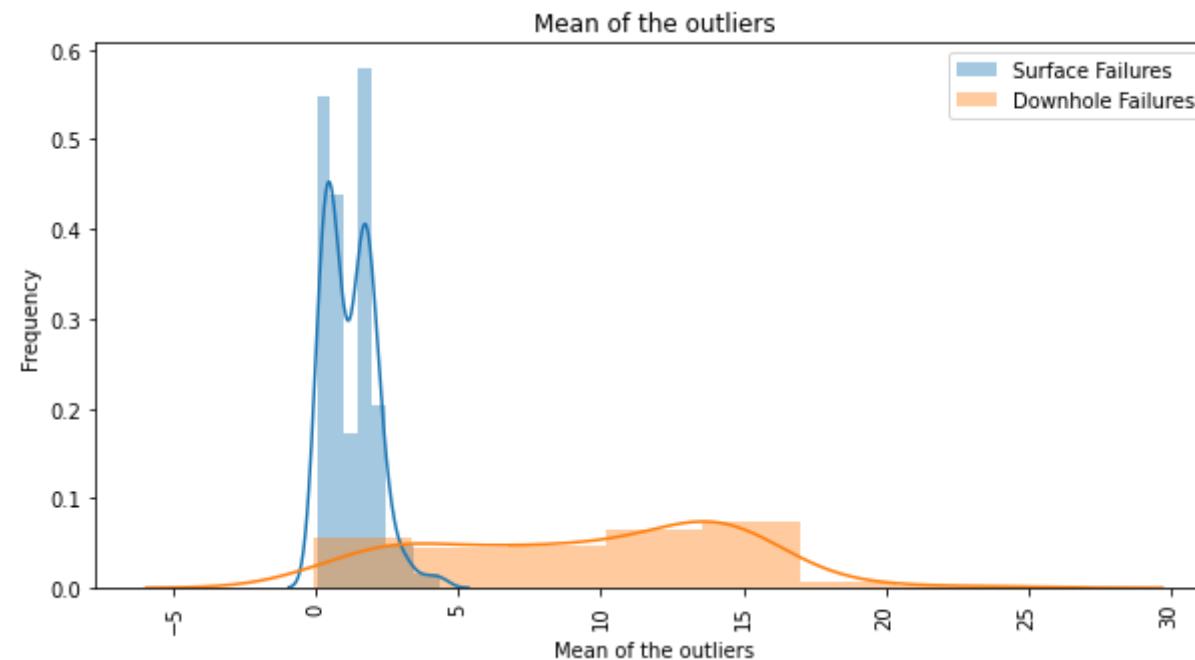
```
In [ ]: median_0=[]
median_1=[]
for i in range(0,len(list_of_outliers_features_classes),2):
    class_0=[ele for ele in list_of_outliers_features_classes[i].values()]
    class_1=[ele for ele in list_of_outliers_features_classes[i+1].values()]
    if (len(class_0)==0) :
        median_0.append(0.01)
    else :
        median_0.append(np.median(class_0))

    if (len(class_1)==0):
        median_1.append(0.01)
    else :
        median_1.append(np.median(class_1))
```

Plotting the mean and median for each feature class wise.

10) Try and plot it's histogram for both the target values.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Mean of the outliers')
plt.ylabel('Frequency')
plt.xlabel('Mean of the outliers')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```

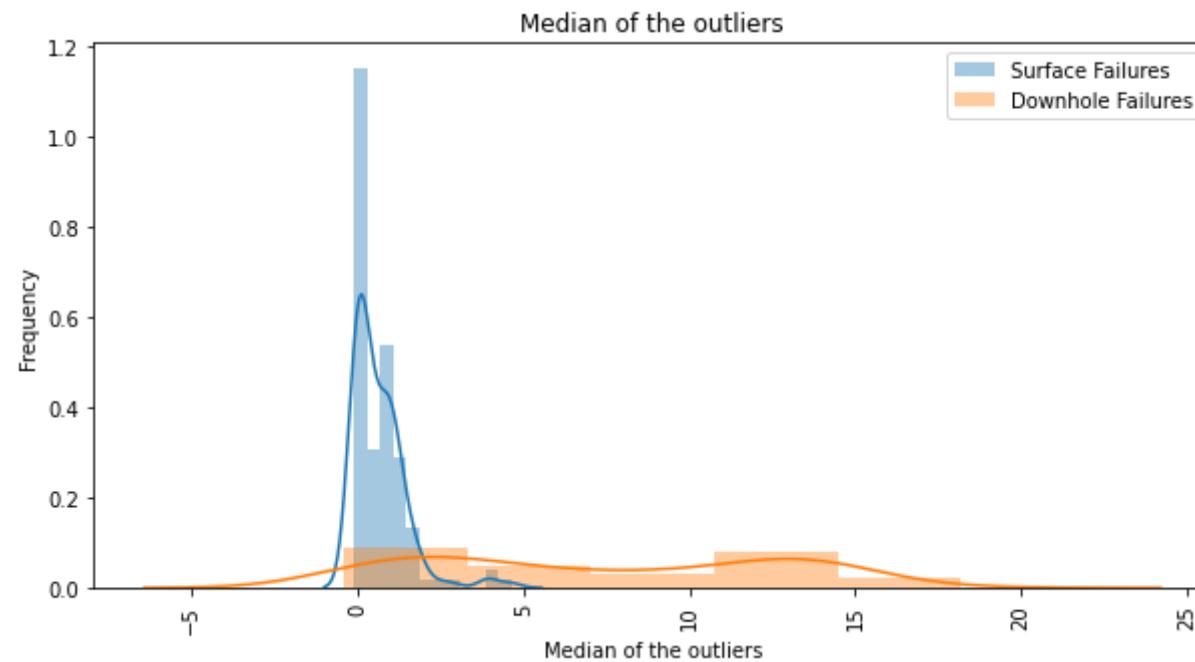


Observation :

More variance for the downhole failures, but more values near to 0 for surface failures.

One of the reasons can be that we have imputed 0 value on surface level failure features, as we have more np.NaN values for the surface failures.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('Median of the outliers')
plt.ylabel('Frequency')
plt.xlabel('Median of the outliers')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```

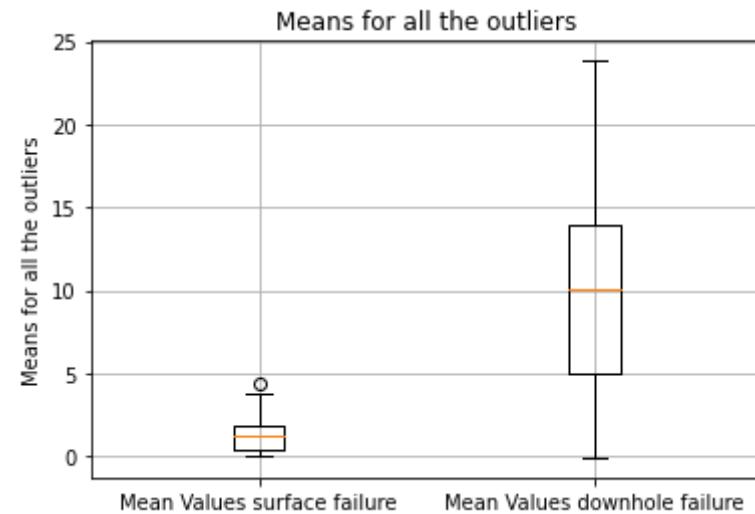


Observation :

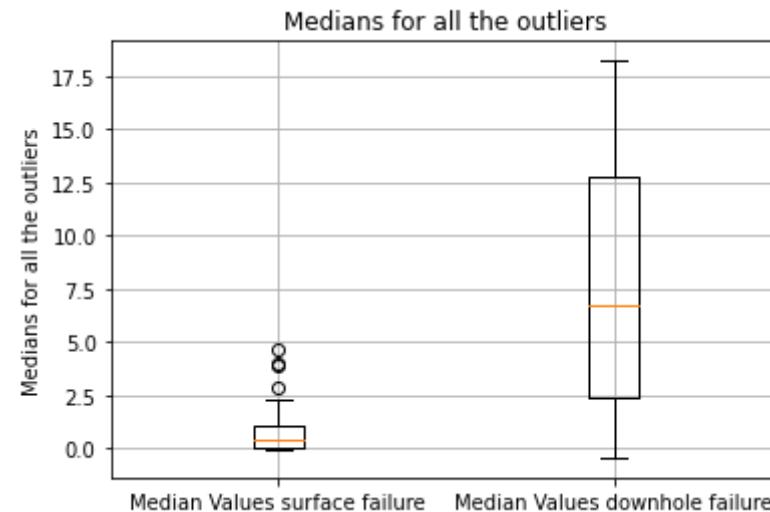
More variance for the downhole outliers, but more values near to 0 for surface outliers.

11) Try box plot / violin plot to visualize the outliers.

```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Means for all the outliers')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the outliers')  
plt.grid()  
plt.show()
```



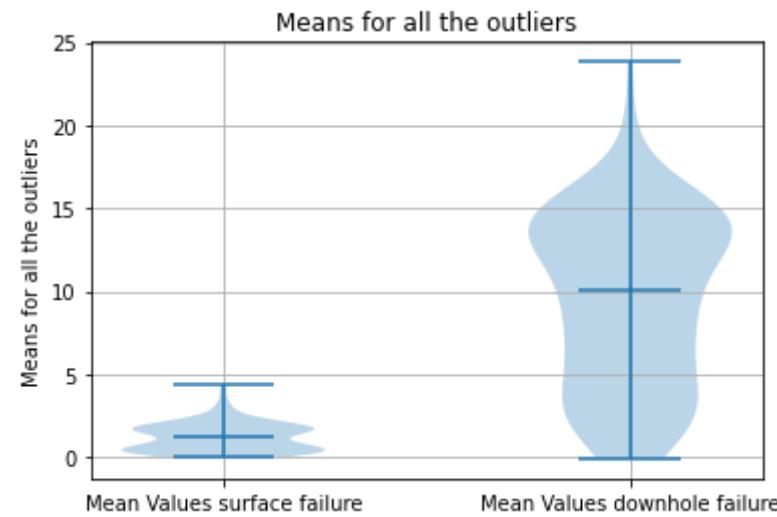
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('Medians for all the outliers')  
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))  
plt.ylabel('Medians for all the outliers')  
plt.grid()  
plt.show()
```



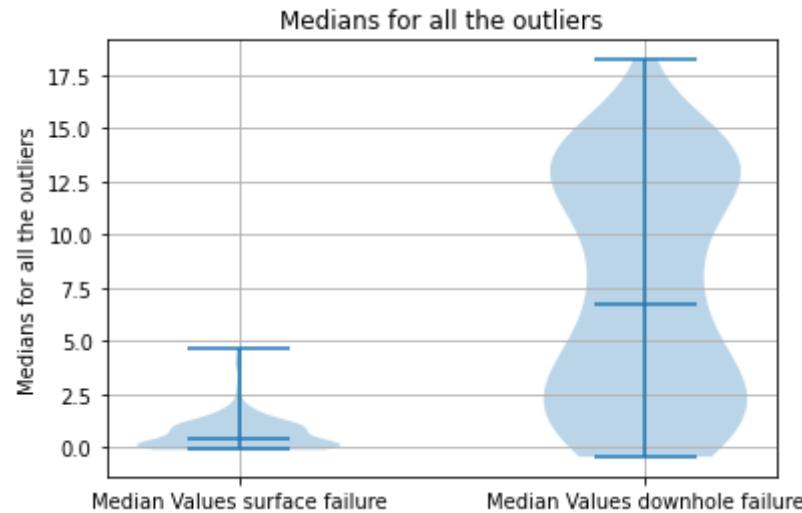
Observations :

We can say that after imputing 0 , the overall value for mean and median is higher for downhole outliers.

```
In [ ]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Means for all the outliers')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the outliers')
plt.grid()
plt.show()
```



```
In [ ]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('Medians for all the outliers')
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))
plt.ylabel('Medians for all the outliers')
plt.grid()
plt.show()
```



Observation :

- 1) If we consider the values for both the means and medians of the outliers then they are less different.
- 2) But if we consider the values for both the means and medians of the main features , then the difference is more.

For the given data points and features enter new features that tell if these values were previously np.NaN or not.

```
In [ ]: new_standard_train_0_all_features_df.drop("target",axis=1,inplace=True)
```

```
In [ ]: def fuse_dataframe_nan(data_dataframe,nan_dataframe,columns):
    output_dataframe=pd.DataFrame()
    for column in columns:
        output_dataframe[column]=nan_dataframe[column]
    return output_dataframe
```

```
In [ ]: nan_standard_train_0_all_features_df=fuse_dataframe_nan(new_standard_train_0_all_features_df,nan_or_not,set(
nan_standard_train_0_all_features_df["target"]=y_train_0_all_features
nan_standard_train_0_all_features_df.head()
```

Out[166]:

	sensor7_histogram_bin2	sensor17_measure	sensor105_histogram_bin3	sensor69_histogram_bin7	sensor64_histogram_bin2	sensor40_measure
0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	1.0

5 rows × 133 columns

Now this is the problem out of the given data that we have , we select the features that have more than 50% np.NaNs

```
In [ ]: counter=0
for column in nan_standard_train_0_all_features_df.columns.values:
    percent_of_nan=nan_standard_train_0_all_features_df[column].value_counts()[1]/\
        (nan_standard_train_0_all_features_df[column].value_counts()[0]+\
         nan_standard_train_0_all_features_df[column].value_counts()[1])
    if (percent_of_nan>.5) :
        counter=counter+1
        print(column," : ",percent_of_nan)
print("Overall : ",counter," features with % of na's more than 50%")

sensor40_measure      :  0.77375
sensor41_measure      :  0.7967916666666667
sensor42_measure      :  0.8133125
sensor38_measure      :  0.6605416666666667
sensor43_measure      :  0.822
sensor39_measure      :  0.7348958333333333
Overall :  6  features with % of na's more than 50%
```

2) Impute np.NaN with mean values with all features.

Now I am taking all the features and replacing the np.NaN readings with the mean of the class to check if PCA and tSNE can visualise the 2 failures separately.

```
In [5]: data_type="mean_impute_data"
```

```
In [ ]: train_mean_all_features=train.copy()
y_train_mean_all_features=y_train.copy()
test_mean_all_features=test.copy()
y_test_mean_all_features=y_test.copy()
```

```
In [ ]: train_mean_all_features.shape
```

Out[178]: (48000, 170)

```
In [ ]: y_train_mean_all_features.shape
```

Out[179]: (48000,)

```
In [ ]: test_mean_all_features.shape
```

```
Out[180]: (12000, 170)
```

```
In [ ]: y_test_mean_all_features.shape
```

```
Out[182]: (12000,)
```

```
In [ ]: for i,j in zip(train_mean_all_features.columns.values,test_mean_all_features.columns.values):
         if (i!=j):
             print(i," : ",j)
```

```
In [ ]: train_mean_all_features,y_train_mean_all_features=reset_index(train_mean_all_features,y_train_mean_all_features)
test_mean_all_features,y_test_mean_all_features=reset_index(test_mean_all_features,y_test_mean_all_features)
```

Filling the np.NaN values with the mean of the category this time.

```
In [ ]: l=[]
        for ele in train_mean_all_features["sensor4_measure"]:
            if np.isnan(ele):
                pass
            else:
                l.append(ele)
        np.mean(l)
```

```
Out[186]: 452.84097318219517
```

```
In [ ]: train_mean_all_features["sensor4_measure"].mean()
```

```
Out[187]: 452.84097
```

Both are the same so the pandas.series.mean() function ignores the value that has the np.NaN and does not even count it.

Divide the whole dataframe into the number of categories that we have and then only enter the mean for that feature for that particular category.

```
In [ ]: def impute_mean_for_train_n_test_as_per_train(train_dataframe,train_target,
                                                    test_dataframe,test_target,data_type):

    train_1=train_dataframe.loc[train_target[train_target==1].reset_index()["index"]]
    train_0=train_dataframe.loc[train_target[train_target==0].reset_index()["index"]]

    train_dataframe.mean().to_pickle("train_dataframe_mean_values_"+str(data_type)+".pickle")

    test_1=test_dataframe.loc[test_target[test_target==1].reset_index()["index"]]
    #The mean that gets imputed in the test data must be that of the train dataset.
    test_1.fillna(train_1.mean(),inplace=True)
    y_test_1=test_target[test_target==1]
    y_test_1.head()

    test_0=test_dataframe.loc[test_target[test_target==0].reset_index()["index"]]
    #The mean that gets imputed in the test data must be that of the train dataset.
    test_0.fillna(train_0.mean(),inplace=True)
    y_test_0=test_target[test_target==0]
    y_test_0.head()

    test_new = pd.concat([test_0,test_1])
    y_test_new = pd.concat([y_test_0,y_test_1])

    train_1.fillna(train_1.mean(),inplace=True)
    y_train_1=train_target[train_target==1]
    y_train_1.head()

    train_0.fillna(train_0.mean(),inplace=True)
    y_train_0=train_target[train_target==0]
    y_train_0.head()

    train_new = pd.concat([train_0,train_1])
    y_train_new = pd.concat([y_train_0,y_train_1])

return train_new,y_train_new,test_new,y_test_new
```

```
In [ ]: train_mean_all_features,y_train_mean_all_features,test_mean_all_features,y_test_mean_all_features=\
impute_mean_for_train_n_test_as_per_train(train_mean_all_features,
                                            y_train_mean_all_features,
                                            test_mean_all_features,
                                            y_test_mean_all_features,data_type)
```

```
In [ ]: train_mean_all_features,y_train_mean_all_features=reset_index(train_mean_all_features,y_train_mean_all_features)
test_mean_all_features,y_test_mean_all_features=reset_index(test_mean_all_features,y_test_mean_all_features)
```

Standardise the dataframe

```
In [ ]: standard_train_mean_all_features_df,standard_test_mean_all_features_df=standardize_test_n_train_wrt_train(tr
```

```
In [ ]: standard_train_mean_all_features_df.shape
```

```
Out[194]: (48000, 170)
```

```
In [ ]: standard_test_mean_all_features_df.shape
```

```
Out[196]: (12000, 170)
```

```
In [ ]: y_train_mean_all_features.head()
```

```
Out[197]: 0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: target, dtype: float32
```

```
In [ ]: y_train_mean_all_features.tail()
```

```
Out[198]: 47995    1.0
47996    1.0
47997    1.0
47998    1.0
47999    1.0
Name: target, dtype: float32
```

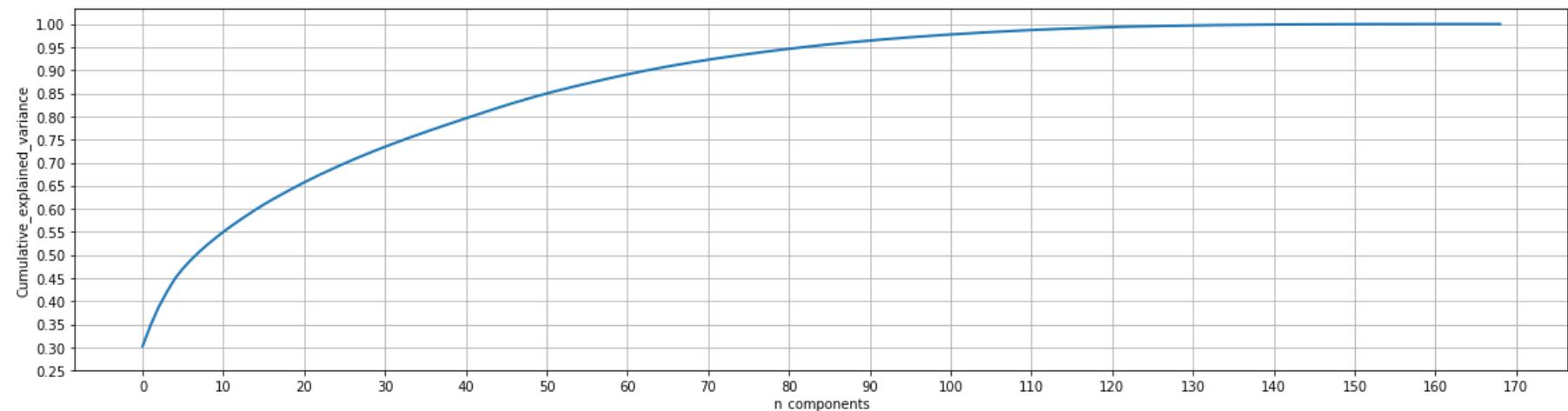
```
In [ ]: "target" in standard_train_mean_all_features.columns
```

```
Out[199]: False
```

Truncated SVD to get the best features

```
In [ ]: #https://stackoverflow.com/questions/50796024/feature-variable-importance-after-a-pca-analysis
```

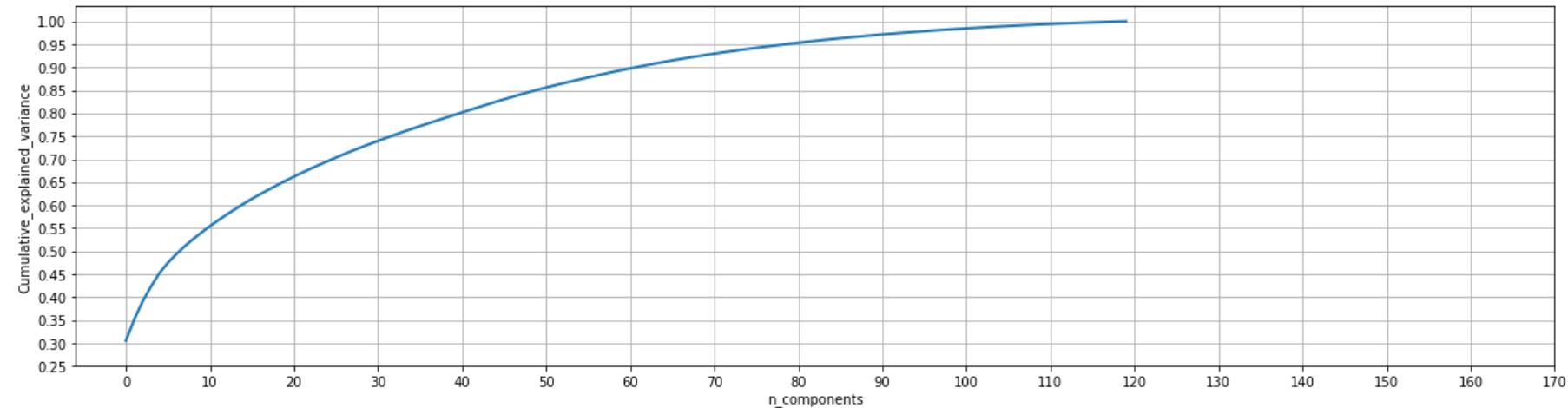
```
In [ ]: truncated_svd(standard_train_mean_all_features_df, 169)
```



```
Out[202]: TruncatedSVD(n_components=169)
```

I choose 120 components to describe my data as 120 components explain atleast 99% variance of the total data.

```
In [ ]: svd=truncated_svd(standard_train_mean_all_features_df, 120)
```



For the 120 principal components using svd.explained_variance_ratio_ find the variance distribution.

```
In [ ]: print(len(svd.explained_variance_ratio_))
```

120

```
In [ ]: np.sum(svd.explained_variance_ratio_)
```

Out[205]: 0.9927956

```
In [ ]: svd.components_.shape
```

Out[206]: (120, 170)

```
In [ ]: #number of features  
svd.components_.shape[1]
```

Out[207]: 170

```
In [ ]: #number of components  
svd.components_.shape[0]
```

Out[208]: 120

For the 120 principal components find the feature importance.

```
In [ ]: set_number_of_features=120
```

Trying Forward Feature selection

Here, in forward feature selection, out of 170 features even if I choose to select 10 features only , still it will take a lot of time.

Therefore I go with recursive feature elimination.

Using Recursive Feature Elimination, I can get the features that I need.

```
In [ ]: new standard train mean all features df.head()
```

Out[211]:

	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_
0	-0.209185	-0.006580	-0.457194	-0.009687	-0.028955	-0.058156	-0.11
1	0.226385	-0.006580	2.302813	0.248253	-0.028955	-0.058156	-0.11
2	-0.411601	-0.412396	-0.457193	-0.136908	-0.028955	-0.058156	-0.11
3	0.027280	-0.006580	-0.457192	0.274966	-0.028955	-0.058156	-0.11
4	-0.132515	-0.006580	-0.457193	0.002868	-0.028955	-0.058156	-0.11

5 rows × 145 columns

Try spearman corelation coefficient to find out co relation. (It is good for cause and affect analysis but it might not be of much use when we find that our features are dependent on other features, we expect our features to be independent of each other.)

```
In [ ]: %%time
spearman=select_spearman(new_standard_train_mean_all_features_df,int(set_number_of_features+np.floor((len(tr
If I take the data having correlation between only -.1 and .1 range then I get 0 features.
If I take the data having correlation between only -.90 and .90 range then I get 60 features.
If I take the data having correlation between only -.95 and .95 range then I get 89 features.
If I take the data having correlation between only -.97 and .97 range then I get 103 features.
If I take the data having correlation between only -.98 and .98 range then I get 112 features.
If I take the data having correlation between only -.99 and .99 range then I get 127 features.
If I take the data having correlation between only -.999 and .999 range then I get 140 features.
CPU times: user 16.4 s, sys: 149 ms, total: 16.6 s
Wall time: 16.5 s
```

```
In [ ]: new_standard_train_mean_all_features_df=perform_spearman(new_standard_train_mean_all_features_df,spearman,.9
```

```
In [ ]: new_standard_train_mean_all_features_df.head()
```

Out[214]:

	sensor2_measure	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_hist
0	-0.006580	-0.457194	-0.009687	-0.028955	-0.058156	-0.116246	
1	-0.006580	2.302813	0.248253	-0.028955	-0.058156	-0.116246	
2	-0.412396	-0.457193	-0.136908	-0.028955	-0.058156	-0.115271	
3	-0.006580	-0.457192	0.274966	-0.028955	-0.058156	-0.116246	
4	-0.006580	-0.457193	0.002868	-0.028955	-0.058156	-0.116246	

5 rows × 127 columns

```
In [ ]: new_standard_train_mean_all_features_df.shape
```

Out[215]: (48000, 127)

```
In [ ]: new_standard_test_mean_all_features_df=new_standard_mean_all_features_df[new_standard_train_mean_all_featu  
new_standard_mean_all_features_df.head()
```

Out[216]:

	sensor2_measure	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_hist
0	-0.006580	-0.457194	-0.009687	-0.028955	-0.058156	-0.116246	
1	-0.006580	2.302813	0.107235	-0.028955	-0.058156	-0.116246	
2	-0.412396	-0.457194	-0.143741	-0.028955	-0.058156	-0.116246	
3	-0.412396	2.302813	-0.143120	-0.028955	-0.058156	-0.116246	
4	-0.006580	2.302813	0.013429	-0.028955	-0.058156	-0.116246	

5 rows × 127 columns

```
In [ ]: #new_standard_train_mean_all_features_df.to_pickle("new_standard_train_mean_all_features_df.pickle")  
#y_train_mean_all_features.to_pickle("y_train_mean_all_features.pickle")  
  
#new_standard_test_mean_all_features_df.to_pickle("new_standard_test_mean_all_features_df.pickle")  
#y_test_mean_all_features.to_pickle("y_test_mean_all_features.pickle")
```

```
In [11]: #new_standard_train_mean_all_features_df=pd.read_pickle("/content/gdrive/My Drive/new_standard_train_mean_all_<ipython>  
#y_train_mean_all_features=pd.read_pickle("/content/gdrive/My Drive/y_train_mean_all_features.pickle")  
  
#new_standard_test_mean_all_features_df=pd.read_pickle("/content/gdrive/My Drive/new_standard_test_mean_all_<ipython>  
#y_test_mean_all_features=pd.read_pickle("/content/gdrive/My Drive/y_test_mean_all_features.pickle")
```

```
In [ ]: np.save('new_standard_train_mean_all_features_df_columns.npy',new_standard_train_mean_all_features_df.columns)
```

```
In [12]: new_standard_train_mean_all_features_df["target"] = y_train_mean_all_features
new_standard_train_mean_all_features_df.head()
```

Out[12]:

	sensor2_measure	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_hist
0	-0.006580	-0.457194	-0.009687	-0.028955	-0.058156	-0.116246	
1	-0.006580	2.302813	0.248253	-0.028955	-0.058156	-0.116246	
2	-0.412396	-0.457193	-0.136908	-0.028955	-0.058156	-0.115271	
3	-0.006580	-0.457192	0.274966	-0.028955	-0.058156	-0.116246	
4	-0.006580	-0.457193	0.002868	-0.028955	-0.058156	-0.116246	

5 rows × 128 columns

Compare the target 0 and target 1 mean values feature wise after mean value imputing

```
In [ ]: column_list=[]
mean_list=[]
one_zero=[]
for column in list(new_standard_train_mean_all_features_df.columns.values):
    if column!="target":

        column_list.append(column)
        one_zero.append(0)
        mean_list.append(new_standard_train_mean_all_features_df[column][new_standard_train_mean_all_features_df["target"]==0].mean())
        new_standard_train_mean_all_features_df[column]=new_standard_train_mean_all_features_df[column].fillna(mean_list[-1])

        column_list.append(column)
        one_zero.append(1)
        mean_list.append(new_standard_train_mean_all_features_df[column][new_standard_train_mean_all_features_df["target"]==1].mean())
        new_standard_train_mean_all_features_df[column]=new_standard_train_mean_all_features_df[column].fillna(mean_list[-1])
```

Find the features where the difference between the means of the two classes is more than .15
descending sorted by class 1

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:mean_list[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

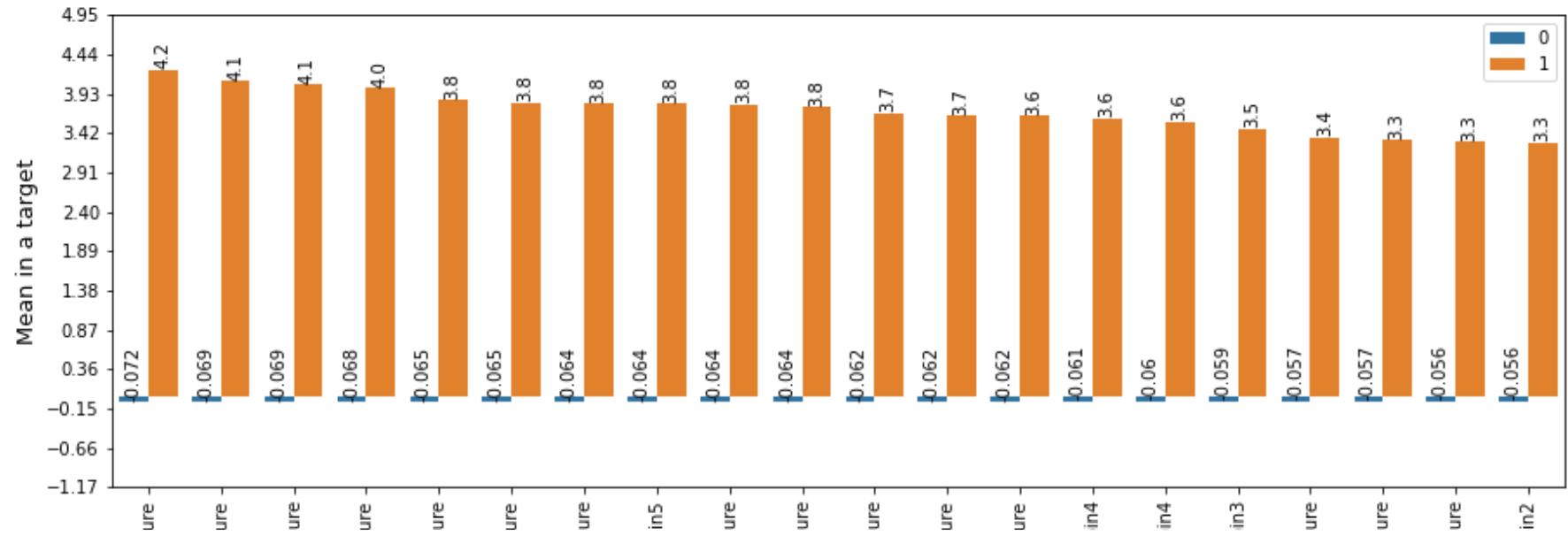
```
In [ ]: counter=0
new_column_list=[]
new_mean_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_mean_list.append(mean_list[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_mean_list.append(mean_list[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 127 sensors.

In []:

```
try:  
    plot_diagram(new_column_list,new_mean_list,new_one_zero,"Mean",1)  
except ValueError:  
    pass
```

Mean in a target value descending sorted by class 1



Observation : Observe that there is a significant difference between the means of the surface failures and the means of the downhole failures. Out of the selected features observe that the most of the features have a larger mean value for the downhole failure rather than that of the surface failures

```
In [ ]: counter=0
mean_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.

    if (np.abs(np.abs(mean_list[i])-np.abs(mean_list[i-1]))>=.15):
        mean_higher_reading.append(mean_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

        mean_higher_reading.append(mean_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

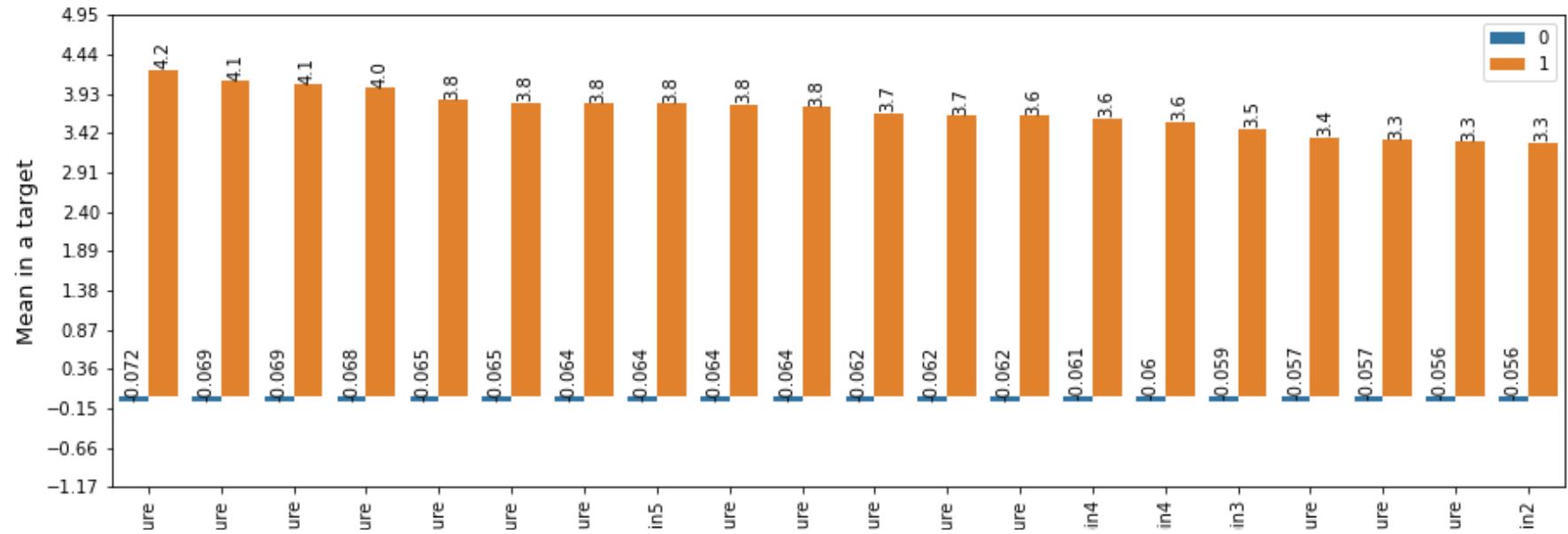
    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between median readings of
```

So we have approximate 124 sensor that show significant difference (> .15) between median readings of surface and downhole failures.

In []: **try:**

```
plot_diagram(column_higher_reading,mean_higher_reading,one_zero_higher_reading,"Mean",1)
except ValueError:
    pass
```

Mean in a target value descending sorted by class 1

**Observations regarding this**

In a descending order for the downhole failure, we can clearly see a difference between the means for the surface failures as well as the downhole failures.

For example sensor 59,16,49 have the highest mean values.

Compare the target 0 and target 1 median values feature wise after mean value imputing

```
In [ ]: column_list=[]
median_list=[]
one_zero=[]

for column in list(new_standard_train_mean_all_features_df.columns.values):
    if column!="target":

        column_list.append(column)
        one_zero.append(0)
        median_list.append(new_standard_train_mean_all_features_df[column][new_standard_train_mean_all_features_df[column].notnull()])

        column_list.append(column)
        one_zero.append(1)
        median_list.append(new_standard_train_mean_all_features_df[column][new_standard_train_mean_all_features_df[column].notnull()])
```

Find the features where the difference between the medians of the two classes is more than .15
descending sorted by class 1

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:median_list[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

```
In [ ]: counter=0
new_column_list=[]
new_median_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_median_list.append(median_list[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_median_list.append(median_list[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 127 sensors.

```
In [ ]: try:
    plot_diagram(new_column_list,new_median_list,new_one_zero,"Median",1)
except ValueError:
    pass
```

Median gives a more accurate measure of a mid value without begin affected by the outliers.

```
In [ ]: counter=0
median_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.

    if (np.abs(np.abs(median_list[i])-np.abs(median_list[i-1]))>=.15):
        median_higher_reading.append(median_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

        median_higher_reading.append(median_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

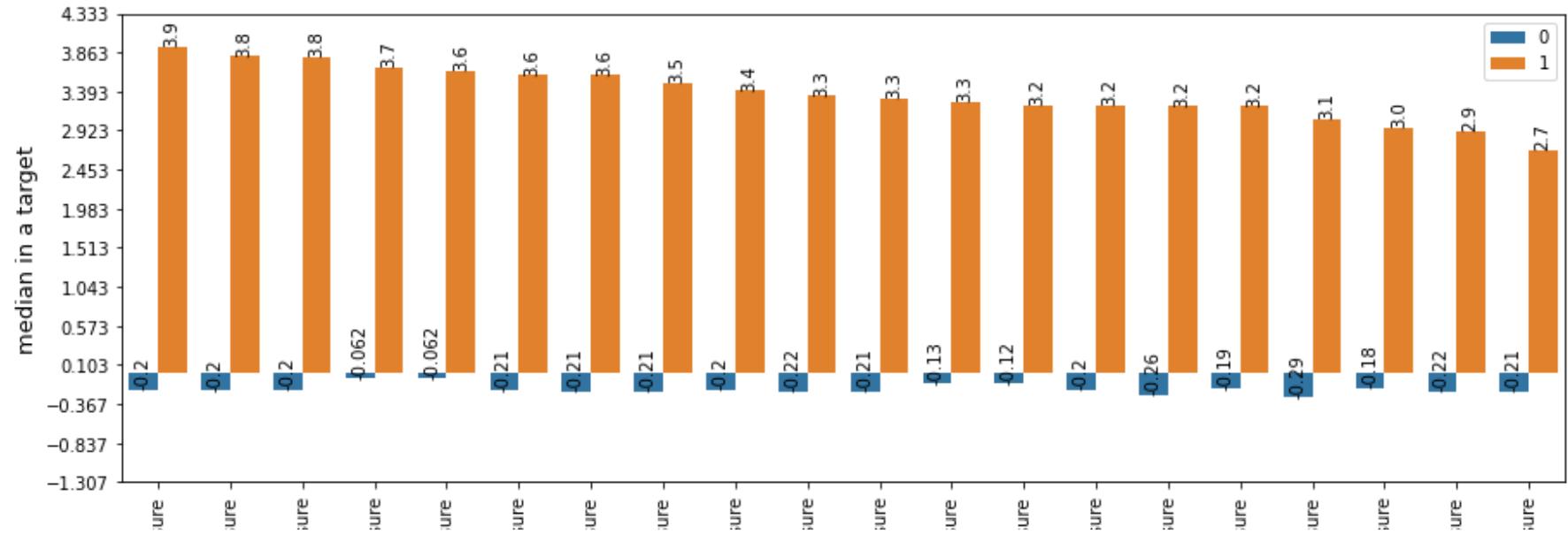
    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between median readings of surface and downhole failures.")
```

So we have approximate 94 sensor that show significant difference (> .15) between median readings of surface and downhole failures.

In []: **try:**

```
plot_diagram(column_higher_reading,median_higher_reading,one_zero_higher_reading,"median",1)
except ValueError:
    pass
```

median in a target value descending sorted by class 1

**Observations regarding this**

- 1) After taking the medians of the given mean imputed values we find those features that have a particular difference in their values.
- 2) All the values which were missing , we filled them with the mean values.

Basic ClassifierIn []: `new_standard_train_mean_all_features_df.drop("target",axis=1,inplace=True)`

```
In [ ]: print(dummy_classifier_based_on_median(new_standard_train_mean_all_features_df.loc[43],
                                              column_list,
                                              median_list))
```

Predicted surface failure.
[120, 7]

```
In [ ]: print(dummy_classifier_based_on_median(new_standard_test_mean_all_features_df.loc[10974],
                                              column_list,
                                              median_list))
```

Predicted surface failure.
[114, 13]

```
In [ ]: new_standard_train_mean_all_features_df["target"] = y_train_mean_all_features
```

EDA for the dataset

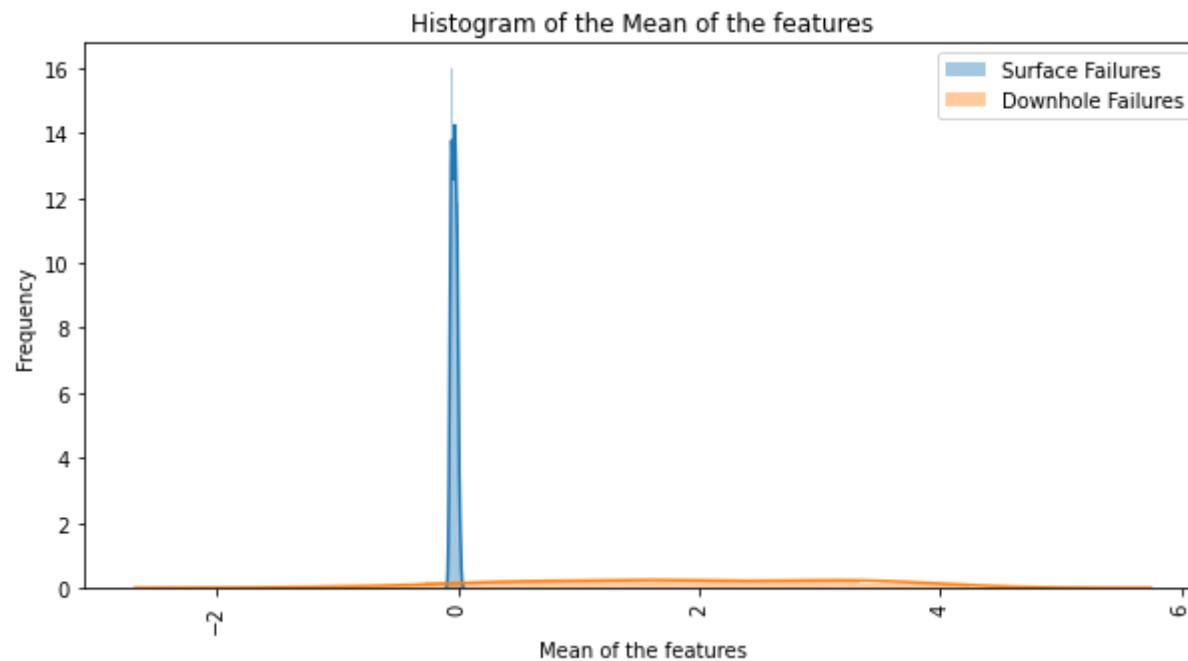
Using histograms, plots find out the nature of means for features for both the classes and thereby, the outliers.

```
In [ ]: mean_0=[]
mean_1=[]
for column in list(new_standard_train_mean_all_features_df.columns.values):
    if column!="target":
        mean_0.append(new_standard_train_mean_all_features_df[new_standard_train_mean_all_features_df.target==0].mean())
        mean_1.append(new_standard_train_mean_all_features_df[new_standard_train_mean_all_features_df.target==1].mean())
```

```
In [ ]: median_0=[]
median_1=[]
for column in list(new_standard_train_mean_all_features_df.columns.values):
    if column!="target":
        median_0.append(new_standard_train_mean_all_features_df[new_standard_train_mean_all_features_df.target==0].median())
        median_1.append(new_standard_train_mean_all_features_df[new_standard_train_mean_all_features_df.target==1].median())
```

10) Try and plot it's histogram for both the target values.

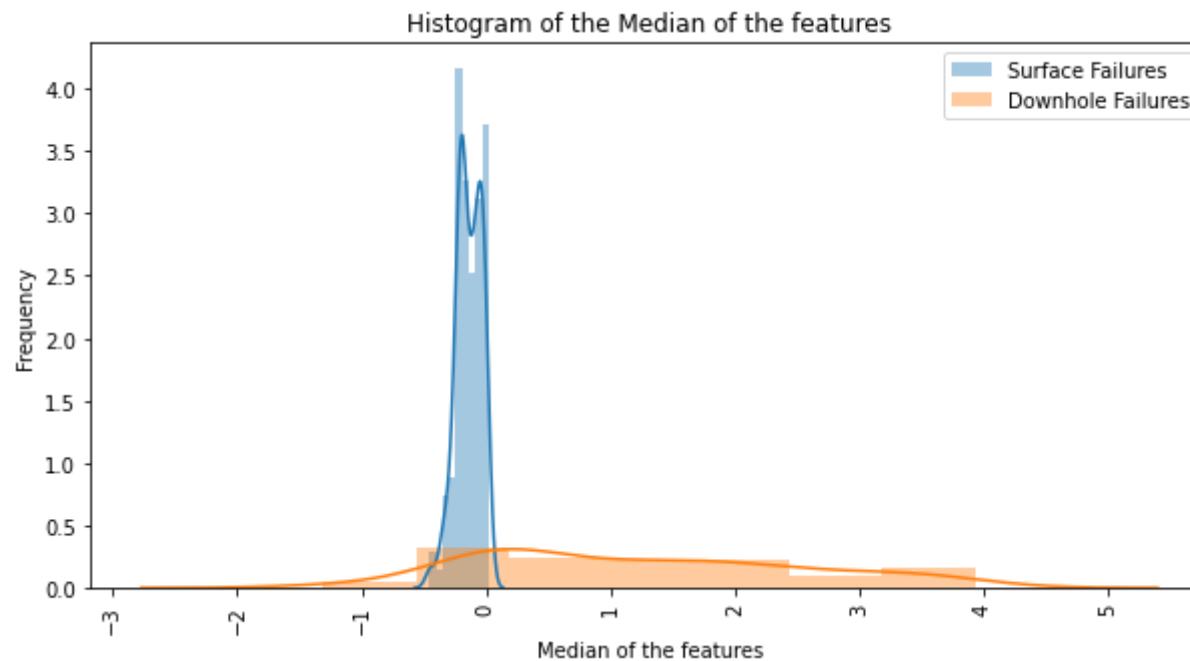
```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Histogram of the Mean of the features')
plt.ylabel('Frequency')
plt.xlabel('Mean of the features')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations :

Here for the surface and downhole failures, we can see that the different downhole features have more variance. Whereas the majority of the surface related features have a very similar value.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('Histogram of the Median of the features')
plt.ylabel('Frequency')
plt.xlabel('Median of the features')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations :

In the histogram of the median values, the extreme values have been reduced to a limit but the main difference is still there, the variance of the downhole failures, more density of feature values near the 0 point for surface points.

