In [27]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [28]:

```
x_train = pd.read_csv('X_train.csv')
y_train = pd.read_csv('y_train.csv')
x_test = pd.read_csv('X_test.csv')
```

In [29]:

```
print("Shape of dataset")
print("X Train: {}\ny Train: {}\nX Test: {}".format(x_train.shape, y_train.shape, x_test.sh
```

```
Shape of dataset
X Train: (487680, 13)
y Train: (3810, 3)
X Test: (488448, 13)
```

In [66]:

```
print(y_train.sample(5))
```

```
      series_id  group_id   surface
3602       3602        43      wood
484         484         1     tiled
1834       1834        60    carpet
3533       3533        46      wood
2540       2540        39  concrete
```

In [31]:

```
x_train.head()
```

Out[31]:

	row_id	series_id	measurement_number	orientation_X	orientation_Y	orientation_Z	orientati
0	0_0	0	0	-0.75853	-0.63435	-0.10488	-0.1
1	0_1	0	1	-0.75853	-0.63434	-0.10490	-0.1
2	0_2	0	2	-0.75853	-0.63435	-0.10492	-0.1
3	0_3	0	3	-0.75852	-0.63436	-0.10495	-0.1
4	0_4	0	4	-0.75852	-0.63435	-0.10495	-0.1

In [32]:

```
x_test.head()
```

Out[32]:

	row_id	series_id	measurement_number	orientation_X	orientation_Y	orientation_Z	orientati
0	0_0	0	0	0.91208	-0.38193	-0.050618	0.·
1	0_1	0	1	0.91220	-0.38165	-0.050573	0.·
2	0_2	0	2	0.91228	-0.38143	-0.050586	0.·
3	0_3	0	3	0.91237	-0.38121	-0.050588	0.·
4	0_4	0	4	0.91247	-0.38096	-0.050546	0.·

In [33]:

```
x_train.dtypes
```

Out[33]:

```
row_id                    object
series_id                  int64
measurement_number         int64
orientation_X            float64
orientation_Y            float64
orientation_Z            float64
orientation_W            float64
angular_velocity_X       float64
angular_velocity_Y       float64
angular_velocity_Z       float64
linear_acceleration_X    float64
linear_acceleration_Y    float64
linear_acceleration_Z    float64
dtype: object
```

In [67]:

```
y_train.dtypes
```

Out[67]:

```
series_id     int64
group_id      int64
surface      object
dtype: object
```

OBSERVATIONS:

X_train and X_test datasets have the following entries:

- **series and measurements identifiers** : row_id, series_id, measurement_number: these identify uniquely a series and measurement; there are 3809 series, each with max 127 measurements
- **measurement orientations** : orientation_X, orientation_Y, orientation_Z, orientation_W
- **angular velocities** : angular_velocity_X, angular_velocity_Y, angular_velocity_Z
- **linear accelerations** : linear_acceleration_X, linear_acceleration_Y, linear_acceleration_Z

y_train has the following columns:

- **series_id -** this corresponds to the series in train data
- **group_id**
- **surface -** this is the surface type that need to be predicted

Checking for missing values :

In [34]:

```
print(x_train.isnull().sum())
```

```
row_id                   0
series_id                0
measurement_number       0
orientation_X            0
orientation_Y            0
orientation_Z            0
orientation_W            0
angular_velocity_X       0
angular_velocity_Y       0
angular_velocity_Z       0
linear_acceleration_X    0
linear_acceleration_Y    0
linear_acceleration_Z    0
dtype: int64
```

In [35]:

```
print(x_test.isnull().sum())
```

```
row_id                  0
series_id               0
measurement_number      0
orientation_X           0
orientation_Y           0
orientation_Z           0
orientation_W           0
angular_velocity_X      0
angular_velocity_Y      0
angular_velocity_Z      0
linear_acceleration_X   0
linear_acceleration_Y   0
linear_acceleration_Z   0
dtype: int64
```

