Richter's Predictor: Modeling Earthquake Damage

Problem statement:

Following the 7.8 Mw Gorkha Earthquake on April 25, 2015, Nepal carried out a massive household survey using mobile technology to assess building damage in the earthquake-affected districts. Although the primary goal of this survey was to identify beneficiaries eligible for government assistance for housing reconstruction, it also collected other useful socio-economic information. In addition to housing reconstruction, this data serves a wide range of uses and users e.g. researchers, newly formed local governments, and citizens at large. Based on aspects of building location and construction, the goal is to predict the level of damage to buildings caused by the 2015 Gorkha earthquake in Nepal.

Data Description:

The data was collected through surveys by the Central Bureau of Statistics that work under the National Planning Commission Secretariat of Nepal. This survey is one of the largest post-disaster datasets ever collected, containing valuable information on earthquake impacts, household conditions, and socio-economic-demographic statistics.

The raw dataset contains 762106 datapoints and consists of 3 files:
1.Structure Data: Contains structural information of the properties
2.Damage Data: Contains Damage assesment data of the prroperties.
3.Ownership Data: Conntains geographical,legal data of the properties.

The cleaned training Dataset contains 609674 datapoints.
The cleaned test Dataset contains 152419 datapoints.
300k datapoints has been used for this case study.

Data Source:

https://eq2015.npc.gov.np/#/

Features

The dataset mainly consists of information on the buildings' structure and their legal ownership. Each row in the dataset represents a specific building in the region that was hit by Gorkha earthquake.

There are 42 columns in this dataset, where the `building_id` column is a unique and random identifier. The remaining 41 features are described in the section below. Categorical variables have been obfuscated random lowercase ascii characters. The appearance of the same character in distinct columns does **not** imply the same original value.

Description

- `geo1`, `geo2`, `geo3` (type: int): geographic region in which building exists, from largest (level 1) to most specific sub-region (level 3). Possible values: level 1: 0-30, level 2: 0-1427, level 3: 0-12567.
- `count_floors_pre_eq` (type: int): number of floors in the building before the earthquake.
- `age` (type: int): age of the building in years.
- `plinth_area_sq_ft` (type: int):area of the building footprint.
- `land_surface_condition` (type: categorical): surface condition of the land where the building was built.
- `foundation_type` (type: categorical): type of foundation used while building.
- `roof_type` (type: categorical): type of roof used while building.
- `ground_floor_type` (type: categorical): type of the ground floor.
- `other_floor_type` (type: categorical): type of constructions used in higher than the ground floors (except of roof).
- `position` (type: categorical): position of the building.
- `plan_configuration` (type: categorical): building plan configuration.
- `has_superstructure_adobe_mud` (type: binary): flag variable that indicates if the superstructure was made of Adobe/Mud.
- `has_superstructure_mud_mortar_stone` (type: binary): flag variable that indicates if the superstructure was made of Mud Mortar - Stone.
- `has_superstructure_stone_flag` (type: binary): flag variable that indicates if the superstructure was made of Stone.
- `has_superstructure_cement_mortar_stone` (type: binary): flag variable that indicates if the superstructure was made of Cement Mortar - Stone.
- `has_superstructure_mud_mortar_brick` (type: binary): flag variable that indicates if the superstructure was made of Mud Mortar - Brick.
- `has_superstructure_cement_mortar_brick` (type: binary): flag variable that indicates if the superstructure was made of Cement Mortar - Brick.
- `has_superstructure_timber` (type: binary): flag variable that indicates if the superstructure was made of Timber.
- `has_superstructure_bamboo` (type: binary): flag variable that indicates if the superstructure was made of Bamboo.
- `has_superstructure_rc_non_engineered` (type: binary): flag variable that indicates if the superstructure was made of non-engineered reinforced concrete.
- `has_superstructure_rc_engineered` (type: binary): flag variable that indicates if the superstructure was made of engineered reinforced concrete.
- `has_superstructure_other` (type: binary): flag variable that indicates if the superstructure was made of any other material.
- `legal_ownership_status` (type: categorical): legal ownership status of the land where building was built. Possible values: a, r, v, w.
- `count_families` (type: int): number of families that live in the building.
- `has_secondary_use` (type: binary): flag variable that indicates if the building was used for any secondary purpose.
- `has_secondary_use_agriculture` (type: binary): flag variable that indicates if the building was used for agricultural purposes.
- `has_secondary_use_hotel` (type: binary): flag variable that indicates if the building was used as a hotel.
- `has_secondary_use_rental` (type: binary): flag variable that indicates if the building was used for rental purposes.

