In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Loading the data set

In [2]:
```
df=pd.read_csv('equip_failures_training_set.csv')
```

Displaying the first five observations of the data set

In [3]:
```
df.head()
```

Out[3]:

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

5 rows × 172 columns

In [4]:
```
df.shape
```

Out[4]:
```
(60000, 172)
```

The data has 172 sensor columns

In [5]:
```
df.dtypes
```

Out[5]:
```
id                          int64
target                      int64
sensor1_measure             int64
sensor2_measure            object
sensor3_measure            object
                            ...
sensor105_histogram_bin7   object
sensor105_histogram_bin8   object
sensor105_histogram_bin9   object
sensor106_measure          object
sensor107_measure          object
Length: 172, dtype: object
```

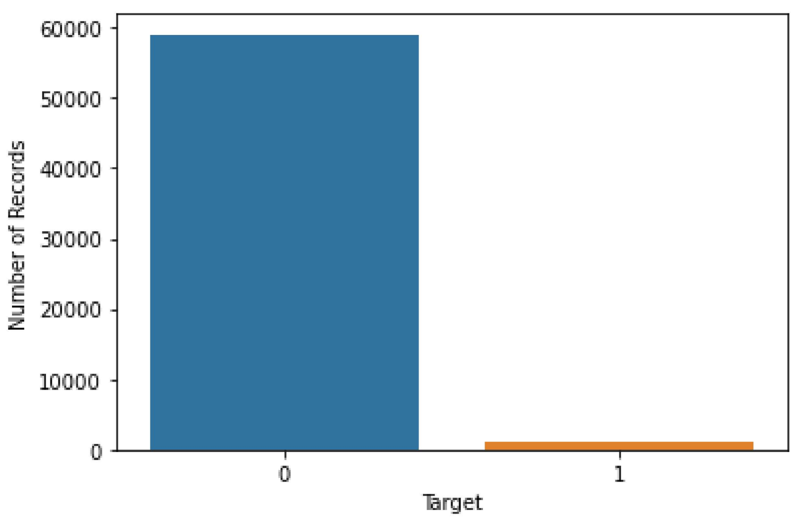
In [6]:
```
df.columns
```

Out[6]:
```
Index(['id', 'target', 'sensor1_measure', 'sensor2_measure', 'sensor3_measure',
       'sensor4_measure', 'sensor5_measure', 'sensor6_measure',
       'sensor7_histogram_bin0', 'sensor7_histogram_bin1',
       ...
       'sensor105_histogram_bin2', 'sensor105_histogram_bin3',
       'sensor105_histogram_bin4', 'sensor105_histogram_bin5',
       'sensor105_histogram_bin6', 'sensor105_histogram_bin7',
       'sensor105_histogram_bin8', 'sensor105_histogram_bin9',
       'sensor106_measure', 'sensor107_measure'],
      dtype='object', length=172)
```

countplot for target variable

In [7]:
```
sns.countplot(x='target', data = df)
plt.xlabel('Target')
plt.ylabel('Number of Records')
```

Out[7]:
```
Text(0, 0.5, 'Number of Records')
```



we can see that we have imbalanced data for the target variable .

Here we use F1 score metric instead of accuracy.

Exploratory Data Analysis[EDA]

In [8]:
```
df.dtypes
```

Out[8]:
```
id                          int64
target                      int64
sensor1_measure             int64
sensor2_measure            object
sensor3_measure            object
                            ...
sensor105_histogram_bin7   object
sensor105_histogram_bin8   object
sensor105_histogram_bin9   object
sensor106_measure          object
sensor107_measure          object
Length: 172, dtype: object
```

to convert the data type into float we should handle missing values first

checking and Treating missing values

In [9]:
```
df.isna().sum()
```

Out[9]:
```
id                          0
target                      0
sensor1_measure             0
sensor2_measure             0
sensor3_measure             0
                           ..
sensor105_histogram_bin7    0
sensor105_histogram_bin8    0
sensor105_histogram_bin9    0
sensor106_measure           0
sensor107_measure           0
Length: 172, dtype: int64
```