11) Try box plot / violin plot to visualize the outliers.

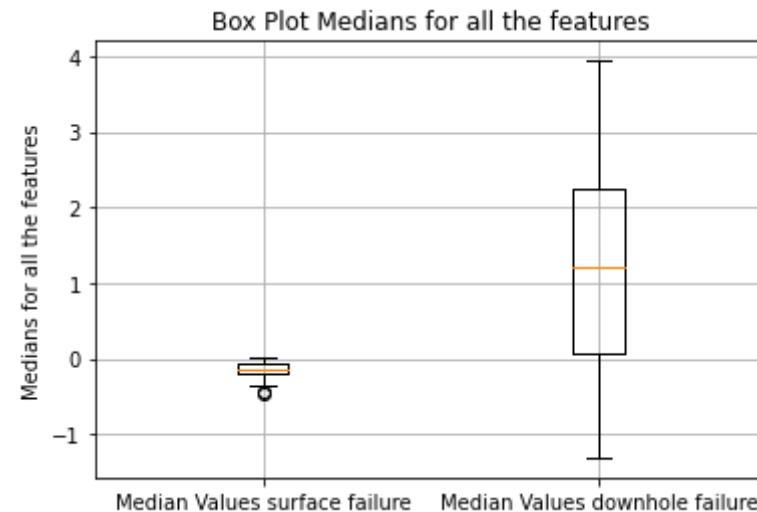
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Box Plot Means for all the features')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the features')  
plt.grid()  
plt.show()
```



Observations :

The majority of the values for the surface failures lie near 0 whereas for the downhole failures, we can see all kinds of variation.

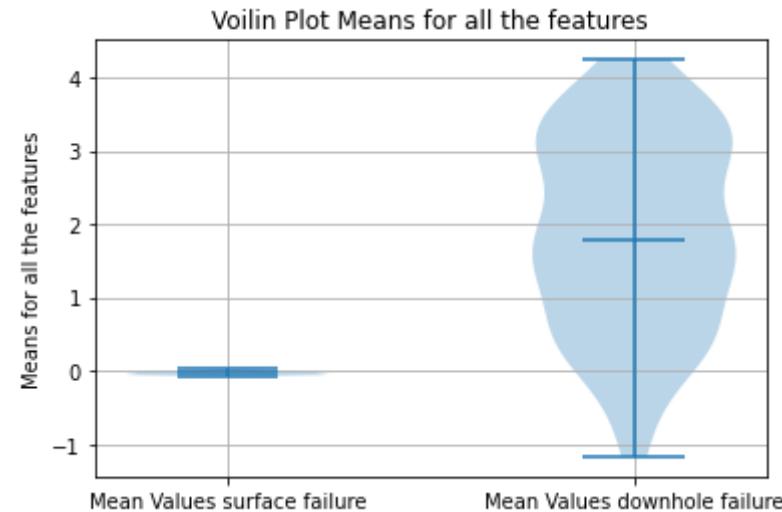
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('Box Plot Medians for all the features')  
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))  
plt.ylabel('Medians for all the features')  
plt.grid()  
plt.show()
```



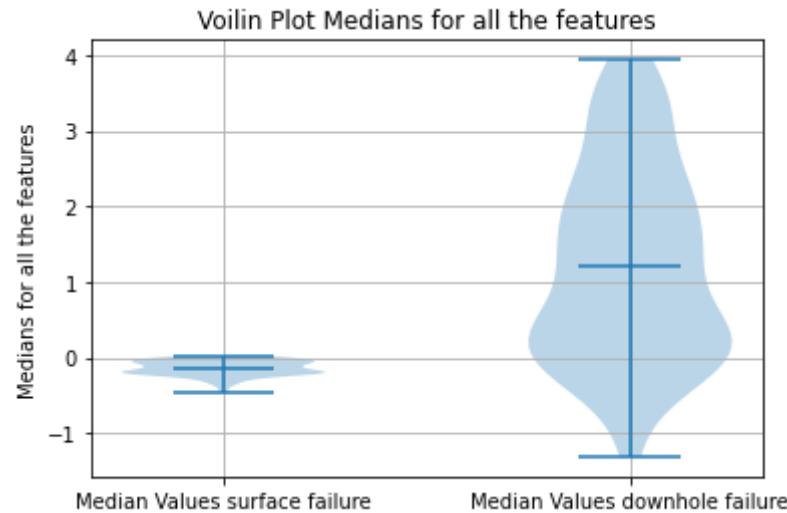
Observations :

The majority of the values for the surface failures lie near 0 whereas for the downhole failures, we can see all kinds of variation.

```
In [ ]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Violin Plot Means for all the features')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the features')
plt.grid()
plt.show()
```



```
In [ ]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('Violin Plot Medians for all the features')
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))
plt.ylabel('Medians for all the features')
plt.grid()
plt.show()
```



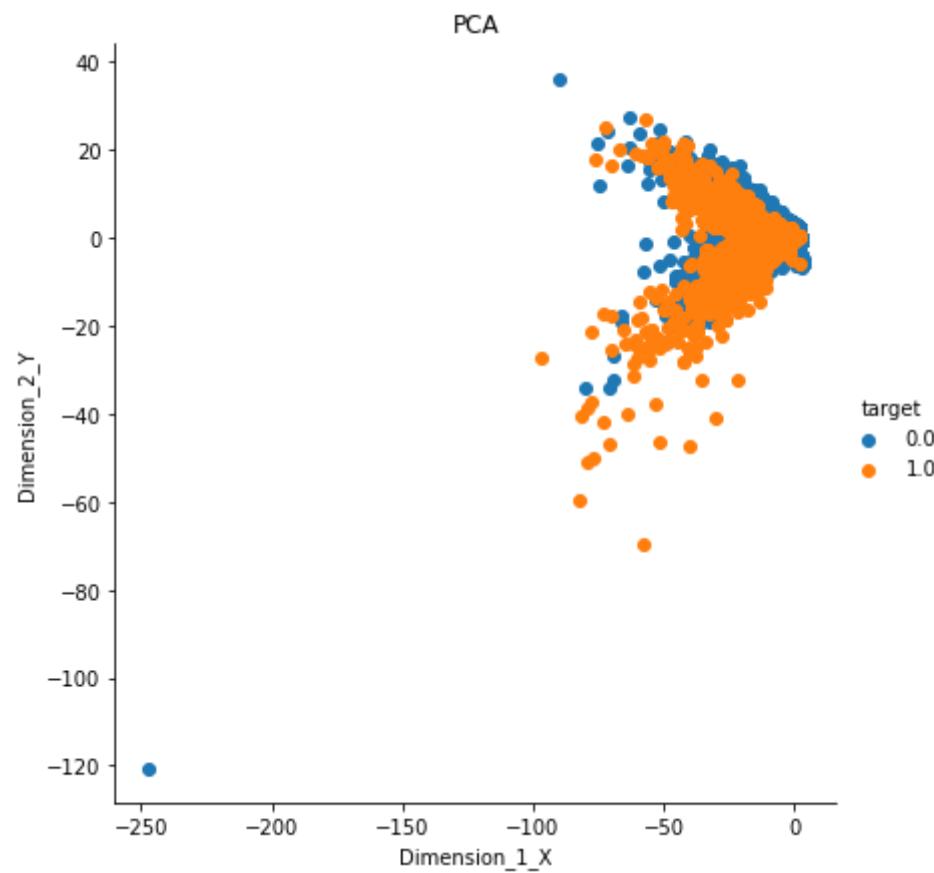
PCA

```
In [ ]: y_train_mean_all_features=new_standard_train_mean_all_features_df.pop("target")
```

In [14]:

```
%%time
run_pca(new_standard_train_mean_all_features_df,y_train_mean_all_features)
```

```
[[ 0.60330766  2.9658337   0.        ]
 [ -2.3501174   1.39494    0.        ]
 [  2.7811787  -0.34686267  0.        ]
 ...
 [-45.243263   -16.67623   1.        ]
 [ -6.3054156   1.2603309   1.        ]
 [ -60.90258   19.436491   1.        ]]
```



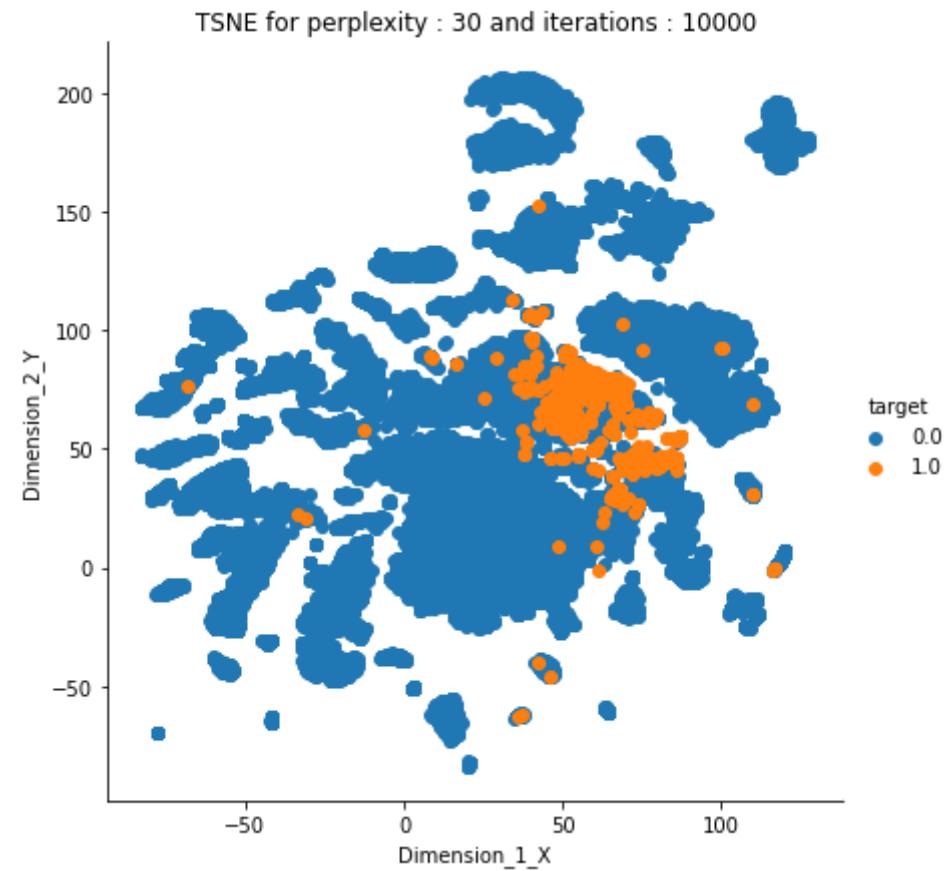
```
CPU times: user 2.11 s, sys: 971 ms, total: 3.08 s
Wall time: 6.55 s
```

tsne

Do not put these values in a loop as we do not get all the output graphs

In []:

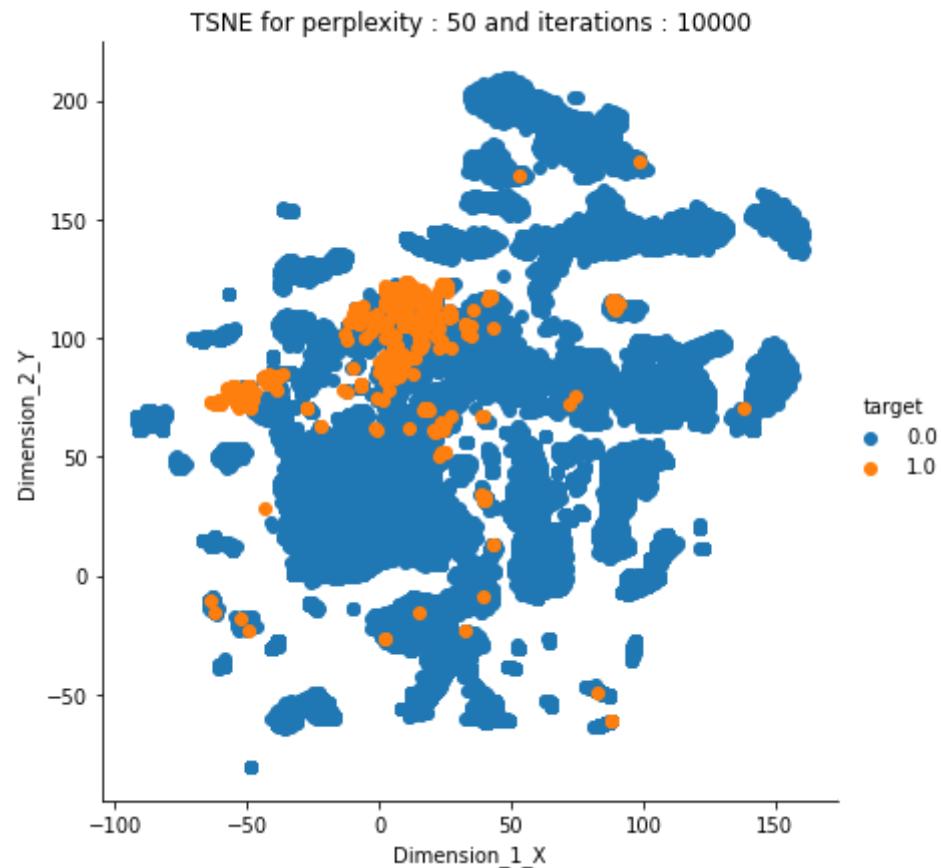
```
%%time  
run_tsne(new_standard_train_new_mean_all_features_df,y_train_mean_all_features,30,10000)
```



```
CPU times: user 13 s, sys: 11.7 s, total: 24.7 s
Wall time: 24.6 s
```

In []:

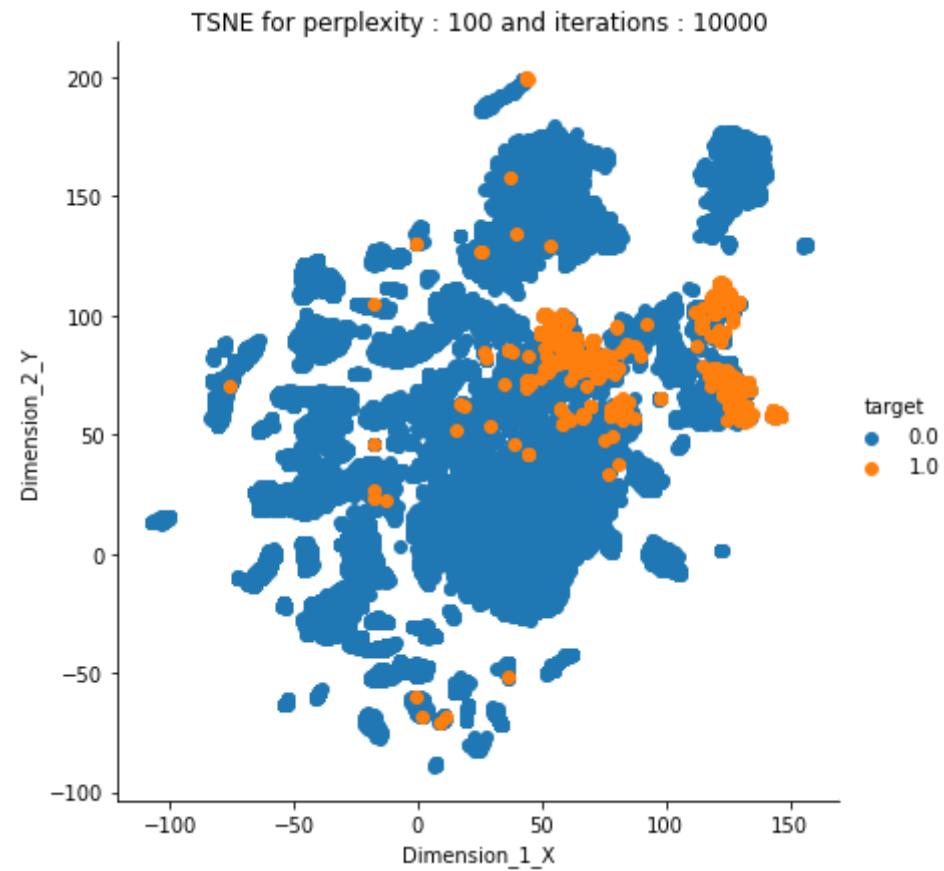
```
%%time  
run_tsne(new_standard_train_new_mean_all_features_df,y_train_mean_all_features,50,10000)
```



CPU times: user 13.1 s, sys: 11.5 s, total: 24.7 s
Wall time: 24.5 s

In []:

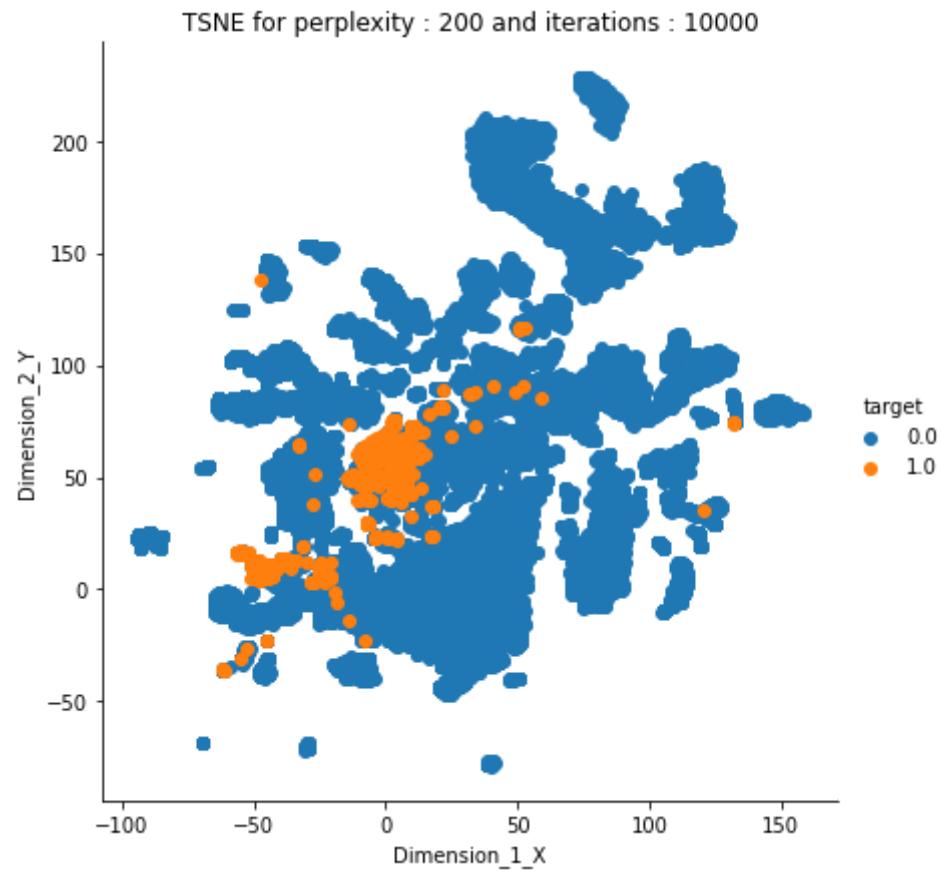
```
%%time  
run_tsne(new_standard_train_new_mean_all_features_df,y_train_mean_all_features,100,10000)
```



CPU times: user 12.8 s, sys: 11.4 s, total: 24.2 s
Wall time: 24 s

In []:

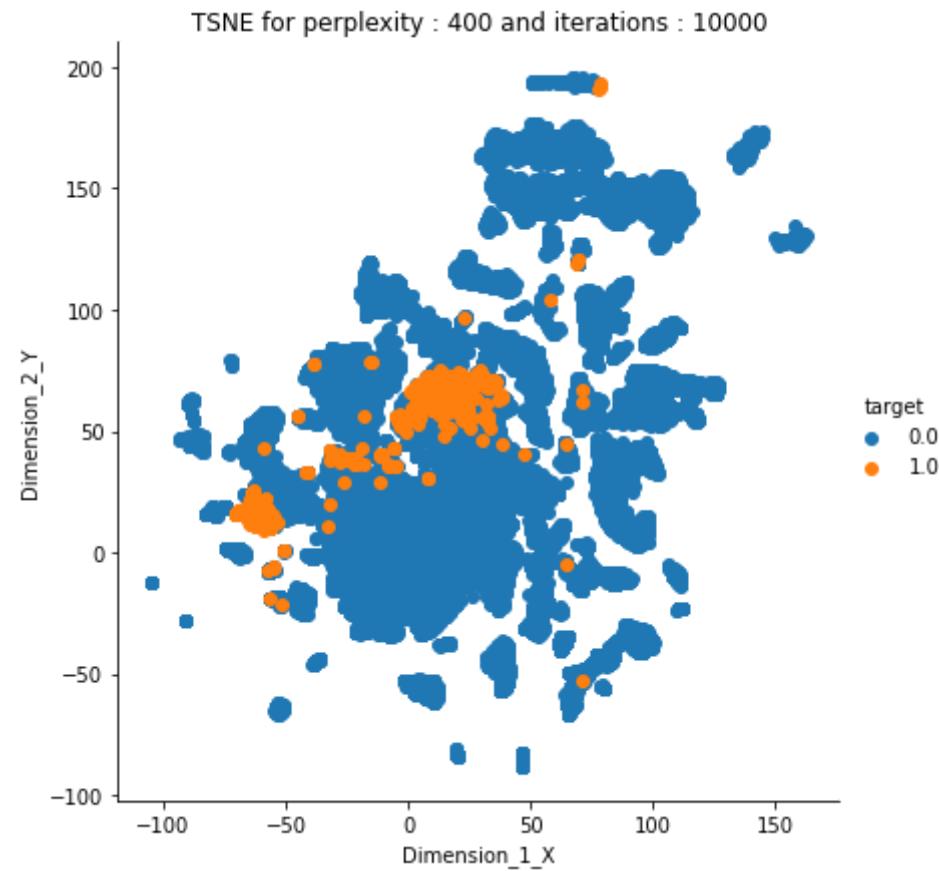
```
%%time  
run_tsne(new_standard_train_new_mean_all_features_df,y_train_mean_all_features,200,10000)
```



CPU times: user 12.6 s, sys: 11.4 s, total: 24 s
Wall time: 23.9 s

In []:

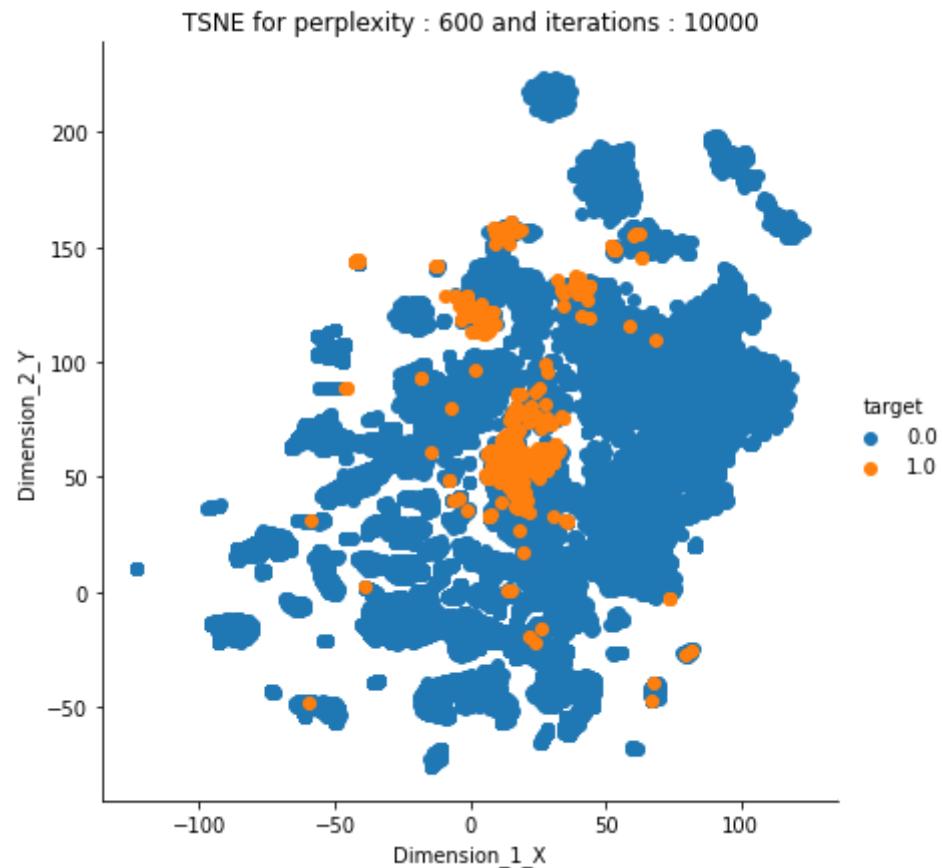
```
%%time  
run_tsne(new_standard_train_new_mean_all_features_df,y_train_mean_all_features,400,10000)
```



CPU times: user 13 s, sys: 11.2 s, total: 24.2 s
Wall time: 24.1 s

In []:

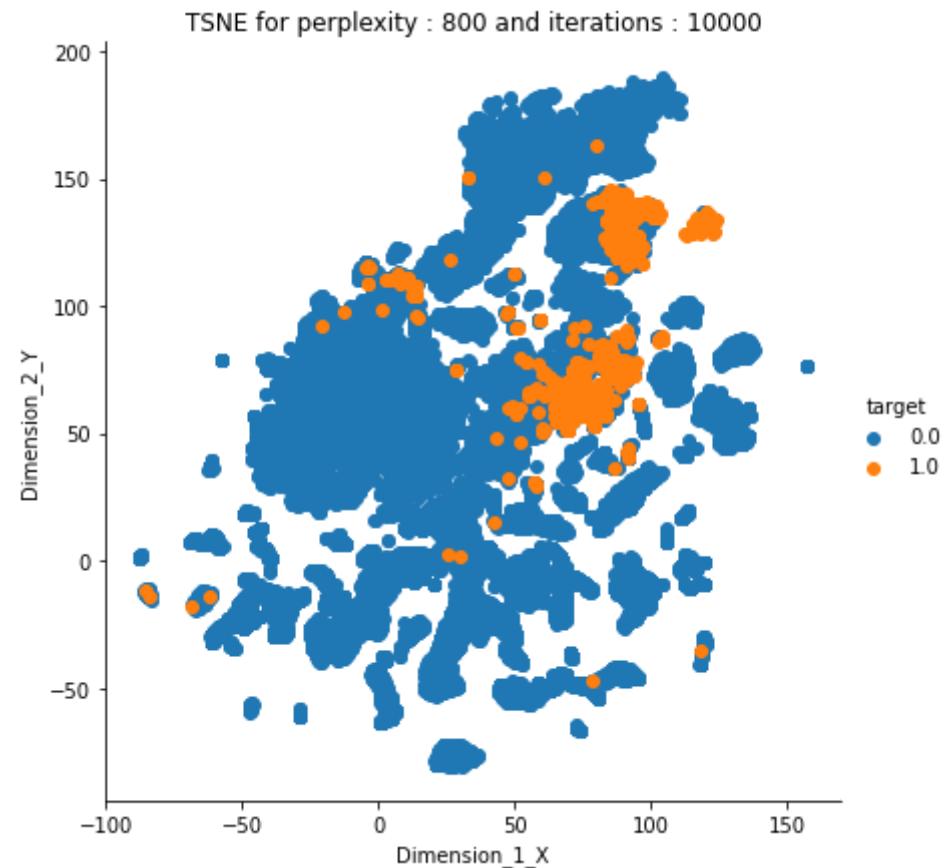
```
%%time  
run_tsne(new_standard_train_new_mean_all_features_df,y_train_mean_all_features,600,10000)
```



CPU times: user 13.1 s, sys: 11.7 s, total: 24.8 s
Wall time: 24.7 s

In []:

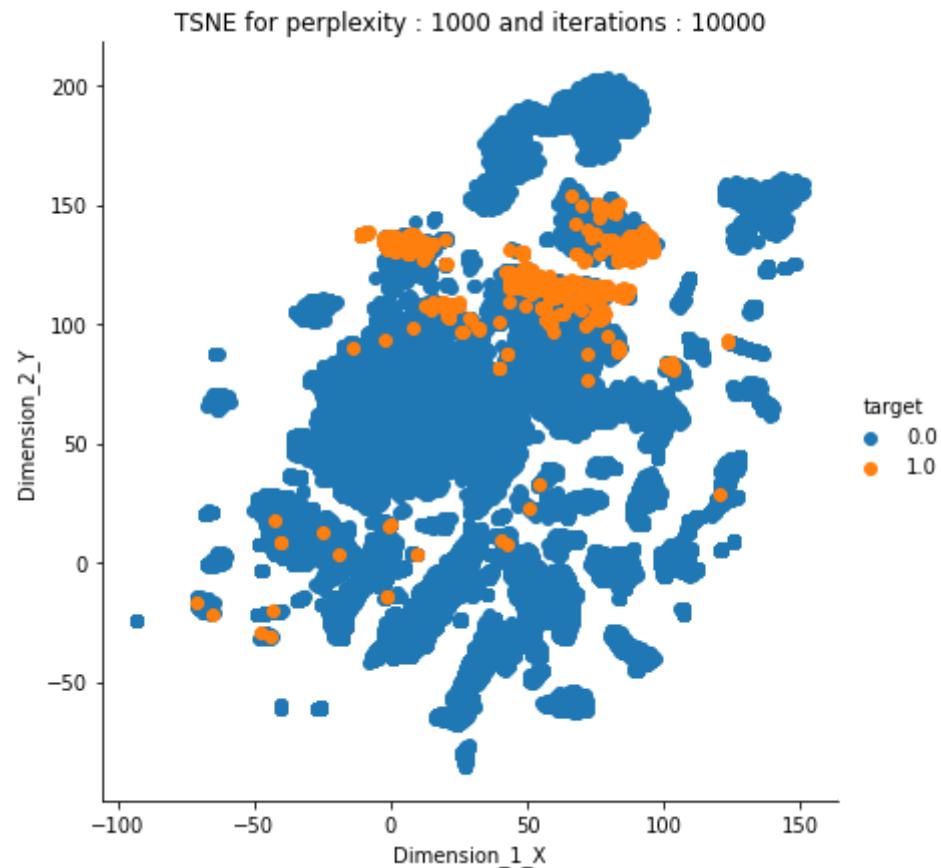
```
%%time  
run_tsne(new_standard_train_new_mean_all_features_df,y_train_mean_all_features,800,10000)
```



CPU times: user 12.9 s, sys: 11.3 s, total: 24.3 s
Wall time: 24.1 s

In []:

```
%%time  
run_tsne(new_standard_train_new_mean_all_features_df,y_train_mean_all_features,1000,10000)
```



CPU times: user 12.8 s, sys: 11.4 s, total: 24.2 s
Wall time: 24 s

Observation :

- 1) Observe that different perplexities tSNE is able to differentiate all the data points into different groups, but it is not able to differentiate properly the surface failures from the downhole failures. If the surface and downhole failures are coming under the same category then they are shown in the same cluster.
- 2) Not all the the surface and downhole features are coming under the same category.

Perform a similar EDA on the outliers.

Here we take the outliers from the features that have been selected.

For each sensor we find the outliers that also classwise.

The issue is that there maybe high outliers here , but there maybe lower outliers here as well.

<https://www.geeksforgeeks.org/interquartile-range-and-quartile-deviation-using-numpy-and-scipy/> (<https://www.geeksforgeeks.org/interquartile-range-and-quartile-deviation-using-numpy-and-scipy/>)

Higher outlier = more than $1.5 \cdot \text{IQR}$ above the third quartile.

Lower outlier = less than $1.5 \cdot \text{IQR}$ below the first quartile.

The IQR is the difference between Q3 and Q1.

```
In [ ]: new_standard_train_mean_all_features_df["target"] = y_train_mean_all_features
```

```
In [ ]: counter=0
main_dictionary=dict()
outliers=[]
list_of_outliers_features_classes=[]
for i in range(len(column_list)):
    d=dict()
    new_list=new_standard_train_mean_all_features_df[new_standard_train_mean_all_features_df.target==one_zero[column_list[i]]]
    new_index=list(new_list.reset_index()["index"])
    QR1=np.percentile(new_list, 25)
    QR3=np.percentile(new_list, 75)
    IQR=QR3-QR1
    for index,value in zip(new_index,new_list):
        if (value>(QR3+(IQR*1.5))) or (value<(QR1-(IQR*1.5))):
            counter=counter+1
            d.update({index:value})
            if index not in main_dictionary:
                main_dictionary.update({index:list(new_standard_train_mean_all_features_df.loc[index])})
    list_of_outliers_features_classes.append(d)
del d
outliers.append(counter)
counter=0
```

```
In [ ]: print("Total number of data points that are outliers in atleast one or more than one feature : ",len(np.unique)
```

```
Total number of data points that are outliers in atleast one or more than one feature :  45060
```

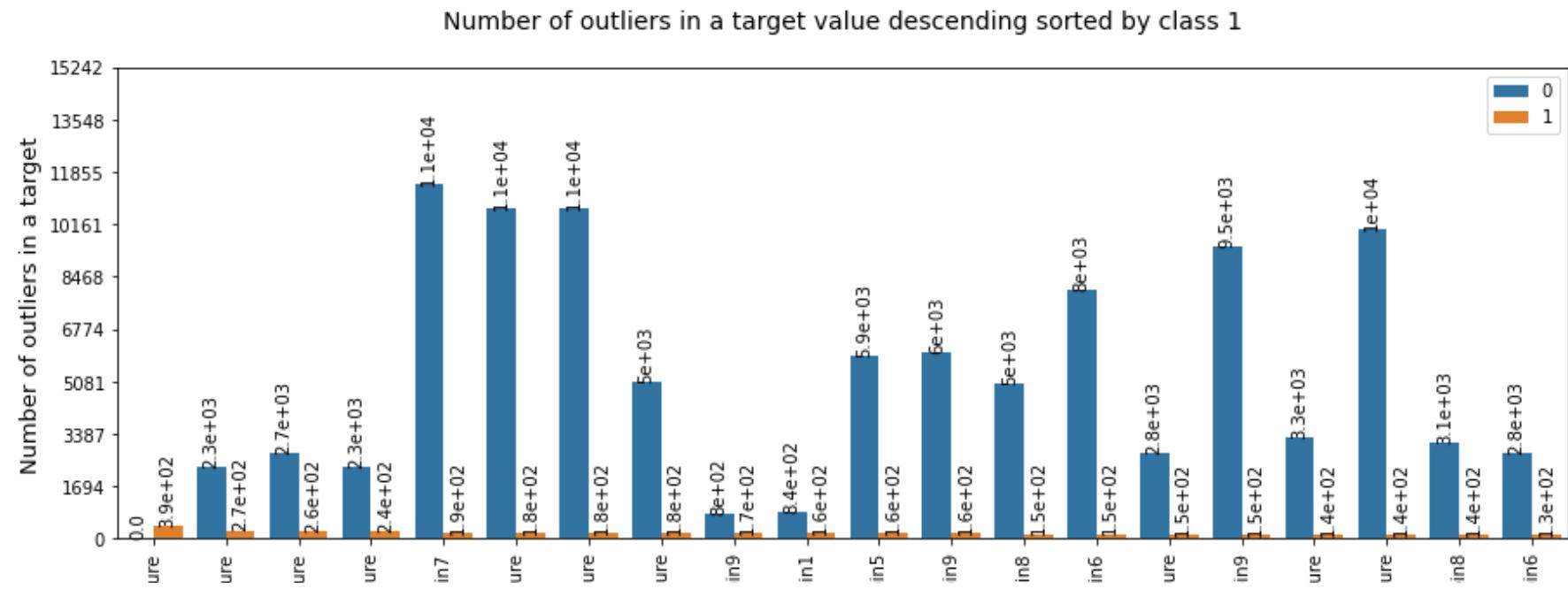
```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:outliers[i] for i in index}

d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])

counter=0
new_column_list=[]
new_outliers_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_outliers_list.append(outliers[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_outliers_list.append(outliers[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 127 sensors.

```
In [ ]: try:  
    plot_diagram(new_column_list,new_outliers_list,new_one_zero,"Number of outliers",1)  
except ValueError:  
    pass
```



Observation :

Number of outliers for surface failures seem to be much more than the number of outliers for the downhole failures. Why ? Because the outliers we have considered for surface failures are of a low value.

For every column feature, get the mean for the outliers for both class 0 and class 1.

```
In [ ]: mean_0=[]
mean_1=[]
for i in range(0,len(list_of_outliers_features_classes),2):
    class_0=[ele for ele in list_of_outliers_features_classes[i].values()]
    class_1=[ele for ele in list_of_outliers_features_classes[i+1].values()]
    if (len(class_0)==0) :
        mean_0.append(0.01)
    else :
        mean_0.append(np.mean(class_0))

    if (len(class_1)==0):
        mean_1.append(0.01)
    else :
        mean_1.append(np.mean(class_1))
```

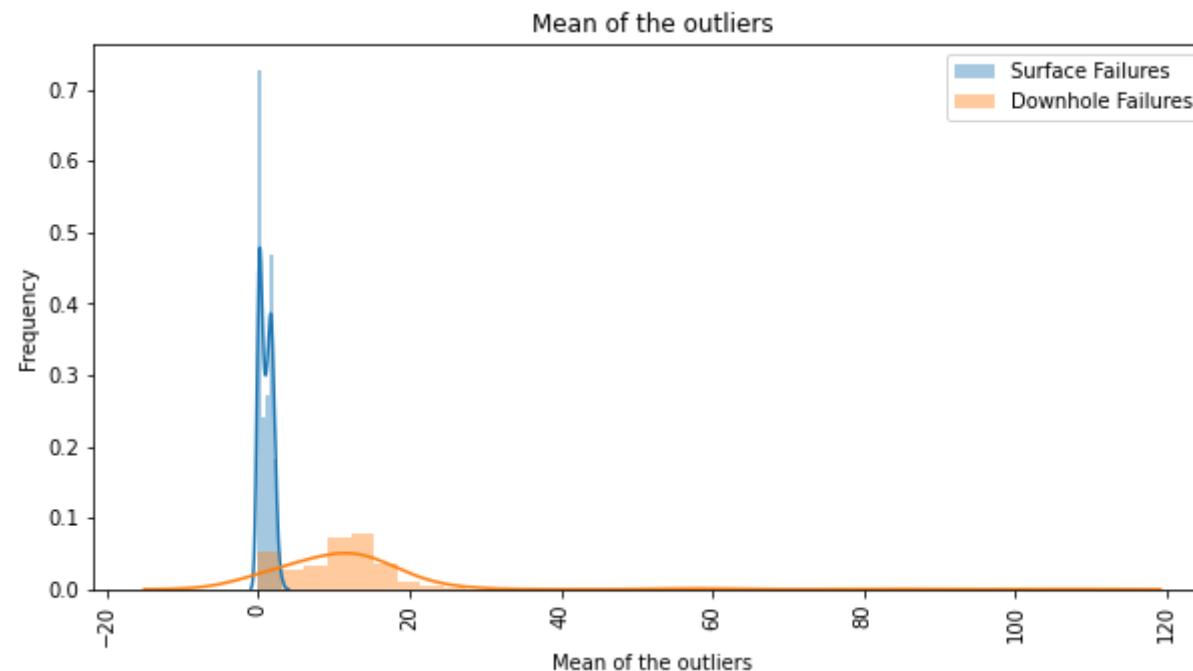
```
In [ ]: median_0=[]
median_1=[]
for i in range(0,len(list_of_outliers_features_classes),2):
    class_0=[ele for ele in list_of_outliers_features_classes[i].values()]
    class_1=[ele for ele in list_of_outliers_features_classes[i+1].values()]
    if (len(class_0)==0) :
        median_0.append(0.01)
    else :
        median_0.append(np.median(class_0))

    if (len(class_1)==0):
        median_1.append(0.01)
    else :
        median_1.append(np.median(class_1))
```

Plotting the mean and median for each feature class wise.

10) Try and plot it's histogram for both the target values.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Mean of the outliers')
plt.ylabel('Frequency')
plt.xlabel('Mean of the outliers')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



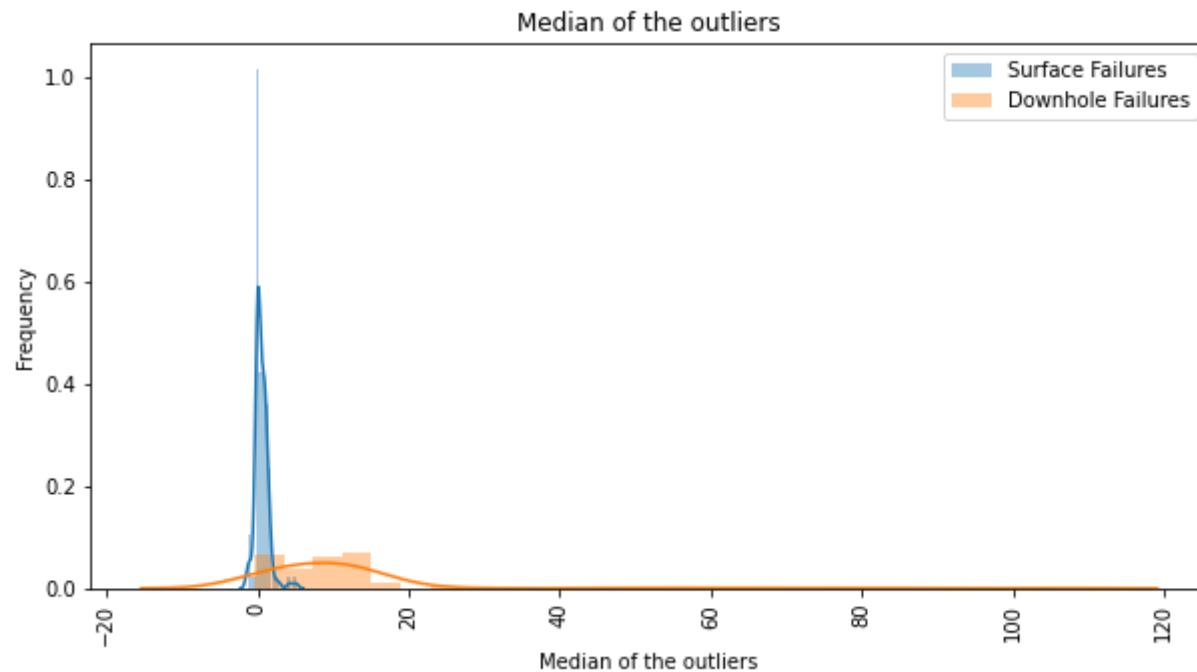
Observations :

The outliers have a similar behaviour to the main features,

The outliers for the surface failures are concentrated around the near 0 value.

The outliers for downhole failures have more variance and not concentrated towards one value.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('Median of the outliers')
plt.ylabel('Frequency')
plt.xlabel('Median of the outliers')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations :

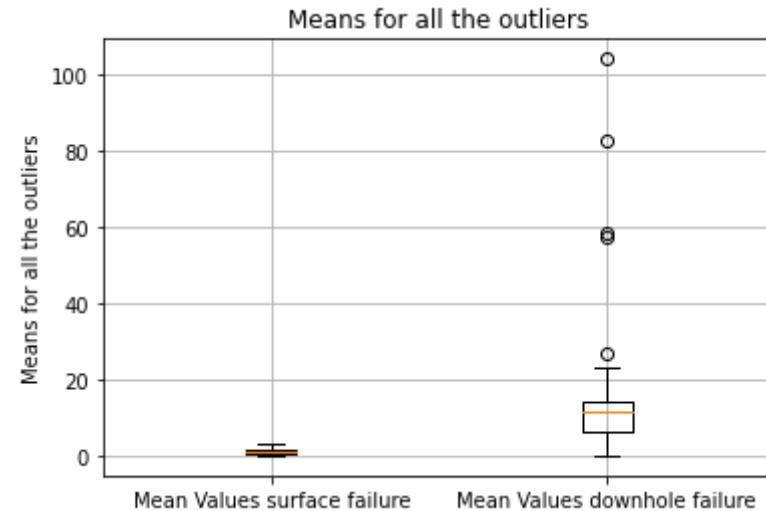
The median outliers for the surface failures are concentrated around the near 0 value.

The median outliers for downhole failures have more variance and not concentrated towards one value.

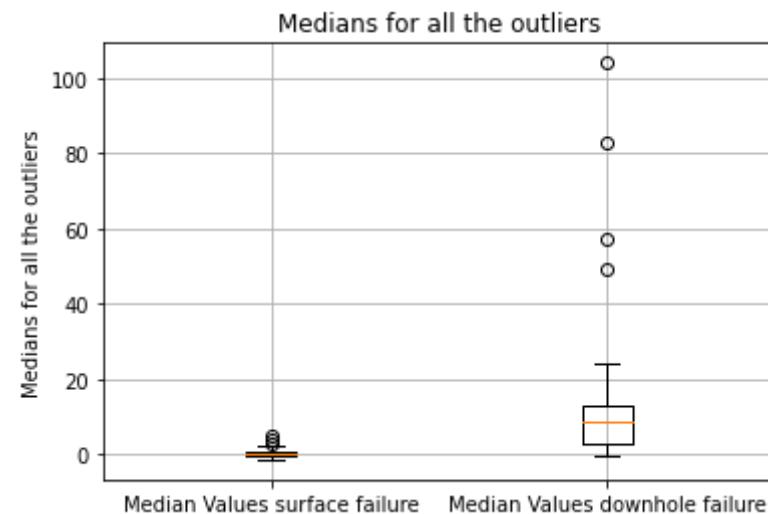
Similar behaviour can be seen in the box plots and violin plots.

11) Try box plot / violin plot to visualize the outliers.

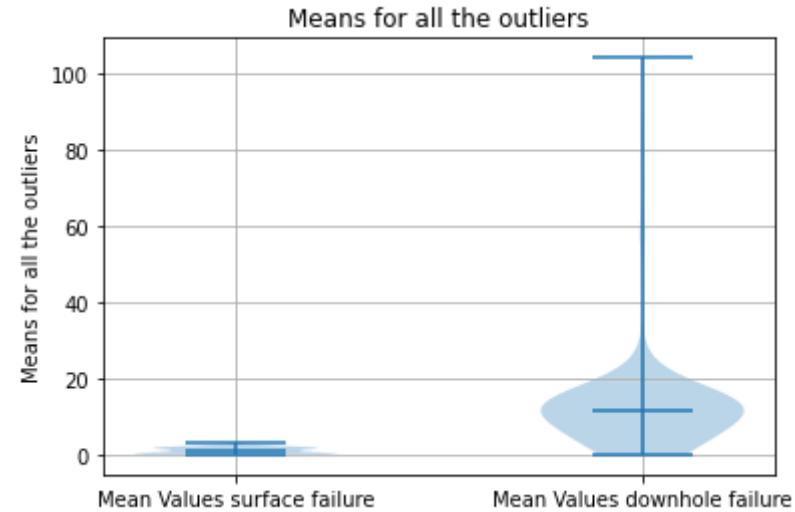
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Means for all the outliers')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the outliers')  
plt.grid()  
plt.show()
```



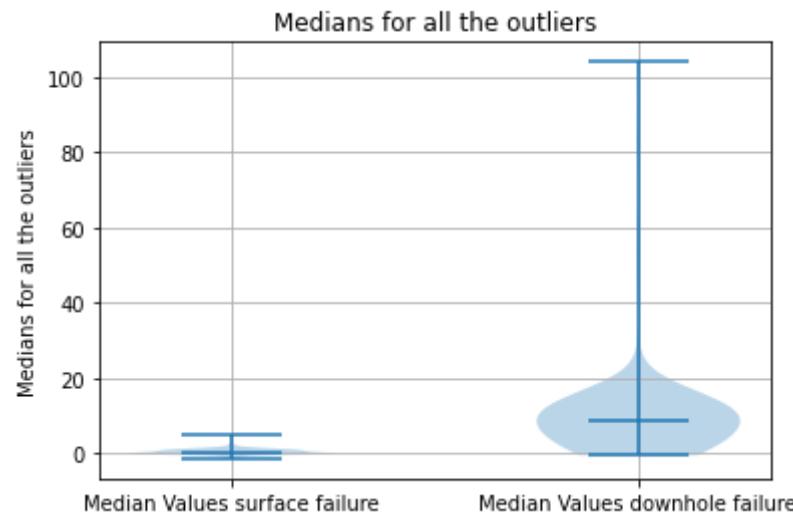
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('Medians for all the outliers')  
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))  
plt.ylabel('Medians for all the outliers')  
plt.grid()  
plt.show()
```



```
In [ ]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Means for all the outliers')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the outliers')
plt.grid()
plt.show()
```



```
In [ ]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('Medians for all the outliers')
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))
plt.ylabel('Medians for all the outliers')
plt.grid()
plt.show()
```



For the given data points and features enter new features that tell if these values were previously np.NaN or not.