- `has_secondary_use_institution` (type: binary): flag variable that indicates if the building was used as a location of any institution.
- `has_secondary_use_school` (type: binary): flag variable that indicates if the building was used as a school.
- `has_secondary_use_industry` (type: binary): flag variable that indicates if the building was used for industrial purposes.
- `has_secondary_use_health_post` (type: binary): flag variable that indicates if the building was used as a health post.
- `has_secondary_use_gov_office` (type: binary): flag variable that indicates if the building was used fas a government office.
- `has_secondary_use_use_police` (type: binary): flag variable that indicates if the building was used as a police station.
- `has_secondary_use_other` (type: binary): flag variable that indicates if the building was secondarily used for other purposes.

Feature data example

field	value
geo1	8
geo2	396
geo3	1108
count_floors_pre_eq	2
age	15
area_percentage	4
height_percentage	7
land_surface_condition	Flat
foundation_type	Mud_stone_OR_brick
roof_type	wood_heavy
ground_floor_type	Brick_or_Stone
other_floor_type	wood_mud
position	Attached_1_side
plan_configuration	Rectangular
has_superstructure_adobe_mud	1
has_superstructure_mud_mortar_stone	1
has_superstructure_stone_flag	0
has_superstructure_cement_mortar_stone	0
has_superstructure_mud_mortar_brick	0
has_superstructure_cement_mortar_brick	1
has_superstructure_timber	0
has_superstructure_bamboo	0
has_superstructure_rc_non_engineered	0
has_superstructure_rc_engineered	0
has_superstructure_other	1
legal_ownership_status	Institutional
count_families	1
has_secondary_use	0
has_secondary_use_agriculture	0
has_secondary_use_hotel	0
has_secondary_use_rental	0
has_secondary_use_institution	0
has_secondary_use_school	0
has_secondary_use_industry	0
has_secondary_use_health_post	0
has_secondary_use_gov_office	0
has_secondary_use_use_police	0
has_secondary_use_other	0

Machine Learning Problem

We have to predict the ordinal variable `damage_grade`, which represents a level of damage to the building that was hit by the earthquake. There are 3 grades of the damage:

- `1` represents low damage
- `2` represents a medium amount of damage
- `3` represents almost complete destruction/High Damage

We do not have any low latency requirment.

Performance metric

We are predicting the level of damage from 1 to 3. The level of damage is an ordinal variable meaning that ordering is important. This can be viewed as a *classification* or an *ordinal regression* problem. (Ordinal regression is sometimes described as an problem somewhere in between classification and regression.)

To measure the performance of our algorithms, we'll use the **F1 score** which balances the precision and recall of a classifier. Traditionally, the F1 score is used to evaluate performance on a binary classifier, but since we have three possible labels we will use a variant called the **micro averaged F1 score**.

$$F_{micro} = \frac{2 \cdot P_{micro} \cdot R_{micro}}{P_{micro} + R_{micro}}$$

where

$$P_{micro} = \frac{\sum_{k=1}^{3} TP_k}{\sum_{k=1}^{3} (TP_k + FP_k)}, \quad R_{micro} = \frac{\sum_{k=1}^{3} TP_k}{\sum_{k=1}^{3} (TP_k + FN_k)}$$

and $|TP|$ is True Positive, $|FP|$ is False Positive, $|FN|$ is False Negative, and $|k|$ represents each class in $|1, 2, 3|$.

Project Structure

This case study is divided in to following files/folders:

1. Raw Data: Contains raw datasets.
2. Clean Data: Contains clean train and test datasets.
3. EQ modelling EDA.ipynb : Ipython notebook used for exploratory data analysis and introduction.
4. EQ Data Cleaning : Ipython notebook used for cleaning raw data and preparing train,test dataset.
5. EQ prediction models.ipynb : Ipython notebook used for model preparation,training,testing.
6. Models : final models in pickle format .