In [36]:

```
print(y_train.isnull().sum())
```

```
series_id    0
group_id     0
surface      0
dtype: int64
```

observation: no missing values in dataset

In [37]:

```
x_train.describe()
```

Out[37]:

	series_id	measurement_number	orientation_X	orientation_Y	orientation_Z	orie
count	487680.000000	487680.000000	487680.000000	487680.000000	487680.000000	487(
mean	1904.500000	63.500000	-0.018050	0.075062	0.012458	
std	1099.853353	36.949327	0.685696	0.708226	0.105972	
min	0.000000	0.000000	-0.989100	-0.989650	-0.162830	
25%	952.000000	31.750000	-0.705120	-0.688980	-0.089466	
50%	1904.500000	63.500000	-0.105960	0.237855	0.031949	
75%	2857.000000	95.250000	0.651803	0.809550	0.122870	
max	3809.000000	127.000000	0.989100	0.988980	0.155710	

In [38]:

```
x_test.describe()
```

Out[38]:

	series_id	measurement_number	orientation_X	orientation_Y	orientation_Z	orie
count	488448.000000	488448.000000	488448.000000	488448.000000	488448.000000	488
mean	1907.500000	63.500000	0.031996	0.120651	0.018735	
std	1101.585403	36.949327	0.671977	0.714522	0.108481	
min	0.000000	0.000000	-0.989720	-0.989810	-0.154680	
25%	953.750000	31.750000	-0.648130	-0.744503	-0.112660	
50%	1907.500000	63.500000	0.132910	0.397860	0.057271	
75%	2861.250000	95.250000	0.575270	0.803600	0.124770	
max	3815.000000	127.000000	0.989320	0.988940	0.154250	

In [68]:

```
y_train.describe(include='object')
```

Out[68]:

	surface
count	3810
unique	9
top	concrete
freq	779

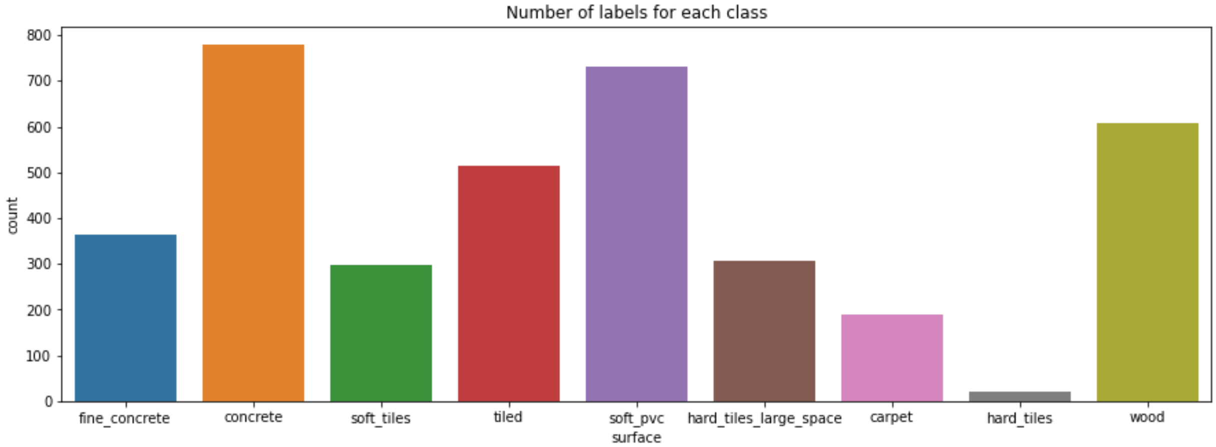
Observation:

There is the same number of series in X_train and y_train, numbered from 0 to 3809 (total 3810). Each series have 128 measurements. Each series in train dataset is part of a group (numbered from 0 to 72, 72 being the half of 128). The number of rows in X_train and X_test differs with 6 x 128, 128 being the number of measurements for each group.

Distribution of target variable:

In [40]:

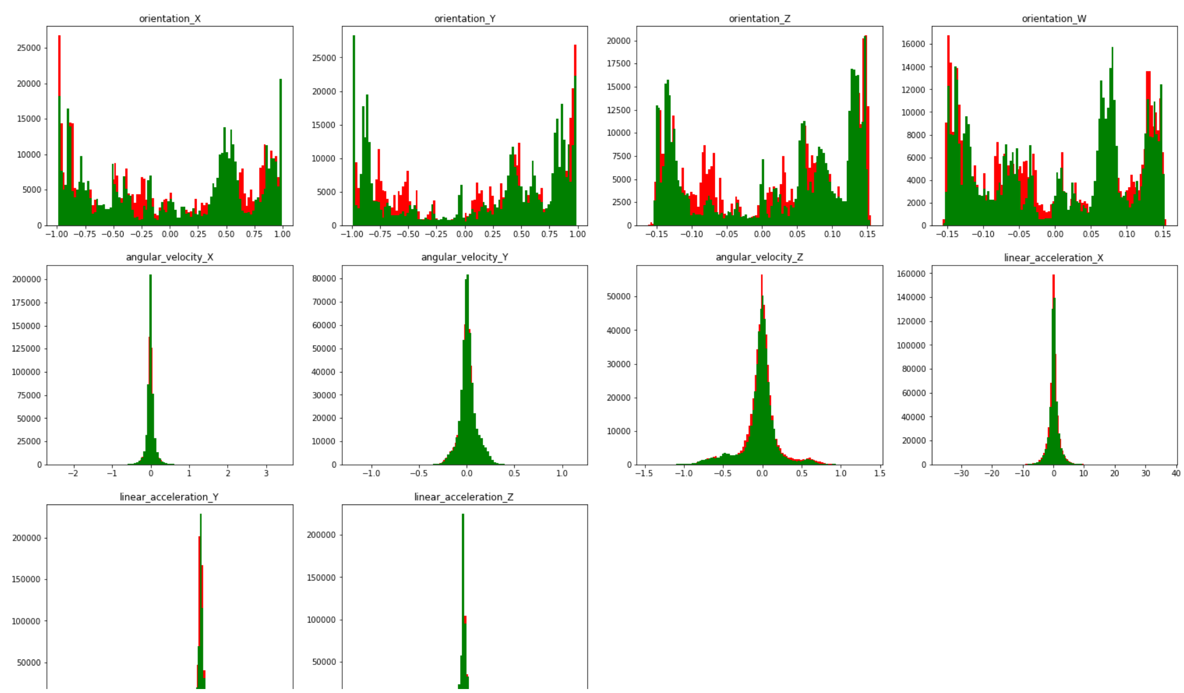
```
import warnings
warnings.filterwarnings('ignore')
f, ax = plt.subplots(1,1, figsize=(15,5))
graph = sns.countplot(y_train['surface'])
graph.set_title("Number of labels for each class")
plt.show()
```



Distribution of Train(red) and Test(green) features:

In [41]:

```
plt.figure(figsize=(26, 16))
for i, col in enumerate(x_train.columns[3:]):
    plt.subplot(3, 4, i+1)
    plt.hist(x_train[col], color='red', bins=100)
    plt.hist(x_test[col], color='green', bins=100)
    plt.title(col)
```

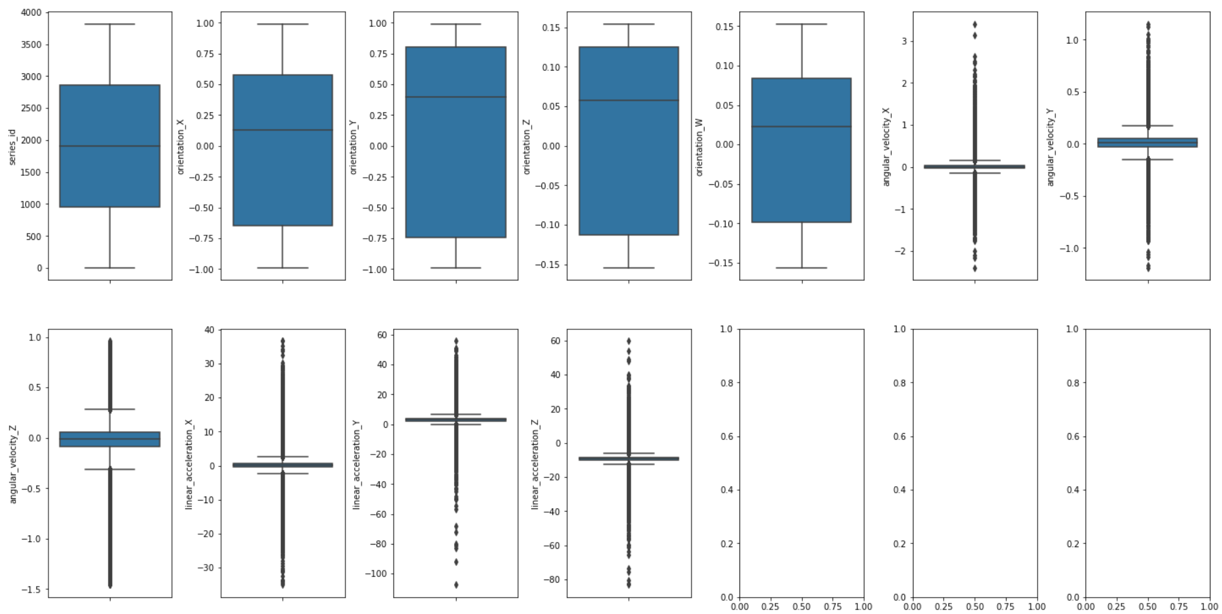


Observations:

- Velocity and acceleration have normal distribution
- Feature distributions in train and test are quite similar.

In [71]:

```
fig, axs = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in x_test.items():
 sns.boxplot(y=k, data=x_test, ax=axs[index])
 index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



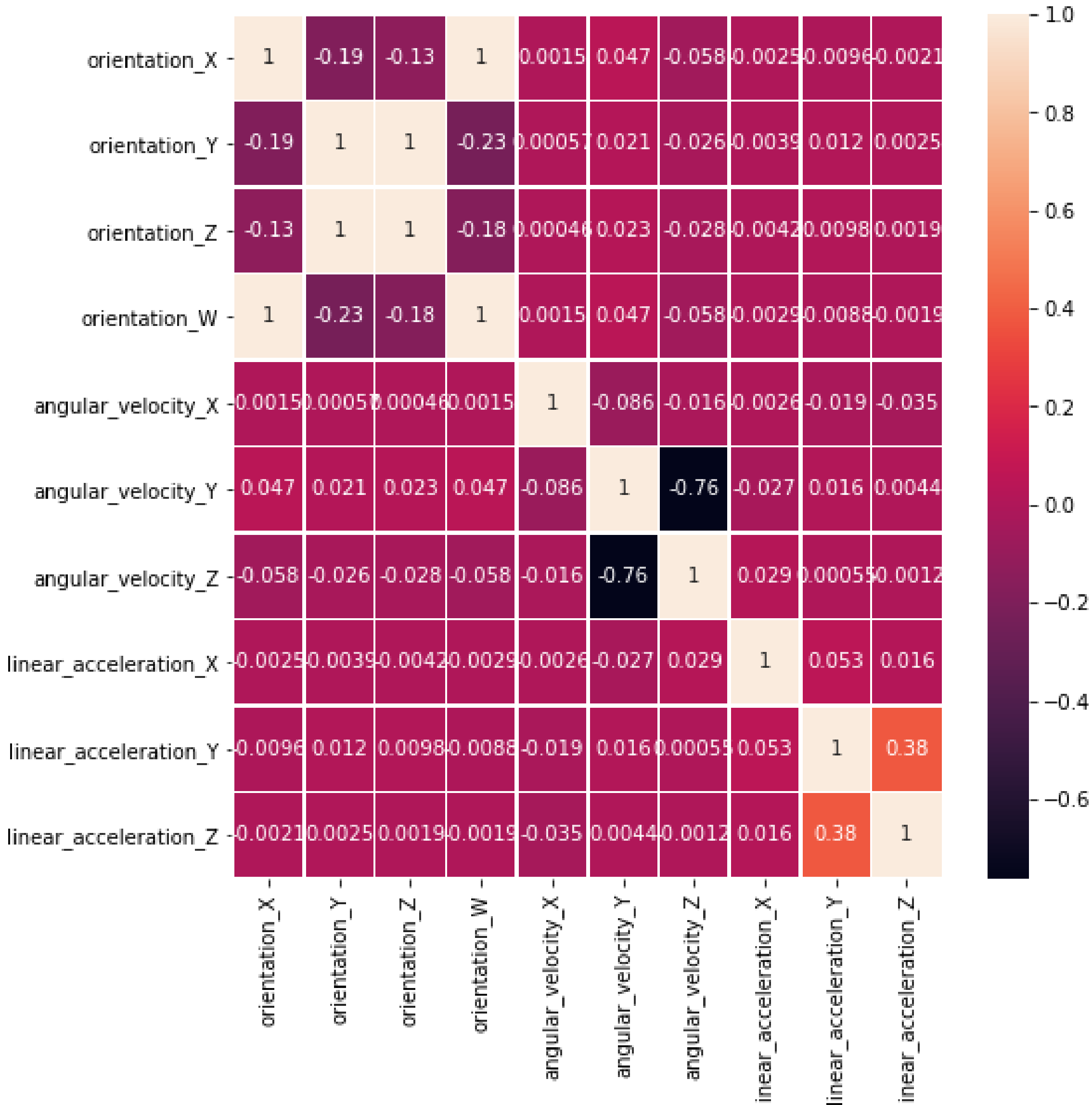
Correlation heatmaps:

In [42]:

```
f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(x_train.iloc[:,3:].corr(), annot=True, linewidths=.5)
```

Out[42]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a28f171b48>
```

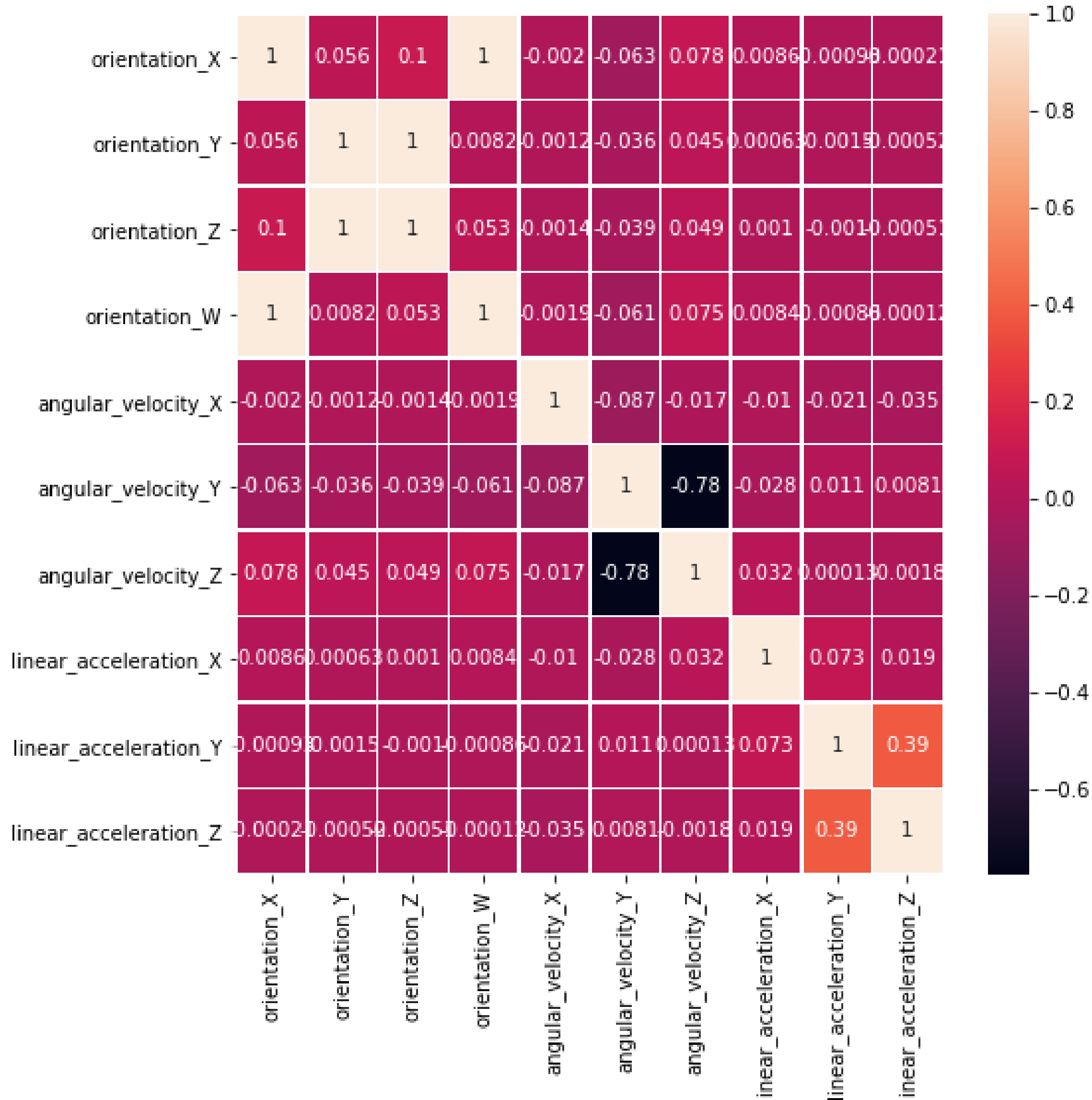


In [43]:

```
f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(x_test.iloc[:,3:].corr(), annot=True, linewidths=.5)
```

Out[43]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a2939f3608>
```



Observation:

There is a strong correlation between:

- angular_velocity_Z and angular_velocity_Y
- orientation_X and orientation_Y
- orientation_Y and orientation_Z

Feature Engineering:

Merging x_train and y_train into a single data-frame

In [44]:

```
df = pd.merge(x_train,y_train,how='left',on='series_id')
df.sample()
```

Out[44]:

	row_id	series_id	measurement_number	orientation_X	orientation_Y	orientation_Z	o
427725	3341_77	3341	77	-0.9848	0.082082	0.007407	

In [45]:

```
df.drop(columns=["row_id","measurement_number","group_id"], inplace=True)
df.sample(2)
```

Out[45]:

	series_id	orientation_X	orientation_Y	orientation_Z	orientation_W	angular_velocity_X
337671	2638	-0.95176	0.26741	0.034794	-0.146390	-0.028428
238244	1861	0.11640	0.98232	0.146260	0.009979	-0.022359

In [46]:

```
just_check =  df.groupby("series_id").mean().reset_index()
df = pd.merge(just_check,y_train,how='left',on='series_id')
df.sample(3)
```

Out[46]:

	series_id	orientation_X	orientation_Y	orientation_Z	orientation_W	angular_velocity_X	an
331	331	-0.388866	-0.908631	-0.144438	-0.048040	0.009423	
69	69	0.800486	-0.580899	-0.082066	0.122626	0.004783	
732	732	0.226724	0.961707	0.151068	0.029913	0.000041	

In [47]:

```
df.drop(columns=["series_id","group_id"],inplace=True)
```

Model Building:

Splitting merged data frame into train and test splits:

In [48]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
train = scaler.fit_transform(df[df.columns[:-1]])
train_x, test_x, train_y, test_y = train_test_split(train,df[df.columns[-1]],test_size = 0.
```

In [49]:

```
train_x.shape
```

Out[49]:

```
(3429, 10)
```

Random Forest base model

In [50]:

```
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(train_x, train_y)
target = model.predict(test_x)
mat = confusion_matrix(test_y, target)
print("******confusion******\n",mat)
```

```
******confusion******
 [[11  3  0  0  0  0  1  3  3]
 [ 0 68  3  0  0  7  0  1  4]
 [ 0  2 24  0  0  0  0  0  2]
 [ 0  0  0  2  1  0  0  0  0]
 [ 1  2  1  0 23  1  1  1  2]
 [ 0  1  2  0  1 71  0  1  2]
 [ 0  6  1  0  1  4 21  1  0]
 [ 1  3  0  0  0  4  0 43  3]
 [ 0  5  3  0  1  2  1  2 34]]
```

In [51]:

```
x_test.drop(columns=["row_id","measurement_number"],inplace=True)
testing = x_test.groupby("series_id").mean().reset_index()
testing.sample(3)
testing.shape
```

Out[51]:

```
(3816, 11)
```

In [52]:

```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
fin_train = scale.fit_transform(df[df.columns[:-1]])
test = scale.transform(testing[testing.columns[1:]])
```

Final model: XG boosted Classifier

In [53]:

```
from xgboost import XGBClassifier

model = XGBClassifier()
model.fit(fin_train,df[df.columns[-1]])
target = model.predict(test)
```

In [54]:

```
a = testing.series_id.tolist()
b = target
submission = pd.DataFrame({"series_id":a,"surface":b})
```

In [55]:

```
submission.surface.value_counts()
```

Out[55]:

```
concrete                 1086
wood                      702
soft_pvc                  569
fine_concrete             378
tiled                     334
hard_tiles_large_space    312
soft_tiles                268
carpet                    163
hard_tiles                  4
Name: surface, dtype: int64
```

In [56]:

```
submission.to_csv("submission_final.csv",index=False)
```

In [57]:

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import confusion_matrix,accuracy_score
from xgboost import XGBClassifier
for i in range(4,10):
    model = XGBClassifier(model_depth = i)
    model.fit(train_x, train_y)
    target = model.predict(test_x)
    print("accuracy : ", accuracy_score(target, test_y))
mat = confusion_matrix(test_y, target)
print("******confusion matrix******\n",mat)
```

```
[07:15:29] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release
_1.2.0\src\learner.cc:516:
Parameters: { model_depth } might not be used.

  This may not be accurate due to some parameters are only used in languag
e bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip t
hrough this
  verification. Please open an issue if you find above cases.


accuracy :  0.7926509186351706
[07:15:31] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release
_1.2.0\src\learner.cc:516:
Parameters: { model_depth } might not be used.

  This may not be accurate due to some parameters are only used in languag
e bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip t
hrough this
  verification. Please open an issue if you find above cases.


accuracy :  0.7926509186351706
[07:15:33] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release
_1.2.0\src\learner.cc:516:
Parameters: { model_depth } might not be used.

  This may not be accurate due to some parameters are only used in languag
e bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip t
hrough this
  verification. Please open an issue if you find above cases.


accuracy :  0.7926509186351706
[07:15:35] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release
_1.2.0\src\learner.cc:516:
Parameters: { model_depth } might not be used.

  This may not be accurate due to some parameters are only used in languag
```

```
e bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip t
hrough this
  verification. Please open an issue if you find above cases.


accuracy :  0.7926509186351706
[07:15:37] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release
_1.2.0\src\learner.cc:516:
Parameters: { model_depth } might not be used.

  This may not be accurate due to some parameters are only used in languag
e bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip t
hrough this
  verification. Please open an issue if you find above cases.


accuracy :  0.7926509186351706
[07:15:39] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release
_1.2.0\src\learner.cc:516:
Parameters: { model_depth } might not be used.

  This may not be accurate due to some parameters are only used in languag
e bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip t
hrough this
  verification. Please open an issue if you find above cases.


accuracy :  0.7926509186351706
******confusion matrix******
 [[14  3  0  0  0  0  0  2  2]
 [ 0 74  2  0  0  4  0  0  3]
 [ 0  1 25  0  0  0  1  0  1]
 [ 0  0  0  1  1  1  0  0  0]
 [ 1  1  1  0 24  2  1  1  1]
 [ 0  1  2  0  1 69  0  1  4]
 [ 0  7  1  0  1  3 21  1  0]
 [ 2  4  0  0  0  3  2 39  4]
 [ 1  4  3  0  0  3  0  2 35]]
```

Final accuracy of model : 0.79

This was my first cut solution to the problem . A more detailed solution will be updated here in the future.