WolfnChaos / **Capstone-Modeling-Earthquake-Damage**

Code Issues Pull requests Actions Projects Wiki Security

main

Capstone-Modeling-Earthquake-Damage / First_Working_Notebook.ipynb

WolfnChaos ReadMe Draft History

1 contributor

3354 lines (3354 sloc) | 863 KB

Overview

A magnitude 7.8 earthquake struck Nepal on April 25, 2015, toppling multi-story buildings in Kathmandu, the capital, and creating landslides and avalanches in the Himalaya Mountains. Nearly 9,000 people died and more than 22,000 suffered injuries.

The quake was followed by hundreds of aftershocks, and only 17 days later, there was another major quake, a magnitude 7.3 temblor. Thirty-nine of the nation's 75 districts with a population of 8 million people — about a third of the national population — were affected. Hundreds of thousands of people lost everything and faced extreme poverty.

More than 600,000 homes were destroyed and more than 288,000 were damaged in the 14 worst-hit districts. Hundreds of thousands of people lost everything and faced extreme poverty and homeless.

Business Problem

The Federal Democratic Republic of Nepal wants to avoid future building damages by reinforcing homes/buildings. They are wanting to know the possible damage risk level that current homes/buildings are at. So, they can better focus their resources, and protect their citizens of Nepal if and when another major earthquake occurs.

Data Understanding

The data that will be used to predict the damage risk level comes from https://www.drivendata.org/ (https://www.drivendata.org/), while original data comes from http://eq2015.npc.gov.np/#/ (http://eq2015.npc.gov.np/#/). One of the largest dataset done on the aftermath of an earthquake.

The dataset mainly consists of 260601 rows each with information on the building structure and ownership. There are 40 columns in this dataset, where the building_id column is a unique and the target. The remaining 38 features are described in the section below.

Driven Data also obfuscated random lowercase ascii characters to the categorical variables. Using the some of the original data we should be able to find out what this varilbes are and hopefully get some insight when doing the Exploratory Data Analysis (EDA).

Target

Target	Info
Grade 1	represents low damage
Grade 2	represents a medium amount of damage
Grade 3	represents almost complete destruction

Features

Feature	Info
geo_level_1_id, geo_level_2_id, geo_level_3_id (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.
area_percentage (type: int):	normalized area of the building footprint.
height_percentage (type: int):	normalized height of the building footprint.
land_surface_condition (type: categorical):	surface condition of the land where the

	building was built. Possible values: n, o, t.
foundation_type (type: categorical):	type of foundation used while building. Possible values: h, i, r, u, w.
roof_type (type: categorical):	type of roof used while building. Possible values: n, q, x.
ground_floor_type (type: categorical):	type of the ground floor. Possible values: f, m, v, x, z.
other_floor_type (type: categorical):	type of constructions used in higher than the ground floors (except of roof). Possible values: j, q, s, x.
position (type: categorical):	position of the building. Possible values: j, o, s, t.
plan_configuration (type: categorical):	building plan configuration. Possible values: a, c, d, f, m, n, o, q, s, u.
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.
	flag variable

has_secondary_use_agriculture (type: binary):	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.	
		flag

has_secondary_use_use_police	binary	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.	

Data Preparation

Import Libraries and Tools

In [51]:
```
# Import pandas and set column display to m
ax.
import pandas as pd
pd.set_option('display.max_columns', None)

# Import matplotlib and seaborn, set style
 theme to whitegrid.
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="whitegrid")

# Import pickle
import pickle

# Import model and tranformers from sklear
n.
from sklearn.tree import DecisionTreeClassi
fier
from sklearn.model_selection import train_t
est_split, GridSearchCV
from sklearn.metrics import accuracy_score,
recall_score, precision_score, f1_score, pl
ot_confusion_matrix
from sklearn.preprocessing import OneHotEnc
oder, StandardScaler
from sklearn.compose import ColumnTransform
er
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsCla
ssifier
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRe
gression
from sklearn.ensemble import RandomForestCl
assifier
from sklearn.inspection import permutation_
importance

# Import xgboost classifer.
from xgboost import XGBClassifier

# Import SMOTE to handle class imbalacne an
d pipeline.
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as P
ipeline
```

Read in Datasets for Model

In [2]:
```
# Read in the data values.
df_values = pd.read_csv('Data/train_values.
csv')

# Read in the data labels.
df_labels = pd.read_csv('Data/train_labels.
csv')

# Merge the values and labes into one datas
et.
df = df_values.merge(df_labels, on='buildin
g_id')
df.head()
```

Out[2]:

	building_id	geo_level_1_id	geo_level_2_id	geo
0	802906	6	487	121
1	28830	8	900	281
2	94947	21	363	897
3	590882	22	418	106
4	201944	11	131	148

Checking for Missing Values

In [3]:
```
# Check of missing values.
df.isna().sum()
```

Out[3]:
```
building_id                               0
geo_level_1_id                            0
geo_level_2_id                            0
geo_level_3_id                            0
```

```
count_floors_pre_eq                       0
age                                       0
area_percentage                           0
height_percentage                         0
land_surface_condition                    0
foundation_type                           0
roof_type                                 0
ground_floor_type                         0
other_floor_type                          0
position                                  0
plan_configuration                        0
has_superstructure_adobe_mud              0
has_superstructure_mud_mortar_stone       0
has_superstructure_stone_flag             0
has_superstructure_cement_mortar_stone    0
has_superstructure_mud_mortar_brick       0
has_superstructure_cement_mortar_brick    0
has_superstructure_timber                 0
has_superstructure_bamboo                 0
has_superstructure_rc_non_engineered      0
has_superstructure_rc_engineered          0
has_superstructure_other                  0
legal_ownership_status                    0
count_families                            0
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
damage_grade                              0
dtype: int64
```

No missing values to worry about.

Checking Featuer Types

In [4]:
```
# Checking featuer types.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 260601 entries, 0 to 260600
Data columns (total 40 columns):
 #   Column
Non-Null Count   Dtype
---  ------
--------------   -----
 0   building_id
260601 non-null  int64
 1   geo_level_1_id
260601 non-null  int64
```

```
260601 non-null  int64
 2   geo_level_2_id
260601 non-null  int64
 3   geo_level_3_id
260601 non-null  int64
 4   count_floors_pre_eq
260601 non-null  int64
 5   age
260601 non-null  int64
 6   area_percentage
260601 non-null  int64
 7   height_percentage
260601 non-null  int64
 8   land_surface_condition
260601 non-null  object
 9   foundation_type
260601 non-null  object
 10  roof_type
260601 non-null  object
 11  ground_floor_type
260601 non-null  object
 12  other_floor_type
260601 non-null  object
 13  position
260601 non-null  object
 14  plan_configuration
260601 non-null  object
 15  has_superstructure_adobe_mud
260601 non-null  int64
 16  has_superstructure_mud_mortar_stone
260601 non-null  int64
 17  has_superstructure_stone_flag
260601 non-null  int64
 18  has_superstructure_cement_mortar_stone
260601 non-null  int64
 19  has_superstructure_mud_mortar_brick
260601 non-null  int64
 20  has_superstructure_cement_mortar_brick
260601 non-null  int64
 21  has_superstructure_timber
260601 non-null  int64
 22  has_superstructure_bamboo
260601 non-null  int64
 23  has_superstructure_rc_non_engineered
260601 non-null  int64
 24  has_superstructure_rc_engineered
260601 non-null  int64
 25  has_superstructure_other
260601 non-null  int64
 26  legal_ownership_status
260601 non-null  object
 27  count_families
260601 non-null  int64
 28  has_secondary_use
260601 non-null  int64
 29  has_secondary_use_agriculture
260601 non-null  int64
 30  has_secondary_use_hotel
```

```
260601 non-null  int64
 31  has_secondary_use_rental
260601 non-null  int64
 32  has_secondary_use_institution
260601 non-null  int64
 33  has_secondary_use_school
260601 non-null  int64
 34  has_secondary_use_industry
260601 non-null  int64
 35  has_secondary_use_health_post
260601 non-null  int64
 36  has_secondary_use_gov_office
260601 non-null  int64
 37  has_secondary_use_use_police
260601 non-null  int64
 38  has_secondary_use_other
260601 non-null  int64
 39  damage_grade
260601 non-null  int64
dtypes: int64(32), object(8)
memory usage: 81.5+ MB
```

Some of the features that are mean to be Booleans are classified as int64.

In [5]:
```
# Making list of all colums that should be
 a boolean type.
bool_list = ['has_superstructure_adobe_mud'
, 'has_superstructure_mud_mortar_stone', 'h
as_superstructure_stone_flag',
             'has_superstructure_cement_mor
tar_stone', 'has_superstructure_mud_mortar_
brick',
             'has_superstructure_cement_mor
tar_brick', 'has_superstructure_timber', 'h
as_superstructure_bamboo',
             'has_superstructure_rc_non_eng
ineered', 'has_superstructure_rc_engineere
d', 'has_superstructure_other',
             'has_secondary_use', 'has_seco
ndary_use_agriculture', 'has_secondary_use_
hotel', 'has_secondary_use_rental',
             'has_secondary_use_institutio
n', 'has_secondary_use_school', 'has_second
ary_use_industry',
             'has_secondary_use_health_pos
t', 'has_secondary_use_gov_office', 'has_se
condary_use_use_police',
             'has_secondary_use_other']

# Loop though list to change types to boole
an.
for name in bool_list:
    df[name] = df[name].astype('bool')

# Checking types one more time.
df.info()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 260601 entries, 0 to 260600
Data columns (total 40 columns):
 #   Column
Non-Null Count   Dtype
---  ------
--------------   -----
 0   building_id
260601 non-null  int64
 1   geo_level_1_id
260601 non-null  int64
 2   geo_level_2_id
260601 non-null  int64
 3   geo_level_3_id
260601 non-null  int64
 4   count_floors_pre_eq
260601 non-null  int64
 5   age
260601 non-null  int64
 6   area_percentage
260601 non-null  int64
 7   height_percentage
260601 non-null  int64
 8   land_surface_condition
260601 non-null  object
 9   foundation_type
260601 non-null  object
 10  roof_type
260601 non-null  object
 11  ground_floor_type
260601 non-null  object
 12  other_floor_type
260601 non-null  object
 13  position
260601 non-null  object
 14  plan_configuration
260601 non-null  object
 15  has_superstructure_adobe_mud
260601 non-null  bool
 16  has_superstructure_mud_mortar_stone
260601 non-null  bool
 17  has_superstructure_stone_flag
260601 non-null  bool
 18  has_superstructure_cement_mortar_stone
260601 non-null  bool
 19  has_superstructure_mud_mortar_brick
260601 non-null  bool
 20  has_superstructure_cement_mortar_brick
260601 non-null  bool
 21  has_superstructure_timber
260601 non-null  bool
 22  has_superstructure_bamboo
260601 non-null  bool
 23  has_superstructure_rc_non_engineered
260601 non-null  bool
 24  has_superstructure_rc_engineered
260601 non-null  bool
```

```
260601 non-null  bool
 25  has_superstructure_other
260601 non-null  bool
 26  legal_ownership_status
260601 non-null  object
 27  count_families
260601 non-null  int64
 28  has_secondary_use
260601 non-null  bool
 29  has_secondary_use_agriculture
260601 non-null  bool
 30  has_secondary_use_hotel
260601 non-null  bool
 31  has_secondary_use_rental
260601 non-null  bool
 32  has_secondary_use_institution
260601 non-null  bool
 33  has_secondary_use_school
260601 non-null  bool
 34  has_secondary_use_industry
260601 non-null  bool
 35  has_secondary_use_health_post
260601 non-null  bool
 36  has_secondary_use_gov_office
260601 non-null  bool
 37  has_secondary_use_use_police
260601 non-null  bool
 38  has_secondary_use_other
260601 non-null  bool
 39  damage_grade
260601 non-null  int64
dtypes: bool(22), int64(10), object(8)
memory usage: 43.2+ MB
```

Categorical Column Value Names Data

All the categorical columns are represented by an obfuscated random lowercase ascii character, will use the some of the original data set to find out what the letters represent.

In [6]:

```
# Read in building structure data.
df_building_structure = pd.read_csv('Data/c
sv_building_structure.csv')

# Read in building owner one use data set.
df_building_ownership_and_use = pd.read_csv
('Data/csv_building_ownership_and_use.csv')

# Merge the structure and owner into one da
ta set.
df_og = df_building_structure.merge(df_buil
ding_ownership_and_use, on=['building_id',
'district_id',

'vdcmun_id', 'ward_id'])
df_og.head()
```

```
df_og.head()
```

Out[6]:

	building_id	district_id	vdcmun_id	ward_id
0	120101000011	12	1207	120703
1	120101000021	12	1207	120703
2	120101000031	12	1207	120703
3	120101000041	12	1207	120703
4	120101000051	12	1207	120703

In [7]:

```
# Making a list of all categorical columns.
categorical_list = ['land_surface_condition', 'foundation_type', 'roof_type', 'ground_floor_type',
                    'other_floor_type', 'position', 'plan_configuration', 'legal_ownership_status']

# Loop though each name in list and print the counts for each value.
for name in categorical_list:
    print('-----------------')
    print(name)
    print('***')
    print(df_og[name].value_counts())
    print('***')
    print(df[name].value_counts())
```

```
-----------------
land_surface_condition
***
Flat                631675
Moderate slope      105640
Steep slope          24791
Name: land_surface_condition, dtype: int64
***
t    216757
n     35528
o      8316
Name: land_surface_condition, dtype: int64
-----------------
foundation_type
***
Mud mortar-Stone/Brick    628716
Bamboo/Timber              57473
Cement-Stone/Brick         39245
RC                         32120
Other                       4552
```

```
Name: foundation_type, dtype: int64
***
r    219196
w     15118
u     14260
i     10579
h      1448
Name: foundation_type, dtype: int64
-----------------
roof_type
***
Bamboo/Timber-Light roof    503748
Bamboo/Timber-Heavy roof    213774
RCC/RB/RBC                   44584
Name: roof_type, dtype: int64
***
n    182842
q     61576
x     16183
Name: roof_type, dtype: int64
-----------------
ground_floor_type
***
Mud            618217
RC              73149
Brick/Stone     66093
Timber           3594
Other            1053
Name: ground_floor_type, dtype: int64
***
f    209619
x     24877
v     24593
z      1004
m       508
Name: ground_floor_type, dtype: int64
-----------------
other_floor_type
***
TImber/Bamboo-Mud    486907
Timber-Planck        123635
Not applicable       118822
RCC/RB/RBC            32742
Name: other_floor_type, dtype: int64
***
q    165282
x     43448
j     39843
s     12028
Name: other_floor_type, dtype: int64
-----------------
position
***
Not attached       604453
Attached-1 side    129432
Attached-2 side     26910
Attached-3 side      1310
Name: position, dtype: int64
```

```
***
s    202090
t     42896
j     13282
o      2333
Name: position, dtype: int64
-----------------
plan_configuration
***
Rectangular                            731257
Square                                  17576
L-shape                                 10079
T-shape                                   969
Multi-projected                           940
Others                                    518
U-shape                                   448
E-shape                                   140
Building with Central Courtyard            98
H-shape                                    80
Name: plan_configuration, dtype: int64
***
d    250072
q      5692
u      3649
s       346
c       325
a       252
o       159
m        46
n        38
f        22
Name: plan_configuration, dtype: int64
-----------------
legal_ownership_status
***
Private          731387
Public            19232
Institutional      7823
Other              3664
Name: legal_ownership_status, dtype: int64
***
v    250939
a      5512
w      2677
r      1473
Name: legal_ownership_status, dtype: int64
```

Look at the print out it looks like most of the random values in the df dataset do correspond to the df_og dataset. Will use the values to replace the random values.

In [8]:
```
# Make a function of replace value in categ
orical columns.
def replace_categorical_value(name):
    '''
    This function will take in a column nam
```

```
e and loop thought the value count index fr
om the df dataset and replace the
    the value with the value count index fo
r the df_og data set.
    '''
    for i in range(len(df[name].value_count
s().index)):
        df[name].replace({df[name].value_co
unts().index[i]:

df_og[name].value_counts().index[i]}, inpla
ce=True)

# Look though each value in list and replac
e them.
for cat_value in categorical_list:
    replace_categorical_value(cat_value)

df.head()
```

Out[8]:

	building_id	geo_level_1_id	geo_level_2_id	geo
0	802906	6	487	121
1	28830	8	900	281
2	94947	21	363	897
3	590882	22	418	106
4	201944	11	131	148

Exploratory Data Analysis (EDA)

Class Counts

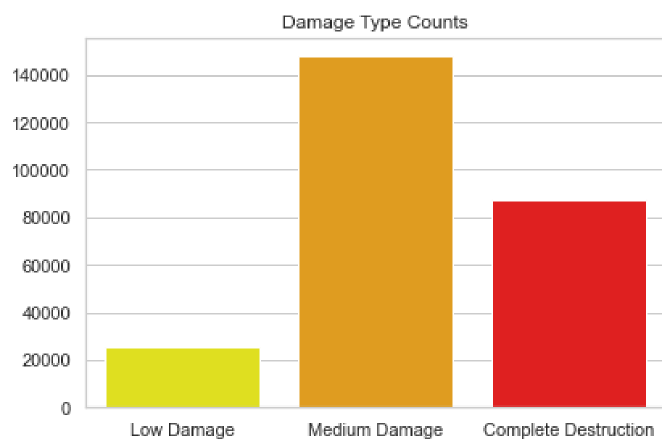
Lets take a look at are target classes and see what kind of counts we have.

In [9]:
```
# Looking at label counts.
df['damage_grade'].value_counts()
```

Out[9]:
```
2    148259
3     87218
1     25124
Name: damage_grade, dtype: int64
```

In [10]:
```
# Making a plot to visualize label counts
```

In [10]:
```
# Making a plot to visualize label counts.
fig, ax = plt.subplots()

# Defining x & y values.
x = df['damage_grade'].value_counts().index
y = df['damage_grade'].value_counts().value
s

# Making list for label names and colors.
labels = ['Low Damage', 'Medium Damage', 'C
omplete Destruction']
color = ['yellow','orange', 'red']

#Making bar plot.
ax = sns.barplot(x=x, y=y, palette=color)
ax.set_xticklabels(labels)
ax.set_title('Damage Type Counts')
ax.set_facecolor('white')
plt.tight_layout()

#Saving plot to Images folder.
fig.savefig('Images/classes.png')
```



Looks like are classes are imbalance this may cuase an issues down the road when we start training are models.

Ploting Categorical Columns

In [160]:
```
# Define a function to loop thought a list
 of column names.
def countplot_loop(column_list,rotation):
    '''
    This function will loop thought a given
list
    and return a countplot for each column
 name in
    a list and then save that plot to the I
mages folder.
    '''
    # Making a loop to go though each name
 in a list.
    for name in column_list:
```

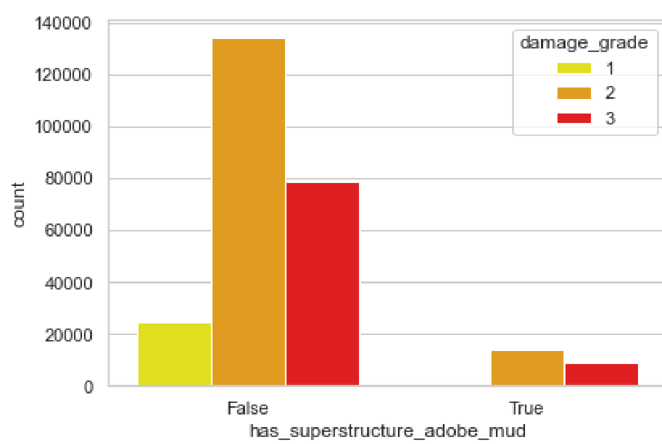
```
    for name in column_list:
        # Making figure.
        fig, ax = plt.subplots()

        # List of colors.
        color = ['yellow','orange', 'red']

        # Making countplot.
        sns.countplot(x=name, hue='damage_g
rade', palette=color, data=df, ax=ax)
        plt.xticks(rotation=rotation)
        plt.tight_layout()

        #Saving plot ot Image folder.
        fig.savefig(f'Images/{name}.png')
```

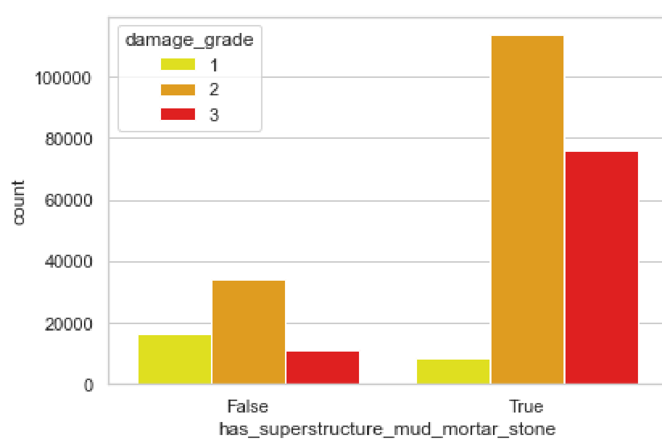
In [ ]:

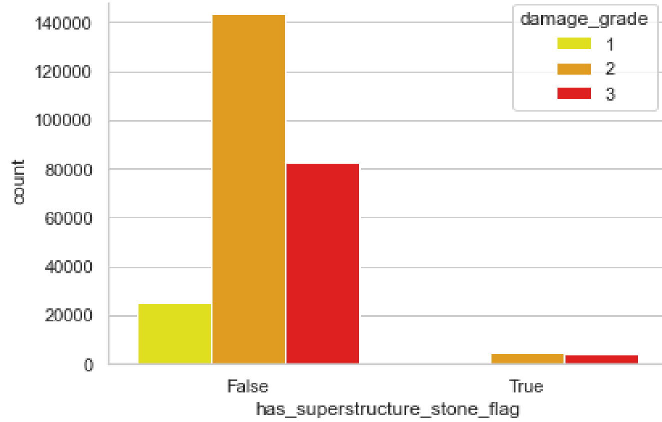
Superstructure Columns

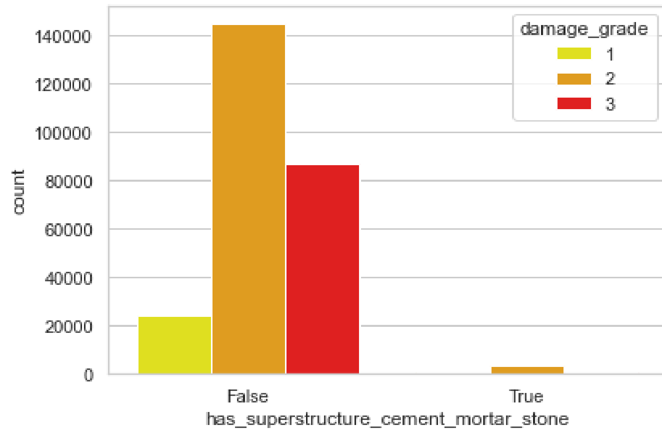
In [161]:

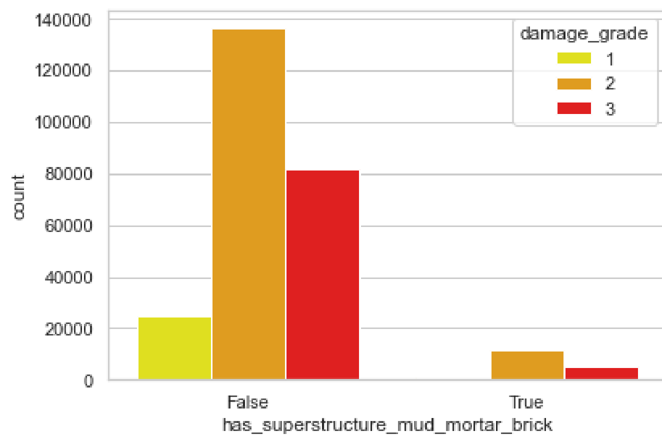
```
# Making list of only the superstucture col
umns.
column_superstructure = list(df.columns)[15
:26]

# Using function to visualize columns as a
 countplot.
countplot_loop(column_superstructure, 0)
```

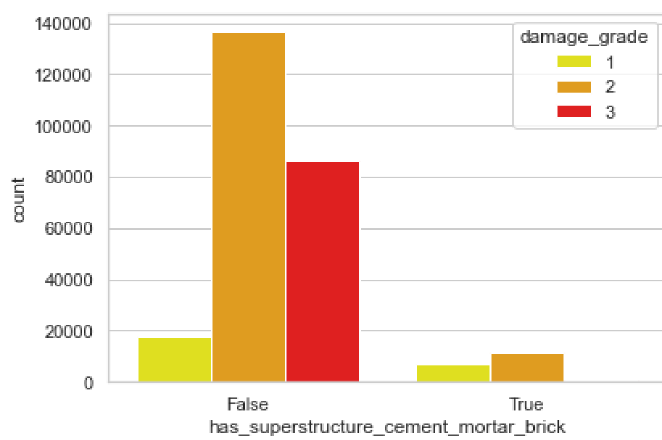


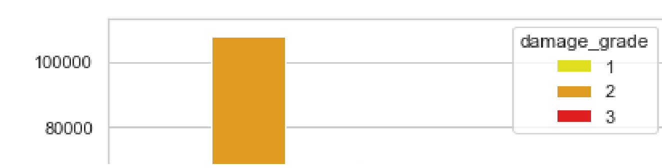


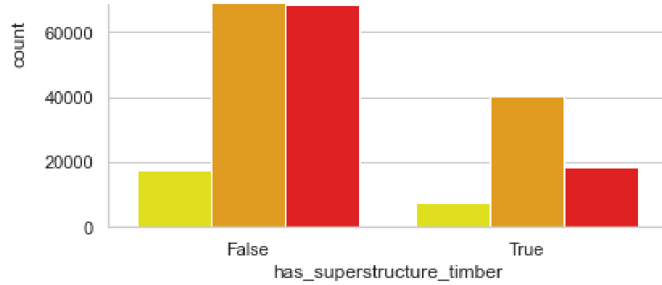

damage_grade
1
2
3
count
140000
120000
100000
80000
60000
40000
20000
0
False
True
has_superstructure_stone_flag


damage_grade
1
2
3
count
140000
120000
100000
80000
60000
40000
20000
0
False
True
has_superstructure_cement_mortar_stone

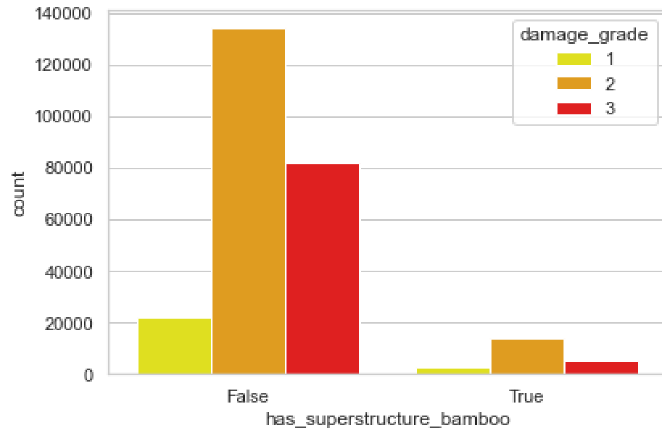

damage_grade
1
2
3
count
140000
120000
100000
80000
60000
40000
20000
0
False
True
has_superstructure_mud_mortar_brick


damage_grade
1
2
3
count
140000
120000
100000
80000
60000
40000
20000
0
False
True
has_superstructure_cement_mortar_brick

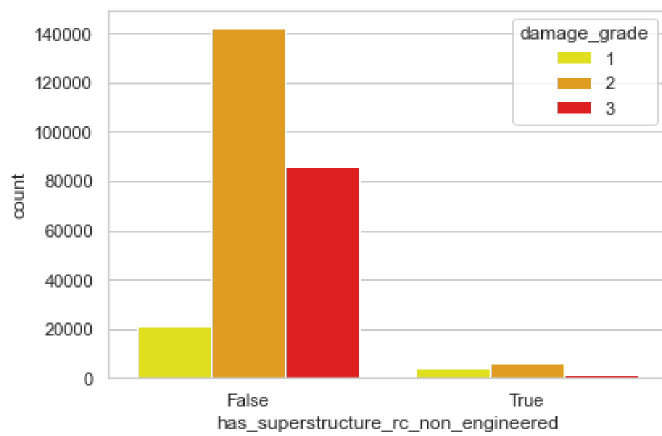

damage_grade
1
2
3
100000
80000



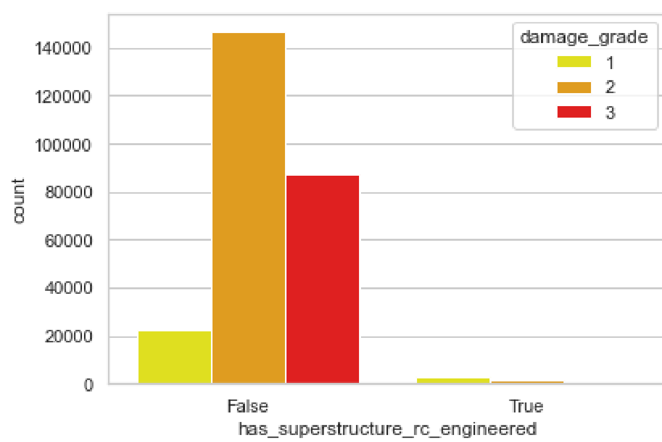
count
60000
40000
20000
0
False
True
has_superstructure_timber



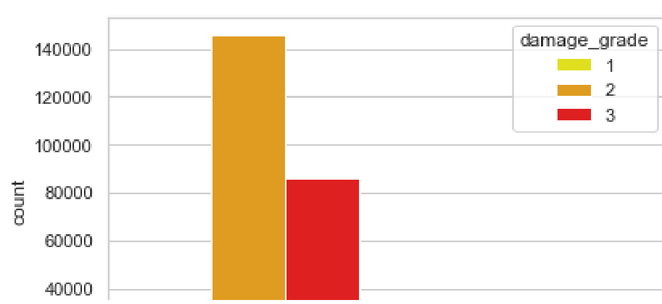
damage_grade
1
2
3
count
has_superstructure_bamboo



damage_grade
1
2
3
count
has_superstructure_rc_non_engineered



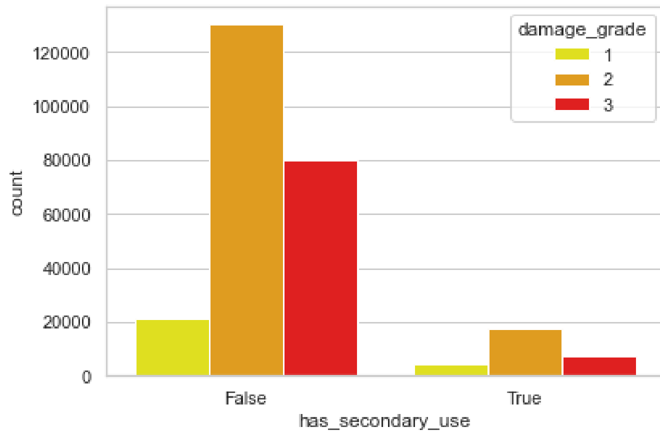
damage_grade
1
2
3
count
has_superstructure_rc_engineered

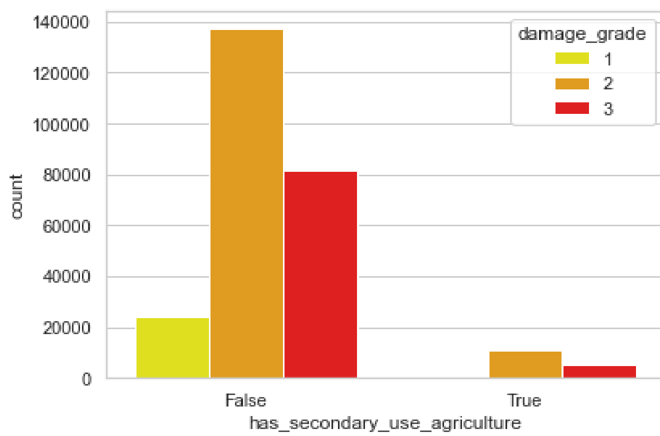


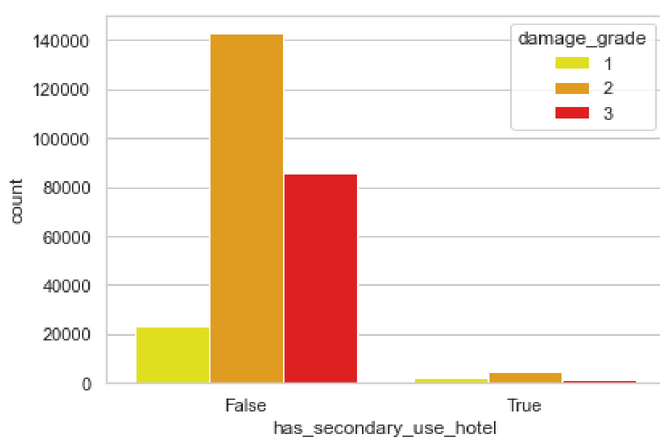
damage_grade
1
2
3
count

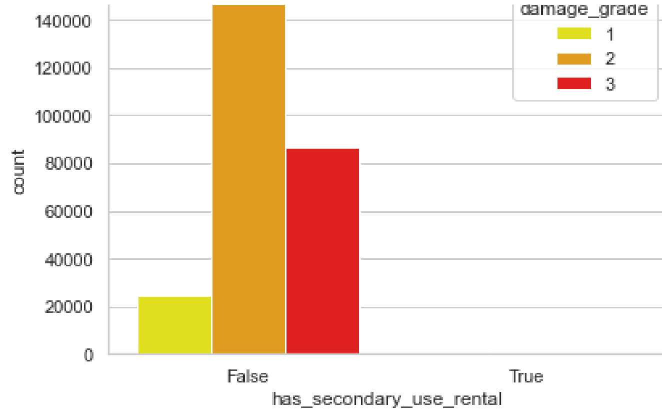
Secondary Columns

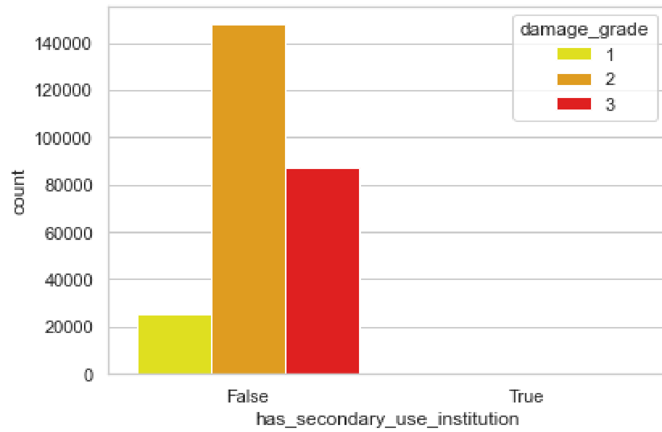
In [162]:
```
# Making list of only the secondary use col
umns.
column_secondary = list(df.columns)[28:-1]

# Using function to visualize columns as a
 countplot.
countplot_loop(column_secondary, 0)
```

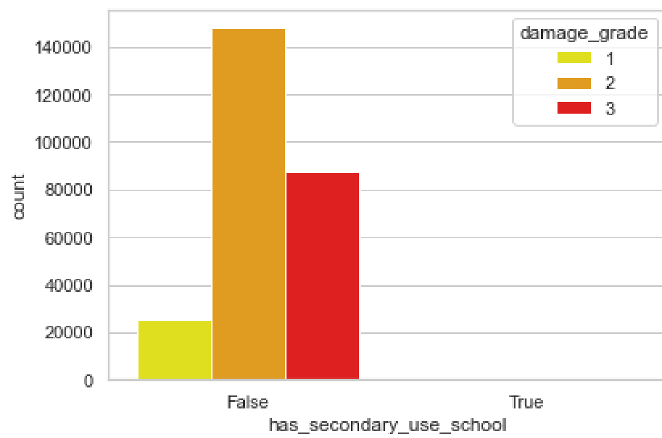


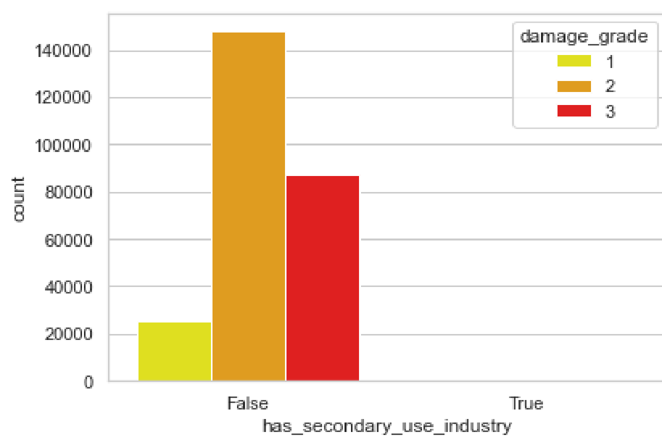


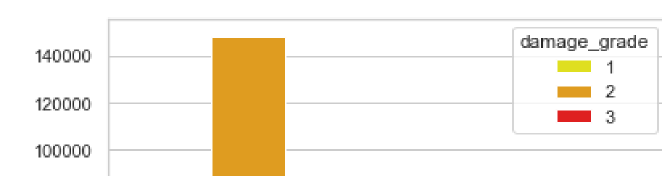


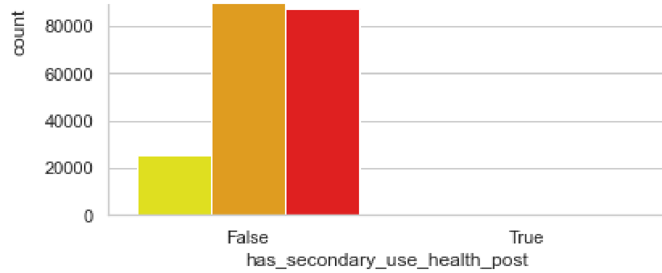

damage_grade
1
2
3
count
140000
120000
100000
80000
60000
40000
20000
0
False
True
has_secondary_use_rental

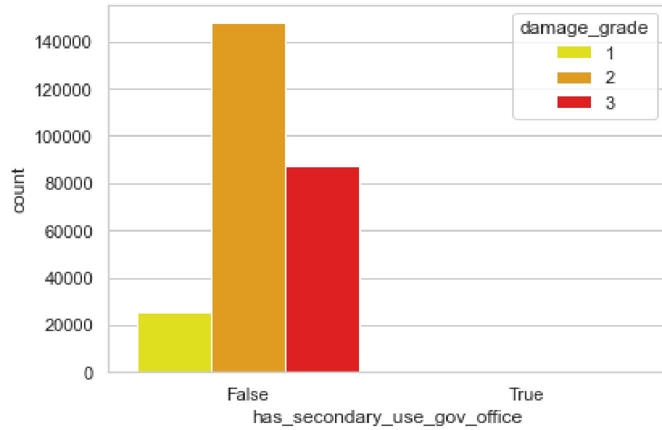

damage_grade
1
2
3
count
140000
120000
100000
80000
60000
40000
20000
0
False
True
has_secondary_use_institution

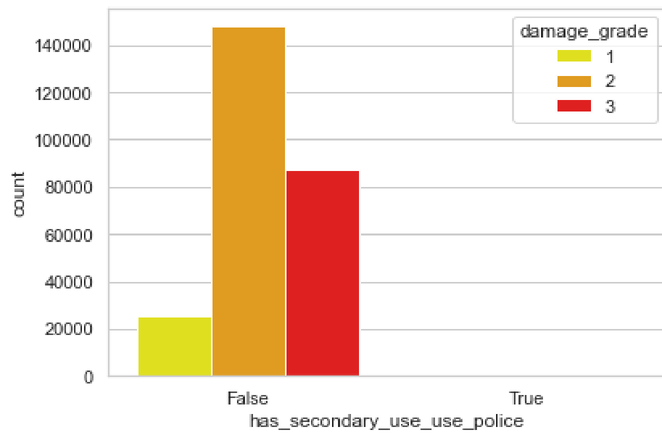

damage_grade
1
2
3
count
140000
120000
100000
80000
60000
40000
20000
0
False
True
has_secondary_use_school

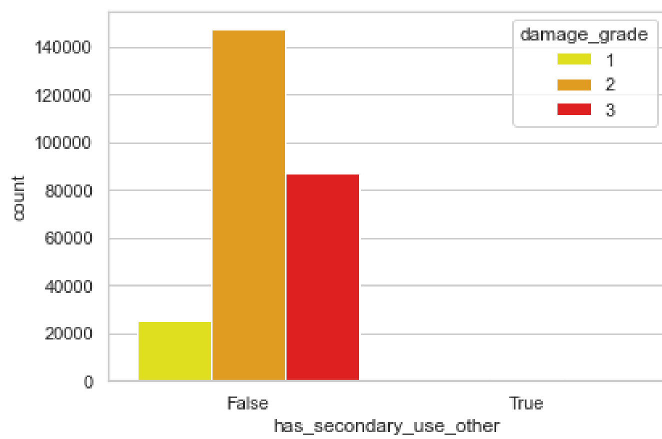

damage_grade
1
2
3
count
140000
120000
100000
80000
60000
40000
20000
0
False
True
has_secondary_use_industry


damage_grade
1
2
3
140000
120000
100000





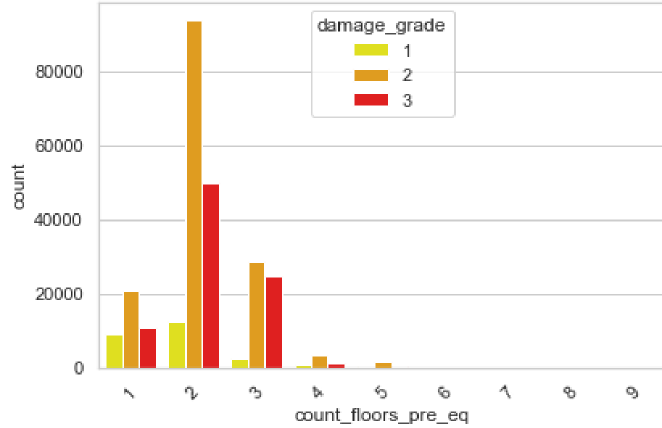


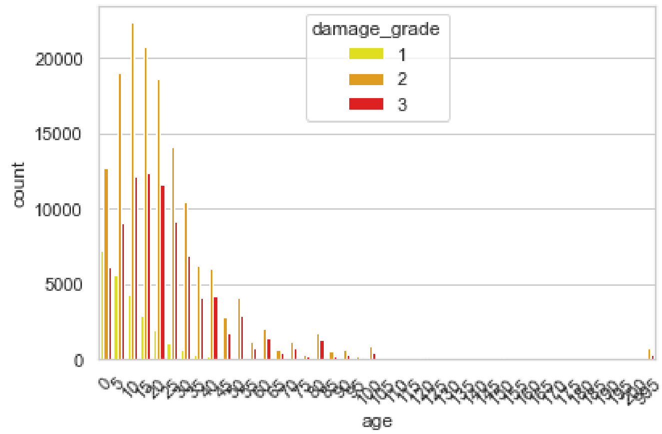


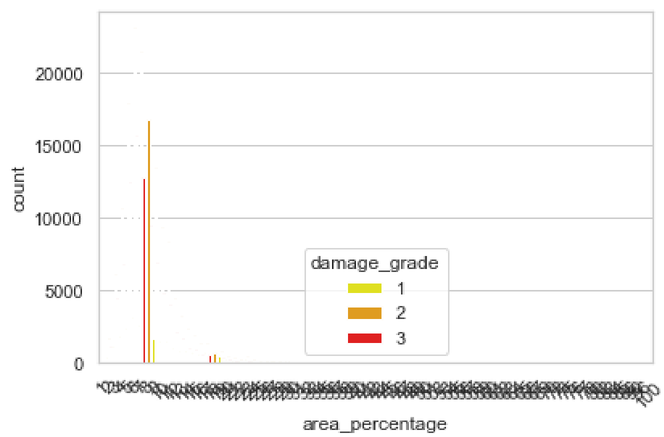
Other Columns

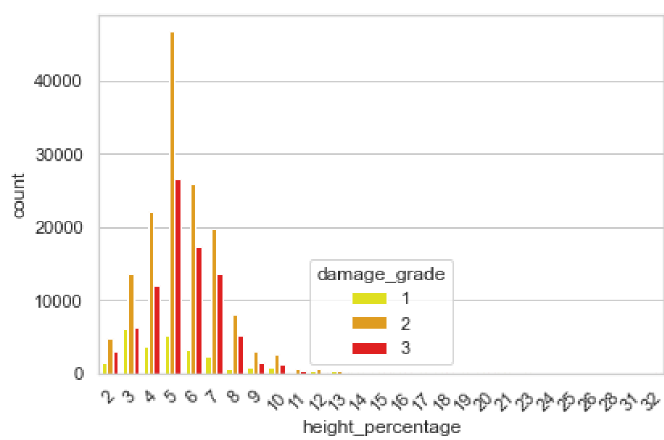
In [163]:
```
# Making list of other columns.
column_other = list(df.columns)[4:15] + lis
t(df.columns)[26:28]

# Using function to visualize columns as a
```

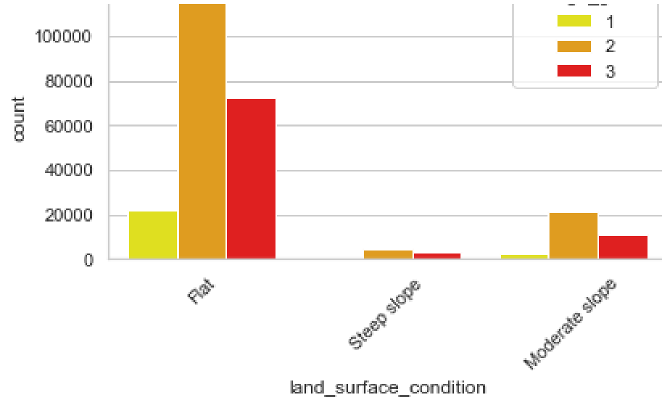
```
 countplot.
countplot_loop(column_other, 45)
```

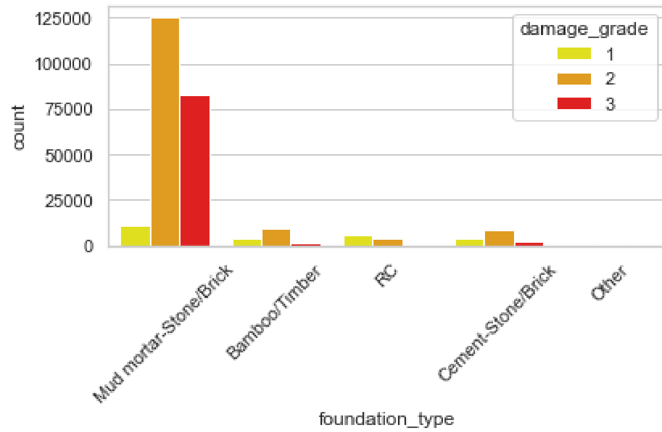


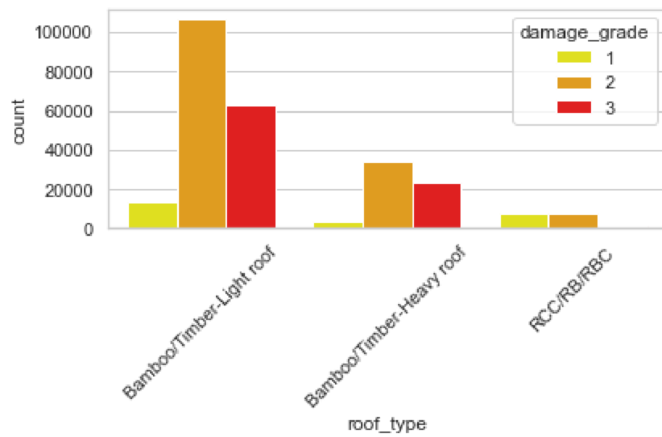


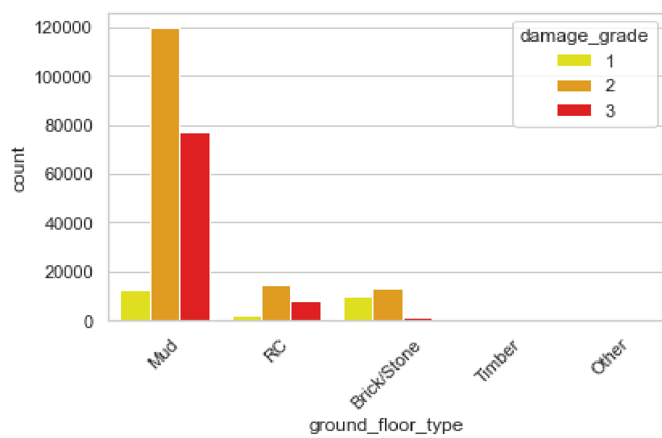




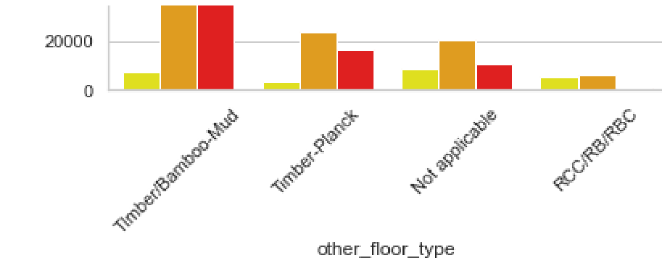

1
2
3
100000
80000
60000
40000
20000
0
count
Flat
Steep slope
Moderate slope
land_surface_condition

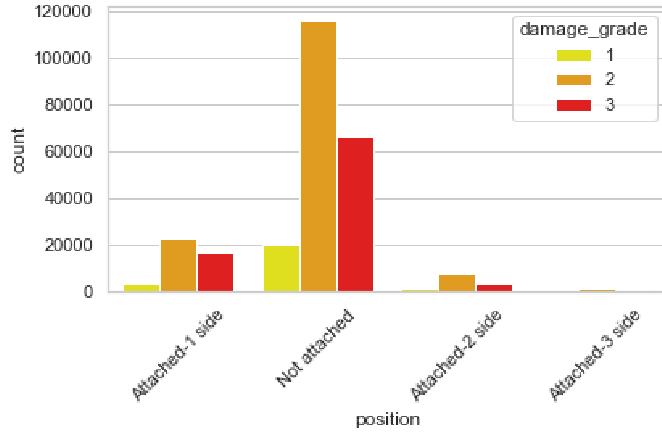

damage_grade
1
2
3
125000
100000
75000
50000
25000
0
count
Mud mortar-Stone/Brick
Bamboo/Timber
RC
Cement-Stone/Brick
Other
foundation_type

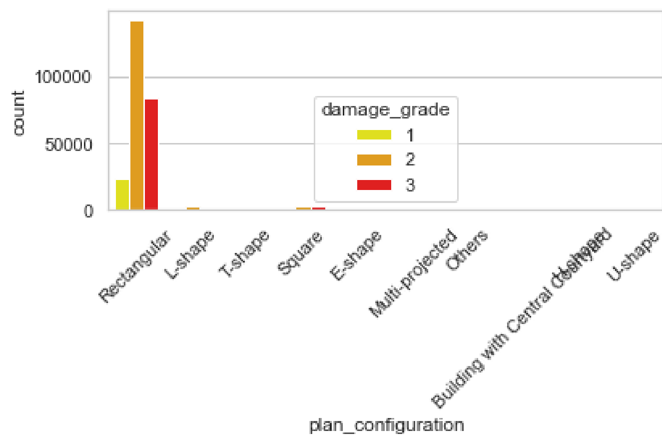

damage_grade
1
2
3
100000
80000
60000
40000
20000
0
count
Bamboo/Timber-Light roof
Bamboo/Timber-Heavy roof
RCC/RB/RBC
roof_type

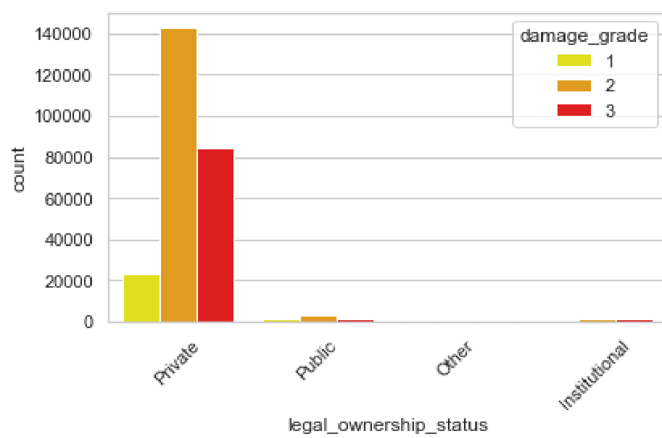

damage_grade
1
2
3
120000
100000
80000
60000
40000
20000
0
count
Mud
RC
Brick/Stone
Timber
Other
ground_floor_type

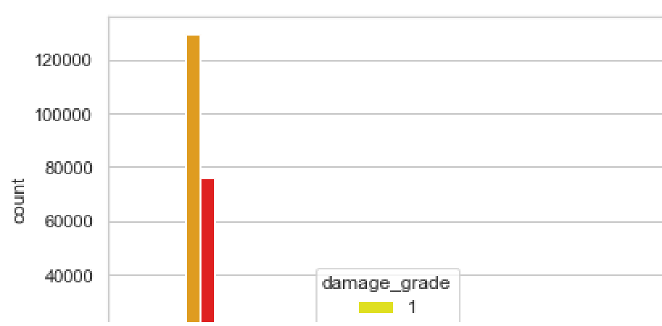

damage_grade
1
2
3
100000
80000
60000
40000
count

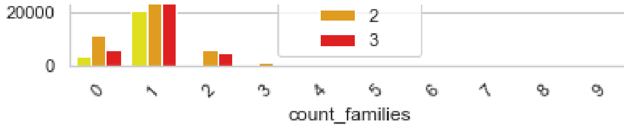

20000
0
Timber/Bamboo-Mud
Timber-Planck
Not applicable
RCC/RB/RBC
other_floor_type


120000
100000
80000
60000
40000
20000
0
count
damage_grade
1
2
3
Attached-1 side
Not attached
Attached-2 side
Attached-3 side
position


100000
50000
0
count
damage_grade
1
2
3
Rectangular
L-shape
T-shape
Square
E-shape
Multi-projected
Others
Building with Central Courtyard
H-shape
U-shape
plan_configuration


140000
120000
100000
80000
60000
40000
20000
0
count
damage_grade
1
2
3
Private
Public
Other
Institutional
legal_ownership_status


120000
100000
80000
60000
40000
count
damage_grade
1



First Simple Model (FSM)

For the FSM we will use just the numerical and boolean columns and the classifier we will use a Decision Tree Classifer. First things first, we need to split that data by doing a train test split.

In [15]:
```
# Difine the X and y values for fsm.
X_fsm = df.drop(['building_id', 'land_surface_condition', 'foundation_type',
         'roof_type', 'ground_floor_type', 'other_floor_type', 'position', 'plan_configuration',
         'legal_ownership_status', 'damage_grade'], axis=1)
y_fsm = df['damage_grade']

# Preform the train test split for fsm.
X_train_fsm, X_test_fsm, y_train_fsm, y_test_fsm = train_test_split(X_fsm, y_fsm, random_state=42, stratify=y_fsm)
```

In [16]:
```
# Instantiate decision tree classifier for a fsm.
fsm_dt = DecisionTreeClassifier(random_state=42)

# Fir the X and y train date to the fsm.
fsm_dt.fit(X_train_fsm, y_train_fsm)
```

Out[16]: DecisionTreeClassifier(random_state=42)

In [19]:
```
# Making function for printing scores.
def print_scores(model, X_train, X_test, y_train, y_test):
    '''
    This function will print the scoure for both train and test.
    '''
    # Predicting labels for both train and test data.
    y_hat_train = model.predict(X_train)
    y_hat_test = model.predict(X_test)

    # Print out scores.
    print(f'        Model Scores')
    print('-----------------------------')
    print(' Accuracy:', accuracy_score(y_train, y_hat_train))
    print('   Recall:', recall_score(y_trai
```

```
n, y_hat_train, average = 'macro'))
    print('Precision:', precision_score(y_t
rain, y_hat_train, average = 'macro'))
    print('       F1:', f1_score(y_train, y
_hat_train, average = 'macro'))
    print('----------------------------')
    print(' Accuracy:', accuracy_score(y_te
st, y_hat_test))
    print('   Recall:', recall_score(y_test
, y_hat_test, average = 'macro'))
    print('Precision:', precision_score(y_t
est, y_hat_test, average = 'macro'))
    print('       F1:', f1_score(y_test, y_
hat_test, average = 'macro'))
```

In [20]:

```
# Print out fsm scores.
print_scores(fsm_dt, X_train_fsm, X_test_fs
m, y_train_fsm, y_test_fsm)
```

```
        Model Scores
-----------------------------
 Accuracy: 0.9742082374008698
   Recall: 0.9746926003984552
Precision: 0.969726887943331
       F1: 0.9718795910888217
-----------------------------
 Accuracy: 0.6604503384445365
   Recall: 0.6149900294776912
Precision: 0.6073121797582424
       F1: 0.6109604665814518
```

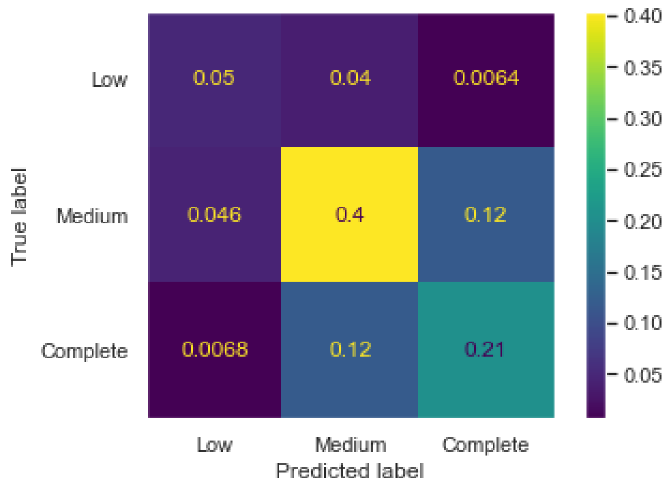
In [130]:

```
# Making a function of confusion matrix.
def plot_cm(model, X, y):
    '''
    This function with change the sns theme
back to plain white,
    plot a confusion matrix and then return
the sns theme back to whitegrid.
    '''
    # Setting theme back to white for confu
sion matrix.
    sns.set_theme(style="dark")

    # Setting confusion matrix as a variabl
e.
    plot = plot_confusion_matrix(model, X,
y, normalize='all', display_labels=['Low',
'Medium', 'Complete']);

    # Returning theme back to whitegrid.
    sns.set_theme(style="whitegrid")

    #Return plot.
    return plot
```

In [131]:

```
# Plotting fsm confusion matrix.
plot_cm(fsm_dt, X_test_fsm, y_test_fsm)
```

Out[131]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x28efaf3b220>



For the FSM 16.6% of the testing data is classified as a false negative.

Preprocessing

Setting up preprocessing for vanilla models.

In [21]:
```
# Difine the X and y values for vms
X = df.drop(['building_id', 'damage_grade'], axis=1)
y = df['damage_grade']

# Preform the train test split for vms.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, stratify=y)

# Making a list of all numeric columns.
num_list = ['geo_level_1_id', 'geo_level_2_id', 'geo_level_3_id', 'count_floors_pre_eq', 'age', 'area_percentage',
            'height_percentage', 'count_families']

# Making a list of all boolean columns.
bool_list =['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',
            'has_secondary_use', 'has_secon
dary_use_agriculture', 'has_secondary_use_h
otel',
            'has_secondary_use_rental', 'ha
s_secondary_use_institution', 'has_secondar
y_use_school',
            'has_secondary_use_industry',
'has_secondary_use_health_post', 'has_secon
dary_use_gov_office',
            'has_secondary_use_use_police',
'has_secondary_use_other']

# Making a list of all categorical columns.
cat_list = ['land_surface_condition', 'foun
dation_type', 'roof_type', 'ground_floor_ty
pe', 'other_floor_type',
            'position', 'plan_configuratio
n', 'legal_ownership_status']

# Setting of column transformer for each co
lumn list.
ct = ColumnTransformer([('ohe', OneHotEncod
er(drop='first'), cat_list),
                        ('ss', SimpleImpute
r(), num_list),
                        ('si', SimpleImpute
r(), bool_list)])

# Fit and transform the train data and only
transform the test data.
X_train_ct = ct.fit_transform(X_train)
X_test_ct = ct.transform(X_test)

# Use smote to handle label imbalance.
sm = SMOTE(random_state=42)
X_train_sm, y_train_sm = sm.fit_sample(X_tr
ain_ct, y_train)
```

Vanilla Models

Now we will use the prepocessed data to train and test some vms. The models that we will be using are: KNeighborsClassifier, GaussianNB, LogisticRegression, Random Forest Classifer, and XGB Classifer.

KNeighborsClassifier

In [22]:
```
# # Instantiate KNeighborsClassifier and fi
t X and y.
# knc = KNeighborsClassifier()
# knc.fit(X_train_sm, y_train_sm)
```

```
# # Pickle model to count done when need to
re run notebook.
# with open('Pickle/kneighbors_vm', 'wb') a
s f:
#     pickle.dump(knc, f)

# Open pickled model.
with open('Pickle/kneighbors_vm', 'rb') as
f:
    knc = pickle.load(f)

# Print model's scores.
print_scores(knc, X_train_ct, X_test_ct, y_
train, y_test)
```