```
In [ ]: new_standard_train_mean_all_features_df.drop("target",axis=1,inplace=True)
```

```
In [ ]: nan_standard_train_mean_all_features_df=fuse_dataframe_nan(new_standard_train_mean_all_features_df,nan_or_no  
nan_standard_train_mean_all_features_df["target"]=y_train_mean_all_features  
nan_standard_train_mean_all_features_df.head()
```

Out[265]:

	sensor7_histogram_bin2	sensor17_measure	sensor105_histogram_bin3	sensor69_histogram_bin7	sensor64_histogram_bin2	sensor107_measure
0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 128 columns

Now this is the problem out of the given data that we have , we select the features that have more than 50% np.NaNs

```
In [ ]: counter=0
for column in new_standard_train_mean_all_features_df.columns.values:
    try:
        percent_of_nan=nan_standard_train_mean_all_features_df[column].value_counts()[1]/\
                    (nan_standard_train_mean_all_features_df[column].value_counts()[0]+\
                     nan_standard_train_mean_all_features_df[column].value_counts()[1])
        if (percent_of_nan>.5) :
            counter=counter+1
            print(column," : ",percent_of_nan)
    except:
        pass
print("Overall : ",counter," features with % of na's more than 50%")

sensor2_measure      :  0.7740833333333333
sensor38_measure     :  0.6605416666666667
sensor39_measure     :  0.7348958333333333
sensor40_measure     :  0.77375
sensor41_measure     :  0.7967916666666667
sensor42_measure     :  0.8133125
sensor43_measure     :  0.822
sensor68_measure     :  0.7740833333333333
Overall :  8  features with % of na's more than 50%
```

3) Impute np.NaN with imputing median values with all features.

```
In [7]: data_type="median_impute_data"
```

Now I am using all the features to check if PCA and tSNE can visualise the 2 failures separately.

```
In [ ]: train_median_all_features=train.copy()
y_train_median_all_features=y_train.copy()
test_median_all_features=test.copy()
y_test_median_all_features=y_test.copy()
```

```
In [ ]: train_median_all_features.shape
```

```
Out[268]: (48000, 170)
```

```
In [ ]: y_train_median_all_features.shape
```

```
Out[269]: (48000,)
```

```
In [ ]: test_median_all_features.shape
```

```
Out[270]: (12000, 170)
```

```
In [ ]: y_test_median_all_features.shape
```

```
Out[271]: (12000,)
```

```
In [ ]: for i,j in zip(train_median_all_features.columns.values,test_median_all_features.columns.values):
         if (i!=j):
             print(i," : ",j)
```

```
In [ ]: train_median_all_features,y_train_median_all_features=reset_index(train_median_all_features,y_train_median_a
test_median_all_features,y_test_median_all_features=reset_index(test_median_all_features,y_test_median_all_f
```

Filling the np.NaN values with the median of the category this time.

```
In [ ]: l=[]
         for ele in train_median_all_features["sensor4_measure"]:
             if np.isnan(ele):
                 pass
             else:
                 l.append(ele)
         np.median(l)
```

```
Out[274]: 126.0
```

```
In [ ]: train_median_all_features["sensor4_measure"].median()
```

```
Out[275]: 126.0
```

Both are the same so the pandas.series.median() function ignores the value that has the np.NaN and does not even count it.

Divide the whole dataframe into the number of categories that we have and then only enter the median for that feature for that particular category.


```
In [19]: def impute_median_for_train_n_test_as_per_train(train_dataframe,train_target,
                                                    test_dataframe,test_target,data_type):

    train_1=train_dataframe.loc[train_target[train_target==1].reset_index()["index"]]
    train_0=train_dataframe.loc[train_target[train_target==0].reset_index()["index"]]

    train_1.median().to_pickle("train_1_dataframe_median_values_"+str(data_type)+".pickle")
    train_0.median().to_pickle("train_0_dataframe_median_values_"+str(data_type)+".pickle")

    train_dataframe.median().to_pickle("train_dataframe_median_values_"+str(data_type)+".pickle")

    test_1=test_dataframe.loc[test_target[test_target==1].reset_index()["index"]]
    #The median that gets imputed in the test data must be that of the train dataset.
    test_1.fillna(train_1.median(),inplace=True)
    y_test_1=test_target[test_target==1]
    y_test_1.head()

    test_0=test_dataframe.loc[test_target[test_target==0].reset_index()["index"]]
    #The median that gets imputed in the test data must be that of the train dataset.
    test_0.fillna(train_0.median(),inplace=True)
    y_test_0=test_target[test_target==0]
    y_test_0.head()

    test_new = pd.concat([test_0,test_1])
    y_test_new = pd.concat([y_test_0,y_test_1])

    train_1.fillna(train_1.median(),inplace=True)
    y_train_1=train_target[train_target==1]
    y_train_1.head()

    train_0.fillna(train_0.median(),inplace=True)
    y_train_0=train_target[train_target==0]
    y_train_0.head()

    train_new = pd.concat([train_0,train_1])
    y_train_new = pd.concat([y_train_0,y_train_1])
```

```
return train_new,y_train_new,test_new,y_test_new
```

```
In [ ]: train_median_all_features,y_train_median_all_features,test_median_all_features,y_test_median_all_features=\
impute_median_for_train_n_test_as_per_train(train_median_all_features,
                                              y_train_median_all_features,
                                              test_median_all_features,
                                              y_test_median_all_features,
                                              data_type)
```

```
In [ ]: train_median_all_features,y_train_median_all_features=reset_index(train_median_all_features,y_train_median_a
test_median_all_features,y_test_median_all_features=reset_index(test_median_all_features,y_test_median_all_f
```

Standardise the dataframe

```
In [ ]: standard_train_median_all_features_df,standard_test_median_all_features_df=standardize_test_n_train_wrt_trai
```

```
In [ ]: standard_train_median_all_features_df.shape
```

```
Out[280]: (48000, 170)
```

```
In [ ]: standard_test_median_all_features_df.shape
```

```
Out[281]: (12000, 170)
```

```
In [ ]: y_train_median_all_features.head()
```

```
Out[282]: 0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: target, dtype: float32
```

```
In [ ]: y_train_median_all_features.tail()
```

```
Out[283]: 47995    1.0
47996    1.0
47997    1.0
47998    1.0
47999    1.0
Name: target, dtype: float32
```

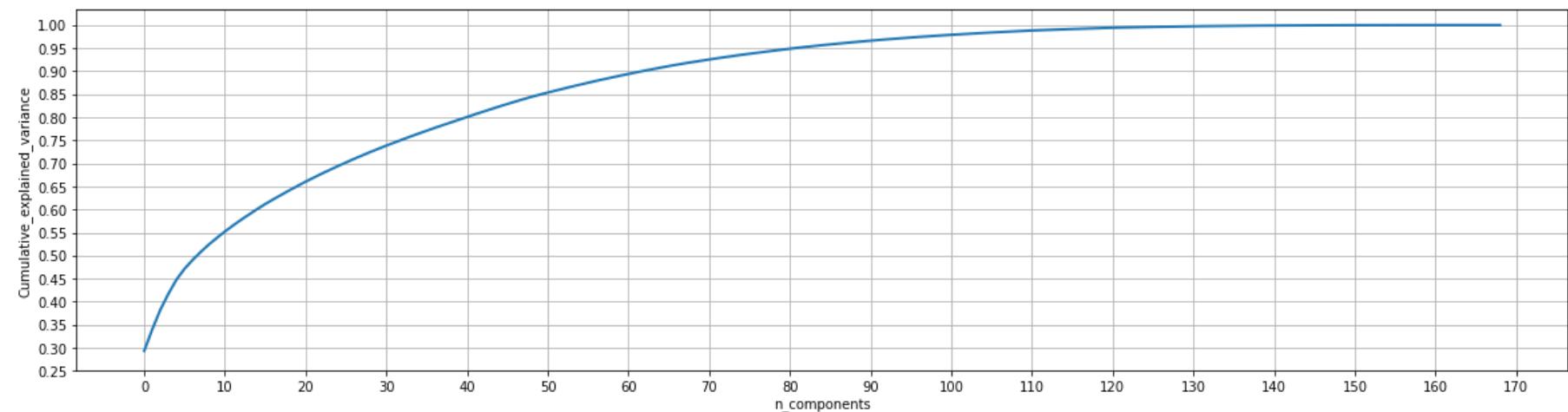
```
In [ ]: "target" in standard_train_median_all_features_df.columns
```

```
Out[284]: False
```

Truncated SVD to get the best features

```
In [ ]: #https://stackoverflow.com/questions/50796024/feature-variable-importance-after-a-pca-analysis
```

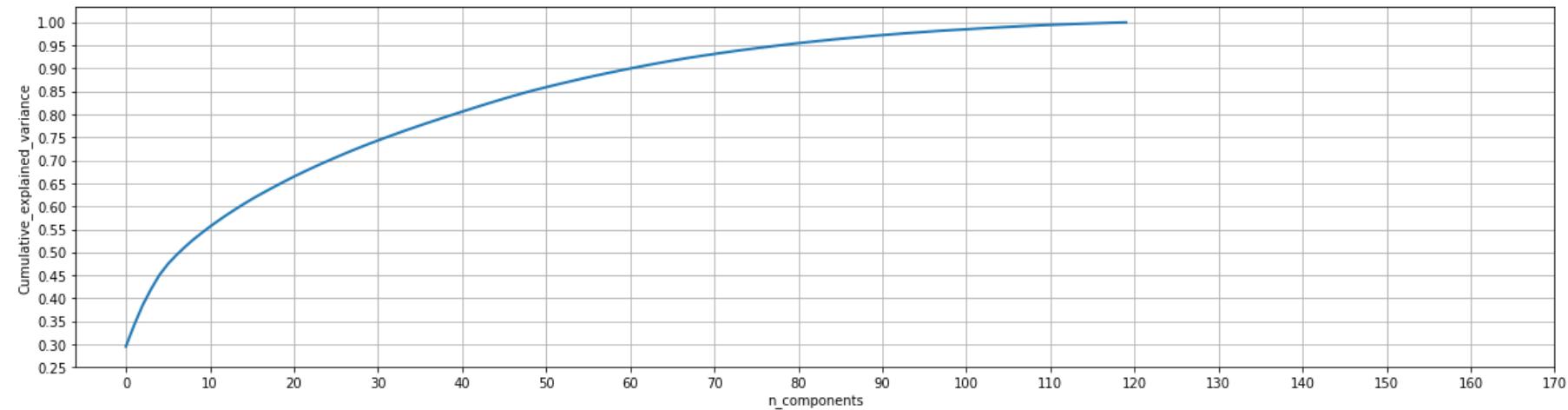
```
In [ ]: truncated_svd(standard_train_median_all_features_df, 169)
```



```
Out[286]: TruncatedSVD(n_components=169)
```

I choose 120 components to describe my data as 120 components explain atleast 99% variance of the total data.

```
In [ ]: svd=truncated_svd(standard_train_median_all_features_df,120)
```



For the 120 principal components using svd.explained_variance_ratio_ find the variance distribution.

```
In [ ]: print(len(svd.explained_variance_ratio_))
```

```
120
```

```
In [ ]: np.sum(svd.explained_variance_ratio_)
```

```
Out[289]: 0.9937009
```

```
In [ ]: svd.components_.shape
```

```
Out[290]: (120, 170)
```

```
In [ ]: #number of features  
svd.components_.shape[1]
```

```
Out[291]: 170
```

```
In [ ]: #number of components  
svd.components_.shape[0]
```

Out[292]: 120

For the 120 principal components find the feature importance.

```
In [ ]: set_number_of_features=120
```

Trying Forward Feature selection

Here, in forward feature selection, out of 170 features even if I choose to select 10 features only , still it will take a lot of time.
Therefore I go with recursive feature elimination.
Using Recursive Feature Elimination, I can get the features that I need.

```
In [ ]: new_standard_train_median_all_features_df.head()
```

Out[295]:

	sensor1_measure	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_hist
0	-0.209185	-0.432133	-0.078741	-0.028788	-0.057799	-0.115367	
1	0.226385	2.314107	0.276872	-0.028788	-0.057799	-0.115367	
2	-0.411601	-0.432133	-0.108583	-0.028788	-0.057799	-0.114391	
3	0.027280	-0.432131	0.303606	-0.028788	-0.057799	-0.115367	
4	-0.132515	-0.432132	0.031300	-0.028788	-0.057799	-0.115367	

5 rows × 145 columns

Try spearman corelation coefficient to find out co relation. (It is good for cause and affect analysis but it might not be of much use when we find that our features are dependent on other features, we expect our features to be independent of each other.)

```
In [ ]: %%time
spearman=select_spearman(new_standard_train_median_all_features_df,int(set_number_of_features+np.floor((len(
If I take the data having correlation between only -.1 and .1 range then I get 0 features.
If I take the data having correlation between only -.90 and .90 range then I get 65 features.
If I take the data having correlation between only -.95 and .95 range then I get 85 features.
If I take the data having correlation between only -.97 and .97 range then I get 102 features.
If I take the data having correlation between only -.98 and .98 range then I get 112 features.
If I take the data having correlation between only -.99 and .99 range then I get 127 features.
If I take the data having correlation between only -.999 and .999 range then I get 129 features.
CPU times: user 13 s, sys: 44.8 ms, total: 13.1 s
Wall time: 13 s
```

```
In [ ]: new_standard_train_median_all_features_df=perform_spearman(new_standard_train_median_all_features_df,spearma
```

```
In [ ]: new_standard_train_median_all_features_df.head()
```

Out[303]:

	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_histogram_bin4	sens
0	-0.432133	-0.078741	-0.028788	-0.057799	-0.115367	-0.179786	
1	2.314107	0.276872	-0.028788	-0.057799	-0.115367	-0.175033	
2	-0.432133	-0.108583	-0.028788	-0.057799	-0.114391	-0.176506	
3	-0.432131	0.303606	-0.028788	-0.057799	-0.115367	-0.180203	
4	-0.432132	0.031300	-0.028788	-0.057799	-0.115367	-0.180360	

5 rows × 127 columns

```
In [ ]: new_standard_train_median_all_features_df.shape
```

Out[304]: (48000, 127)

```
In [ ]: new_standard_test_median_all_features_df=standard_test_median_all_features_df[new_standard_train_median_all_features_df.head()]
```

Out[305]:

	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_histogram_bin4	sensor7_histogram_bin5
0	-0.432133	-0.078741	-0.028788	-0.057799	-0.115367	-0.180209	-0.180209
1	2.314107	0.135746	-0.028788	-0.057799	-0.115367	-0.180150	-0.180150
2	-0.432133	-0.115422	-0.028788	-0.057799	-0.115367	-0.180461	-0.180461
3	2.314107	-0.114800	-0.028788	-0.057799	-0.115367	-0.180276	-0.180276
4	2.314107	0.041869	-0.028788	-0.057799	-0.115367	-0.180369	-0.180369

5 rows × 127 columns

```
In [ ]: new_standard_test_median_all_features_df=standard_test_median_all_features_df[new_standard_train_median_all_features_df.head()]
```

Out[306]:

	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_histogram_bin4	sensor7_histogram_bin5
0	-0.432133	-0.078741	-0.028788	-0.057799	-0.115367	-0.180209	-0.180209
1	2.314107	0.135746	-0.028788	-0.057799	-0.115367	-0.180150	-0.180150
2	-0.432133	-0.115422	-0.028788	-0.057799	-0.115367	-0.180461	-0.180461
3	2.314107	-0.114800	-0.028788	-0.057799	-0.115367	-0.180276	-0.180276
4	2.314107	0.041869	-0.028788	-0.057799	-0.115367	-0.180369	-0.180369

5 rows × 127 columns

```
In [ ]: #new_standard_train_median_all_features_df.to_pickle("new_standard_train_median_all_features_df.pickle")
#y_train_median_all_features.to_pickle("y_train_median_all_features.pickle")

#new_standard_test_median_all_features_df.to_pickle("new_standard_test_median_all_features_df.pickle")
#y_test_median_all_features.to_pickle("y_test_median_all_features.pickle")
```

```
In [23]: #new_standard_train_median_all_features_df=pd.read_pickle("/content/gdrive/My Drive/new_standard_train_median_all_features_df.pickle")
#y_train_median_all_features=pd.read_pickle("/content/gdrive/My Drive/y_train_median_all_features.pickle")

#new_standard_test_median_all_features_df=pd.read_pickle("/content/gdrive/My Drive/new_standard_test_median_all_features_df.pickle")
#y_test_median_all_features=pd.read_pickle("/content/gdrive/My Drive/y_test_median_all_features.pickle")
```

```
In [ ]: np.save('new_standard_train_median_all_features_df_columns.npy',new_standard_train_median_all_features_df.columns)
```

```
In [30]: new_standard_train_median_all_features_df["target"] = y_train_median_all_features
new_standard_train_median_all_features_df.head()
```

Out[30]:

	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_histogram_bin4	sens
0	-0.432133	-0.078741	-0.028788	-0.057799	-0.115367	-0.179786	
1	2.314107	0.276872	-0.028788	-0.057799	-0.115367	-0.175033	
2	-0.432133	-0.108583	-0.028788	-0.057799	-0.114391	-0.176506	
3	-0.432131	0.303606	-0.028788	-0.057799	-0.115367	-0.180203	
4	-0.432132	0.031300	-0.028788	-0.057799	-0.115367	-0.180360	

5 rows × 128 columns

Compare the target 0 and target 1 mean values feature wise after median value imputing

```
In [31]: column_list=[]
mean_list=[]
one_zero=[]
for column in list(new_standard_train_median_all_features_df.columns.values):
    if column!="target":
        column_list.append(column)
        one_zero.append(0)
        mean_list.append(new_standard_train_median_all_features_df[column][new_standard_train_median_all_fea
            column_list.append(column)
            one_zero.append(1)
            mean_list.append(new_standard_train_median_all_features_df[column][new_standard_train_median_all_fea
```

Find the features where the difference between the means of the two classes is more than .15
descending sorted by class 1

```
In [32]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:mean_list[i] for i in index}
```

```
In [33]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

```
In [34]: counter=0
new_column_list=[]
new_mean_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_mean_list.append(mean_list[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_mean_list.append(mean_list[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 127 sensors.

```
In [37]: try:
    plot_diagram(new_column_list,new_mean_list,new_one_zero,"Mean",1)
except ValueError:
    pass
```

Observation : Observe that there is a significant difference between the means of the surface failures and the means of the downhole failures(Descending order for the downhole failures).

```
In [38]: counter=0
mean_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.

    if (np.abs(np.abs(mean_list[i])-np.abs(mean_list[i-1]))>=.15):
        mean_higher_reading.append(mean_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

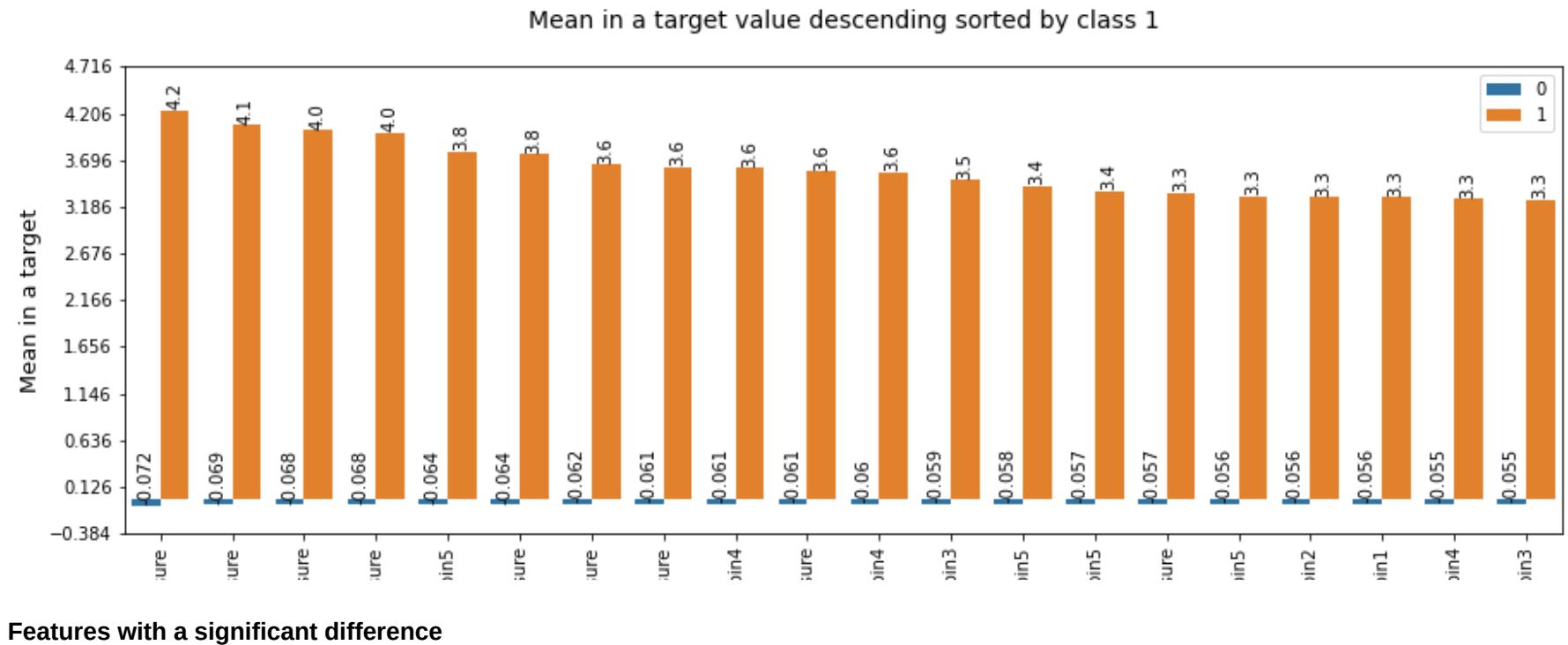
        mean_higher_reading.append(mean_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between median readings of
```

So we have approximate 122 sensor that show significant difference (> .15) between median readings of surface and downhole failures.

In [39]:

```
try:  
    plot_diagram(column_higher_reading,mean_higher_reading,one_zero_higher_reading,"Mean",1)  
except ValueError:  
    pass
```



Compare the target 0 and target 1 median values feature wise after median value imputing

```
In [40]: column_list=[]
median_list=[]
one_zero=[]

for column in list(new_standard_train_median_all_features_df.columns.values):
    if column!="target":
        column_list.append(column)
        one_zero.append(0)
        median_list.append(new_standard_train_median_all_features_df[column][new_standard_train_median_all_f
            column_list.append(column)
            one_zero.append(1)
            median_list.append(new_standard_train_median_all_features_df[column][new_standard_train_median_all_f
```

Find the features where the difference between the medians of the two classes is more than .15
descending sorted by class 1

```
In [41]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:median_list[i] for i in index}
```

```
In [42]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

```
In [43]: counter=0
new_column_list=[]
new_median_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_median_list.append(median_list[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_median_list.append(median_list[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 127 sensors.

```
In [44]: try:
    plot_diagram(new_column_list,new_median_list,new_one_zero,"Median",1)
except ValueError:
    pass
```

Observation : Observe that median failures for the downhole failures(Descending order for the downhole failures) is less than that of the downhole mean.

```
In [45]: counter=0
median_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.

    if (np.abs(np.abs(median_list[i])-np.abs(median_list[i-1]))>=.15):
        median_higher_reading.append(median_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

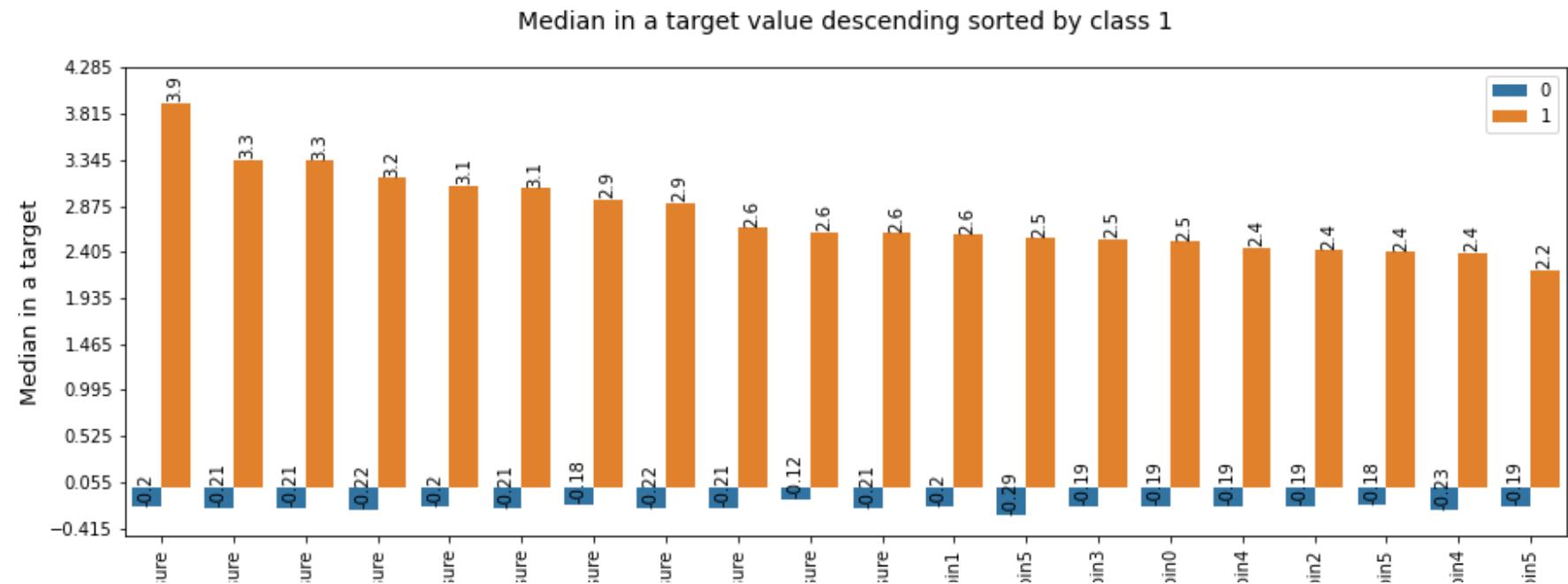
        median_higher_reading.append(median_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between median readings of surface and downhole failures.")
```

So we have approximate 70 sensor that show significant difference (> .15) between median readings of surface and downhole failures.

```
In [46]: try:
```

```
    plot_diagram(column_higher_reading,median_higher_reading,one_zero_higher_reading,"Median",1)
except ValueError:
    pass
```



Observations regarding this

- 1) All the values which were missing , we filled them with the median values.
 - 2) Higher difference for mean values but possibly the right difference for the median value.

Basic Classifier

```
In [47]: new standard train median all features df.drop("target",axis=1,inplace=True)
```

```
In [52]: print(dummy_classifier_based_on_median(new_standard_train_median_all_features_df.loc[41996],
                                              column_list,
                                              median_list))
```

Predicted surface failure.
[118, 9]

```
In [53]: print(dummy_classifier_based_on_median(new_standard_test_median_all_features_df.loc[11971],
                                              column_list,
                                              median_list))
```

Predicted surface failure.
[88, 39]

```
In [57]: new_standard_train_median_all_features_df["target"] = y_train_median_all_features
```

EDA for the dataset

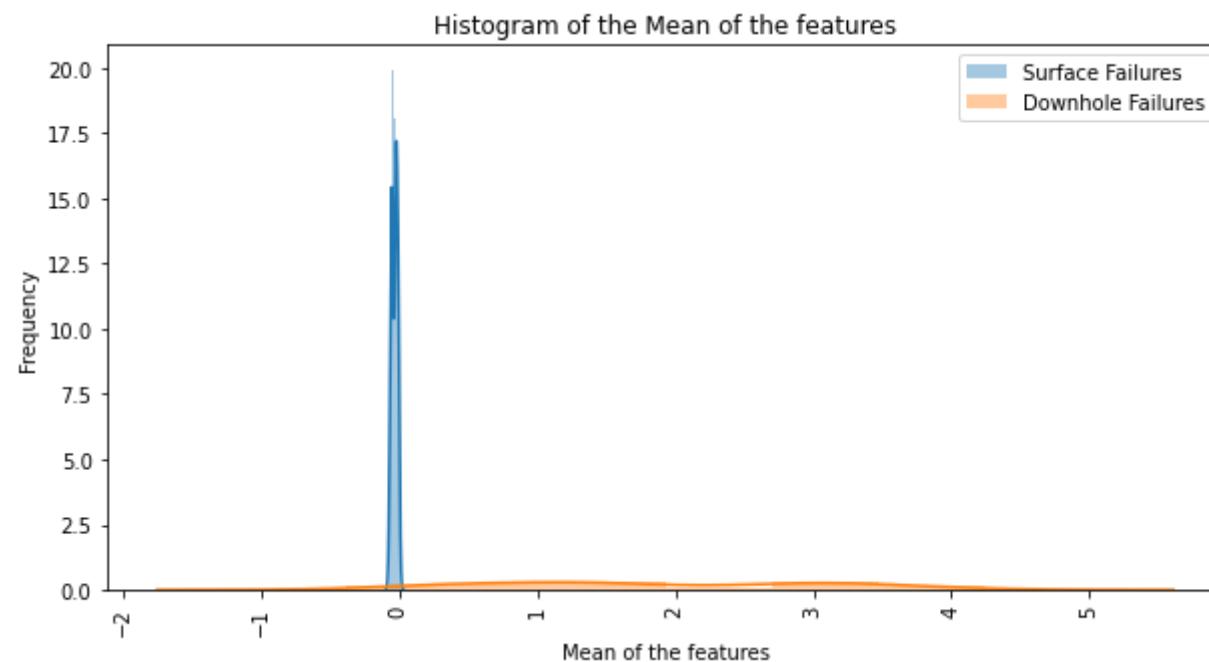
Using histograms, plots find out the nature of means for features for both the classes and thereby, the outliers.

```
In [58]: mean_0=[]
mean_1=[]
for column in list(new_standard_train_median_all_features_df.columns.values):
    if column!="target":
        mean_0.append(new_standard_train_median_all_features_df[new_standard_train_median_all_features_df.ta
        mean_1.append(new_standard_train_median_all_features_df[new_standard_train_median_all_features_df.ta
```

```
In [59]: median_0=[]
median_1=[]
for column in list(new_standard_train_median_all_features_df.columns.values):
    if column!="target":
        median_0.append(new_standard_train_median_all_features_df[new_standard_train_median_all_features_df.ta
        median_1.append(new_standard_train_median_all_features_df[new_standard_train_median_all_features_df.ta
```

10) Try and plot it's histogram for both the target values.

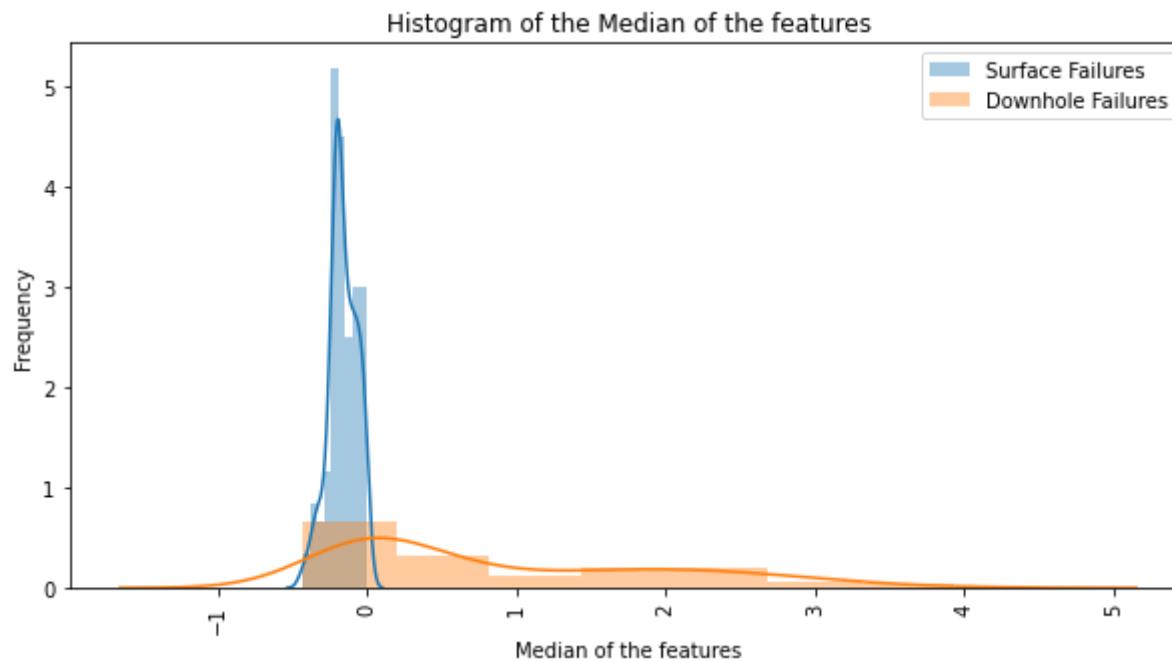
```
In [61]: import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Histogram of the Mean of the features')
plt.ylabel('Frequency')
plt.xlabel('Mean of the features')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations :

Here for the surface and downhole failures, we can see that the different downhole features have more variance. Whereas the majority of the surface related features have a very similar value.

```
In [62]: plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('Histogram of the Median of the features')
plt.ylabel('Frequency')
plt.xlabel('Median of the features')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```

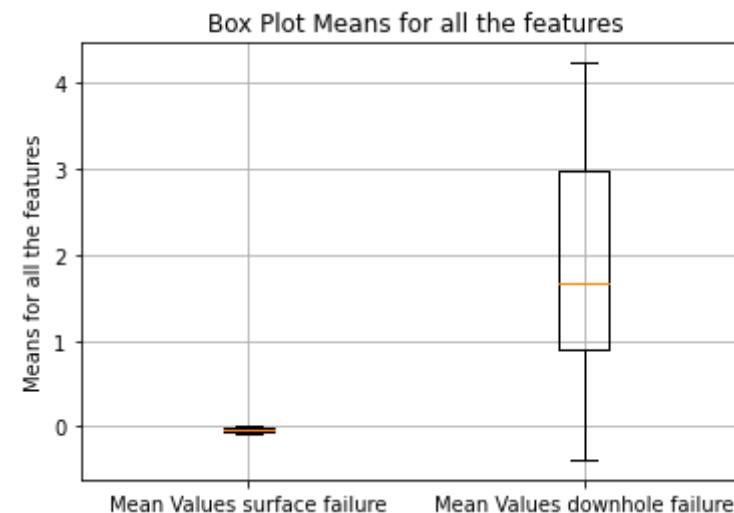


Observations :

Here for the surface and downhole failures, we can see that the different downhole features have more variance. Whereas the majority of the surface related features have a very similar value.

11) Try box plot / violin plot to visualize the outliers.

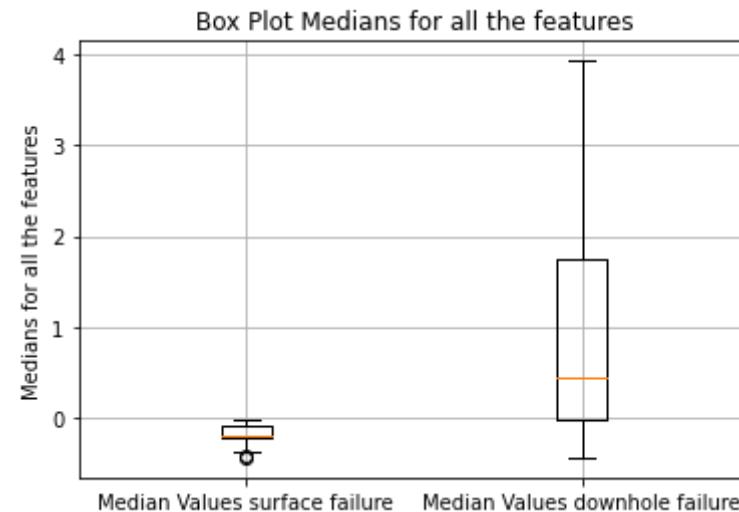
```
In [63]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Box Plot Means for all the features')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the features')  
plt.grid()  
plt.show()
```



Observations :

The majority of the values for the surface failures lie near 0 whereas for the downhole failures, we can see all kinds of variation.

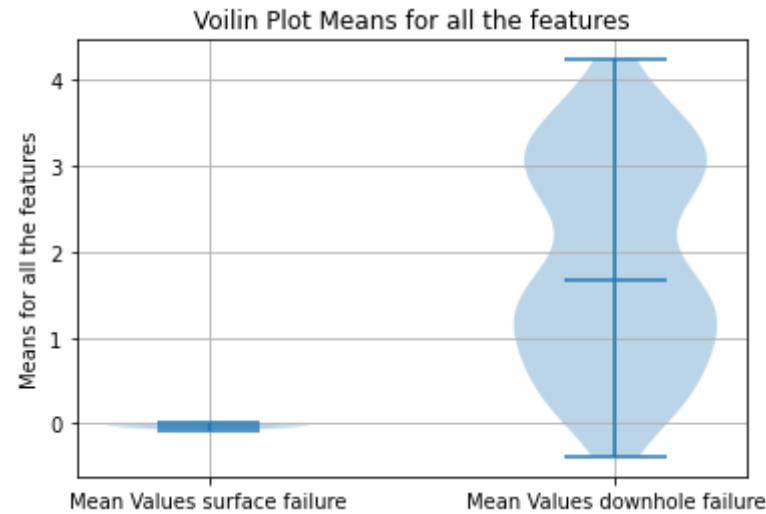
```
In [64]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('Box Plot Medians for all the features')  
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))  
plt.ylabel('Medians for all the features')  
plt.grid()  
plt.show()
```



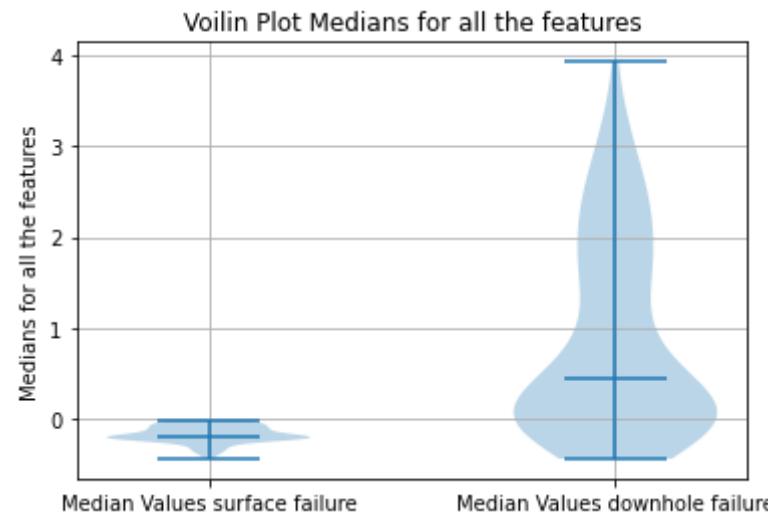
Observations :

Similar behaviour with the medians.

