PROJECT: Classifying Cow’s activities

AIM : Classify Cow’s activities into 9 categories based on Data collected from IMU SENSORS

Data

IMU Data (Accelerometer, Gyroscope, Magnetometer)

What is IMU?

Inertial measurement unit, used to describe a collection of measurement tools, when installed in some device, catches movement with the help of accelerometer, gyroscope and magnetometer, in 3d space.

Variable names:-

- acc_x,acc_y,acc_z: accelerometer output for all 3 dimensions movement.
- gyr_x,gyr_y,gyr_z: gyroscope outputs, it measures rotation, rotation rate (angular velocity).
- mag_x,mag_y,mag_z: magnetometer outputs, catches magnetic field around the device.
- All three (Acc, Gyr, Mag) gives output in different SI Units i.e The scale for all three are different, so Data must be normalized

Classes and their Encoded values:-

- eating = 1
- drinking = 2
- walking = 3
- standing =4
- lying = 5
- ruminating standing = 6
- ruminating lying = 7
- grooming = 8
- idle/other = 9

Notebook Contents :

1. Dataset Information
2. Exploratory Data Analysis (EDA)
3. Feature Engineering
4. Modeling
5. Conclusion

1. Dataset Information

In [1]:
```
#importing the common libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#importing the dataset
drinking = pd.read_csv(r'D:\dsap class\project\COW DATASET\Classify cow positions usir
eating = pd.read_csv(r'D:\dsap class\project\COW DATASET\Classify cow positions using
walking = pd.read_csv(r'D:\dsap class\project\COW DATASET\Classify cow positions using
grooming = pd.read_csv(r'D:\dsap class\project\COW DATASET\Classify cow positions usir
idle = pd.read_csv(r'D:\dsap class\project\COW DATASET\Classify cow positions using Ma
lying = pd.read_csv(r'D:\dsap class\project\COW DATASET\Classify cow positions using M
ruminating_lying = pd.read_csv(r'D:\dsap class\project\COW DATASET\Classify cow positi
ruminating_standing = pd.read_csv(r'D:\dsap class\project\COW DATASET\Classify cow pos
standing = pd.read_csv(r'D:\dsap class\project\COW DATASET\Classify cow positions usir
```

In [3]:
```
# concatenating all the csv files
df = pd.concat([drinking, eating, walking, grooming, idle, lying, ruminating_lying, ru
```

In [4]:
```
#viewing the concatenated dataset
df
```

Out[4]:

	time	acc_x	acc_y	acc_z	gyr_x	gyr_y	gyr_z	mag_x	mag_
0	1628079761	-0.187012	1.071289	0.321289	7.934570	-40.527344	17.028809	-526.5	1437.
1	1628079761	-0.137207	1.099121	0.294922	0.305176	-44.677734	15.930176	-528.0	1423.
2	1628079761	0.028320	1.053711	0.215820	-9.216309	-42.541504	10.681152	-510.0	1419.
3	1628079761	0.151856	0.960938	0.206543	-15.502930	-31.433105	4.943848	-529.5	1428.
4	1628079761	0.171387	0.883301	0.247070	-0.061035	-27.832031	-2.685547	-526.5	1420.
...	...	...	...	...	...	...	...	...	
2539801	1628397674	0.296875	0.908691	0.293457	0.000000	-1.953125	3.540039	-643.5	880.
2539802	1628397674	0.298828	0.915527	0.294434	0.610352	-2.624512	2.563477	-637.5	903.
2539803	1628397674	0.301270	0.911133	0.295410	1.037598	-2.746582	3.112793	-622.5	885.
2539804	1628397674	0.301758	0.920410	0.284668	2.014160	-2.868652	3.662109	-621.0	909.
2539805	1628397674	0.301270	0.911133	0.289062	0.915527	-2.319336	2.868652	-645.0	900.

12263524 rows × 11 columns

In [5]:
```
#to check the number of rows n columns
df.shape
```

Out[5]: (12263524, 11)

In [6]:
```
#checking for null values
df.isnull().sum()
```

Out[6]:
```
time     0
acc_x    0
acc_y    0
acc_z    0
gyr_x    0
gyr_y    0
gyr_z    0
mag_x    0
mag_y    0
mag_z    0
label    0
dtype: int64
```

2. Exploratory Data Analysis

In [7]:
```
#checking how many rows and columns are present
df.shape
```

Out[7]:
```
(12263524, 11)
```

In [8]:
```
#getting to know the column names
df.columns
```

Out[8]:
```
Index(['time', 'acc_x', 'acc_y', 'acc_z', 'gyr_x', 'gyr_y', 'gyr_z', 'mag_x',
       'mag_y', 'mag_z', 'label'],
      dtype='object')
```

In [9]:
```
#creating a function to create a table that has feature_name, dtype, missing values ar
def insights_table(df):
    summary = pd.DataFrame(df.dtypes,columns=['dtypes'])
    summary = summary.reset_index()
    summary['Feature_name'] = summary['index']
    summary = summary[['Feature_name','dtypes']]
    summary['Missing_values'] = df.isnull().sum().values
    summary['No. Uniques_values'] = df.nunique().values
    return summary
insights_table(df)
```

Out[9]:

	Feature_name	dtypes	Missing_values	No. Uniques_values
0	time	int64	0	106341
1	acc_x	float64	0	6843
2	acc_y	float64	0	9373
3	acc_z	float64	0	7552
4	gyr_x	float64	0	9525
5	gyr_y	float64	0	11236
6	gyr_z	float64	0	6473
7	mag_x	float64	0	1280
8	mag_y	float64	0	1920
9	mag_z	float64	0	1404
10	label	int64	0	9

Observation:

1. Missing Data : we don't have any missing data.
2. There is no object type data, all are either int or float.

In [10]:
```
df.describe()
```

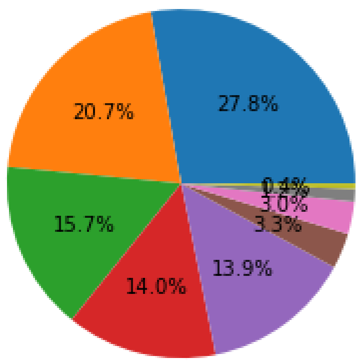
Out[10]:

	time	acc_x	acc_y	acc_z	gyr_x	gyr_y	
count	1.226352e+07	1.226352e+07	1.226352e+07	1.226352e+07	1.226352e+07	1.226352e+07	1.
mean	1.628243e+09	-2.807402e-02	6.211683e-01	1.081455e-01	8.722215e-01	-1.837340e+00	-6
std	1.205662e+05	2.622167e-01	7.084800e-01	1.888613e-01	1.282481e+01	2.014151e+01	1.
min	1.628067e+09	-1.599756e+01	-1.314160e+01	-1.599121e+01	-1.999756e+03	-1.998779e+03	-1.
25%	1.628106e+09	-2.182617e-01	8.281250e-01	3.173830e-02	-2.014160e+00	-5.676270e+00	-2.
50%	1.628312e+09	-2.490230e-02	9.370118e-01	1.230469e-01	8.544922e-01	-1.892090e+00	-6
75%	1.628335e+09	1.616211e-01	9.736329e-01	2.006836e-01	3.723145e+00	1.892090e+00	1.
max	1.628421e+09	1.051514e+01	1.182617e+01	1.599854e+01	1.999756e+03	1.999939e+03	1.

In [11]:
```
df['label'].value_counts()
```

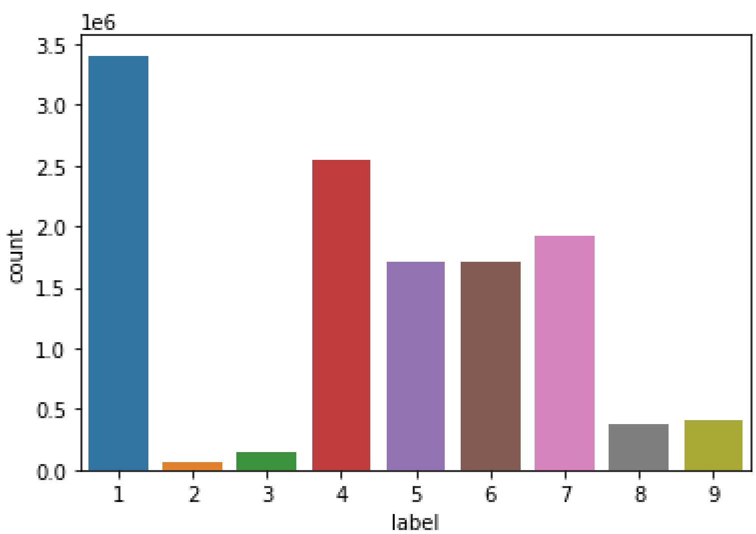
Out[11]:
```
1    3405702
4    2539806
7    1928968
5    1711687
6    1703683
9     405263
8     368607
3     145369
2      54439
Name: label, dtype: int64
```

In [12]:
```
#distribution of df
plt.pie(df['label'].value_counts(), autopct = '%1.1f%%')
plt.show()
```



In [13]:
```
#understanding our target variable
sns.countplot(df['label'])
```

Out[13]: <AxesSubplot:xlabel='label', ylabel='count'>



In [14]:
```
#checking the percentage of each class of the target variable present in the data
(df['label'].value_counts()/len(df['label']))*100
```

Out[14]:

```
1    27.770990
4    20.710246
7    15.729312
5    13.957546
6    13.892279
9     3.304621
8     3.005718
3     1.185377
2     0.443910
Name: label, dtype: float64
```

Observation:

The target variables are imbalanced!

Correlation

In [15]:

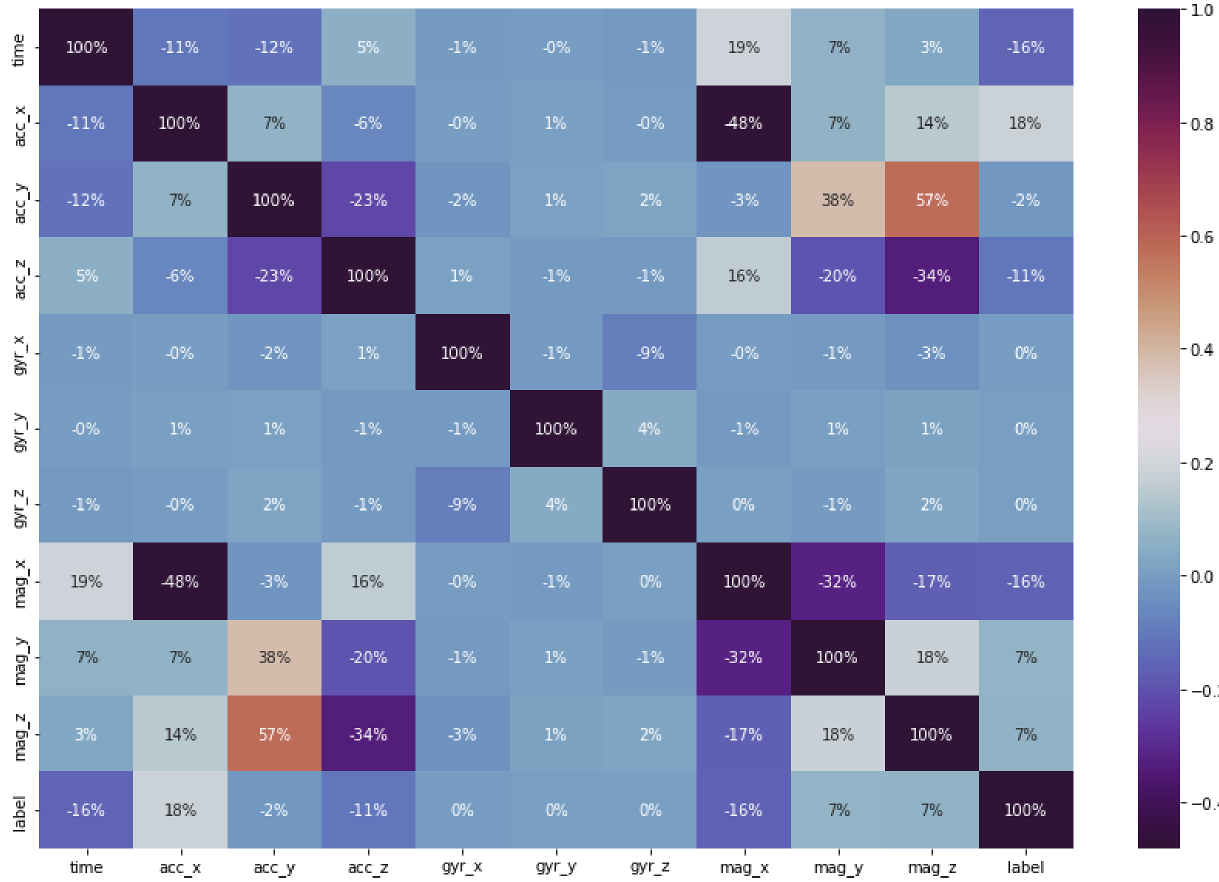
```
df.corr()
```

Out[15]:

	time	acc_x	acc_y	acc_z	gyr_x	gyr_y	gyr_z	mag_x	mag_
time	1.000000	-0.108821	-0.118527	0.054428	-0.005030	-0.002011	-0.008394	0.187664	0.06635
acc_x	-0.108821	1.000000	0.068235	-0.064783	-0.001430	0.006372	-0.004203	-0.481286	0.06821
acc_y	-0.118527	0.068235	1.000000	-0.227910	-0.023733	0.006674	0.020762	-0.025147	0.38258
acc_z	0.054428	-0.064783	-0.227910	1.000000	0.012389	-0.009027	-0.010025	0.157934	-0.19868
gyr_x	-0.005030	-0.001430	-0.023733	0.012389	1.000000	-0.009946	-0.088246	-0.004642	-0.00983
gyr_y	-0.002011	0.006372	0.006674	-0.009027	-0.009946	1.000000	0.041503	-0.008726	0.00592
gyr_z	-0.008394	-0.004203	0.020762	-0.010025	-0.088246	0.041503	1.000000	0.004679	-0.00641
mag_x	0.187664	-0.481286	-0.025147	0.157934	-0.004642	-0.008726	0.004679	1.000000	-0.32313
mag_y	0.066354	0.068210	0.382588	-0.198680	-0.009834	0.005924	-0.006413	-0.323132	1.00000
mag_z	0.032587	0.144633	0.574042	-0.337945	-0.033924	0.013075	0.022479	-0.168128	0.17949
label	-0.156390	0.181582	-0.018375	-0.109248	0.000211	0.002118	0.001470	-0.160687	0.07091