Exploratory data analysis

In [2]:
```
#Importing necessary Libraries
%matplotlib inline
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc,accuracy_score,f1_score,precision_score,recall_score
from sklearn.linear_model import ElasticNet, Lasso,  BayesianRidge, LassoLarsIC,Ridge
from sklearn.ensemble import RandomForestRegressor,  GradientBoostingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.svm import LinearSVC
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import precision_score,recall_score,auc
import xgboost as xgb
import lightgbm as lgb
from mlxtend.regressor import StackingRegressor,StackingCVRegressor
```

In [3]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [4]:
```
train_data=pd.read_csv('train_values.csv')
test_data=pd.read_csv('test_values.csv')
train_y=pd.read_csv('train_labels.csv')
test_y=pd.read_csv('test_labels.csv')
```

In [5]:
```
print('Training data shape:',train_data.shape)
print('Test data shape:',test_data.shape)
```

```
Training data shape: (609674, 42)
Test data shape: (152419, 42)
```

Data Preparation:

In [6]:
```
# overview of dataset
train_data.head(5)
```

Out[6]:

	building_id	geo1	geo2	geo3	count_floors_pre_eq	count_floors_post_eq	age_building	plinth_area_sq_ft	height_ft_pre_eq	height_ft_post_eq	...	has_
0	201801000561	20	2001	200101	2	2	15	303	18	18	...	
1	360505001221	36	3604	360406	2	2	30	735	12	12	...	
2	245606000381	24	2404	240403	2	2	9	391	14	14	...	
3	234201000741	23	2309	230902	2	2	5	560	14	14	...	
4	223707000531	22	2205	220503	3	0	16	310	18	0	...	

5 rows × 42 columns

In [6]:
```
# features
train_data.columns
```

Out[6]:
```
Index(['building_id', 'geo1', 'geo2', 'geo3', 'count_floors_pre_eq',
       'count_floors_post_eq', 'age_building', 'plinth_area_sq_ft',
       'height_ft_pre_eq', 'height_ft_post_eq', 'land_surface_condition',
       'foundation_type', 'roof_type', 'ground_floor_type', 'other_floor_type',
       'position', 'plan_configuration', 'has_superstructure_adobe_mud',
       'has_superstructure_mud_mortar_stone', 'has_superstructure_stone_flag',
       'has_superstructure_cement_mortar_stone',
       'has_superstructure_mud_mortar_brick',
       'has_superstructure_cement_mortar_brick', 'has_superstructure_timber',
       'has_superstructure_bamboo', 'has_superstructure_rc_non_engineered',
       'has_superstructure_rc_engineered', 'has_superstructure_other',
       'damage_grade', 'legal_ownership_status', 'count_families',
       'has_secondary_use', 'has_secondary_use_agriculture',
       'has_secondary_use_hotel', 'has_secondary_use_rental',
       'has_secondary_use_institution', 'has_secondary_use_school',
       'has_secondary_use_industry', 'has_secondary_use_health_post',
       'has_secondary_use_gov_office', 'has_secondary_use_use_police',
```

```
 'has_secondary_use_other'],
dtype='object')
```

Checking for Duplicates:

In [7]:
```
print('No of duplicates in train: {}'.format(sum(train_data.duplicated())))
print('No of duplicates in test : {}'.format(sum(test_data.duplicated())))
```

```
No of duplicates in train: 0
No of duplicates in test : 0
```

Checking for Missing values:

In [8]:
```
print('We have {} NaN/Null values in train'.format(train_data.isnull().values.sum()))
print('We have {} NaN/Null values in test'.format(test_data.isnull().values.sum()))
```

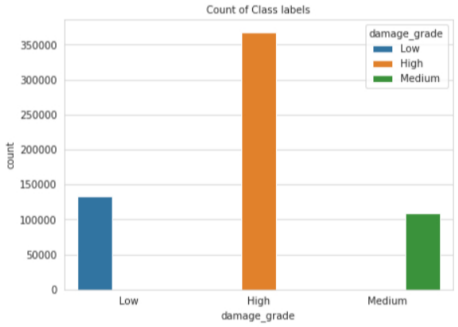
```
We have 0 NaN/Null values in train
We have 0 NaN/Null values in test
```

In [9]:
```
train_data.damage_grade=train_data.damage_grade.replace({1:'Low',2:'Medium',3:'High'})
test_data.damage_grade=test_data.damage_grade.replace({1:'Low',2:'Medium',3:'High'})
```

In [10]:
```
train=train_data
```

Checking for Imbalance:

In [11]:
```
sns.set_style('whitegrid')
plt.rcParams['font.family'] = 'Dejavu Sans'
plt.figure(figsize=(7,5))
plt.title('Count of Class labels', fontsize=10)
sns.countplot(x='damage_grade',data=train,hue='damage_grade')
plt.show()
```



The dataset is imbalanced.Class 2 is dominant over other classes.

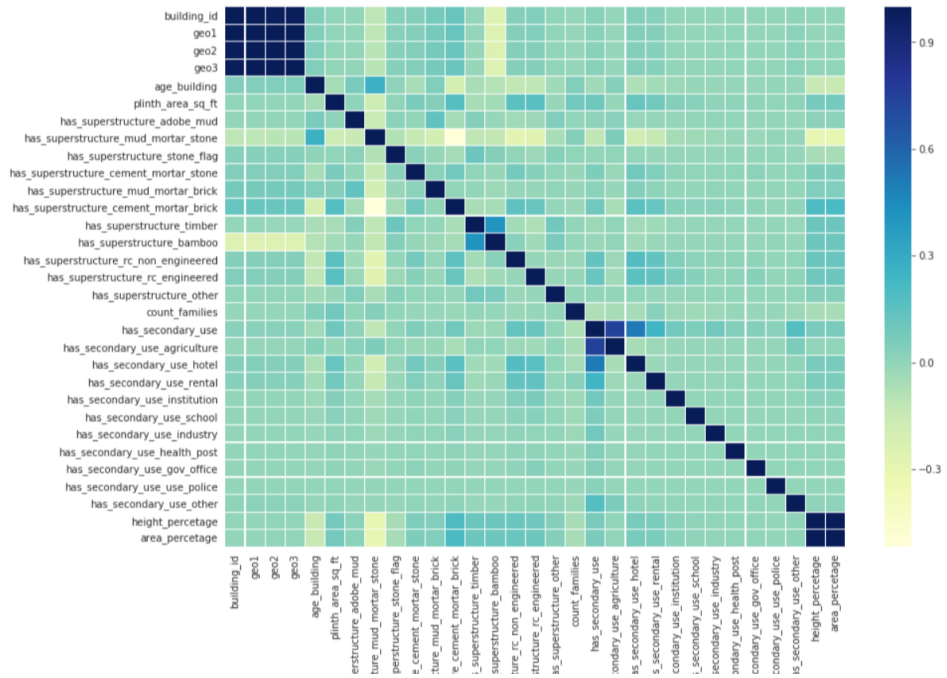
In [9]:
```
print('Number of properties with low damage:{0:.2f}%'.format((len(train[train['damage_grade']=='Low'])/len(train))*100))
print('Number of properties with medium damage:{0:.2f}%'.format((len(train[train['damage_grade']=='Medium'])/len(train))*100))
print('Number of properties with high damage:{0:.2f}%'.format((len(train[train['damage_grade']=='High'])/len(train))*100))
```

```
Number of properties with low damage:21.79%
Number of properties with medium damage:17.90%
Number of properties with high damage:60.31%
```

Checking for correletion:

In [30]:
```
corrmat=train.corr(method='spearman')
f,ax=plt.subplots(figsize=(14,10))
sns.heatmap(corrmat,ax=ax,cmap="YlGnBu", linewidths=0.1)
```

Out[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23b83099400>
```



Oservations:

- height percentage is highly correleted with number of floors.
- All other predicters has low correletion value among themselves.

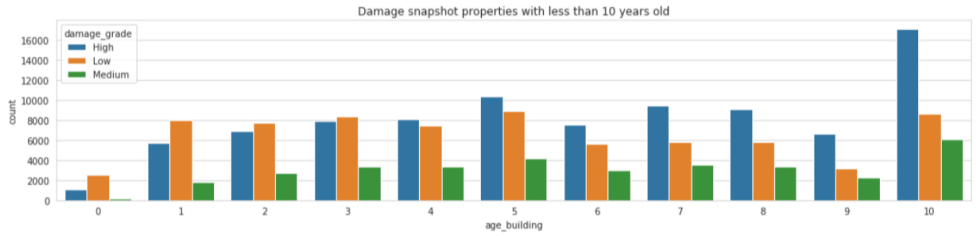
Feature Analysis : Age

In [6]:
```
train.age_building.describe()
```

Out[6]:
```
count    609674.000000
mean         24.336650
std          65.106458
min           0.000000
25%           9.000000
50%          16.000000
75%          27.000000
max         999.000000
Name: age_building, dtype: float64
```

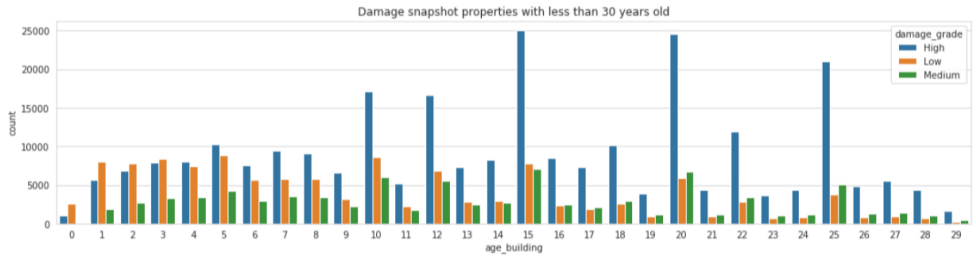
In [97]:
```
plt.figure(figsize=(18,8))
plt.subplot(211)
plt.title('Damage snapshot properties with less than 10 years old')
tr=train[train['age_building']<=10]
sns.countplot(x='age_building',data=tr,hue='damage_grade')
```

Out[97]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23b84cfce80>
```



In [96]:
```
plt.figure(figsize=(18,9))
plt.subplot(211)
plt.title('Damage snapshot properties with less than 30 years old')
tr=train[train['age_building']<30]
sns.countplot(x='age_building',data=tr,hue='damage_grade')
```

Out[96]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23b9a35afd0>
```

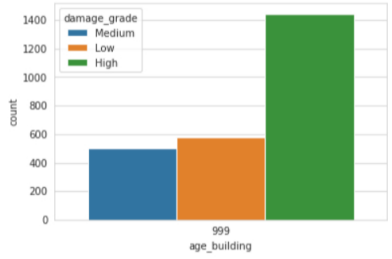


In [13]:
```
print('We have',len(train_950),'datapoints where age of the building is greater than 950 years!')
```

```
We have 2521 datapoints where age of the building is greater than 950 years!
```

In [14]:
```
train_950=train[train.age_building>950]
sns.countplot(x='age_building',data=train_950,hue='damage_grade')
```

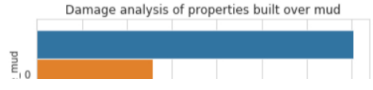
Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x19c2022dcc0>
```

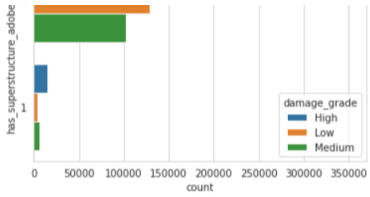


Feature Analysis : has_superstructure_adobe_mud

In [98]:
```
plt.title('Damage analysis of properties built over mud')
sns.countplot(y='has_superstructure_adobe_mud',data=train,hue='damage_grade')
```

Out[98]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23b84c83a90>
```





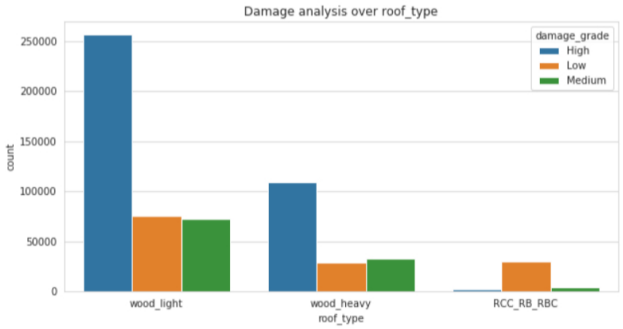
In [99]:
```
tr_no=train[train['has_superstructure_adobe_mud']==0]
tr_yes=train[train['has_superstructure_adobe_mud']==1]
print('-----------Damage snapshot of properties not built over mud----------------')
print()
print('Number of properties with low damage:{0:.2f}%'.format((len(tr_no[tr_no['damage_grade']=='Low'])/len(tr_no))*100))
print('Number of properties with medium damage:{0:.2f}%'.format((len(tr_no[tr_no['damage_grade']=='Medium'])/len(tr_no))*100))
print('Number of properties with high damage:{0:.2f}%'.format((len(tr_no[tr_no['damage_grade']=='High'])/len(tr_no))*100))
print()
print('-----------Damage snapshot of properties  built over mud----------------')
print()
print('Number of properties with low damage:{0:.2f}%'.format((len(tr_yes[tr_yes['damage_grade']=='Low'])/len(tr_yes))*100))
print('Number of properties with mdium damage:{0:.2f}%'.format((len(tr_yes[tr_yes['damage_grade']=='Medium'])/len(tr_yes))*100))
print('Number of properties with high damage:{0:.2f}%'.format((len(tr_yes[tr_yes['damage_grade']=='High'])/len(tr_yes))*100))
```

```
-----------Damage snapshot of properties not built over mud----------------

Number of properties with low damage:22.07%
Number of properties with medium damage:17.59%
Number of properties with high damage:60.34%

-----------Damage snapshot of properties  built over mud----------------

Number of properties with low damage:15.56%
Number of properties with mdium damage:24.81%
Number of properties with high damage:59.62%
```

Feature Analysis : roof_type

In [100]:
```
plt.figure(figsize=(10,5))
plt.title('Damage analysis over roof_type')
sns.countplot(x='roof_type',data=train,hue='damage_grade')
```

Out[100]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23b85a00c50>
```



In [102]:
```
tr_n=train[train['roof_type']=='wood_light']
tr_q=train[train['roof_type']=='wood_heavy']
tr_x=train[train['roof_type']=='RCC_RB_RBC']
print('-----------Damage snapshot of properties with roof type wood_light----------------')
print()
print('Number of properties with low damage:{0:.2f}%'.format((len(tr_n[tr_n['damage_grade']=='Low'])/len(tr_n))*100))
print('Number of properties with medium damage:{0:.2f}%'.format((len(tr_n[tr_n['damage_grade']=='Medium'])/len(tr_n))*100))
print('Number of properties with high damage:{0:.2f}%'.format((len(tr_n[tr_n['damage_grade']=='High'])/len(tr_n))*100))
print()
print('-----------Damage snapshot of properties with roof type wood_heavy----------------')
print()
print('Number of properties with low damage:{0:.2f}%'.format((len(tr_q[tr_q['damage_grade']=='Low'])/len(tr_q))*100))
print('Number of properties with medium damage:{0:.2f}%'.format((len(tr_q[tr_q['damage_grade']=='Medium'])/len(tr_q))*100))
print('Number of properties with high damage:{0:.2f}%'.format((len(tr_q[tr_q['damage_grade']=='High'])/len(tr_q))*100))
print()
print('-----------Damage snapshot of properties with roof type RCC_RB_RBC ----------------')
print()
print('Number of properties with low damage:{0:.2f}%'.format((len(tr_x[tr_x['damage_grade']=='Low'])/len(tr_x))*100))
print('Number of properties with medium damage:{0:.2f}%'.format((len(tr_x[tr_x['damage_grade']=='Medium'])/len(tr_x))*100))
print('Number of properties with high damage:{0:.2f}%'.format((len(tr_x[tr_x['damage_grade']=='High'])/len(tr_x))*100))
print()
```

```
-----------Damage snapshot of properties with roof type wood_light----------------

Number of properties with low damage:18.57%
Number of properties with medium damage:17.82%
Number of properties with high damage:63.61%

-----------Damage snapshot of properties with roof type wood_heavy----------------

Number of properties with low damage:16.80%
Number of properties with medium damage:19.34%
Number of properties with high damage:63.87%

-----------Damage snapshot of properties with roof type RCC_RB_RBC ----------------

Number of properties with low damage:82.12%
Number of properties with medium damage:11.94%
Number of properties with high damage:5.94%
```

Step(s):

1. Raw data has been accquired from the mentioned data source.
2. Structural,ownership and damage information has been cleaned and concatenated in order to prepare train and test data set.
3. Data has been sampled manualy in order to deal with the imbalanced data set.From the initial cleaned 700k data 100k data has been sampled of each class and a training set of 300k datapoints has been prepared for training.Stratified sampling has been maintained throughout the process.
4. Numerical variables have been normalized,categorical variables have been vectorized using Label Encoder.Data matrix has been prepared as required for

4. Numerical variables have been normalized,categorical variables have been vectorized using Label Encoder.Data matrix has been prepared as required for various machine learning and deep learning architectures.
5. Used a random model as baseline,tried machine learning models like logistic regression,linear SVM with nystrome approximation(for kernel trick),Random Forest,LGBM etc.
6. Simple models has been choosen as baseline,model complexity has been incresed gradualy.
7. A majority voting classifier has been developed using the logistic regression,random forest and light GBM model.
8. Tried various DNN architecture like LSTM,CNN but not able to beat machine learning models(light GBM).
9. GridSearchCV,simple crossvalidation etc. has been used for cross validation at various step.
10. Custom micro average metric has been developed for lightGBM and Deep learning models.
11. In practice lightGBM (0.78) and majority voting classifier(0.74) acheived best f1 score.