```
In [65]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Violin Plot Means for all the features')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the features')
plt.grid()
plt.show()
```



```
In [66]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('Violin Plot Medians for all the features')
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))
plt.ylabel('Medians for all the features')
plt.grid()
plt.show()
```



PCA

```
In [67]: new_standard_train_median_all_features_df.drop("target",axis=1,inplace=True)
```

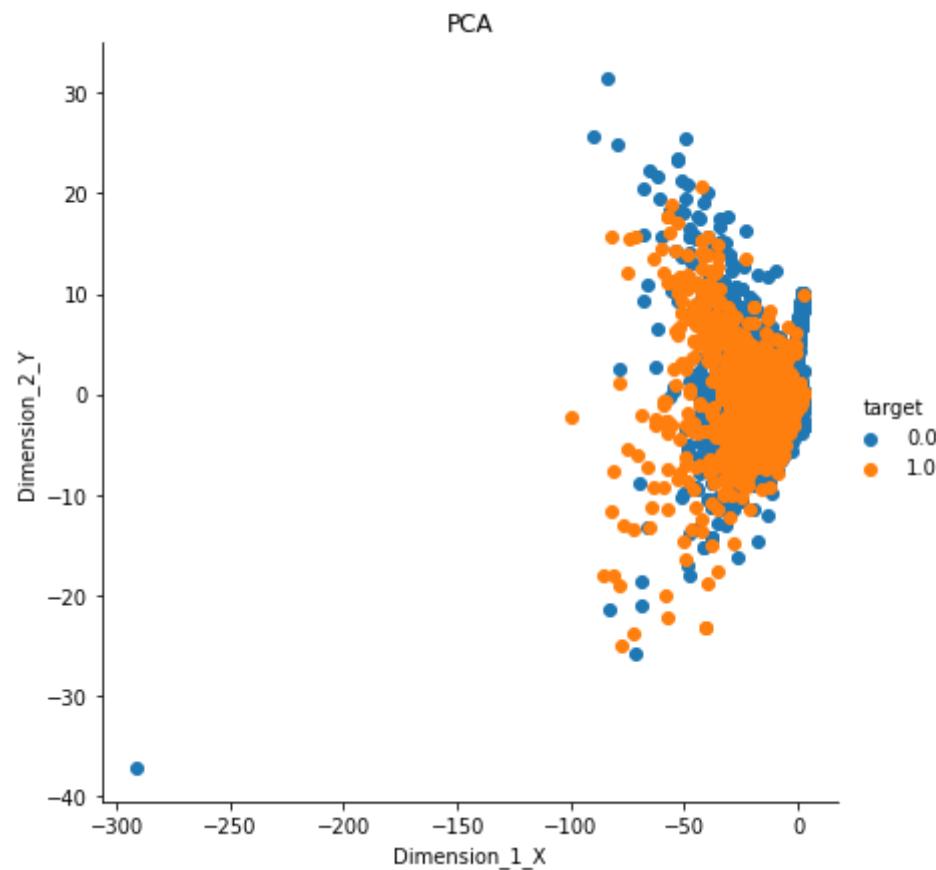
```
In [68]: "target" in new_standard_train_median_all_features_df.columns
```

```
Out[68]: False
```

In [71]:

```
%%time
run_pca(new_standard_train_median_all_features_df,y_train_median_all_features)
```

```
[[ 1.3551275 -2.7165384  0.        ]
 [ -2.426687   -0.21472278  0.        ]
 [  2.8133297  -0.507791   0.        ]
 ...
 [-46.629208  -3.5711164  1.        ]
 [ -3.1961012  -0.28361306  1.        ]
 [-63.428383  13.535967   1.        ]]
```



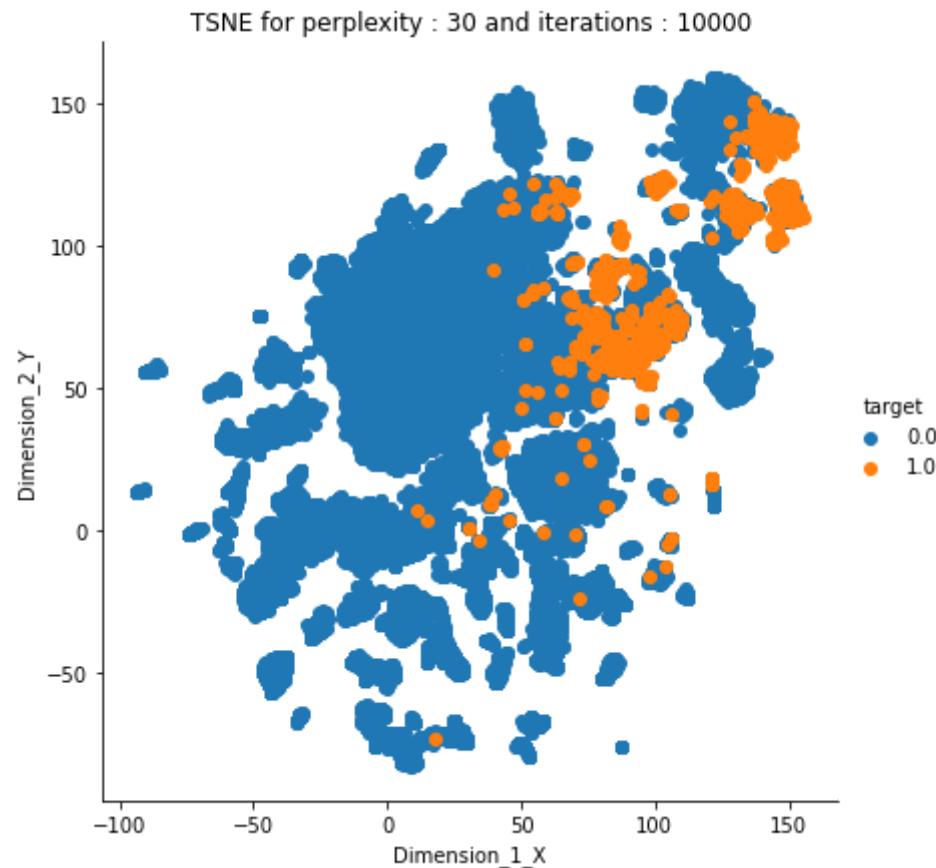
```
CPU times: user 630 ms, sys: 109 ms, total: 739 ms
Wall time: 619 ms
```

tsne

Do not put these values in a loop as we do not get all the output graphs

In []:

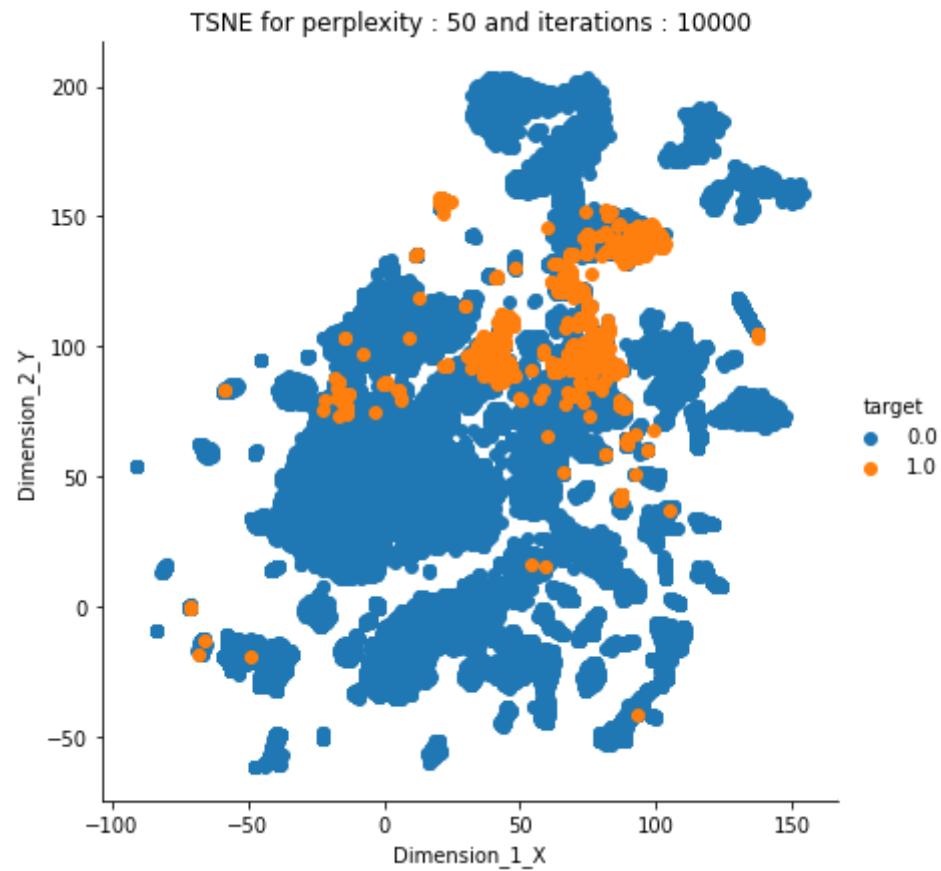
```
%%time  
run_tsne(new_standard_train_new_median_all_features_df,y_train_median_all_features,30,10000)
```



CPU times: user 13 s, sys: 11.6 s, total: 24.6 s
Wall time: 24.5 s

In []:

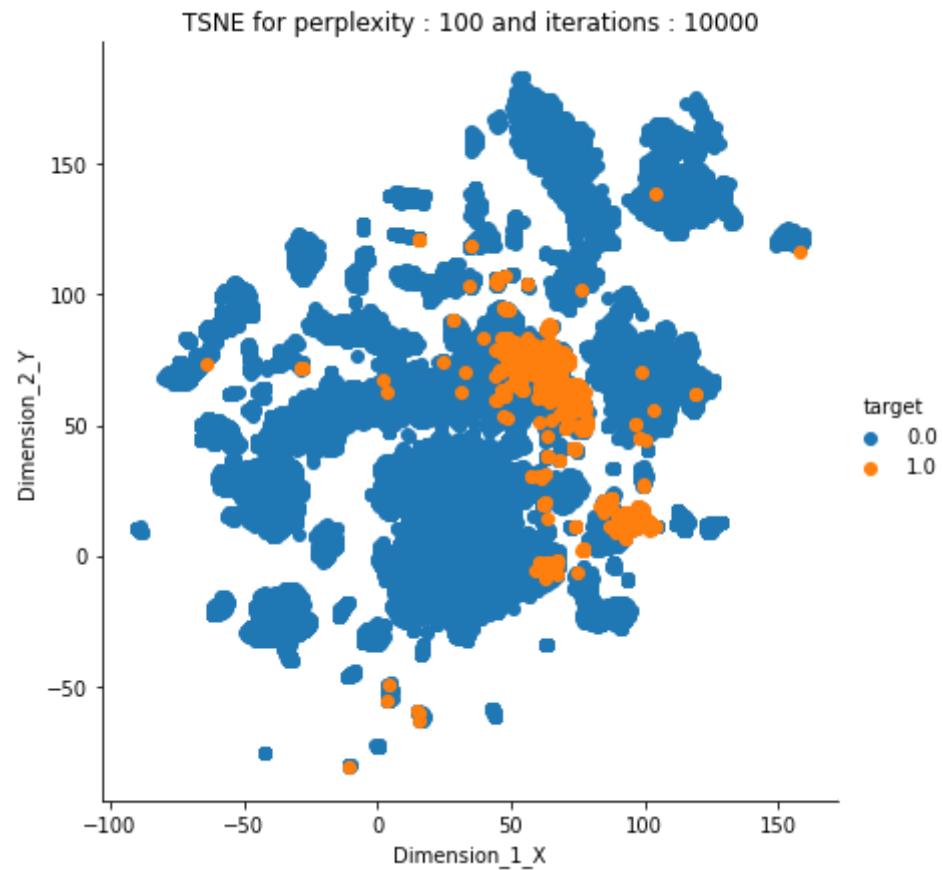
```
%%time  
run_tsne(new_standard_train_new_median_all_features_df,y_train_median_all_features,50,10000)
```



CPU times: user 12.9 s, sys: 11.6 s, total: 24.5 s
Wall time: 24.3 s

In []:

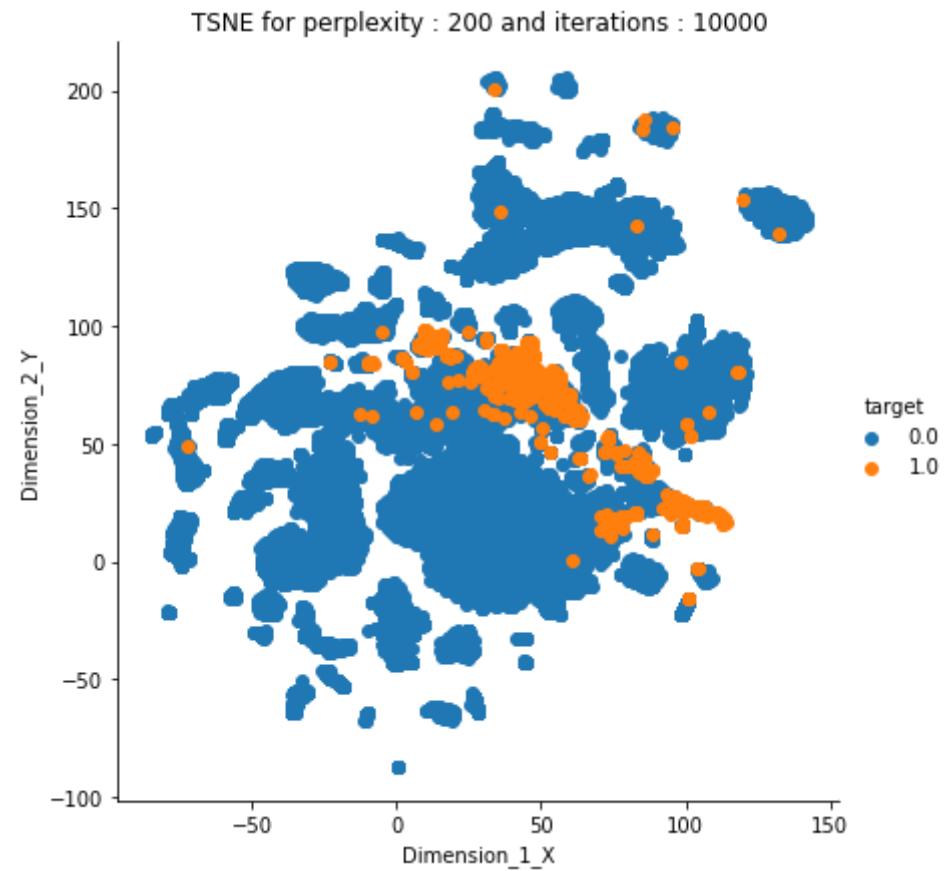
```
%%time  
run_tsne(new_standard_train_new_median_all_features_df,y_train_median_all_features,100,10000)
```



CPU times: user 12.7 s, sys: 11.4 s, total: 24.1 s
Wall time: 23.9 s

In []:

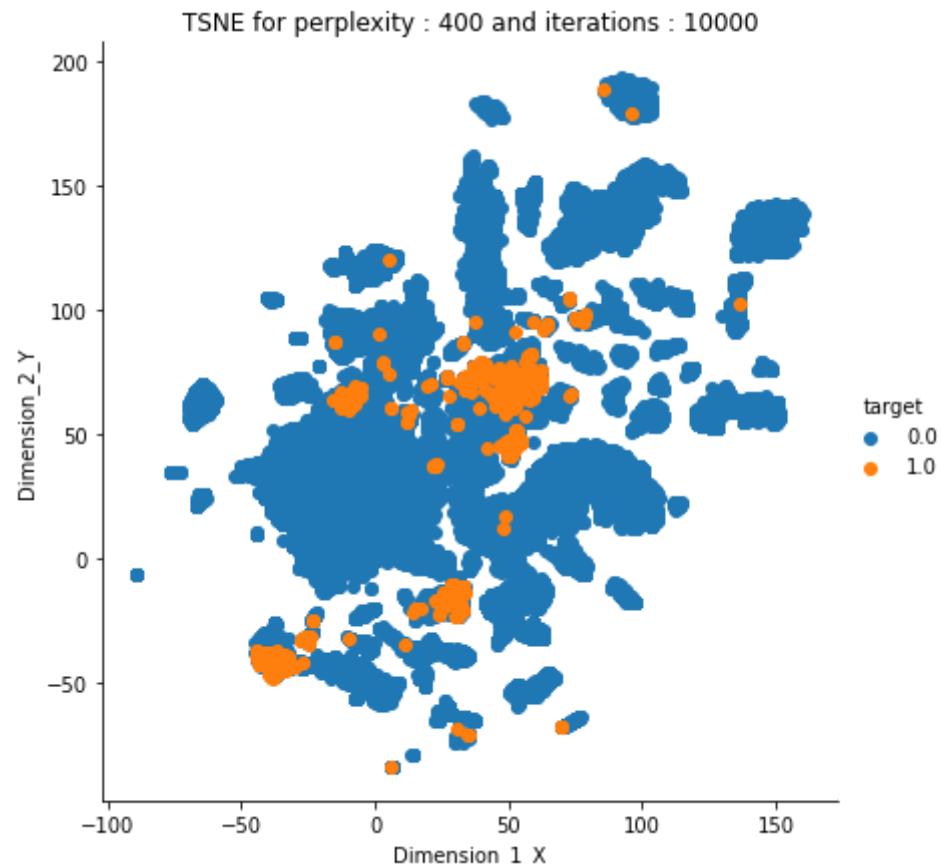
```
%%time  
run_tsne(new_standard_train_new_median_all_features_df,y_train_median_all_features,200,10000)
```



CPU times: user 13 s, sys: 11.6 s, total: 24.6 s
Wall time: 24.4 s

In []:

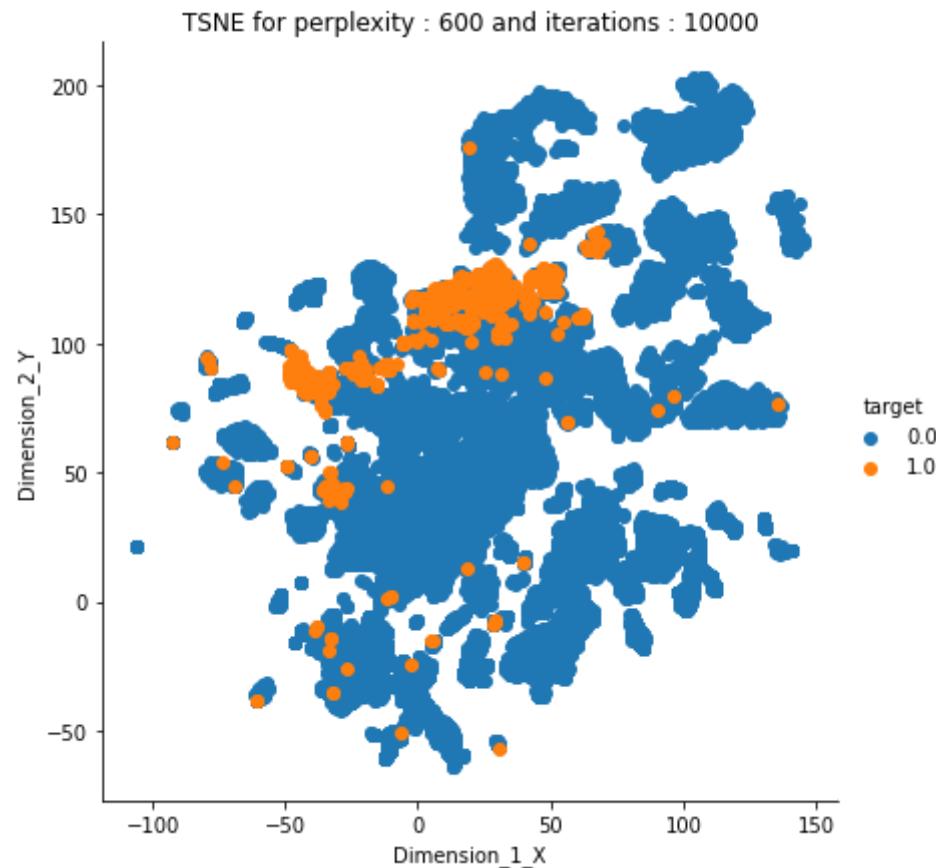
```
%%time  
run_tsne(new_standard_train_new_median_all_features_df,y_train_median_all_features,400,10000)
```



CPU times: user 13.1 s, sys: 11.9 s, total: 24.9 s
Wall time: 24.8 s

In []:

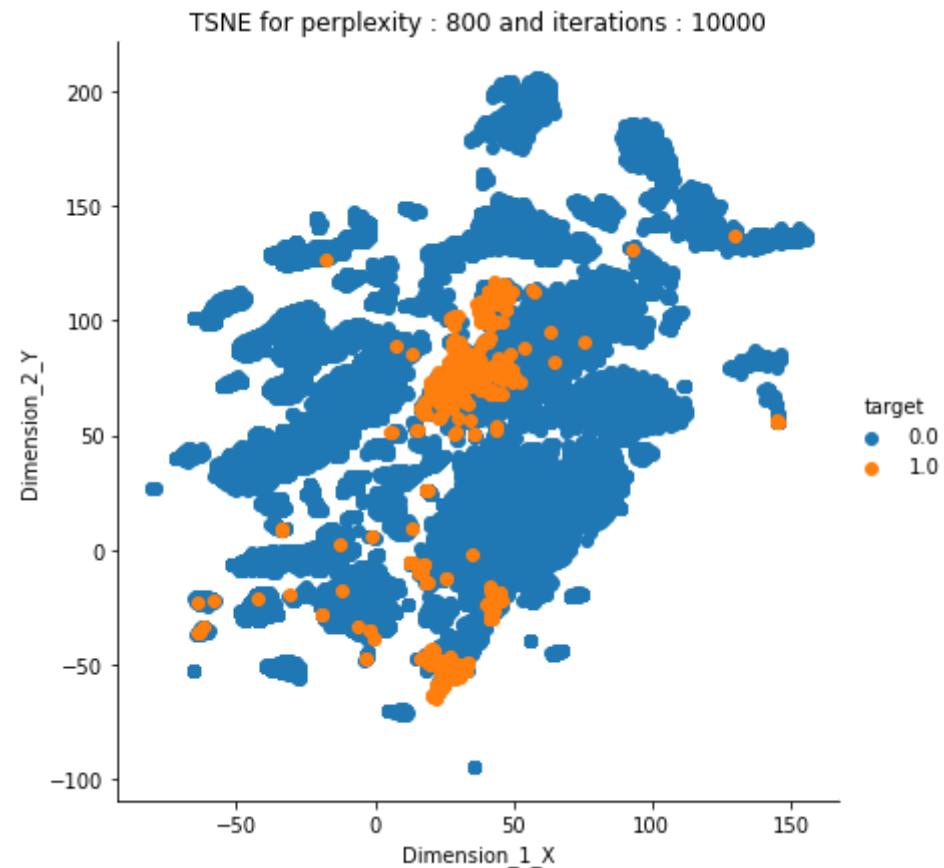
```
%%time  
run_tsne(new_standard_train_new_median_all_features_df,y_train_median_all_features,600,10000)
```



CPU times: user 12.7 s, sys: 11.4 s, total: 24.1 s
Wall time: 24 s

In []:

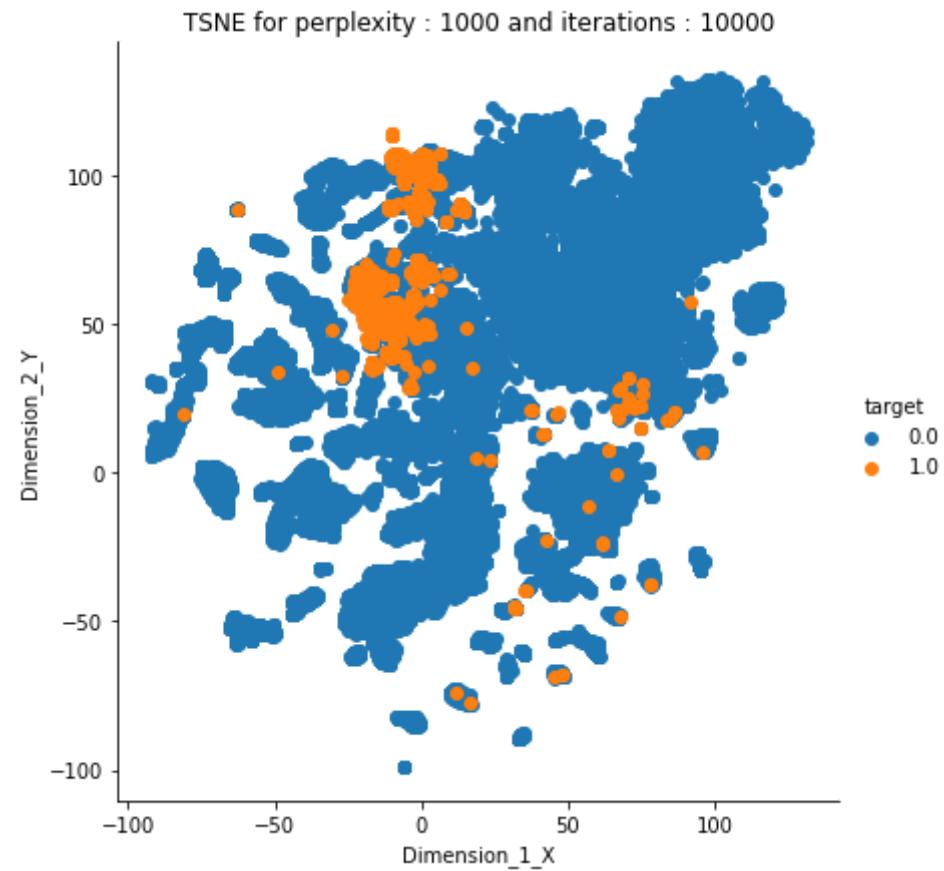
```
%%time  
run_tsne(new_standard_train_new_median_all_features_df,y_train_median_all_features,800,10000)
```



CPU times: user 12.9 s, sys: 11.6 s, total: 24.5 s
Wall time: 24.3 s

In []:

```
%%time  
run_tsne(new_standard_train_new_median_all_features_df,y_train_median_all_features,1000,10000)
```



CPU times: user 12.9 s, sys: 11.7 s, total: 24.5 s
Wall time: 24.4 s

Observation :

- 1) Observe that different perplexities tSNE is able to differentiate all the data points into different groups, but it is not able to differentiate properly the surface failures from the downhole failures. If the surface and downhole failures are coming under the same category then they are shown in the same cluster.
- 2) Not all the the surface and downhole features are coming under the same category.

Perform a similar EDA on the outliers.

Here we take the outliers from the features that have been selected.

For each sensor we find the outliers that also classwise.

The issue is that there maybe high outliers here , but there maybe lower outliers here as well.

<https://www.geeksforgeeks.org/interquartile-range-and-quartile-deviation-using-numpy-and-scipy/> (<https://www.geeksforgeeks.org/interquartile-range-and-quartile-deviation-using-numpy-and-scipy/>)

Higher outlier = more than $1.5 \cdot \text{IQR}$ above the third quartile.

Lower outlier = less than $1.5 \cdot \text{IQR}$ below the first quartile.

The IQR is the difference between Q3 and Q1.

```
In [ ]: new_standard_train_median_all_features_df["target"] = y_train_median_all_features
```

```
In [ ]: counter=0
main_dictionary=dict()
outliers=[]
list_of_outliers_features_classes=[]
for i in range(len(column_list)):
    d=dict()
    new_list=new_standard_train_median_all_features_df[new_standard_train_median_all_features_df.target==one
[column_list[i]]]
    new_index=list(new_list.reset_index()["index"])
    QR1=np.percentile(new_list, 25)
    QR3=np.percentile(new_list, 75)
    IQR=QR3-QR1
    for index,value in zip(new_index,new_list):
        if (value>(QR3+(IQR*1.5))) or (value<(QR1-(IQR*1.5))):
            counter=counter+1
            d.update({index:value})
            if index not in main_dictionary:
                main_dictionary.update({index:list(new_standard_train_median_all_features_df.loc[index])})
    list_of_outliers_features_classes.append(d)
del d
outliers.append(counter)
counter=0
```

```
In [ ]: print("Total number of data points that are outliers in atleast one or more than one feature : ",len(np.unique
```

```
Total number of data points that are outliers in atleast one or more than one feature : 44635
```

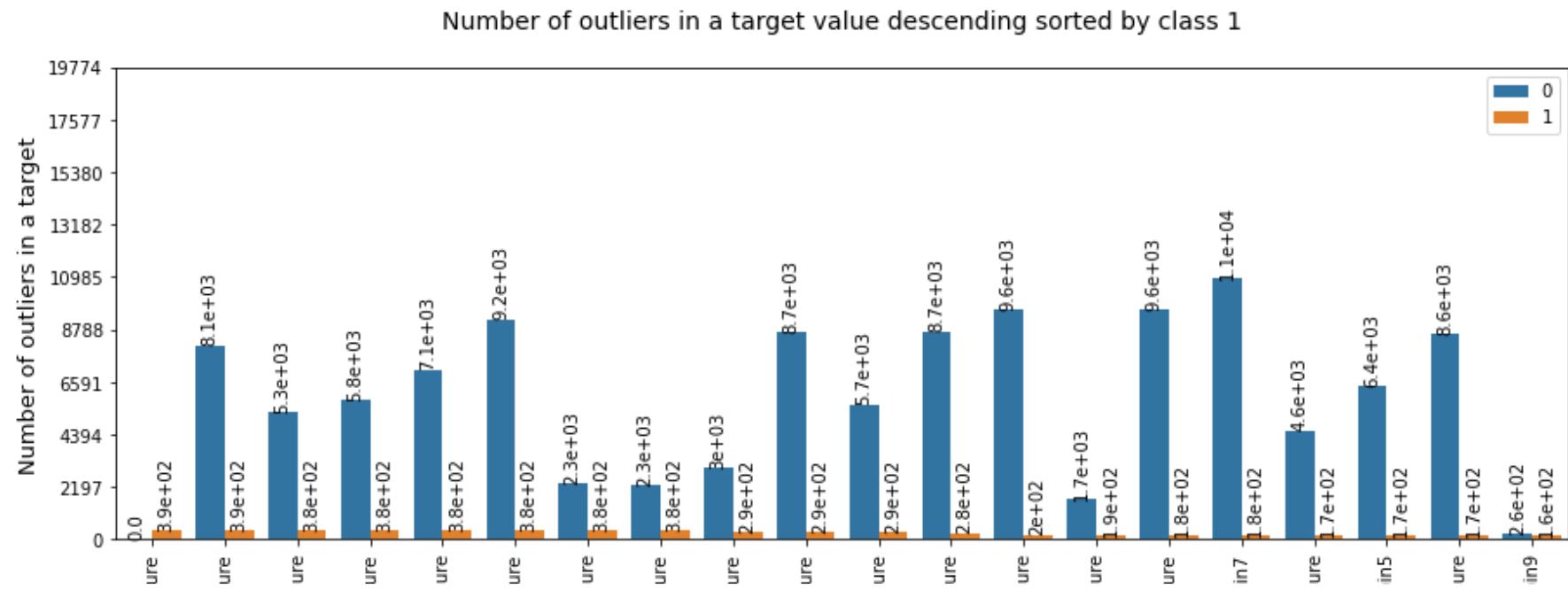
```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:outliers[i] for i in index}

d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])

counter=0
new_column_list=[]
new_outliers_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_outliers_list.append(outliers[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_outliers_list.append(outliers[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 127 sensors.

```
In [ ]: try:  
    plot_diagram(new_column_list,new_outliers_list,new_one_zero,"Number of outliers",1)  
except ValueError:  
    pass
```



Observation :

Number of outliers for surface failures seem to be much more than the number of outliers for the downhole failures.

For every column feature, get the mean for the outliers for both class 0 and class 1.

```
In [ ]: mean_0=[]
mean_1=[]
for i in range(0,len(list_of_outliers_features_classes),2):
    class_0=[ele for ele in list_of_outliers_features_classes[i].values()]
    class_1=[ele for ele in list_of_outliers_features_classes[i+1].values()]
    if (len(class_0)==0) :
        mean_0.append(0.01)
    else :
        mean_0.append(np.mean(class_0))

    if (len(class_1)==0):
        mean_1.append(0.01)
    else :
        mean_1.append(np.mean(class_1))
```

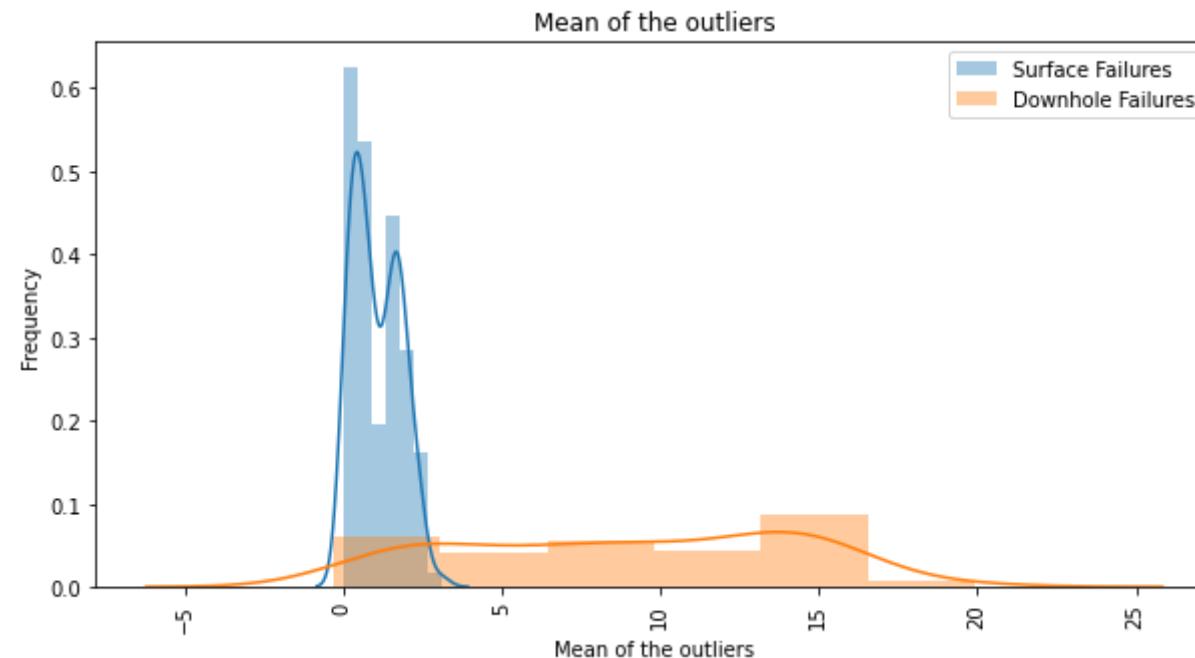
```
In [ ]: median_0=[]
median_1=[]
for i in range(0,len(list_of_outliers_features_classes),2):
    class_0=[ele for ele in list_of_outliers_features_classes[i].values()]
    class_1=[ele for ele in list_of_outliers_features_classes[i+1].values()]
    if (len(class_0)==0) :
        median_0.append(0.01)
    else :
        median_0.append(np.median(class_0))

    if (len(class_1)==0):
        median_1.append(0.01)
    else :
        median_1.append(np.median(class_1))
```

Plotting the mean and median for each feature class wise.

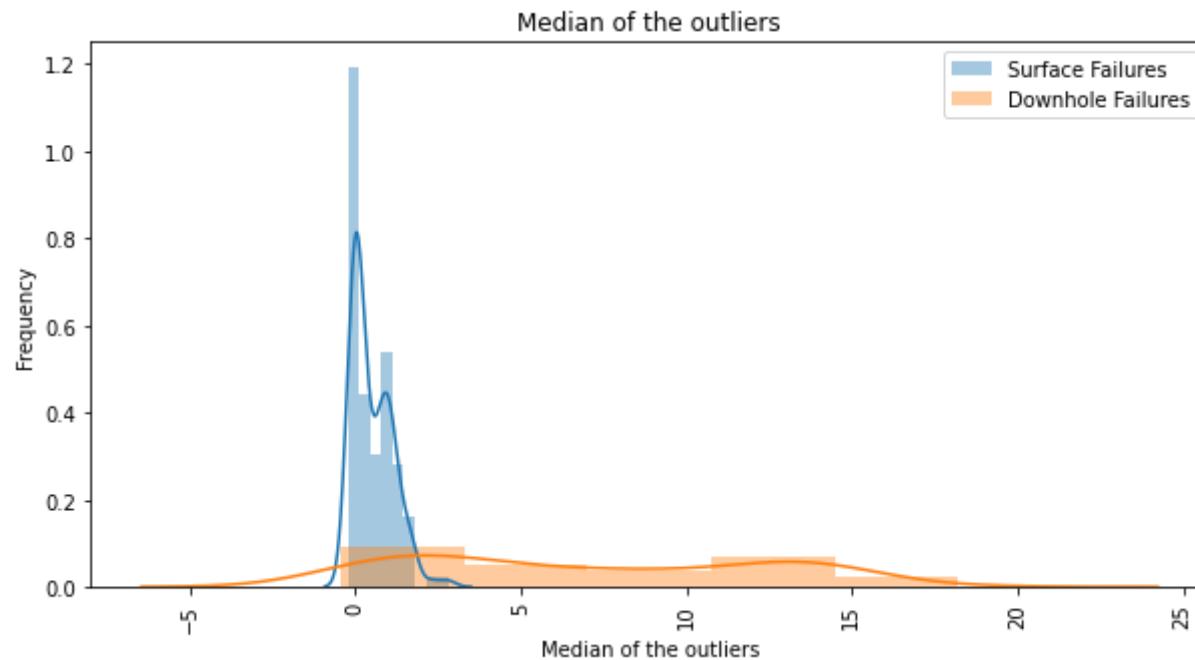
10) Try and plot it's histogram for both the target values.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Mean of the outliers')
plt.ylabel('Frequency')
plt.xlabel('Mean of the outliers')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations : The distribution of the outlier is similar to the distribution of the features.
Downhole outliers have more variance.
Whereas the majority of the surface outliers are all concentrated near the same value 0.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('Median of the outliers')
plt.ylabel('Frequency')
plt.xlabel('Median of the outliers')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations : The distribution of the outlier is similar to the distribution of the features.

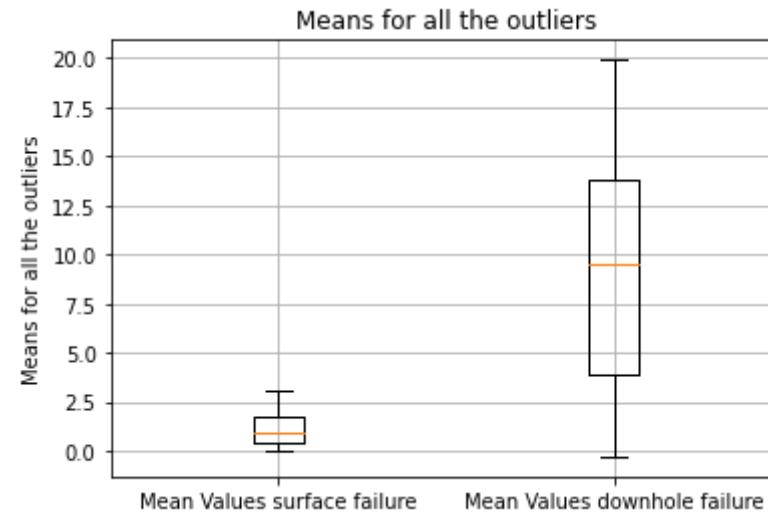
Downhole outliers have more variance.

Whereas the majority of the surface outliers are all concentrated near the same value 0.

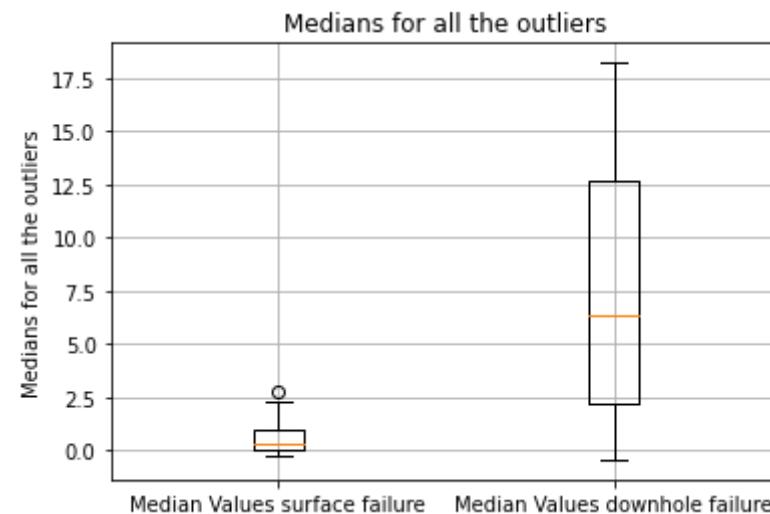
Similar behaviour in box plot and violin plot.

11) Try box plot / violin plot to visualize the outliers.

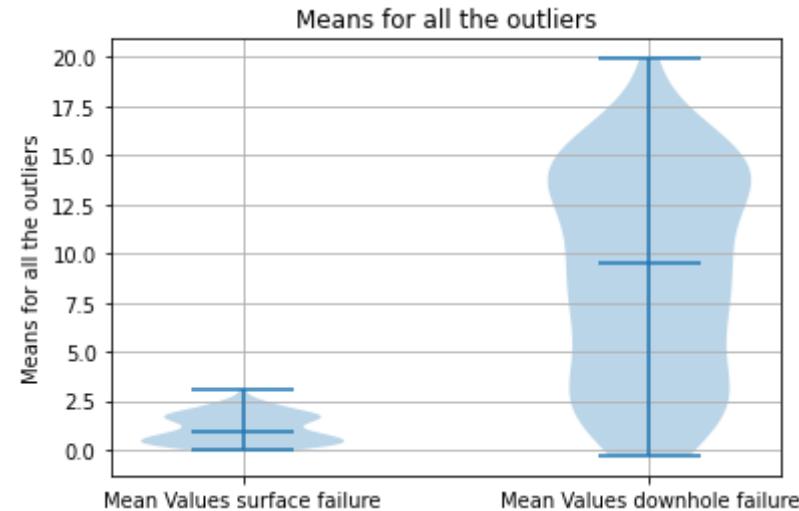
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Means for all the outliers')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the outliers')  
plt.grid()  
plt.show()
```



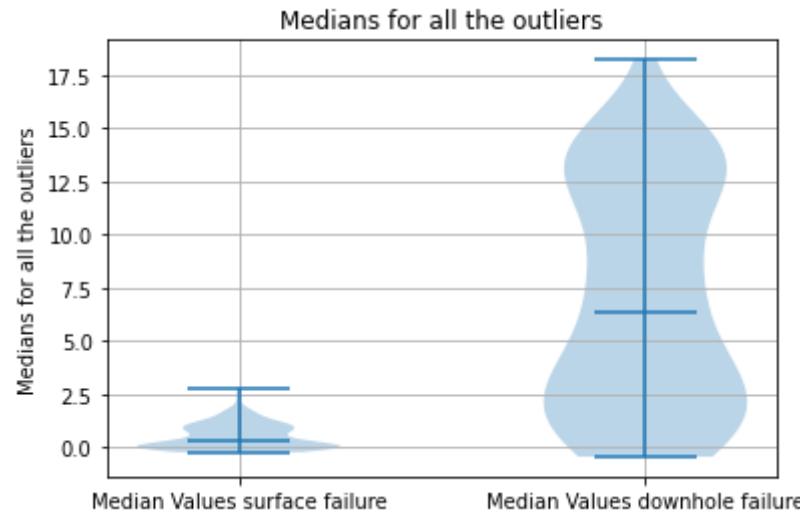
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('Medians for all the outliers')  
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))  
plt.ylabel('Medians for all the outliers')  
plt.grid()  
plt.show()
```



```
In [ ]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Means for all the outliers')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the outliers')
plt.grid()
plt.show()
```



```
In [ ]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('Medians for all the outliers')
plt.xticks([1,2],('Median Values surface failure','Median Values downhole failure'))
plt.ylabel('Medians for all the outliers')
plt.grid()
plt.show()
```



All the outliers are very seem to be very similar.

For the given data points and features enter new features that tell if these values were previously np.NaN or not.

```
In [ ]: new_standard_train_median_all_features_df.drop("target",axis=1,inplace=True)
```

```
In [ ]: nan_standard_train_median_all_features_df=fuse_dataframe_nan(new_standard_train_median_all_features_df,nan_o  
nan_standard_train_median_all_features_df[ "target" ]=y_train_median_all_features  
nan_standard_train_median_all_features_df.head()
```

Out[356]:

	sensor7_histogram_bin2	sensor17_measure	sensor105_histogram_bin3	sensor69_histogram_bin7	sensor64_histogram_bin2	sensor40_measure
0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	1.0

5 rows × 128 columns

Now this is the problem out of the given data that we have , we select the features that have more than 50% np.NaNs

```
In [ ]: counter=0
for column in new_standard_train_median_all_features_df.columns.values:
    try :
        percent_of_nan=nan_standard_train_median_all_features_df[column].value_counts()[1]/\
                    (nan_standard_train_median_all_features_df[column].value_counts()[0]+\
                     nan_standard_train_median_all_features_df[column].value_counts()[1])
        if (percent_of_nan>.5) :
            counter=counter+1
            print(column," : ",percent_of_nan)
    except :
        pass
print("Overall : ",counter," features with % of na's more than 50%")
```

```
sensor38_measure      :  0.6605416666666667
sensor39_measure      :  0.7348958333333333
sensor40_measure      :  0.77375
sensor41_measure      :  0.7967916666666667
sensor42_measure      :  0.8133125
sensor43_measure      :  0.822
Overall :  6  features with % of na's more than 50%
```

4) Impute np.NaN with median and then perform Adasyn on the values.

```
In [10]: data_type="adasyn_data"
```

```
In [ ]: train_adasyn_all_features=train.copy()
y_train_adasyn_all_features=y_train.copy()
test_adasyn_all_features=test.copy()
y_test_adasyn_all_features=y_test.copy()
```

```
In [ ]: train_adasyn_all_features.shape
```

```
Out[362]: (48000, 170)
```

```
In [ ]: y_train_adasyn_all_features.shape
```

```
Out[363]: (48000,)
```

```
In [ ]: test_adasyn_all_features.shape
```

```
Out[364]: (12000, 170)
```

```
In [ ]: y_test_adasyn_all_features.shape
```

```
Out[365]: (12000,)
```

```
In [ ]: for i,j in zip(train_adasyn_all_features.columns.values,test_adasyn_all_features.columns.values):
         if (i!=j):
             print(i," : ",j)
```

```
In [ ]: train_adasyn_all_features,y_train_adasyn_all_features=reset_index(train_adasyn_all_features,y_train_adasyn_all_features)
test_adasyn_all_features,y_test_adasyn_all_features=reset_index(test_adasyn_all_features,y_test_adasyn_all_features)
```

Filling the np.NaN values with the median of the category this time.

```
In [ ]: l=[]
         for ele in train_adasyn_all_features["sensor4_measure"]:
             if np.isnan(ele):
                 pass
             else:
                 l.append(ele)
np.median(l)
```

```
Out[369]: 126.0
```

```
In [ ]: train_adasyn_all_features["sensor4_measure"].median()
```

```
Out[370]: 126.0
```

Both are the same so the pandas.series.median() function ignores the value that has the np.NaN and does not even count it.

Divide the whole dataframe into the number of categories that we have and then only enter the median for that feature for that particular category.

```
In [ ]: train_adasyn_all_features,y_train_adasyn_all_features,test_adasyn_all_features,y_test_adasyn_all_features=\
impute_median_for_train_n_test_as_per_train(train_adasyn_all_features,
                                              y_train_adasyn_all_features,
                                              test_adasyn_all_features,
                                              y_test_adasyn_all_features,data_type)
```

```
In [ ]: train_adasyn_all_features,y_train_adasyn_all_features=reset_index(train_adasyn_all_features,y_train_adasyn_a
test_adasyn_all_features,y_test_adasyn_all_features=reset_index(test_adasyn_all_features,y_test_adasyn_all_f
```

Standardise the dataframe

```
In [ ]: standard_train_adasyn_all_features_df,standard_test_adasyn_all_features_df=standardize_test_n_train_wrt_trai
```

```
In [ ]: standard_train_adasyn_all_features_df.shape
```

```
Out[377]: (48000, 170)
```

```
In [ ]: standard_test_adasyn_all_features_df.shape
```

```
Out[378]: (12000, 170)
```

```
In [ ]: y_train_adasyn_all_features.head()
```

```
Out[379]: 0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: target, dtype: float32
```

```
In [ ]: y_train_adasyn_all_features.tail()
```

```
Out[380]: 47995    1.0
47996    1.0
47997    1.0
47998    1.0
47999    1.0
Name: target, dtype: float32
```

```
In [ ]: "target" in standard_train_adasyn_all_features_df.columns
```

```
Out[381]: False
```

Perform Adasyn

```
In [ ]: from collections import Counter
from imblearn.over_sampling import ADASYN

ada = ADASYN(random_state=42, sampling_strategy=.75, n_jobs=-1)
x_adasyn, y_adasyn = ada.fit_resample(standard_train_adasyn_all_features_df, y_train_adasyn_all_features)
print("done")
```

done

```
In [ ]: x_adasyn.shape
```

```
Out[385]: (82593, 170)
```

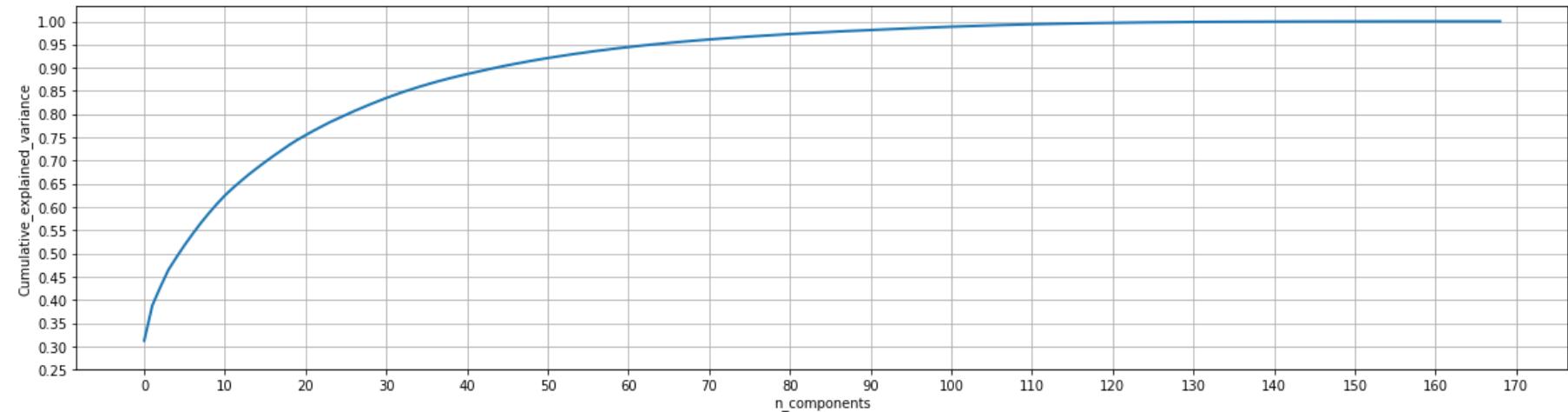
```
In [ ]: y_adasyn.shape
```

```
Out[386]: (82593, )
```

Truncated SVD to get the best features

```
In [ ]: #https://stackoverflow.com/questions/50796024/feature-variable-importance-after-a-pca-analysis
```

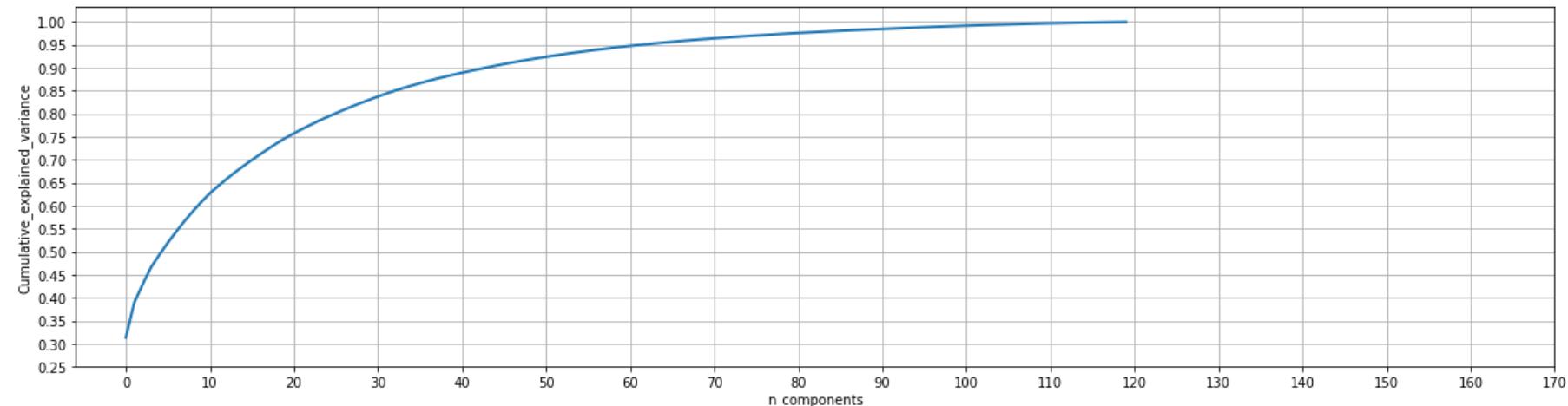
In []: `truncated_svd(x_adasyn, 169)`



Out[388]: TruncatedSVD(n_components=169)

I choose 120 components to describe my data as 120 components explain atleast 99% variance of the total data.

In []: `svd=truncated_svd(x_adasyn, 120)`



For the 120 principal components using svd.explained_variance_ratio_ find the variance distribution.

```
In [ ]: print(len(svd.explained_variance_ratio_))
```

```
120
```

```
In [ ]: np.sum(svd.explained_variance_ratio_)
```

```
Out[391]: 0.9965708
```

```
In [ ]: svd.components_.shape
```

```
Out[392]: (120, 170)
```

```
In [ ]: #number of features
```

```
svd.components_.shape[1]
```

```
Out[393]: 170
```

```
In [ ]: #number of components
```

```
svd.components_.shape[0]
```

```
Out[394]: 120
```

For the 120 principal components find the feature importance.

```
In [ ]: set_number_of_features=120
```

Trying Forward Feature selection

Here, in forward feature selection, out of 170 features even if I choose to select 10 features only , still it will take a lot of time.

Therefore I go with recursive feature elimination.

Using Recursive Feature Elimination, I can get the features that I need.

In []: x_adasyn.shape

Out[396]: (82593, 170)

```
In [ ]: new_x_adasyn_df.head()
```

Out[398]:

	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor
0	-0.209185	-0.09191	-0.432133	-0.078741	-0.057533	-0.056429		-0.010456
1	0.226385	-0.09191	2.314107	0.276872	-0.057533	-0.056429		-0.010456
2	-0.411601	-0.09191	-0.432133	-0.108583	-0.057533	-0.056429		-0.010456
3	0.027280	-0.09191	-0.432131	0.303606	-0.057533	-0.056429		-0.010456
4	-0.132515	-0.09191	-0.432132	0.031300	-0.057533	-0.056429		-0.010456

5 rows × 170 columns

Try spearman corelation coefficient to find out co relation. (It is good for cause and affect analysis but it might not be of much use when we find that our features are dependent on other features, we expect our features to be independent of each other.).

```
In [ ]: %%time
spearman=select_spearman(new_x_adasyn_df,int(set_number_of_features+np.floor((len(train.columns.values)-set_
If I take the data having correlation between only -.1 and .1 range then I get 1 features.
If I take the data having correlation between only -.90 and .90 range then I get 80 features.
If I take the data having correlation between only -.95 and .95 range then I get 100 features.
If I take the data having correlation between only -.97 and .97 range then I get 114 features.
If I take the data having correlation between only -.98 and .98 range then I get 118 features.
If I take the data having correlation between only -.99 and .99 range then I get 144 features.
If I take the data having correlation between only -.999 and .999 range then I get 164 features.
CPU times: user 30.2 s, sys: 126 ms, total: 30.3 s
Wall time: 30.2 s
```

```
In [ ]: new_x_adasyn_df=perform_spearman(new_x_adasyn_df,spearman,.99)
```

```
In [ ]: new_x_adasyn_df.head()
```

Out[401]:

	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1
0	-0.09191	-0.432133	-0.078741	-0.057533	-0.056429	-0.010456	-0.028788
1	-0.09191	2.314107	0.276872	-0.057533	-0.056429	-0.010456	-0.028788
2	-0.09191	-0.432133	-0.108583	-0.057533	-0.056429	-0.010456	-0.028788
3	-0.09191	-0.432131	0.303606	-0.057533	-0.056429	-0.010456	-0.028788
4	-0.09191	-0.432132	0.031300	-0.057533	-0.056429	-0.010456	-0.028788

5 rows × 144 columns

```
In [ ]: new_x_adasyn_df.shape
```

Out[402]: (82593, 144)

```
In [ ]: y_adasyn.shape
```

Out[403]: (82593,)

```
In [ ]: new_standard_test_adasyn_all_features_df=standard_test_adasyn_all_features_df[new_x_adasyn_df.columns.values]
new_standard_test_adasyn_all_features_df.head()
```

Out[405]:

	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1
0	-0.09191	-0.432133	-0.078741	-0.057533	-0.056429	-0.010456	-0.028788
1	-0.09191	2.314107	0.135746	-0.057533	-0.056429	-0.010456	-0.028788
2	-0.09191	-0.432133	-0.115422	-0.057533	-0.056429	-0.010456	-0.028788
3	-0.09191	2.314107	-0.114800	-0.057533	-0.056429	-0.010456	-0.028788
4	-0.09191	2.314107	0.041869	-0.057533	-0.056429	-0.010456	-0.028788

5 rows × 144 columns

```
In [ ]: #new_x_adasyn_df.to_pickle("new_x_adasyn_df.pickle")
#y_adasyn.to_pickle("y_adasyn.pickle")
```

```
#new_standard_test_adasyn_all_features_df.to_pickle("new_standard_test_adasyn_all_features_df.pickle")
#y_test_adasyn_all_features.to_pickle("y_test_adasyn_all_features.pickle")
```

```
In [73]: #new_x_adasyn_df=pd.read_pickle("/content/gdrive/My Drive/new_x_adasyn_df.pickle")
#y_adasyn=pd.read_pickle("/content/gdrive/My Drive/y_adasyn.pickle")
```

```
#new_standard_test_adasyn_all_features_df=pd.read_pickle("/content/gdrive/My Drive/new_standard_test_adasyn_
#y_test_adasyn_all_features=pd.read_pickle("/content/gdrive/My Drive/y_test_adasyn_all_features.pickle")
```

```
In [ ]: np.save('new_x_adasyn_df_columns.npy',new_x_adasyn_df.columns.values)
```

```
In [74]: new_x_adasyn_df["target"] = y_adasyn
new_x_adasyn_df.head()
```

Out[74]:

	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1
0	-0.09191	-0.432133	-0.078741	-0.057533	-0.056429	-0.010456	-0.028788
1	-0.09191	2.314107	0.276872	-0.057533	-0.056429	-0.010456	-0.028788
2	-0.09191	-0.432133	-0.108583	-0.057533	-0.056429	-0.010456	-0.028788
3	-0.09191	-0.432131	0.303606	-0.057533	-0.056429	-0.010456	-0.028788
4	-0.09191	-0.432132	0.031300	-0.057533	-0.056429	-0.010456	-0.028788

5 rows × 145 columns

Compare the target 0 and target 1 mean values feature wise after median value imputing

```
In [ ]: column_list = []
mean_list = []
one_zero = []
for column in list(new_x_adasyn_df.columns.values):
    if column != "target":
        column_list.append(column)
        one_zero.append(0)
        mean_list.append(new_x_adasyn_df[column][new_x_adasyn_df.target==0].mean())

        column_list.append(column)
        one_zero.append(1)
        mean_list.append(new_x_adasyn_df[column][new_x_adasyn_df.target==1].mean())
```

Find the features where the difference between the means of the two classes is more than .15
descending sorted by class 1

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:mean_list[i] for i in index}
```

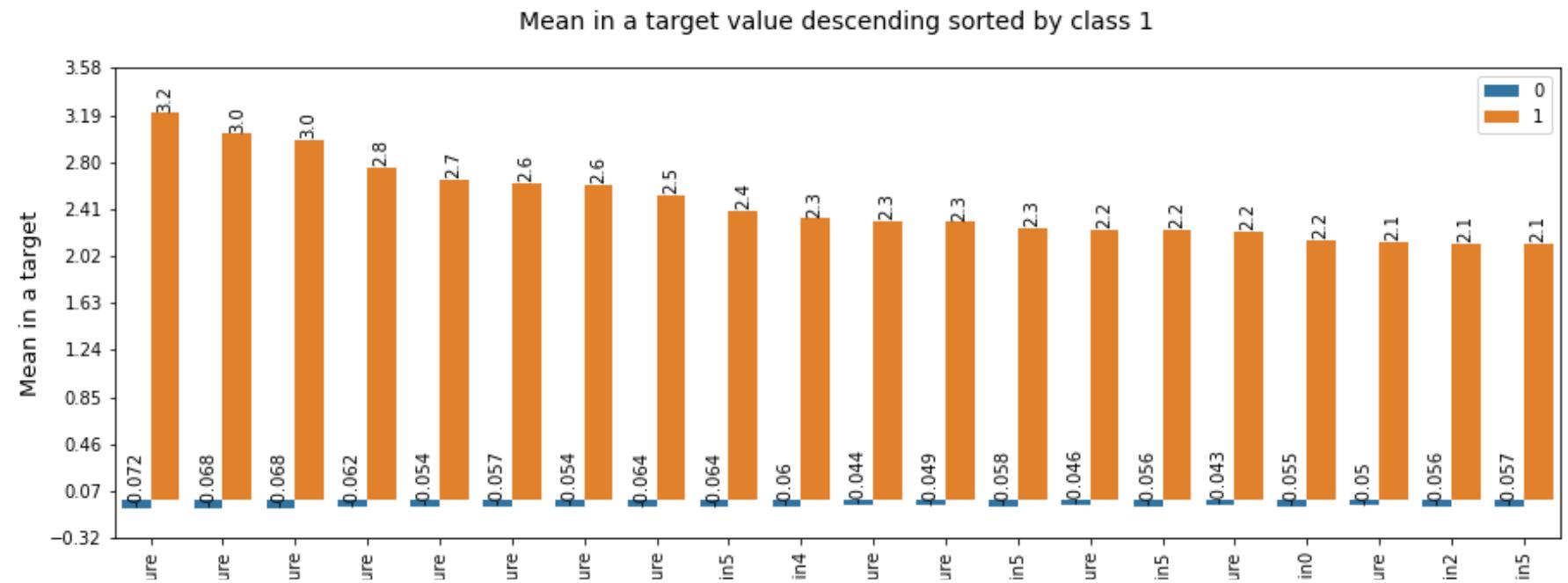
```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

```
In [ ]: counter=0
new_column_list=[]
new_mean_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_mean_list.append(mean_list[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_mean_list.append(mean_list[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 144 sensors.

In []:

```
try:  
    plot_diagram(new_column_list,new_mean_list,new_one_zero,"Mean",1)  
except ValueError:  
    pass
```



Observation : Observe that there is a significant difference between the median imputed adasyn means of the surface failures and the downhole failures.

Out of the selected features observe that the most of the features have a larger mean value for the downhole failure rather than that of the surface failures.

```
In [ ]: counter=0
mean_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.

    if (np.abs(np.abs(mean_list[i])-np.abs(mean_list[i-1]))>=.15):
        mean_higher_reading.append(mean_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

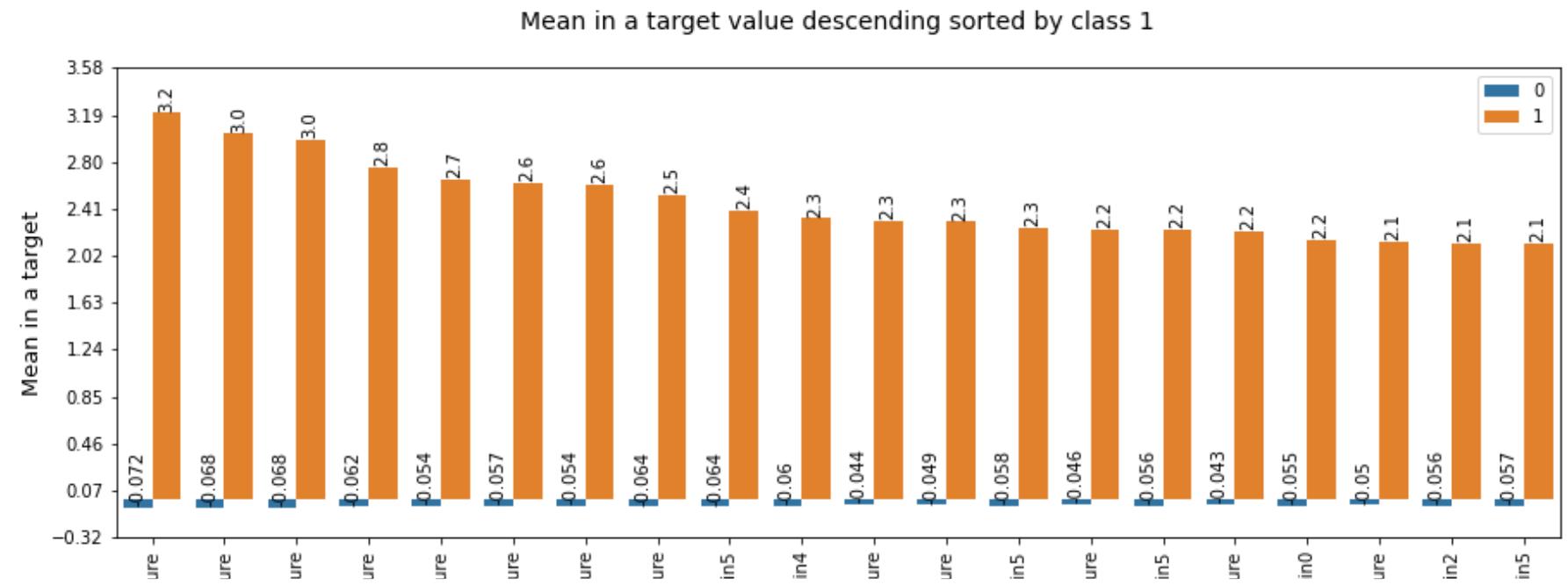
        mean_higher_reading.append(mean_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between Mean readings of su
```

So we have approximate 123 sensor that show significant difference (> .15) between Mean readings of surface and downhole failures.

In []:

```
try:  
    plot_diagram(column_higher_reading,mean_higher_reading,one_zero_higher_reading,"Mean",1)  
except ValueError:  
    pass
```



Compare the target 0 and target 1 Median values feature wise after median value imputing

```
In [ ]: column_list=[]
median_list=[]
one_zero=[]

for column in list(new_x_adasyn_df.columns.values):
    if column!="target":
        column_list.append(column)
        one_zero.append(0)
        median_list.append(new_x_adasyn_df[column][new_x_adasyn_df.target==0].median())
        column_list.append(column)
        one_zero.append(1)
        median_list.append(new_x_adasyn_df[column][new_x_adasyn_df.target==1].median())
```

**Find the features where the difference between the medians of the two classes is more than .15
descending sorted by class 1**

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:median_list[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

```
In [ ]: counter=0
new_column_list=[]
new_median_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_median_list.append(median_list[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_median_list.append(median_list[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 144 sensors.

```
In [ ]: try:
    plot_diagram(new_column_list,new_median_list,new_one_zero,"Median",1)
except ValueError:
    pass
```

Observation : Observe that there is a significant difference between the median imputed adasyn median of the surface failures and the downhole failures.

As this is a median , so the values are lesser as compared to the mean.

```
In [ ]: counter=0
median_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.

    if (np.abs(np.abs(median_list[i])-np.abs(median_list[i-1]))>=.15):
        median_higher_reading.append(median_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

        median_higher_reading.append(median_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

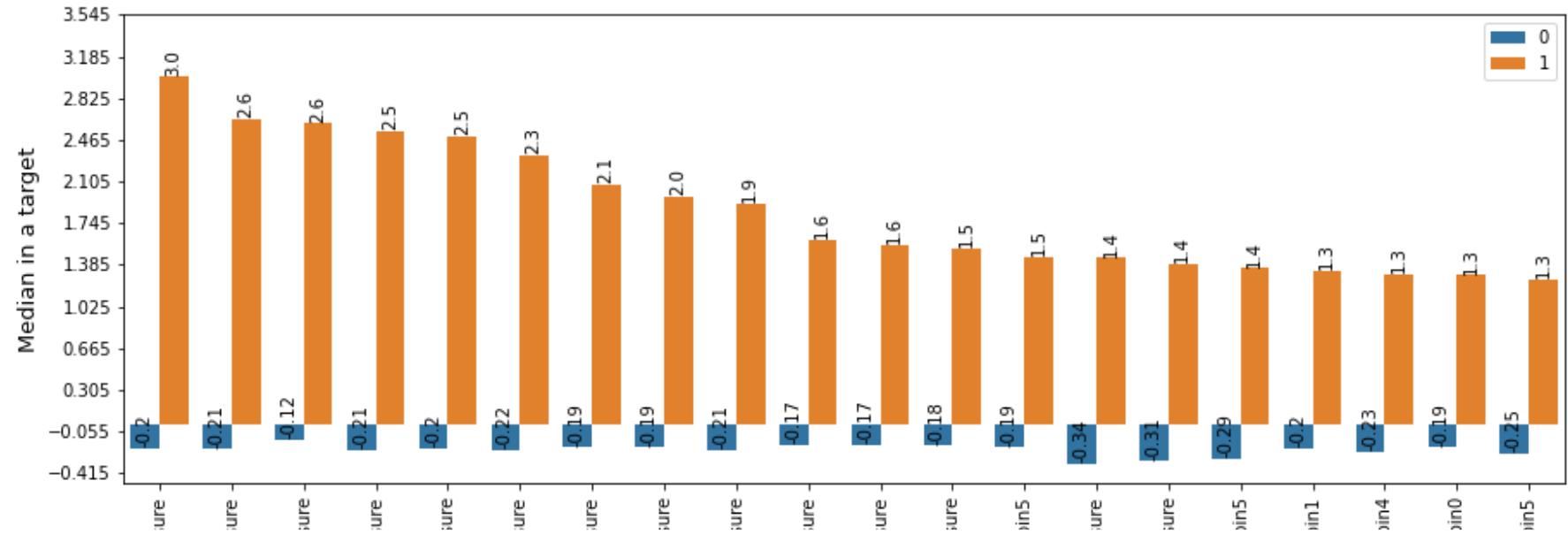
    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between median readings of
```

So we have approximate 59 sensor that show significant difference (> .15) between median readings of surface and downhole failures.

In []: **try:**

```
plot_diagram(column_higher_reading,median_higher_reading,one_zero_higher_reading,"Median",1)
except ValueError:
    pass
```

Median in a target value descending sorted by class 1



Basic Classifier

In []: `new_x_adasyn_df.drop("target",axis=1,inplace=True)`

```
print(dummy_classifier_based_on_median(new_x_adasyn_df.loc[72124],
                                         column_list,
                                         median_list))
```

Predicted surface failure.
[92, 52]

```
In [ ]: print(dummy_classifier_based_on_median(new_standard_test_adasyn_all_features_df.loc[11997],  
                                              column_list,  
                                              median_list))
```

Predicted downhole failure.
[68, 76]

```
In [ ]: new_x_adasyn_df["target"] = y_adasyn
```

EDA for the dataset

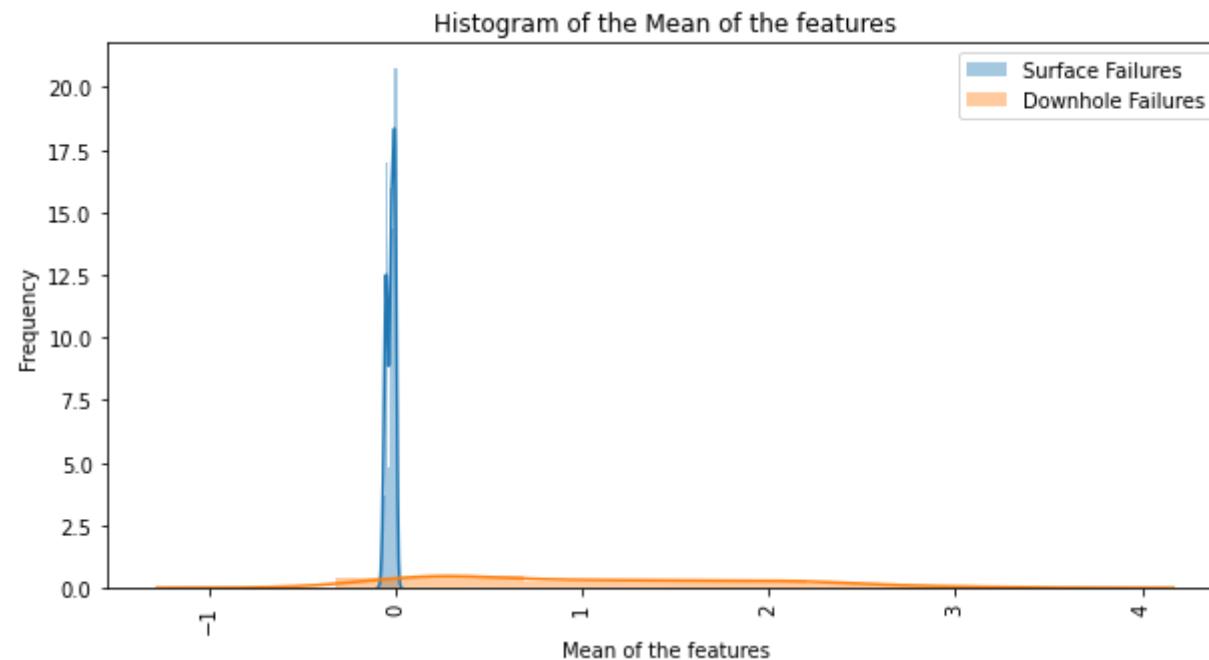
Using histograms, plots find out the nature of means for features for both the classes and thereby, the outliers.

```
In [ ]: mean_0 = []  
mean_1 = []  
for column in list(new_x_adasyn_df.columns.values):  
    if column != "target":  
        mean_0.append(new_x_adasyn_df[new_x_adasyn_df.target == 0][column].mean())  
        mean_1.append(new_x_adasyn_df[new_x_adasyn_df.target == 1][column].mean())
```

```
In [ ]: median_0 = []  
median_1 = []  
for column in list(new_x_adasyn_df.columns.values):  
    if column != "target":  
        median_0.append(new_x_adasyn_df[new_x_adasyn_df.target == 0][column].median())  
        median_1.append(new_x_adasyn_df[new_x_adasyn_df.target == 1][column].median())
```

10) Try and plot it's histogram for both the target values.

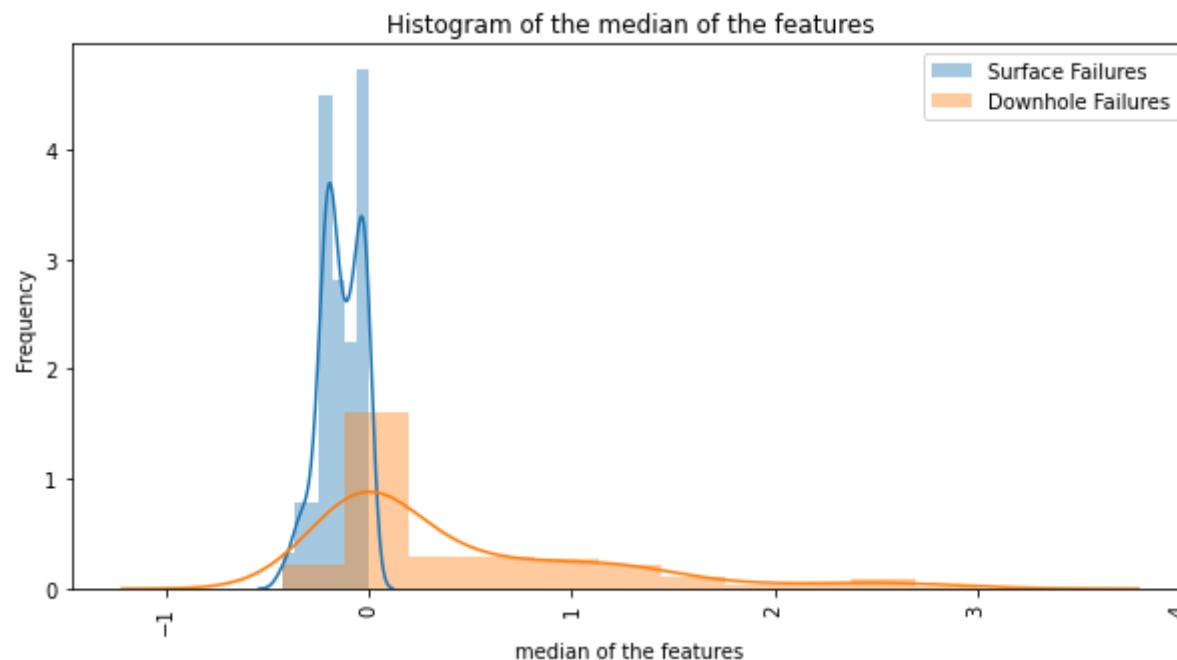
```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Histogram of the Mean of the features')
plt.ylabel('Frequency')
plt.xlabel('Mean of the features')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations :

Here for the surface and downhole failures, we can see that the different downhole features have more variance. Whereas the majority of the surface related features have a very similar value.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('Histogram of the median of the features')
plt.ylabel('Frequency')
plt.xlabel('median of the features')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



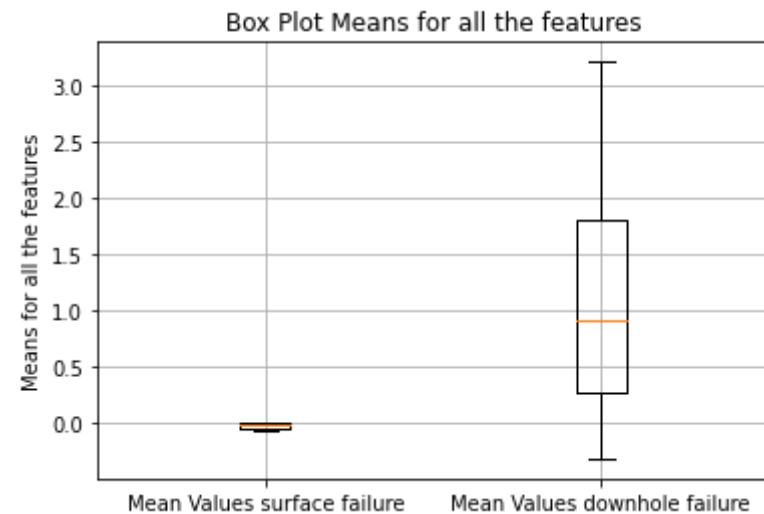
Observations :

Here for the surface and downhole failures, we can see that the different downhole features have more variance. Whereas the majority of the surface related features have a very similar value.

Similarity can be observed in the box plots and violin plots.

11) Try box plot / violin plot to visualize the outliers.

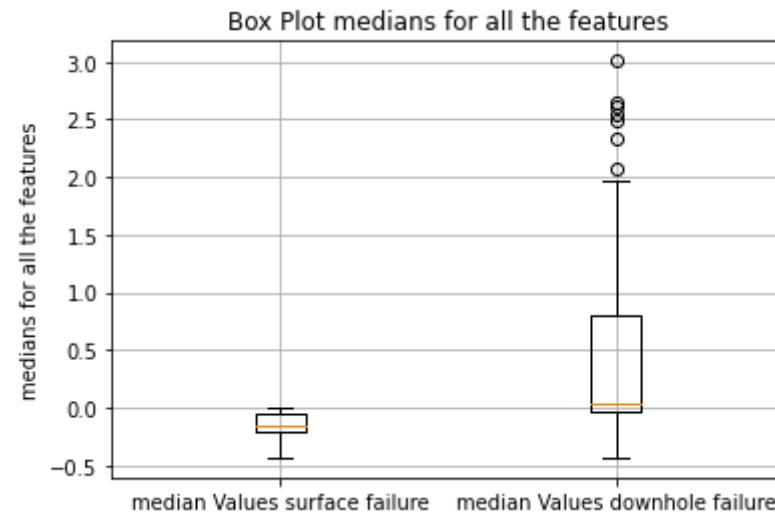
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Box Plot Means for all the features')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the features')  
plt.grid()  
plt.show()
```



Observations :

The majority of the values for the surface failures lie near 0 whereas for the downhole failures, we can see all kinds of variation.

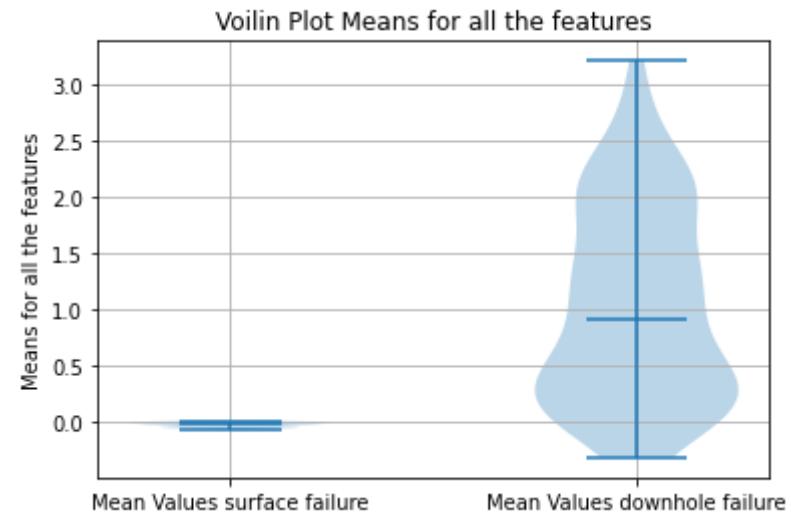
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('Box Plot medians for all the features')  
plt.xticks([1,2],('median Values surface failure','median Values downhole failure'))  
plt.ylabel('medians for all the features')  
plt.grid()  
plt.show()
```



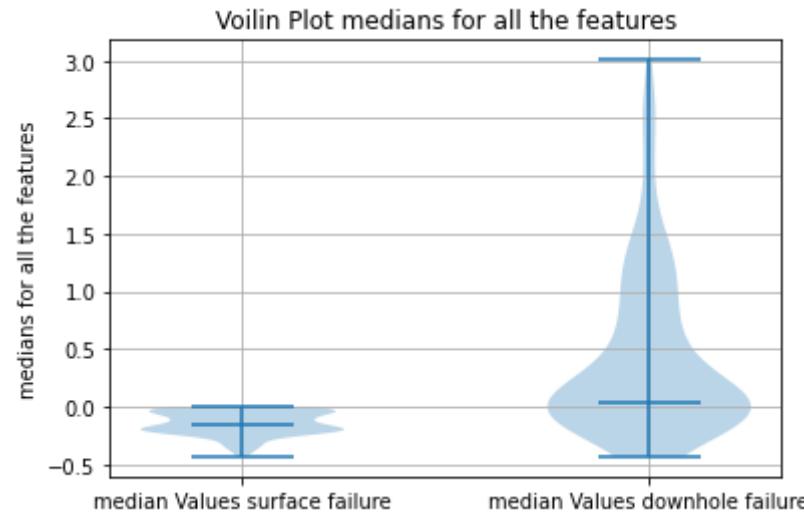
Observations :

The majority of the values for the surface failures lie near 0 whereas for the downhole failures, we can see all kinds of variation.

```
In [ ]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Violin Plot Means for all the features')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the features')
plt.grid()
plt.show()
```



```
In [ ]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('Violin Plot medians for all the features')
plt.xticks([1,2],('median Values surface failure','median Values downhole failure'))
plt.ylabel('medians for all the features')
plt.grid()
plt.show()
```



PCA

```
In [75]: new_x_adasyn_df.drop("target",axis=1,inplace=True)
```

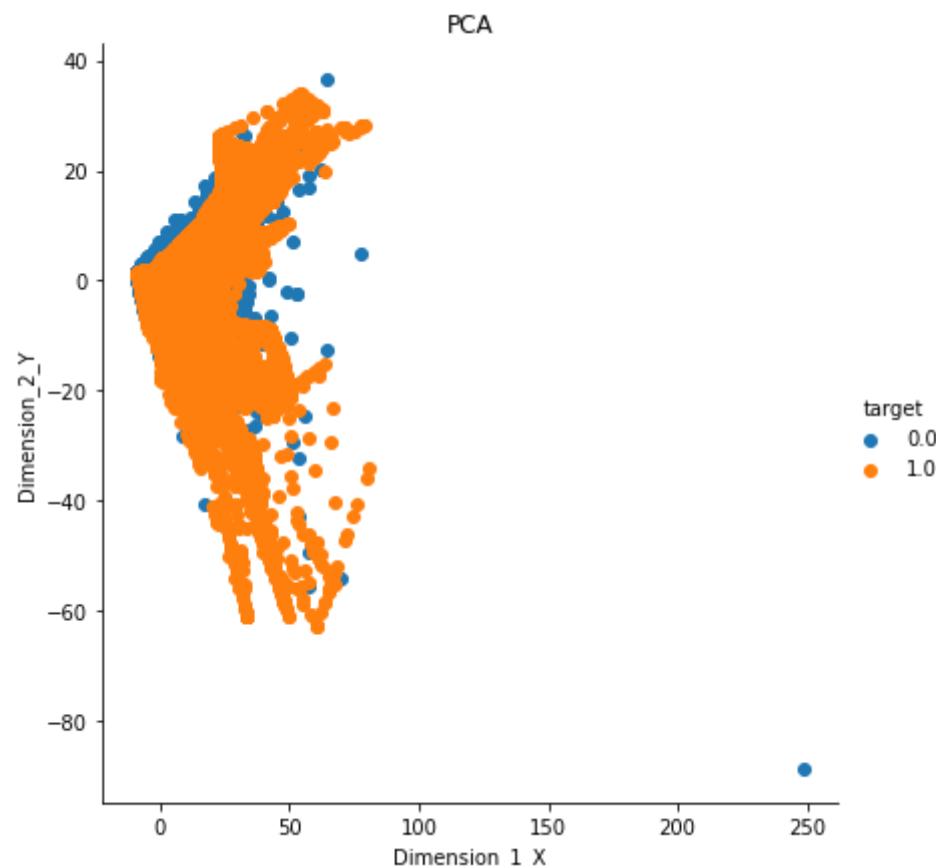
```
In [ ]: "target" in new_x_adasyn_df.columns
```

```
Out[465]: False
```

In [76]:

```
%%time  
run_pca(new_x_adasyn_df,y_adasyn)
```

```
[[ -7.816123   1.8352222   0.        ]  
 [-4.13487    1.2684838   0.        ]  
 [-9.20787    1.3386774   0.        ]  
 ...  
 [ 0.754264   -3.3310704   1.        ]  
 [ 3.1483057  -4.1678286   1.        ]  
 [-1.4895833  -1.9928197   1.        ]]
```



CPU times: user 898 ms, sys: 147 ms, total: 1.05 s
Wall time: 932 ms

Observation :

- 1) Observe that PCA is not able to differentiate properly the surface failures from the downhole failures. If the surface and downhole failures are coming under the same category then they are shown in the same cluster.
- 2) Not all the the surface and downhole features are coming under the same category.

Perform a similar EDA on the outliers.

Here we take the outliers from the features that have been selected.

For each sensor we find the outliers that also classwise.

The issue is that there maybe high outliers here , but there maybe lower outliers here as well.

<https://www.geeksforgeeks.org/interquartile-range-and-quartile-deviation-using-numpy-and-scipy/> (<https://www.geeksforgeeks.org/interquartile-range-and-quartile-deviation-using-numpy-and-scipy/>)

Higher outlier = more than $1.5 \cdot \text{IQR}$ above the third quartile.

Lower outlier = less than $1.5 \cdot \text{IQR}$ below the first quartile.

The IQR is the difference between Q3 and Q1.

```
In [ ]: new_x_adasyn_df["target"] = y_adasyn
```

```
In [ ]: counter=0
main_dictionary=dict()
outliers=[]
list_of_outliers_features_classes=[]
for i in range(len(column_list)):
    d=dict()
    new_list=new_x_adasyn_df[new_x_adasyn_df.target==one_zero[i]]\
    [column_list[i]]
    new_index=list(new_list.reset_index()["index"])
    QR1=np.percentile(new_list, 25)
    QR3=np.percentile(new_list, 75)
    IQR=QR3-QR1
    for index,value in zip(new_index,new_list):
        if (value>(QR3+(IQR*1.5))) or (value<(QR1-(IQR*1.5))):
            counter=counter+1
            d.update({index:value})
            if index not in main_dictionary:
                main_dictionary.update({index:list(new_x_adasyn_df.loc[index])})
    list_of_outliers_features_classes.append(d)
del d
outliers.append(counter)
counter=0
```

```
In [ ]: print("Total number of data points that are outliers in atleast one or more than one feature : ",len(np.unique(outliers)))
```

Total number of data points that are outliers in atleast one or more than one feature : 79888

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:outliers[i] for i in index}

d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])

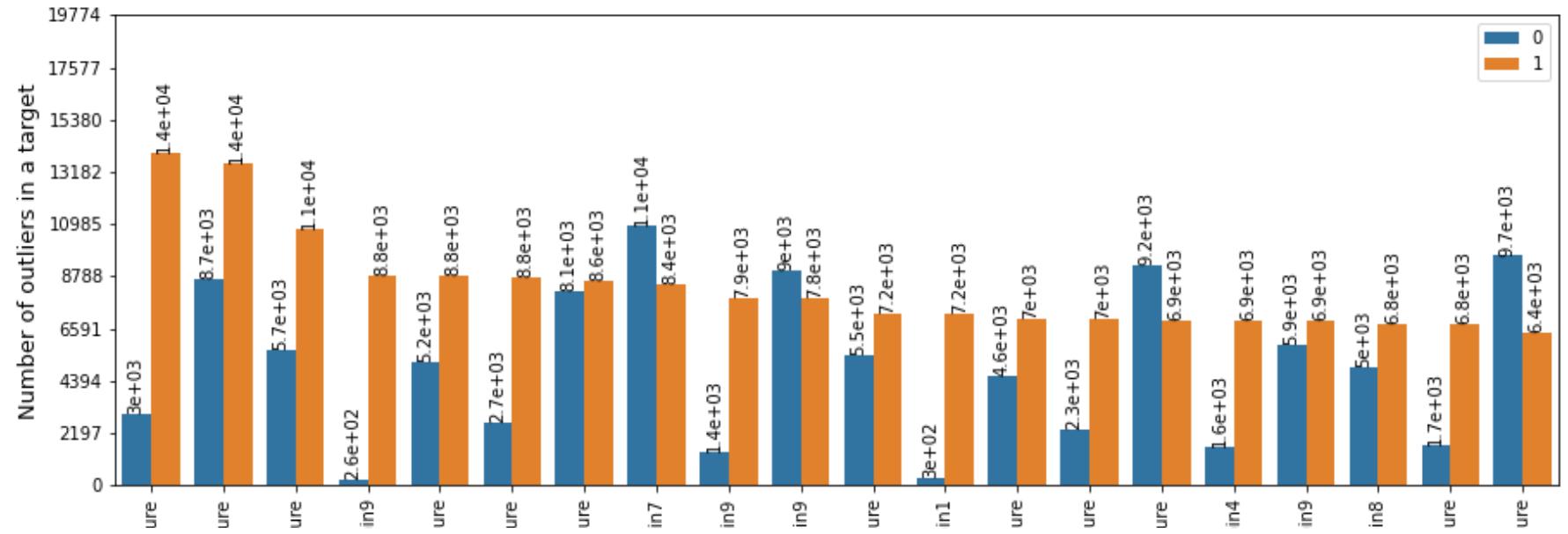
counter=0
new_column_list=[]
new_outliers_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_outliers_list.append(outliers[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_outliers_list.append(outliers[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 144 sensors.

In []: **try:**

```
plot_diagram(new_column_list,new_outliers_list,new_one_zero,"Number of outliers",1)
except ValueError:
    pass
```

Number of outliers in a target value descending sorted by class 1

**Observation :**

Now as we have oversampled the minority class , we can see that number of outliers for the downhole failures have started rising other than.

For every column feature, get the mean for the outliers for both class 0 and class 1.

```
In [ ]: mean_0=[]
mean_1=[]
for i in range(0,len(list_of_outliers_features_classes),2):
    class_0=[ele for ele in list_of_outliers_features_classes[i].values()]
    class_1=[ele for ele in list_of_outliers_features_classes[i+1].values()]
    if (len(class_0)==0) :
        mean_0.append(0.01)
    else :
        mean_0.append(np.mean(class_0))

    if (len(class_1)==0):
        mean_1.append(0.01)
    else :
        mean_1.append(np.mean(class_1))
```

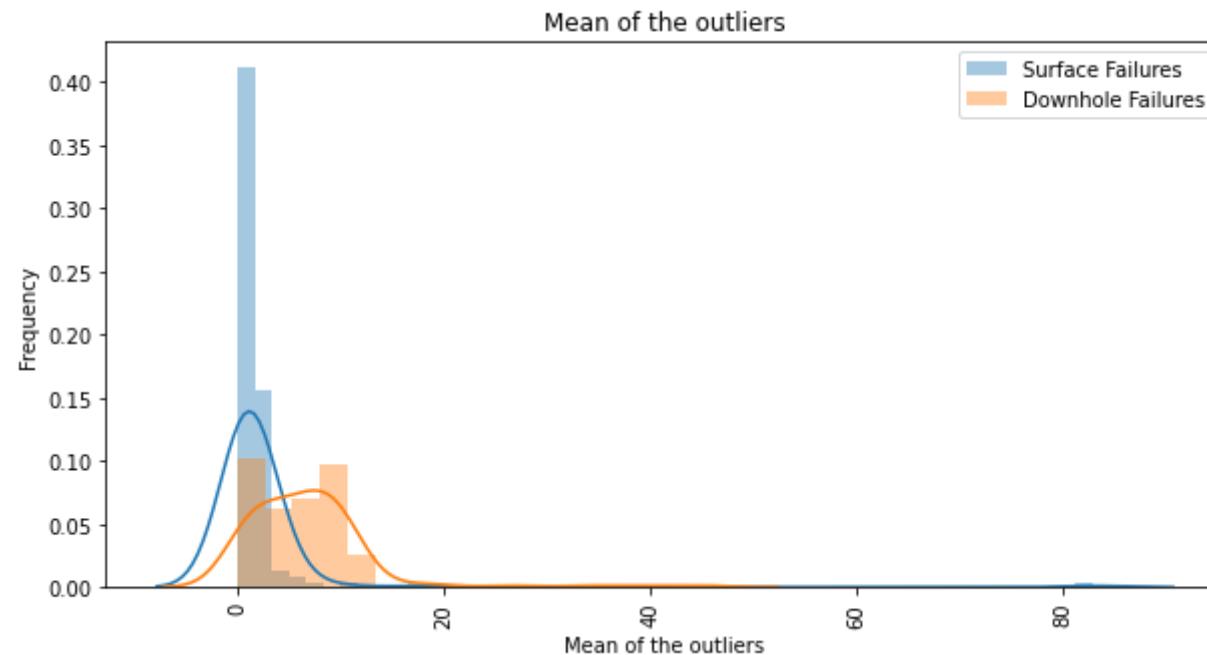
```
In [ ]: median_0=[]
median_1=[]
for i in range(0,len(list_of_outliers_features_classes),2):
    class_0=[ele for ele in list_of_outliers_features_classes[i].values()]
    class_1=[ele for ele in list_of_outliers_features_classes[i+1].values()]
    if (len(class_0)==0) :
        median_0.append(0.01)
    else :
        median_0.append(np.median(class_0))

    if (len(class_1)==0):
        median_1.append(0.01)
    else :
        median_1.append(np.median(class_1))
```

Plotting the mean and median for each feature class wise.

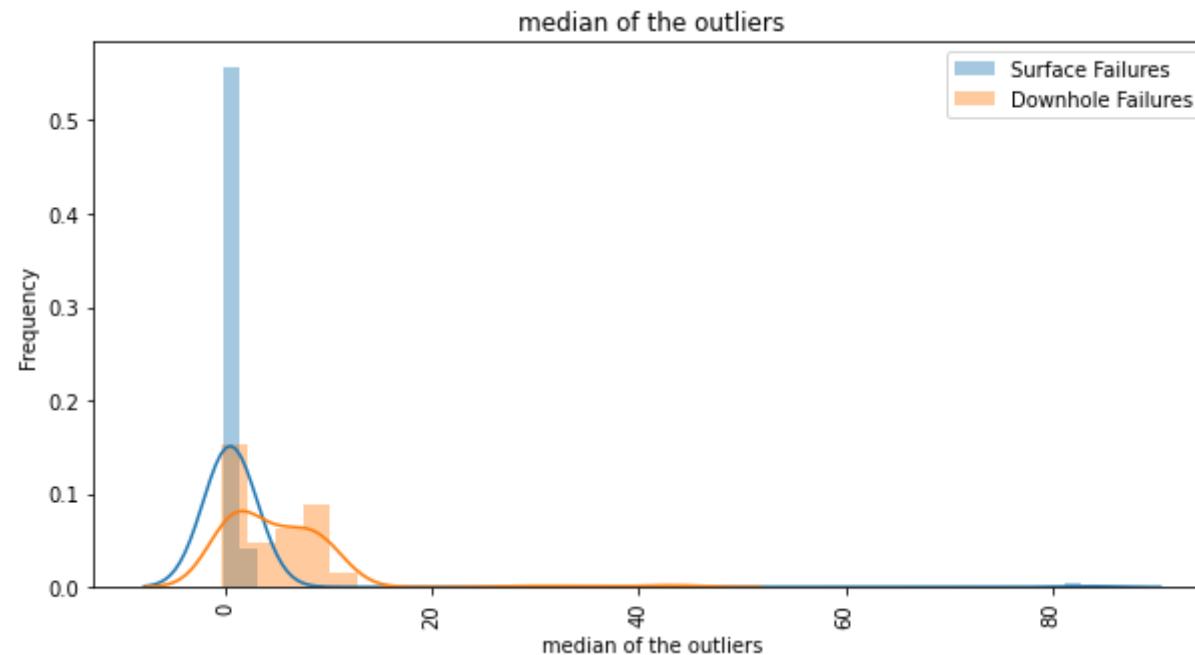
10) Try and plot it's histogram for both the target values.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Mean of the outliers')
plt.ylabel('Frequency')
plt.xlabel('Mean of the outliers')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations : For the outliers, majority of them seem to be concentrated around 0 for the surface outliers with variance as well. For downhole outliers they have lesser variance but have more density in the other range values

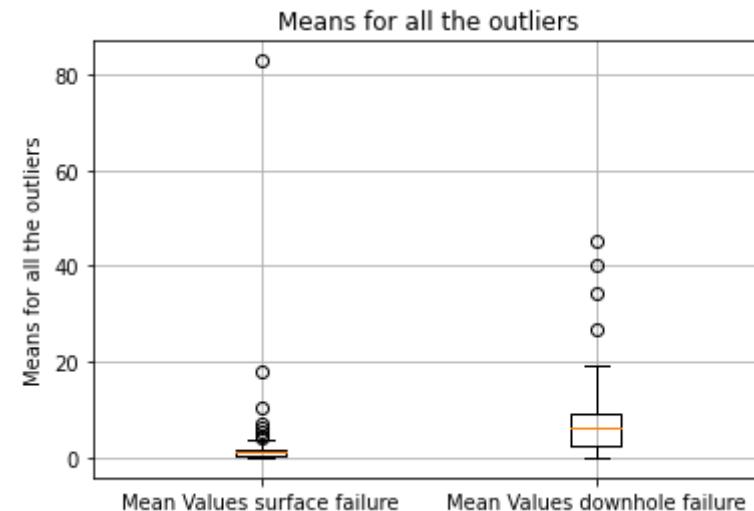
```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('median of the outliers')
plt.ylabel('Frequency')
plt.xlabel('median of the outliers')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



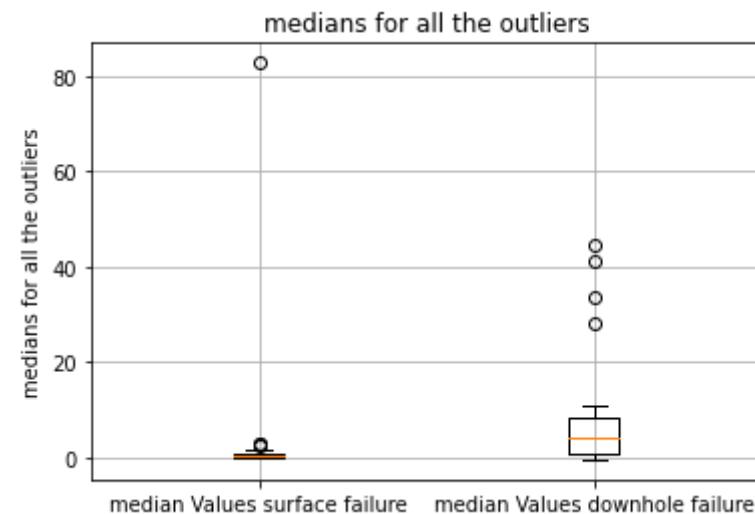
Observations : For the outliers, majority of them seem to be concentrated around 0 for the surface outliers with variance as well. For downhole outliers they have lesser variance but have more density in the other range values

11) Try box plot / violin plot to visualize the outliers.

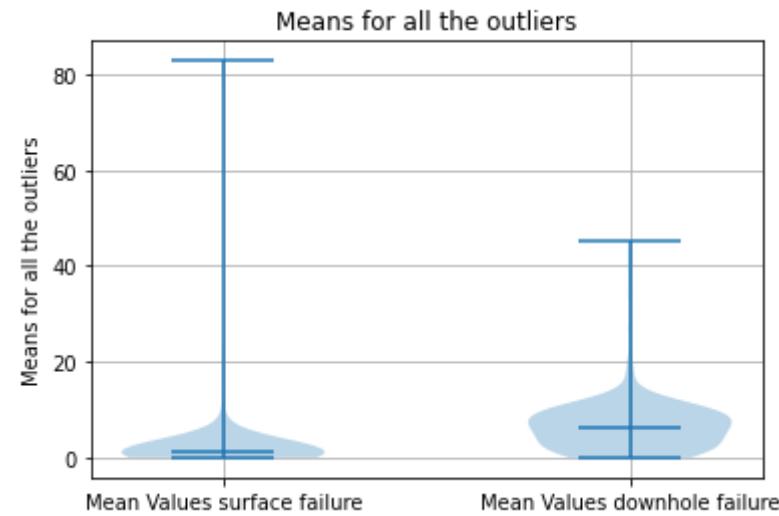
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Means for all the outliers')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the outliers')  
plt.grid()  
plt.show()
```



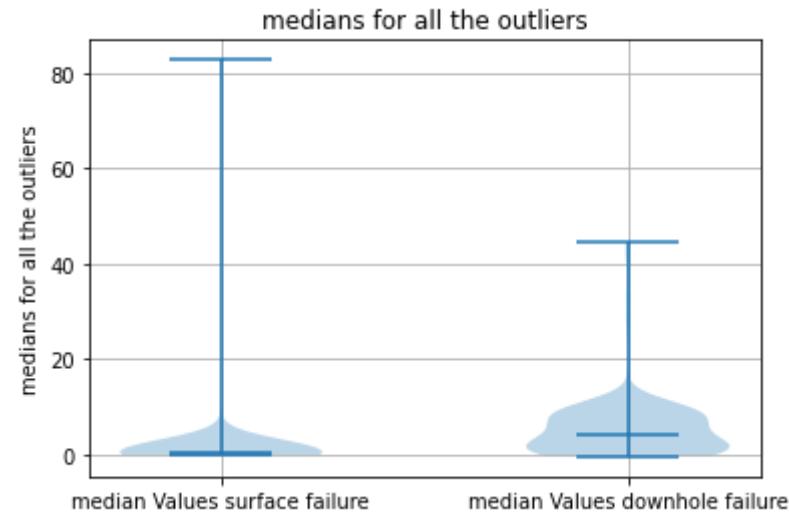
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('medians for all the outliers')  
plt.xticks([1,2],('median Values surface failure','median Values downhole failure'))  
plt.ylabel('medians for all the outliers')  
plt.grid()  
plt.show()
```



```
In [ ]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Means for all the outliers')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the outliers')
plt.grid()
plt.show()
```



```
In [ ]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('medians for all the outliers')
plt.xticks([1,2],('median Values surface failure','median Values downhole failure'))
plt.ylabel('medians for all the outliers')
plt.grid()
plt.show()
```



The outliers although for the surface related failures are higher. But both the means and the medians of the outliers are similar.

For the given data points and features enter new features that tell if these values were previously np.NaN or not.

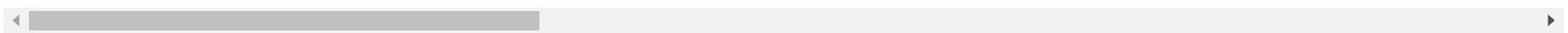
```
In [ ]: new_x_adasyn_df.drop("target",axis=1,inplace=True)
```

```
In [ ]: nan_x_adasyn_df=fuse_dataframe_nan(new_x_adasyn_df,nan_or_not,set(new_column_list))
nan_x_adasyn_df["target"]=y_adasyn
nan_x_adasyn_df.head()
```

Out[480]:

	sensor7_histogram_bin2	sensor105_histogram_bin3	sensor69_histogram_bin7	sensor64_histogram_bin2	sensor107_measure	sensor40_measure
0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	1.0

5 rows × 145 columns



Now this is the problem out of the given data that we have , we select the features that have more than 50% np.NANs

```
In [ ]: counter=0
for column in new_x_adasyn_df.columns.values:
    try:
        percent_of_nan=nan_x_adasyn_df[column].value_counts()[1]/\
            (nan_x_adasyn_df[column].value_counts()[0]+\
            nan_x_adasyn_df[column].value_counts()[1])
        if (percent_of_nan>.5) :
            counter=counter+1
            print(column," : ",percent_of_nan)
    except KeyError:
        pass
print("Overall : ",counter," features with % of na's more than 50%")
```

```
sensor2_measure      :  0.7740833333333333
sensor38_measure     :  0.6605416666666667
sensor39_measure     :  0.7348958333333333
sensor40_measure     :  0.77375
sensor41_measure     :  0.7967916666666667
sensor42_measure     :  0.8133125
sensor43_measure     :  0.822
sensor68_measure     :  0.7740833333333333
Overall :  8  features with % of na's more than 50%
```

5) Impute np.NaN with median and then perform smotetomek on the values.

```
In [15]: data_type="smotetomek_data"
```

```
In [ ]: train_smotetomek_all_features=train.copy()
y_train_smotetomek_all_features=y_train.copy()
test_smotetomek_all_features=test.copy()
y_test_smotetomek_all_features=y_test.copy()
```

```
In [ ]: train_smotetomek_all_features.shape
```

```
Out[484]: (48000, 170)
```

```
In [ ]: y_train_smotetomek_all_features.shape
```

```
Out[485]: (48000,)
```

```
In [ ]: test_smotetomek_all_features.shape
```

```
Out[486]: (12000, 170)
```

```
In [ ]: y_test_smotetomek_all_features.shape
```

```
Out[487]: (12000,)
```

```
In [ ]: for i ,j in zip(train_smotetomek_all_features.columns.values,test_smotetomek_all_features.columns.values):
         if (i!=j):
             print(i," : ",j)
```

```
In [ ]: train_smotetomek_all_features,y_train_smotetomek_all_features=reset_index(train_smotetomek_all_features,y_tr
test_smotetomek_all_features,y_test_smotetomek_all_features=reset_index(test_smotetomek_all_features,y_test_
```

Filling the np.NaN values with the median of the category this time.

```
In [ ]: l=[]
         for ele in train_smotetomek_all_features["sensor4_measure"]:
             if np.isnan(ele):
                 pass
             else:
                 l.append(ele)
         np.median(l)
```

```
Out[490]: 126.0
```

```
In [ ]: train_smotetomek_all_features["sensor4_measure"].median()
```

```
Out[491]: 126.0
```

Both are the same so the pandas.series.median() function ignores the value that has the np.NaN and does not even count it.

Divide the whole dataframe into the number of categories that we have and then only enter the median for that feature for that particular category.

```
In [ ]: train_smotetomek_all_features,y_train_smotetomek_all_features,test_smotetomek_all_features,y_test_smotetomek_impute_median_for_train_n_test_as_per_train(train_smotetomek_all_features,
                                                    y_train_smotetomek_all_features,
                                                    test_smotetomek_all_features,
                                                    y_test_smotetomek_all_features,data_type)
```

```
In [ ]: train_smotetomek_all_features,y_train_smotetomek_all_features=reset_index(train_smotetomek_all_features,y_train_smotetomek_all_features),
test_smotetomek_all_features,y_test_smotetomek_all_features=reset_index(test_smotetomek_all_features,y_test_smotetomek_all_features)
```

Standardise the dataframe

```
In [ ]: standard_train_smotetomek_all_features_df,standard_test_smotetomek_all_features_df=standardize_test_n_train_
```

```
In [ ]: standard_train_smotetomek_all_features_df.shape
```

```
Out[495]: (48000, 170)
```

```
In [ ]: standard_test_smotetomek_all_features_df.shape
```

```
Out[496]: (12000, 170)
```

```
In [ ]: y_train_smotetomek_all_features.head()
```

```
Out[497]: 0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: target, dtype: float32
```

```
In [ ]: y_train_smotetomek_all_features.tail()
```

```
Out[498]: 47995    1.0
47996    1.0
47997    1.0
47998    1.0
47999    1.0
Name: target, dtype: float32
```

```
In [ ]: "target" in standard_train_smotetomek_all_features_df.columns
```

```
Out[499]: False
```

Perform smotetomek

```
In [ ]: %%time
from imblearn.combine import SMOTETomek
x_smotetomek, y_smotetomek = SMOTETomek(random_state=42, sampling_strategy=.75, n_jobs=-1).fit_resample(standard_train_smotetomek_all_features_df)
print("done")
```

```
done
CPU times: user 41min 17s, sys: 604 ms, total: 41min 18s
Wall time: 6min 16s
```

```
In [ ]: x_smotetomek.shape
```

```
Out[501]: (82598, 170)
```

```
In [ ]: y_smotetomek.value_counts()
```

```
Out[502]: 0.0    47199
1.0    35399
Name: target, dtype: int64
```

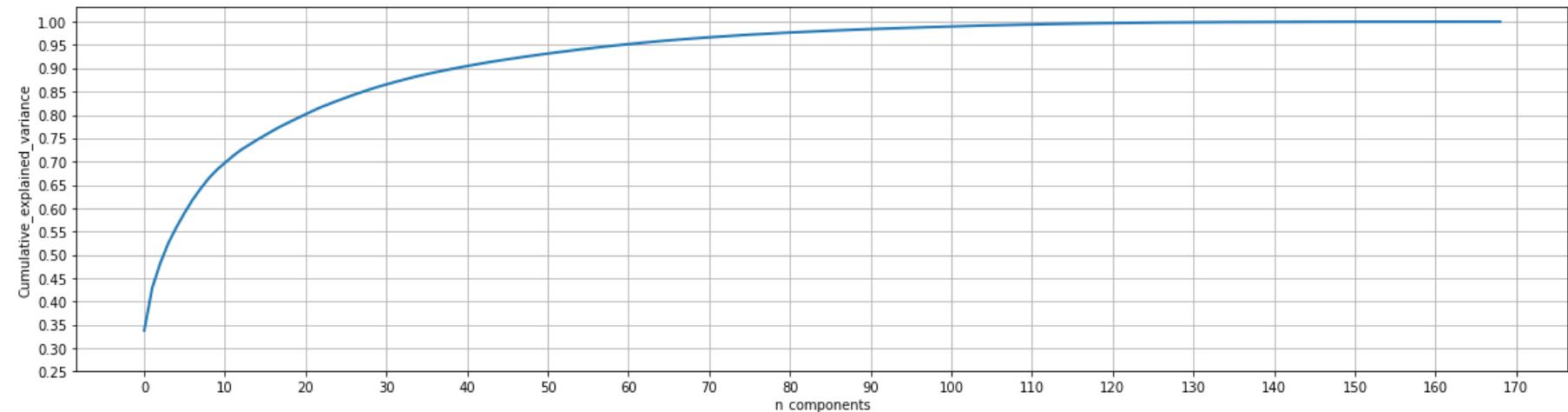
```
In [ ]: y_smotetomek.shape
```

```
Out[503]: (82598,)
```

Truncated SVD to get the best features

```
In [ ]: #https://stackoverflow.com/questions/50796024/feature-variable-importance-after-a-pca-analysis
```

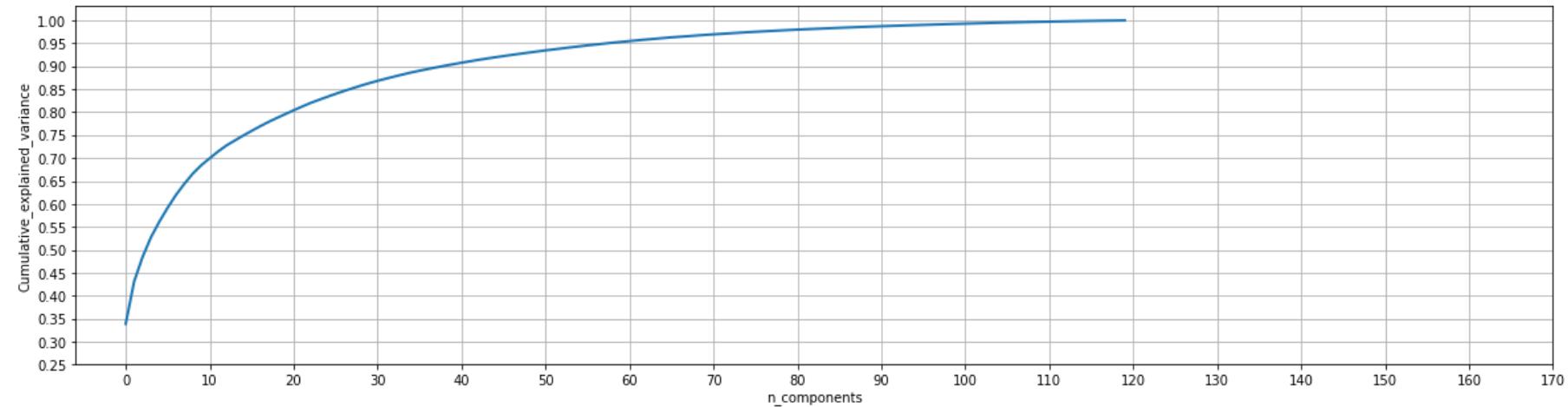
```
In [ ]: truncated_svd(x_smotetomek, 169)
```



```
Out[505]: TruncatedSVD(n_components=169)
```

I choose 120 components to describe my data as 120 components explain atleast 99% variance of the total data.

```
In [ ]: svd=truncated_svd(x_smotetomek, 120)
```



For the 120 principal components using svd.explained_variance_ratio_ find the variance distribution.

```
In [ ]: print(len(svd.explained_variance_ratio_))
```

120

```
In [ ]: np.sum(svd.explained_variance_ratio_)
```

Out[508]: 0.9967885

```
In [ ]: svd.components_.shape
```

```
Out[509]: (120, 170)
```

```
In [ ]: #number of features  
svd.components_.shape[1]
```

```
Out[510]: 170
```

```
In [ ]: #number of components  
svd.components_.shape[0]
```

```
Out[511]: 120
```

For the 120 principal components find the feature importance.

```
In [ ]: set_number_of_features=120
```

Trying Forward Feature selection

Here, in forward feature selection, out of 170 features even if I choose to select 10 features only , still it will take a lot of time.

Therefore I go with recursive feature elimination.

Using Recursive Feature Elimination, I can get the features that I need.

```
In [ ]: x_smotetomek.shape
```

```
Out[513]: (82598, 170)
```

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

Out[515]:

	sensor1_measure	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_hist
0	-0.209185	-0.432133	-0.078741	-0.028788	-0.057799	-0.115367	
1	0.226385	2.314107	0.276872	-0.028788	-0.057799	-0.115367	
2	-0.411601	-0.432133	-0.108583	-0.028788	-0.057799	-0.114391	
3	0.027280	-0.432131	0.303606	-0.028788	-0.057799	-0.115367	
4	-0.132515	-0.432132	0.031300	-0.028788	-0.057799	-0.115367	

5 rows × 145 columns

Try spearman corelation coefficient to find out co relation. (It is good for cause and affect analysis but it might not be of much use when we find that our features are dependent on other features, we expect our features to be independent of each other.)

```
In [ ]: %%time
spearman=select_spearman(new_x_smotetomek_df,int(set_number_of_features+np.floor((len(train.columns.values)-1)/2)))
If I take the data having correlation between only -.1 and .1 range then I get 0 features.
If I take the data having correlation between only -.90 and .90 range then I get 54 features.
If I take the data having correlation between only -.95 and .95 range then I get 76 features.
If I take the data having correlation between only -.97 and .97 range then I get 87 features.
If I take the data having correlation between only -.98 and .98 range then I get 97 features.
If I take the data having correlation between only -.99 and .99 range then I get 117 features.
If I take the data having correlation between only -.999 and .999 range then I get 139 features.
CPU times: user 22.2 s, sys: 56.2 ms, total: 22.3 s
Wall time: 22.2 s
```

```
In [ ]: new_x_smotetomek_df=perform_spearman(new_x_smotetomek_df,spearman,.99)
```

```
In [ ]: new_x_smotetomek_df.head()
```

Out[518]:

	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_histogram_bin4	sensc
0	-0.432133	-0.078741	-0.028788	-0.057799	-0.115367	-0.179786	
1	2.314107	0.276872	-0.028788	-0.057799	-0.115367	-0.175033	
2	-0.432133	-0.108583	-0.028788	-0.057799	-0.114391	-0.176506	
3	-0.432131	0.303606	-0.028788	-0.057799	-0.115367	-0.180203	
4	-0.432132	0.031300	-0.028788	-0.057799	-0.115367	-0.180360	

5 rows × 117 columns

```
In [ ]: new_x_smotetomek_df.shape
```

Out[519]: (82598, 117)

```
In [ ]: y_smotetomek.shape
```

```
Out[520]: (82598,)
```

```
In [ ]: new_standard_test_smotetomek_all_features_df=standard_test_smotetomek_all_features_df[new_x_smotetomek_df.co  
new_standard_test_smotetomek_all_features_df.head()
```

```
Out[521]:
```

	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_histogram_bin4	sens
0	-0.432133	-0.078741	-0.028788	-0.057799	-0.115367	-0.180209	
1	2.314107	0.135746	-0.028788	-0.057799	-0.115367	-0.180150	
2	-0.432133	-0.115422	-0.028788	-0.057799	-0.115367	-0.180461	
3	2.314107	-0.114800	-0.028788	-0.057799	-0.115367	-0.180276	
4	2.314107	0.041869	-0.028788	-0.057799	-0.115367	-0.180369	

5 rows × 117 columns

```
In [ ]: #new_x_smotetomek_df.to_pickle("new_x_smotetomek_df.pickle")  
#y_smotetomek.to_pickle("y_smotetomek.pickle")
```

```
#new_standard_test_smotetomek_all_features_df.to_pickle("new_standard_test_smotetomek_all_features_df.pickle")  
#y_test_smotetomek_all_features.to_pickle("y_test_smotetomek_all_features.pickle")
```

```
In [79]: #new_x_smotetomek_df=pd.read_pickle("/content/gdrive/My Drive/new_x_smotetomek_df.pickle")  
#y_smotetomek=pd.read_pickle("/content/gdrive/My Drive/y_smotetomek.pickle")
```

```
#new_standard_test_smotetomek_all_features_df=pd.read_pickle("/content/gdrive/My Drive/new_standard_test_smotetomek_all_features_df.pickle")  
#y_test_smotetomek_all_features=pd.read_pickle("/content/gdrive/My Drive/y_test_smotetomek_all_features.pickle")
```

```
In [ ]: np.save('new_x_smotetomek_df_columns.npy',new_x_smotetomek_df.columns.values)
```

```
In [80]: new_x_smotetomek_df["target"] = y_smotetomek
new_x_smotetomek_df.head()
```

Out[80]:

	sensor3_measure	sensor4_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_bin3	sensor7_histogram_bin4	ser
0	-0.432133	-0.078741	-0.028788	-0.057799	-0.115367	-0.179786	
1	2.314107	0.276872	-0.028788	-0.057799	-0.115367	-0.175033	
2	-0.432133	-0.108583	-0.028788	-0.057799	-0.114391	-0.176506	
3	-0.432131	0.303606	-0.028788	-0.057799	-0.115367	-0.180203	
4	-0.432132	0.031300	-0.028788	-0.057799	-0.115367	-0.180360	

5 rows × 118 columns

Compare the target 0 and target 1 mean values feature wise after Median value imputing

```
In [ ]: column_list=[]
mean_list=[]
one_zero=[]
for column in list(new_x_smotetomek_df.columns.values):
    if column!="target":
        column_list.append(column)
        one_zero.append(0)
        mean_list.append(new_x_smotetomek_df[column][new_x_smotetomek_df.target==0].mean())

        column_list.append(column)
        one_zero.append(1)
        mean_list.append(new_x_smotetomek_df[column][new_x_smotetomek_df.target==1].mean())
```

Find the features where the difference between the means of the two classes is more than .15
descending sorted by class 1

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:mean_list[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:-len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

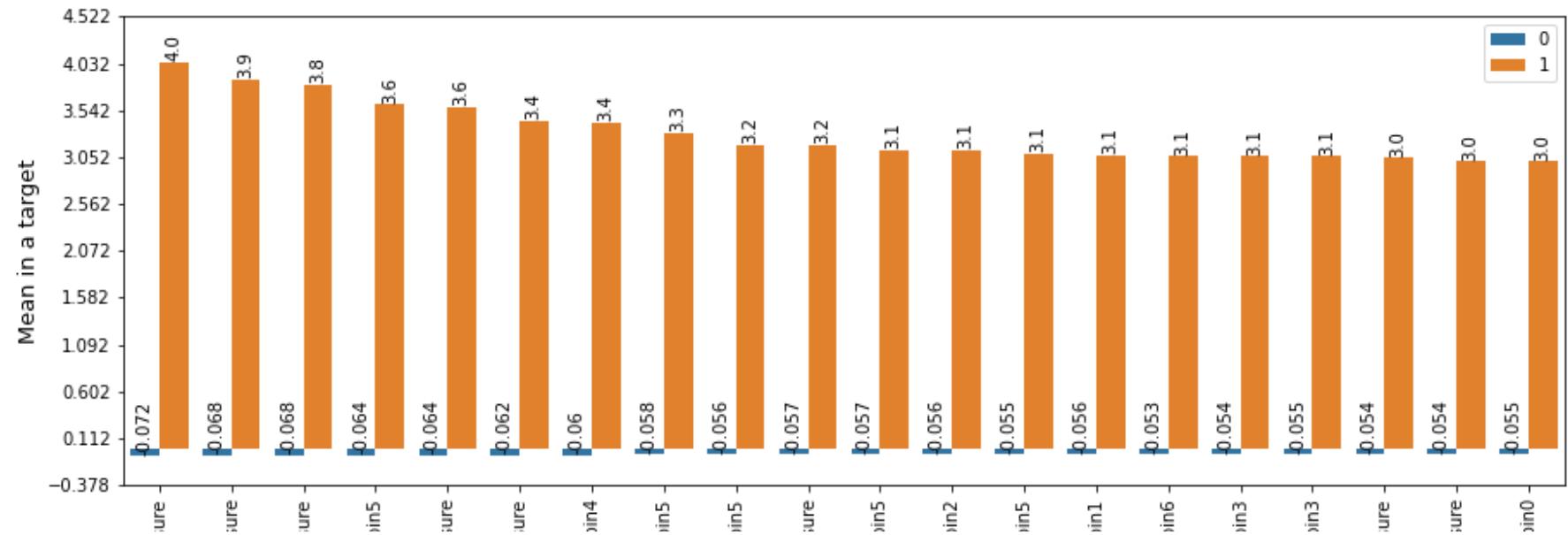
```
In [ ]: counter=0
new_column_list=[]
new_mean_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_mean_list.append(mean_list[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_mean_list.append(mean_list[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 117 sensors.

In []:

```
try:  
    plot_diagram(new_column_list,new_mean_list,new_one_zero,"Mean",1)  
except ValueError:  
    pass
```

Mean in a target value descending sorted by class 1



Observation : All the mean values that we have taken from every median imputed feature, we come to know that there is a difference between both the types of the failures.

```
In [ ]: counter=0
mean_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.

    if (np.abs(np.abs(mean_list[i])-np.abs(mean_list[i-1]))>=.15):
        mean_higher_reading.append(mean_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

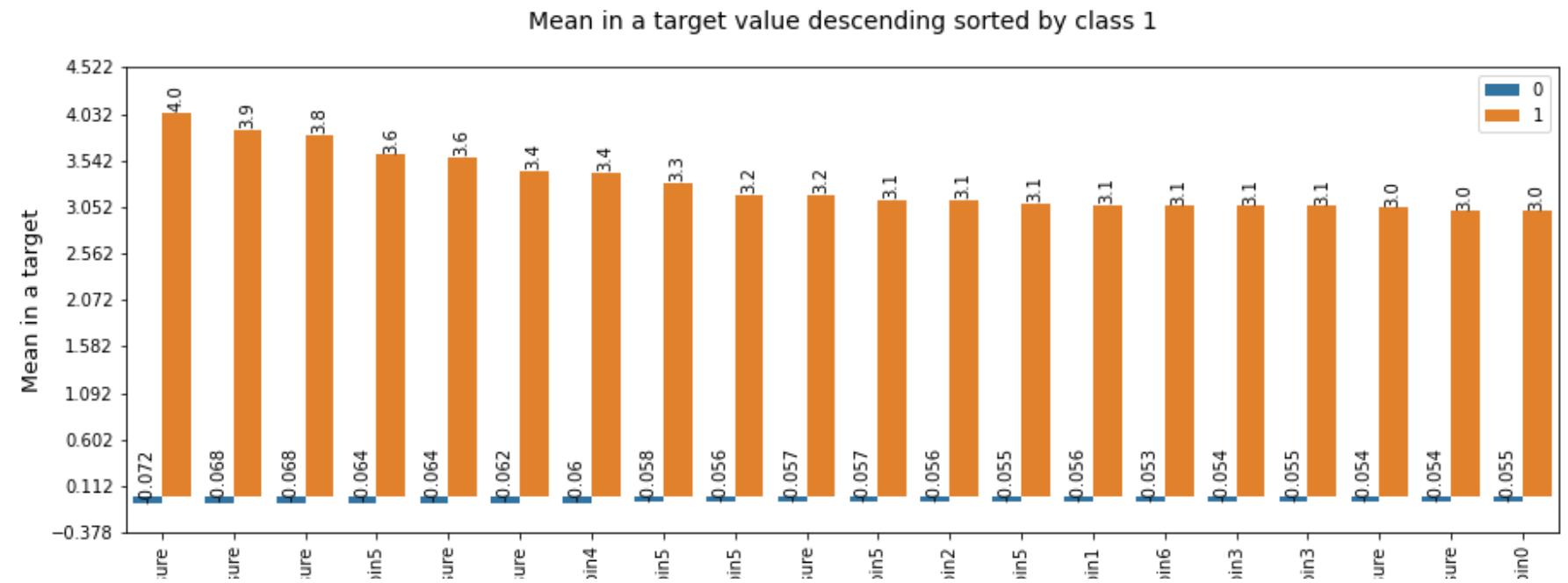
        mean_higher_reading.append(mean_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between mean readings of su
```

So we have approximate 108 sensor that show significant difference (> .15) between mean readings of surface and downhole failures.

In []:

```
try:  
    plot_diagram(column_higher_reading,mean_higher_reading,one_zero_higher_reading,"Mean",1)  
except ValueError:  
    pass
```



Compare the target 0 and target 1 median values feature wise after median value imputing

```
In [ ]: column_list=[]
median_list=[]
one_zero=[]

for column in list(new_x_smotetomek_df.columns.values):
    if column!="target":
        column_list.append(column)
        one_zero.append(0)
        median_list.append(new_x_smotetomek_df[column][new_x_smotetomek_df.target==0].median())
        column_list.append(column)
        one_zero.append(1)
        median_list.append(new_x_smotetomek_df[column][new_x_smotetomek_df.target==1].median())
```

**Find the features where the difference between the medians of the two classes is more than .15
descending sorted by class 1**

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:median_list[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

```
In [ ]: counter=0
new_column_list=[]
new_median_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_median_list.append(median_list[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_median_list.append(median_list[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 117 sensors.

```
In [ ]: try:
    plot_diagram(new_column_list,new_median_list,new_one_zero,"Median",1)
except ValueError:
    pass
```

Observation : All the median values that we have taken from every median imputed feature, we come to know that there is a difference between both the types of the failures but most of the surface failures go down to a negative value.

```
In [ ]: counter=0
median_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.

    if (np.abs(np.abs(median_list[i])-np.abs(median_list[i-1]))>=.15):
        median_higher_reading.append(median_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

        median_higher_reading.append(median_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

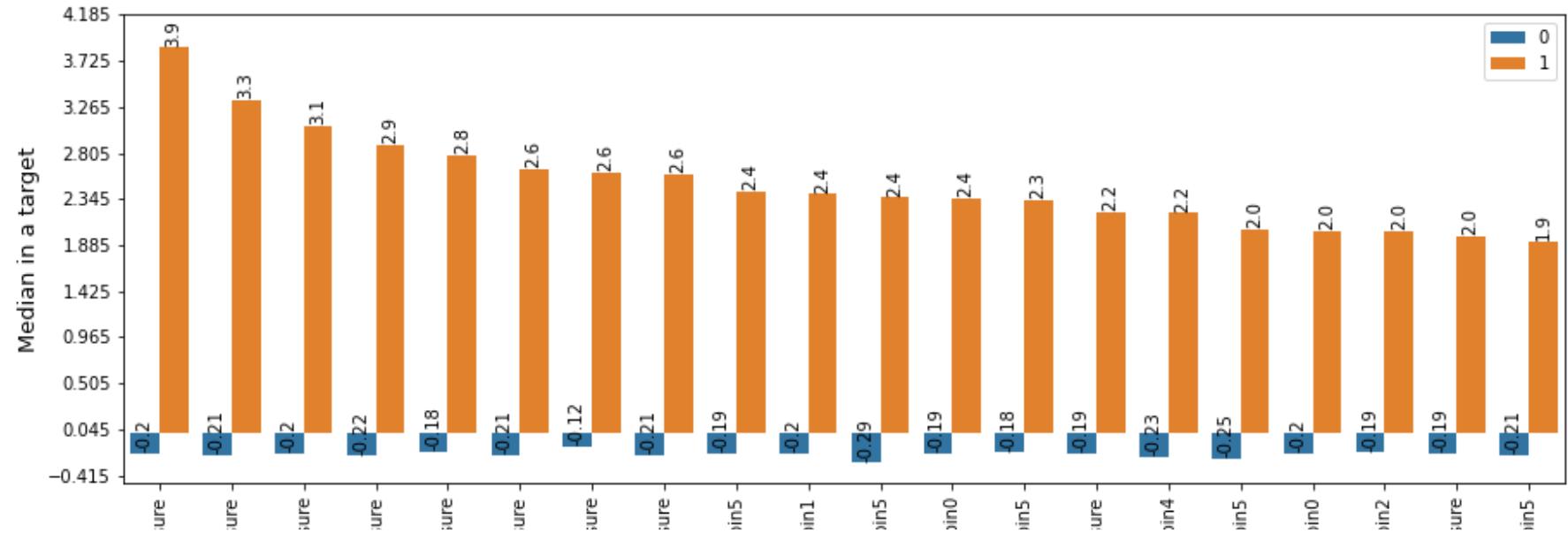
    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between median readings of surface and downhole failures.")
```

So we have approximate 64 sensor that show significant difference (> .15) between median readings of surface and downhole failures.

In []: **try:**

```
plot_diagram(column_higher_reading,median_higher_reading,one_zero_higher_reading,"Median",1)
except ValueError:
    pass
```

Median in a target value descending sorted by class 1



Basic Classifier

In []: `new_x_smotetomek_df.drop("target",axis=1,inplace=True)`

```
print(dummy_classifier_based_on_median(new_x_smotetomek_df.loc[63454],
                                         column_list,
                                         median_list))
```

Predicted downhole failure.
[32, 85]

```
In [ ]: print(dummy_classifier_based_on_median(new_standard_test_smotetomek_all_features_df.loc[1700],  
                                              column_list,  
                                              median_list))
```

Predicted surface failure.
[109, 8]

```
In [ ]: new_x_smotetomek_df["target"] = y_smotetomek
```

EDA for the dataset

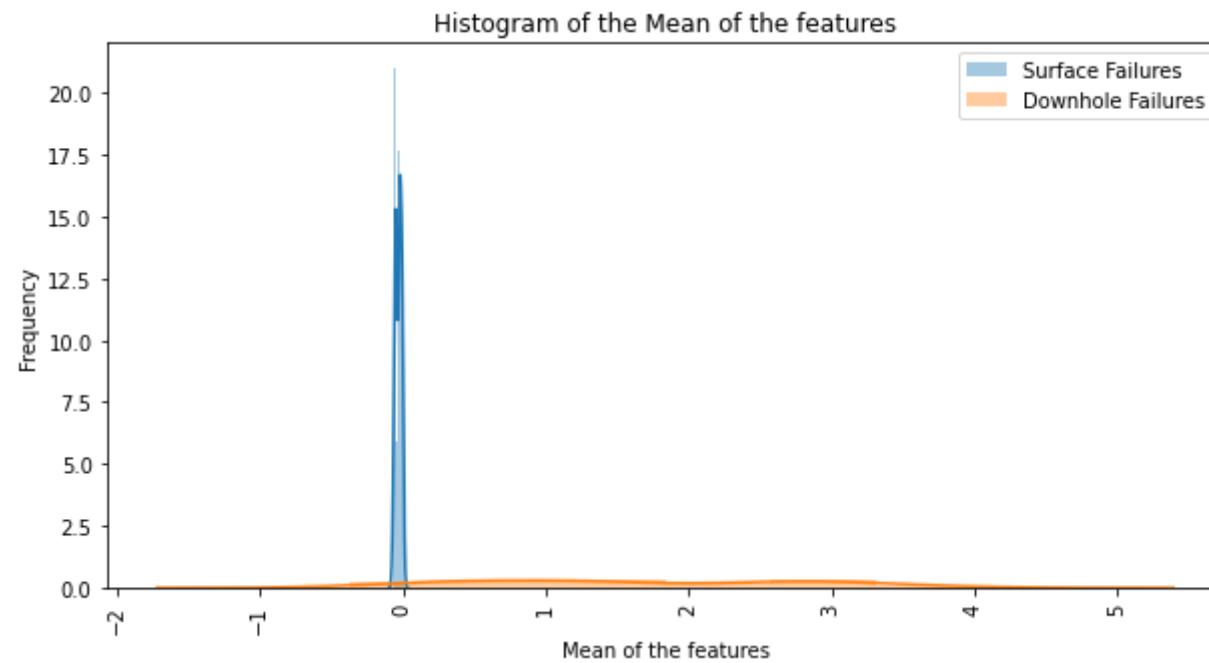
Using histograms, plots find out the nature of means for features for both the classes and thereby, the outliers.

```
In [ ]: mean_0 = []  
mean_1 = []  
for column in list(new_x_smotetomek_df.columns.values):  
    if column != "target":  
        mean_0.append(new_x_smotetomek_df[new_x_smotetomek_df.target == 0][column].mean())  
        mean_1.append(new_x_smotetomek_df[new_x_smotetomek_df.target == 1][column].mean())
```

```
In [ ]: median_0 = []  
median_1 = []  
for column in list(new_x_smotetomek_df.columns.values):  
    if column != "target":  
        median_0.append(new_x_smotetomek_df[new_x_smotetomek_df.target == 0][column].median())  
        median_1.append(new_x_smotetomek_df[new_x_smotetomek_df.target == 1][column].median())
```

10) Try and plot it's histogram for both the target values.

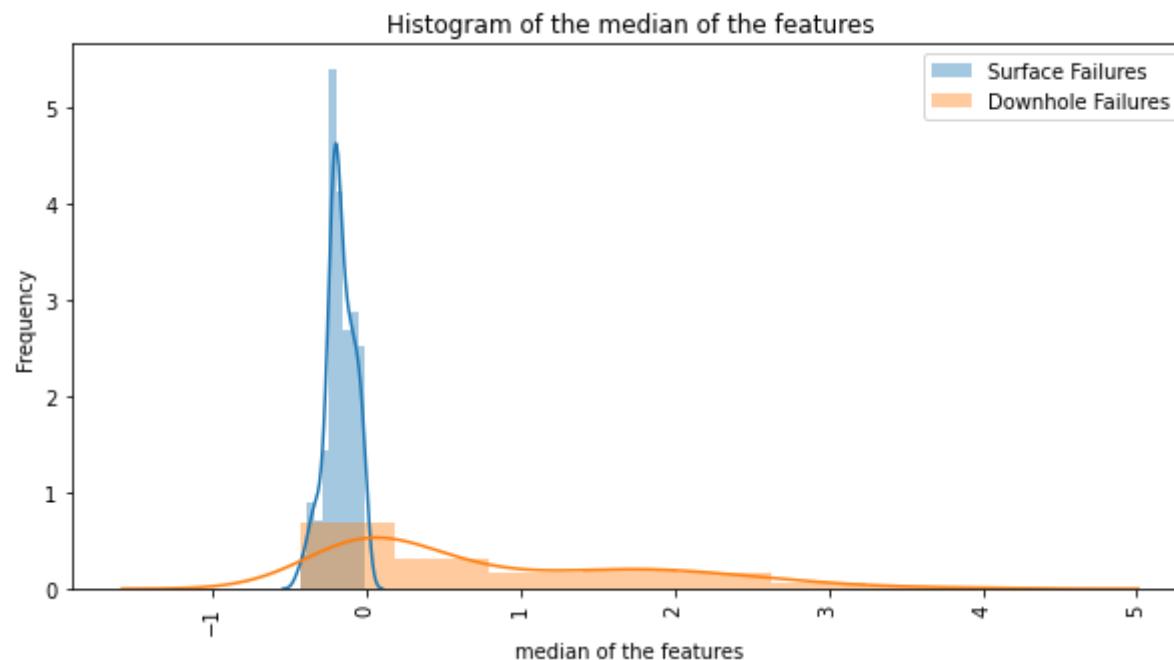
```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Histogram of the Mean of the features')
plt.ylabel('Frequency')
plt.xlabel('Mean of the features')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations :

Here for the surface and downhole failures, we can see that the different downhole features have more variance. Whereas the majority of the surface related features have a very similar value.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('Histogram of the median of the features')
plt.ylabel('Frequency')
plt.xlabel('median of the features')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```

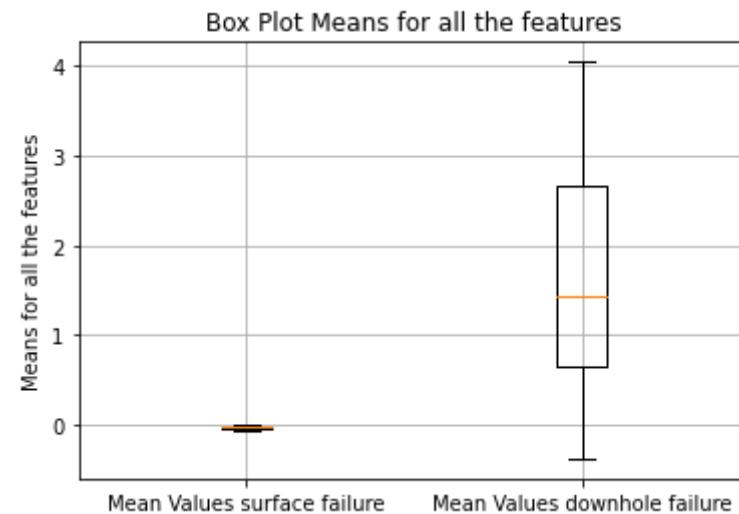


Observations :

Here for the surface and downhole failures, we can see that the different downhole features have more variance. Whereas the majority of the surface related features have a very similar value.

11) Try box plot / violin plot to visualize the outliers.

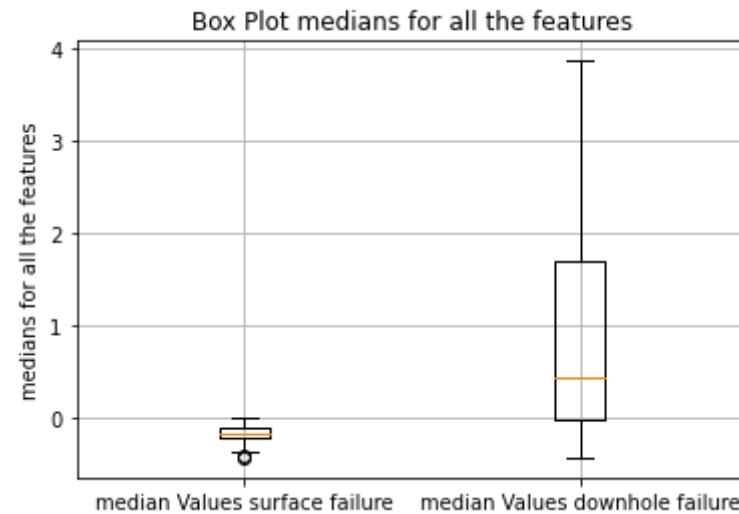
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Box Plot Means for all the features')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the features')  
plt.grid()  
plt.show()
```



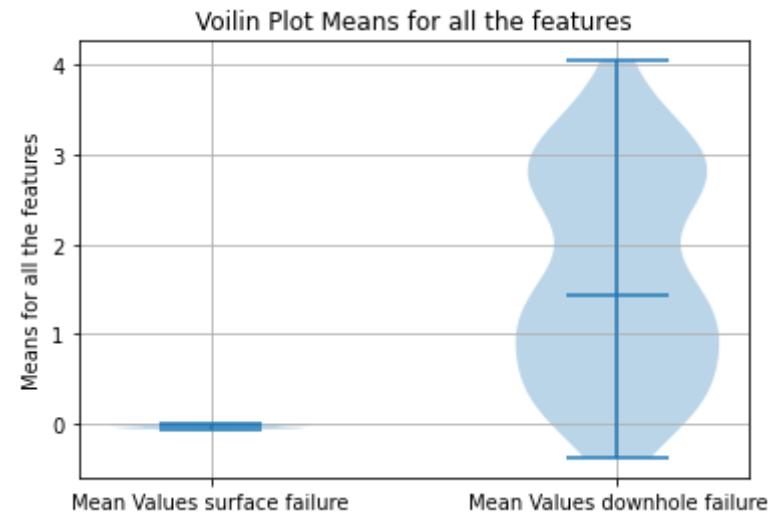
Observations :

The majority of the values for the surface failures lie near 0 whereas for the downhole failures, we can see all kinds of variation.

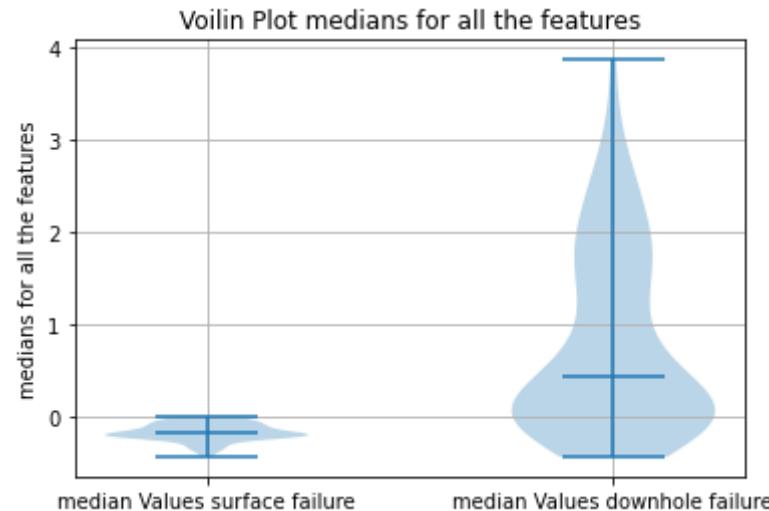
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('Box Plot medians for all the features')  
plt.xticks([1,2],('median Values surface failure','median Values downhole failure'))  
plt.ylabel('medians for all the features')  
plt.grid()  
plt.show()
```



```
In [ ]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Violin Plot Means for all the features')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the features')
plt.grid()
plt.show()
```



```
In [ ]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('Violin Plot medians for all the features')
plt.xticks([1,2],('median Values surface failure','median Values downhole failure'))
plt.ylabel('medians for all the features')
plt.grid()
plt.show()
```



PCA

```
In [81]: new_x_smotetomek_df.drop("target",axis=1,inplace=True)
```

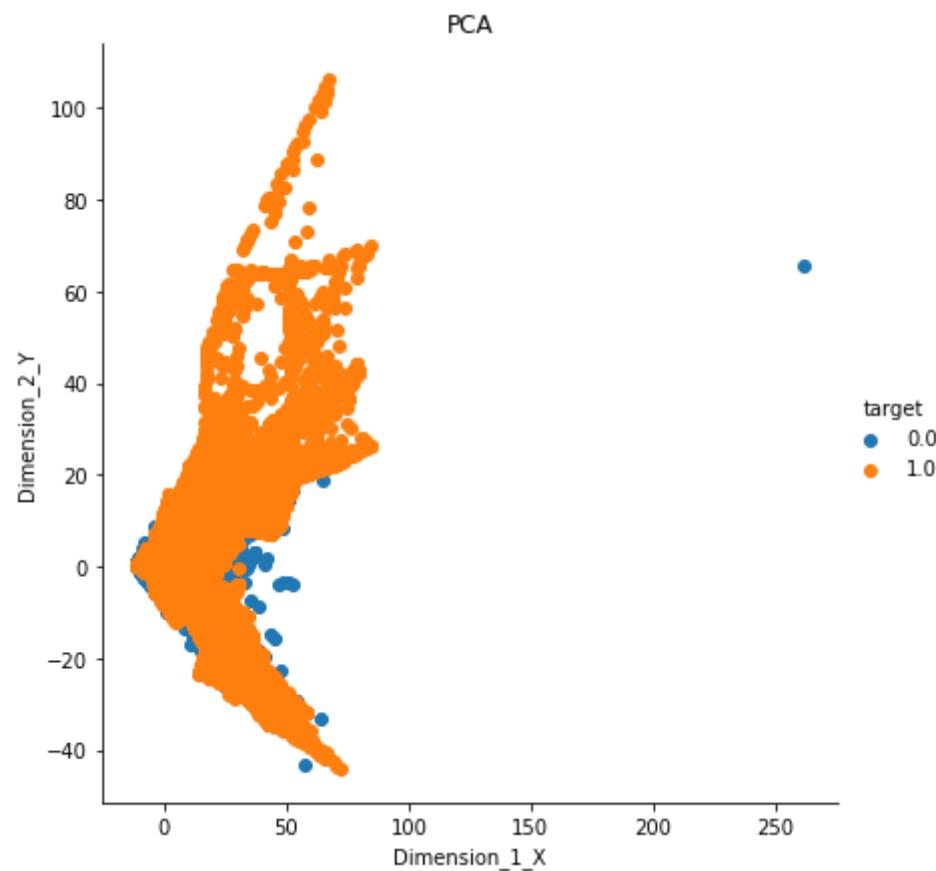
```
In [82]: "target" in new_x_smotetomek_df.columns
```

```
Out[82]: False
```

In [83]:

```
%%time  
run_pca(new_x_smotetomek_df,y_smotetomek)
```

```
[[-10.234793 -0.3979547  0.        ]  
 [-6.767308  -0.92511797  0.        ]  
 [-11.480434   0.30759192  0.        ]  
 ...  
 [-1.288987  -0.20813158  1.        ]  
 [ 14.657278   4.2855315   1.        ]  
 [ 17.489021  -14.223328   1.        ]]
```



CPU times: user 789 ms, sys: 118 ms, total: 907 ms
Wall time: 785 ms

Observation :

- 1) Observe that PCA is not able to differentiate properly the surface failures from the downhole failures.
- 2) Not all the the surface and downhole features are coming under the same category.

Perform a similar EDA on the outliers.

Here we take the outliers from the features that have been selected.

For each sensor we find the outliers that also classwise.

The issue is that there maybe high outliers here , but there maybe lower outliers here as well.

<https://www.geeksforgeeks.org/interquartile-range-and-quartile-deviation-using-numpy-and-scipy/> (<https://www.geeksforgeeks.org/interquartile-range-and-quartile-deviation-using-numpy-and-scipy/>)

Higher outlier = more than $1.5 \cdot \text{IQR}$ above the third quartile.

Lower outlier = less than $1.5 \cdot \text{IQR}$ below the first quartile.

The IQR is the difference between Q3 and Q1.

```
In [ ]: new_x_smotetomek_df["target"] = y_smotetomek
```

```
In [ ]: counter=0
main_dictionary=dict()
outliers=[]
list_of_outliers_features_classes=[]
for i in range(len(column_list)):
    d=dict()
    new_list=new_x_smotetomek_df[new_x_smotetomek_df.target==one_zero[i]]\
    [column_list[i]]
    new_index=list(new_list.reset_index()["index"])
    QR1=np.percentile(new_list, 25)
    QR3=np.percentile(new_list, 75)
    IQR=QR3-QR1
    for index,value in zip(new_index,new_list):
        if (value>(QR3+(IQR*1.5))) or (value<(QR1-(IQR*1.5))):
            counter=counter+1
            d.update({index:value})
            if index not in main_dictionary:
                main_dictionary.update({index:list(new_x_smotetomek_df.loc[index])})
    list_of_outliers_features_classes.append(d)
del d
outliers.append(counter)
counter=0
```

```
In [ ]: print("Total number of data points that are outliers in atleast one or more than one feature : ",len(np.unique
```

```
Total number of data points that are outliers in atleast one or more than one feature : 79724
```

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:outliers[i] for i in index}

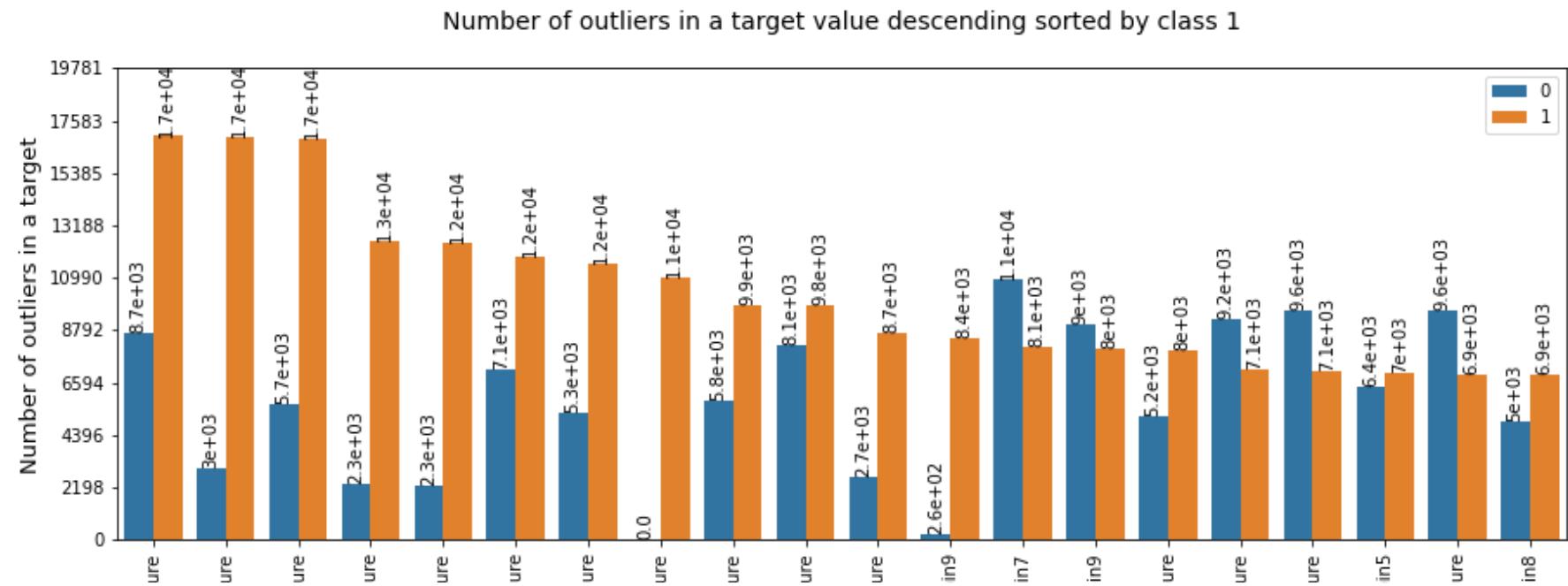
d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])

counter=0
new_column_list=[]
new_outliers_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_outliers_list.append(outliers[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_outliers_list.append(outliers[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 117 sensors.

In []: **try:**

```
    plot_diagram(new_column_list,new_outliers_list,new_one_zero,"Number of outliers",1)
except ValueError:
    pass
```



Observation :

Number of outliers are present for both the surface failures as well as the downhole failure.

The number of outliers for downhole failures have been increased.

For every column feature, get the mean for the outliers for both class 0 and class 1.

```
In [ ]: mean_0=[]
mean_1=[]
for i in range(0,len(list_of_outliers_features_classes),2):
    class_0=[ele for ele in list_of_outliers_features_classes[i].values()]
    class_1=[ele for ele in list_of_outliers_features_classes[i+1].values()]
    if (len(class_0)==0) :
        mean_0.append(0.01)
    else :
        mean_0.append(np.mean(class_0))

    if (len(class_1)==0):
        mean_1.append(0.01)
    else :
        mean_1.append(np.mean(class_1))
```

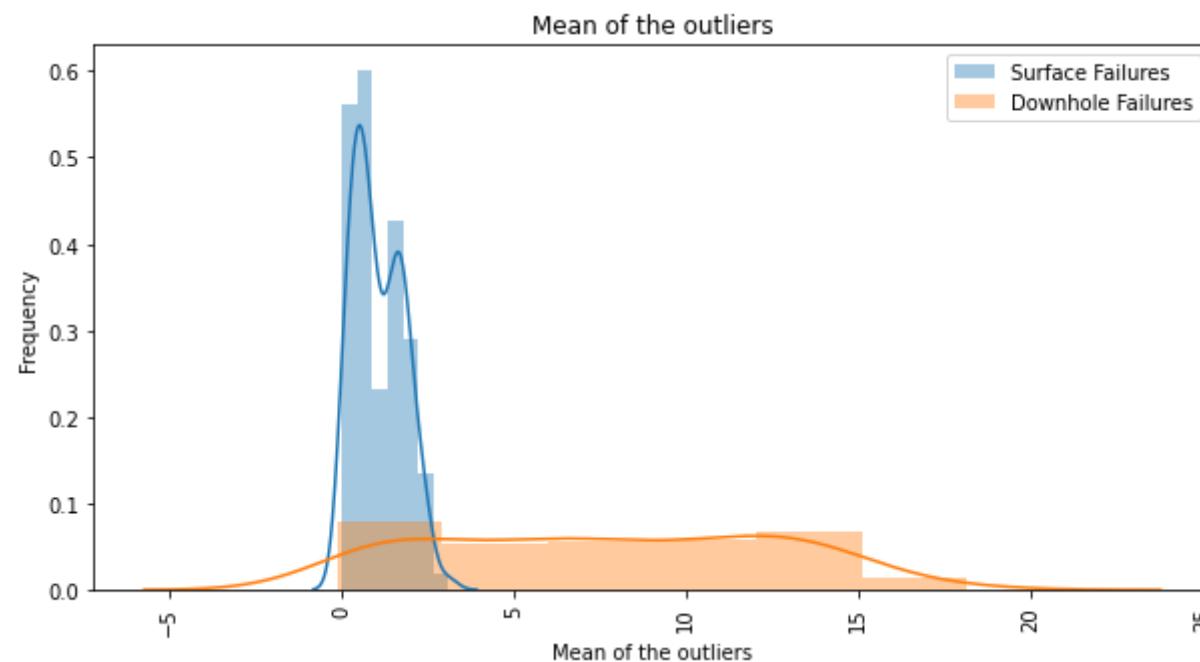
```
In [ ]: median_0=[]
median_1=[]
for i in range(0,len(list_of_outliers_features_classes),2):
    class_0=[ele for ele in list_of_outliers_features_classes[i].values()]
    class_1=[ele for ele in list_of_outliers_features_classes[i+1].values()]
    if (len(class_0)==0) :
        median_0.append(0.01)
    else :
        median_0.append(np.median(class_0))

    if (len(class_1)==0):
        median_1.append(0.01)
    else :
        median_1.append(np.median(class_1))
```

Plotting the mean and median for each feature class wise.

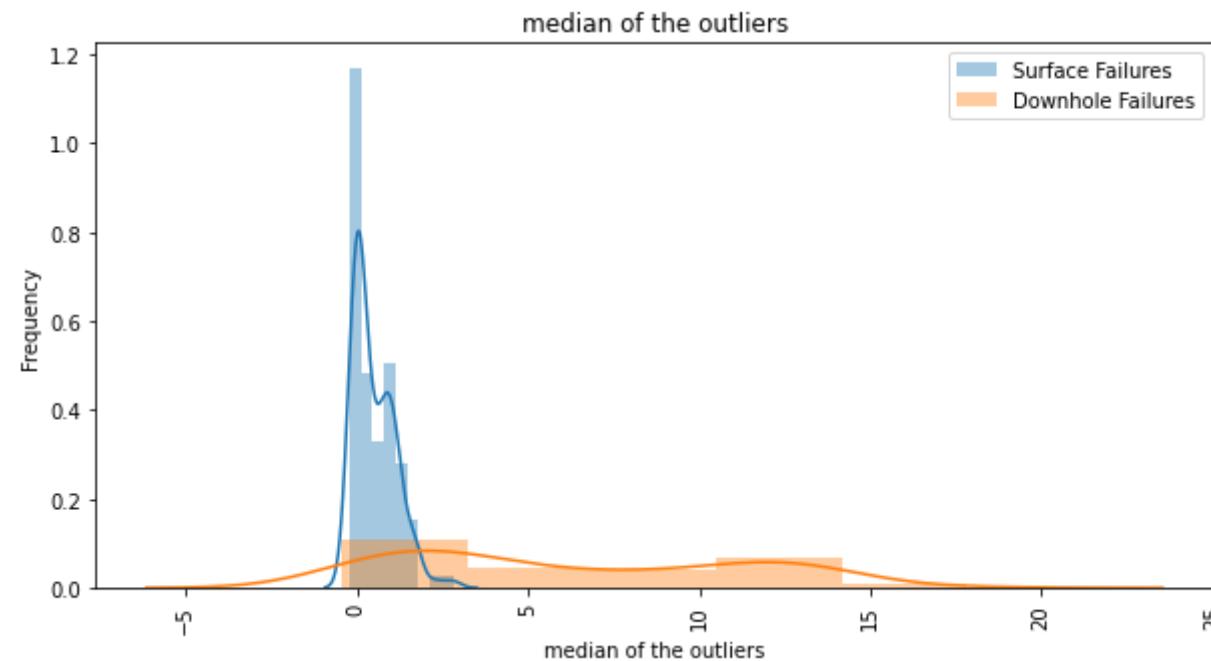
10) Try and plot it's histogram for both the target values.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Mean of the outliers')
plt.ylabel('Frequency')
plt.xlabel('Mean of the outliers')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations : For the outliers, majority of them seem to be concentrated around 0 for the surface outliers with variance as well. For downhole outliers they have lesser variance but have more density in the other range values

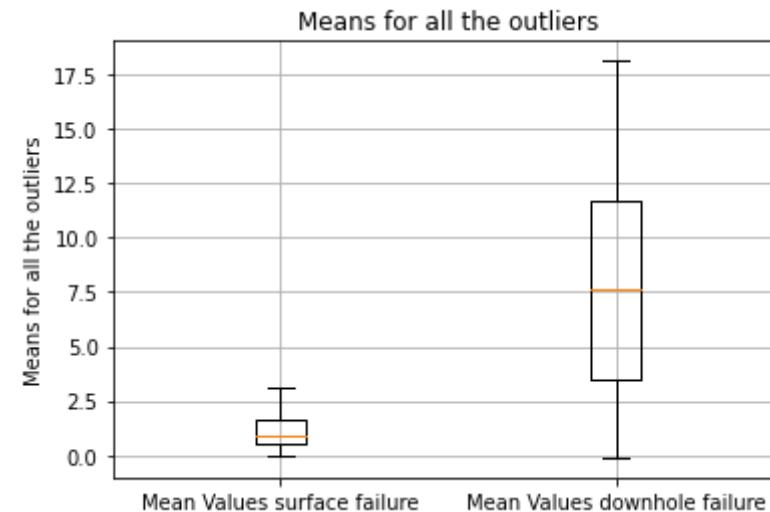
```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('median of the outliers')
plt.ylabel('Frequency')
plt.xlabel('median of the outliers')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



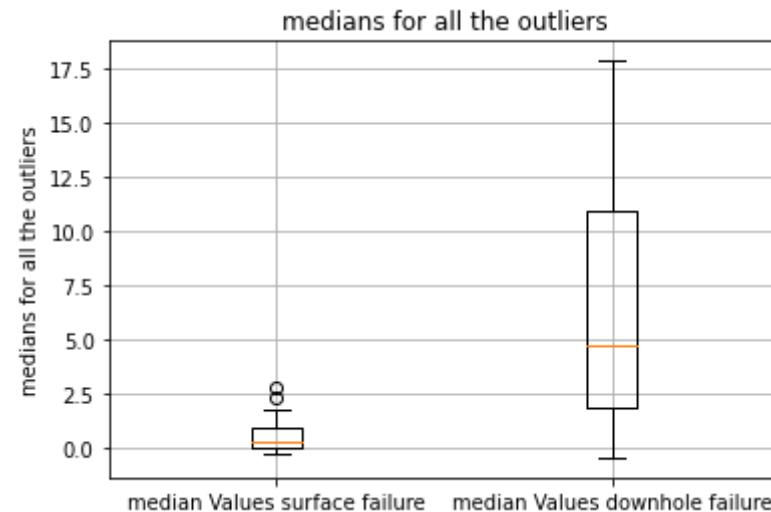
Observations : For the outliers, majority of them seem to be concentrated around 0 for the surface outliers with variance as well. For downhole outliers they have lesser variance but have more density in the other range values

11) Try box plot / violin plot to visualize the outliers.

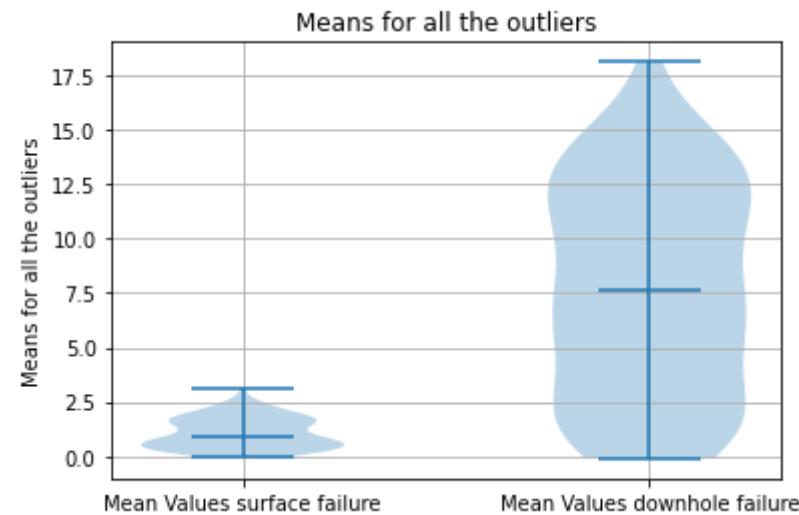
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Means for all the outliers')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the outliers')  
plt.grid()  
plt.show()
```



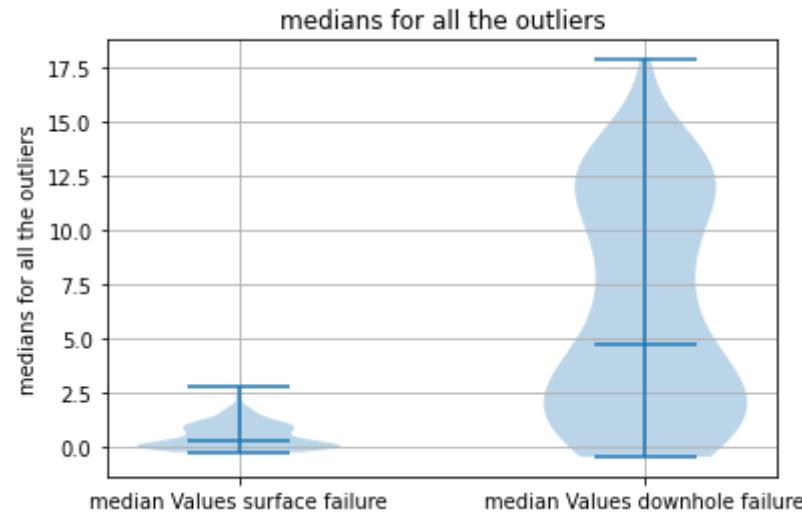
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('medians for all the outliers')  
plt.xticks([1,2],('median Values surface failure','median Values downhole failure'))  
plt.ylabel('medians for all the outliers')  
plt.grid()  
plt.show()
```



```
In [ ]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Means for all the outliers')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the outliers')
plt.grid()
plt.show()
```



```
In [ ]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('medians for all the outliers')
plt.xticks([1,2],('median Values surface failure','median Values downhole failure'))
plt.ylabel('medians for all the outliers')
plt.grid()
plt.show()
```



The outliers although for the surface related failures are higher. But both the means and the medians of the outliers are similar.

For the given data points and features enter new features that tell if these values were previously np.NaN or not.

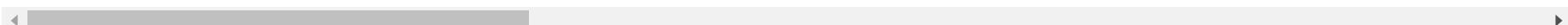
```
In [ ]: new_x_smotetomek_df.drop("target",axis=1,inplace=True)
```

```
In [ ]: nan_x_smotetomek_df=fuse_dataframe_nan(new_x_smotetomek_df,nan_or_not, set(new_column_list))
nan_x_smotetomek_df[ "target" ]=y_smotetomek
nan_x_smotetomek_df.head()
```

Out[573]:

	sensor7_histogram_bin2	sensor105_histogram_bin3	sensor69_histogram_bin7	sensor64_histogram_bin2	sensor40_measure	sensor26_histogram
0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	1.0

5 rows × 118 columns



Now this is the problem out of the given data that we have , we select the features that have more than 50% np.NaNs

```
In [ ]: counter=0
for column in new_x_smotetomek_df.columns.values:
    try:
        percent_of_nan=nan_x_smotetomek_df[column].value_counts()[1]/\
                      (nan_x_smotetomek_df[column].value_counts()[0]+\
                       nan_x_smotetomek_df[column].value_counts()[1])
        if (percent_of_nan>.5) :
            counter=counter+1
            print(column," : ",percent_of_nan)
    except KeyError:
        pass
print("Overall : ",counter," features with % of na's more than 50%")
```

```
sensor38_measure      :  0.6605416666666667
sensor39_measure      :  0.7348958333333333
sensor40_measure      :  0.77375
sensor41_measure      :  0.7967916666666667
sensor42_measure      :  0.8133125
sensor43_measure      :  0.822
Overall :  6  features with % of na's more than 50%
```

6) Impute np.NaN with median and then perform OneSidedSelection undersampling on the values.

```
In [18]: data_type="oss_data"
```

```
In [ ]: train_oss_all_features=train.copy()
y_train_oss_all_features=y_train.copy()
test_oss_all_features=test.copy()
y_test_oss_all_features=y_test.copy()
```

```
In [ ]: train_oss_all_features.shape
```

```
Out[576]: (48000, 170)
```

```
In [ ]: y_train_oss_all_features.shape
```

```
Out[577]: (48000,)
```

```
In [ ]: test_oss_all_features.shape
```

```
Out[578]: (12000, 170)
```

```
In [ ]: y_test_oss_all_features.shape
```

```
Out[579]: (12000,)
```

```
In [ ]: for i,j in zip(train_oss_all_features.columns.values,test_oss_all_features.columns.values):
         if (i!=j):
             print(i, " : ",j)
```

```
In [ ]: train_oss_all_features,y_train_oss_all_features=reset_index(train_oss_all_features,y_train_oss_all_features)
        test_oss_all_features,y_test_oss_all_features=reset_index(test_oss_all_features,y_test_oss_all_features)
```

Filling the np.NaN values with the median of the category this time.

```
In [ ]: l=[]
         for ele in train_oss_all_features["sensor4_measure"]:
             if np.isnan(ele):
                 pass
             else:
                 l.append(ele)
         np.median(l)
```

```
Out[583]: 126.0
```

```
In [ ]: train_oss_all_features["sensor4_measure"].median()
```

```
Out[584]: 126.0
```

Both are the same so the pandas.series.median() function ignores the value that has the np.NaN and does not even count it.

Divide the whole dataframe into the number of categories that we have and then only enter the median for that feature for that particular category.

```
In [ ]: train_oss_all_features,y_train_oss_all_features,test_oss_all_features,y_test_oss_all_features=\
impute_median_for_train_n_test_as_per_train(train_oss_all_features,
                                              y_train_oss_all_features,
                                              test_oss_all_features,
                                              y_test_oss_all_features,data_type)
```

```
In [ ]: train_oss_all_features,y_train_oss_all_features=reset_index(train_oss_all_features,y_train_oss_all_features)
test_oss_all_features,y_test_oss_all_features=reset_index(test_oss_all_features,y_test_oss_all_features)
```

Standardise the dataframe

```
In [ ]: standard_train_oss_all_features_df,standard_test_oss_all_features_df=standardize_test_n_train_wrt_train(trai
```

```
In [ ]: standard_train_oss_all_features_df.shape
```

```
Out[590]: (48000, 170)
```

```
In [ ]: standard_test_oss_all_features_df.shape
```

```
Out[593]: (12000, 170)
```

```
In [ ]: y_train_oss_all_features.head()
```

```
Out[594]: 0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: target, dtype: float32
```

```
In [ ]: y_train_oss_all_features.tail()
```

```
Out[595]: 47995    1.0
47996    1.0
47997    1.0
47998    1.0
47999    1.0
Name: target, dtype: float32
```

```
In [ ]: "target" in standard_train_oss_all_features_df.columns
```

```
Out[596]: False
```

Perform oss undersampling

```
In [ ]: %%time
from imblearn.under_sampling import OneSidedSelection
x_oss, y_oss = OneSidedSelection(random_state=42, sampling_strategy='majority', n_jobs=-1).fit_resample(standard_train_oss_all_features_df, y_train_oss_all_features)
print("done")
```

```
done
CPU times: user 8min 29s, sys: 75.1 ms, total: 8min 30s
Wall time: 8min 8s
```

```
In [ ]: x_oss.shape
```

```
Out[602]: (32461, 170)
```

```
In [ ]: y_oss.value_counts()
```

```
Out[603]: 0.0    31661
1.0     800
Name: target, dtype: int64
```

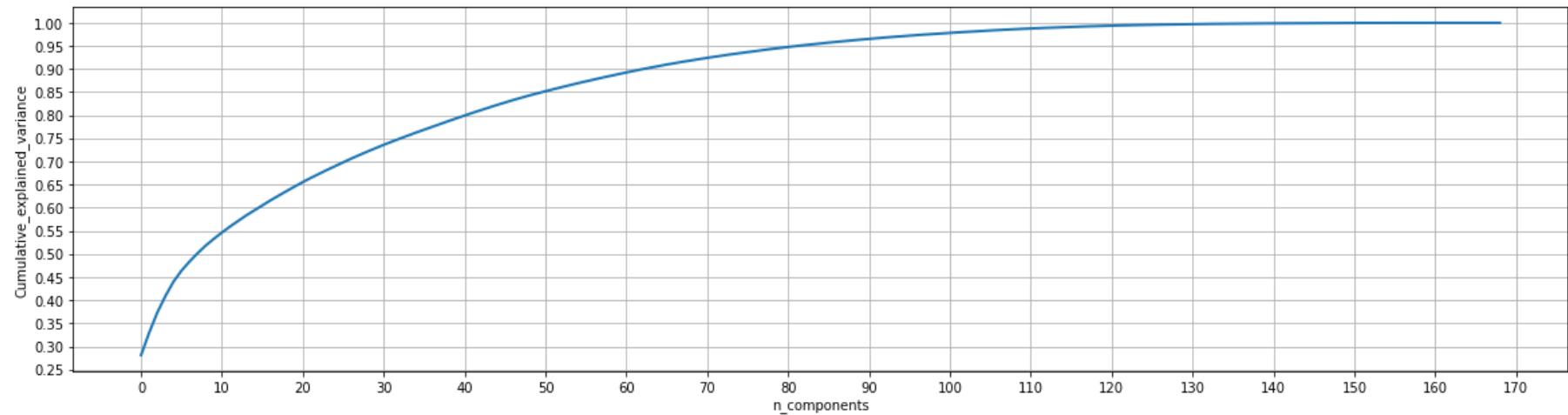
```
In [ ]: y_oss.shape
```

```
Out[604]: (32461,)
```

Truncated SVD to get the best features

```
In [ ]: #https://stackoverflow.com/questions/50796024/feature-variable-importance-after-a-pca-analysis
```

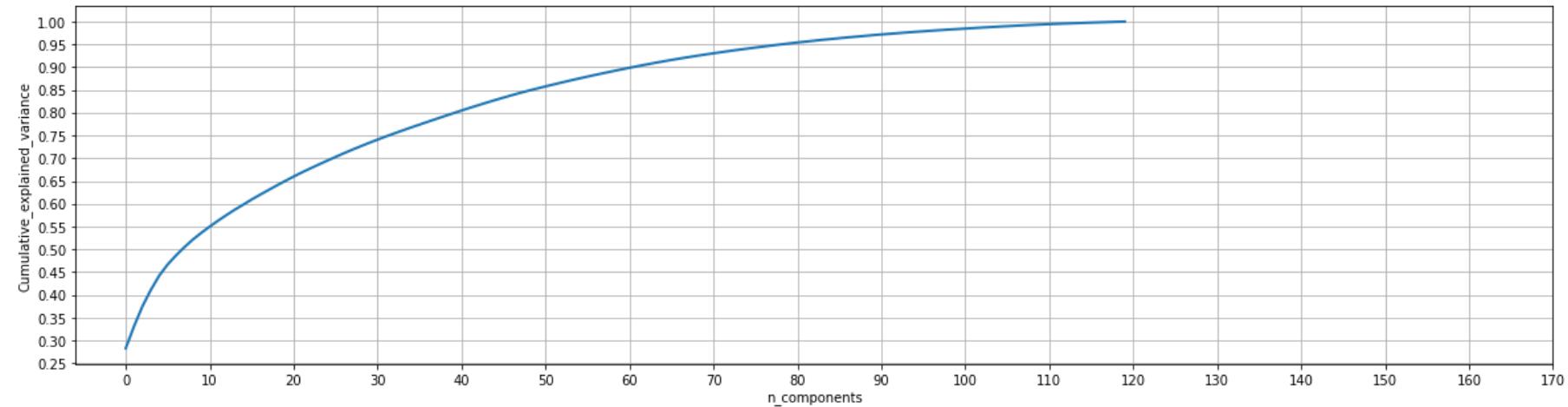
```
In [ ]: truncated_svd(x_oss,169)
```



Out[606]: TruncatedSVD(n_components=169)

I choose 120 components to describe my data as 120 components explain atleast 99% variance of the total data.

```
In [ ]: svd=truncated_svd(x_oss,120)
```



For the 120 principal components using svd.explained_variance_ratio_ find the variance distribution.

```
In [ ]: print(len(svd.explained_variance_ratio_))
```

120

```
In [ ]: np.sum(svd.explained_variance_ratio_)
```

Out[609]: 0.99351096

```
In [ ]: svd.components_.shape
```

Out[610]: (120, 170)

```
In [ ]: #number of features  
svd.components_.shape[1]
```

Out[611]: 170

```
In [ ]: #number of components  
svd.components_.shape[0]
```

Out[612]: 120

For the 120 principal components find the feature importance.

```
In [ ]: set_number_of_features=120
```

Trying Forward Feature selection

Here, in forward feature selection, out of 170 features even if I choose to select 10 features only , still it will take a lot of time.

Therefore I go with recursive feature elimination.

Using Recursive Feature Elimination, I can get the features that I need.

```
In [ ]: x_oss.shape
```

Out[614]: (32461, 170)

```
In [ ]: new x oss df.head()
```

Out[616]:

	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor6_measure	sensor7_histogram_bin1	sensor7_histogram_bin2
0	-0.411826	-0.09191	-0.432133	-0.110448	-0.056429	-0.028788	-0.057799
1	-0.209185	-0.09191	-0.432133	-0.078741	-0.056429	-0.028788	-0.057799
2	0.226385	-0.09191	2.314107	0.276872	-0.056429	-0.028788	-0.057799
3	0.027280	-0.09191	-0.432131	0.303606	-0.056429	-0.028788	-0.057799
4	-0.365136	-0.09191	2.314107	-0.081228	-0.056429	-0.028788	-0.057799

5 rows × 155 columns

Try spearman corelation coefficient to find out co relation. (It is good for cause and affect analysis but it might not be of much use when we find that our features are dependent on other features, we expect our features to be independent of each other.).

In []:

```
%%time
spearman=select_spearman(new_x_oss_df,int(set_number_of_features+np.floor((len(train.columns.values)-set_num
If I take the data having correlation between only -.1 and .1 range then I get 0 features.
If I take the data having correlation between only -.90 and .90 range then I get 89 features.
If I take the data having correlation between only -.95 and .95 range then I get 107 features.
If I take the data having correlation between only -.97 and .97 range then I get 128 features.
If I take the data having correlation between only -.98 and .98 range then I get 135 features.
If I take the data having correlation between only -.99 and .99 range then I get 139 features.
If I take the data having correlation between only -.999 and .999 range then I get 148 features.
CPU times: user 13.1 s, sys: 48.4 ms, total: 13.1 s
Wall time: 13.1 s
```

In []:

```
new_x_oss_df=perform_spearman(new_x_oss_df,spearman,.97)
```

In []:

```
new_x_oss_df.head()
```

Out[619]:

	sensor2_measure	sensor3_measure	sensor4_measure	sensor6_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram
0	-0.09191	-0.432133	-0.110448	-0.056429	-0.028788	-0.057799	-0.11
1	-0.09191	-0.432133	-0.078741	-0.056429	-0.028788	-0.057799	-0.11
2	-0.09191	2.314107	0.276872	-0.056429	-0.028788	-0.057799	-0.11
3	-0.09191	-0.432131	0.303606	-0.056429	-0.028788	-0.057799	-0.11
4	-0.09191	2.314107	-0.081228	-0.056429	-0.028788	-0.057799	-0.11

5 rows × 128 columns

In []:

```
new_x_oss_df.shape
```

Out[620]:

```
(32461, 128)
```

```
In [ ]: y_oss.shape
```

```
Out[621]: (32461,)
```

```
In [ ]: new_standard_test_oss_all_features_df=standard_test_oss_all_features_df[new_x_oss_df.columns.values]
new_standard_test_oss_all_features_df.head()
```

```
Out[622]:
```

	sensor2_measure	sensor3_measure	sensor4_measure	sensor6_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram
0	-0.09191	-0.432133	-0.078741	-0.056429	-0.028788	-0.057799	-0.11
1	-0.09191	2.314107	0.135746	-0.056429	-0.028788	-0.057799	-0.11
2	-0.09191	-0.432133	-0.115422	-0.056429	-0.028788	-0.057799	-0.11
3	-0.09191	2.314107	-0.114800	-0.056429	-0.028788	-0.057799	-0.11
4	-0.09191	2.314107	0.041869	-0.056429	-0.028788	-0.057799	-0.11

5 rows × 128 columns

```
In [ ]: #new_x_oss_df.to_pickle("new_x_oss_df.pickle")
#y_oss.to_pickle("y_oss.pickle")
```

```
#new_standard_test_oss_all_features_df.to_pickle("new_standard_test_oss_all_features_df.pickle")
#y_test_oss_all_features.to_pickle("y_test_oss_all_features.pickle")
```

```
In [84]: #new_x_oss_df=pd.read_pickle("/content/gdrive/My Drive/new_x_oss_df.pickle")
#y_oss=pd.read_pickle("/content/gdrive/My Drive/y_oss.pickle")
```

```
#new_standard_test_oss_all_features_df=pd.read_pickle("/content/gdrive/My Drive/new_standard_test_oss_all_fe
#y_test_oss_all_features=pd.read_pickle("/content/gdrive/My Drive/y_test_oss_all_features.pickle")
```

```
In [ ]: np.save('new_x_oss_df_columns.npy',new_x_oss_df.columns.values)
```

```
In [85]: new_x_oss_df["target"] = y_oss
new_x_oss_df.head()
```

Out[85]:

	sensor2_measure	sensor3_measure	sensor4_measure	sensor6_measure	sensor7_histogram_bin1	sensor7_histogram_bin2	sensor7_histogram_
0	-0.09191	-0.432133	-0.110448	-0.056429	-0.028788	-0.057799	-0.11
1	-0.09191	-0.432133	-0.078741	-0.056429	-0.028788	-0.057799	-0.11
2	-0.09191	2.314107	0.276872	-0.056429	-0.028788	-0.057799	-0.11
3	-0.09191	-0.432131	0.303606	-0.056429	-0.028788	-0.057799	-0.11
4	-0.09191	2.314107	-0.081228	-0.056429	-0.028788	-0.057799	-0.11

5 rows × 129 columns

Compare the target 0 and target 1 mean values feature wise after median value imputing

```
In [ ]: column_list=[]
mean_list=[]
one_zero=[]
for column in list(new_x_oss_df.columns.values):
    if column!="target":
        column_list.append(column)
        one_zero.append(0)
        mean_list.append(new_x_oss_df[column][new_x_oss_df.target==0].mean())

        column_list.append(column)
        one_zero.append(1)
        mean_list.append(new_x_oss_df[column][new_x_oss_df.target==1].mean())
```

Find the features where the difference between the means of the two classes is more than .15
descending sorted by class 1

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:mean_list[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:-len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

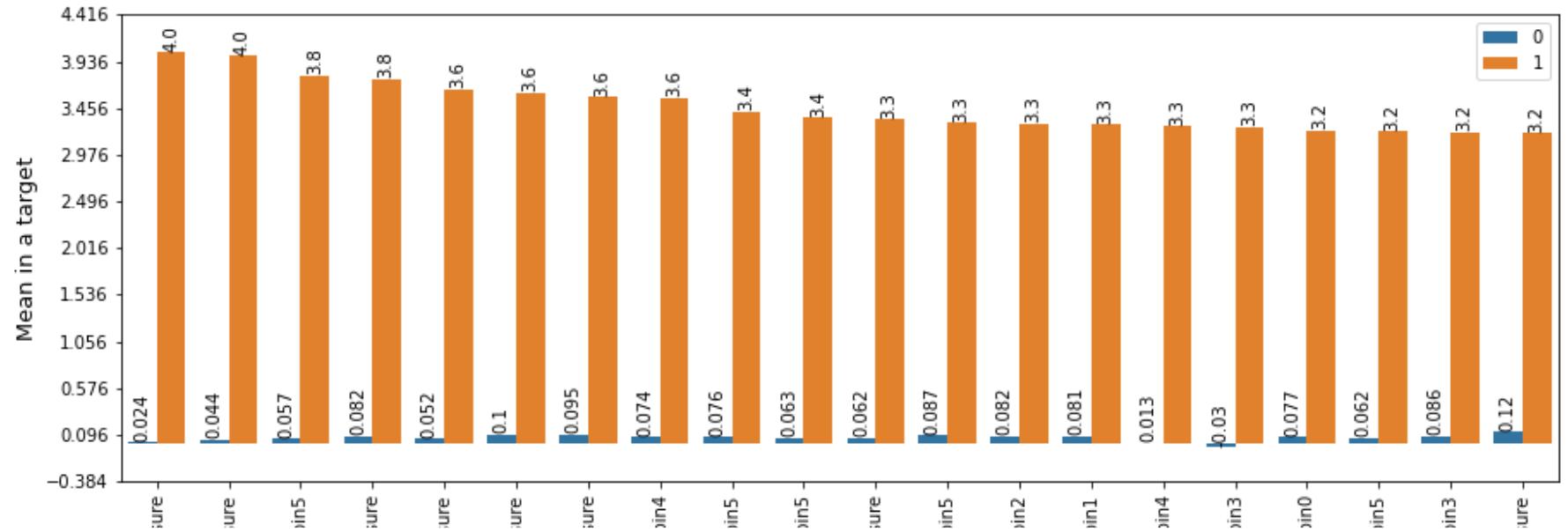
```
In [ ]: counter=0
new_column_list=[]
new_mean_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_mean_list.append(mean_list[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_mean_list.append(mean_list[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 128 sensors.

In []:

```
try:
    plot_diagram(new_column_list,new_mean_list,new_one_zero,"Mean",1)
except ValueError:
    pass
```

Mean in a target value descending sorted by class 1



Observation : All the mean values that we have taken from every median imputed oss feature, Observe that there is a significant difference between the means of the surface failures and the means of the downhole failures. Out of the selected features observe that the most of the features have a larger mean value for the downhole failure rather than that of the surface failures

```
In [ ]: counter=0
mean_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.

    if (np.abs(np.abs(mean_list[i])-np.abs(mean_list[i-1]))>=.15):
        mean_higher_reading.append(mean_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

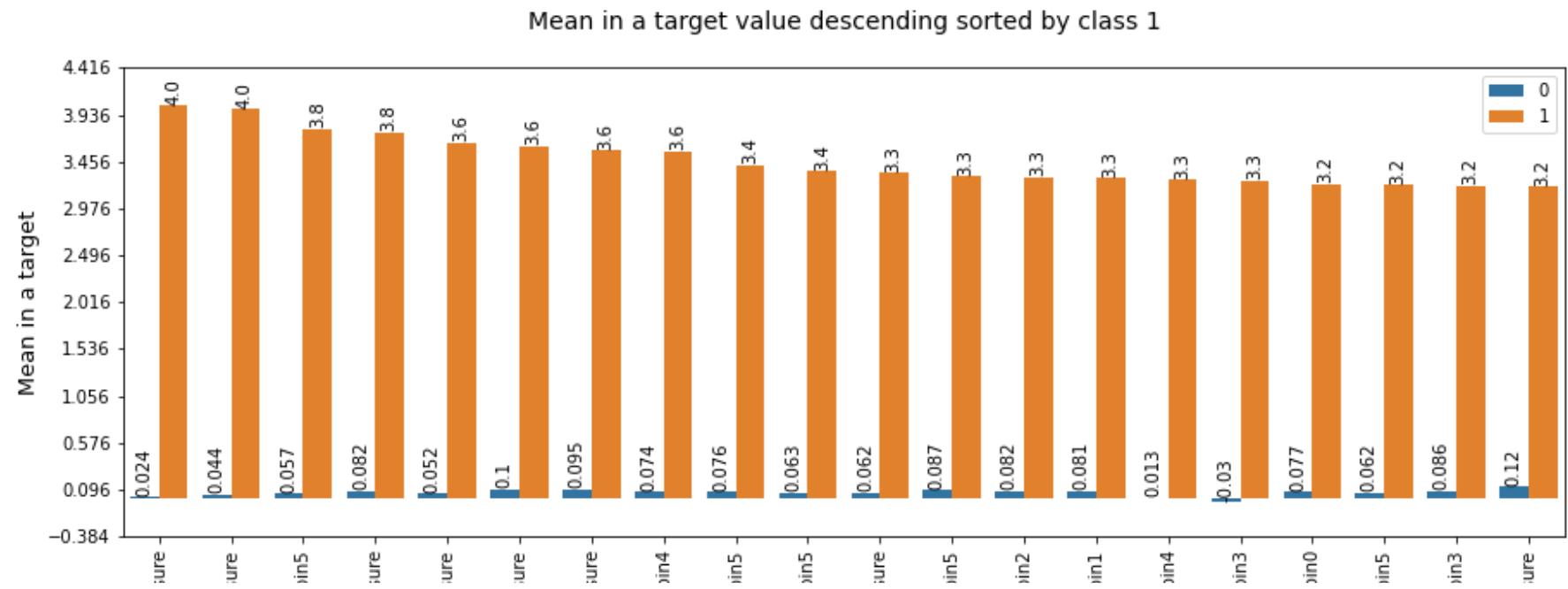
        mean_higher_reading.append(mean_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between Mean readings of su
```

So we have approximate 117 sensor that show significant difference (> .15) between Mean readings of surface and downhole failures.

In []: **try:**

```
    plot_diagram(column_higher_reading,mean_higher_reading,one_zero_higher_reading,"Mean",1)
except ValueError:
    pass
```



Compare the target 0 and target 1 Median values feature wise after Median value imputing

```
In [ ]: column_list=[]
median_list=[]
one_zero=[]

for column in list(new_x_oss_df.columns.values):
    if column!="target":
        column_list.append(column)
        one_zero.append(0)
        median_list.append(new_x_oss_df[column][new_x_oss_df.target==0].median())
        column_list.append(column)
        one_zero.append(1)
        median_list.append(new_x_oss_df[column][new_x_oss_df.target==1].median())
```

**Find the features where the difference between the medians of the two classes is more than .15
descending sorted by class 1**

```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:median_list[i] for i in index}
```

```
In [ ]: d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])
```

```
In [ ]: counter=0
new_column_list=[]
new_median_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_median_list.append(median_list[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_median_list.append(median_list[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 128 sensors.

```
In [ ]: try:
    plot_diagram(new_column_list,new_median_list,new_one_zero,"Medians",1)
except ValueError:
    pass
```

```
In [ ]: counter=0
median_higher_reading=[]
column_higher_reading=[]
one_zero_higher_reading=[]
for i in d_keys:
    # I am considering this difference should be .15 It can change.

    if (np.abs(np.abs(median_list[i])-np.abs(median_list[i-1]))>=.15):
        median_higher_reading.append(median_list[i-1])
        column_higher_reading.append(column_list[i-1])
        one_zero_higher_reading.append(one_zero[i-1])

        median_higher_reading.append(median_list[i])
        column_higher_reading.append(column_list[i])
        one_zero_higher_reading.append(one_zero[i])

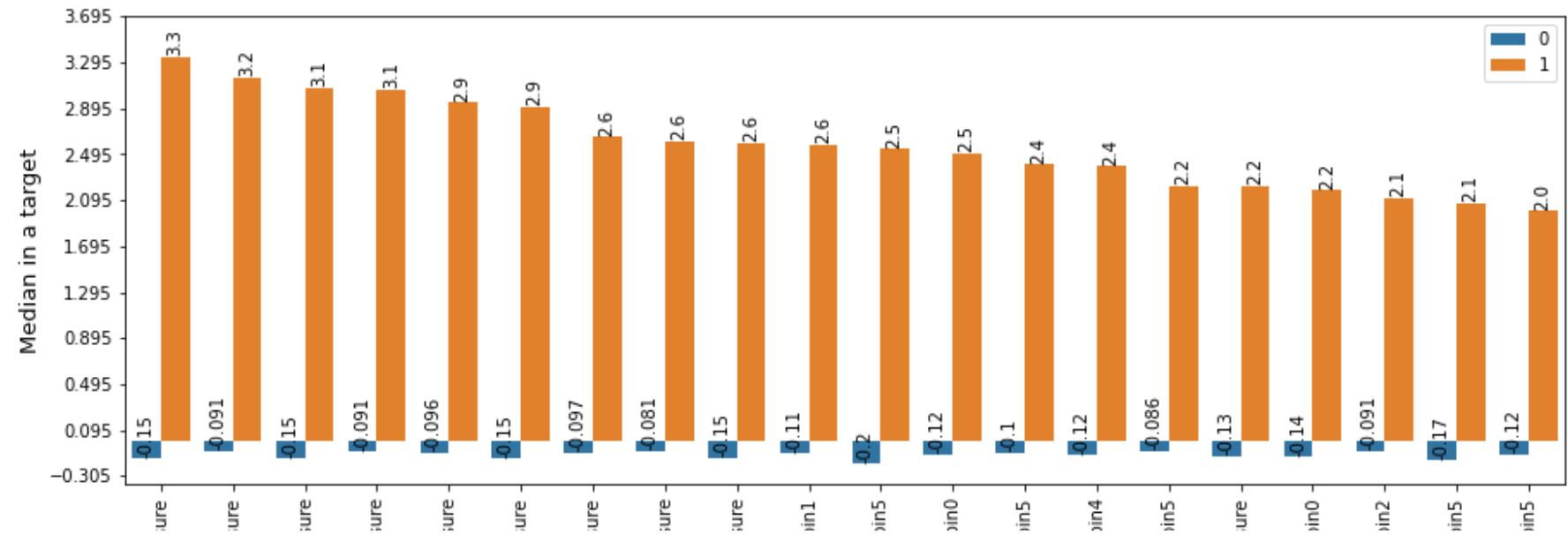
    counter=counter+1
print("So we have approximate {} sensor that show significant difference (> .15) between median readings of
```

So we have approximate 67 sensor that show significant difference (> .15) between median readings of surface and downhole failures.

In []: **try:**

```
plot_diagram(column_higher_reading,median_higher_reading,one_zero_higher_reading,"Median",1)
except ValueError:
    pass
```

Median in a target value descending sorted by class 1



Observation : All the median values that we have taken from every median imputed oss feature, decrease in value than that of the mean. Observe that there is a significant difference between the medians of the surface failures and the medians of the downhole failures. Out of the selected features observe that the most of the features have a larger median value for the downhole failure rather than that of the surface failures

Basic Classifier

In []: `new_x_oss_df.drop("target",axis=1,inplace=True)`

```
In [ ]: print(dummy_classifier_based_on_median(new_x_oss_df.loc[20000],  
                                              column_list,  
                                              median_list))
```

Predicted surface failure.
[111, 17]

```
In [ ]: print(dummy_classifier_based_on_median(new_standard_test_oss_all_features_df.loc[7997],  
                                              column_list,  
                                              median_list))
```

Predicted surface failure.
[116, 12]

```
In [ ]: new_x_oss_df["target"] = y_oss
```

EDA for the dataset

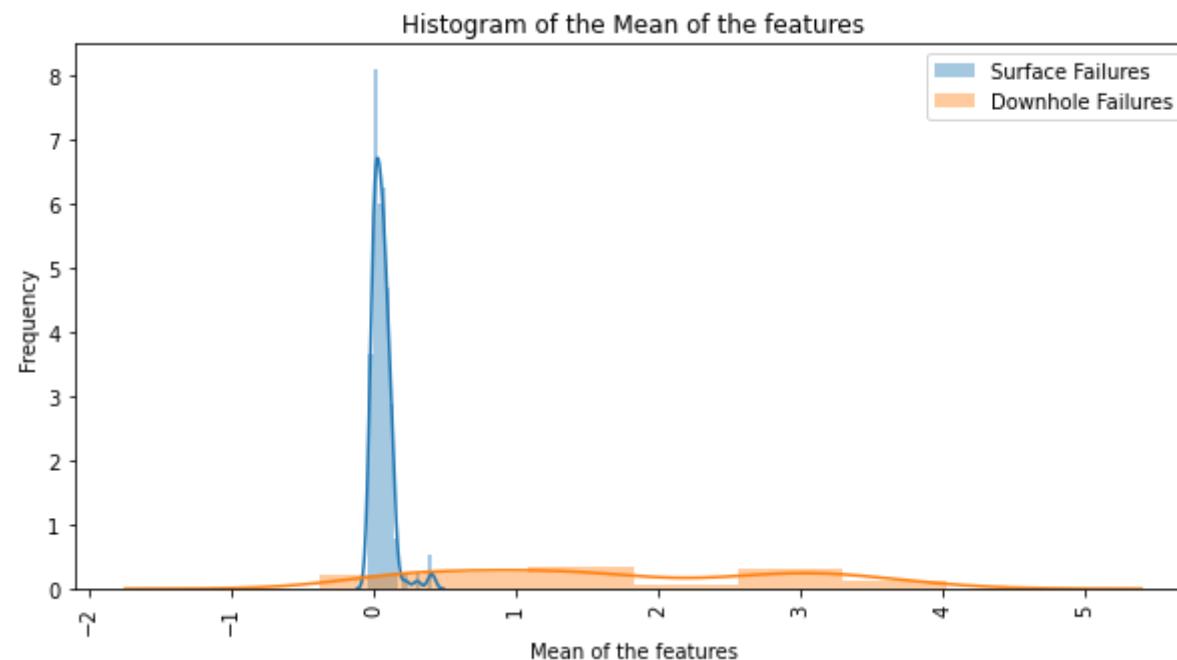
Using histograms, plots find out the nature of means for features for both the classes and thereby, the outliers.

```
In [ ]: mean_0=[]  
mean_1=[]  
for column in list(new_x_oss_df.columns.values):  
    if column!="target":  
        mean_0.append(new_x_oss_df[new_x_oss_df.target==0][column].mean())  
        mean_1.append(new_x_oss_df[new_x_oss_df.target==1][column].mean())
```

```
In [ ]: median_0=[]  
median_1=[]  
for column in list(new_x_oss_df.columns.values):  
    if column!="target":  
        median_0.append(new_x_oss_df[new_x_oss_df.target==0][column].median())  
        median_1.append(new_x_oss_df[new_x_oss_df.target==1][column].median())
```

10) Try and plot it's histogram for both the target values.

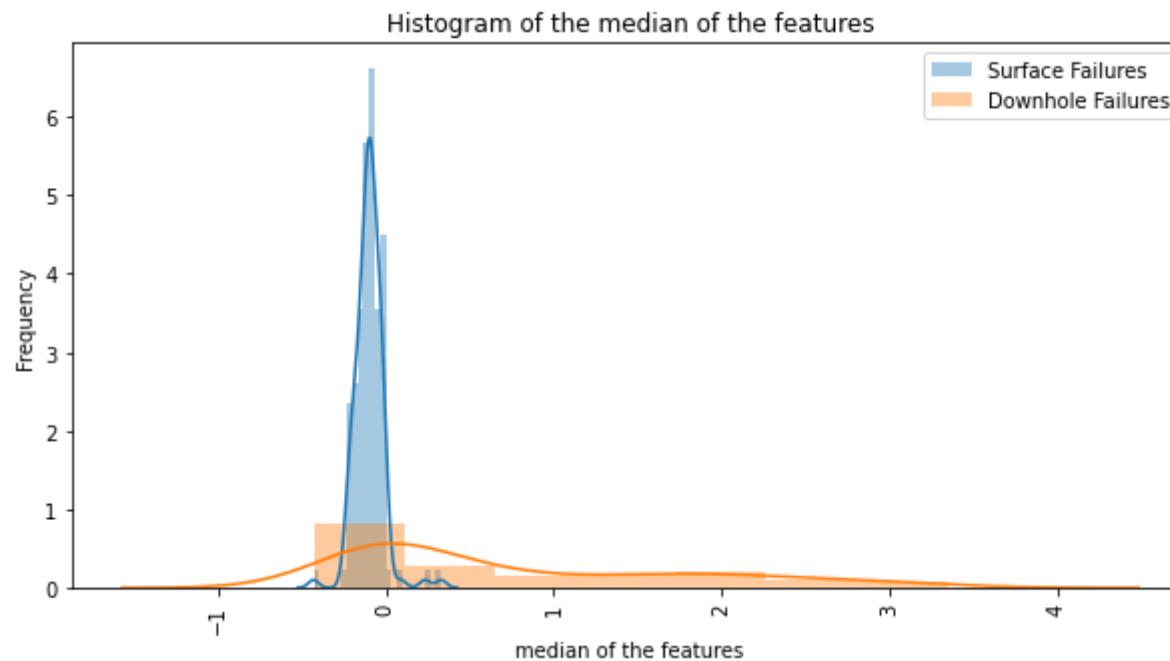
```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Histogram of the Mean of the features')
plt.ylabel('Frequency')
plt.xlabel('Mean of the features')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations :

Here for the surface and downhole failures, we can see that the different downhole features have more variance. Whereas the majority of the surface related features have a very similar value.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('Histogram of the median of the features')
plt.ylabel('Frequency')
plt.xlabel('median of the features')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```

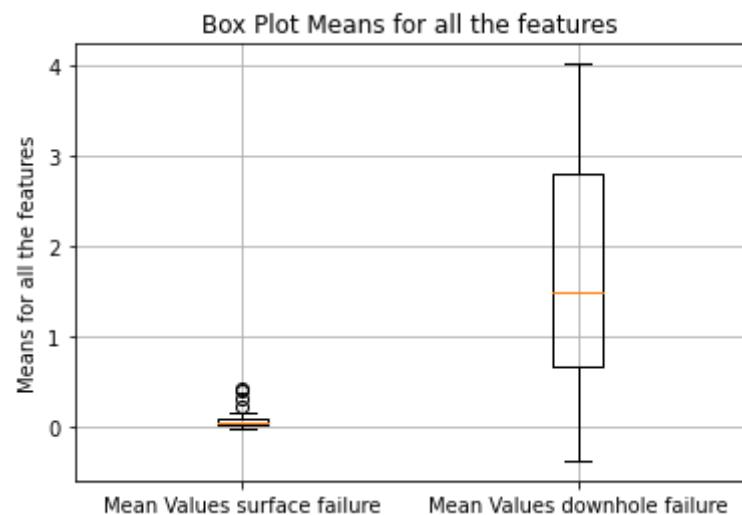


Observations :

Here for the surface and downhole failures, we can see that the different downhole features have more variance. Whereas the majority of the surface related features have a very similar value.
Similar can be said for the box plot and violin plot.

11) Try box plot / violin plot to visualize the outliers.

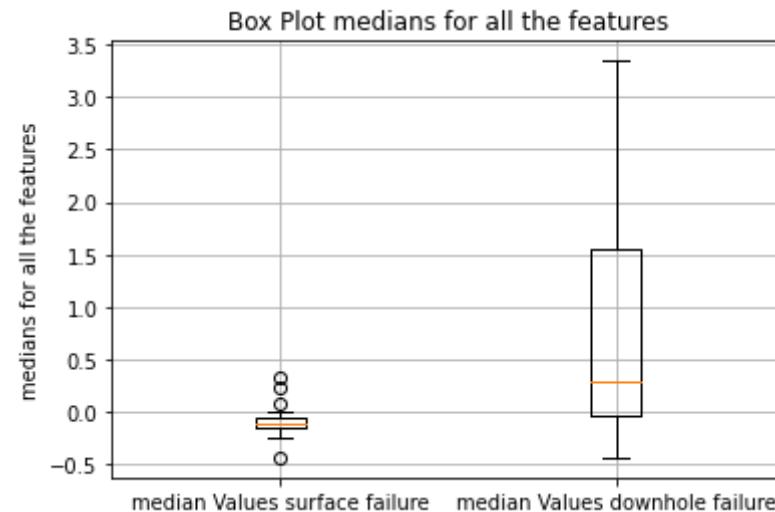
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Box Plot Means for all the features')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the features')  
plt.grid()  
plt.show()
```



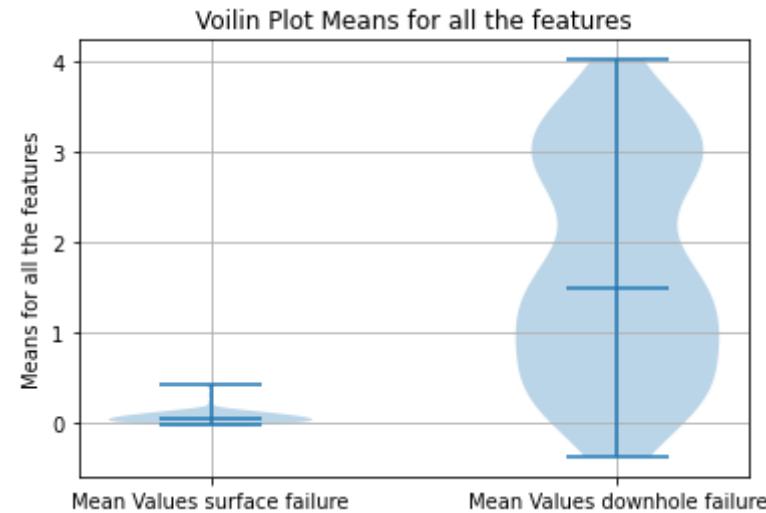
Observations :

The majority of the values for the surface failures lie near 0 whereas for the downhole failures, we can see all kinds of variation.

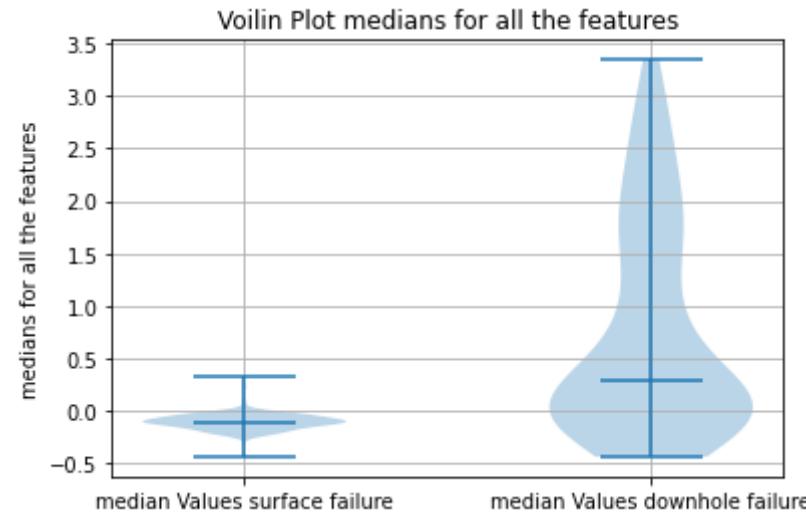
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('Box Plot medians for all the features')  
plt.xticks([1,2],('median Values surface failure','median Values downhole failure'))  
plt.ylabel('medians for all the features')  
plt.grid()  
plt.show()
```



```
In [ ]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Violin Plot Means for all the features')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the features')
plt.grid()
plt.show()
```



```
In [ ]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('Violin Plot medians for all the features')
plt.xticks([1,2],('median Values surface failure','median Values downhole failure'))
plt.ylabel('medians for all the features')
plt.grid()
plt.show()
```



PCA

```
In [86]: new_x_oss_df.drop("target",axis=1,inplace=True)
```

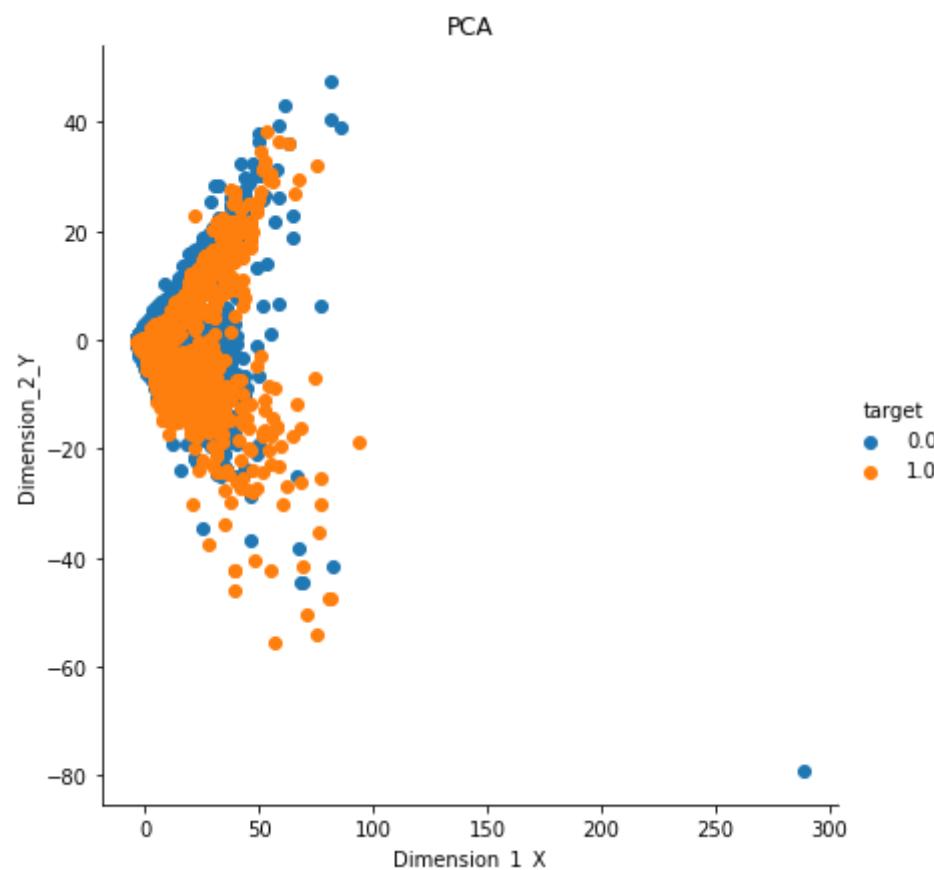
In [87]: "target" in new_x_oss_df.columns

Out[87]: False

In [88]:

```
%%time  
run_pca(new_x_oss_df,y_oss)
```

```
[[ -3.6391144   -0.27239496   0.          ]  
[ -2.2456899    0.11712268   0.          ]  
[  1.1500471    0.17947987   0.          ]  
...  
[ 43.470196   -12.275338    1.          ]  
[  0.54847836  -1.3452878   1.          ]  
[ 55.52745     29.296225    1.          ]]
```



CPU times: user 514 ms, sys: 121 ms, total: 634 ms
Wall time: 513 ms

Observation :

- 1) Observe that PCA is not able to differentiate all the data points into different groups.
- 2) Not all the the surface and downhole features are coming under the same category.

Perform a similar EDA on the outliers.

Here we take the outliers from the features that have been selected.

For each sensor we find the outliers that also classwise.

The issue is that there maybe high outliers here , but there maybe lower outliers here as well.

<https://www.geeksforgeeks.org/interquartile-range-and-quartile-deviation-using-numpy-and-scipy/> (<https://www.geeksforgeeks.org/interquartile-range-and-quartile-deviation-using-numpy-and-scipy/>)

Higher outlier = more than $1.5 \cdot \text{IQR}$ above the third quartile.

Lower outlier = less than $1.5 \cdot \text{IQR}$ below the first quartile.

The IQR is the difference between Q3 and Q1.

In []: new_x_oss_df["target"] = y_oss

```
In [ ]: counter=0
main_dictionary=dict()
outliers=[]
list_of_outliers_features_classes=[]
for i in range(len(column_list)):
    d=dict()
    new_list=new_x_oss_df[new_x_oss_df.target==one_zero[i]]\
    [column_list[i]]
    new_index=list(new_list.reset_index()["index"])
    QR1=np.percentile(new_list, 25)
    QR3=np.percentile(new_list, 75)
    IQR=QR3-QR1
    for index,value in zip(new_index,new_list):
        if (value>(QR3+(IQR*1.5))) or (value<(QR1-(IQR*1.5))):
            counter=counter+1
            d.update({index:value})
            if index not in main_dictionary:
                main_dictionary.update({index:list(new_x_oss_df.loc[index])})
    list_of_outliers_features_classes.append(d)
    del d
    outliers.append(counter)
    counter=0
```

```
In [ ]: print("Total number of data points that are outliers in atleast one or more than one feature : ",len(np.unique(outliers)))
Total number of data points that are outliers in atleast one or more than one feature : 31928
```

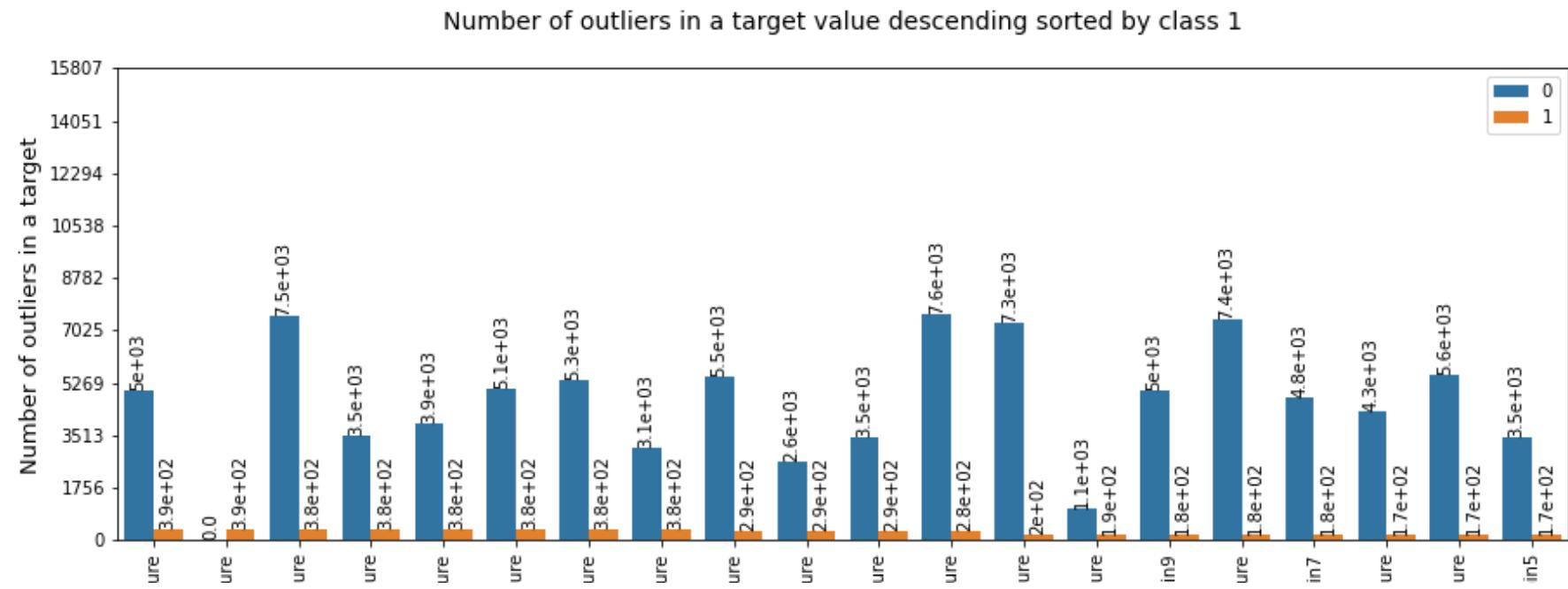
```
In [ ]: index=np.sort([i for i in range(0,len(column_list)) if i%2!=0])
d={i:outliers[i] for i in index}

d_keys=[]
for index in np.argsort(list(d.values()))[::-1][:len(list(d.values()))]:
    d_keys.append(list(d.keys())[index])

counter=0
new_column_list=[]
new_outliers_list=[]
new_one_zero=[]
for i in d_keys:
    new_column_list.append(column_list[i-1])
    new_outliers_list.append(outliers[i-1])
    new_one_zero.append(one_zero[i-1])
    new_column_list.append(column_list[i])
    new_outliers_list.append(outliers[i])
    new_one_zero.append(one_zero[i])
    counter=counter+1
print("Total {} sensors.".format(counter))
```

Total 128 sensors.

```
In [ ]: try:  
        plot_diagram(new_column_list,new_outliers_list,new_one_zero,"Number of outliers",1)  
    except ValueError:  
        pass
```



Observation :

Number of outliers are present for both the surface failures as well as the downhole failures.

For every column feature, get the mean for the outliers for both class 0 and class 1.

```
In [ ]: mean_0=[]
mean_1=[]
for i in range(0,len(list_of_outliers_features_classes),2):
    class_0=[ele for ele in list_of_outliers_features_classes[i].values()]
    class_1=[ele for ele in list_of_outliers_features_classes[i+1].values()]
    if (len(class_0)==0) :
        mean_0.append(0.01)
    else :
        mean_0.append(np.mean(class_0))

    if (len(class_1)==0):
        mean_1.append(0.01)
    else :
        mean_1.append(np.mean(class_1))
```

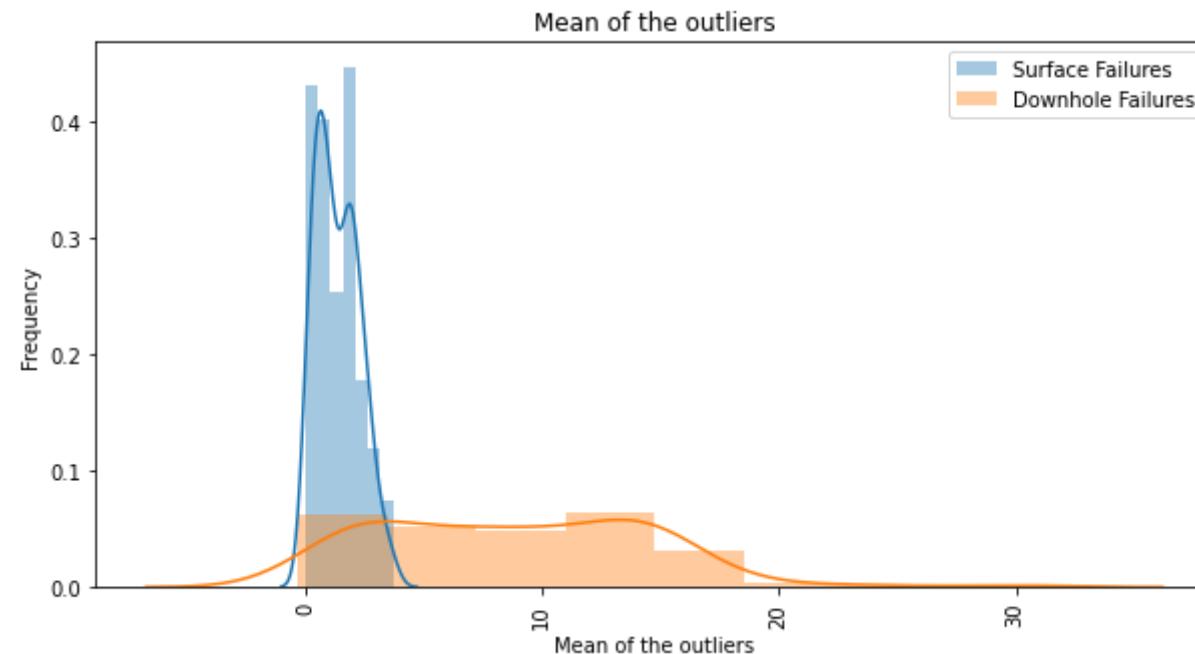
```
In [ ]: median_0=[]
median_1=[]
for i in range(0,len(list_of_outliers_features_classes),2):
    class_0=[ele for ele in list_of_outliers_features_classes[i].values()]
    class_1=[ele for ele in list_of_outliers_features_classes[i+1].values()]
    if (len(class_0)==0) :
        median_0.append(0.01)
    else :
        median_0.append(np.median(class_0))

    if (len(class_1)==0):
        median_1.append(0.01)
    else :
        median_1.append(np.median(class_1))
```

Plotting the mean and median for each feature class wise.

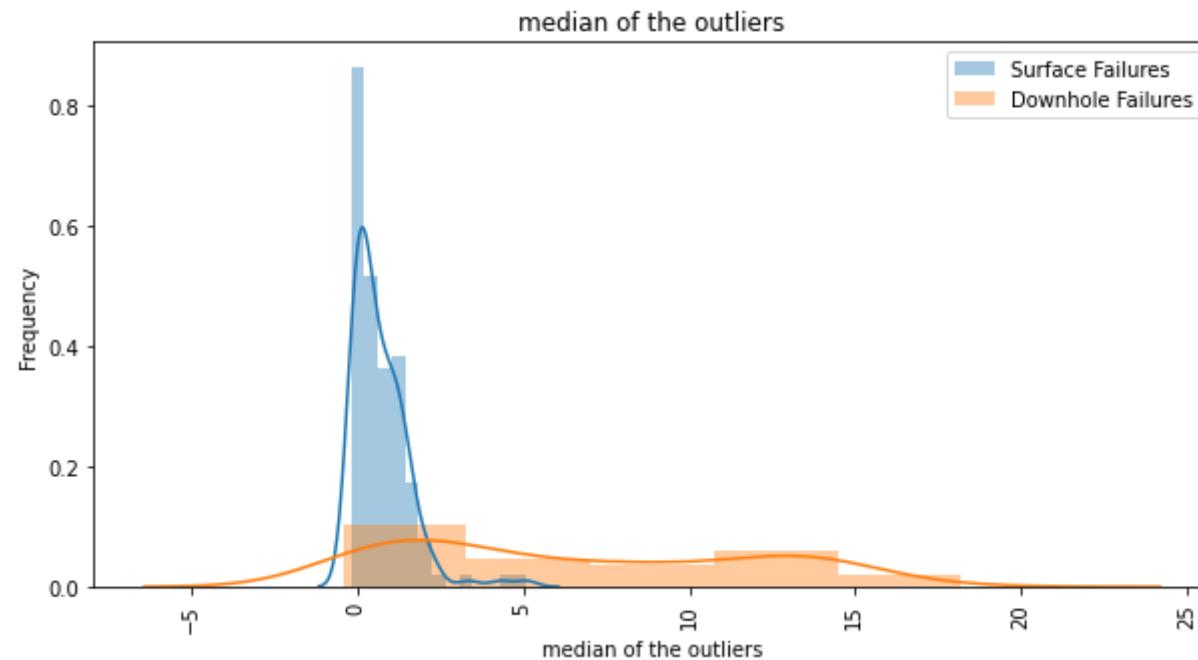
10) Try and plot it's histogram for both the target values.

```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(mean_0, hist=True, label="Surface Failures")
sns.distplot(mean_1, hist=True, label="Downhole Failures")
plt.title('Mean of the outliers')
plt.ylabel('Frequency')
plt.xlabel('Mean of the outliers')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



Observations : For the outliers, majority of them seem to be concentrated around 0 for the surface outliers with variance as well. For downhole outliers they have lesser variance but have more density in the other range values

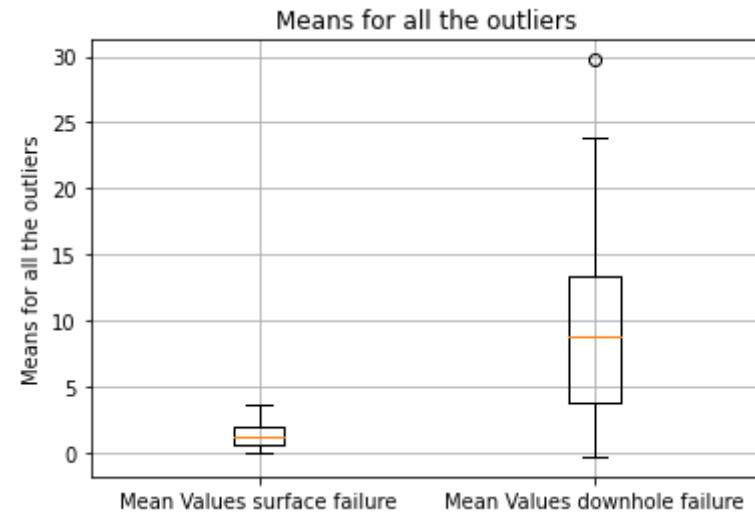
```
In [ ]: plt.figure(figsize=(10,5))
sns.distplot(median_0, hist=True, label="Surface Failures")
sns.distplot(median_1, hist=True, label="Downhole Failures")
plt.title('median of the outliers')
plt.ylabel('Frequency')
plt.xlabel('median of the outliers')
plt.xticks(rotation=90)
plt.legend()
plt.show()
```



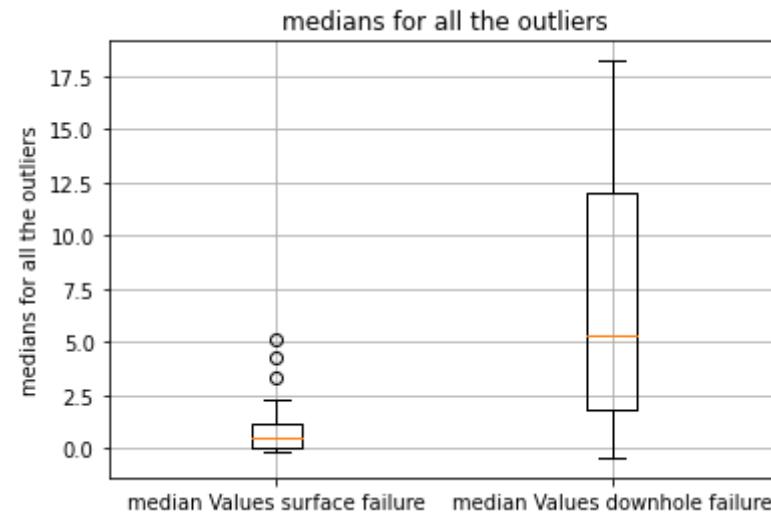
Observations : Here for the surface and downhole outlier, we can see that the different downhole features have more variance. Whereas the majority of the surface related features have a very similar value.

11) Try box plot / violin plot to visualize the outliers.

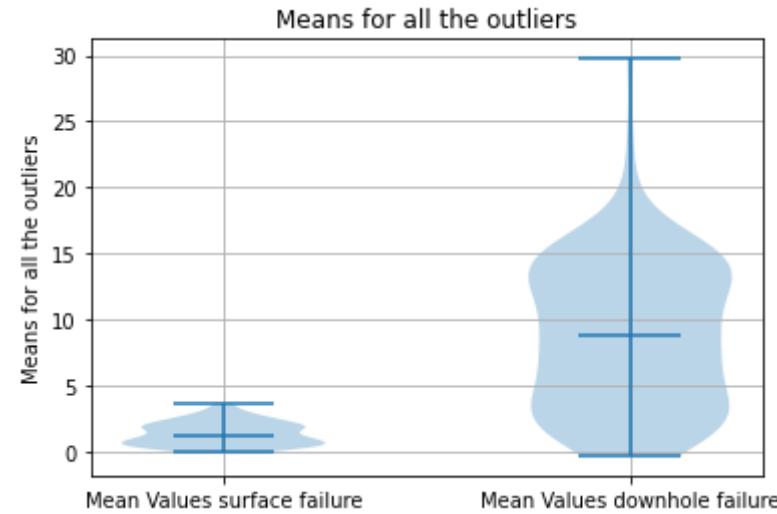
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([mean_0, mean_1])  
plt.title('Means for all the outliers')  
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))  
plt.ylabel('Means for all the outliers')  
plt.grid()  
plt.show()
```



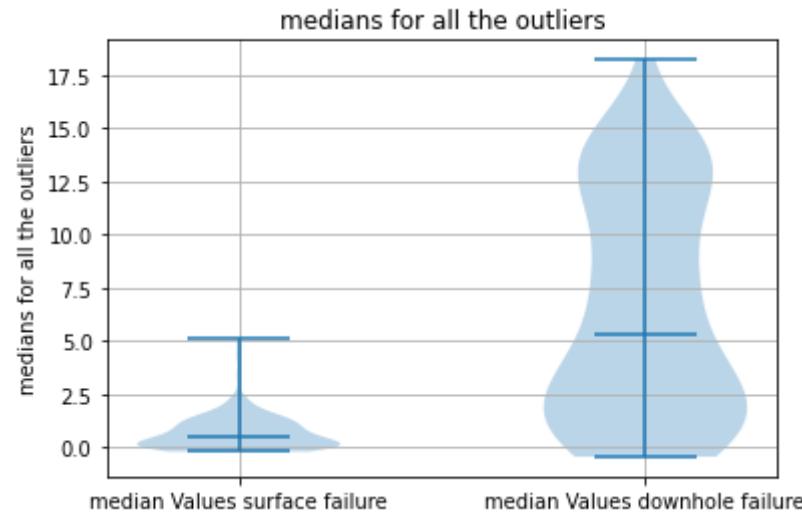
```
In [ ]: #Box plot for the number of words in the approved and not approved projects  
#https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html  
plt.boxplot([median_0, median_1])  
plt.title('medians for all the outliers')  
plt.xticks([1,2],('median Values surface failure','median Values downhole failure'))  
plt.ylabel('medians for all the outliers')  
plt.grid()  
plt.show()
```



```
In [ ]: plt.violinplot([mean_0, mean_1], showmeans=False, showmedians=True)
plt.title('Means for all the outliers')
plt.xticks([1,2],('Mean Values surface failure','Mean Values downhole failure'))
plt.ylabel('Means for all the outliers')
plt.grid()
plt.show()
```



```
In [ ]: plt.violinplot([median_0, median_1], showmeans=False, showmedians=True)
plt.title('medians for all the outliers')
plt.xticks([1,2],('median Values surface failure','median Values downhole failure'))
plt.ylabel('medians for all the outliers')
plt.grid()
plt.show()
```



The outliers although for the surface related failures are higher. But both the means and the medians of the outliers are similar.

For the given data points and features enter new features that tell if these values were previously np.NaN or not.

```
In [ ]: new_x_oss_df.drop("target",axis=1,inplace=True)
```

```
In [ ]: nan_x_oss_df=fuse_dataframe_nan(new_x_oss_df,nan_or_not,set(new_column_list))
nan_x_oss_df["target"]=y_oss
nan_x_oss_df.head()
```

Out[675]:

	sensor7_histogram_bin2	sensor105_histogram_bin3	sensor69_histogram_bin7	sensor64_histogram_bin2	sensor107_measure	sensor40_measure
0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	1.0

5 rows × 129 columns

Now this is the problem out of the given data that we have , we select the features that have more than 50% np.NaNs

```
In [ ]: counter=0
for column in new_x_oss_df.columns.values:
    try:
        percent_of_nan=nan_x_oss_df[column].value_counts()[1]/\
                      (nan_x_oss_df[column].value_counts()[0]+\
                       nan_x_oss_df[column].value_counts()[1])
        if (percent_of_nan>.5) :
            counter=counter+1
            print(column," : ",percent_of_nan)
    except KeyError:
        pass
print("Overall : ",counter," features with % of na's more than 50%")
```

```
sensor2_measure      :  0.7740833333333333
sensor38_measure     :  0.6605416666666667
sensor39_measure     :  0.7348958333333333
sensor40_measure     :  0.77375
sensor41_measure     :  0.7967916666666667
sensor42_measure     :  0.8133125
sensor43_measure     :  0.822
Overall : 7 features with % of na's more than 50%
```

Summary :

The data contained a lot of missing values and is highly imbalanced.

- a) I did the analysis on the data first without imputing any values.
- b) Here without imputing I got very less data as the calculation was done on the data sets that had real values for all the features.
- c) Here I was able to find out that a few sensors were more responsible for the downhole failures and few were more responsible surface related failures, with the help of histograms, box plots, violin plots class wise median value for each sensor.

I took various methods to fill in the missing na values.

1)

- a) Impute 0 :
 - i) Standardize.
 - ii) Perform truncated svd to find the number of required features useless features.
 - iii) Perform Recursive Feature Selection to reduce the features.
 - iv) Perform spearman correlation to get rid of highly correlated features.
 - b) Take the mean of the values(0 this time) and do analysis for both surface and downhole failures.
 - c) Take the median of the values(0 this time) and do analysis for both surface and downhole failure s.
 - d) Try to find the sensors involved.
 - e) Given the features that we have selected , try a basic classifier on the features and predict a v alue.
 - f) Do the EDA of this particular config of data.
 - g) Do the EDA for the outliers.
 - h) Out of the features that we have selected , try to find the features that previously had more tha n 50% np.NaNs
 - i) Perform pca and tsne to see if these features get separated or not.
- 2)

- a) Impute mean :
 - i) Standardize.
 - ii) Perform truncated svd to find the number of required features useless features.
 - iii) Perform Recursive Feature Selection to reduce the features.
 - iv) Perform spearman correlation to get rid of highly correlated features.

- b) Take the mean of the values(mean this time) and do analysis for both surface and downhole failure s.
- c) Take the median of the values(mean this time) and do analysis for both surface and downhole failu res.
- d) Try to find the sensors involved.
- e) Given the features that we have selected , try a basic classifier on the features and predict a v alue.
- f) Do the EDA of this particular config of data.
- g) Do the EDA for the outliers.
- h) Out of the features that we have selected , try to find the features that previously had more tha n 50% np.NaNs
- i) Perform pca and tsne to see if these features get separated or not.

3)

- a) Impute median :
 - i) Standardize.
 - ii) Perform truncated svd to find the number of required features useless features.
 - iii) Perform Recursive Feature Selection to reduce the features.
 - iv) Perform spearman correlation to get rid of highly correlated features.

- b) Take the mean of the values(median this time) and do analysis for both surface and downhole failures.
- c) Take the median of the values(median this time) and do analysis for both surface and downhole failures.
- d) Try to find the sensors involved.
- e) Given the features that we have selected , try a basic classifier on the features and predict a value.
- f) Do the EDA of this particular config of data.
- g) Do the EDA for the outliers.
- h) Out of the features that we have selected , try to find the features that previously had more than 50% np.NaNs
- i) Perform pca and tsne to see if these features get separated or not.

I found the median values to be less affected than the outliers.So with median imputation.I performed various sampling techniques.

The data set is highly imbalanced , and all the features are numerical so I will try :

Smote, BorderLineSmote, Adasyn(oversampling)

OneSidedSelection (undersampling).

Smoteen or SmoteTomek etc for the combination of over sampling and undersampling.

4)

- a) Impute median :
 - i) Standardize.
 - ii) Perform truncated svd to find the number of required features useless features.
 - iii) Perform Recursive Feature Selection to reduce the features.
 - iv) Perform spearman correlation to get rid of highly correlated features.
 - v) Perform Adasyn to oversample the minority.

 - b) Take the mean of the values(median this time) and do analysis for both surface and downhole failures.
 - c) Take the median of the values(median this time) and do analysis for both surface and downhole failures.
 - d) Try to find the sensors involved.
 - e) Given the features that we have selected , try a basic classifier on the features and predict a value.
 - f) Do the EDA of this particular config of data.
 - g) Do the EDA for the outliers.
 - h) Out of the features that we have selected , try to find the features that previously had more than 50% np.NaNs
 - i) Perform pca to see if we get the features separated visually or not.
- 5)

- a) Impute median :
 - i) Standardize.
 - ii) Perform truncated svd to find the number of required features useless features.
 - iii) Perform Recursive Feature Selection to reduce the features.
 - iv) Perform spearman correlation to get rid of highly correlated features.
 - v) Perform SmoteTomek for both over under sampling combo on the minority.
- b) Take the mean of the values(median this time) and do analysis for both surface and downhole failures.
- c) Take the median of the values(median this time) and do analysis for both surface and downhole failures.
- d) Try to find the sensors involved.
- e) Given the features that we have selected , try a basic classifier on the features and predict a value.
- f) Do the EDA of this particular config of data.
- g) Do the EDA for the outliers.
- h) Out of the features that we have selected , try to find the features that previously had more than 50% np.NaNs
- i) Perform pca to see if we get the features separated visually or not.

6)

- a) Impute median :
 - i) Standardize.
 - ii) Perform truncated svd to find the number of required features useless features.
 - iii) Perform Recursive Feature Selection to reduce the features.
 - iv) Perform spearman correlation to get rid of highly correlated features.
 - v) Perform One sided Sampling on the majority.

- b) Take the mean of the values(median this time) and do analysis for both surface and downhole failures.
- c) Take the median of the values(median this time) and do analysis for both surface and downhole failures.
- d) Try to find the sensors involved.
- e) Given the features that we have selected , try a basic classifier on the features and predict a value.
- f) Do the EDA of this particular config of data.
- g) Do the EDA for the outliers.
- h) Out of the features that we have selected , try to find the features that previously had more than 50% np.NaNs
- i) Perform pca to see if we get the features separated visually or not.

Observation : The most groups that I could see was in the histogram measures, but none of the methods are able to separate the surface and downhole failures.

Mean values and Median values feature wise and class wise was performed 2 times :

1) With no imputing => Here the data is less as only the data points with all the features are considered. Considerable gap between the median values of both the failures.

2) With mean imputing => Here there was more data involved as more all the data points were included after mean imputing.
We found that on taking the mean value we can tell out the difference between the 2 features although the values decreased when we took the median.

3) With median imputing => Here there was more data involved as more all the data points were included after median imputing.
We found that on taking the mean value we can tell out the difference between the 2 features although the values decreased when we took the median.

The issue we face with while performing spearman correlation,
we cannot make their correlation go away but we can do one thing that is remove the features that are more co related.
Issue is that when we remove the co related features then only 2 features with the co relation of -10% to +10% range remains.
So again to have more features,I decide to accept atleast till -98% to +98% range because I need the required amount of the features
(adequate number found in svd and features retrieved in Recursive Feature Selection) to perform the analysis.

I can also try the median imputed median can I get results ? I will give three sets of datasets that is median imputed median data,
adasyn and smotetomek data to the DL/ML algorithm so that I can come to know through which datasets I can get the best results.