In [16]:

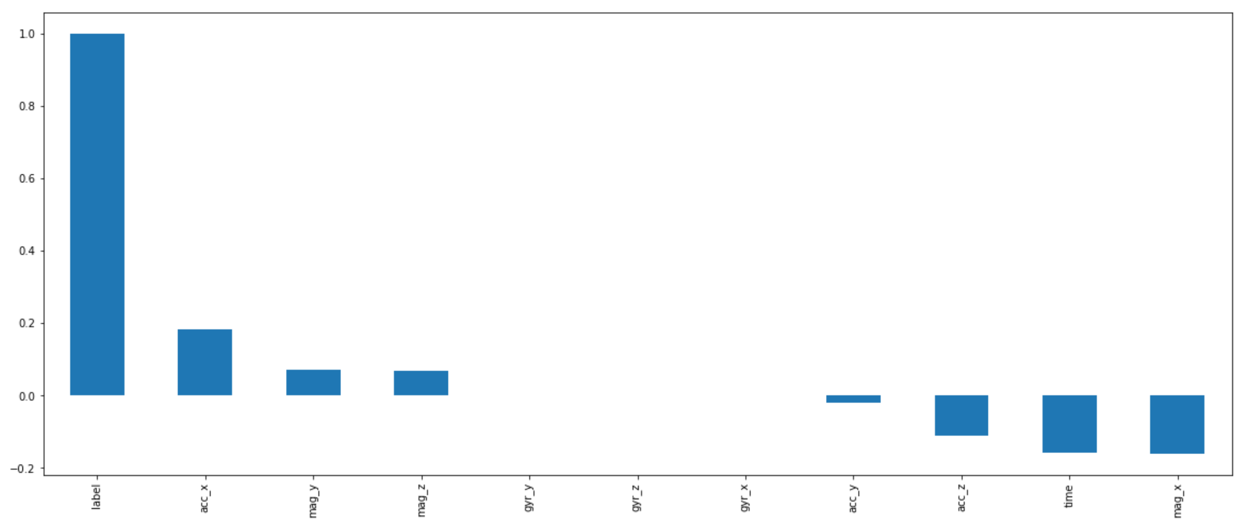
```
#checking the correlation between all the features
plt.figure(figsize=(15,10))
cor = df.corr()
sns.heatmap(cor,annot = True, cmap="twilight_shifted", fmt = '.0%' )
plt.show()
```



In [17]:

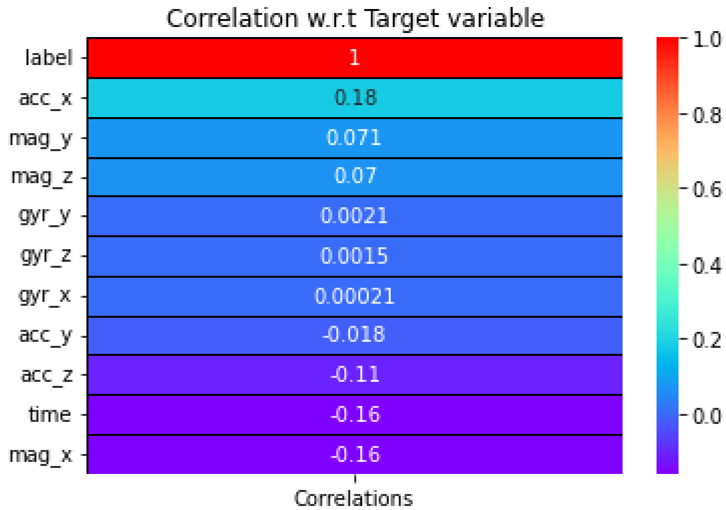
```
plt.figure(figsize=(20,8))
df.corr()['label'].sort_values(ascending = False).plot(kind='bar')
```

Out[17]: <AxesSubplot:>



In [18]:

```
corr = df.corrwith(df['label']).sort_values(ascending = False).to_frame()
corr.columns = ['Correlations']
sns.heatmap(corr,annot = True,cmap = 'rainbow',linewidths = 0.6,linecolor = 'black');
plt.title('Correlation w.r.t Target variable');
```



Seperating the dependent and independent variables

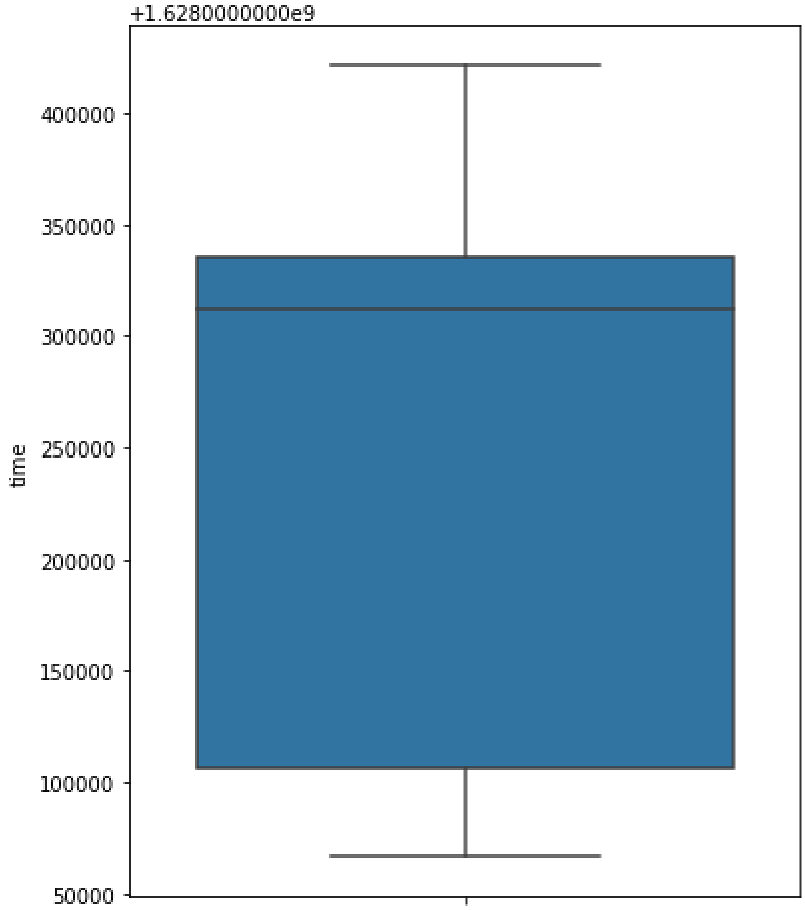
In [19]:
```
#seperating the x and y variables
x = df.drop('label', axis = 1) #independent features
#x = df.iloc[:,:-1]

y = df['label'] #dependent features
#y = df.iloc[:,-1]
```

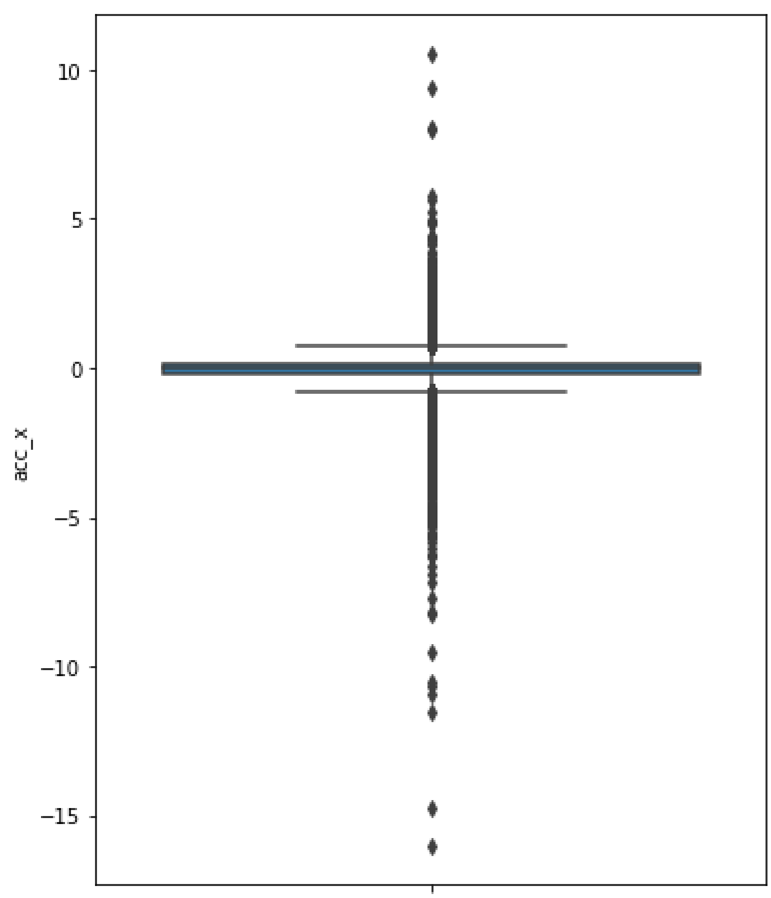
In [20]:
```
#defining a variable named column_name n giving it all the column names expect label
column_name = ['time','acc_x', 'acc_y', 'acc_z', 'gyr_x', 'gyr_y', 'gyr_z', 'mag_x','m
```

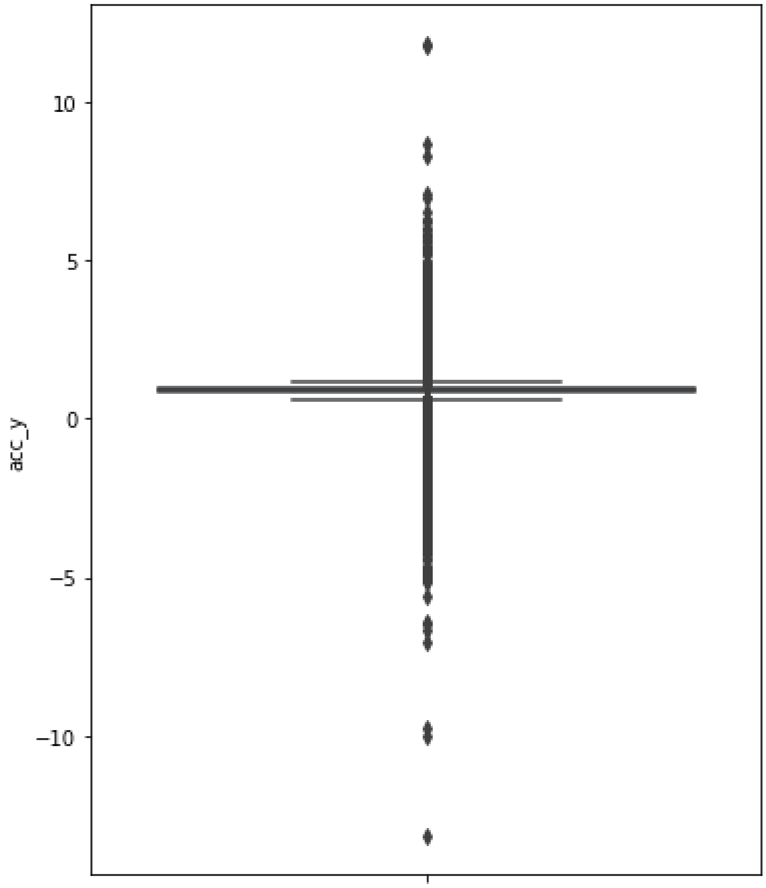
Checking for outliers

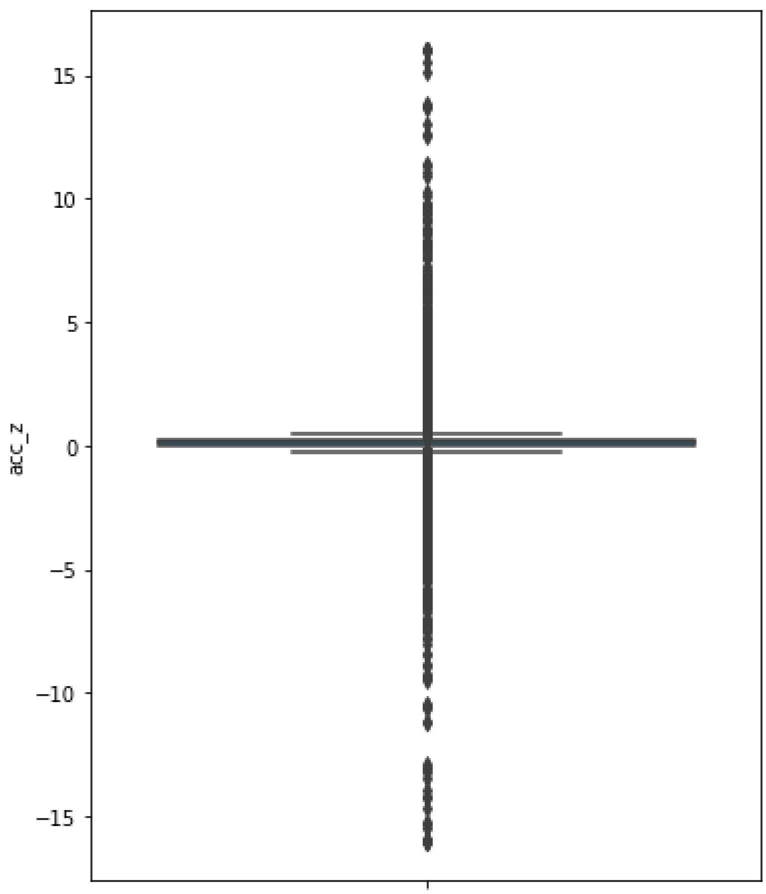
In [21]:
```
# Using a for loop inside a function to get the box plots(seaborn) of all the columns
def identify_outliers(give_df_name, give_column_name):
    for i in column_name:
        fig = plt.figure(figsize=(6,8))
        sns.boxplot(data = x, y = i)
plt.show()
```