Replacing string 'na' with NaN values

In [10]:
```
df = df.replace('na',np.nan)
```

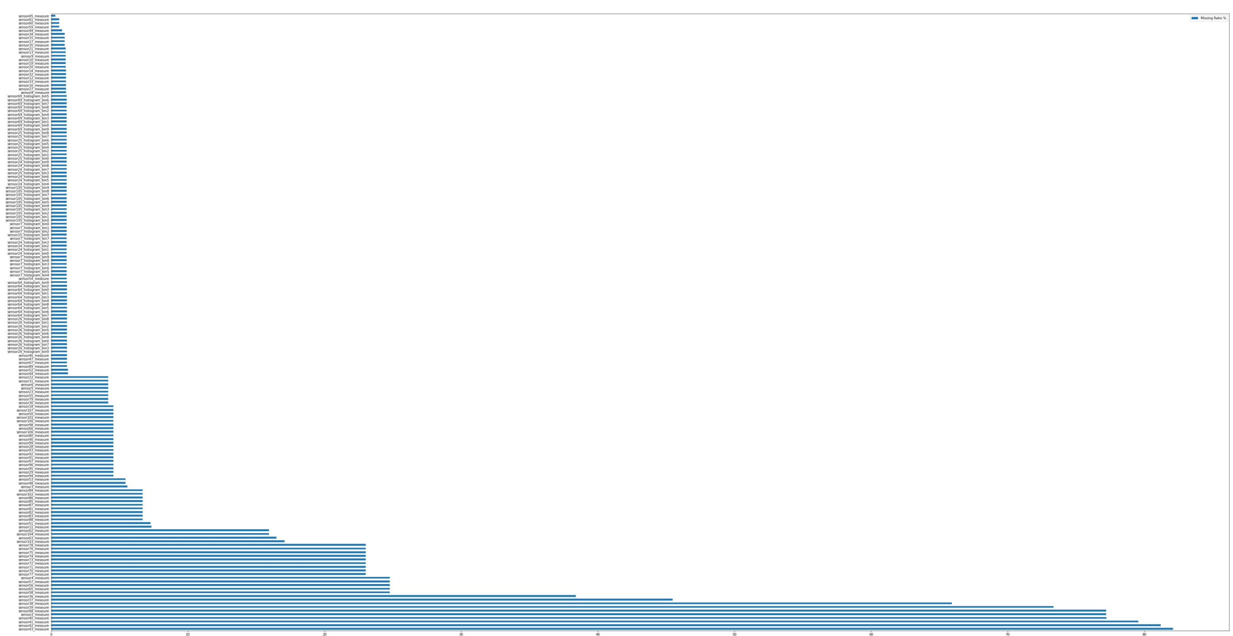
In [11]:
```
df.isna().sum()
```

Out[11]:
```
id                           0
target                       0
sensor1_measure              0
sensor2_measure          46329
sensor3_measure           3335
                         ...
sensor105_histogram_bin7   671
sensor105_histogram_bin8   671
sensor105_histogram_bin9   671
sensor106_measure         2724
sensor107_measure         2723
Length: 172, dtype: int64
```

Visual representation of missing ratio percentage

In [12]:
```
def plot_null(df: pd.DataFrame):
    if df.isnull().sum().sum() != 0:
        na_df = (df.isnull().sum() / len(df)) * 100
        na_df = na_df.drop(na_df[na_df == 0].index).sort_values(ascending=False)
        missing_data = pd.DataFrame({'Missing Ratio %' :na_df})
        missing_data.plot(kind = "barh")
        plt.show()
    else:
        print('No NAs found')
```

In [15]:
```
plot_null(df)
plot_width, plot_height = (60,35)
plt.rcParams['figure.figsize'] = (plot_width,plot_height)
```



In [ ]:
```
df_null=df.isnull().sum() / len(df) * 100
df_null.sort_values(ascending=False)[:15]
```

Removing columns which mostly have null values - more than 50%

In [16]:
```
df.drop(columns=['sensor43_measure', 'sensor42_measure', 'sensor41_measure', 'ser
```

In [17]:
```
df.shape
```

Out[17]:
```
(60000, 164)
```

Replacing NaN with median values

Since most of the obeservations in each sensor measure is close to 0 and rest of the observations have an extremely high value, imputing missing values using mean would result in incorrect high values. Hence we choose to impute the missing values using the median of each column

For most of the features, the percentage of NULL values are in the range of 0% to 20 % of their data

A very small majority of the features have NULL values in the percentage of 50% to 80% of their data.

These features might be redundant or do not have much information which might contribute towards the training of the model.

Those features can be dropped during the data preprocessing stage.

In [18]:
```
for col in df.columns:
    if col not in ['id','target']:
        df[col] = df[col].fillna(df[col].median())
```

In [19]:
```
plot_null(df)
```

```
No NAs found
```

Converting all measures to numerical data type

In [20]:
```
for col in df.columns:
    if col not in ['id', 'target']:
        df[col] = df[col].astype(np.float)
```

In [21]:
```
df.dtypes
```

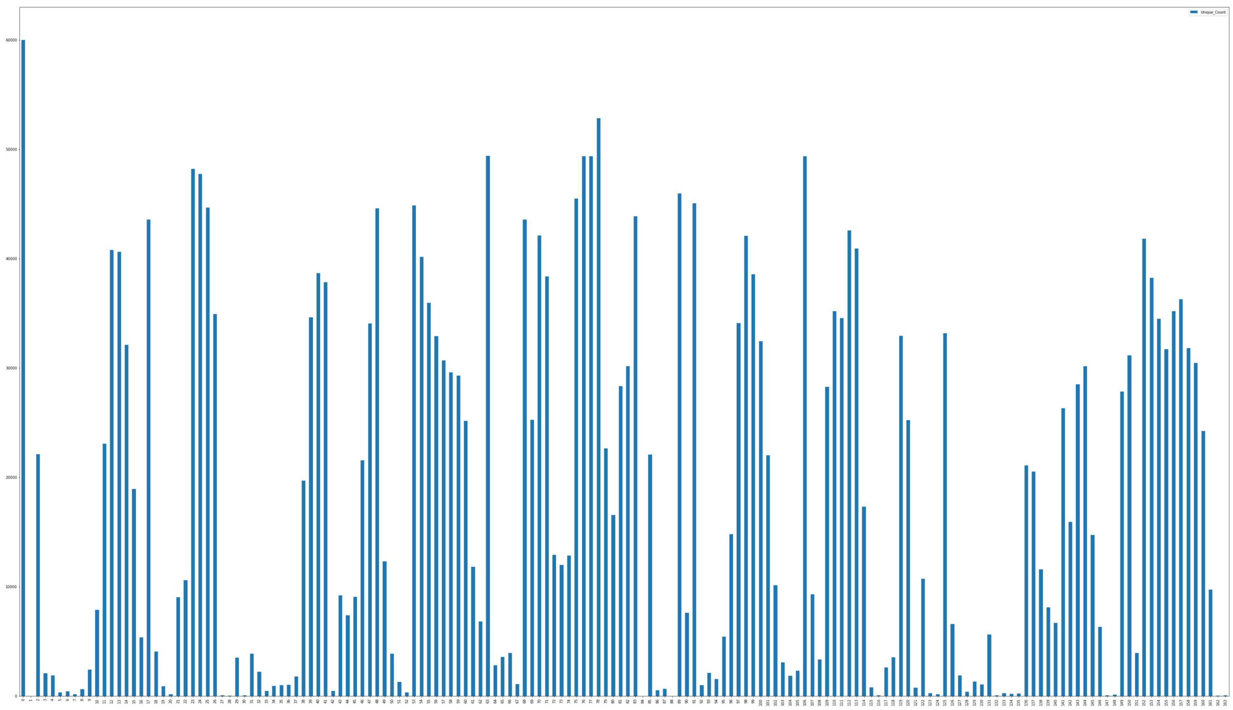
Out[21]:
```
id                              int64
target                          int64
sensor1_measure               float64
sensor3_measure               float64
sensor4_measure               float64
                               ...
sensor105_histogram_bin7      float64
sensor105_histogram_bin8      float64
sensor105_histogram_bin9      float64
sensor106_measure             float64
sensor107_measure             float64
Length: 164, dtype: object
```

Checking the columns which have less than 3 unique values across the data set because columns with constant values do not support our model's prediction

Checking for unique values in each column in full dataset

In [22]:
```
unique_data = df.nunique().reset_index()
unique_data.columns = ['Name','Unique_Count']
plt.figure (figsize = (180,10))
unique_data.plot(kind='bar')
plot_width, plot_height = (16,28)
plt.rcParams['figure.figsize'] = (plot_width,plot_height)
```

```
<Figure size 12960x720 with 0 Axes>
```



In [23]:
```
unique_data
```

Out[23]:

	Name	Unique_Count
0	id	60000
1	target	2
2	sensor1_measure	22095
3	sensor3_measure	2061
4	sensor4_measure	1886
...	...	...
159	sensor105_histogram_bin7	30469
160	sensor105_histogram_bin8	24213
161	sensor105_histogram_bin9	9724
162	sensor106_measure	28
163	sensor107_measure	49

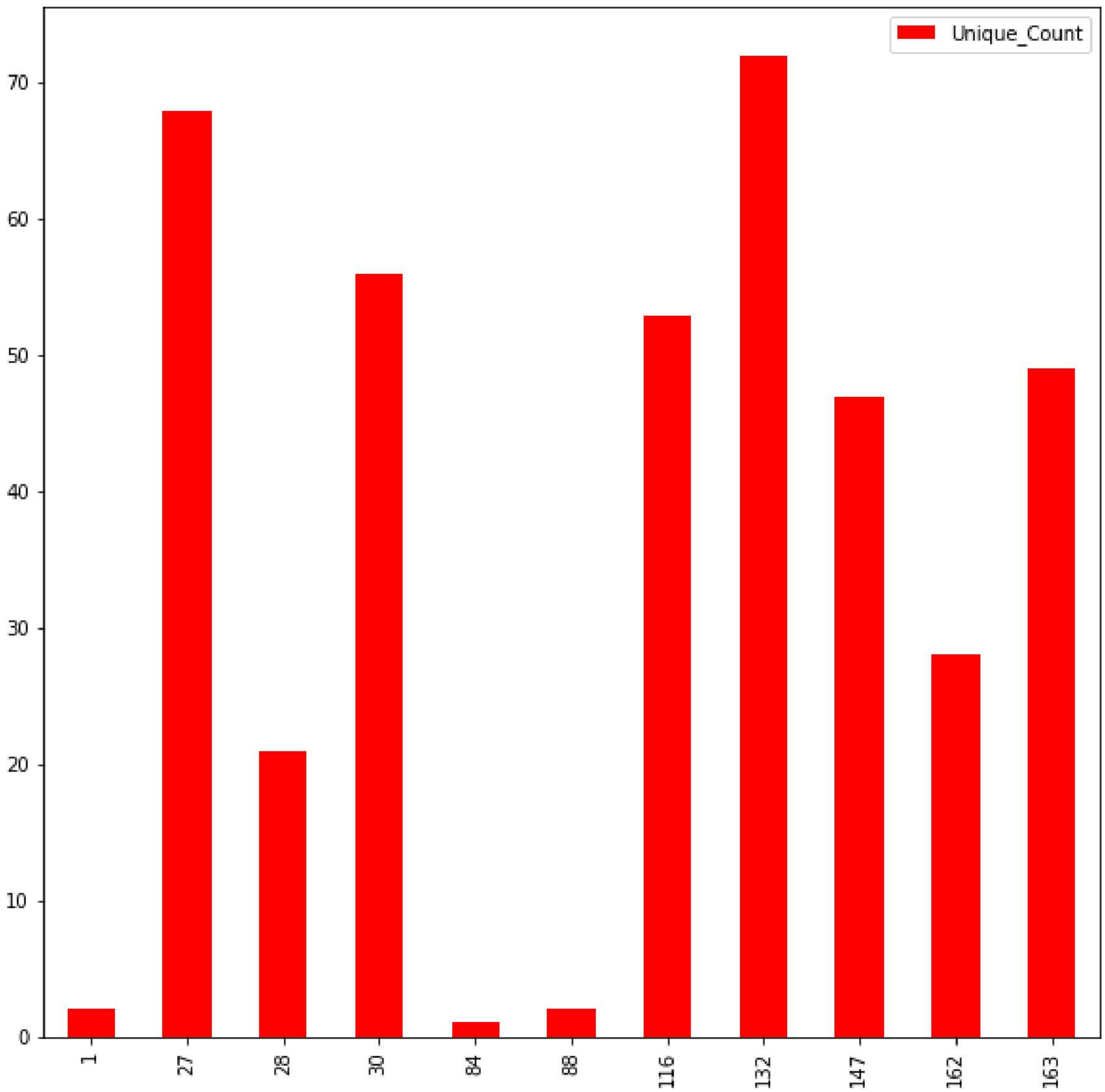
164 rows × 2 columns

Checking the columns which have less than 3 unique values across the

Checking the columns which have less than 3 unique values across the data set because columns with constant values do not support our model's prediction

In [27]:
```
unique_data[unique_data.Unique_Count<=100].plot(kind='bar',color="r")

plot_width, plot_height = (10,10)
plt.rcParams['figure.figsize'] = (plot_width,plot_height)
```



In [28]:
```
unique_data[unique_data.Unique_Count<=3]
```

Out[28]:

	Name	Unique_Count
1	target	2
84	sensor54_measure	1
88	sensor58_measure	2

Dropping sensor54_measure, 'sensor58_measure', as it has constant values

In [29]:
```
df.drop(columns=['sensor54_measure','sensor58_measure',], axis=1, inplace=True)
```

In [30]:
```
df.shape
```

Out[30]: (60000, 162)

In [31]:
```
df.describe(include="all")
```

Out[31]:

	id	target	sensor1_measure	sensor3_measure	sensor4_measure	sensor5
count	60000.000000	60000.000000	6.000000e+04	6.000000e+04	6.000000e+04	600
mean	30000.500000	0.016667	5.933650e+04	3.362258e+08	1.434383e+05	
std	17320.652413	0.128020	1.454301e+05	7.767625e+08	3.504525e+07	1
min	1.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	
25%	15000.750000	0.000000	8.340000e+02	2.000000e+01	4.200000e+01	
50%	30000.500000	0.000000	3.077600e+04	1.520000e+02	1.260000e+02	
75%	45000.250000	0.000000	4.866800e+04	8.480000e+02	2.920000e+02	
max	60000.000000	1.000000	2.746564e+06	2.130707e+09	8.584298e+09	210

8 rows × 162 columns

Dropping features having more than 65% of their data points as 0

In [34]:
```
redundant=[]
print("These features have more than or equal to 65% of their datapoints as 0 whi
for i in df.columns:
    if (df[i]==0).sum()>54000 and i!='target':
        df.drop(i,inplace=True,axis=1)
        redundant.append(i)
        print(i)
```

```
These features have more than or equal to 65% of their datapoints as 0 which do
es not contribute much to training :
sensor5_measure
sensor6_measure
sensor7_histogram_bin0
sensor7_histogram_bin1
sensor7_histogram_bin2
sensor9_measure
sensor11_measure
sensor18_measure
sensor19_measure
sensor20_measure
sensor21_measure
sensor24_histogram_bin0
sensor24_histogram_bin1
sensor24_histogram_bin2
sensor24_histogram_bin3
sensor24_histogram_bin4
sensor24_histogram_bin9
sensor25_histogram_bin8
sensor25_histogram_bin9
sensor64_histogram_bin0
sensor69_histogram_bin9
sensor76_measure
sensor81_measure
sensor82_measure
sensor85_measure
sensor86_measure
sensor87_measure
sensor88_measure
sensor100_measure
sensor101_measure
sensor106_measure
sensor107_measure
```

Having many zeroes implies that there is less variability among the features.Less variability implies that there isn't much information for the model to learn from the feature.

In [35]:
```
df.shape
```

Out[35]:
```
(60000, 130)
```

Correlation Matrix used to understand relationship between two or more continuous variables

In [37]:
```
df.corr()
```

Out[37]:

	id	target	sensor1_measure	sensor3_measure	sensor4_meas
id	1.000000	-0.012163	-0.008329	0.000468	0.003
target	-0.012163	1.000000	0.536978	-0.050996	-0.000
sensor1_measure	-0.008329	0.536978	1.000000	-0.063876	-0.001
sensor3_measure	0.000468	-0.050996	-0.063876	1.000000	-0.001
sensor4_measure	0.003766	-0.000530	-0.001590	-0.001765	1.000
...	...	...	...	...	
sensor105_histogram_bin5	-0.005130	0.485831	0.724399	-0.039648	-0.001
sensor105_histogram_bin6	-0.004598	0.415300	0.724102	-0.047105	-0.001
sensor105_histogram_bin7	0.000300	0.160284	0.603888	-0.048648	-0.000
sensor105_histogram_bin8	-0.001446	0.235401	0.469836	-0.003714	-0.001
sensor105_histogram_bin9	0.004272	0.115925	0.247149	0.013764	-0.000

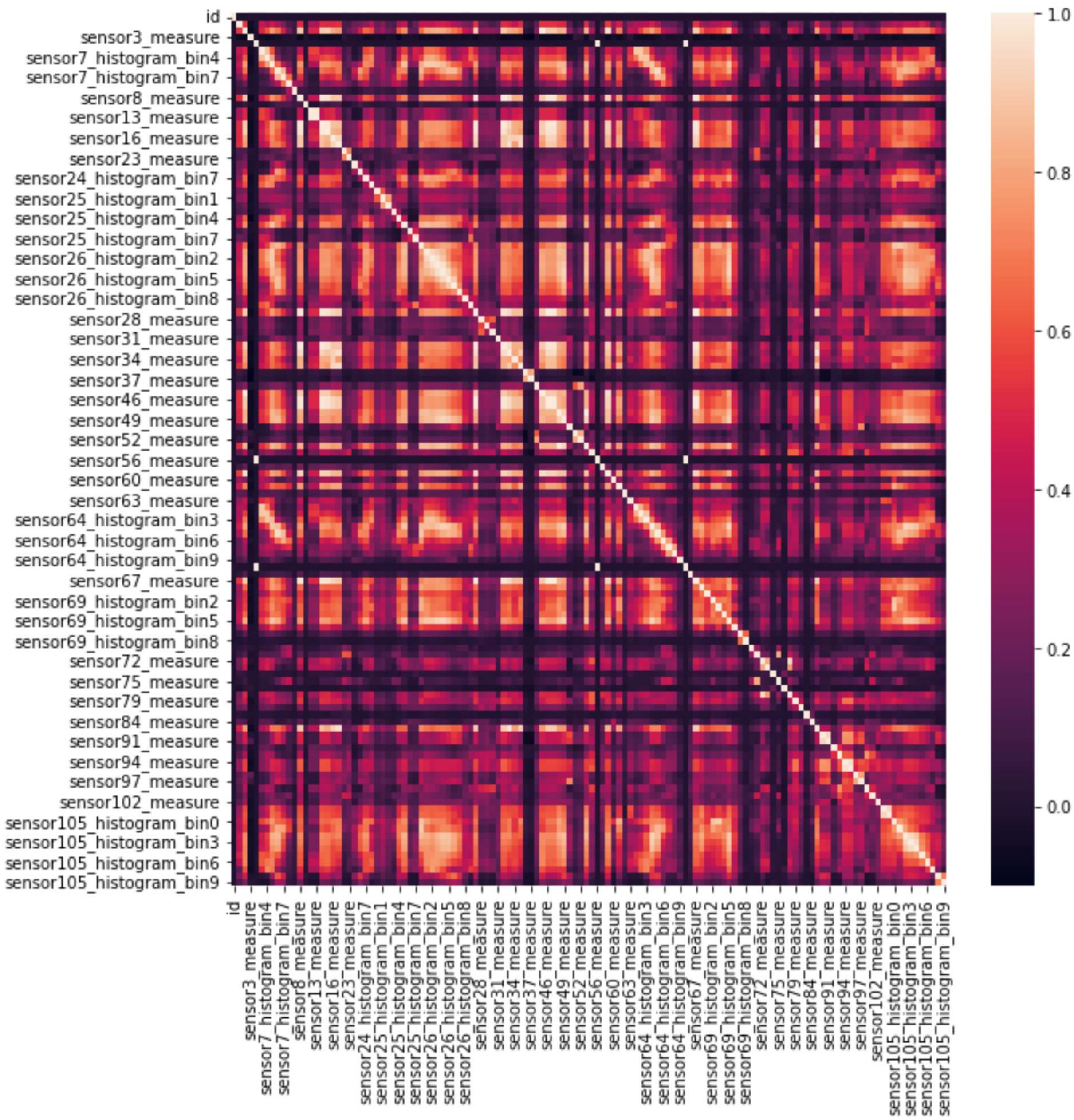
130 rows × 130 columns

In [38]:
```
matrix = df.corr() #Methods method = 'spearman'

#Heat Map - Visual representation of the correlation plot
#fig, ax = plt.subplots(figsize = (30, 18))
sns.heatmap(matrix)

plt.show()

plot_width, plot_height = (10,10)
plt.rcParams['figure.figsize'] = (plot_width,plot_height)
```



In [39]:
```
# Dropping id feature as it does not affect model performance
df.drop(columns=['id'], axis=1, inplace=True)
```

In [40]:
```
df.shape
```

Out[40]: (60000, 129)

Model fitting

In [41]:
```
df.head()
```

Out[41]:

	target	sensor1_measure	sensor3_measure	sensor4_measure	sensor7_histogram_bin3	sensor7_
0	0	76698.0	2.130706e+09	280.0	0.0	
1	0	33058.0	0.000000e+00	126.0	0.0	
2	0	41040.0	2.280000e+02	100.0	0.0	
3	0	12.0	7.000000e+01	66.0	318.0	
4	0	60874.0	1.368000e+03	458.0	0.0	

5 rows × 129 columns

In [42]:
```
#Evaluation and HYpertuning
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import f1_score

X=df.drop(['target'],axis=1)
y=df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_sta
```

In [43]:
```
X_train.shape
```

Out[43]: (48000, 128)

In [44]:
```
X_test.shape
```

Out[44]: (12000, 128)

In [45]:
```
y_test.shape
```

Out[45]: (12000,)

In [46]:
```
y_train.shape
```

Out[46]: (48000,)

Model fitting classification algorithms

Logistic Regression

In [47]:
```
#classification algorithms
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

Out[47]: LogisticRegression()

In [48]:
```
#Predicting validation set results
y_pred1 = lr.predict(X_test)
```

Checking f1 score based on validation set results

In [49]:
```
f1_score(y_test, y_pred1)
```

Out[49]: 0.6079545454545455

Random Forest

In [50]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, n_jobs=4,random_state=0)
rf.fit(X_train, y_train)
```

Out[50]:
```
RandomForestClassifier(n_jobs=4, random_state=0)
```

In [51]:
```
#Predicting validation set results
y_pred = rf.predict(X_test)
```

In [52]:
```
#Checking f1 score based on validation set results
f1_score(y_test, y_pred)
```

Out[52]:
```
0.7507692307692307
```

From the results, we see that Random Forest classifier performs the best on our validation set. Hence we will be using this model to predict our test set results

In [53]:
```
Prediction=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
Prediction
```

Out[53]:

	Actual	Predicted
50767	0	0
2693	0	0
943	0	0
29296	0	0
4605	0	0
...	...	...
10433	0	0
34793	0	0
39606	0	0
8080	0	0
9410	0	0

12000 rows × 2 columns

Important Variables

In [54]:
```
# Sort the feature importance in descending order
importances = rf.feature_importances_
sorted_indices = np.argsort(importances)[::-1]
```

In [55]:
```
importances = list(rf.feature_importances_)
feature_list=X_train.columns
d = {'FeatureLabels':feature_list,'Importances':importances} # dictionary data ty

df1=pd.DataFrame(d)
df1.sort_values(['Importances'], ascending=False)[:10]
```

Out[55]:

	FeatureLabels	Importances
50	sensor35_measure	0.034048
3	sensor7_histogram_bin3	0.030711
66	sensor59_measure	0.030166
12	sensor12_measure	0.028800
13	sensor13_measure	0.028612
68	sensor61_measure	0.026737
71	sensor64_histogram_bin1	0.023260
123	sensor105_histogram_bin5	0.023255
55	sensor46_measure	0.023139
56	sensor47_measure	0.023113

prediction on unseen data using RF for important variables

In [59]:
```
a=df[['sensor35_measure','sensor61_measure','sensor13_measure','sensor12_measure'
b=df['target']
```

In [60]:
```
from sklearn.ensemble import RandomForestClassifier
rf2 = RandomForestClassifier(n_estimators=100,n_jobs=4,random_state=0).fit(a, b)
```

In [61]:
```
df1=pd.read_csv('equip_failures_test_set.csv')
df1.head()
```

Out[61]:

	id	sensor1_measure	sensor2_measure	sensor3_measure	sensor4_measure	sensor5_measure
0	1	66888	na	2130706438	332	0
1	2	91122	na	na	na	0
2	3	218924	na	na	na	na
3	4	16	0	30	28	0
4	5	39084	na	1054	1032	0

5 rows × 171 columns

In [62]:
```
new=df1[['sensor35_measure','sensor61_measure','sensor13_measure','sensor12_measu
new.head()
```

Out[62]:

	sensor35_measure	sensor61_measure	sensor13_measure	sensor12_measure	sensor17_measure
0	1026922	621043.2	0	0	1359980
1	564314	583557.12	0	0	525024
2	1613280	1113039.36	1351044	804746	1135476
3	1794	2857.92	0	0	712
4	295666	283138.56	0	0	228912

In [63]:
```
new.shape
```

Out[63]: (16001, 11)

In [64]:
```
# Replacing string 'na' with NaN values

new = new.replace('na',np.nan)

# Replacing NaN with median values

for col in new.columns:
    if col not in ['id']:
        new[col] = new[col].fillna(new[col].median())
```

In [65]:
```
y_pred = rf2.predict(new)
```

In [66]:
```
Prediction=pd.DataFrame({'Predicted':y_pred})
Prediction
```

Out[66]:

	Predicted
0	0
1	0
2	0
3	0
4	0
...	...
15996	0
15997	0
15998	0
15999	0
16000	0

16001 rows × 1 columns

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: