Loading Data

```
#importing libraries

import matplotlib
import matplotlib.pyplot as plt
import random

import pandas as pd
import numpy as np

from tqdm import tqdm

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE


#!curl --header 'Host: doc-08-3c-docs.googleusercontent.com' --user-agent 'Mozilla/5.0 (X11; Linux x86_64; rv:81.0) Gecko/20100101 Firefox/81.0' --header 'Accept: text/html,appl
#!unzip /content/falls.zip
#!pip install seaborn --upgrade
import seaborn as sns

print(sns.__version__)
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 21.3M    0 21.3M    0     0  25.4M      0 --:--:-- --:--:-- --:--:-- 25.4M
Archive:  /content/falls.zip
   creating: falls/
  inflating: falls/equip_failures_test_set.csv
  inflating: falls/equip_failures_training_set.csv
Collecting seaborn
  Downloading https://files.pythonhosted.org/packages/bc/45/5118a05b0d61173e6eb12bc5804f0fbb6f196adb0a20e0b16efc2b8e98be/seaborn-0.11.0-py3-none-any.whl (283kB)
     |████████████████████████████████| 286kB 2.5MB/s
Requirement already satisfied, skipping upgrade: scipy>=1.0 in /usr/local/lib/python3.6/dist-packages (from seaborn) (1.4.1)
Requirement already satisfied, skipping upgrade: pandas>=0.23 in /usr/local/lib/python3.6/dist-packages (from seaborn) (1.0.5)
Requirement already satisfied, skipping upgrade: numpy>=1.15 in /usr/local/lib/python3.6/dist-packages (from seaborn) (1.18.5)
Requirement already satisfied, skipping upgrade: matplotlib>=2.2 in /usr/local/lib/python3.6/dist-packages (from seaborn) (3.2.2)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.6.1 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.23->seaborn) (2.8.1)
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.23->seaborn) (2018.9)
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=2.2->seaborn) (1.2.0)
Requirement already satisfied, skipping upgrade: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=2.2->seaborn) (2.4.7)
Requirement already satisfied, skipping upgrade: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-dateutil>=2.6.1->pandas>=0.23->seaborn) (1.15.0)
Installing collected packages: seaborn
  Found existing installation: seaborn 0.10.1
    Uninstalling seaborn-0.10.1:
      Successfully uninstalled seaborn-0.10.1
Successfully installed seaborn-0.11.0
0.11.0
```

```
df = pd.read_csv("/content/falls/equip_failures_training_set.csv")

df.head()
```

	id	target	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1	sensor7_h
0	1	0	76698	na	2130706438	280	0	0	0	0	
1	2	0	33058	na	0	na	0	0	0	0	
2	3	0	41040	na	228	100	0	0	0	0	
3	4	0	12	0	70	66	0	10	0	0	
4	5	0	60874	na	1368	458	0	0	0	0	

5 rows × 172 columns

Make dataset interpretable to machine

Replace na with np.nan

```
"""Instead of nan value we have na, so we will replace na with np.nan"""

df = df.replace('na', np.NaN)
df.head()
```

	id	target	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1	sensor7_h
0	1	0	76698	NaN	2130706438	280	0	0	0	0	
1	2	0	33058	NaN	0	NaN	0	0	0	0	
2	3	0	41040	NaN	228	100	0	0	0	0	
3	4	0	12	0	70	66	0	10	0	0	
4	5	0	60874	NaN	1368	458	0	0	0	0	

5 rows × 172 columns

Change data-type of dataframe

```
df.dtypes
```

```
id                            int64
target                        int64
```

```
"We could see that few coloumns are of int type, and other are of object type,So for using data we need to make them float data type"

df = df.astype("float32")
df.dtypes
```

```
id                           float32
target                       float32
sensor1_measure              float32
sensor2_measure              float32
sensor3_measure              float32
                              ...
sensor105_histogram_bin7     float32
sensor105_histogram_bin8     float32
sensor105_histogram_bin9     float32
sensor106_measure            float32
sensor107_measure            float32
Length: 172, dtype: object
```

Drop useless coloumn from feature

```
"""id coloumn is just index, we don't need it , so we will drop it"""
df = df.drop(["id"],axis=1)
df.head()
```

	target	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1	sensor7_hist
0	0.0	76698.0	NaN	2.130706e+09	280.0	0.0	0.0	0.0	0.0	
1	0.0	33058.0	NaN	0.000000e+00	NaN	0.0	0.0	0.0	0.0	
2	0.0	41040.0	NaN	2.280000e+02	100.0	0.0	0.0	0.0	0.0	
3	0.0	12.0	0.0	7.000000e+01	66.0	0.0	10.0	0.0	0.0	
4	0.0	60874.0	NaN	1.368000e+03	458.0	0.0	0.0	0.0	0.0	

5 rows × 171 columns

Understanding dataframe in numerical way

Getting count,mean,standard deviation, min, max ,25th ,50th and 75th percentile of each feature in data

```
df.describe()
```

	target	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure	sensor6_measure	sensor7_histogram_bin0	sensor7_histogram_bin1	sens
count	60000.000000	6.000000e+04	13671.000000	5.666500e+04	4.513900e+04	57500.000000	57500.000000	5.932900e+04	5.932900e+04	
mean	0.016667	5.933432e+04	0.713189	3.560139e+08	1.906050e+05	6.819130	11.006818	2.216364e+02	9.757225e+02	
std	0.128069	1.454207e+05	3.479168	7.948017e+08	4.040431e+07	161.485977	209.747253	2.047733e+04	3.418985e+04	
min	0.000000	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000e+00	0.000000e+00	
25%	0.000000	8.340000e+02	0.000000	1.600000e+01	2.400000e+01	0.000000	0.000000	0.000000e+00	0.000000e+00	
50%	0.000000	3.077600e+04	0.000000	1.520000e+02	1.260000e+02	0.000000	0.000000	0.000000e+00	0.000000e+00	
75%	0.000000	4.866800e+04	0.000000	9.640000e+02	4.300000e+02	0.000000	0.000000	0.000000e+00	0.000000e+00	
max	1.000000	2.746564e+06	204.000000	2.130707e+09	8.584298e+09	21050.000000	20070.000000	3.376892e+06	4.109372e+06	

8 rows × 171 columns

Exploratory Data Analysis

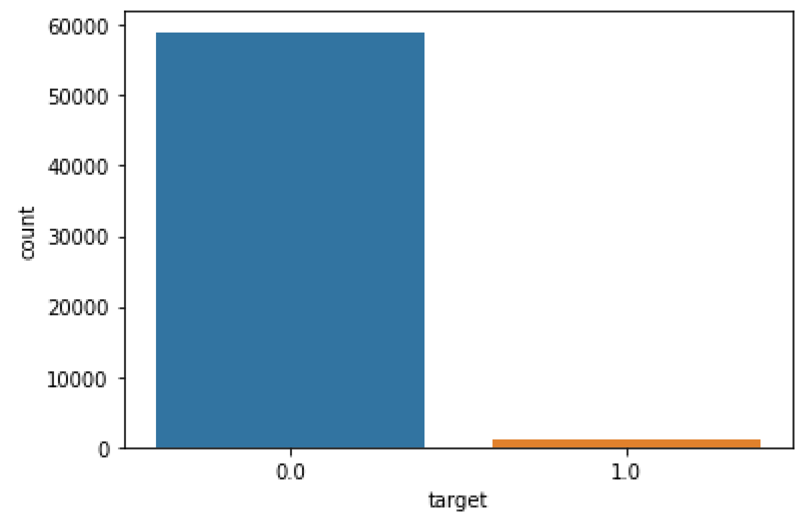
Get the distribution of classes

Downhole failure is signified by target value = 1

Surface failure is signified by target value = 0

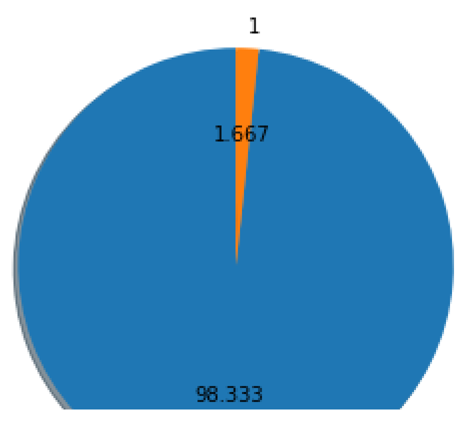
```
g = sns.countplot(x = "target" , data = df)
plt.xlabel("target")
plt.ylabel("count")
```

```
Text(0, 0.5, 'count')
```



```
#Percentage view of data distribution

plt.figure(figsize=(5,5))
plt.pie(df['target'].value_counts(),startangle=90,autopct="%.3f",labels=[0,1],shadow=True)
```

```
([<matplotlib.patches.Wedge at 0x7f906b3f5eb8>,
  <matplotlib.patches.Wedge at 0x7f906b3e6630>],
 [Text(-0.05756949701481714, -1.0984924911047236, '0'),
  Text(0.05756943916265415, 1.0984924941366225, '1')],
 [Text(-0.03140154382626389, -0.5991777224207583, '98.333'),
  Text(0.03140151227053862, 0.5991777240745213, '1.667')])
```



98.333% data is of class 0

1.667% data is of class 1

That means most of the failure happen on surface

Analysis between features and target

Getting correlation of features with target

```
y = df["target"]
X = df.drop(labels="target" , axis=1)

coloumns = df.columns



def get_highly_correlated_feature(dataframe=df,correlation="pearson",top_features=10,with_="target"):
    """
    correlation can be {‘pearson’, ‘kendall’, ‘spearman’}
    top feature: it will give you top n correlated feature
    with: pass the coloumn name, with whome you want correlation
    datagrame: pass pandas dataframe
    """

    pearson_corr_dict = dataframe.corr(method=correlation)[with_].to_dict()

    #sorted_dict = dict(sorted(pearson_corr_dict.items(), key=lambda x: x[1], reverse=True))
    top_n_features = dict(sorted(pearson_corr_dict.items(), key=lambda x: abs(x[1]) , reverse=True)[:top_features])

    return top_n_features

top_n_features = get_highly_correlated_feature(dataframe=df,correlation="pearson",top_features=5,with_="target")
print(top_n_features)
```

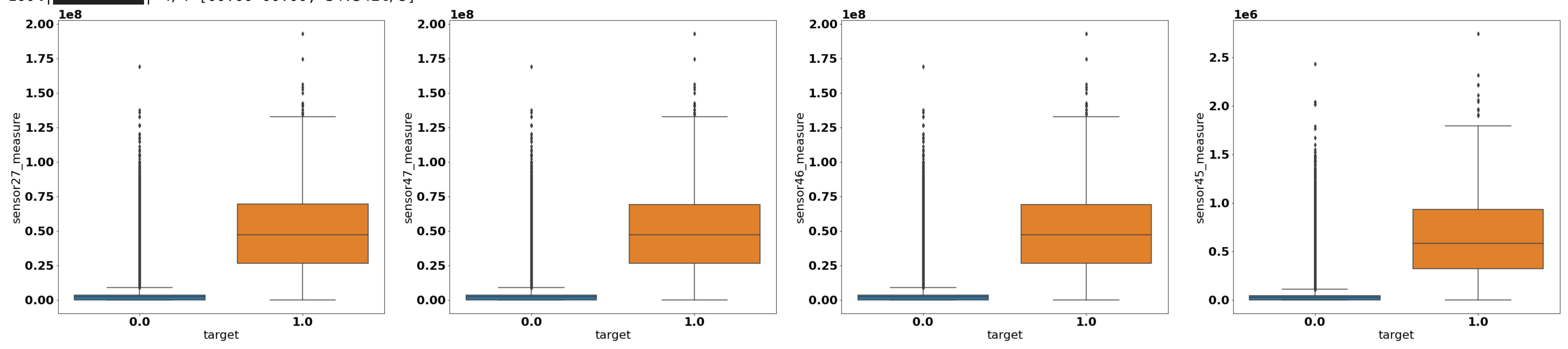
```
{'target': 1.0, 'sensor27_measure': 0.5427437085929686, 'sensor47_measure': 0.5415981338770961, 'sensor46_measure': 0.5415981089669337, 'sensor45_measure': 0.53745219727095
```

Univariate analysis

Box plot of top 4 features

```
font = {'family' : 'DejaVu Sans',
        'weight' : 'bold',
        'size'   : 22}

matplotlib.rc('font', **font)


figure, axis = plt.subplots(1 , 4, figsize=(50, 10), squeeze=False)

top_coloumns = list(top_n_features.keys())[1:]



for i in tqdm(range(0,4)):

    sns.boxplot( x= "target",    y=top_coloumns[i]       , data=df   , orient='v'       , ax = axis[0,i])
```

```
100%|██████████| 4/4 [00:00<00:00, 34.54it/s]
```



Observations:-

- Here we could see that highly correlated features are seperable
- There is lot of outliers

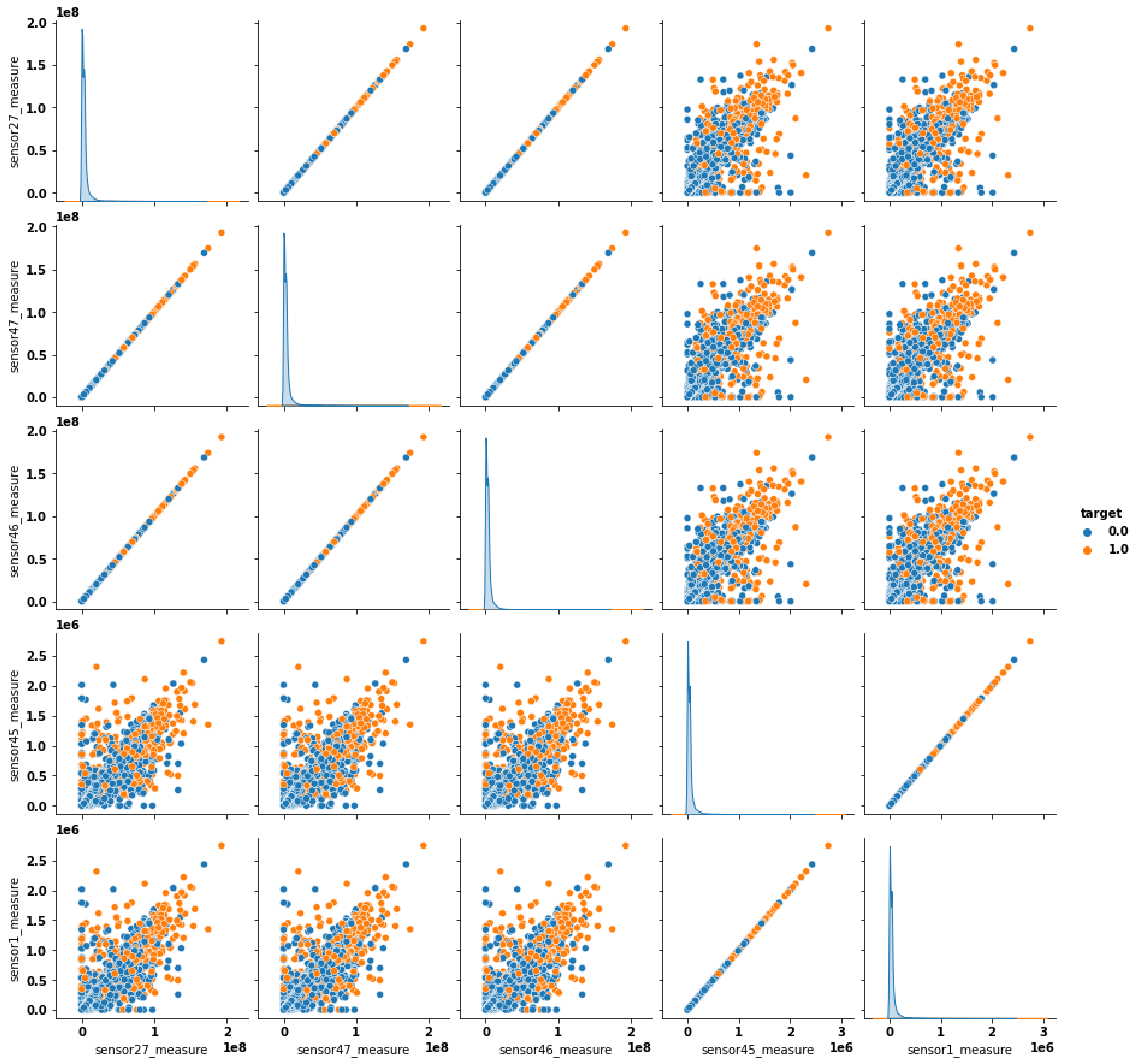
Conclusion:-

- We could use these top features to distinguish between our target
- We need to handle outliers

Bivariate analysis

```
""""Creating dataframe of coloumns that have higher potential to sperate both the classes"""

top_5_feature = list(get_highly_correlated_feature(dataframe=df,correlation="pearson",top_features=6,with_="target").keys())

Highly_Seperable = df.filter(top_5_feature, axis=1)


font = {'family' : 'DejaVu Sans',
        'weight' : 'bold',
        'size'   : 10}

matplotlib.rc('font', **font)

sns.pairplot(Highly_Seperable, hue="target")
```

```
<seaborn.axisgrid.PairGrid at 0x7f906b434ac8>
```



Observations

- We could see that few features are highly collinear

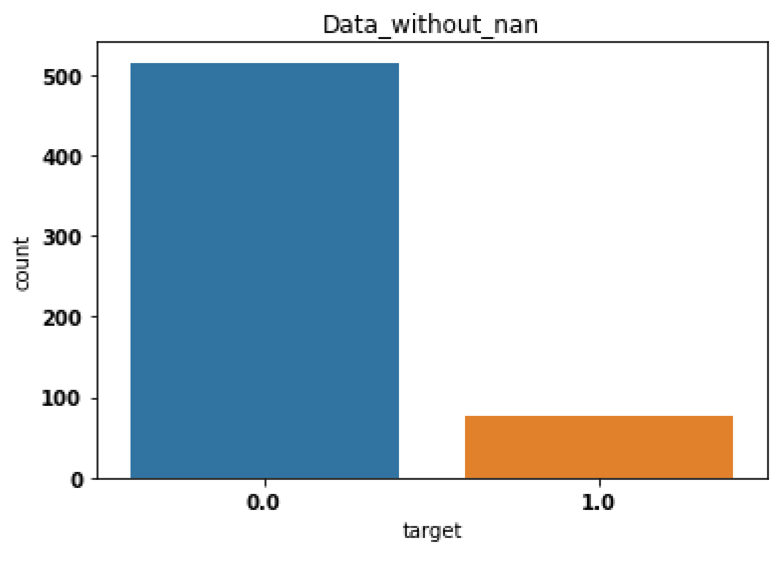
Conclusion

- We need to remove highly correlated features

Multivarriate analysis of data

```
df_without_nan = df.dropna()


g = sns.countplot(x = "target" , data = df_without_nan)
plt.xlabel("target")
plt.ylabel("count")
plt.title("Data_without_nan")
```

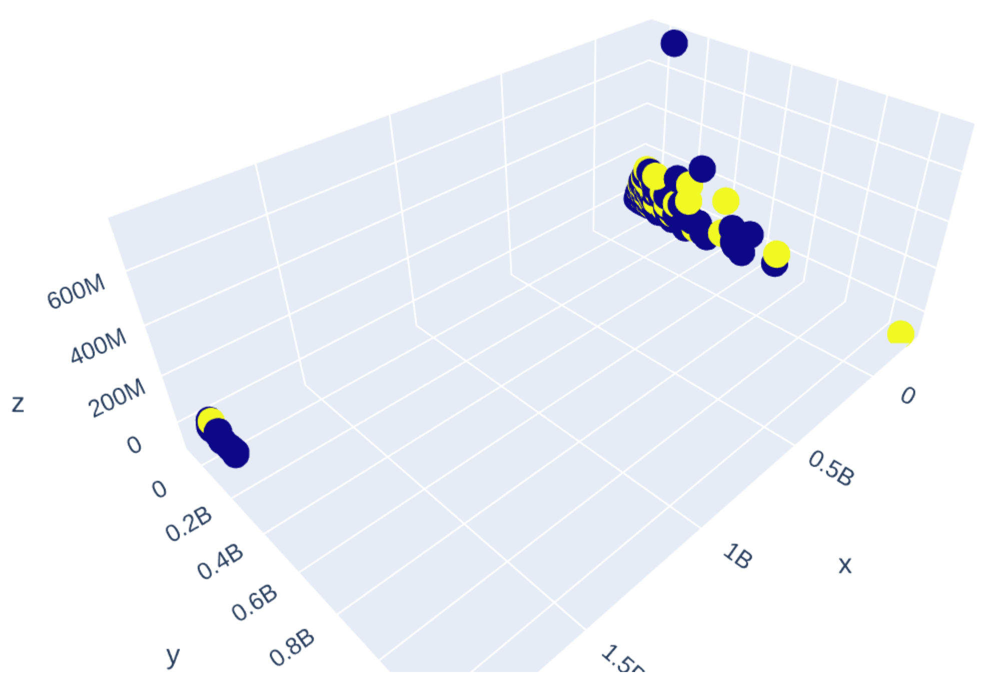
```
Text(0.5, 1.0, 'Data_without_nan')
```



PCA 3d plot

```
X = df_without_nan.drop(["target"],axis=1)
target = df_without_nan["target"].tolist()


pca = PCA(n_components=3)
```

```
pca_result = pca.fit_transform(X.values)

x = pca_result[:,0]
y = pca_result[:,1]
z = pca_result[:,2]



import plotly.express as px

pca_df = pd.DataFrame(list(zip(x, y, z, target)), columns =['x', 'y', 'z', 'target'])

fig = px.scatter_3d(pca_df, x='x', y='y', z='z',
              color='target')
fig.show()
```



Observations

- From 3d plot we could see that, data is not completely seperable.

Conclusion

- So we need to use most of the features, and we will also need to create new features.

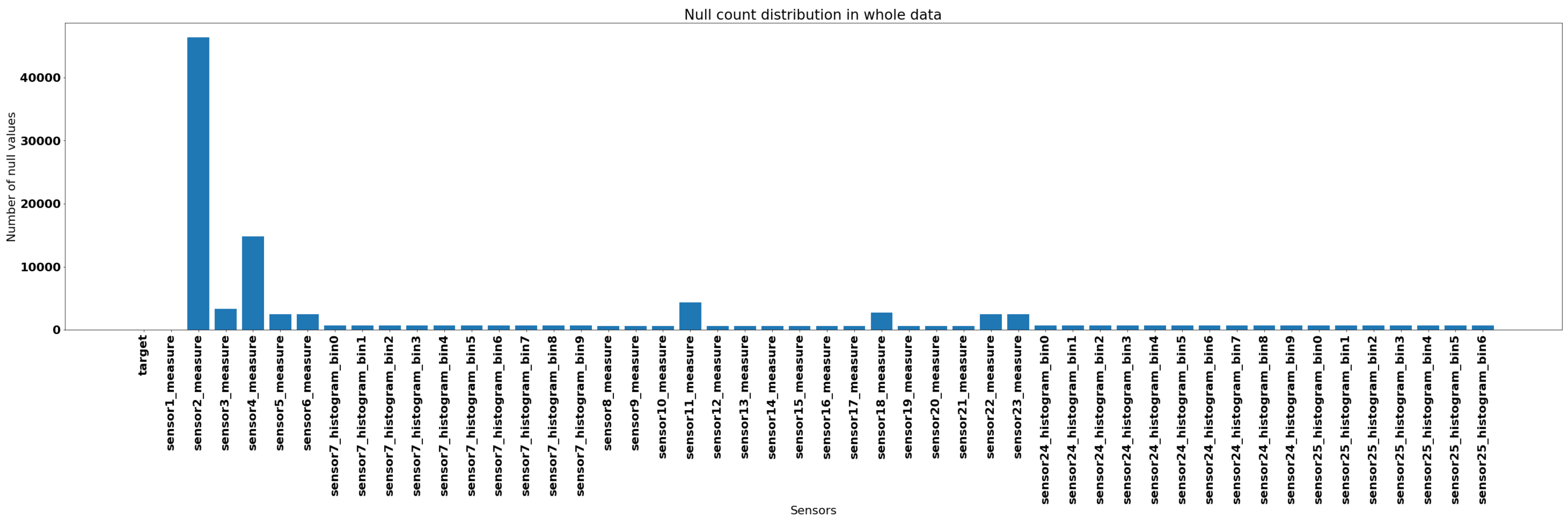
Analysis of nan values for each class

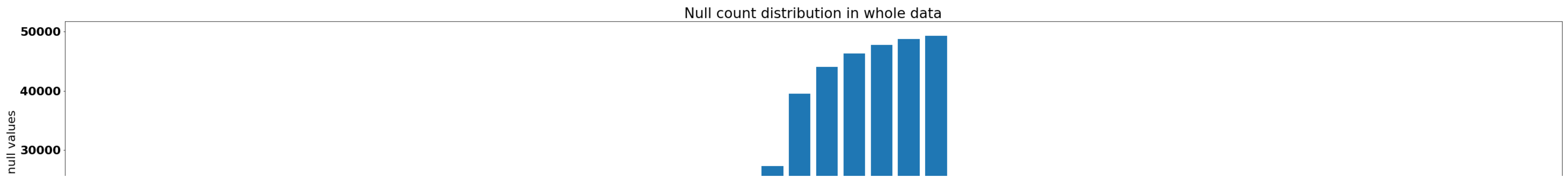
```
"""Create data frame for both the classes"""

df_1 = df[df["target"] == 1]
df_0 = df[df["target"] == 0]
```

Count wise null plot for whole data, for class 1 and for class 0

```
"""These dictionary will contain number of null values for each coloumns"""

null = dict(df.isnull().sum())
null_1 = dict(df_1.isnull().sum())
null_0 = dict(df_0.isnull().sum())

font = {'family' : 'DejaVu Sans',
        'weight' : 'bold',
        'size'   : 22}

matplotlib.rc('font', **font)



"""Null bar plot for whole data"""

coloumns = list(null.keys())
values = list(null.values())

col = []
val = []

col.append(coloumns[0:50])
val.append(values[0:50])

col.append(coloumns[50:100])
val.append(values[50:100])

col.append(coloumns[100:150])
val.append(values[100:150])

col.append(coloumns[150:171])
val.append(values[150:171])




figure, axis = plt.subplots(4, 1, figsize=(50, 80), squeeze=False)

for i in range(0,4):

    axis[i,0].bar(col[i], val[i])
    axis[i,0].set_xticks(range(len(col[i])), col[i])
    axis[i,0].set_xticklabels(col[i],rotation=90)
    axis[i,0].set_xlabel("Sensors")
    axis[i,0].set_ylabel("Number of null values")
    axis[i,0].set_title("Null count distribution in whole data")

plt.subplots_adjust(hspace=1.2)
plt.show()
```

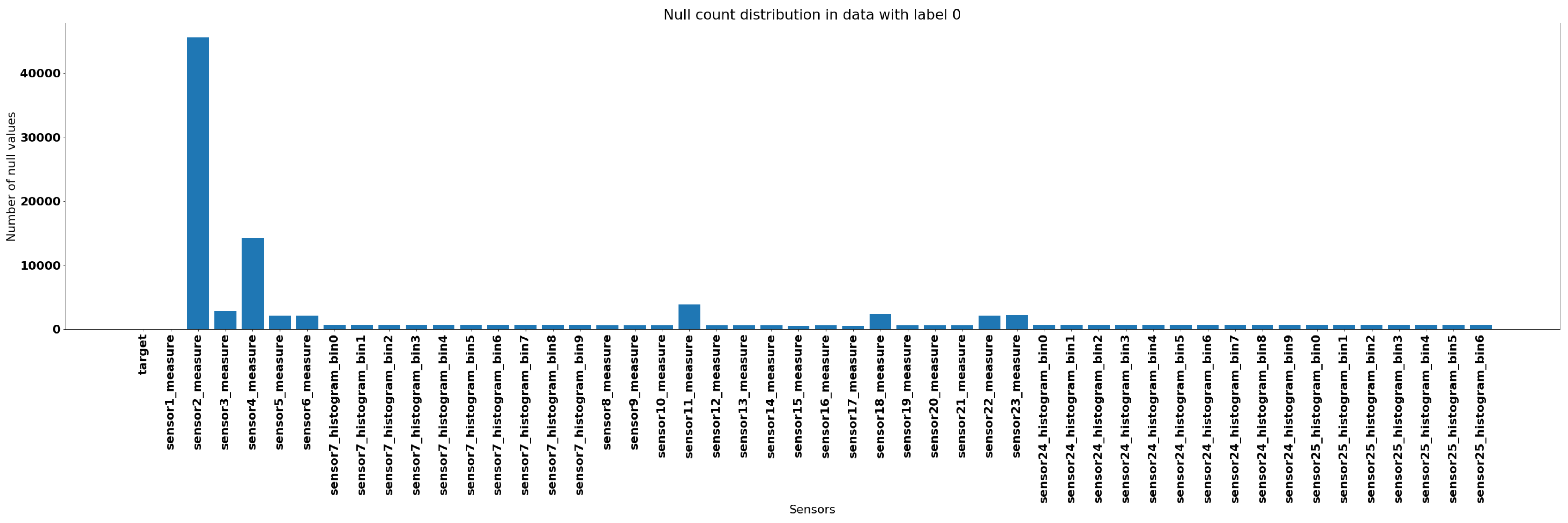
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:29: MatplotlibDeprecationWarning:

Passing the minor parameter of set_xticks() positionally is deprecated since Matplotlib 3.2; the parameter will become keyword-only two minor releases later.
```

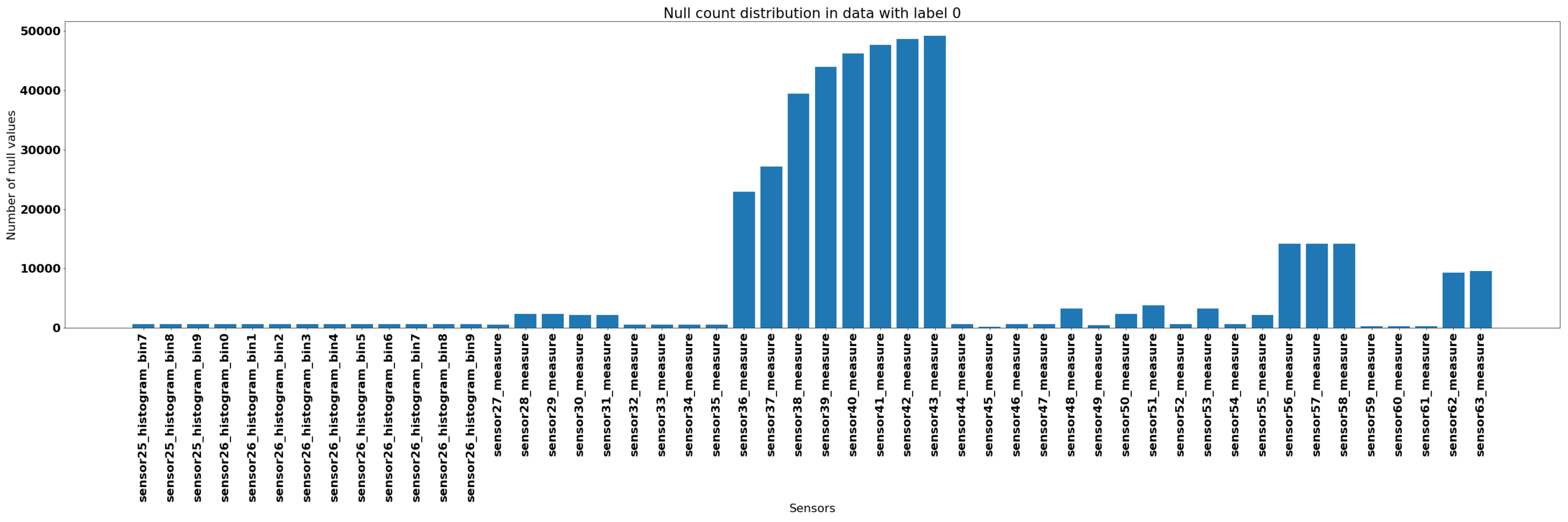


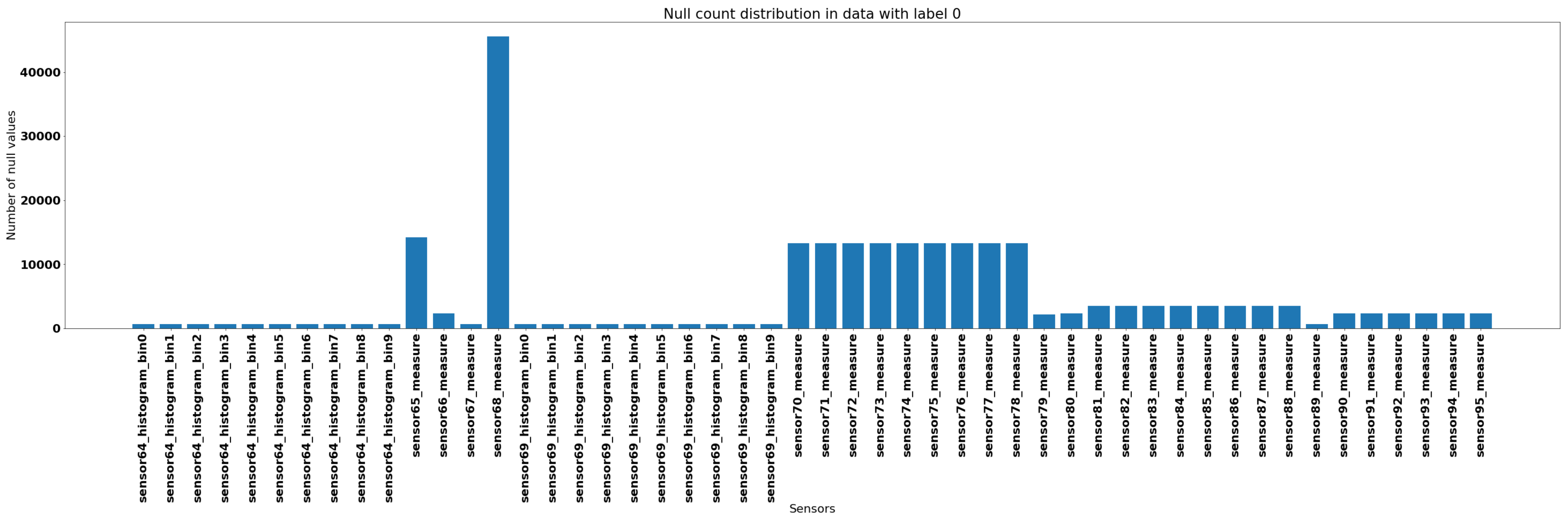


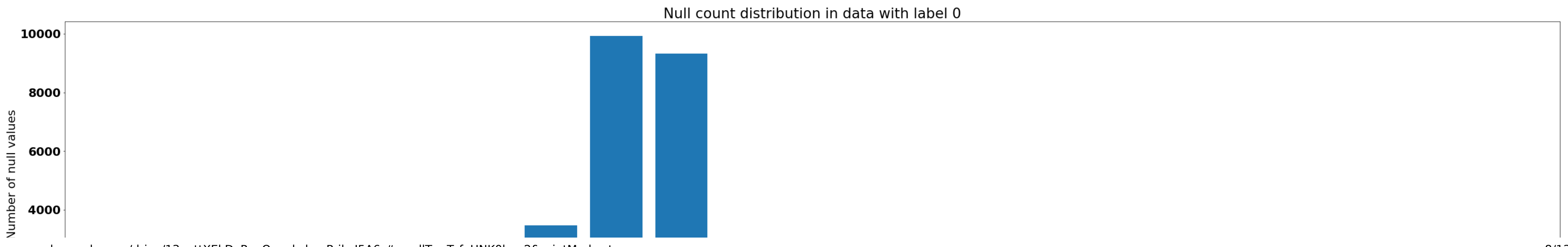
```
"""Null bar plot for whole data with label 0"""

coloumns = list(null_0.keys())
values = list(null_0.values())

col = []
val = []

col.append(coloumns[0:50])
val.append(values[0:50])

col.append(coloumns[50:100])
val.append(values[50:100])

col.append(coloumns[100:150])
val.append(values[100:150])

col.append(coloumns[150:171])
val.append(values[150:171])



figure, axis = plt.subplots(4, 1, figsize=(50, 80), squeeze=False)

for i in range(0,4):

    axis[i,0].bar(col[i], val[i])
    axis[i,0].set_xticks(range(len(col[i])), col[i])
    axis[i,0].set_xticklabels(col[i],rotation=90)
    axis[i,0].set_xlabel("Sensors")
    axis[i,0].set_ylabel("Number of null values")
    axis[i,0].set_title("Null count distribution in data with label 0")


plt.subplots_adjust(hspace=1.2)
plt.show()
```

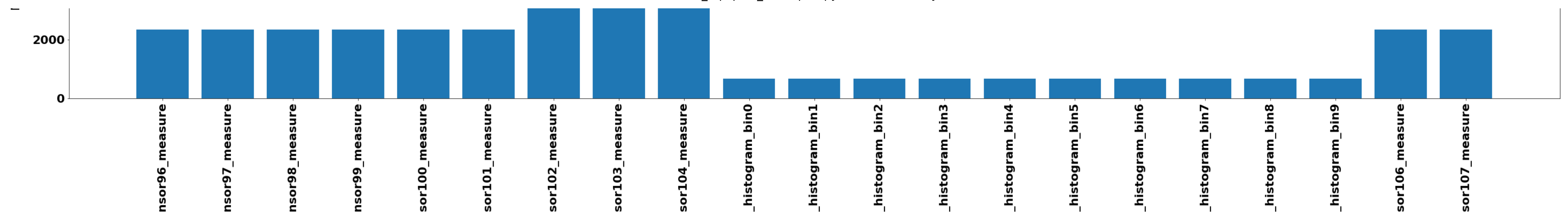
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:29: MatplotlibDeprecationWarning:

Passing the minor parameter of set_xticks() positionally is deprecated since Matplotlib 3.2; the parameter will become keyword-only two minor releases later.
```





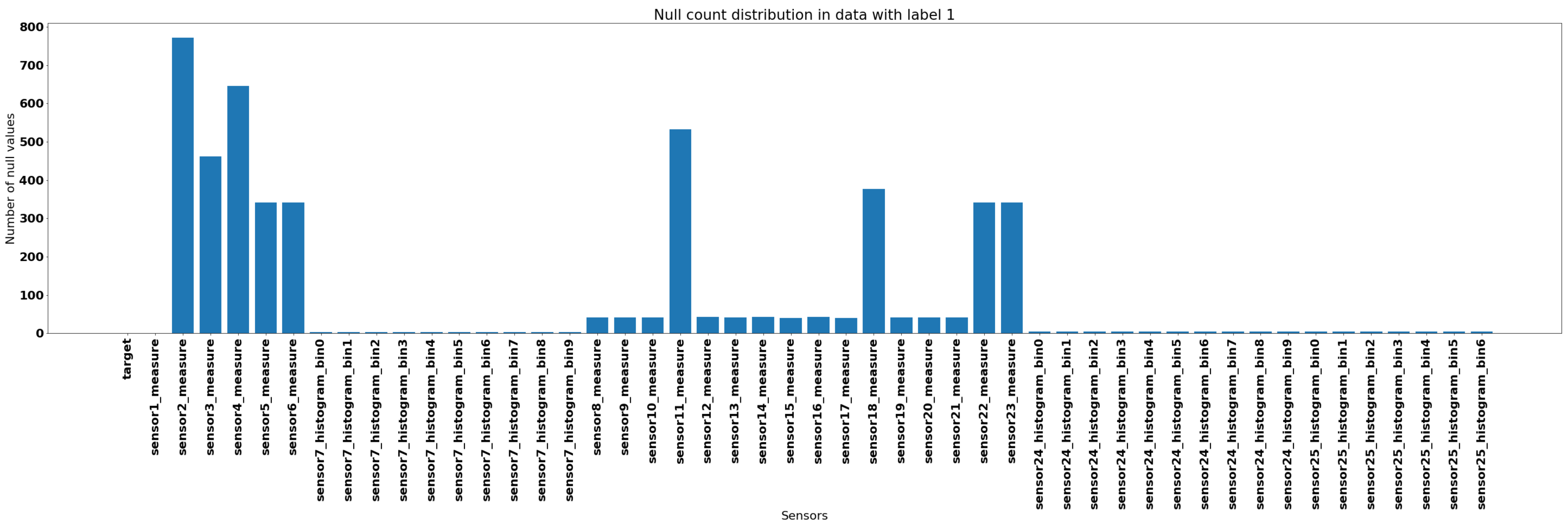


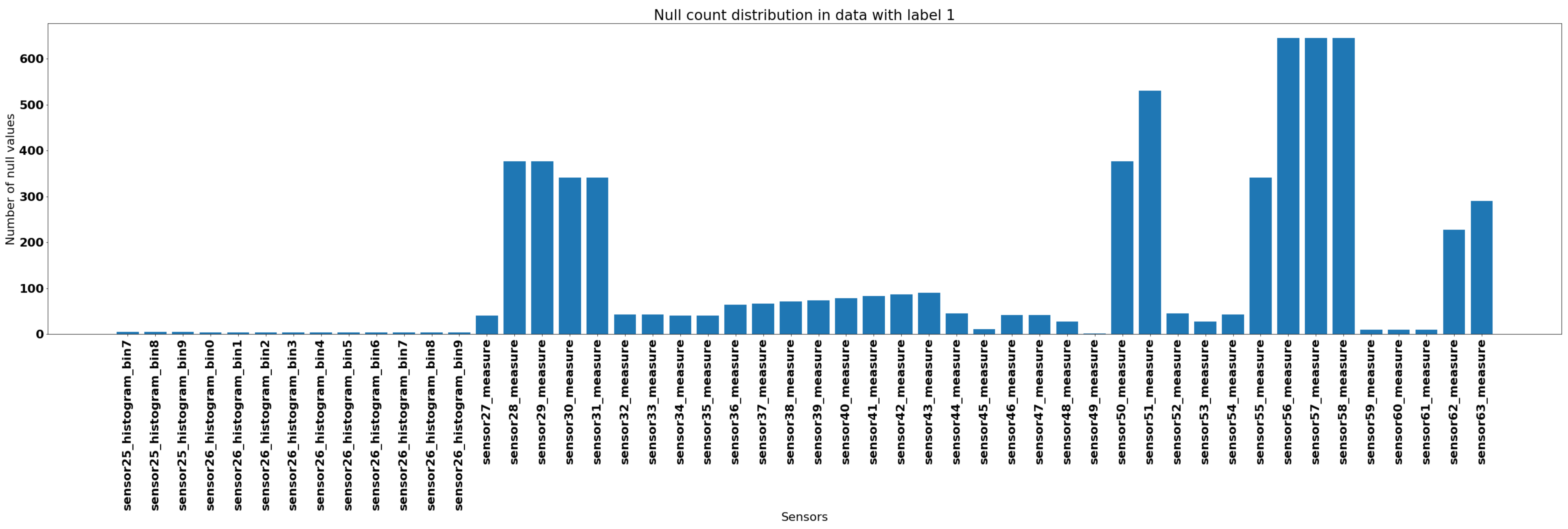


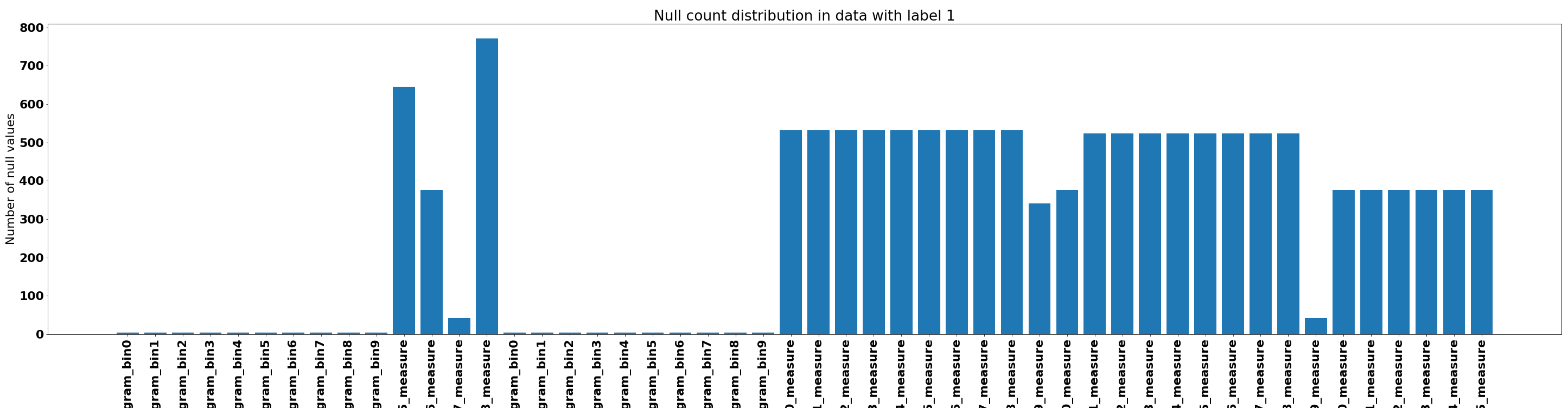


```
"""Null bar plot for whole data with label 1"""

coloumns = list(null_1.keys())
values = list(null_1.values())

col = []
val = []

col.append(coloumns[0:50])
val.append(values[0:50])

col.append(coloumns[50:100])
val.append(values[50:100])

col.append(coloumns[100:150])
val.append(values[100:150])

col.append(coloumns[150:171])
val.append(values[150:171])



figure, axis = plt.subplots(4, 1, figsize=(50, 80), squeeze=False)

for i in range(0,4):

    axis[i,0].bar(col[i], val[i])
    axis[i,0].set_xticks(range(len(col[i])), col[i])
    axis[i,0].set_xticklabels(col[i],rotation=90)
    axis[i,0].set_xlabel("Sensors")
    axis[i,0].set_ylabel("Number of null values")
    axis[i,0].set_title("Null count distribution in data with label 1")


plt.subplots_adjust(hspace=1.2)
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:29: MatplotlibDeprecationWarning:

Passing the minor parameter of set_xticks() positionally is deprecated since Matplotlib 3.2; the parameter will become keyword-only two minor releases later.
```



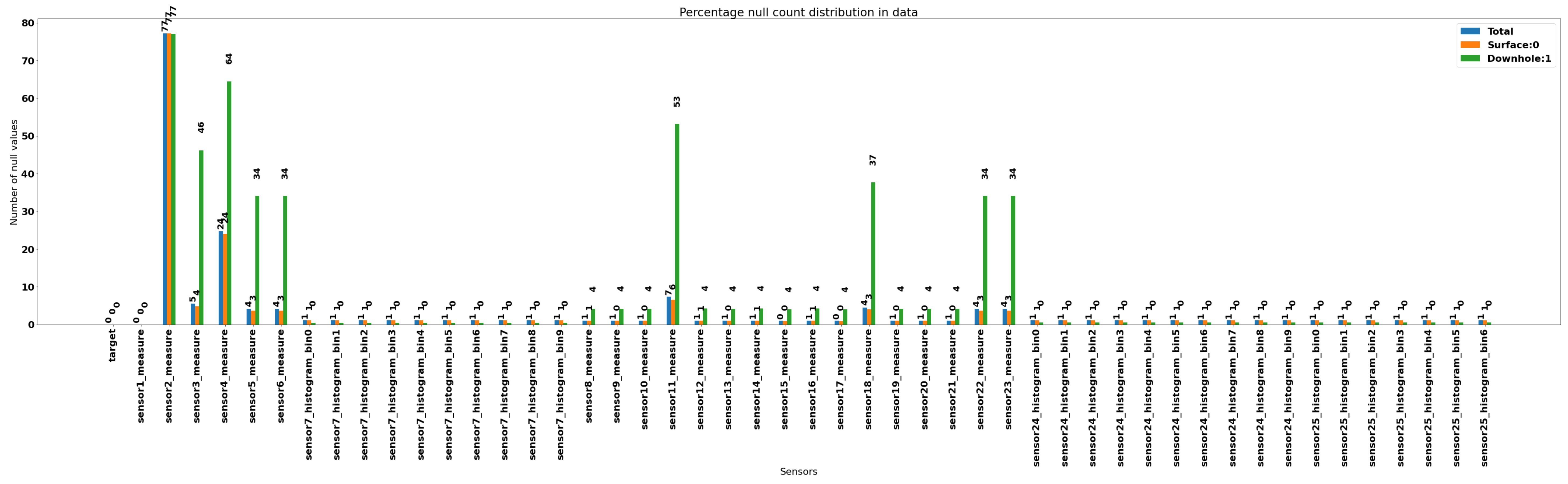


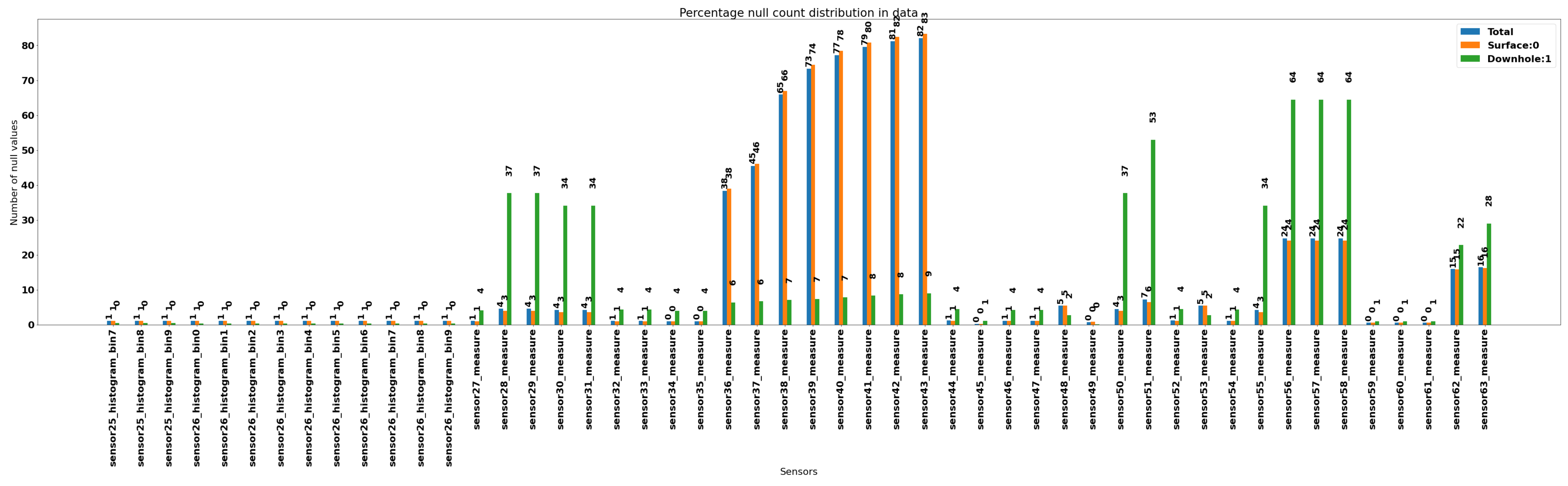


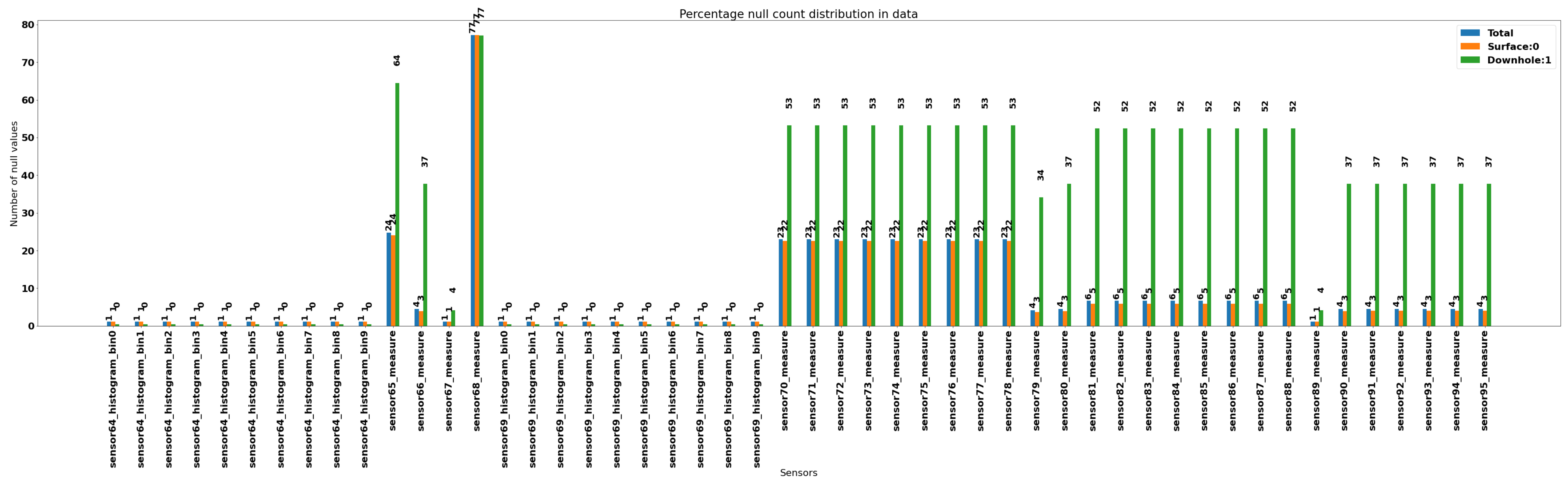
Comaprision of percentage wise null values

```
"""Percentage wise stacked plot"""

null_percent   = {key: ((value/len(df))*100) for key, value in null.items()}
null_0_percent = {key: ((value/len(df_0))*100) for key, value in null_0.items()}
null_1_percent = {key: ((value/len(df_1))*100) for key, value in null_1.items()}


#https://matplotlib.org/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py


coloumns = list(null_percent.keys())

values = list(null_percent.values())
values_0 = list(null_0_percent.values())
values_1 = list(null_1_percent.values())

col = []
val = []
val_0 = []
val_1 = []
```

```
col.append(coloumns[0:50])
val.append(values[0:50])
val_0.append(values_0[0:50])
val_1.append(values_1[0:50])

col.append(coloumns[50:100])
val.append(values[50:100])
val_0.append(values_0[50:100])
val_1.append(values_1[50:100])

col.append(coloumns[100:150])
val.append(values[100:150])
val_0.append(values_0[100:150])
val_1.append(values_1[100:150])

col.append(coloumns[150:171])
val.append(values[150:171])
val_0.append(values_0[150:171])
val_1.append(values_1[150:171])

rects = []

def autolabel(rects,i,loc):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        axis[i,0].annotate('{}'.format(int(height)),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, loc),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom',rotation=90,size=20)



figure, axis = plt.subplots(4, 1, figsize=(50, 80), squeeze=False)

for i in range(0,4):

    x = np.arange(len(col[i]))  # the label locations
    width = 0.15  # the width of the bars

    rects   =  axis[i,0].bar(x - (width)     , val[i]   ,width , label="Total")
    rects_0 =  axis[i,0].bar(x                  , val_0[i] ,width , label="Surface:0")
    rects_1 =  axis[i,0].bar(x  + (width)  , val_1[i] ,width , label="Downhole:1")

    axis[i,0].set_xticks(x)
    axis[i,0].set_xticklabels(col[i],rotation=90)
    axis[i,0].set_xlabel("Sensors")
    axis[i,0].set_ylabel("Number of null values")
    axis[i,0].set_title("Percentage null count distribution in data")
    axis[i,0].legend()

    autolabel(rects   , i , 5)
    autolabel(rects_0 , i , 25)
    autolabel(rects_1 , i , 40)

figure.tight_layout()
plt.subplots_adjust(hspace=1.2)
plt.show()
```







Observation

- In some coloumns nan distribution for downhole and surface is significantly diffrent

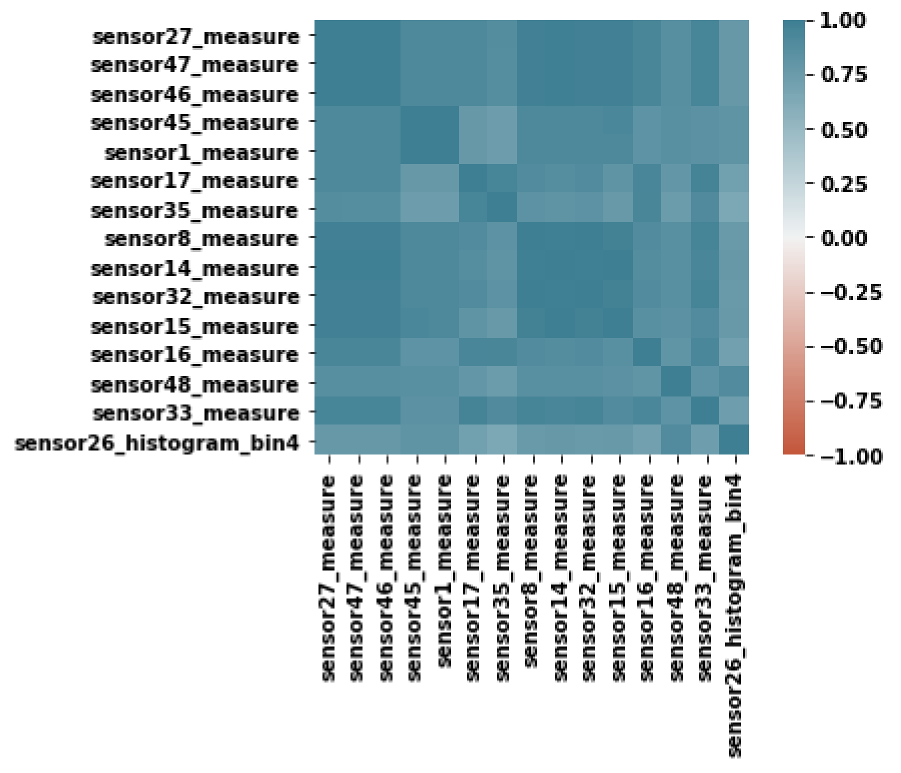
Conclusion

- Nan values contain some information , so we can use nan values to distingush between the class

Analysis of collinearty among feature

```
font = {'family' : 'DejaVu Sans',
        'weight' : 'bold',
        'size'   : 10}

matplotlib.rc('font', **font)
```

```
top_15_feature = list(get_highly_correlated_feature(dataframe=df,correlation="pearson",top_features=16,with_="target").keys())[1:]
df_top_15 = df.filter(top_15_feature, axis=1)

top_15_features_correlation = df_top_15.corr(method="pearson")

cmap = sns.diverging_palette(20, 220, n=255, as_cmap=True)

ax = sns.heatmap(top_15_features_correlation , vmin=-1, vmax=1, center=0 ,cmap=cmap ,square=True)
```



Observations

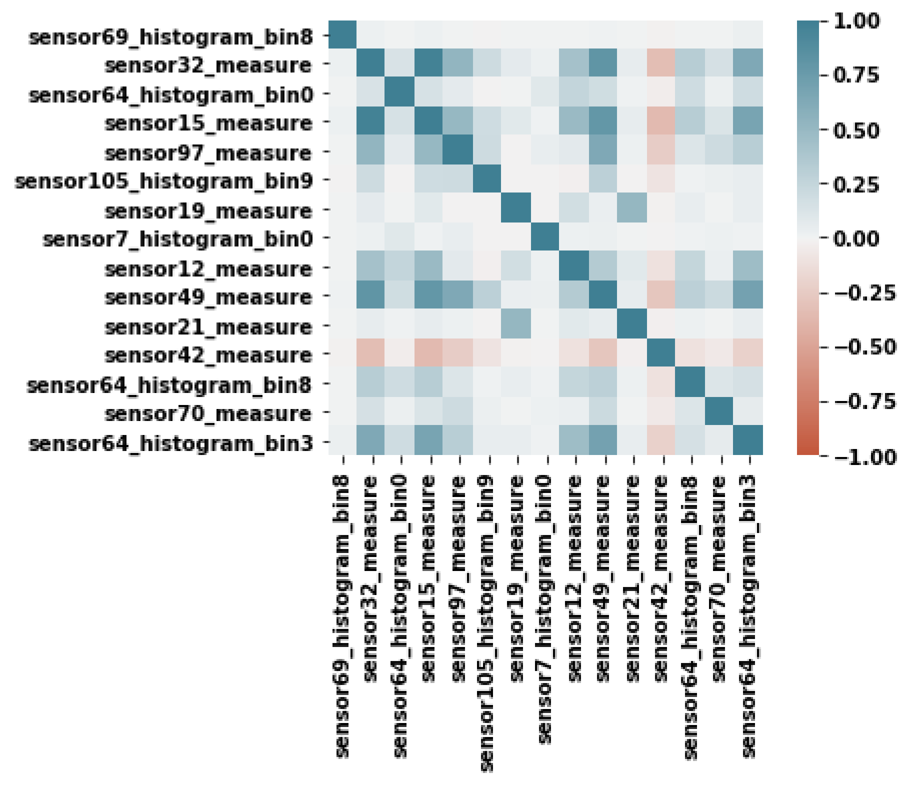
- Here we could see that , features that are highly correlated with class, they are correlated with themselves also

Conclusion

- We need to remove highly correlated features, beacuse these features do not help in prediction of class

Collinearty analysis among random features

```
coloumns = df.columns

def sampling_without_replacement(list_ = coloumns ,number_of_elements = 15):
    """
    pass list in first argument
    in second argument specify the number of elements you want

    it will return list of randomly, non repeated elements of given list
    """

    selected = []

    while(len(selected) < number_of_elements):
      selected.append(random.choice(list_))

      selected = list(set(selected))

    return selected

selected_15 = sampling_without_replacement(list_ = coloumns ,number_of_elements = 15)



df_selected_15 = df.filter(selected_15, axis=1)

selected_15_features_correlation = df_selected_15.corr(method="pearson")

cmap = sns.diverging_palette(20, 220, n=255, as_cmap=True)

ax = sns.heatmap(selected_15_features_correlation , vmin=-1, vmax=1, center=0 ,cmap=cmap ,square=True)
```



Observation

- few features are highly correlated, few features have very less correlation, and few are negatively correlated with each other.

Conclusion

- We got all type data, this is good for model.