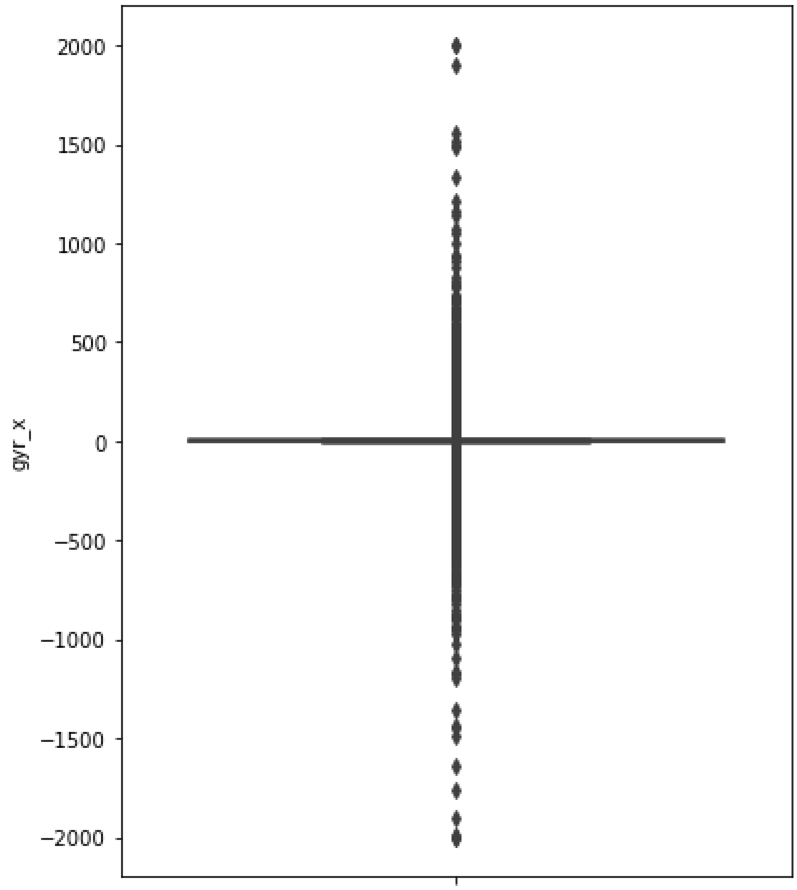
In [22]:
```
identify_outliers(x, column_name)
```

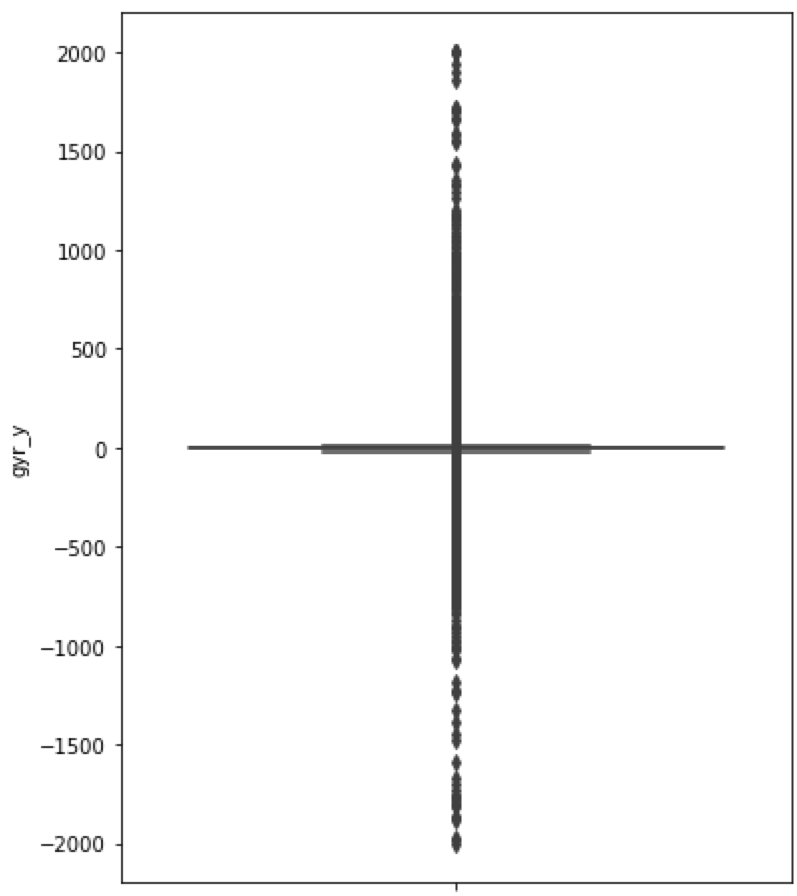

+1.6280000000e9
400000
350000
300000
250000
200000
150000
100000
50000
time

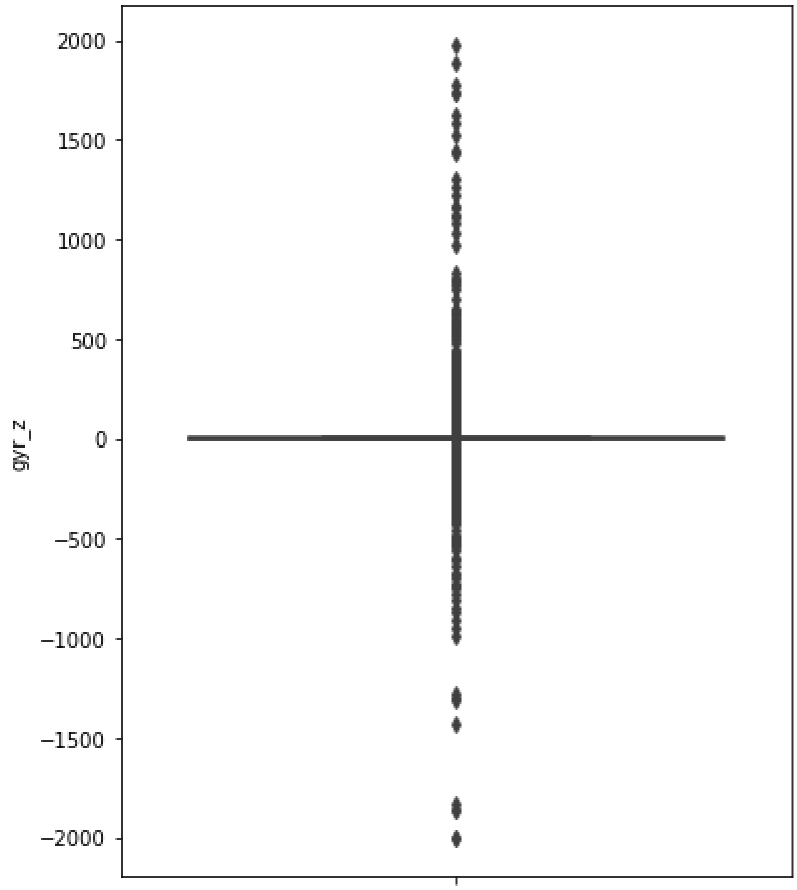

10
5
0
−5
−10
−15
acc_x

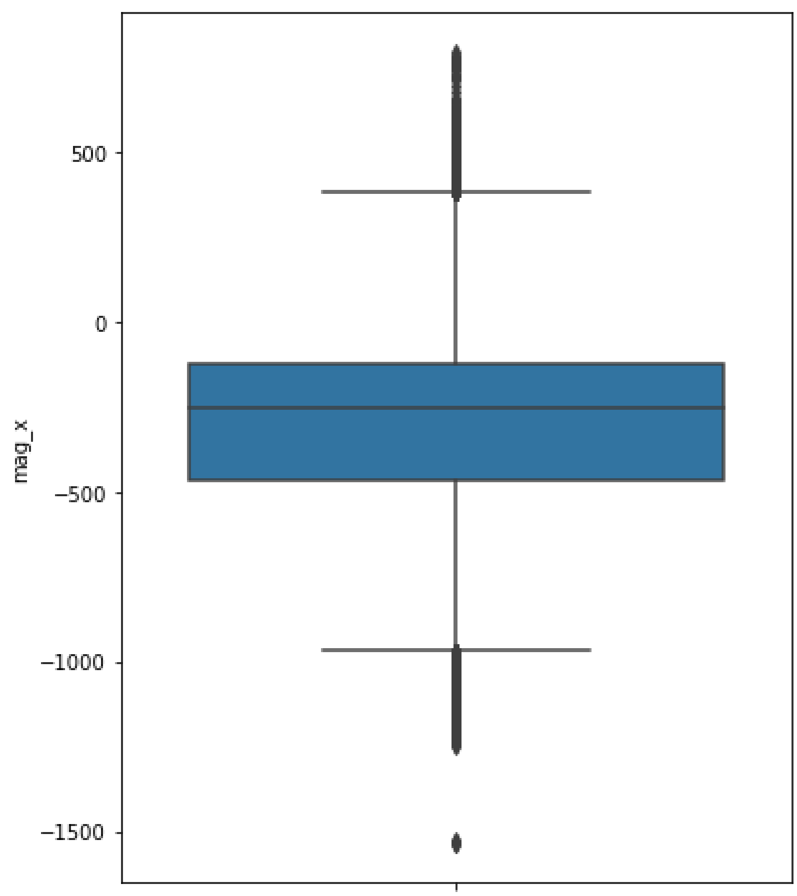

10
5
0
acc_y
−5
−10

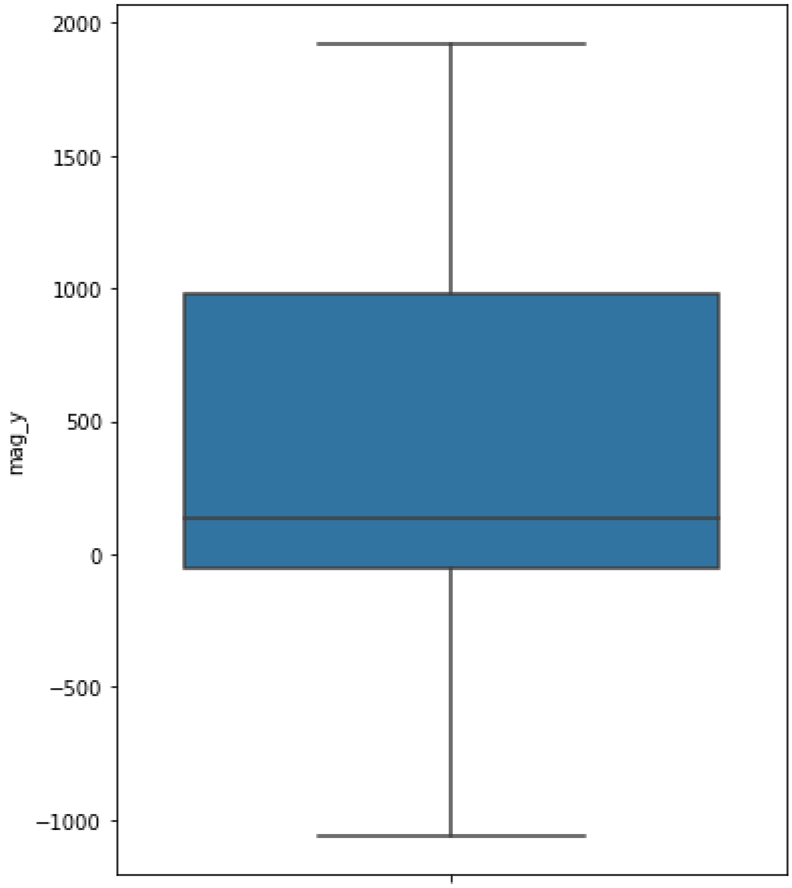

15
10
5
0
acc_z
−5
−10
−15

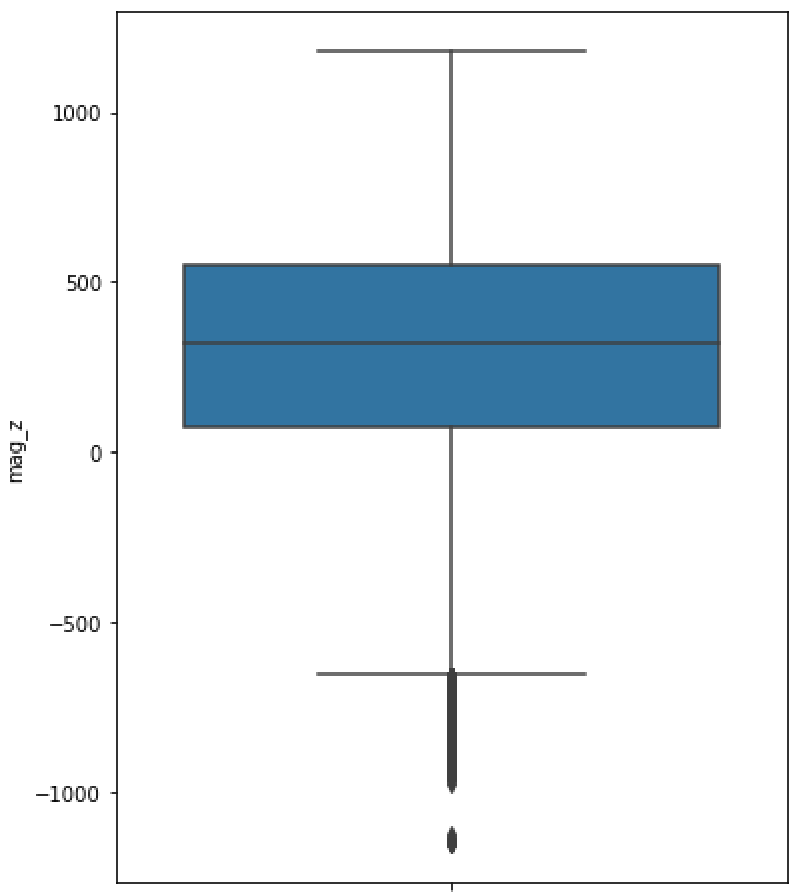

2000
1500
1000
500
0
−500
−1000
−1500
−2000
gyr_x


2000
1500
1000
500
0
−500
−1000
−1500
−2000
gyr_y


2000
1500
1000
500
0
−500
−1000
−1500
−2000
gyr_z


500
0
−500
−1000
−1500
mag_x





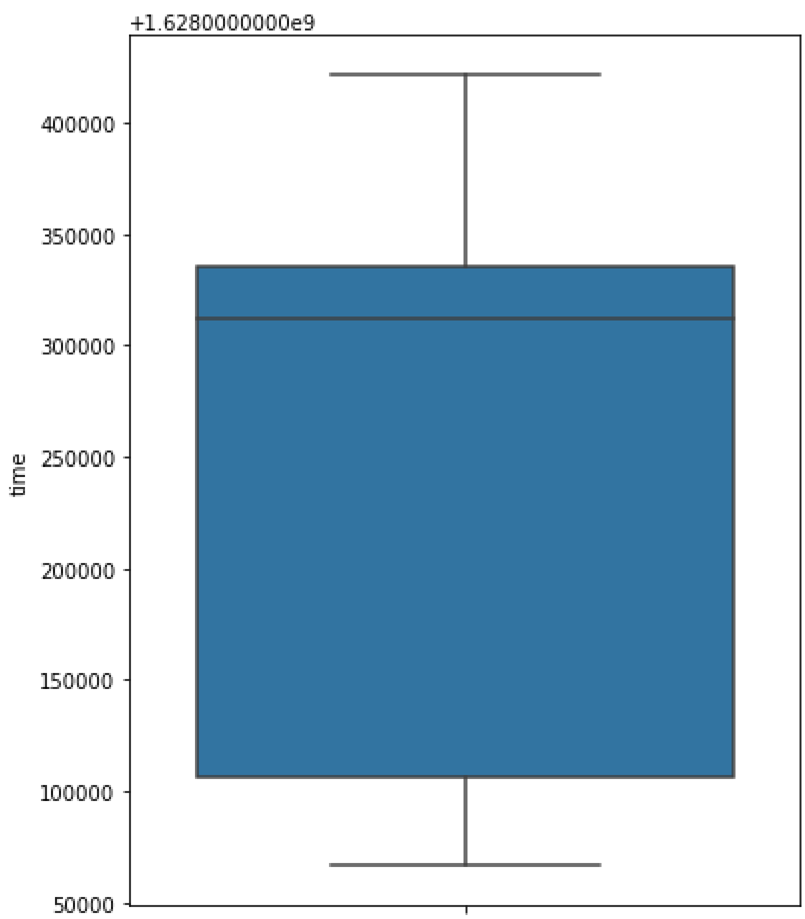
3. Feature Engineering

Replacing the outliers with meadian value

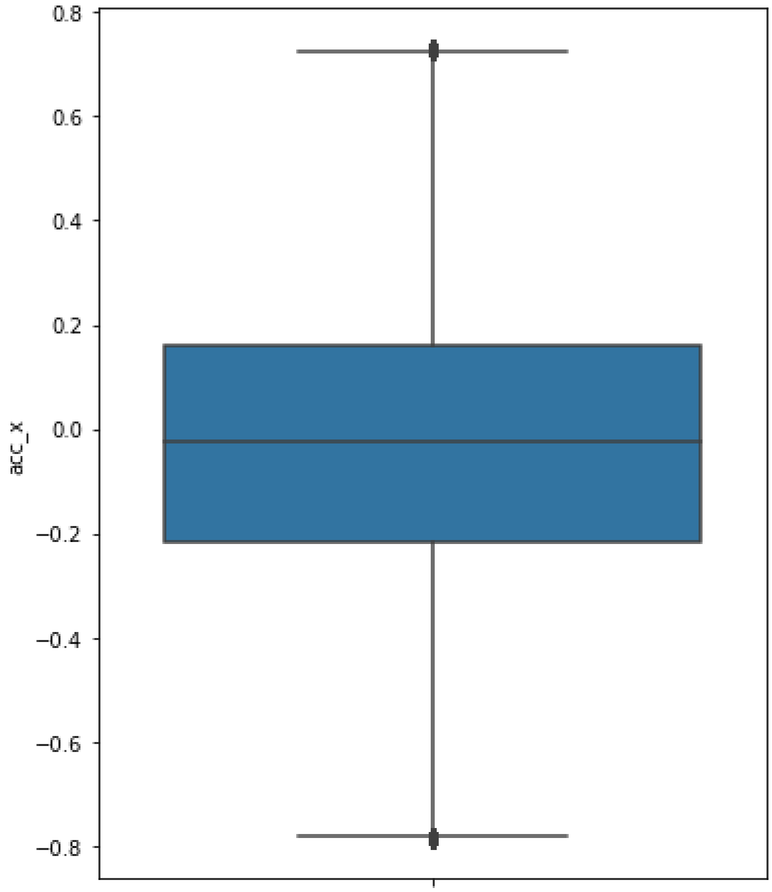
In [23]:
```
# Using a for loop inside a function to replace the outliers with median value
def replace_outlier(x, column_name):
    for i in column_name:
        print('column name : ',i)
        Q1 = np.percentile(x[i], 25)
        Q2 = np.percentile(x[i], 50)
        Q3 = np.percentile(x[i], 75)
        IQR = Q3 - Q1
        print('Q1 =',Q1,'Q2 = ',Q2,'Q3 = ',Q3)

        upper_val = Q3 + (1.5 * IQR)
        print('upper', upper_val)
        lower_val = Q1 - (1.5 * IQR)
        print('lower', lower_val)

        x.loc[x[i] > upper_val, i] = np.median(x[i])
        x.loc[x[i] < lower_val, i] = np.median(x[i])

        fig = plt.figure(figsize = (6,8))
        sns.boxplot(data = x,y = i)
        plt.xticks(rotation = 'horizontal')
        plt.show()
```

In [24]:
```
replace_outlier(x, column_name)
```

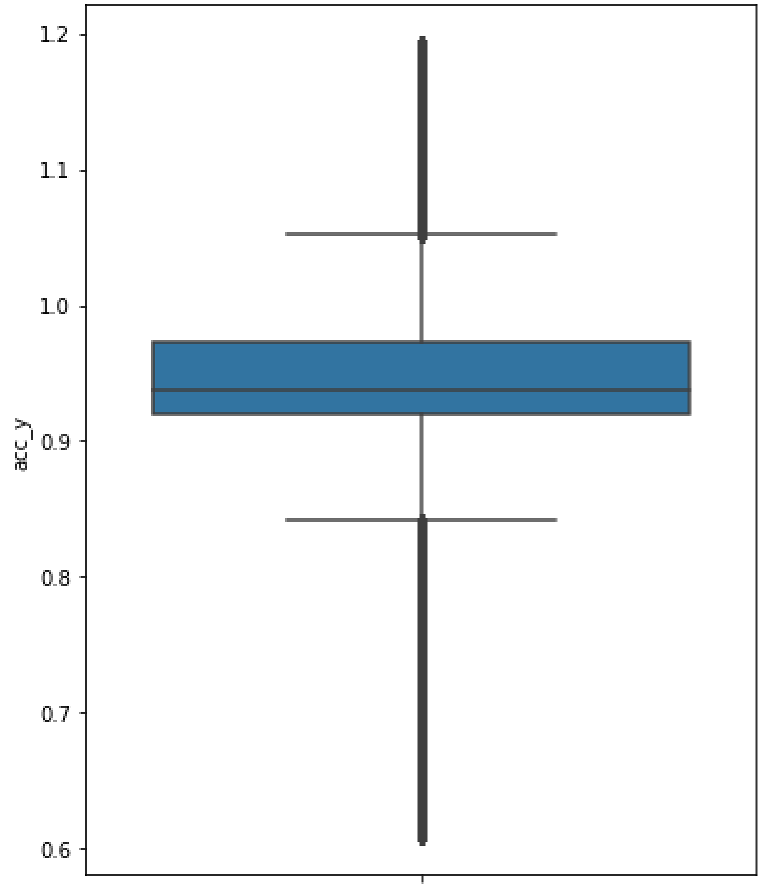
```
column name :  time
Q1 = 1628105948.0 Q2 =  1628312235.0 Q3 =  1628335043.0
upper 1628678685.5
lower 1627762305.5
```



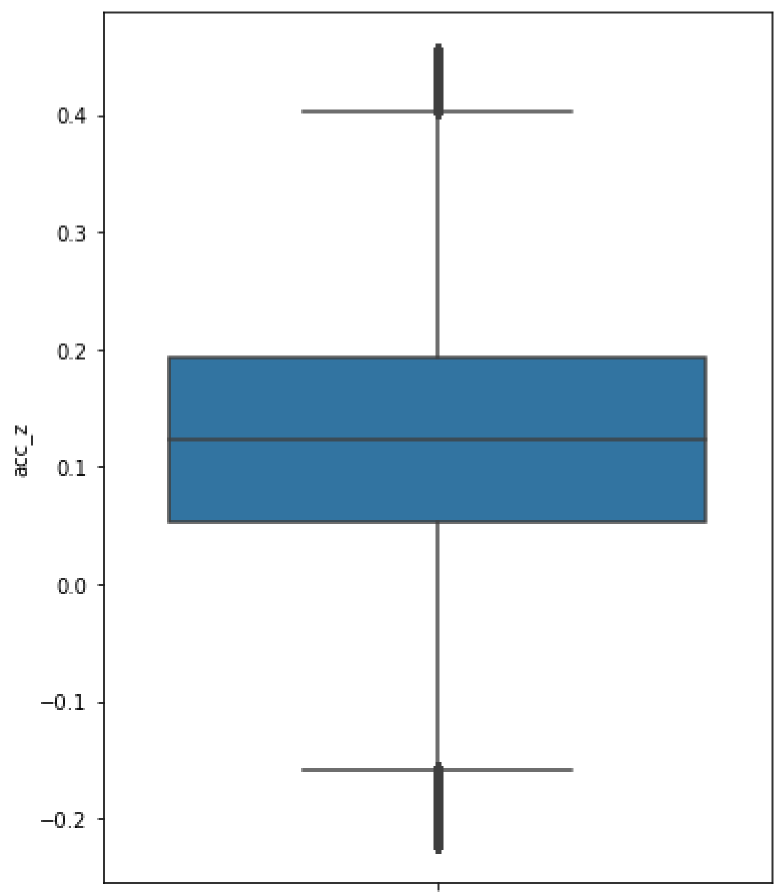
```
column name :  acc_x
Q1 = -0.2182617000000001 Q2 =  -0.0249023 Q3 =  0.1616211
upper 0.7314453000000001
lower -0.7880859000000002
```



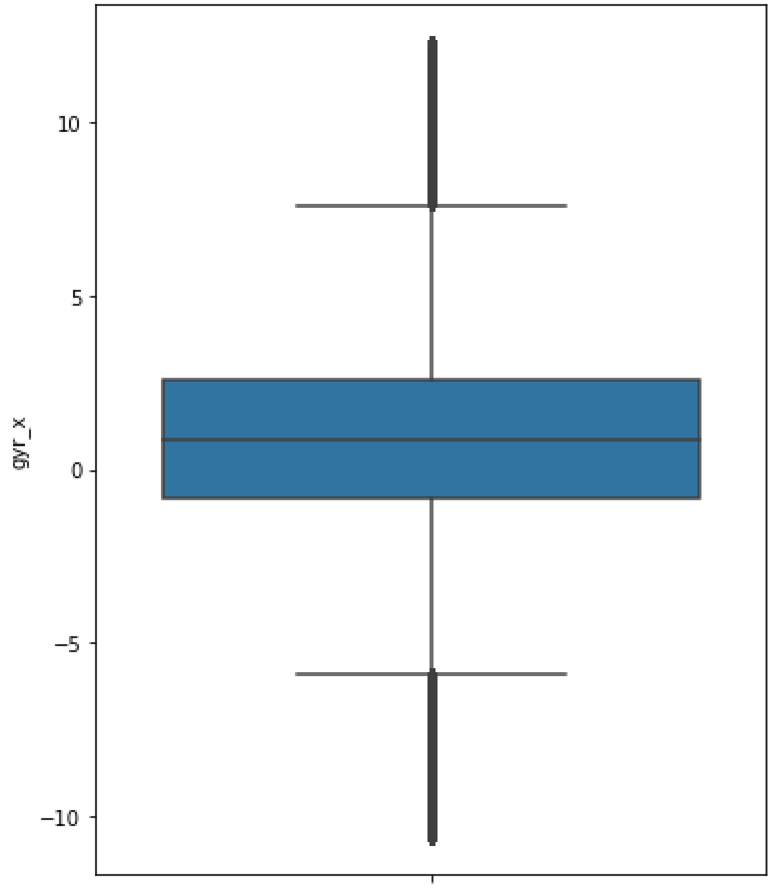

```
column name :  acc_y
Q1 = 0.828125 Q2 =  0.9370118 Q3 =  0.9736329
upper 1.1918947500000001
lower 0.60986315
```



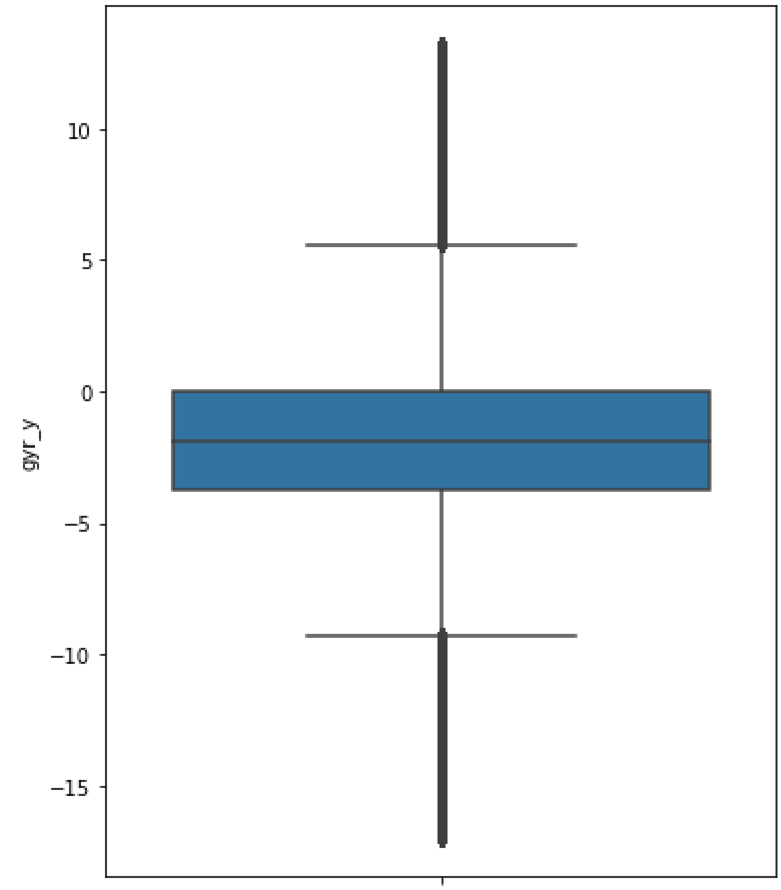

```
column name :  acc_z
Q1 = 0.0317383 Q2 =  0.1230469 Q3 =  0.2006836
upper 0.45410154999999996
lower -0.22167965
```



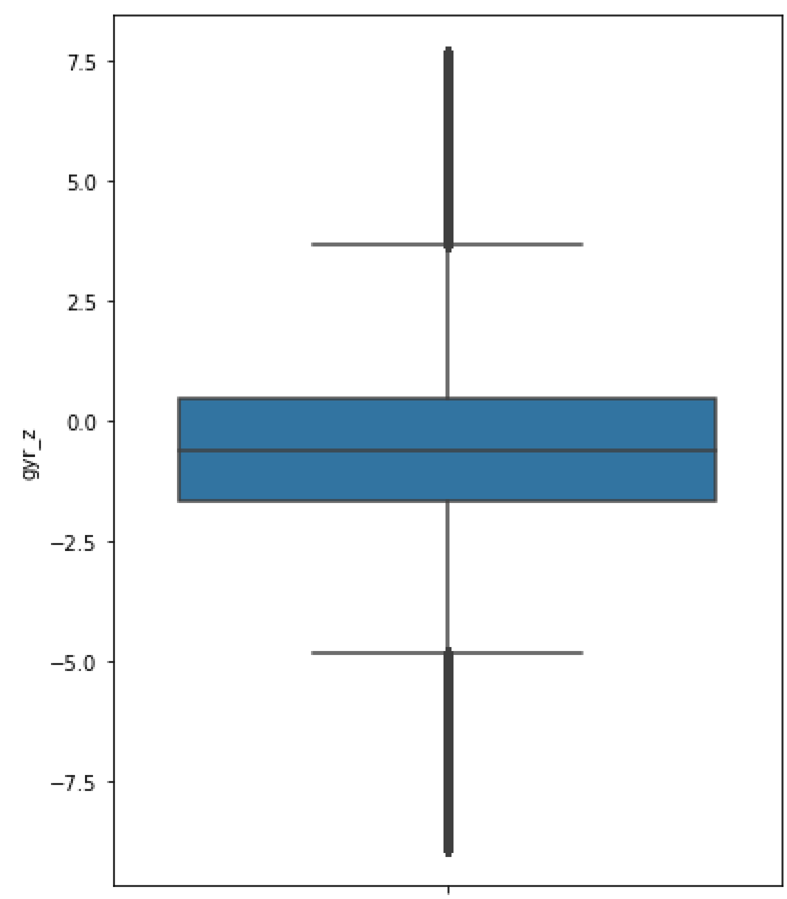
```
column name :  gyr_x
Q1 = -2.0141601 Q2 =  0.8544921999999998 Q3 =  3.7231445
upper 12.329101399999999
lower -10.620116999999999
```



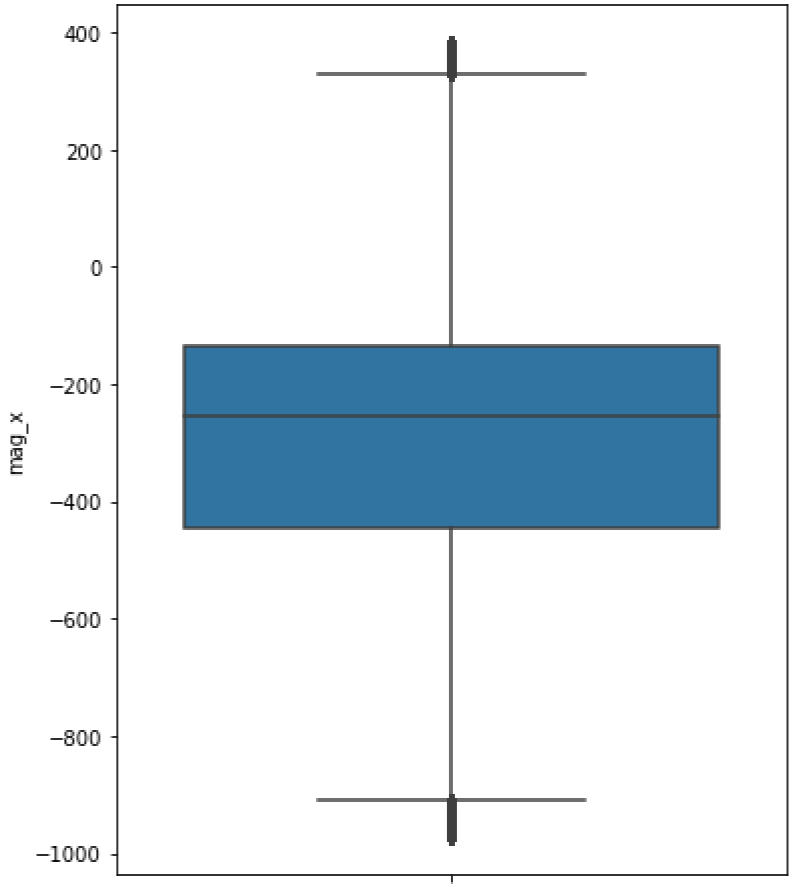

```
column name :  gyr_y
Q1 = -5.6762695 Q2 =  -1.8920898 Q3 =  1.8920898
upper 13.24462875
lower -17.02880845
```



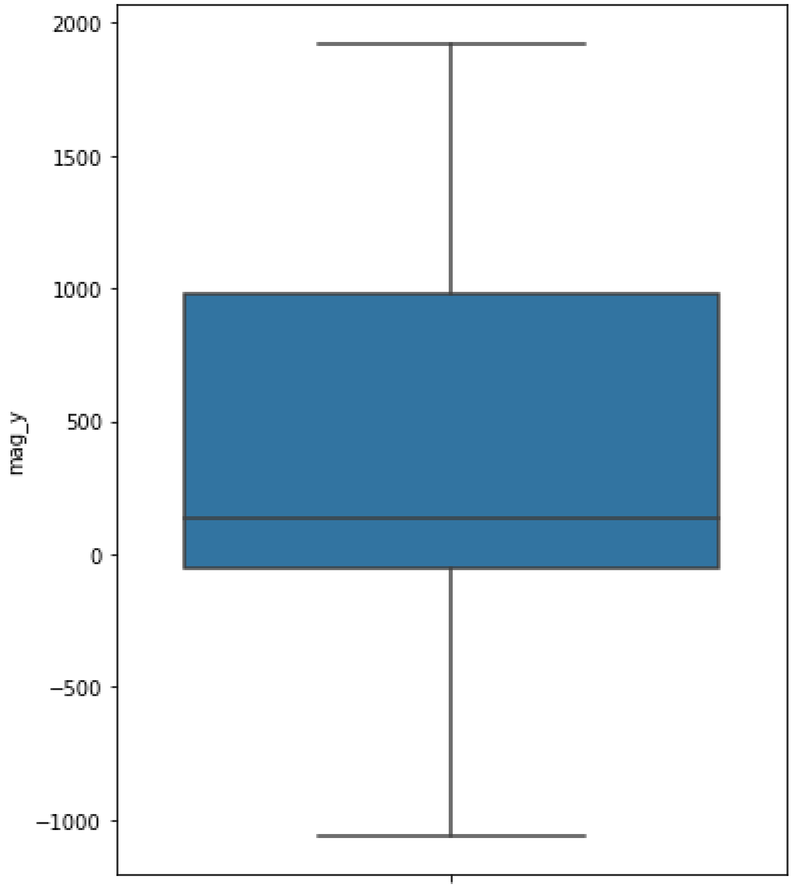

```
column name :  gyr_z
Q1 = -2.6855469 Q2 =  -0.6103516 Q3 =  1.4648437
upper 7.6904296
lower -8.911132799999999
```



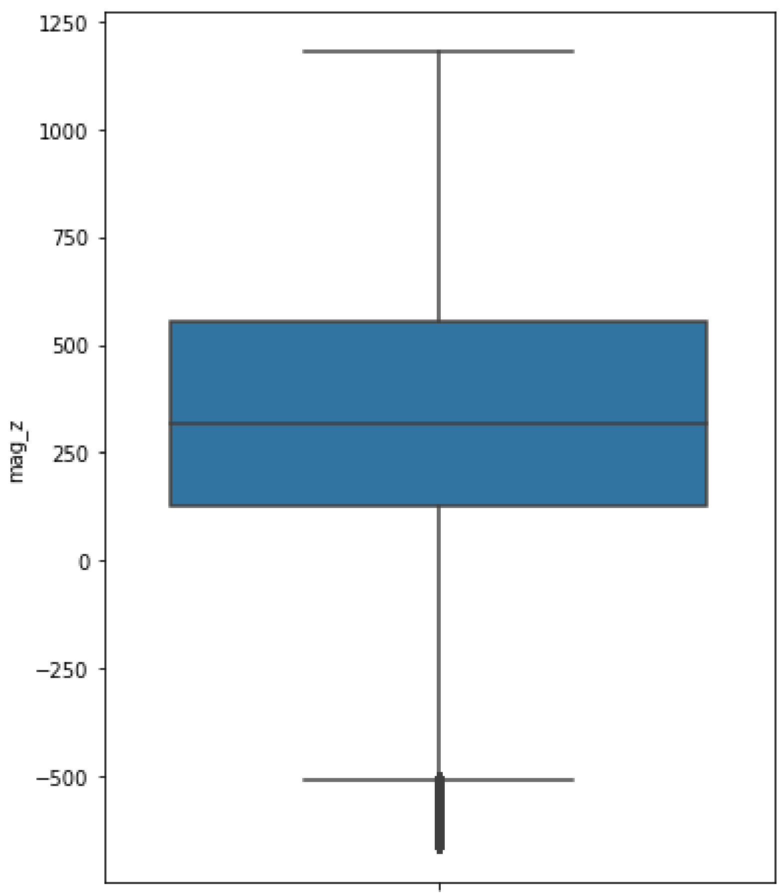
```
column name :  mag_x
Q1 = -463.5 Q2 =  -253.5 Q3 =  -126.0
upper 380.25
lower -969.75
```




```
column name :  mag_y
Q1 = -49.5 Q2 =  139.5 Q3 =  979.5
upper 2523.0
lower -1593.0
```

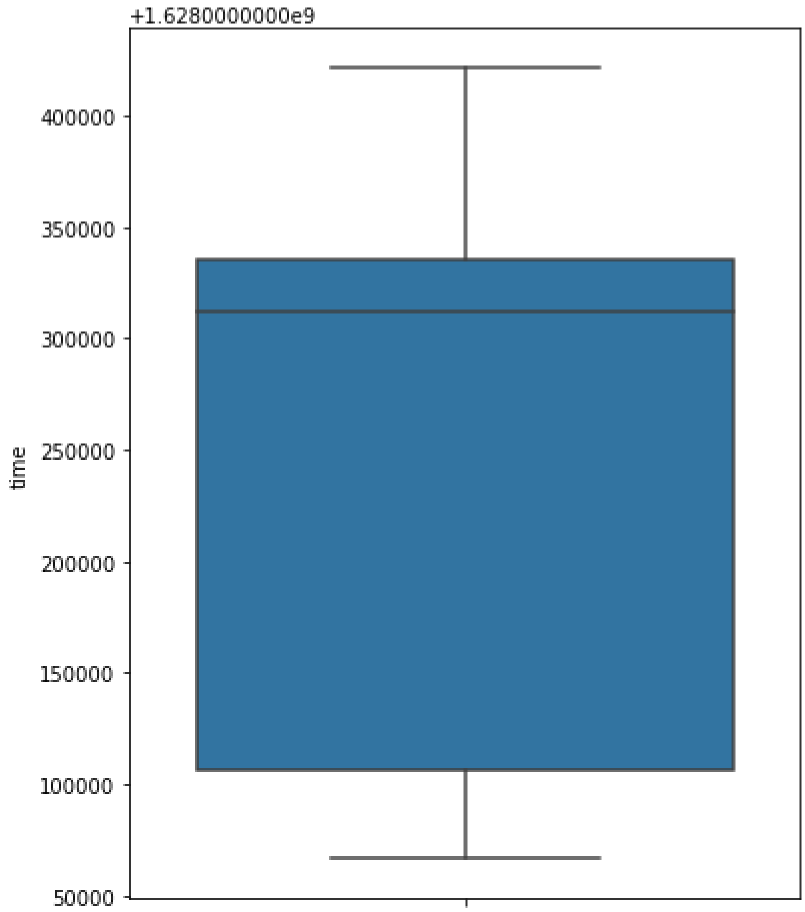


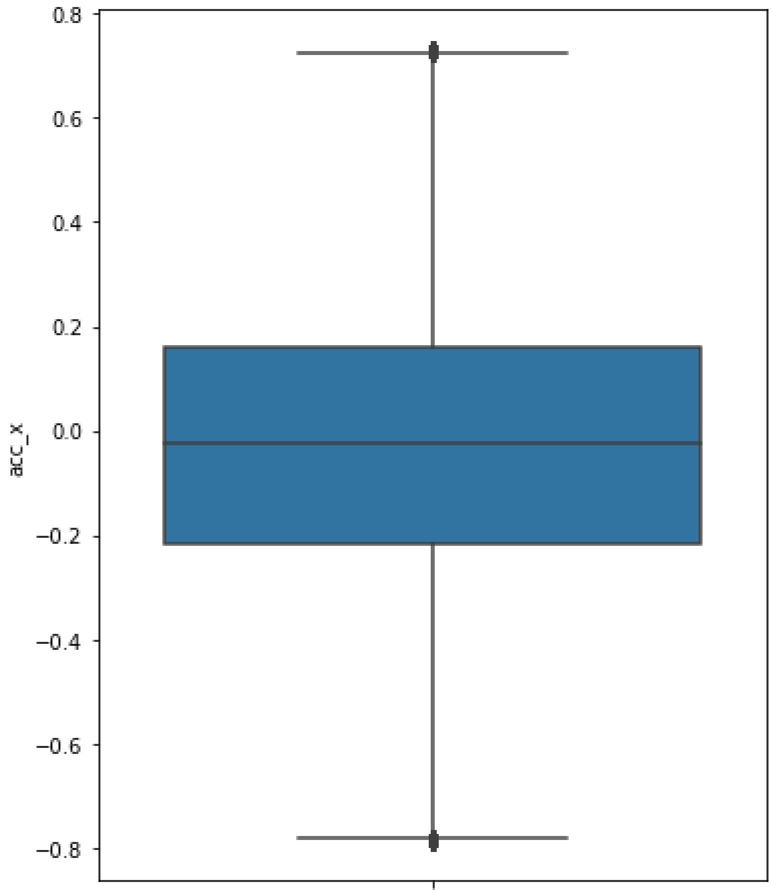

```
column name :  mag_z
Q1 = 69.0 Q2 =  316.5 Q3 =  552.0
upper 1276.5
lower -655.5
```

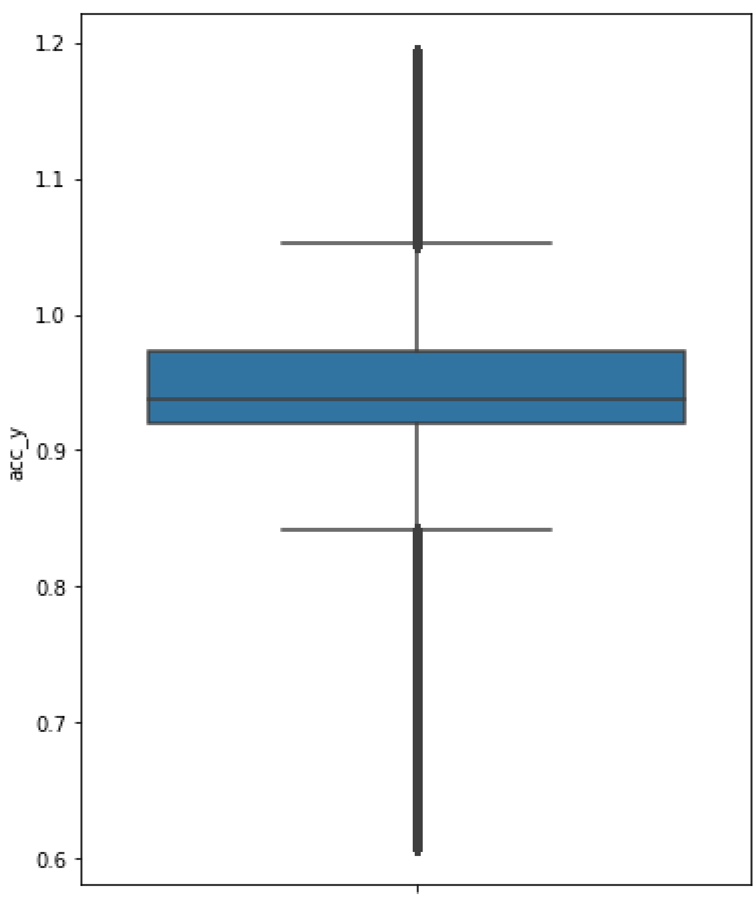


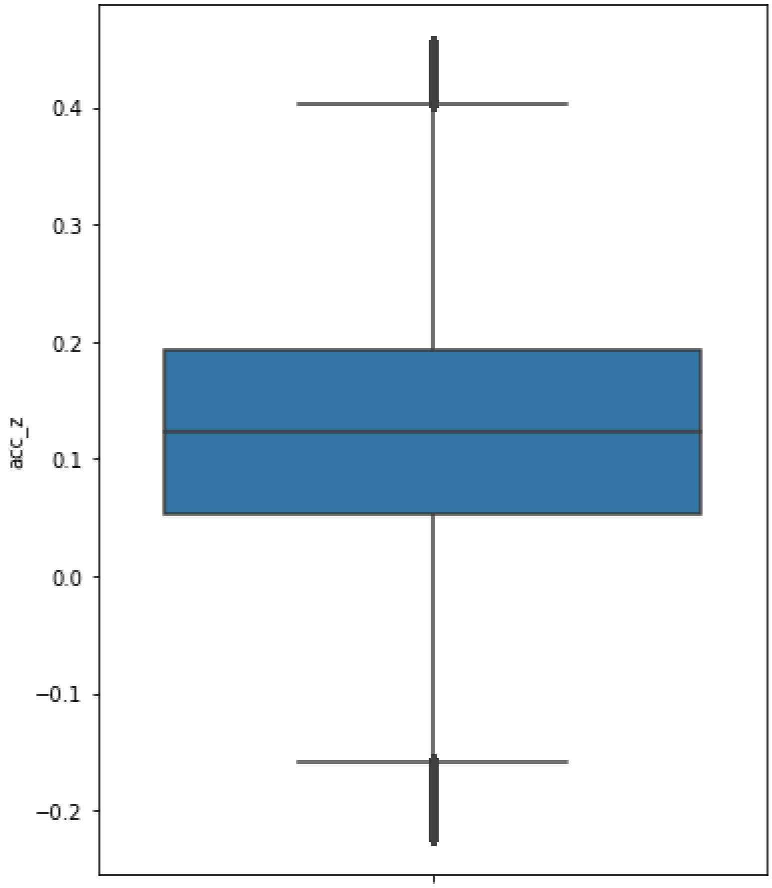
looking for outliers after imputing with median

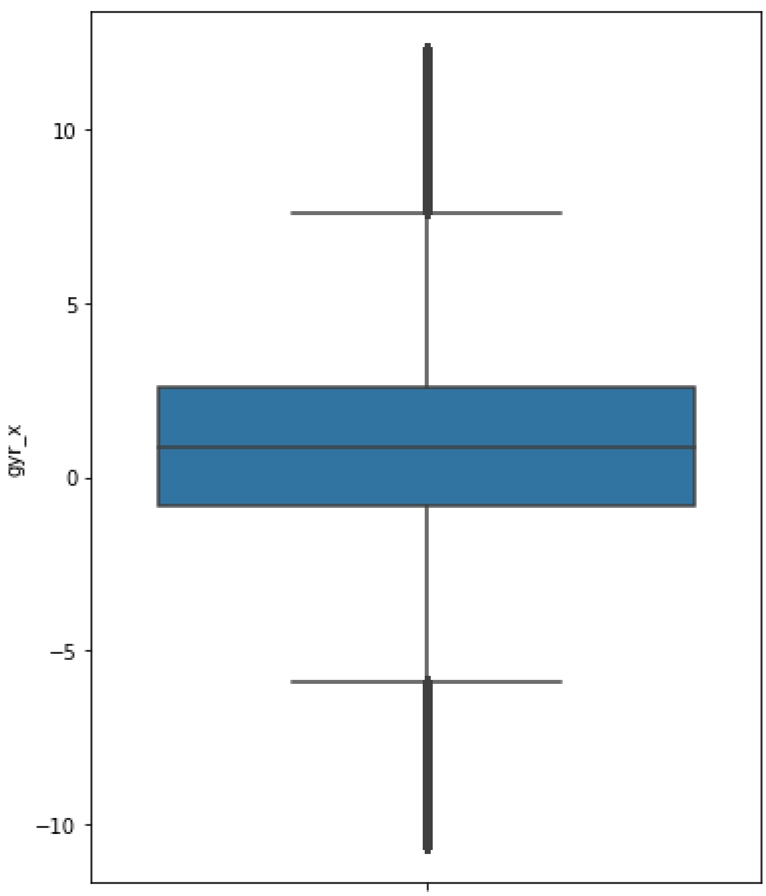
In [25]:
```
identify_outliers(x, column_name)
```

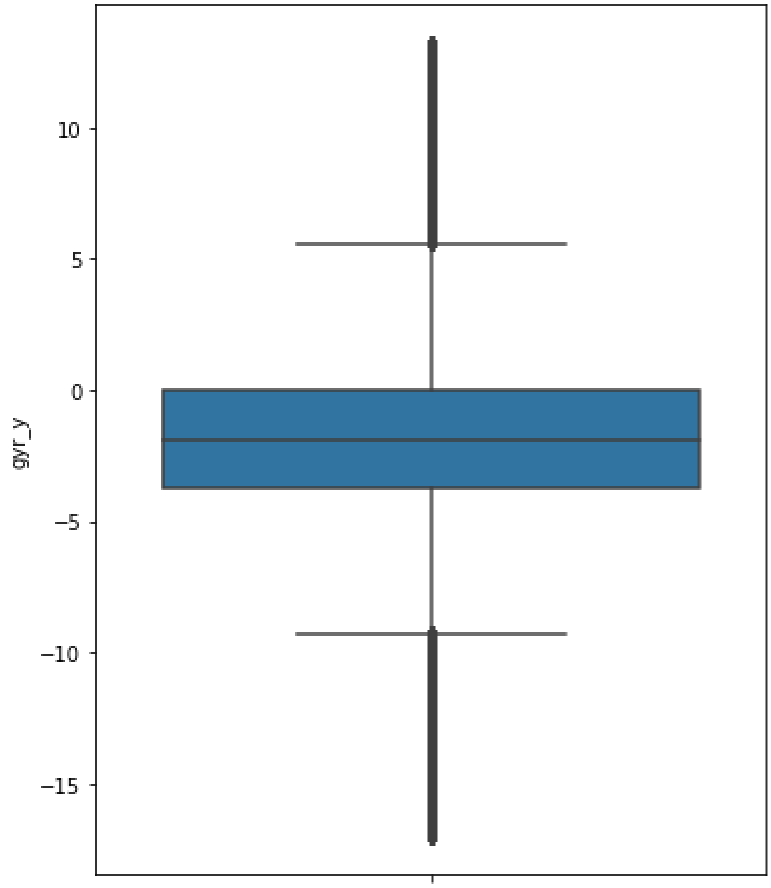


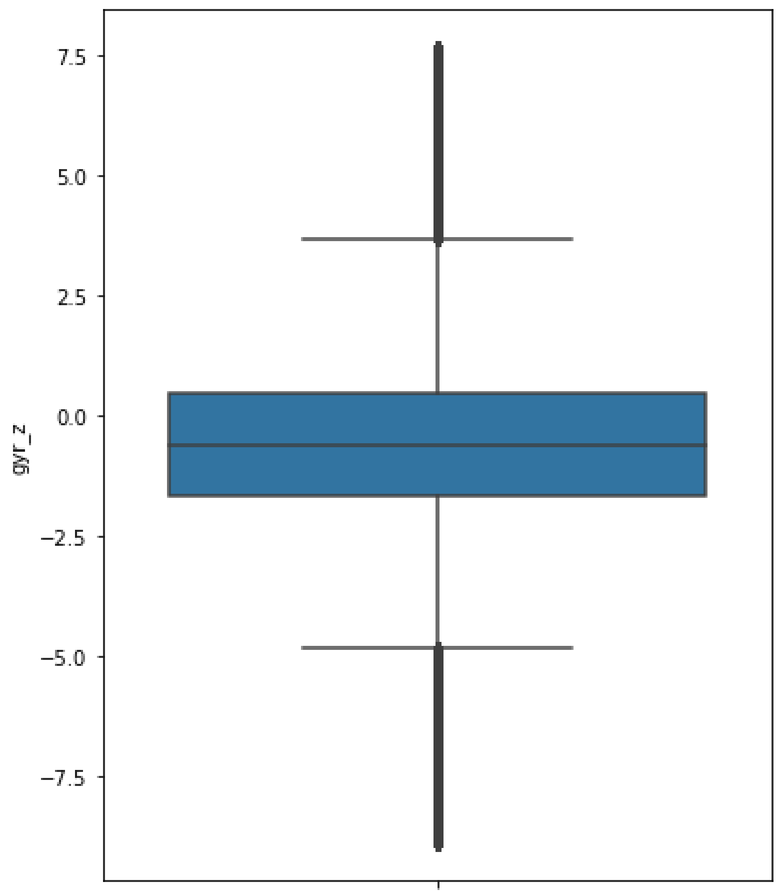

acc_x
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

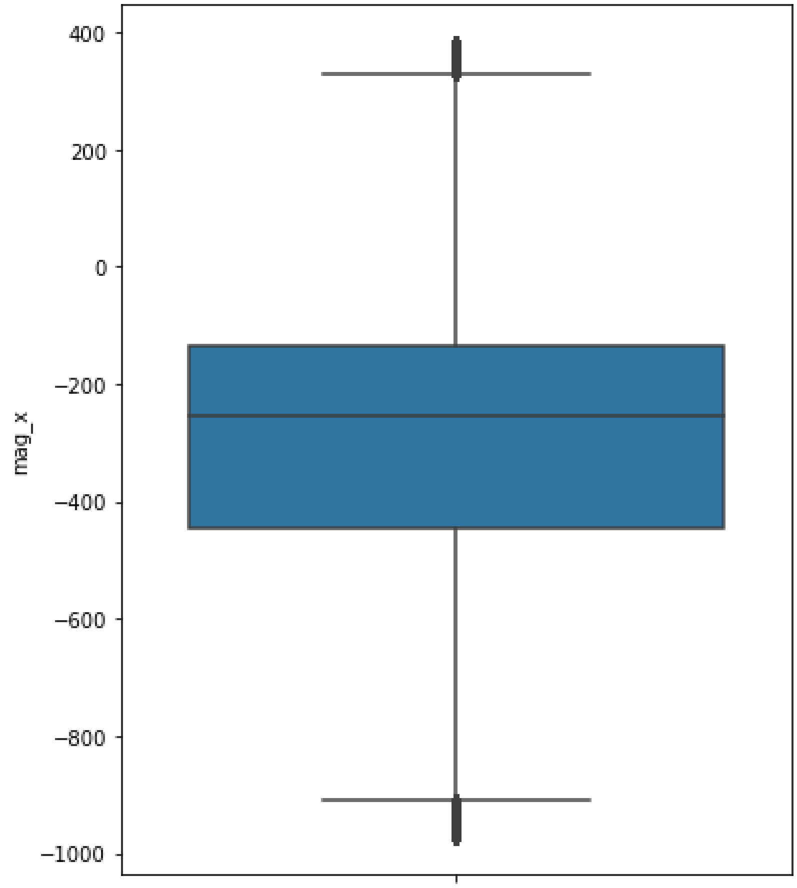

acc_y
1.2
1.1
1.0
0.9
0.8
0.7
0.6

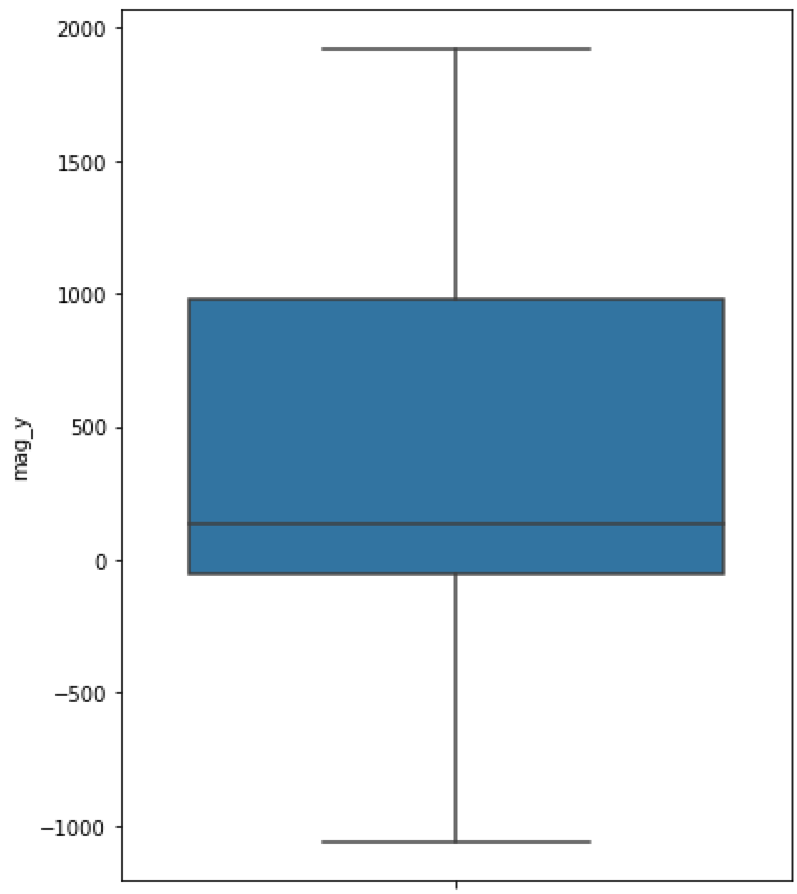

0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
acc_z

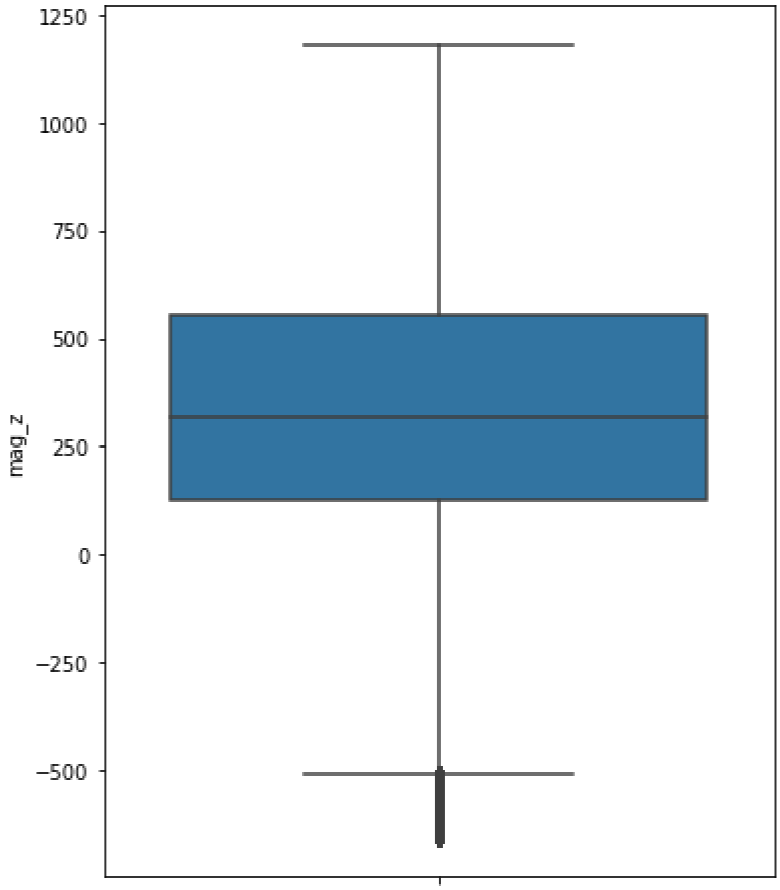

10
5
0
−5
−10
gyr_x


10
5
0
gyr_y
−5
−10
−15


7.5
5.0
2.5
0.0
gyr_z
−2.5
−5.0
−7.5


400
200
0
−200
−400
−600
−800
−1000
mag_x


2000
1500
1000
500
0
−500
−1000
mag_y



Observation

Outliers are now imputed with the median value

Normalization

All three (Acc, Gyr, Mag) gives output in different SI Units i.e The scale for all three are different, so Data must be normalized

In [26]:
```
from sklearn.preprocessing import MinMaxScaler
scaling = MinMaxScaler()
scaling.fit_transform(x)
```

Out[26]:
```
array([[0.03725611, 0.3953713 , 0.79278516, ..., 0.32814238, 0.83820565,
        0.6369583 ],
       [0.03725611, 0.42815819, 0.84060391, ..., 0.32703003, 0.83366935,
        0.62714636],
       [0.03725611, 0.53712634, 0.76258386, ..., 0.3403782 , 0.83215726,
        0.62142273],
       ...,
       [0.93311879, 0.71681137, 0.51761761, ..., 0.25695217, 0.65272177,
        0.47751431],
       [0.93311879, 0.71713276, 0.53355714, ..., 0.25806452, 0.66078629,
        0.46197874],
       [0.93311879, 0.71681137, 0.51761761, ..., 0.24026696, 0.6577621 ,
        0.4627964 ]])
```

In [27]:
```
x_normalized = pd.DataFrame(scaling.fit_transform(x))
x_normalized.head(3)
```

Out[27]:

	0	1	2	3	4	5	6	7	8	9
0	0.037256	0.395371	0.792785	0.803907	0.810160	0.5	0.5	0.328142	0.838206	0.636958
1	0.037256	0.428158	0.840604	0.764834	0.475936	0.5	0.5	0.327030	0.833669	0.627146
2	0.037256	0.537126	0.762584	0.647612	0.058824	0.5	0.5	0.340378	0.832157	0.621423

In [28]:

```
x_normalized = pd.DataFrame(scaling.fit_transform(x),columns = x.columns)
x_normalized.head(3)
```

Out[28]:

	time	acc_x	acc_y	acc_z	gyr_x	gyr_y	gyr_z	mag_x	mag_y	mag_z
0	0.037256	0.395371	0.792785	0.803907	0.810160	0.5	0.5	0.328142	0.838206	0.636958
1	0.037256	0.428158	0.840604	0.764834	0.475936	0.5	0.5	0.327030	0.833669	0.627146
2	0.037256	0.537126	0.762584	0.647612	0.058824	0.5	0.5	0.340378	0.832157	0.621423

Feature Selection

Selecting KBest Features using chi2

In [29]:

```
#select k best
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [30]:

```
#ranking the features
select_k_best_rank_features = SelectKBest(score_func = chi2, k = 5)
k_best_features = select_k_best_rank_features.fit(x_normalized,y)

df_k_scores = pd.DataFrame(k_best_features.scores_, columns = ['score'])
dfcolumns = pd.DataFrame(x_normalized.columns)

k_best_feature_rank = pd.concat([dfcolumns, df_k_scores], axis = 1)

k_best_feature_rank.columns = ('features', 'k_score')
print(k_best_feature_rank.nlargest(6, 'k_score'))
```

```
  features        k_score
0     time  551696.294963
1    acc_x  150058.553659
7    mag_x  106664.110641
8    mag_y   67694.199464
3    acc_z   43636.740650
2    acc_y   36914.462173
```

Observation:

- From chi2 we see that the top 6 features are time, acc_x, mag_x, mag_y, acc_z, acc_y

Creating a new dataframe with x_normalized and y values/

In [31]:

```
new_df = x_normalized
new_df['label'] = y.values
new_df
```

Out[31]:

	time	acc_x	acc_y	acc_z	gyr_x	gyr_y	gyr_z	mag_x	mag_y
0	0.037256	0.395371	0.792785	0.803907	0.810160	0.500000	0.500000	0.328142	0.838206
1	0.037256	0.428158	0.840604	0.764834	0.475936	0.500000	0.500000	0.327030	0.833669
2	0.037256	0.537126	0.762584	0.647612	0.058824	0.500000	0.500000	0.340378	0.832157
3	0.037256	0.618451	0.603188	0.633864	0.500000	0.500000	0.837037	0.325918	0.835181
4	0.037256	0.631308	0.469799	0.693922	0.459893	0.500000	0.374074	0.328142	0.832661
...	...	...	...	...	...	...	...	...	...
12263519	0.933119	0.713918	0.513423	0.762663	0.462567	0.497976	0.751852	0.241379	0.651210
12263520	0.933119	0.715204	0.525168	0.764110	0.489305	0.475709	0.692593	0.245829	0.658770
12263521	0.933119	0.716811	0.517618	0.765557	0.508021	0.471660	0.725926	0.256952	0.652722
12263522	0.933119	0.717133	0.533557	0.749638	0.550802	0.467611	0.759259	0.258065	0.660786
12263523	0.933119	0.716811	0.517618	0.756150	0.502674	0.485830	0.711111	0.240267	0.657762

12263524 rows × 11 columns

4. Modeling

In [49]:
```
#importing libraries
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score, recall_score
```

In [32]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train,y_test = train_test_split(x_normalized, y,
                                                   test_size = 0.3,
                                                   random_state = 42)
```

In [41]:
```
x_train.shape, y_train.shape, x_test.shape,y_test.shape
```

Out[41]:
```
((239047, 11), (239047,), (102449, 11), (102449,))
```

In [42]:
```
239047+102449
```

Out[42]:
```
341496
```

In [33]:
```
from imblearn.under_sampling import NearMiss
```

In [34]:
```
# Implementing Undersampling for Handling Imbalanced
```

```
nm = NearMiss()
x_res,y_res=nm.fit_resample(x_train,y_train)
x_res.shape, y_res.shape
```

Out[34]:
```
((341496, 11), (341496,))
```

In [37]:
```
from collections import Counter
print('Original dataset shape{}'.format(Counter(y_train)))
print('Resampled dataset shape{}'.format(Counter(y_res)))
```

```
Original dataset shapeCounter({8: 26687, 9: 26630, 6: 26612, 7: 26584, 2: 26555, 3: 2
6530, 5: 26500, 4: 26481, 1: 26468})
Resampled dataset shapeCounter({1: 37944, 2: 37944, 3: 37944, 4: 37944, 5: 37944, 6:
37944, 7: 37944, 8: 37944, 9: 37944})
```

In [46]:
```
#splitting the data
x_train, x_test, y_train, y_test = train_test_split(x_res, y_res, test_size = 0.3, rar
print(x_train.shape, y_train.shape, x_test.shape)
print('Classes and number of values in trainset before nearmiss:',Counter(y_train),'\r
print('Classes and number of values in trainset after nearmiss:',Counter(y_res),'\n')
print('Classes and number of values in trainset after nearmiss:',Counter(y_test),'\n')
```

```
(239047, 11) (239047,) (102449, 11)
Classes and number of values in trainset before nearmiss: Counter({8: 26687, 9: 2663
0, 6: 26612, 7: 26584, 2: 26555, 3: 26530, 5: 26500, 4: 26481, 1: 26468})

Classes and number of values in trainset after nearmiss: Counter({1: 37944, 2: 37944,
3: 37944, 4: 37944, 5: 37944, 6: 37944, 7: 37944, 8: 37944, 9: 37944})

Classes and number of values in trainset after nearmiss: Counter({1: 11476, 4: 11463,
5: 11444, 3: 11414, 2: 11389, 7: 11360, 6: 11332, 9: 11314, 8: 11257})
```

1. Logistic Regression

In [47]:
```
# importing Logistic Regression
from sklearn.linear_model import LogisticRegression
classifier_lr = LogisticRegression(random_state = 1000, multi_class = 'multinomial',pe

#fitting the logistic regression model to x1_train and y1_train
classifier_lr.fit(x_train, y_train)
y_pred = classifier_lr.predict(x_test)
print('model.predict :',y_pred)
print('model.score :', classifier_lr.score(x_train, y_train))

#accuracy_score
from sklearn.metrics import accuracy_score
accuracy_lr = accuracy_score(y_test, y_pred)
print('Accuracy : ',accuracy_lr)
```

```
model.predict : [6 3 6 ... 6 1 4]
model.score : 0.9909055541378892
Accuracy :  0.9911760973752795
```

In [50]:
```
#f1_score
f1_score_lr = f1_score(y_test, y_pred, average='weighted')
print('F1-score (average = weighted): {:.2f}'.format(f1_score_lr))

#precision
```

```
precision_score_lr = precision_score(y_test, y_pred, average='weighted')
print('Precision (average = weighted): {:.2f}'.format(precision_score_lr))

#recall
recall_score_lr = recall_score(y_test, y_pred, average='weighted')
print('Recall (average = weighted): {:.2f}'.format(recall_score_lr))
```

```
F1-score (average = weighted): 0.99
Precision (average = weighted): 0.99
Recall (average = weighted): 0.99
```

2. Decision Tree

In [59]:
```
#importing Decision Trees
from sklearn.tree import DecisionTreeClassifier
classifier_dtc = DecisionTreeClassifier(random_state=1000,max_depth=6,min_samples_leaf

#fitting the logistic regression model to x1_train and y1_train
classifier_dtc.fit(x_train, y_train)
y_pred = classifier_dtc.predict(x_test)
print('model.predict :',y_pred)
print('model.score :', classifier_dtc.score(x_train, y_train))

#accuracy_score
from sklearn.metrics import accuracy_score
accuracy_dtc = accuracy_score(y_test, y_pred)
print('Accuracy : ',accuracy_dtc)
```

```
model.predict : [6 3 6 ... 6 1 4]
model.score : 1.0
Accuracy :  1.0
```

In [56]:
```
#f1_score
f1_score_dtc = f1_score(y_test, y_pred, average='weighted')
print('F1-score (average = weighted): {:.2f}'.format(f1_score_dtc))

#precision
precision_score_dtc = precision_score(y_test, y_pred, average='weighted')
print('Precision (average = weighted): {:.2f}'.format(precision_score_dtc))

#recall
recall_score_dtc = recall_score(y_test, y_pred, average='weighted')
print('Recall (average = weighted): {:.2f}'.format(recall_score_dtc))
```

```
F1-score (average = weighted): 1.00
Precision (average = weighted): 1.00
Recall (average = weighted): 1.00
```

3. Random Forest

In [53]:
```
#import random forest
from sklearn.ensemble import RandomForestClassifier
classifier_rf = RandomForestClassifier(n_estimators=20, random_state=23)

#fitting the logistic regression model to x1_train and y1_train
classifier_rf.fit(x_train, y_train)
y_pred = classifier_rf.predict(x_test)
print('model.predict :',y_pred)
print('model.score :', classifier_rf.score(x_train, y_train))
```

```
#accuracy_score
from sklearn.metrics import accuracy_score
accuracy_rf = accuracy_score(y_test, y_pred)
print('Accuracy : ',accuracy_rf)
```

```
model.predict : [6 3 6 ... 6 1 4]
model.score : 1.0
Accuracy :  1.0
```

In [54]:
```
#f1_score
f1_score_rf = f1_score(y_test, y_pred, average='weighted')
print('F1-score (average = weighted): {:.2f}'.format(f1_score_rf))

#precision
precision_score_rf = precision_score(y_test, y_pred, average='weighted')
print('Precision (average = weighted): {:.2f}'.format(precision_score_rf))

#recall
recall_score_rf = recall_score(y_test, y_pred, average='weighted')
print('Recall (average = weighted): {:.2f}'.format(recall_score_rf))
```

```
F1-score (average = weighted): 1.00
Precision (average = weighted): 1.00
Recall (average = weighted): 1.00
```

4. Support Vector Machine

In [60]:
```
#importing svc
from sklearn.svm import SVC
classifier_svc=SVC(decision_function_shape='ovo')

#fitting the logistic regression model to x1_train and y1_train
classifier_svc.fit(x_train, y_train)
y_pred = classifier_svc.predict(x_test)
print('model.predict :',y_pred)
print('model.score :', classifier_svc.score(x_train, y_train))

#accuracy_score
from sklearn.metrics import accuracy_score
accuracy_svc = accuracy_score(y_test, y_pred)
print('Accuracy : ',accuracy_svc)
```

```
model.predict : [6 3 6 ... 6 1 4]
model.score : 1.0
Accuracy :  1.0
```

In [61]:
```
#f1_score
f1_score_svc = f1_score(y_test, y_pred, average='weighted')
print('F1-score (average = weighted): {:.2f}'.format(f1_score_svc))

#precision
precision_score_svc = precision_score(y_test, y_pred, average='weighted')
print('Precision (average = weighted): {:.2f}'.format(precision_score_svc))

#recall
recall_score_svc = recall_score(y_test, y_pred, average='weighted')
print('Recall (average = weighted): {:.2f}'.format(recall_score_svc))
```

```
F1-score (average = weighted): 1.00
Precision (average = weighted): 1.00
Recall (average = weighted): 1.00
```

5. KNearest Neighbors

In [62]:
```
#importing kneighbours classifier
from sklearn.neighbors import KNeighborsClassifier
classifier_knn= KNeighborsClassifier()

#fitting the logistic regression model to x1_train and y1_train
classifier_knn.fit(x_train, y_train)
y_pred = classifier_knn.predict(x_test)
print('model.predict :',y_pred)
print('model.score :', classifier_knn.score(x_train, y_train))

#accuracy_score
from sklearn.metrics import accuracy_score
accuracy_knn = accuracy_score(y_test, y_pred)
print('Accuracy : ',accuracy_knn)
```

```
model.predict : [6 3 6 ... 6 1 4]
model.score : 1.0
Accuracy :  1.0
```

In [63]:
```
#f1_score
f1_score_knn = f1_score(y_test, y_pred, average='weighted')
print('F1-score (average = weighted): {:.2f}'.format(f1_score_knn))

#precision
precision_score_knn = precision_score(y_test, y_pred, average='weighted')
print('Precision (average = weighted): {:.2f}'.format(precision_score_knn))

#recall
recall_score_knn = recall_score(y_test, y_pred, average='weighted')
print('Recall (average = weighted): {:.2f}'.format(recall_score_knn))
```

```
F1-score (average = weighted): 1.00
Precision (average = weighted): 1.00
Recall (average = weighted): 1.00
```

6. AdaBoost Classifier

In [64]:
```
#importing adaboost classifier
from sklearn.ensemble import AdaBoostClassifier
classifier_ada = AdaBoostClassifier()

#fitting the logistic regression model to x1_train and y1_train
classifier_ada.fit(x_train, y_train)
y_pred = classifier_ada.predict(x_test)
print('model.predict :',y_pred)
print('model.score :', classifier_ada.score(x_train, y_train))

#accuracy_score
from sklearn.metrics import accuracy_score
accuracy_ada = accuracy_score(y_test, y_pred)
print('Accuracy : ',accuracy_ada)
```

```
model.predict : [6 6 6 ... 6 1 6]
model.score : 0.5561751454734843
Accuracy :  0.5541098497789144
```

In [65]:
```
#f1_score
f1_score_ada = f1_score(y_test, y_pred, average='weighted')
print('F1-score (average = weighted): {:.2f}'.format(f1_score_ada))

#precision
precision_score_ada = precision_score(y_test, y_pred, average='weighted')
print('Precision (average = weighted): {:.2f}'.format(precision_score_ada))

#recall
recall_score_ada = recall_score(y_test, y_pred, average='weighted')
print('Recall (average = weighted): {:.2f}'.format(recall_score_ada))
```

```
F1-score (average = weighted): 0.48
Precision (average = weighted): 0.47
Recall (average = weighted): 0.55
```

MAKING A DATAFRAME OF ALL THE SCORES FOR EVERY MODEL BUILT

In [104…
```
#Making a dataframe of all the scores for every model
scores = [("Logistic Regression", accuracy_lr, f1_score_lr,precision_score_lr,recall_s
          ("Decision Tree",accuracy_dtc, f1_score_dtc,precision_score_dtc,recall_scor
          ("Random Forest", accuracy_rf, f1_score_rf,precision_score_rf,recall_score_
          ("KNN", accuracy_knn,f1_score_knn,precision_score_knn,recall_score_knn),
          ('Support vector machine',accuracy_svc,f1_score_svc,precision_score_svc,rec
          ('AdaBoost Classifier',accuracy_ada, f1_score_ada,precision_score_ada,recal

Scores =pd.DataFrame(data = scores, columns=['Model Name','Test Accuracy', 'F1 Score',
Scores.set_index('Model Name', inplace = True)
```

In [105…
```
Scores.style.background_gradient(cmap='YlGn')
```

Out[105]:

Model Name	Test Accuracy	F1 Score	Precision	Recall
Logistic Regression	0.965151	0.965244	0.965520	0.965151
Decision Tree	0.809880	0.749018	0.720360	0.809880
Random Forest	0.999997	0.999997	0.999997	0.999997
KNN	1.000000	1.000000	1.000000	1.000000
Support vector machine	1.000000	1.000000	1.000000	1.000000
AdaBoost Classifier	0.554110	0.480179	0.465484	0.554110

Conclusion:

We see that almost all the algorithms expect adaboost classifier gives almost 99% accuracy

In [ ]: