Exploratory-Data-Analysis (Main Notebook)

December 3, 2021

1 Exploratory Data Analysis

2 Data Science Project (Earthquake Dataset)

Project Repository

3 Understand building and land characteristics associated with earthquakes, by getting insights into data.

A Practical Project undertaken by students at Kingston University with Applied Data Programming for Group 22

*Under **Dr. Nabajeet Barman** supervision, the group members are:*

- **Aswin Vijayakumar**, **K2142045**, k2142045@kingston.ac.uk
- **Prem Sai Kanigalpula**, **K2042054**, k2042054@kingston.ac.uk
- **Puneet Sharma**, **K2055698**, k2055698@kingston.ac.uk
- **Srinivas Rao Kanisetti**, **K2101945**, k2101945@kingston.ac.uk

4 Dataset

4.1 ATTRIBUTE CLASSIFICATION OF THE DATASET

4.1.1 Geographical Attributes (input_features.csv)

- **1.INT.1** ***geo_level_1_id:*** Level 1 Geographical Region of a building, Ranges from 0-30
- **2.INT.2** ***geo_level_2_id:*** Level 2 Geographical Region of a building, Ranges from 0-1427
- **3.INT.3** ***geo_level_3_id:*** Level 3 Geographical Region of a building, Ranges from 0-12567

4.1.2 Numerical Measures (input_features.csv)

- **4.INT.1** ***count_floors_pre_eq:*** Number of floors of a building before the earthquake
- **5.INT.2** ***age:*** The building age (in years)
- **6.INT.3** ***area_percentage:*** Normalized area of building footprint
- **7.INT.4** ***height_percentage:*** Normalized height of building footprint
- **8.INT.5** ***count_families:*** Number of families that live in a building

4.1.3 Main Building/Land Characteristics (input_features.csv)

- **9.CATEGORICAL.1** ***ground_floor_type:*** type of the ground floor (GFT), Discrete: f,m,v,x,z
- **10.CATEGORICAL.2** ***other_floor_type:*** type of construction used in higher than the ground floors (except for the roof) (OFT), Discrete: j,q,s,x
- **11.CATEGORICAL.3** ***legal_ownership_status:*** legal ownership status of the land where the building was built, Discrete: a,r,v,w
- **12.CATEGORICAL.4** ***plan_configuration:*** building plan configuration, Discrete: a,c,d,f,m,n,o,q,s,u

4.1.4 Sub Building/Land Characteristics (input_features.csv)

- **13.CATEGORICAL.1** ***land_surface_condition:*** Surface condition of the land where the building was built, Discrete: n,o,t
- **14.CATEGORICAL.2** ***foundation_type:*** type of foundation used while building, Discrete: h,i,r,u,w
- **15.CATEGORICAL.3** ***roof_type:*** type of roof used while building, Discrete: n,q,x
- **16.CATEGORICAL.4** ***position:*** Position of the building, Discrete: n,o,t

4.1.5 Superstructure Construction Attributes (input_features.csv)

- **17.BINARY.1** ***has_superstructure_adobe_mud:*** flag variable that indicates if the superstructure was made of Adobe/Mud
- **18.BINARY.2** ***has_superstructure_mud_mortar_stone:*** flag variable that indicates if the superstructure was made of Mud Mortar - Stone
- **19.BINARY.3** ***has_superstructure_stone_flag:*** flag variable that indicates if the superstructure was made of Stone
- **20.BINARY.4** ***has_superstructure_cement_mortar_stone:*** flag variable that indicates if the superstructure was made of Cement Mortar - Stone
- **21.BINARY.5** ***has_superstructure_mud_mortar_brick:*** flag variable that indicates if the superstructure was made of Mud Mortar - Brick
- **22.BINARY.6** ***has_superstructure_cement_mortar_brick:*** flag variable that indicates if the superstructure was made of Cement Mortar - Brick
- **23.BINARY.7** ***has_superstructure_timber:*** flag variable that indicates if the superstructure was made of Timber
- **24.BINARY.8** ***has_superstructure_bamboo:*** flag variable that indicates if the superstructure was made of Bamboo
- **25.BINARY.9** ***has_superstructure_rc_non_engineered:*** flag variable that indicates if the superstructure was made of non-engineered reinforced concrete
- **26.BINARY.10** ***has_superstructure_rc_engineered:*** flag variable that indicates if the superstructure was made of engineered reinforced concrete
- **27.BINARY.11** ***has_superstructure_rc_engineered:*** flag variable that indicates if the superstructure was made of any other material

4.1.6 Secondary Usage Attributes (input_features.csv)

- **28.BINARY.12** ***has_secondary_use:*** flag variable that indicates if the building was used for any secondary purpose

- **29.BINARY.13** ***has_secondary_use_agriculture:*** flag variable that indicates if the building was used for agricultural purposes
- **30.BINARY.14** ***has_secondary_use_hotel:*** flag variable that indicates if the building was used as a hotel
- **31.BINARY.15** ***has_secondary_use_rental:*** flag variable that indicates if the building was used for rental purposes
- **32.BINARY.16** ***has_secondary_use_institution:*** flag variable that indicates if the building was used as a location of any institution
- **33.BINARY.17** ***has_secondary_use_school:*** flag variable that indicates if the building was used as a school
- **34.BINARY.18** ***has_secondary_use_industry:*** flag variable that indicates if the building was used for industrial purposes
- **35.BINARY.19** ***has_secondary_use_health_post:*** flag variable that indicates if the building was used as a health post
- **36.BINARY.20** ***has_secondary_use_gov_office:*** flag variable that indicates if the building was used fas a government office
- **37.BINARY.21** ***has_secondary_use_use_police:*** flag variable that indicates if the building was used as a police station
- **38.BINARY.22** ***has_secondary_use_other:*** flag variable that indicates if the building was secondarily used for other purposes

4.1.7 Damage Impact Attributes (target_values.csv)

- **39.ORDINAL.1** ***building_id:*** unique random identifier of a building
- **40.ORDINAL.2** ***damage_grade:*** represents a level of damage to a building that was hit by earthquake,
  - 1 represents low damage
  - 2 represents a medium amount of damage
  - 3 represents almost complete destruction

The dataset is a structured dataset containing information on geographical attributes and different building and land attributes/characteristics. The geo levels (geographical) attributes, designate a hierarchy of values increasing from 0 onwards at each level.

The dataset is also part of Richter's Predictor: Modeling Earthquake Damage which is about Nepal Earthquake Disaster.

4.2 Theory

(Source: Guevara, P. & L, T., 2012. Soft story and weak story in earthquake resistant design: A multidisciplinary approach. 15WCEE. Lisboa. Retrieved from https://www. iitk. ac. in/nicee/wcee/article/WCEE2012_0183. pdf.)

The Diagram shows the Shear Forces and distrubution of forces experienced by a Building due to Ground Motion. An earthquake resistant building should have such irregularities in floors (such as weak-storey and soft storey) such that such distrubutions due to shear are minimised in a building.

There are Ground Floor Type (GFT) and Other Floor Type (OFT) set of attributes which are explored in many Research Questions.

The Age of the building has been explored in several Research Questions as well as Height and Area

The Families and Number of Floors is explained to form a trend in one of the Research Questions and is key in clustering buildings.

The Superstructures and Secondary Use Buildings have been explored in this Report as RQs and Other

The aim of this report is to promote the use of standards, best practices, usage of durable materials and design of earthquake resistant buildings for increased quality and take preventative measures that may happen due to Earthquakes

5 0.1 Initial Data Analysis

[1]:
```
# install plotly
!pip install plotly==5.3.1
# part of plotly
!pip install -U kaleido
```

```
Requirement already satisfied: plotly==5.3.1 in
c:\users\burse\anaconda3\envs\studyenv\lib\site-packages (5.3.1)
Requirement already satisfied: six in
c:\users\burse\anaconda3\envs\studyenv\lib\site-packages (from plotly==5.3.1)
(1.16.0)
Requirement already satisfied: tenacity>=6.2.0 in
c:\users\burse\anaconda3\envs\studyenv\lib\site-packages (from plotly==5.3.1)
(8.0.1)
Requirement already satisfied: kaleido in
c:\users\burse\anaconda3\envs\studyenv\lib\site-packages (0.2.1)
```

[2]:
```
# import necessary libraries
import itertools
from IPython.display import SVG, Image
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from matplotlib.patches import Rectangle, Circle
from matplotlib_venn import venn2
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
from plotly.subplots import make_subplots
from plotly.offline import init_notebook_mode, iplot, plot
from scipy.stats.kde import gaussian_kde
from scipy import stats
import seaborn as sns
```

```
plt.rcParams.update({'font.size': 14})
%matplotlib inline
```

5.1 - UTILITY FUNCTIONS

Plotting Utility Functions

[3]:
```
def setup_gridspec__one_main__two_side_subplots(plt):
    # start with a square Figure
    fig = plt.figure(figsize=(16, 16))
    fig.tight_layout(pad=2.0)

    gs = fig.add_gridspec(2, 2,  width_ratios=(7,4), height_ratios=(3,7),
                          left=0.1, right=0.9, bottom=0.1, top=0.9,
                          wspace=0.15, hspace=0.15)
    ax_0_0 = fig.add_subplot(gs[1,0])

    ax1_histx = fig.add_subplot(gs[0, 0], sharex=ax_0_0)
    ax1_histy = fig.add_subplot(gs[1, 1], sharey=ax_0_0)

    return {"gridspec": gs, "ax": ax_0_0, "axx": ax1_histx, "axy": ax1_histy,␣
 ↪"fig": fig}

def setup_gridspec__four_main__two_side_subplots(plt):
    # start with a square Figure
    fig = plt.figure(figsize=(12, 16))
    fig.tight_layout(pad=5.0)

    gs = fig.add_gridspec(4, 4,  width_ratios=(8,3,8,3),␣
 ↪height_ratios=(3,7,3,7),
                          left=0.1, right=0.9, bottom=0.1, top=0.9,
                          wspace=0.15, hspace=0.15)
    ax1 = fig.add_subplot(gs[1, 0])
    ax2 = fig.add_subplot(gs[1, 2])
    ax3 = fig.add_subplot(gs[3, 0])
    ax4 = fig.add_subplot(gs[3, 2])

    axx1 = fig.add_subplot(gs[0, 0], sharex=ax1)
    axy1 = fig.add_subplot(gs[1, 1], sharey=ax1)

    axx2 = fig.add_subplot(gs[0, 2], sharex=ax2)
    axy2 = fig.add_subplot(gs[1, 3], sharey=ax2)

    axx3 = fig.add_subplot(gs[2, 0], sharex=ax3)
    axy3 = fig.add_subplot(gs[3, 1], sharey=ax3)

    axx4 = fig.add_subplot(gs[2, 2], sharex=ax4)
```

```
    axy4 = fig.add_subplot(gs[3, 3], sharey=ax4)

    return {"gridspec": gs, "ax": (ax1,ax2,ax3,ax4), "axx": (axx1, axx2, axx3,␣
↪axx4),
            "axy": (axy1, axy2, axy3, axy4), "fig": fig}

def plot_loadings_plot(plt, X_pca, df, ax, eigen_vectors=(0,1,2,3,4)):
    # obtain color palette
    palette = np.array(sns.color_palette("hls", 10))
    # Features x Dimensions, eigen vector is a column matrix, loadings for␣
↪arrow plotting
    loadings = pc.T
    # plot eigen vectors
    arrow_size, text_pos = 1.0, 1.12
    for ii,i in enumerate(eigen_vectors):
        ax.arrow(0,0,arrow_size*loadings[i,0], arrow_size*loadings[i,1], color␣
↪= palette[ii],head_width=0.01, head_length=0.01, linewidth=2, alpha=0.4)
        ax.text(loadings[i,0]*text_pos, loadings[i,1]*text_pos, df.columns[i],␣
↪color='black',
                ha='center', va='center', fontsize=12, alpha=0.65)

    return None
```

Helper Utility Functions

[4]:
```
# Make zero mean for the dataframe
def demean_data(X_df):
    return (X_df - X_df.mean(axis=0))

# returns transformed x, prin components, var explained
def principal_components_analysis(data):
    # get the original dimensions of a matrix
    dimensions = data.shape[1]
    # make zero mean of matrix
    z = demean_data(data)
    # make a matrix symmetric, invertible
    symmetric_matrix = make_a_matrix_symmetric_invertible(z)
    # find eigen values and eigen vectors
    (eigenvalues, eigenvectors) = np.linalg.eig(symmetric_matrix)  #␣
↪'right-hand'
    # returns transformed matrix
    transformed_matrix = pca_transformed(z, eigenvectors, dimensions)
    # find the principal components
    pc = eigenvectors.T
    # find explained variances
    explained_variance = np.var(transformed_matrix, axis=0, ddof=1)  # col␣
↪sample var
```

```
    # take the sum of variances to 1 degree
    sum_of_variances = np.sum(explained_variance)
    # normalise the variances (take the ratio)
    explained_variance_ratio = explained_variance / sum_of_variances
    # order everything based on explained variance ratio
    ordering = np.argsort(explained_variance_ratio)[::-1]
    # order the transformed matrix
    transformed_matrix = transformed_matrix[:,ordering]
    pc = pc[ordering,:]
    explained_variance_ratio = explained_variance_ratio[ordering]
    return transformed_matrix, pc, explained_variance_ratio

# this code will make a non-square matrix a square matrix, a symmetric matrix␣
↪as well as an invertible matrix if the determinant is non-zero
def make_a_matrix_symmetric_invertible(z):
    return np.dot(z.T, z)

# get the transformed matrix space
def pca_transformed(z, eigenvectors, dimensions):
    return np.dot(z, eigenvectors[:,0:dimensions])
```

5.2 0.2 Import the Dataset

[5]:
```
# read input data from file
df = pd.read_csv('https://gis-bucket-aswinvk28.s3.eu-west-2.amazonaws.com/adp/
↪dataset/input_features.csv')

# read target values from file
target = pd.read_csv('https://gis-bucket-aswinvk28.s3.eu-west-2.amazonaws.com/
↪adp/dataset/target_values.csv')
```

0.3 See top 10 rows

[6]:
```
# see top 10 rows
df.head(10).T
```

[6]:
```
                             0      1      2       3       4  \
building_id             802906  28830  94947  590882  201944
geo_level_1_id               6      8     21      22      11
geo_level_2_id             487    900    363     418     131
geo_level_3_id           12198   2812   8973   10694    1488
count_floors_pre_eq          2      2      2       2       3
age                         30     10     10      10      30
area_percentage              6      8      5       6       8
height_percentage            5      7      5       5       9
land_surface_condition       t      o      t       t       t
foundation_type              r      r      r       r       r
```

roof_type	n	n	n	n	n
ground_floor_type	f	x	f	f	f
other_floor_type	q	q	x	x	x
position	t	s	t	s	s
plan_configuration	d	d	d	d	d
has_superstructure_adobe_mud	1	0	0	0	1
has_superstructure_mud_mortar_stone	1	1	1	1	0
has_superstructure_stone_flag	0	0	0	0	0
has_superstructure_cement_mortar_stone	0	0	0	0	0
has_superstructure_mud_mortar_brick	0	0	0	0	0
has_superstructure_cement_mortar_brick	0	0	0	0	0
has_superstructure_timber	0	0	0	1	0
has_superstructure_bamboo	0	0	0	1	0
has_superstructure_rc_non_engineered	0	0	0	0	0
has_superstructure_rc_engineered	0	0	0	0	0
has_superstructure_other	0	0	0	0	0
legal_ownership_status	v	v	v	v	v
count_families	1	1	1	1	1
has_secondary_use	0	0	0	0	0
has_secondary_use_agriculture	0	0	0	0	0
has_secondary_use_hotel	0	0	0	0	0
has_secondary_use_rental	0	0	0	0	0
has_secondary_use_institution	0	0	0	0	0
has_secondary_use_school	0	0	0	0	0
has_secondary_use_industry	0	0	0	0	0
has_secondary_use_health_post	0	0	0	0	0
has_secondary_use_gov_office	0	0	0	0	0
has_secondary_use_use_police	0	0	0	0	0
has_secondary_use_other	0	0	0	0	0

	5	6	7	8	9
building_id	333020	728451	475515	441126	989500
geo_level_1_id	8	9	20	0	26
geo_level_2_id	558	475	323	757	886
geo_level_3_id	6089	12066	12236	7219	994
count_floors_pre_eq	2	2	2	2	1
age	10	25	0	15	0
area_percentage	9	3	8	8	13
height_percentage	5	4	6	6	4
land_surface_condition	t	n	t	t	t
foundation_type	r	r	w	r	i
roof_type	n	n	q	q	n
ground_floor_type	f	x	v	f	v
other_floor_type	q	q	x	q	j
position	s	s	s	s	s
plan_configuration	d	d	u	d	d
has_superstructure_adobe_mud	0	0	0	0	0

```
has_superstructure_mud_mortar_stone          1       1       0       1       0
has_superstructure_stone_flag                0       0       0       0       0
has_superstructure_cement_mortar_stone       0       0       0       0       0
has_superstructure_mud_mortar_brick          0       0       0       0       0
has_superstructure_cement_mortar_brick       0       0       1       0       1
has_superstructure_timber                    0       0       1       1       0
has_superstructure_bamboo                    0       0       0       0       0
has_superstructure_rc_non_engineered         0       0       0       0       0
has_superstructure_rc_engineered             0       0       0       0       0
has_superstructure_other                     0       0       0       0       0
legal_ownership_status                       v       v       v       v       v
count_families                               1       1       1       1       1
has_secondary_use                            1       0       0       0       0
has_secondary_use_agriculture                1       0       0       0       0
has_secondary_use_hotel                      0       0       0       0       0
has_secondary_use_rental                     0       0       0       0       0
has_secondary_use_institution                0       0       0       0       0
has_secondary_use_school                     0       0       0       0       0
has_secondary_use_industry                   0       0       0       0       0
has_secondary_use_health_post                0       0       0       0       0
has_secondary_use_gov_office                 0       0       0       0       0
has_secondary_use_use_police                 0       0       0       0       0
has_secondary_use_other                      0       0       0       0       0
```

[7]:
```
# see top 10 rows
target.head(10)
```

[7]:

	building_id	damage_grade
0	802906	3
1	28830	2
2	94947	3
3	590882	2
4	201944	3
5	333020	2
6	728451	3
7	475515	1
8	441126	2
9	989500	1

0.4 See bottom 10 rows

[8]:
```
# see bottom 10 rows
df.tail(10).T
```

[8]:
```
                                        260591  260592  260593  260594  \
building_id                             560805  207683  226421  159555
geo_level_1_id                              20      10       8      27
```

geo_level_2_id	368	1382	767	181
geo_level_3_id	5980	1903	8613	1537
count_floors_pre_eq	1	2	2	6
age	25	25	5	0
area_percentage	5	5	13	13
height_percentage	3	5	5	12
land_surface_condition	n	t	t	t
foundation_type	r	r	r	r
roof_type	n	n	n	n
ground_floor_type	f	f	f	f
other_floor_type	j	q	q	x
position	s	s	s	j
plan_configuration	d	d	d	d
has_superstructure_adobe_mud	0	0	0	0
has_superstructure_mud_mortar_stone	1	1	1	0
has_superstructure_stone_flag	0	0	0	0
has_superstructure_cement_mortar_stone	0	0	0	0
has_superstructure_mud_mortar_brick	0	0	0	1
has_superstructure_cement_mortar_brick	0	0	0	0
has_superstructure_timber	0	1	0	0
has_superstructure_bamboo	0	0	0	0
has_superstructure_rc_non_engineered	0	0	0	0
has_superstructure_rc_engineered	0	0	0	0
has_superstructure_other	0	0	0	0
legal_ownership_status	v	v	v	v
count_families	1	1	1	1
has_secondary_use	1	0	1	0
has_secondary_use_agriculture	1	0	1	0
has_secondary_use_hotel	0	0	0	0
has_secondary_use_rental	0	0	0	0
has_secondary_use_institution	0	0	0	0
has_secondary_use_school	0	0	0	0
has_secondary_use_industry	0	0	0	0
has_secondary_use_health_post	0	0	0	0
has_secondary_use_gov_office	0	0	0	0
has_secondary_use_use_police	0	0	0	0
has_secondary_use_other	0	0	0	0

	260595	260596	260597	260598 \
building_id	827012	688636	669485	602512
geo_level_1_id	8	25	17	17
geo_level_2_id	268	1335	715	51
geo_level_3_id	4718	1621	2060	8163
count_floors_pre_eq	2	1	2	3
age	20	55	0	55
area_percentage	8	6	6	6
height_percentage	5	3	5	7

land_surface_condition	t	n	t	t
foundation_type	r	r	r	r
roof_type	n	n	n	q
ground_floor_type	f	f	f	f
other_floor_type	q	j	q	q
position	s	s	s	s
plan_configuration	d	q	d	d
has_superstructure_adobe_mud	0	0	0	0
has_superstructure_mud_mortar_stone	1	1	1	1
has_superstructure_stone_flag	0	0	0	0
has_superstructure_cement_mortar_stone	0	0	0	0
has_superstructure_mud_mortar_brick	0	0	0	0
has_superstructure_cement_mortar_brick	0	0	0	0
has_superstructure_timber	0	0	0	0
has_superstructure_bamboo	0	0	0	0
has_superstructure_rc_non_engineered	0	0	0	0
has_superstructure_rc_engineered	0	0	0	0
has_superstructure_other	0	0	0	0
legal_ownership_status	v	v	v	v
count_families	1	1	1	1
has_secondary_use	0	0	0	0
has_secondary_use_agriculture	0	0	0	0
has_secondary_use_hotel	0	0	0	0
has_secondary_use_rental	0	0	0	0
has_secondary_use_institution	0	0	0	0
has_secondary_use_school	0	0	0	0
has_secondary_use_industry	0	0	0	0
has_secondary_use_health_post	0	0	0	0
has_secondary_use_gov_office	0	0	0	0
has_secondary_use_use_police	0	0	0	0
has_secondary_use_other	0	0	0	0

	260599	260600
building_id	151409	747594
geo_level_1_id	26	21
geo_level_2_id	39	9
geo_level_3_id	1851	9101
count_floors_pre_eq	2	3
age	10	10
area_percentage	14	7
height_percentage	6	6
land_surface_condition	t	n
foundation_type	r	r
roof_type	x	n
ground_floor_type	v	f
other_floor_type	s	q
position	j	j

```
plan_configuration                          d        d
has_superstructure_adobe_mud                0        0
has_superstructure_mud_mortar_stone         0        1
has_superstructure_stone_flag               0        0
has_superstructure_cement_mortar_stone      0        0
has_superstructure_mud_mortar_brick         0        0
has_superstructure_cement_mortar_brick      1        0
has_superstructure_timber                   0        0
has_superstructure_bamboo                   0        0
has_superstructure_rc_non_engineered        0        0
has_superstructure_rc_engineered            0        0
has_superstructure_other                    0        0
legal_ownership_status                      v        v
count_families                              1        3
has_secondary_use                           0        0
has_secondary_use_agriculture               0        0
has_secondary_use_hotel                     0        0
has_secondary_use_rental                    0        0
has_secondary_use_institution               0        0
has_secondary_use_school                    0        0
has_secondary_use_industry                  0        0
has_secondary_use_health_post               0        0
has_secondary_use_gov_office                0        0
has_secondary_use_use_police                0        0
has_secondary_use_other                     0        0
```

[9]:
```
# see bottom 10 rows
target.tail(10)
```

[9]:

	building_id	damage_grade
260591	560805	3
260592	207683	2
260593	226421	2
260594	159555	2
260595	827012	3
260596	688636	2
260597	669485	3
260598	602512	3
260599	151409	2
260600	747594	3

0.5 Checking for Duplicates

[10]:
```
# printing the duplicated rows of data in input features
df[df.duplicated()]
```

```
[10]: Empty DataFrame
Columns: [building_id, geo_level_1_id, geo_level_2_id, geo_level_3_id,
count_floors_pre_eq, age, area_percentage, height_percentage,
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,
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]
Index: []

[0 rows x 39 columns]
```

```
[11]: # printing the duplicated rows of data in target values
target[target.duplicated()]
```

```
[11]: Empty DataFrame
Columns: [building_id, damage_grade]
Index: []
```

0.6 Merging DataFrames

- Checking for dtypes
- Checking for missing values

```
[12]: # Merge feature and target variables.
join_df = pd.merge(df, target, on='building_id', how='left')
join_df.head(5)
```

```
[12]:    building_id  geo_level_1_id  geo_level_2_id  geo_level_3_id  \
0       802906               6             487           12198
1        28830               8             900            2812
2        94947              21             363            8973
3       590882              22             418           10694
4       201944              11             131            1488

   count_floors_pre_eq  age  area_percentage  height_percentage  \
0                    2   30                6                  5
1                    2   10                8                  7
2                    2   10                5                  5
```

```
3                     2   10                6                  5
4                     3   30                8                  9

  land_surface_condition foundation_type  … has_secondary_use_hotel  \
0                      t               r  …                       0
1                      o               r  …                       0
2                      t               r  …                       0
3                      t               r  …                       0
4                      t               r  …                       0

  has_secondary_use_rental has_secondary_use_institution  \
0                        0                              0
1                        0                              0
2                        0                              0
3                        0                              0
4                        0                              0

  has_secondary_use_school has_secondary_use_industry  \
0                        0                           0
1                        0                           0
2                        0                           0
3                        0                           0
4                        0                           0

   has_secondary_use_health_post  has_secondary_use_gov_office  \
0                              0                             0
1                              0                             0
2                              0                             0
3                              0                             0
4                              0                             0

   has_secondary_use_use_police  has_secondary_use_other  damage_grade
0                             0                        0             3
1                             0                        0             2
2                             0                        0             3
3                             0                        0             2
4                             0                        0             3

[5 rows x 40 columns]
```

[13]:
```
# Finding the number of rows and the number of columns in datas
join_df.shape
```

[13]: (260601, 40)

0.7 Checking for dtypes

[14]:
```
# checking for dtypes
join_df.dtypes
```

[14]:
```
building_id                                int64
geo_level_1_id                             int64
geo_level_2_id                             int64
geo_level_3_id                             int64
count_floors_pre_eq                        int64
age                                        int64
area_percentage                            int64
height_percentage                          int64
land_surface_condition                    object
foundation_type                           object
roof_type                                 object
ground_floor_type                         object
other_floor_type                          object
position                                  object
plan_configuration                        object
has_superstructure_adobe_mud               int64
has_superstructure_mud_mortar_stone        int64
has_superstructure_stone_flag              int64
has_superstructure_cement_mortar_stone     int64
has_superstructure_mud_mortar_brick        int64
has_superstructure_cement_mortar_brick     int64
has_superstructure_timber                  int64
has_superstructure_bamboo                  int64
has_superstructure_rc_non_engineered       int64
has_superstructure_rc_engineered           int64
has_superstructure_other                   int64
legal_ownership_status                    object
count_families                             int64
has_secondary_use                          int64
has_secondary_use_agriculture              int64
has_secondary_use_hotel                    int64
has_secondary_use_rental                   int64
has_secondary_use_institution              int64
has_secondary_use_school                   int64
has_secondary_use_industry                 int64
has_secondary_use_health_post              int64
has_secondary_use_gov_office               int64
has_secondary_use_use_police               int64
has_secondary_use_other                    int64
damage_grade                               int64
dtype: object
```

[15]:
```
# printing attribute information of the merged dataframe
join_df.info()
```

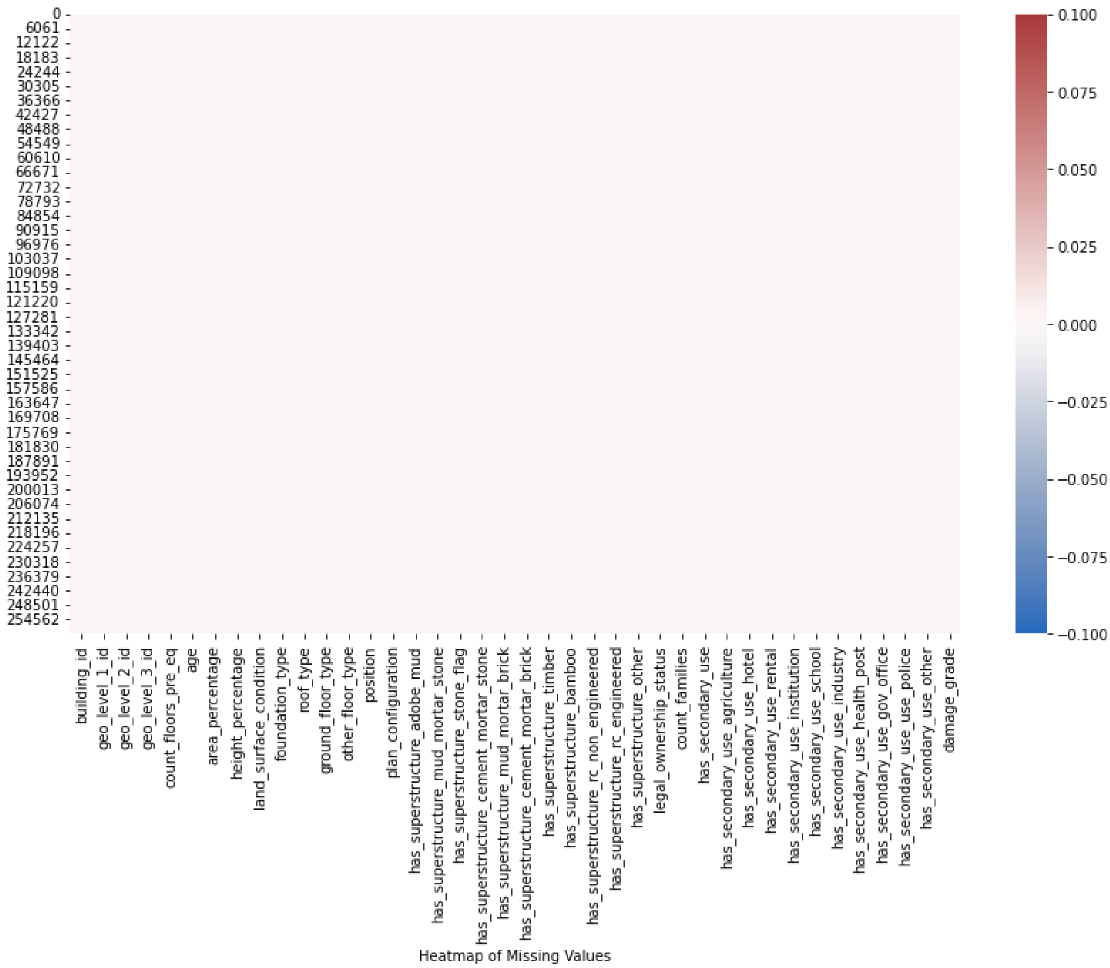
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 260601 entries, 0 to 260600
Data columns (total 40 columns):
 #   Column                                  Non-Null Count   Dtype 
---  ------                                  --------------   ----- 
 0   building_id                             260601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

0.8 Checking for missing values

[16]:
```python
# check for missing values using isnull
join_df.isnull().sum()
```

[16]:
```
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
```

[17]:
```
# plotting all missing values in a heatmap to make the 39 attributes list in␣
↪columns in a heatmap
fig = plt.figure(figsize=(14,8))
sns.heatmap(join_df.isnull(), cbar=True, cmap="vlag")
plt.xlabel("Heatmap of Missing Values")
plt.show()
```



0.9 Checking for Non-NA Values

[18]:
```
# count of non-NA values
join_df.count()
```

[18]:
```
building_id                               260601
geo_level_1_id                            260601
geo_level_2_id                            260601
geo_level_3_id                            260601
count_floors_pre_eq                       260601
```

```
age                                        260601
area_percentage                            260601
height_percentage                          260601
land_surface_condition                     260601
foundation_type                            260601
roof_type                                  260601
ground_floor_type                          260601
other_floor_type                           260601
position                                   260601
plan_configuration                         260601
has_superstructure_adobe_mud               260601
has_superstructure_mud_mortar_stone        260601
has_superstructure_stone_flag              260601
has_superstructure_cement_mortar_stone     260601
has_superstructure_mud_mortar_brick        260601
has_superstructure_cement_mortar_brick     260601
has_superstructure_timber                  260601
has_superstructure_bamboo                  260601
has_superstructure_rc_non_engineered       260601
has_superstructure_rc_engineered           260601
has_superstructure_other                   260601
legal_ownership_status                     260601
count_families                             260601
has_secondary_use                          260601
has_secondary_use_agriculture              260601
has_secondary_use_hotel                    260601
has_secondary_use_rental                   260601
has_secondary_use_institution              260601
has_secondary_use_school                   260601
has_secondary_use_industry                 260601
has_secondary_use_health_post              260601
has_secondary_use_gov_office               260601
has_secondary_use_use_police               260601
has_secondary_use_other                    260601
damage_grade                               260601
dtype: int64
```

5.3 0.10 Understanding the Domain

Attribute Set of the Dataset (Understanding the Domain)

- Checking for Unique Values in the dataset
- Creating a list of attributes under attribute set
- Checking for zero values in numerical measures
- Checking for missing combinations of categories in categorical attributes

0.10.1 Checking for Number of Unique Values in the Dataset

[19]:
```
# printing number of unique values in the attribute domain of the dataset
join_df.nunique()
```

[19]:
```
building_id                               260601
geo_level_1_id                                31
geo_level_2_id                              1414
geo_level_3_id                             11595
count_floors_pre_eq                            9
age                                           42
area_percentage                               84
height_percentage                             27
land_surface_condition                         3
foundation_type                                5
roof_type                                      3
ground_floor_type                              5
other_floor_type                               4
position                                       4
plan_configuration                            10
has_superstructure_adobe_mud                   2
has_superstructure_mud_mortar_stone            2
has_superstructure_stone_flag                  2
has_superstructure_cement_mortar_stone         2
has_superstructure_mud_mortar_brick            2
has_superstructure_cement_mortar_brick         2
has_superstructure_timber                      2
has_superstructure_bamboo                      2
has_superstructure_rc_non_engineered           2
has_superstructure_rc_engineered               2
has_superstructure_other                       2
legal_ownership_status                         4
count_families                                10
has_secondary_use                              2
has_secondary_use_agriculture                  2
has_secondary_use_hotel                        2
has_secondary_use_rental                       2
has_secondary_use_institution                  2
has_secondary_use_school                       2
has_secondary_use_industry                     2
has_secondary_use_health_post                  2
has_secondary_use_gov_office                   2
has_secondary_use_use_police                   2
has_secondary_use_other                        2
damage_grade                                   3
dtype: int64
```

0.10.2 Creating a list of attributes under the attribute set

- Geographical
- Numerical Measures
- Main Building/Land
- Sub Building/Land
- Superstructure Construction
- Secondary Usage

[20]:
```
# Creating attribute set for geographical attributes
geographical_attributes = ['geo_level_1_id', 'geo_level_2_id', 'geo_level_3_id']
# Creating attribute set for numerical measures
numerical_measures = ['count_floors_pre_eq', 'age', 'area_percentage',␣
 ↪'height_percentage', 'count_families']
# Creating attribute set for main categorical data involving building and land␣
 ↪characteristics
main_building_land_attributes = ['ground_floor_type', 'other_floor_type',␣
 ↪'legal_ownership_status', 'plan_configuration']
# Creating attribute set for sub categorical data involving building and land␣
 ↪characteristics
sub_building_land_attributes = ['land_surface_condition', 'foundation_type',␣
 ↪'roof_type', 'position']
# Creating attribute set for superstructure construction attributes
superstructure_attributes =␣
 ↪['has_superstructure_adobe_mud','has_superstructure_bamboo','has_superstructure_cement_mort
# Creating attribute set for secondary usage attributes
secondary_usage_attributes = ['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']
```

0.10.3 Checking for Unique Values in the Dataset

[21]:
```
# printing unique values of categorical variables/attributes
for attr in (main_building_land_attributes + sub_building_land_attributes):
    print("Unique Attributes for: ", attr)
    print(join_df[attr].unique())
```

```
Unique Attributes for:  ground_floor_type
['f' 'x' 'v' 'z' 'm']
Unique Attributes for:  other_floor_type
['q' 'x' 'j' 's']
Unique Attributes for:  legal_ownership_status
['v' 'a' 'r' 'w']
Unique Attributes for:  plan_configuration
['d' 'u' 's' 'q' 'm' 'c' 'a' 'n' 'f' 'o']
Unique Attributes for:  land_surface_condition
['t' 'o' 'n']
```

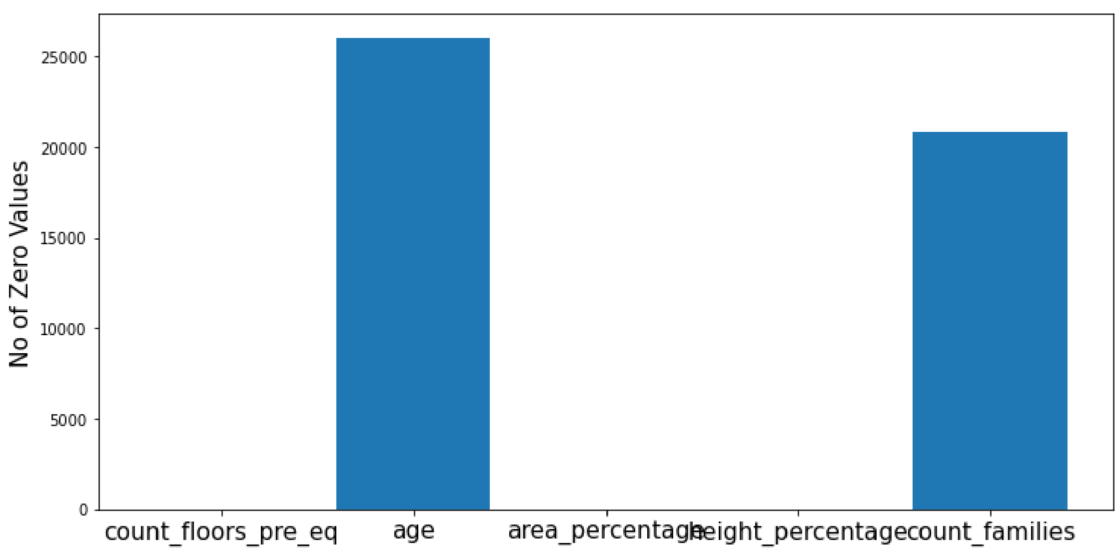
```
Unique Attributes for:  foundation_type
['r' 'w' 'i' 'u' 'h']
Unique Attributes for:  roof_type
['n' 'q' 'x']
Unique Attributes for:  position
['t' 's' 'j' 'o']
```

0.10.4 Checking for zero values in numerical measures

[22]:
```
# checking zero values on numerical measures only,
# zero values on binary and geographical attributes have direct semantic␣
↪meanings
zero_values = (join_df.loc[:, numerical_measures] == 0).astype('int32').
↪sum(axis=0)
zero_values
```

[22]:
```
count_floors_pre_eq        0
age                    26041
area_percentage            0
height_percentage          0
count_families         20862
dtype: int64
```

[23]:
```
# plotting the count of zero values in the numerical measures
fig = plt.figure(figsize=(12,6))
plt.bar(zero_values.index, zero_values.values)
plt.xticks(zero_values.index, fontsize=15.5)
plt.ylabel("No of Zero Values", fontsize=15.5)
plt.show()
```

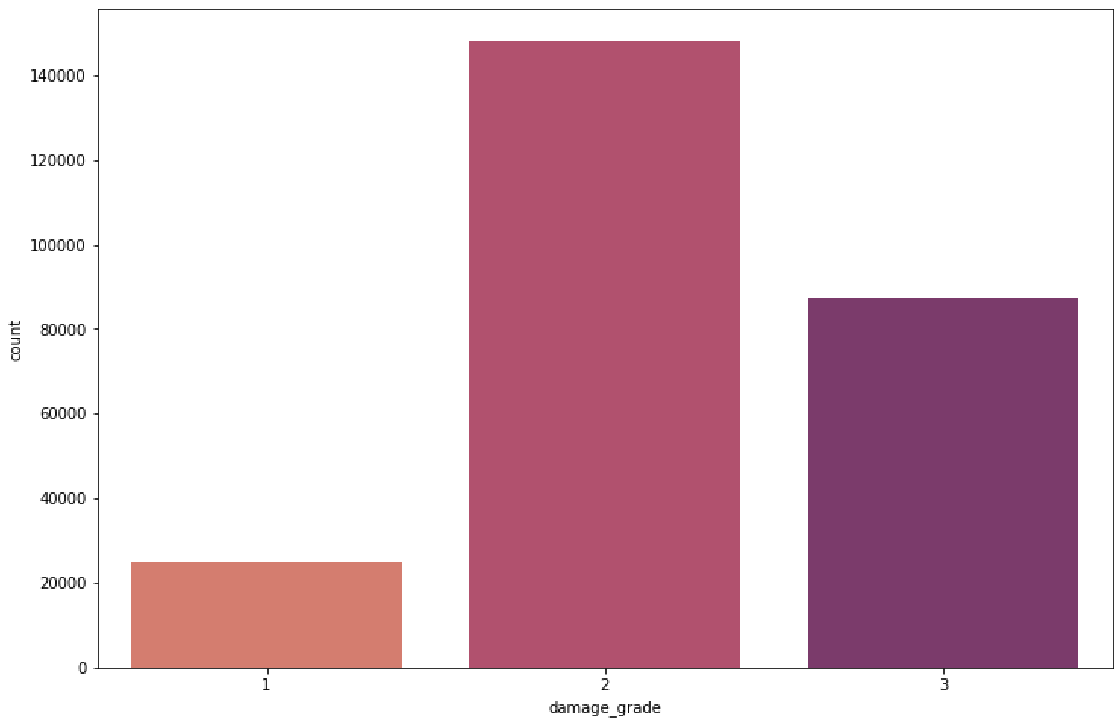


Assigning correct dtypes for dataset

[24]:
```
# assigning category dtype to categorical variables
join_df = join_df.astype({x: 'category' for x in main_building_land_attributes})
join_df = join_df.astype({x: 'category' for x in sub_building_land_attributes})
# assigning category dtype for target variable
join_df = join_df.astype({'damage_grade': 'category'})
# assigning int32 for numerical measures
join_df = join_df.astype({x: 'int32' for x in numerical_measures})
# assigning int32 for geo level attributes
join_df = join_df.astype({x: 'int32' for x in geographical_attributes})
```

Distribution of Damage Grade (A Count Plot)

[25]:
```
# Distribution of damage across damage levels.
plt.figure(figsize=(12,8))
ax = sns.countplot(x="damage_grade", data=join_df, palette="flare")
```



Checking for missing combinations in categorical attributes

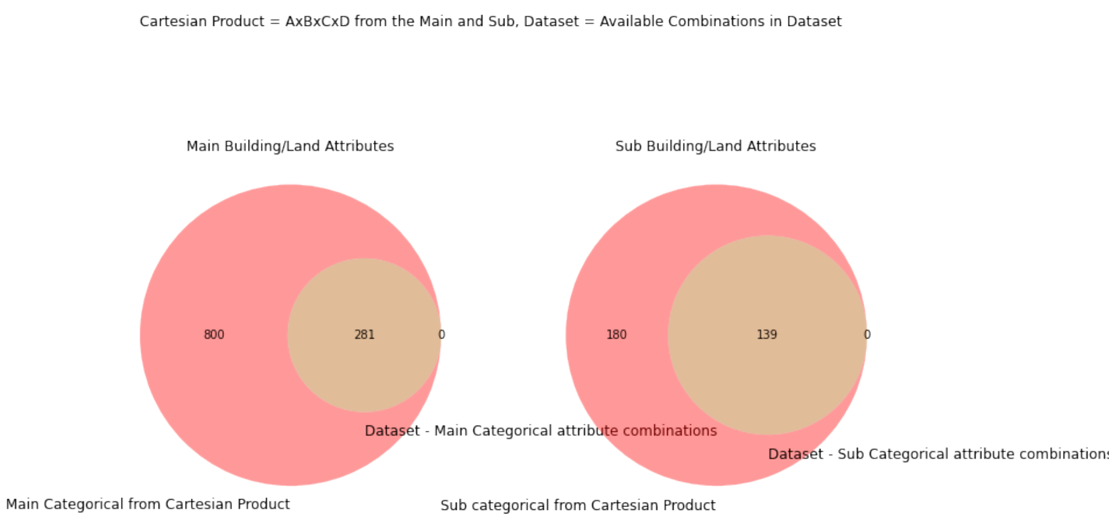
[26]:
```
# zero values does not make any sense with categorical attributes
# the combinations of categorical attributes may have missing entries as␣
↪compared with combinations of all such attributes
# this is relevant to the understanding of the domain
```

```
# checking for missing combinations for the first set of building/land␣
↪attributes (categorical attributes)
# adding 'building_id' for groupby remaining column
main_df = join_df.loc[:, main_building_land_attributes + ['building_id']].
↪groupby(main_building_land_attributes).count()
main_df.drop(labels=main_df[main_df['building_id'] == 0].index, inplace=True)
# listing all possible cartesian product of main building/land attributes␣
↪(categorical attributes)
main_categorical_attributes_table = np.array(list(itertools.product(
    *[join_df[attr].unique().tolist() for attr in␣
↪main_building_land_attributes]))
)
# listing all available combinations of main building/land attributes␣
↪(categorical attributes)
# get_level_values will return MultiIndex Values according to the levels 1,2,3,4
main_available_attributes_table = np.array([main_df.index.get_level_values(i).
↪values.tolist() for i in range(len(main_building_land_attributes))]).T

# checking for missing combinations for the second set of building/land␣
↪attributes (categorical attributes)
sub_df = join_df.loc[:, sub_building_land_attributes + ['building_id']].
↪groupby(sub_building_land_attributes).count()
sub_df.drop(labels=sub_df[sub_df['building_id'] == 0].index, inplace=True)
# listing all possible cartesian product of sub building/land attributes␣
↪(categorical attributes)
sub_categorical_attributes_table = np.array(list(itertools.product(
    *[join_df[attr].unique().tolist() for attr in␣
↪sub_building_land_attributes]))
)
# listing all available combinations of sub building/land attributes␣
↪(categorical attributes)
sub_available_attributes_table = np.array([sub_df.index.get_level_values(i).
↪values.tolist() for i in range(len(sub_building_land_attributes))]).T
```

[27]:
```
# plotting the missing attributes ysing matplotlib-venn
fig, (ax1,ax2) = plt.subplots(1, 2, figsize=(12,8))
# create a venn diagram showing actual dataset groups of main categorical␣
↪attributes as a subset of the Cartesian Product
venn2(subsets=(len(main_categorical_attributes_table), 0,␣
↪len(main_available_attributes_table)), set_labels=['Main Categorical from␣
↪Cartesian Product', 'Dataset - Main Categorical attribute combinations'],
      ax=ax1)
# create a venn diagram showing actual dataset groups of sub categorical␣
↪attributes as a subset of the Cartesian Product
```

```
venn2(subsets=(len(sub_categorical_attributes_table), 0,␣
↪len(sub_available_attributes_table)), set_labels=['Sub categorical from␣
↪Cartesian Product', 'Dataset - Sub Categorical attribute combinations'],
      ax=ax2)
# setting the titles for both venn diagrams
ax1.set_title("Main Building/Land Attributes")
ax2.set_title("Sub Building/Land Attributes")
fig.suptitle("Cartesian Product = AxBxCxD from the Main and Sub, Dataset =␣
↪Available Combinations in Dataset")
fig.show()
```

```
C:\Users\burse\AppData\Local\Temp/ipykernel_24064/2630837216.py:13: UserWarning:

Matplotlib is currently using module://matplotlib_inline.backend_inline, which
is a non-GUI backend, so cannot show the figure.
```



Observations about Main Categorical

- There are 281 Available Attribute combinations in the dataset for Main categorical
- There are 519 Combinations that are not registered in the dataset
- Main Categorical (Building/Land Attributes) Include Ground Floor Type, Other Floor Type, Legal Ownership Status, Plan Configuration

Observations about Sub Categorical

- There are 139 Available Attribute combinations in the dataset for Sub Categorical
- There are only 41 Combinations that are not registered in the dataset
- Sub Categorical (Building/Land Attributes) include Land Surface condition, Foundation Type, Roof Type, Position

5.4 0.11 Numerical Measures (Understanding the Domain)

5.4.1 Transformation Candidates Overview

0.11.1. Demean Data (First step of Standardisation)

[28]:
```
# Make zero mean for the dataframe
def demean_data(X_df):
    return (X_df - X_df.mean(axis=0))
```

0.11.2. MinMax Scaling of Data

[29]:
```
# Min Max Scaling
def min_max_scale(X_s, start=0, end=1, loc=0):
    return (X_s - X_s.min()) / (X_s.max() - X_s.min()) * (end-start) + loc
```

0.11.3. Normalization of Data

[30]:
```
# Normalizing the data
def normalization(X_df):
    return (X_df - X_df.mean()) / X_df.std()
```

0.11.4. Log Scaling Transformation

[31]:
```
# Log scaling the data
def log_scaling(X_s):
    return np.log(X_s)
```

Histogram plot of Numerical Measures

- Count of Families
- Age
- Area Percentage
- Height Percentage
- Count of Floors

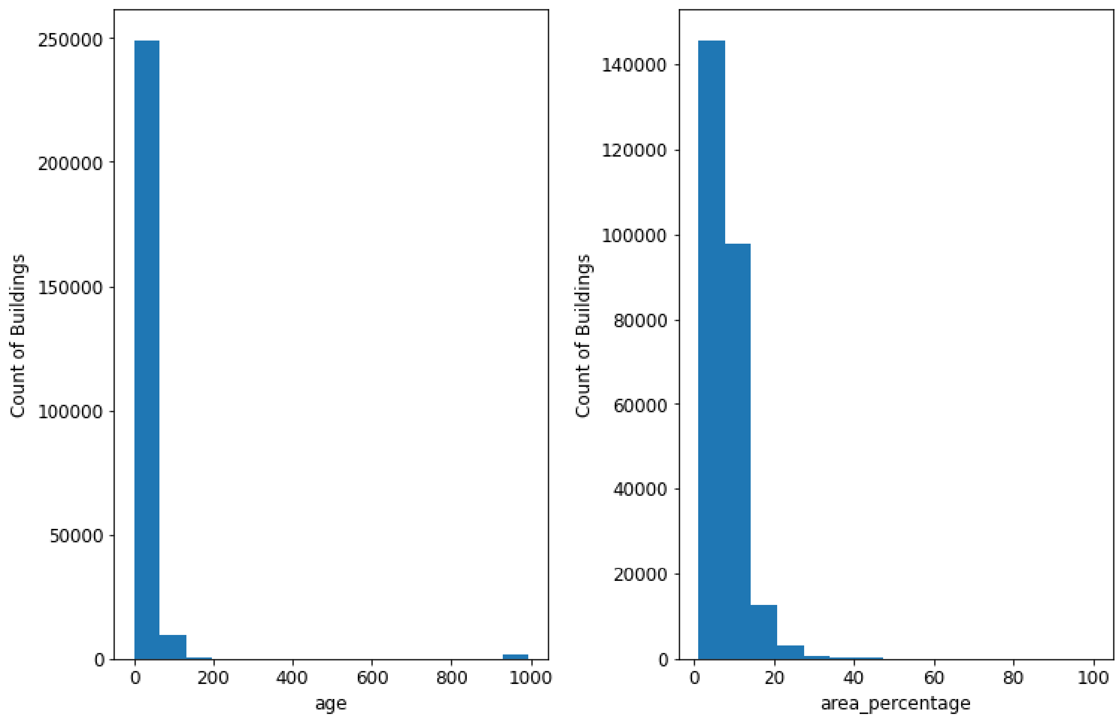
Transformation Candidates

- Log Scaling
- Normalization

5.5 0.12 Histogram plot of Numerical Measures

[32]:
```
# copy the dataframe in order to preserve the original format after␣
↪transformation
numerical_df = join_df.copy()
# selecting only numerical measures
numerical_df = numerical_df.loc[:, numerical_measures]
# plotting the histograms
plt.rcParams.update({'font.size': 12})
```

```
fig, ax = plt.subplots(1,2, figsize=(12,8))
numerical_df.loc[:, ['age']].hist(bins=15, figsize = (10,5), grid=False,␣
↪ax=ax[0])
numerical_df.loc[:, ['area_percentage']].hist(bins=15, figsize = (10,5),␣
↪grid=False, ax=ax[1])
ax[0].set(xlabel="age", ylabel="Count of Buildings")
ax[1].set(xlabel="area_percentage", ylabel="Count of Buildings")
ax[0].set_title("")
ax[1].set_title("")
plt.show()
```



[33]:
```
age_second_skewness_coefficient = 3 * (numerical_df.age.mean() - numerical_df.
↪age.median()) / numerical_df.age.std()
age_first_skewness_coefficient = (numerical_df.age.mean() - numerical_df.age.
↪mode()) / numerical_df.age.std()

area_percentage_second_skewness_coefficient = 3 * (numerical_df.area_percentage.
↪mean() - numerical_df.area_percentage.median()) / numerical_df.
↪area_percentage.std()
area_percentage_first_skewness_coefficient = (numerical_df.area_percentage.
↪mean() - numerical_df.area_percentage.mode()) / numerical_df.area_percentage.
↪std()
```

```
height_percentage_second_skewness_coefficient = 3 * (numerical_df.
↪height_percentage.mean() - numerical_df.height_percentage.median()) /␣
↪numerical_df.height_percentage.std()
height_percentage_first_skewness_coefficient = (numerical_df.height_percentage.
↪mean() - numerical_df.height_percentage.mode()) / numerical_df.
↪height_percentage.std()

skewness_coeff_table = pd.DataFrame([
    [age_second_skewness_coefficient, age_first_skewness_coefficient.iloc[0]],
    [area_percentage_second_skewness_coefficient,␣
↪area_percentage_first_skewness_coefficient.iloc[0]],
    [height_percentage_second_skewness_coefficient,␣
↪height_percentage_first_skewness_coefficient.iloc[0]],
], columns=['Second Skewness Coefficient', 'First Skewness Coefficient'])

skewness_coeff_table.set_index(pd.Index(['age', 'area_percentage',␣
↪'height_percentage']), inplace=True)
skewness_coeff_table.style.background_gradient(cmap='coolwarm')
```

[33]: <pandas.io.formats.style.Styler at 0x1f40a168730>

Observations

- Most of the numerical features show some values whose Frequency is very low as compared to the Maximum Frequency Bin
- This is a problem of visualization with Actual Values

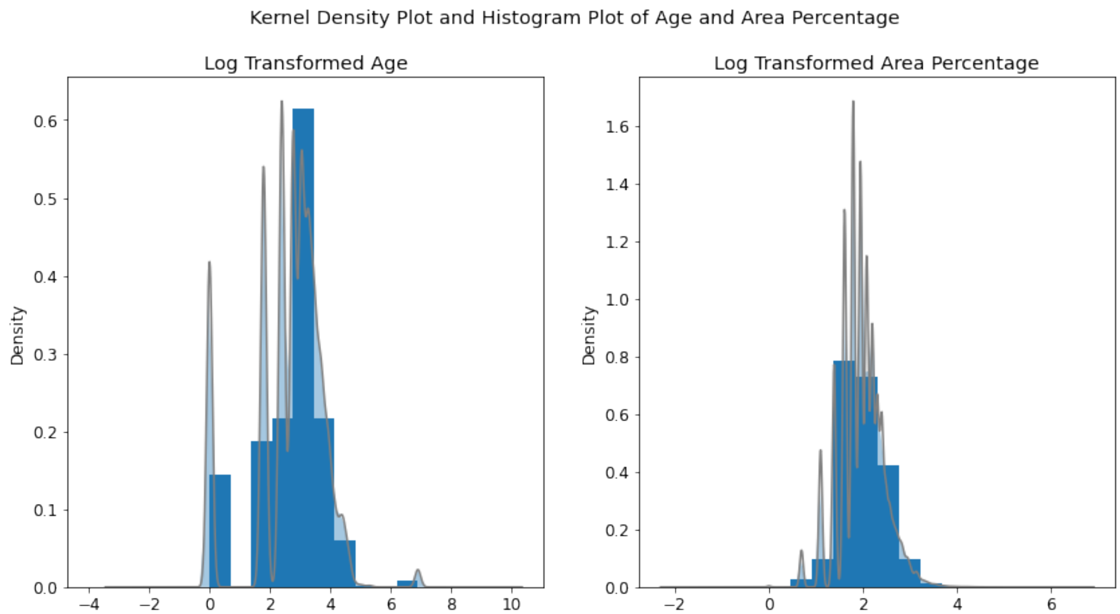
Findings

- Normalized Value Counts could solve this problem and allow visualization on a better scale for comparison

5.5.1 Histogram plot of Numerical Measures after Log Scaling

[34]:
```
def plot_kde_histogram_numerical():
    # copying the dataframe to preserve the original format after transformation
    numerical_df = join_df.copy()
    numerical_df = numerical_df.loc[:, numerical_measures]
    # making 0 years 1 year value for log scaling compatibility (shifting by 1␣
↪year)
    numerical_df.age += 1.0
    # Gaussian KDE is the transformation that is used by kde plot of Pandas
    kde = gaussian_kde(np.log(numerical_df.age))
    # specify the span of the kde plot
    xspan = np.linspace(-4, 10, 100)
    # calculate the kde distribution in order to fill between
    pdf = kde(xspan)
```

```
    # plot histogram, density = True is used to equalise the kde plot
    np.log(numerical_df.age).plot(kind='hist', figsize=(14,7), rot=0,␣
 ↪label='Log Transformed Age', title='Log Transformed Age', density=True,␣
 ↪ax=ax1);
    # plot kernel density estimation plot
    np.log(numerical_df.age).plot(kind='kde', figsize=(14,7), rot=0, label='Log␣
 ↪Transformed Age', title='Log Transformed Age', color='gray', ax=ax1);
    # fill between
    ax1.fill_between(xspan, 0, pdf, alpha=0.4)

    # subplot of area percentage with log transformation
    kde = gaussian_kde(np.log(numerical_df.area_percentage))
    # span of the kde plot
    xspan = np.linspace(-2, 6, 100)
    # calculate the pdf of kde
    pdf = kde(xspan)
    # plot histogram
    np.log(numerical_df.area_percentage).plot(kind='hist', figsize=(14,7),␣
 ↪rot=0, label='Log Transformed Area Percentage', title='Log Transformed Area␣
 ↪Percentage', density=True, ax=ax2);
    # plot kde
    np.log(numerical_df.area_percentage).plot(kind='kde', figsize=(14,7),␣
 ↪rot=0, label='Log Transformed Area Percentage', title='Log Transformed Area␣
 ↪Percentage', color='gray',ax=ax2);
    # fill between
    ax2.fill_between(xspan, 0, pdf, alpha=0.4)
    # set super title
    fig.suptitle("Kernel Density Plot and Histogram Plot of Age and Area␣
 ↪Percentage")

# subplot of age with value counts normalized
fig, (ax1,ax2) = plt.subplots(1,2)
plot_kde_histogram_numerical()
```



Observations

- Now all the values are visible and the bars have increased their sizes as compared to showing less value for non-normalized values
- Possibly, Not every attribute require log scaling

Recommendations

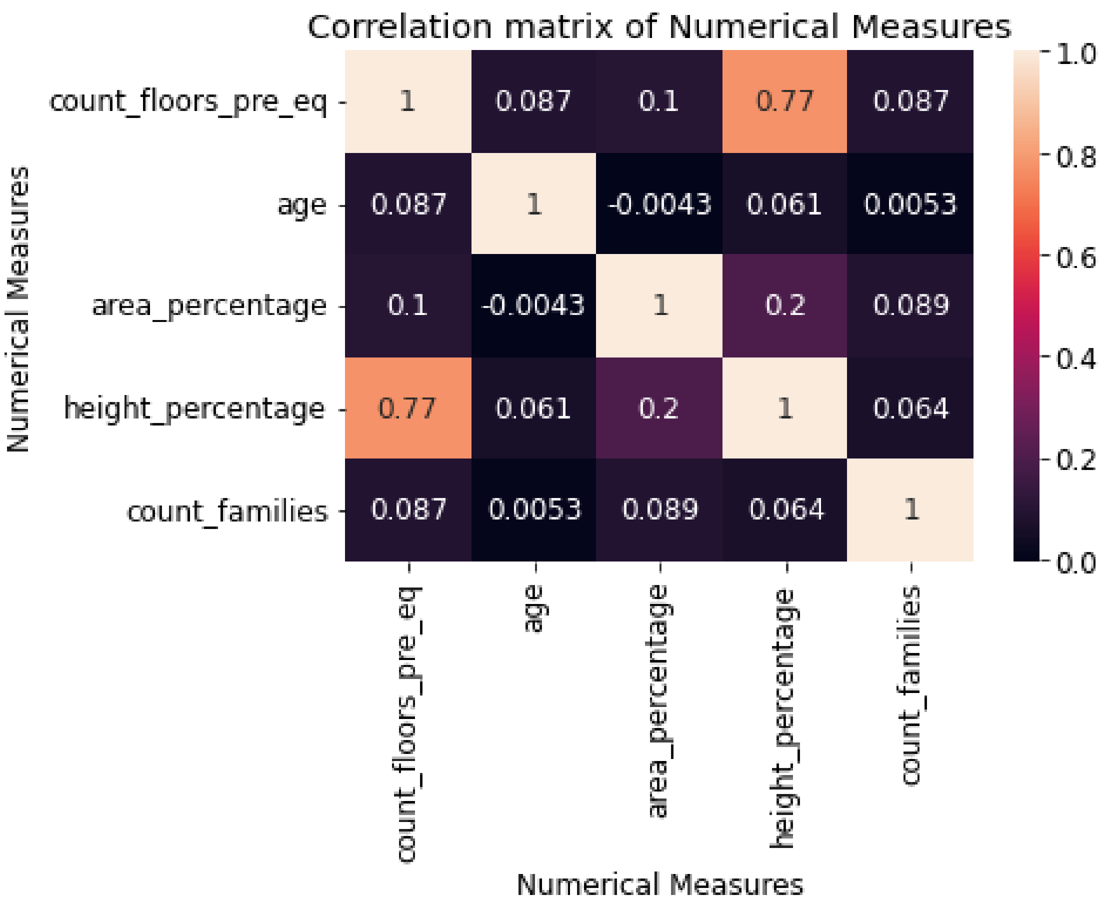
- However, The attributes for which the normalization seems to be essential are:
  - Age
  - Area Percentage
  - Height Percentage
- Count of Floors and Count of Families may remain the same
- This is because their values show significant change in terms of the distribution, the distribution has become more cleaner and sharper
- Such normalization will reduce the variance thereby enabling a machine learning algorithm to learn better

Problem Statement

- In order for the government to implement governance plans the dataset must have reported correlated property for the numerical attributes
- Such correlation will help in understanding the relationship between two attributes such as: Count of Floors and Average Height Percentage of a Building

5.6 0.13 Correlation of Numerical Features only

[35]:
```
# copy from original dataframe
numerical_df = join_df.copy()
numerical_df = numerical_df.loc[:, numerical_measures]
# find correlation of the matrix
corr = numerical_df.corr()
# set the background gradient of correlation matrix such that higher values and␣
 ↪lower values are distinguishable
hm = sns.heatmap(corr, annot = True)
hm.set(xlabel='Numerical Measures', ylabel='Numerical Measures', title =␣
 ↪"Correlation matrix of Numerical Measures")
plt.show()
```



Observations

- The Average height Percentage of a building and its count of floors before earthquake are highly correlated
- The Age and Average Area Percentage of a building are slightly negatively correlated implying

when building is old, the area becomes smaller generally

Recommendations

- These insights generate good results on the numerical attributes
- A scatter plot of average height percentage and count of floors can visualize the building and helps in investigating one property if the another one is known

6 0.14 Descriptive Statistics

- Generating Summary Statistics
- Help Answer Research Questions
- Outliers and Boxplots

Generating summary statistics

- Show summary statistics
- Obtains Rank for Each Building data

[36]:
```
# generate summary statistics
join_df.describe().T
```

[36]:
```
                                          count           mean  \
building_id                            260601.0  525675.482773
geo_level_1_id                         260601.0      13.900353
geo_level_2_id                         260601.0     701.074685
geo_level_3_id                         260601.0    6257.876148
count_floors_pre_eq                    260601.0       2.129723
age                                    260601.0      26.535029
area_percentage                        260601.0       8.018051
height_percentage                      260601.0       5.434365
has_superstructure_adobe_mud           260601.0       0.088645
has_superstructure_mud_mortar_stone    260601.0       0.761935
has_superstructure_stone_flag          260601.0       0.034332
has_superstructure_cement_mortar_stone 260601.0       0.018235
has_superstructure_mud_mortar_brick    260601.0       0.068154
has_superstructure_cement_mortar_brick 260601.0       0.075268
has_superstructure_timber              260601.0       0.254988
has_superstructure_bamboo              260601.0       0.085011
has_superstructure_rc_non_engineered   260601.0       0.042590
has_superstructure_rc_engineered       260601.0       0.015859
has_superstructure_other               260601.0       0.014985
count_families                         260601.0       0.983949
has_secondary_use                      260601.0       0.111880
has_secondary_use_agriculture          260601.0       0.064378
has_secondary_use_hotel                260601.0       0.033626
has_secondary_use_rental               260601.0       0.008101
has_secondary_use_institution          260601.0       0.000940
```

has_secondary_use_school	260601.0	0.000361
has_secondary_use_industry	260601.0	0.001071
has_secondary_use_health_post	260601.0	0.000188
has_secondary_use_gov_office	260601.0	0.000146
has_secondary_use_use_police	260601.0	0.000088
has_secondary_use_other	260601.0	0.005119

	std	min	25%	\
building_id	304544.999032	4.0	261190.0	
geo_level_1_id	8.033617	0.0	7.0	
geo_level_2_id	412.710734	0.0	350.0	
geo_level_3_id	3646.369645	0.0	3073.0	
count_floors_pre_eq	0.727665	1.0	2.0	
age	73.565937	0.0	10.0	
area_percentage	4.392231	1.0	5.0	
height_percentage	1.918418	2.0	4.0	
has_superstructure_adobe_mud	0.284231	0.0	0.0	
has_superstructure_mud_mortar_stone	0.425900	0.0	1.0	
has_superstructure_stone_flag	0.182081	0.0	0.0	
has_superstructure_cement_mortar_stone	0.133800	0.0	0.0	
has_superstructure_mud_mortar_brick	0.252010	0.0	0.0	
has_superstructure_cement_mortar_brick	0.263824	0.0	0.0	
has_superstructure_timber	0.435855	0.0	0.0	
has_superstructure_bamboo	0.278899	0.0	0.0	
has_superstructure_rc_non_engineered	0.201931	0.0	0.0	
has_superstructure_rc_engineered	0.124932	0.0	0.0	
has_superstructure_other	0.121491	0.0	0.0	
count_families	0.418389	0.0	1.0	
has_secondary_use	0.315219	0.0	0.0	
has_secondary_use_agriculture	0.245426	0.0	0.0	
has_secondary_use_hotel	0.180265	0.0	0.0	
has_secondary_use_rental	0.089638	0.0	0.0	
has_secondary_use_institution	0.030647	0.0	0.0	
has_secondary_use_school	0.018989	0.0	0.0	
has_secondary_use_industry	0.032703	0.0	0.0	
has_secondary_use_health_post	0.013711	0.0	0.0	
has_secondary_use_gov_office	0.012075	0.0	0.0	
has_secondary_use_use_police	0.009394	0.0	0.0	
has_secondary_use_other	0.071364	0.0	0.0	

	50%	75%	max
building_id	525757.0	789762.0	1052934.0
geo_level_1_id	12.0	21.0	30.0
geo_level_2_id	702.0	1050.0	1427.0
geo_level_3_id	6270.0	9412.0	12567.0
count_floors_pre_eq	2.0	2.0	9.0
age	15.0	30.0	995.0

```
area_percentage                              7.0      9.0    100.0
height_percentage                            5.0      6.0     32.0
has_superstructure_adobe_mud                 0.0      0.0      1.0
has_superstructure_mud_mortar_stone          1.0      1.0      1.0
has_superstructure_stone_flag                0.0      0.0      1.0
has_superstructure_cement_mortar_stone       0.0      0.0      1.0
has_superstructure_mud_mortar_brick          0.0      0.0      1.0
has_superstructure_cement_mortar_brick       0.0      0.0      1.0
has_superstructure_timber                    0.0      1.0      1.0
has_superstructure_bamboo                    0.0      0.0      1.0
has_superstructure_rc_non_engineered         0.0      0.0      1.0
has_superstructure_rc_engineered             0.0      0.0      1.0
has_superstructure_other                     0.0      0.0      1.0
count_families                               1.0      1.0      9.0
has_secondary_use                            0.0      0.0      1.0
has_secondary_use_agriculture                0.0      0.0      1.0
has_secondary_use_hotel                      0.0      0.0      1.0
has_secondary_use_rental                     0.0      0.0      1.0
has_secondary_use_institution                0.0      0.0      1.0
has_secondary_use_school                     0.0      0.0      1.0
has_secondary_use_industry                   0.0      0.0      1.0
has_secondary_use_health_post                0.0      0.0      1.0
has_secondary_use_gov_office                 0.0      0.0      1.0
has_secondary_use_use_police                 0.0      0.0      1.0
has_secondary_use_other                      0.0      0.0      1.0
```

[37]:

```
# show the rank of individual columns in the dataset that represent their
↪values (Their ordering) from a random data sample
join_df.rank()
```

[37]:

```
        building_id  geo_level_1_id  geo_level_2_id  geo_level_3_id  \
0          198724.0         44632.0         91563.5        253118.0
1            7211.0         85356.5        167275.0         59875.5
2           23775.0        200356.0         68039.5        186051.5
3          146213.0        210926.5         77937.0        221457.0
4           50438.0        125043.5         23807.5         31856.5
…               …              …              …              …
260596     170651.0        219295.5        242624.5         34647.5
260597     165885.0        161228.0        132699.0         43863.0
260598     149085.0        161228.0         11084.0        169119.5
260599      37872.0        233415.0          7514.5         39651.5
260600     185250.0        200356.0          1048.0        188184.0

        count_floors_pre_eq       age  area_percentage  height_percentage  \
0                  118753.0  200206.5          88075.0           112282.0
1                  118753.0   79186.5         160056.0           215748.0
2                  118753.0   79186.5          50706.5           112282.0
```

```
3                 118753.0   79186.5         88075.0          112282.0
4                 224873.0  200206.5        160056.0          250070.5
…                        …         …               …                 …
260596             20221.0  243474.0         88075.0           22284.0
260597            118753.0   13021.0         88075.0          112282.0
260598            224873.0  243474.0         88075.0          215748.0
260599            118753.0   79186.5        241474.5          174777.0
260600            224873.0   79186.5        127457.5          174777.0

        land_surface_condition  foundation_type  …  has_secondary_use_hotel  \
0                     152223.0         121625.5  …                 125919.5
1                      39686.5         121625.5  …                 125919.5
2                     152223.0         121625.5  …                 125919.5
3                     152223.0         121625.5  …                 125919.5
4                     152223.0         121625.5  …                 125919.5
…                          …              …  …                      …
260596                 17764.5         121625.5  …                 125919.5
260597                152223.0         121625.5  …                 125919.5
260598                152223.0         121625.5  …                 125919.5
260599                152223.0         121625.5  …                 125919.5
260600                 17764.5         121625.5  …                 125919.5

        has_secondary_use_rental  has_secondary_use_institution  \
0                       129245.5                       130178.5
1                       129245.5                       130178.5
2                       129245.5                       130178.5
3                       129245.5                       130178.5
4                       129245.5                       130178.5
…                            …                            …
260596                  129245.5                       130178.5
260597                  129245.5                       130178.5
260598                  129245.5                       130178.5
260599                  129245.5                       130178.5
260600                  129245.5                       130178.5

        has_secondary_use_school  has_secondary_use_industry  \
0                       130254.0                    130161.5
1                       130254.0                    130161.5
2                       130254.0                    130161.5
3                       130254.0                    130161.5
4                       130254.0                    130161.5
…                            …                         …
260596                  130254.0                    130161.5
260597                  130254.0                    130161.5
260598                  130254.0                    130161.5
260599                  130254.0                    130161.5
260600                  130254.0                    130161.5
```

```
        has_secondary_use_health_post  has_secondary_use_gov_office  \
0                            130276.5                      130282.0
1                            130276.5                      130282.0
2                            130276.5                      130282.0
3                            130276.5                      130282.0
4                            130276.5                      130282.0
…                                   …                             …
260596                       130276.5                      130282.0
260597                       130276.5                      130282.0
260598                       130276.5                      130282.0
260599                       130276.5                      130282.0
260600                       130276.5                      130282.0

        has_secondary_use_use_police  has_secondary_use_other  damage_grade
0                           130289.5                 129634.0      216992.5
1                           130289.5                 129634.0       99254.0
2                           130289.5                 129634.0      216992.5
3                           130289.5                 129634.0       99254.0
4                           130289.5                 129634.0      216992.5
…                                  …                        …             …
260596                      130289.5                 129634.0       99254.0
260597                      130289.5                 129634.0      216992.5
260598                      130289.5                 129634.0      216992.5
260599                      130289.5                 129634.0       99254.0
260600                      130289.5                 129634.0      216992.5

[260601 rows x 40 columns]
```

6.1 0.15 QUALITY OF MEASUREMENTS

Scatter and Line Plot of Count of Floors vs Height Percentage

- Relationship between Count of Floors and Height Percentage

Height Percentage may be measured by **using LIDAR data** and count of floors by a known method by the government of Nepal. There may be **quality differences** observed in the measurements. The plot provides the relationship between Height Percentage and Count of Floors.

The High Variance region may denote the tall **Tower-like Buildings** that had been damaged due to Earthquake, may have been counted as 2 to 8 floors in the dataset.

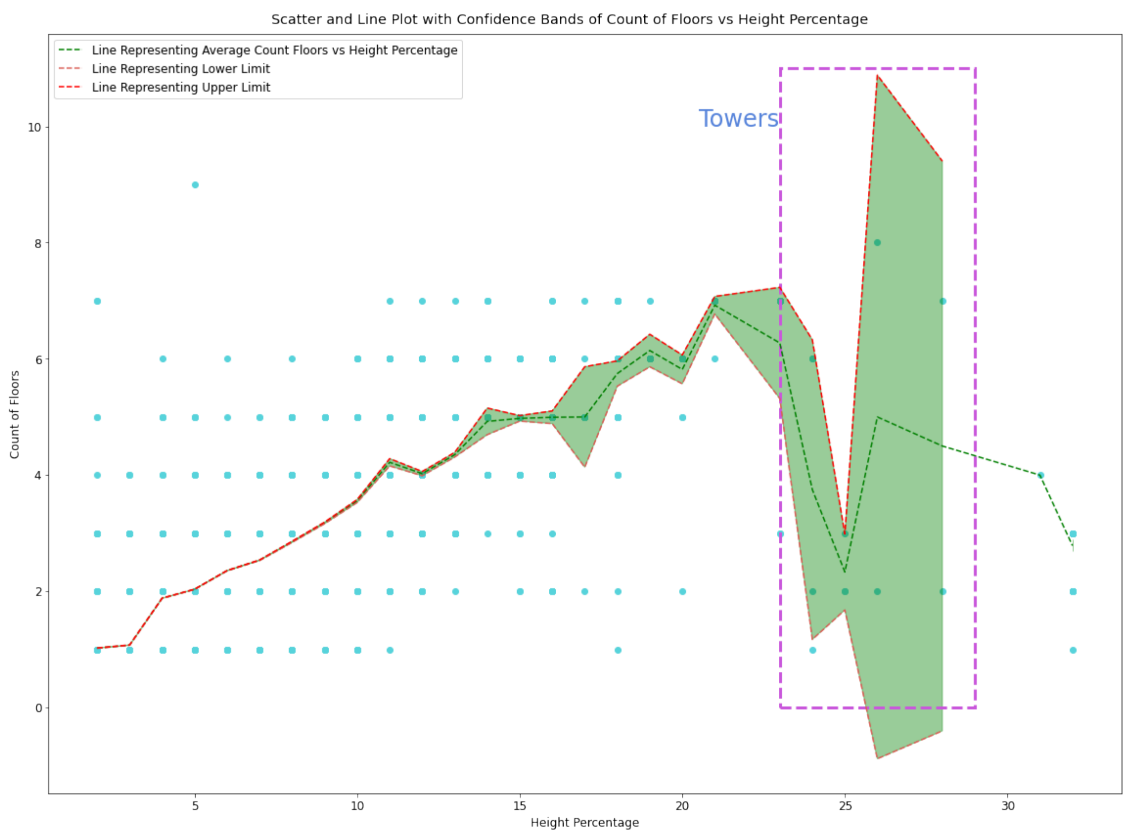
- The Pearson R correlation coefficient between Height Percentage and Count of Floors is **0.772734**.

[38]:
```
fig, ax = plt.subplots(1, 1, figsize=(16,12))
colors = sns.color_palette("hls", 10)

def plot_count_floors_vs_height_percentage(join_df):
```

```
    '''
    Plots Count of Floors and Height Percentage in a scatter plot
    @param join_df: Main DataFrame
    @return:
    '''
    # scatter plot between Count of floors and Average height Percentage
    ax.scatter(join_df.height_percentage, join_df.count_floors_pre_eq,␣
 ↪color=colors[5])

    # set title, xlabel and ylabel
    ax.set(xlabel="Height Percentage", ylabel="Count of Floors")
    fig.suptitle("Scatter and Line Plot with Confidence Bands of Count of␣
 ↪Floors vs Height Percentage")
    # tight layout
    fig.tight_layout(pad=1.0)

def plot_average_line_representing_count_floors_over_height_percentage(join_df):
    '''
    Plots Average Line of Count Floors over Height Percentage
    @param join_df: Main DataFrame
    @return:
    '''
    # aggregation of count floors with mean and standard error
    g = join_df.groupby('height_percentage')['count_floors_pre_eq'].
 ↪agg(['mean', 'sem'])
    # plot the average line
    ax.plot(g.index, g['mean'], color='green', label='Line Representing Average␣
 ↪Count Floors vs Height Percentage', ls='dashed')
    # plot the lower limit of the average line
    ax.plot(g.index, g['mean']-1.96*g['sem'], color=colors[0], label='Line␣
 ↪Representing Lower Limit', ls='dashed')
    # plot the upper limit of the average line
    ax.plot(g.index, g['mean']+1.96*g['sem'], color='red', label='Line␣
 ↪Representing Upper Limit', ls='dashed')
    # fill the confidence intervals
    ax.fill_between(g.index, g['mean']+1.96*g['sem'], g['mean']-1.96*g['sem'],␣
 ↪edgecolor='g', facecolor='g', alpha=0.4)

    # display the legend
    ax.legend()

def plot_high_variance_area():
    '''
    Plots the High variance Area detected by the fluctuations in Area␣
 ↪Percentage values
    @return:
```

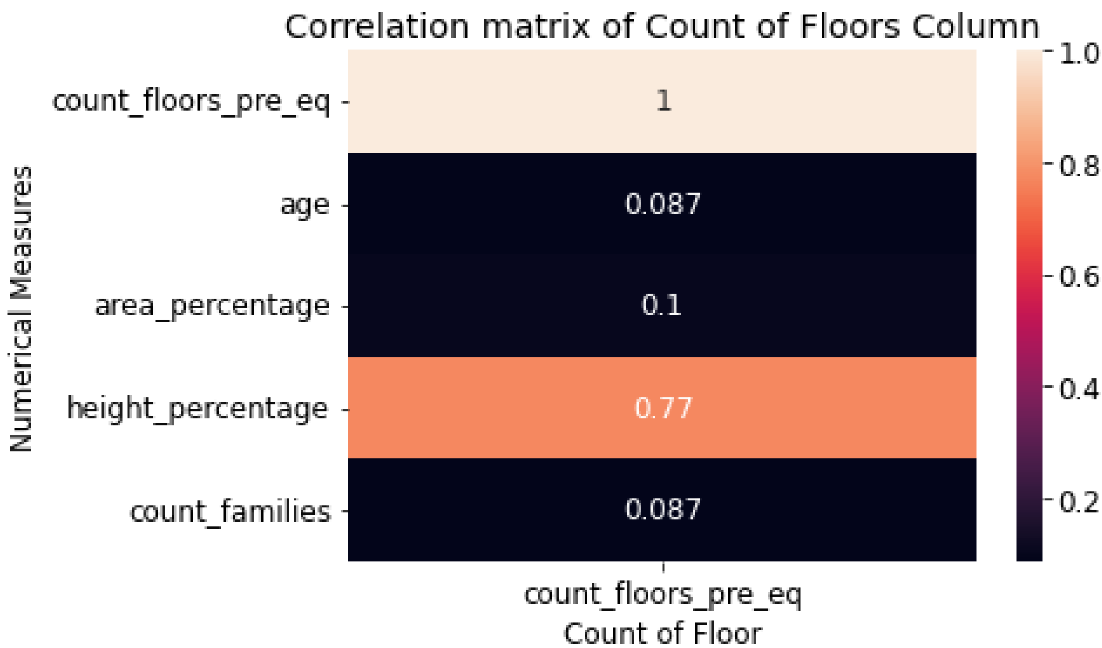
```
    '''
    # Loop over data points; create box from errors at each point
    high_variance_box = Rectangle((23, 0), 6, 11, fill=False, ls='dashed',␣
↪lw=3, color=colors[8])
    ax.add_patch(high_variance_box)
    ax.text(20.5, 10, "Towers", fontsize=24, color=colors[6])

plot_count_floors_vs_height_percentage(join_df)
plot_average_line_representing_count_floors_over_height_percentage(join_df)
plot_high_variance_area()
fig.show()
```

```
C:\Users\burse\AppData\Local\Temp/ipykernel_24064/3067398662.py:52: UserWarning:

Matplotlib is currently using module://matplotlib_inline.backend_inline, which
is a non-GUI backend, so cannot show the figure.
```



Correlation between Height Percentage and Count of Floors

```
[39]: # calculate correlation of numerical measures
corr = join_df.loc[:, numerical_measures].corr()
# evaluate the correlation matrix of 1st column using background gradient
hm = sns.heatmap(corr.iloc[:, [0]], annot = True)
hm.set(xlabel='Count of Floor', ylabel='Numerical Measures', title =␣
 ↪"Correlation matrix of Count of Floors Column")
plt.show()
```

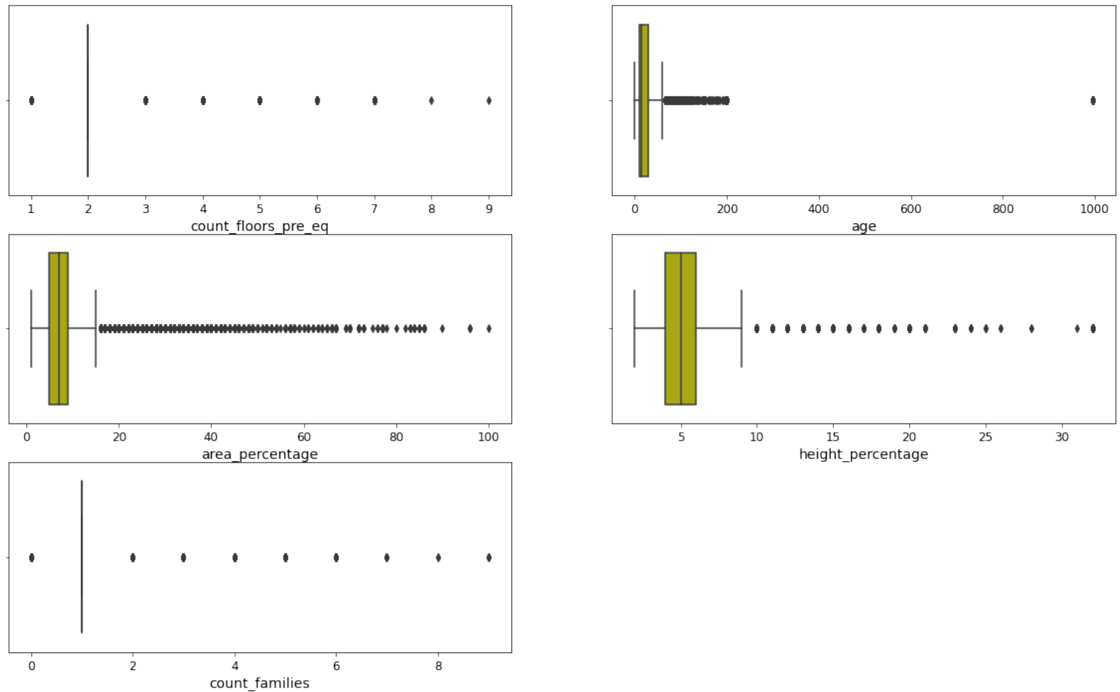


6.1.1 0.16 Box Plot of Numerical Measures

- To find outliers and extreme values
- To determine skewness

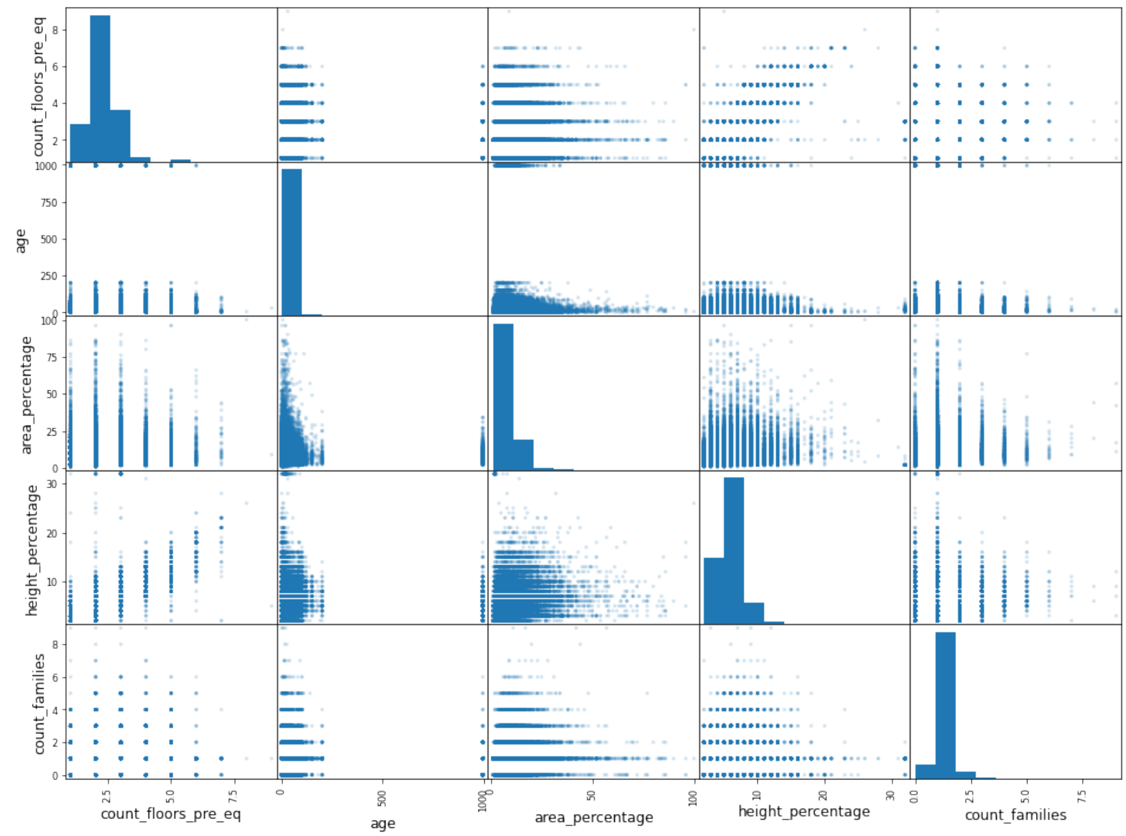
```
[40]: # set a = 1 to increment
a=1
# set figure size
plt.figure(figsize=(20,12))
# iterate through numerical measures
for attr in numerical_measures:
    # create subplots
    plt.subplot(3,2,a)
    # plot boxplot
    ax=sns.boxplot(x=attr, data=join_df, color='y')
    # set label
    plt.xlabel(attr, fontsize=14)
    # increment a
```

```
    a+=1
# show plot
plt.show()
```



6.2 0.17 Scatter Matrix for Numerical Measures

[41]:
```
# plot a scatter matrix using pandas plotting
pd.plotting.scatter_matrix(join_df.loc[:, numerical_measures], alpha=0.2,␣
↪figsize=(16,12))
# show the plot
plt.show()
```

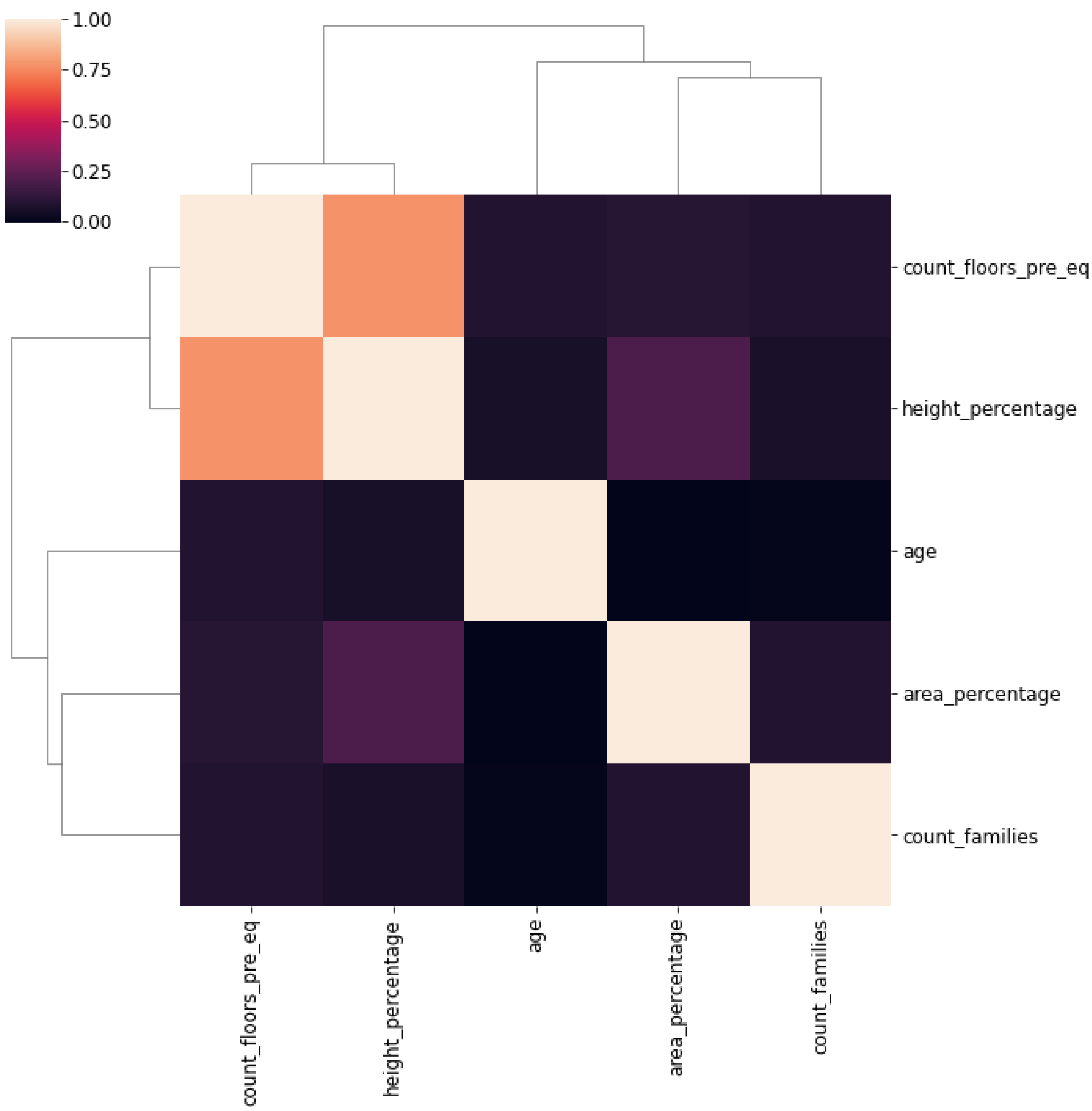


The above Scatter matrix provides the summary of histograms and scatter plots between each numerical features

6.3 0.18 Correlation Cluster Map using Seaborn for Numerical Measures

Heirarchical clustering using Ward Linkage for only Numerical Measures Source: Given two pairs of clusters whose centers are equally far apart, Ward's method will prefer to merge the smaller ones.

[42]:
```
def plot_correlation_clustermap_numerical():
    main_df = pd.get_dummies(join_df.loc[:, numerical_measures])
    sns.clustermap(data=main_df.corr(), method='ward')

plot_correlation_clustermap_numerical()
```



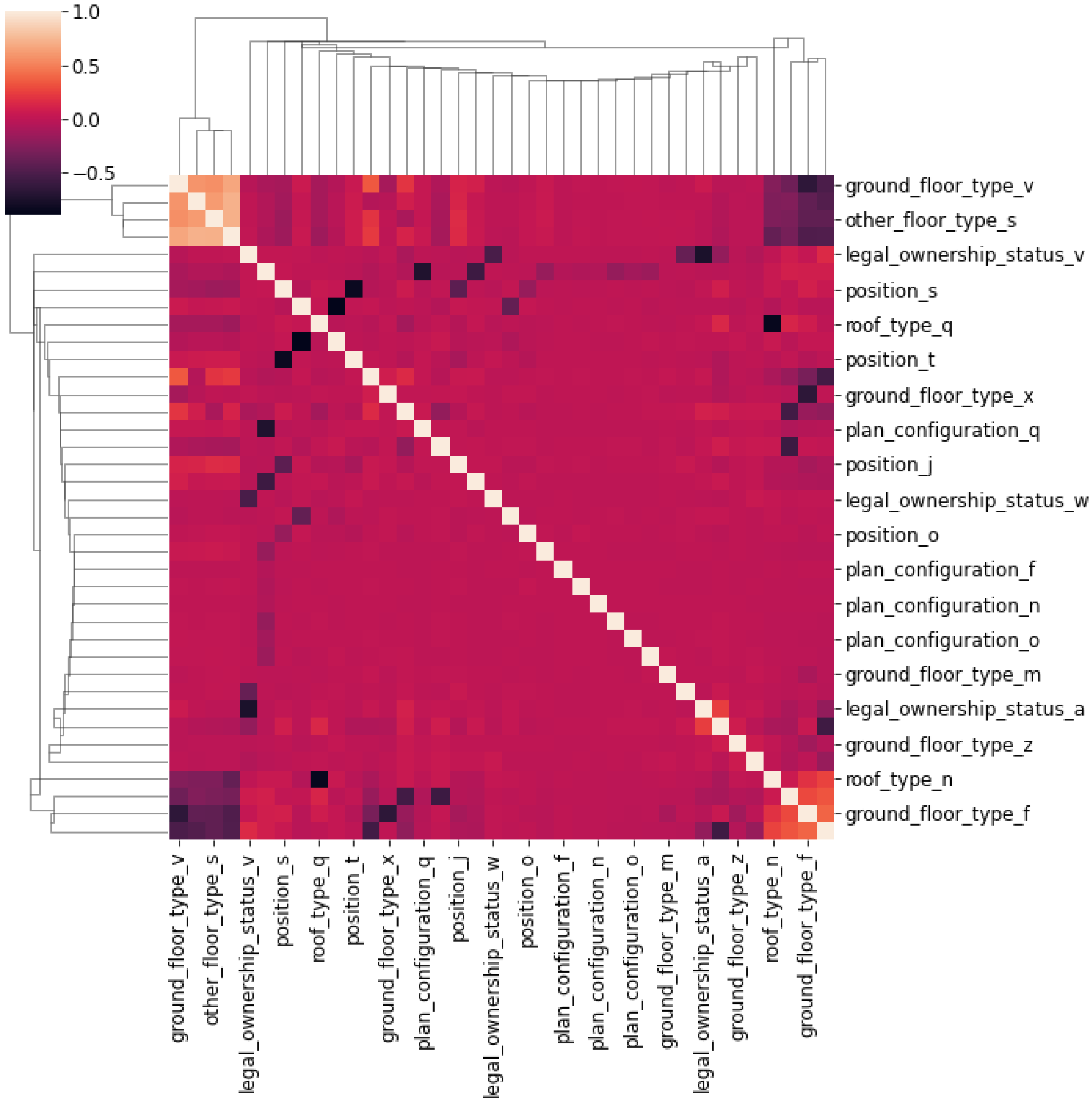
6.4 0.19 Correlation Cluster Map using Seaborn for Categrical Variables/Attributes

Heirarchical clustering using Centroid for only Categorical Measures Centroid checks for Euclidean Similarity, and the technique is popular in KMeans Algorithm

[43]:
```
def plot_correlation_clustermap_categorical():
    fig = plt.figure(figsize=(18,18))
    main_df = pd.get_dummies(join_df.loc[:,␣
 ↪main_building_land_attributes+sub_building_land_attributes])
    sns.clustermap(data=main_df.corr(), method='centroid')

plot_correlation_clustermap_categorical()
```

```
plt.show()
```

```
<Figure size 1296x1296 with 0 Axes>
```



7 1.0 Research Questions

7.0.1 Bar plot of Categorical Features

7.1 1.1 Research Question 1

1.1 What are the most frequently occurring Seismic Vulnerability Factors within Building/Land Characteristics? The Seismic Vulnerability Factors are:

- Land Surface Condition (LSC)

- Foundation Type (FT)
- Roof Type (RT)
- Ground Floor Type (GFT)
- Other Floor Type (OFT)
- Position
- Plan Configuration
- Legal Ownership Status

Calculation of Error Bars (using Population Proportion) 'p' is the Probability with which each earthquake event occurs. This happens on a Multinomial Distribution with 3 or more categorical variables defining the Damage. Over a population of data, the frequency with which the event occurs is known presently.

The number of earthqukes have been taken to be 1000, and the error bars have been projected to a different scale for ease of representation.

Based on the Population Proportion,

Standard Error (s.e) for a single Earthquake = *** $\sqrt{\frac{p(1-p)}{n}}$

Standard Error (s.e) of N earthquakes = *** $\sqrt{\frac{N*p(1-p)}{n}}$

This is because the 95% Confidence of all 95% Confidence Intervals will include the Population Proportion. Hence the Variance increases by factor N and standard error by Square Root of N

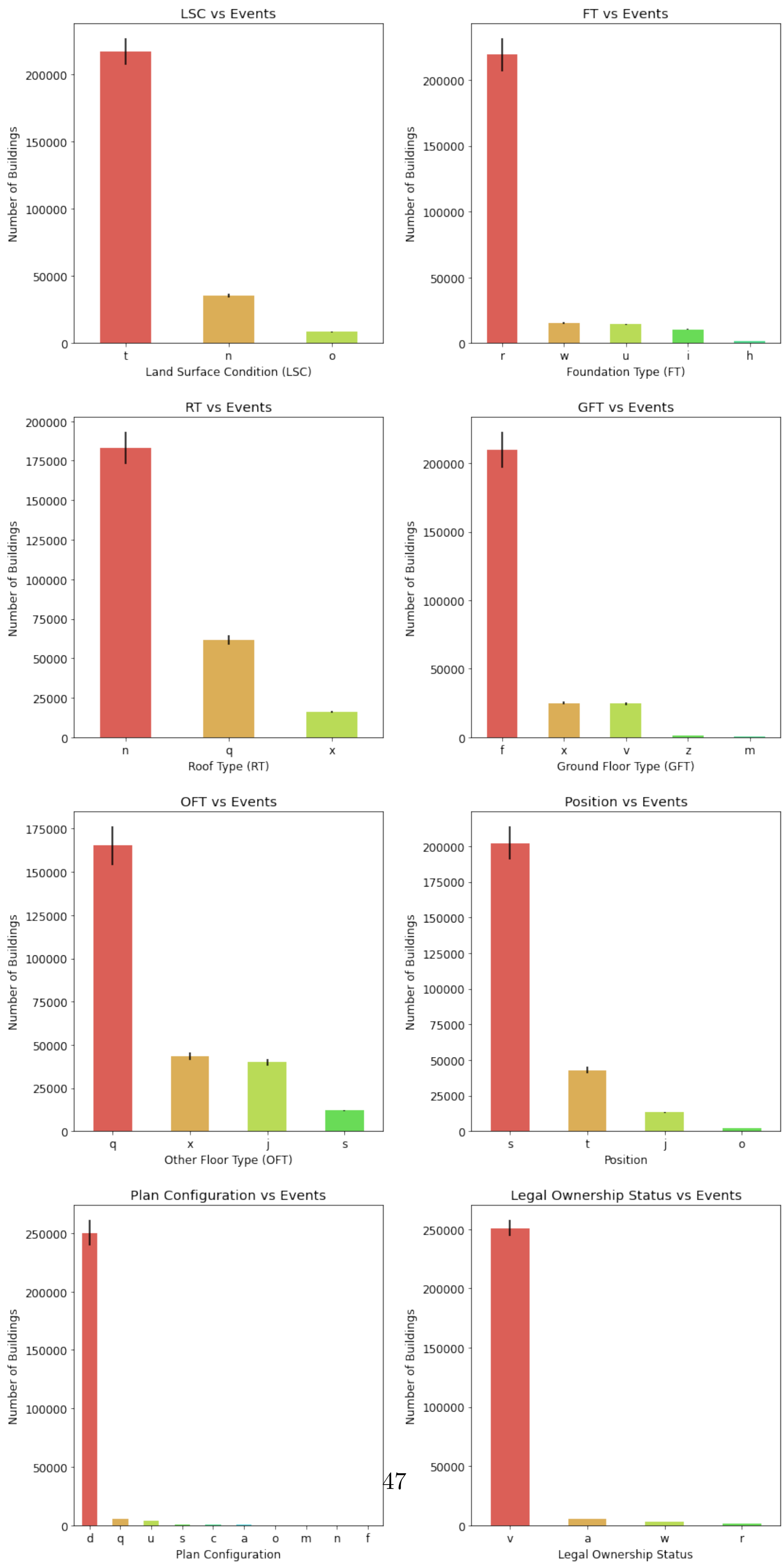
[44]:
```
colors = sns.color_palette("hls", 10)
# combine all categorical attributes from main categorical and sub categorical␣
↪(Attribute Classification of the dataset)
more_destructions_causes = join_df.loc[:, main_building_land_attributes +␣
↪sub_building_land_attributes]

# find errors for bar plot
def error_bars(value_counts, no_of_earthquakes=10):
    # number of such bars generated by a single earthquake
    no_of_events = no_of_earthquakes * len(value_counts.values)
    # calculation of probabilities
    probabilities = value_counts / value_counts.sum()
    # calculation of standard error for Population Proportion By Bernoulli␣
↪Distribution
    sep = [np.sqrt(no_of_events*p*(1-p)/len(more_destructions_causes)) for k,p␣
↪in probabilities.iteritems()]
    return sep
```

[45]:
```
def plot_pandas_kind_bar(df, attr, ax1, c, xlabel="", ylabel="", title=""):
```

```
    # value counts of building/land characteristics
    value_counts = df[attr].value_counts().sort_values(ascending=False)

    # calculate error bars
    errors = error_bars(value_counts, no_of_earthquakes=10)
    z_score = 1.96
    yerr = np.array(errors) * z_score

    # testing code for error bars
    number_of_earthquakes = 10
    # number of such bars generated by a single earthquake
    no_of_events = number_of_earthquakes * len(value_counts.values)
    # calculation of probabilities
    probabilities = value_counts / value_counts.sum()
    # calculation of standard error by Multinomial Distribution
    sep = [np.sqrt(no_of_events*p*(1-p)/len(more_destructions_causes)) for k,p␣
 ↪in probabilities.iteritems()]

    assert (np.array(sep) * 1.96).tolist() == yerr.tolist(), "Test #1 Failed"

    # for ease of representation of error bars
    projection = (1.5e6 * probabilities)
    yerr = yerr * projection

    ax1.set_xticks(range(0,len(value_counts.index.get_level_values(0))))
    ax1.set_xticklabels(value_counts.index.get_level_values(0))

    assert [t.get_text() for t in ax1.get_xticklabels()] == value_counts.index.
 ↪get_level_values(0).values.tolist(), "Test #2 Failed"

    # plotting
    value_counts.plot(kind='bar', rot=0, color=colors[:len(value_counts.index.
 ↪get_level_values(0))], ax=ax1, yerr=yerr)
    # setting labels
    ax1.set(xlabel=xlabel, ylabel=ylabel, title=title)

# plot subplots
fig, ax = plt.subplots(4,2, figsize=(12,24))

fig.suptitle("Figure 1.1 - Most Frequently Occuring Seismic Vulnerability␣
 ↪Factors")

# subplot LSC using value counts and order by descending order
plot_pandas_kind_bar(more_destructions_causes, "land_surface_condition",␣
 ↪ax[0,0], colors[:3],
                     xlabel="Land Surface Condition (LSC)", ylabel="Number of␣
 ↪Buildings", title="LSC vs Events")
```

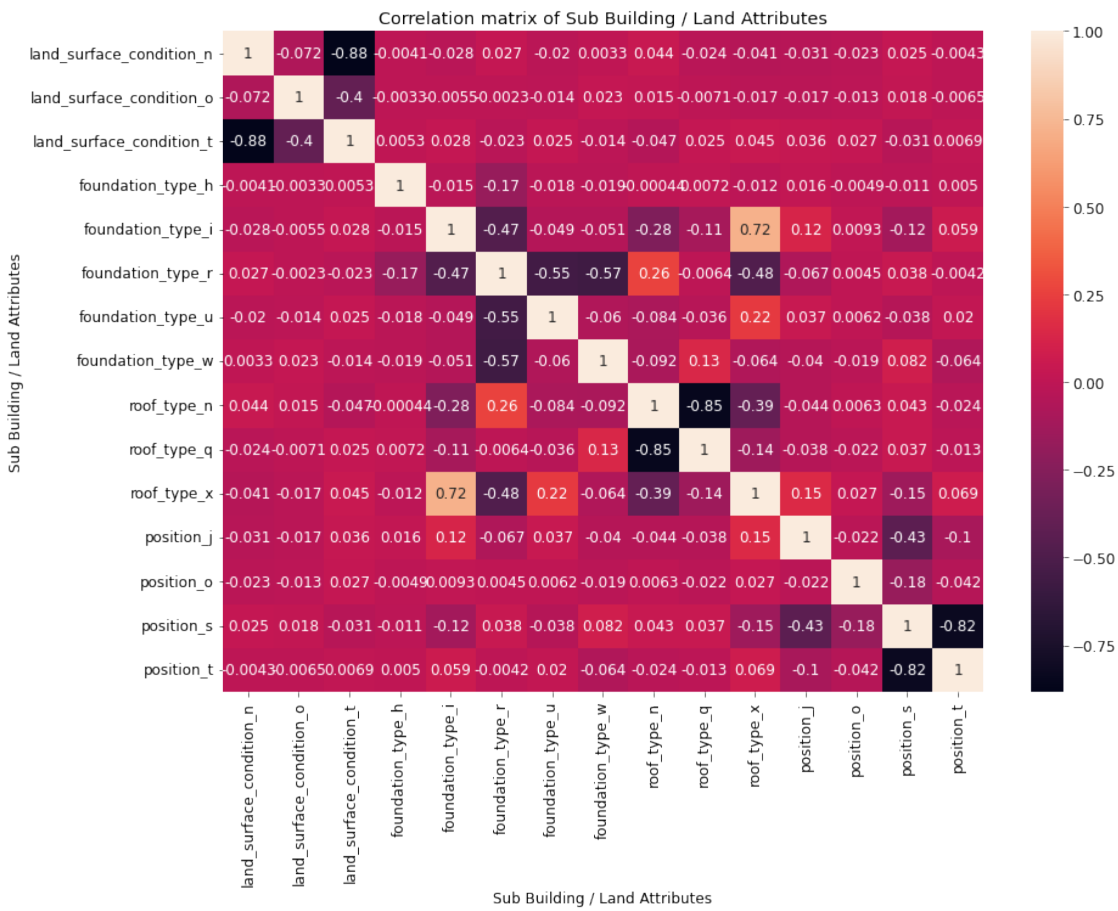
```
# subplot FT using value counts and order by descending order
plot_pandas_kind_bar(more_destructions_causes, "foundation_type", ax[0,1],␣
↪colors[:5],
                     xlabel="Foundation Type (FT)", ylabel="Number of␣
↪Buildings", title="FT vs Events")
# subplot RT using value counts and order by descending order
plot_pandas_kind_bar(more_destructions_causes, "roof_type", ax[1,0], colors[:3],
                     xlabel="Roof Type (RT)", ylabel="Number of Buildings",␣
↪title="RT vs Events")
# subplot GFT using value counts and order by descending order
plot_pandas_kind_bar(more_destructions_causes, "ground_floor_type", ax[1,1],␣
↪colors[:5],
                     xlabel="Ground Floor Type (GFT)", ylabel="Number of␣
↪Buildings", title="GFT vs Events")
# subplot OFT using value counts and order by descending order
plot_pandas_kind_bar(more_destructions_causes, "other_floor_type", ax[2,0],␣
↪colors[:4],
                     xlabel="Other Floor Type (OFT)", ylabel="Number of␣
↪Buildings", title="OFT vs Events")
# subplot Position using value counts and order by descending order
plot_pandas_kind_bar(more_destructions_causes, "position", ax[2,1], colors[:4],
                     xlabel="Position", ylabel="Number of Buildings",␣
↪title="Position vs Events")
# subplot Plan Configuration using value counts and order by descending order
plot_pandas_kind_bar(more_destructions_causes, "plan_configuration", ax[3,0],␣
↪colors[:10],
                     xlabel="Plan Configuration", ylabel="Number of Buildings",␣
↪title="Plan Configuration vs Events")
# subplot Legal Ownership Status using value counts and order by descending␣
↪order
plot_pandas_kind_bar(more_destructions_causes, "legal_ownership_status",␣
↪ax[3,1], colors[:4],
                     xlabel="Legal Ownership Status", ylabel="Number of␣
↪Buildings", title="Legal Ownership Status vs Events")
# tight_layout for plot
plt.tight_layout(pad=2.0)
```

Figure 1.1 - Most Frequently Occuring Seismic Vulnerability Factors



7.1.1 Correlation of Categorical Features

[46]:
```python
corr = pd.get_dummies(join_df.loc[:, sub_building_land_attributes]).corr()
fig = plt.figure(figsize=(14,10))
hm = sns.heatmap(corr, annot = True)
hm.set(xlabel='Sub Building / Land Attributes', ylabel='Sub Building / Land␣
↪Attributes', title = "Correlation matrix of Sub Building / Land Attributes")
plt.show()
```



7.1.2 Supporting methods

- To find most frequently occurring Seismic Vulnerability Factors within Building/Land Characteristics creating Bar Graphs visualization with Pandas Matplotlib
- Represent counts of seismic vulnerability factors.
- Bars in the graph in decreasing order of measured values

7.1.3 Facts for each Seismic Vulnerability

Land Surface Condition (LSC) Facts:

Land Surface Condition	Count	Probability
LSC (t)	216757	0.8318
LSC (n)	35528	0.1363
LSC (o)	8316	0.0319

- According to the dataset, 't' is the most commonly occurring LSC. Considering the population of buildings before damage and after damage, the assumption is that 't' must remain the most frequently occurring construction parameter within LSC.
- According to the literature review, 't' could be Terrain and terrain surfaces are commonly seen in the Earthquake sites of Nepal.
- If 't' is terrain, the literature review states Plains region, implying the assumption is 'n' is Normal and 'o' is Other.

Foundation Type (FT) Facts:

Foundation Type	Count	Probability
FT (r)	219196	0.8411
FT (w)	15118	0.058
FT (u)	14260	0.0547
FT (i)	10579	0.0406
FT (h)	1448	0.000556

- According to the dataset, 'r' is most commonly occuring FT. Assumption is that r will remain the most frequently occuring construction parameter within Foundation Type.
- According to the literature review, 'r' could be Raft Foundation Type, 'w' could be Wide-Strip, 'h' could be hardcore (which is the least commonly occuring).
- 'r' is positively correlated when compared to n (Normal) than o and t (terrain) which are negatively correlated.
- 'h' is positively correlatd to the Terrain land surface condition with Pearson R correlation coeffiicient = **0.005329**

Roof Type (RT) Facts:

Roof Type	Count	Probability
RT (n)	182842	0.7016
RT (q)	61576	0.2363
RT (x)	16183	0.0621

- According to the dataset, 'n' is most commonly occuring RT. Assumption is that 'n' will remain the most frequently occuring construction parameter within RT.
- According to the literature review, 'n' could be Normal, 'q' could be Quartz and 'x' could be

Truss.

- RT ‘x’ is highly correlated with ‘i’ Foundation Type.

Ground Floor Type (GFT) Facts:

Ground Floor Type	Count	Probability
GFT (f)	209619	0.8044
GFT (x)	24877	0.0955
GFT 9v)	24593	0.0944
GFT (z)	1004	0.000385
GFT 9m)	508	0.000195

Other Floor Type (OFT) Facts:

Other Floor Type	Count	Probability
OFT (q)	165282	0.6342
OFT (x)	43448	0.1667
OFT (j)	39843	0.1529
OFT 9s)	12028	0.0462

Position Facts:

Position	Count	Probability
s	202090	0.7755
t	42896	0.1646
j	13282	0.05097
o	2333	0.000895

Plan Configuration Facts:

Plan Configuration	Count	Probability
d	250072	0.9596
q	5692	0.0218
u	3649	0.014
s	346	0.000133
c	325	0.000125
a	252	9.67E-04
o	159	6.10E-04
m	46	1.77E-04
n	38	1.46E-04
f	22	8.44E-05

Legal Ownership Status Facts:

Legal Ownership Status	Count	Probability
v	250939	0.9629
a	5512	0.0212
w	2677	0.0103
r	1473	5.65E-03

7.1.4 Observations:

- **Land Surface Condition (LSC)**, t (terrain surfaces) is most affected by earthquakes. ‘t’ (terrain surfaces) occurs more compared to ‘n’ (Normal) and ‘o’ (Other)
- **In Foundation Type (FT)**, ‘r’ (Raft Foundation Type) is most commonly occurring with compared to ‘w’ (Wide-Strip), ‘h’ (hardcore).
- ‘h’ (hardcore) is positively correlated to the Terrain land surface condition(LSC). Similarly Foundation Type (FT) ‘r’ is positively correlated with land surface conditions(LSC) types like n (Normal) than o and t (terrain) which are negatively correlated.
- In **Roof Type (RT)** ‘n’ (Normal) is the most commonly occurring Roof Type compared to ‘q’ Quartz, ‘x’ Truss. In Roof Type (RT) ‘x’ is highly correlated with ‘i’ Foundation Type.
- In **Ground Floor Type (GFT)**, f (Floating) Ground Floor Type is affecting more compared to other GFT. M (Mud) ground floor type is least affecting in the earthquake
- In **Other Floor Type (OFT)** ‘q’ type floors are more affected by earthquakes compared to other floor types like ‘x’, ‘j’, ‘s’.
- Similarly, the **position** of the building ‘s’ position affects more compared to other positions like ‘t’, ‘j’, ‘o’ when an earthquake occurs.
- In **Plan Configuration** ‘d’ type of plan configuration, and finally in **Legal Ownership Status** ‘v’ type of Legal Ownership Status affecting more in the earthquake.

7.1.5 Answer to the Research Question

With the above analysis the conclusion on the most frequently occurring Seismic Vulnerability is high in the below conditions. * Land Surface Condition (LSC) is t (Terrain). * Foundation type (FT) is r (Raft Foundations). * roof_type (RT) is n (Normal). * ground_floor_type is f (Floating) * other_floor_type is q * position of the building is s * building plan configuration is d * Legal Ownership Status is ‘v’

7.2 2.1 Research Question 2

7.2.1 2.1 What is the Percentage of Superstructure Construction Buildings that have undergone low, medium, and high levels of damage?

Plot of Superstructure Attributes showing their percentage contribution towards damage grade 1,2,3

[47]:
```
def melt_dataframe(join_df, ss_attributes):
    # applying melt method on dataframe to get the number of buildings damaged␣
↪due to low, medium and high damage, for each superstructure construction.
    temp_df = pd.melt(join_df.loc[:, ss_attributes + ['damage_grade']],␣
↪var_name="building_type", id_vars=['damage_grade'])
```

```
    # extracting only those entries with superstructure constructions
    temp_df = temp_df.loc[temp_df['value'] == 1]
    # apply map changing from 1,2,3 to low, medium, high
    temp_df["damage_grade"] = temp_df["damage_grade"].map({1: "low", 2:␣
↪"medium", 3: "high"})
    return temp_df
```

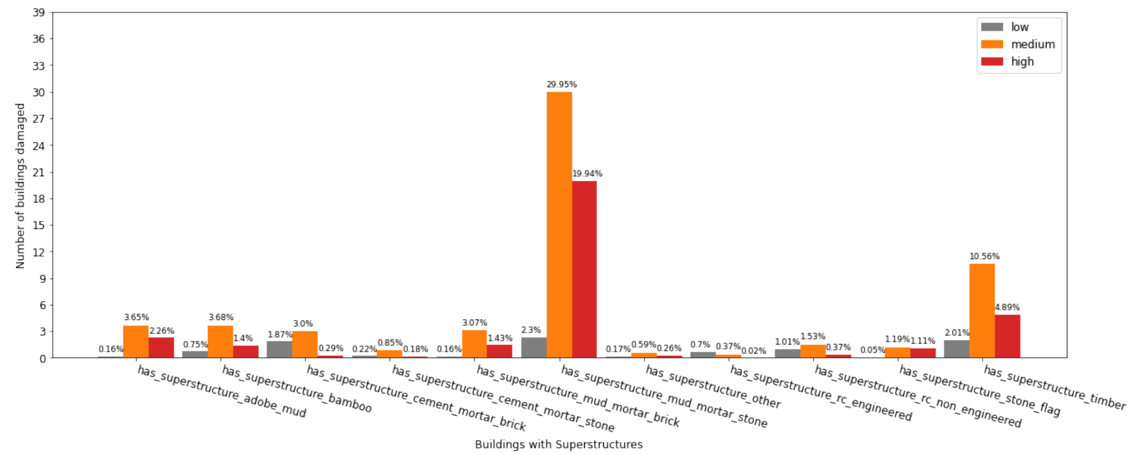
Function to create a cross tabulation representation of dataframe

[48]:
```
def create_crosstab(temp_df):
    # applying cross tab to form a cross tabulation of building type and damage␣
↪grade.
    df = pd.crosstab(temp_df["building_type"], temp_df["damage_grade"])
    #calculating percentages between groups (axis=1)
    df = df.apply(lambda x: round(x / (df['low'].sum()+df['medium'].
↪sum()+df['high'].sum()) * 100,2))
    return df
```

Function to plot side by side bar plot to display damage impact to superstructurs that have undergone low, medium and high damage

[49]:
```
def plot_sidebarplot(superstructure_attributes):
    # get cross tab values
    df = create_crosstab(melt_dataframe(join_df, superstructure_attributes))
    # define label for x axis
    x= np.arange(len(superstructure_attributes))
    # define label for y axis
    y_label = np.arange(0,40,3)
    plt.figure(figsize=(20,7))
    # bar plot for low, medium and high percentage
    plot_low= plt.bar(x-0.3, height= df["low"], width=0.3, color='tab:gray')
    plot_medium= plt.bar(x, height= df["medium"],width=0.3, color='tab:orange')
    plot_high= plt.bar(x+0.3, height= df["high"],width=0.3, color='tab:red')
    # plotting xticks
    plt.xticks(x, superstructure_attributes, ha="left",rotation=345)
    plt.yticks(y_label)
    # display percentage as text for bars representing low impact.
    for i in plot_low.patches:
      plt.text(i.get_x(), i.get_height()+0.5, str(i.get_height())+'%',␣
↪fontsize=9,color='0')

    # display percentage as text for bars representing medium impact.
    for i in plot_medium.patches:
      plt.text(i.get_x(), i.get_height()+0.6, str(i.get_height())+'%',␣
↪fontsize=9,color='black')

    # display percentage as text for bars representing high impact.
```

```python
    for i in plot_high.patches:
      plt.text(i.get_x(), i.get_height()+0.5, str(i.get_height())+'%', fontsize=9,color='black')

    # adding title and lables for plot.
    fig.suptitle("Figure 2.1 - Damage By Buildings constructed with Structure Type")
    plt.xlabel("Buildings with Superstructures")
    plt.ylabel("Number of buildings damaged")
    # plotting legend
    plt.legend(["low", "medium", "high"], loc = 'upper right')
    plt.show()
plot_sidebarplot(superstructure_attributes)
```



7.3 Supporting methods

- Visualization with Matplotlib and Side-by-side bar graphs are used
- Pandas CrossTab used after DataFrame Melt
- Sliced by Superstructures

7.4 Facts:

Superstructure buildings damaged due to three different damage grades(**below information is arranged in ascending order of total damage**)

- Superstructure RC engineered buildings: *Level 1: 0.7%,* *Level 2: 0.37%,* *Level 3: 0.02%*
- Superstructure Other: *Level 1: 0.17%,* *Level 2: 0.59%,* *Level 3: 0.26%*
- Superstructure cement mortar stone: *Level 1: 0.22%,* *Level 2: 0.85%,* *Level 3: 0.18%*
- Superstructure stone flag: *Level 1: 0.05%,* *Level 2: 1.19%,* *Level 3: 1.11%*
- Superstructure RC non engineered: *Level 1: 1.01%,* *Level 2: 1.53%,* *Level 3: 0.37%*
- Superstructure mud mortar brick: *Level 1: 0.16%,* *Level 2: 3.07%,* *Level 3: 1.43%*
- Superstructure cement mortar brick: *Level 1: 1.87%,* *Level 2: 3.0%,* *Level 3: 0.29%*

- Superstructure bamboo: *Level 1: 0.75%, Level 2: 3.68%, Level 3: 1.4%*
- Superstructure adobe mud: *Level 1: 0.16%, Level 2: 3.65%, Level 3: 2.26%*
- Superstructure Timber: *Level 1: 2.01%, Level 2: 10.56%, Level 3: 4.89%*
- Superstructure mud mortar and stone: *Level 1: 2.3%, Level 2: 29.95%, Level 3: 19.94%*

7.5 Observations:

- Buildings constructed with superstructure **'mud mortar stone'** had the maximum destruction caused by all three level of damage among all the superstructures.
- Buildings constructed with superstructure **'Timber'** had second highest destruction caused by all three level of damage among all the superstructures.
- Buildings constructed with superstructure **'RC engineered'** had minimum damage posed by both high and medium level of damage grade.

7.6 Answer to the Research Question:

Maximum damage was posed to the buildings constructed with **mud mortar and stone** with 19.94%, 29.95%, and 2.3% buildings damaged due to level 3, level 2 and level 1 grade respectively. On the other side, **minimum damage** was posed to the buildings constructed with **RC engineered** with 0.02%, 0.37% and 0.7% buildings damaged due to level 3, level 2 and level 1 grade respectively. Therefore RC engineered superstructures are recommended in future construction.

Buildings constructed with **Superstructure Other** have had **second lowest damage** with 0.26%, 0.59% and 0.17% buildings damaged due to level 3, level 2 and level 1 grade respectively.Buildings constructed with **Timber** have had **second highest damage** with 4.89%, 10.56% and 2.01% buildings damaged due to level 3, level 2 and level 1 grade respectively.

7.7 3.1 Research Question 3

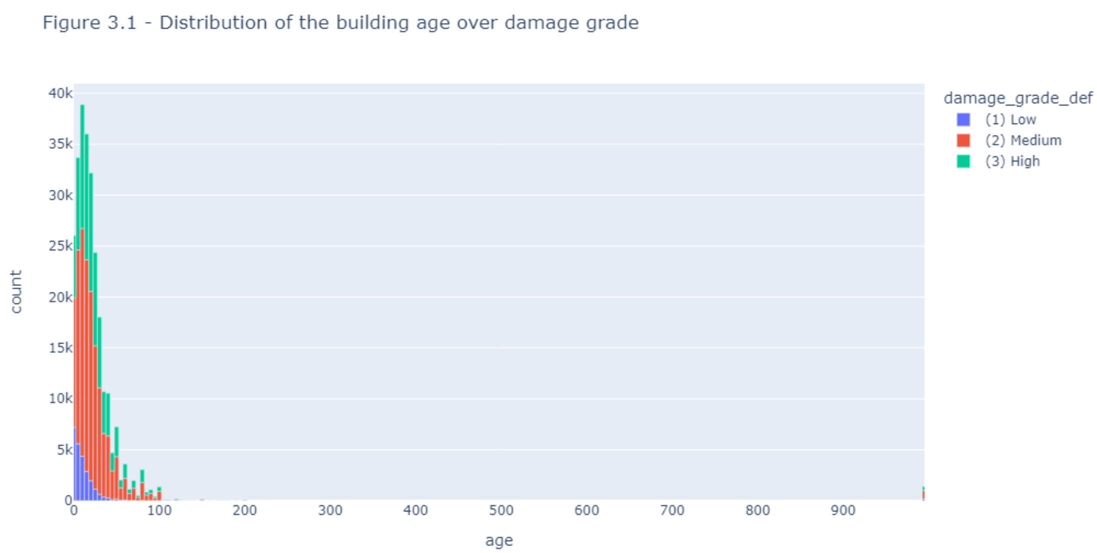
7.7.1 3.1 What is the distribution of building age over damage grade, and the percentage of damage for age ranges such as 0-10, 10-15, 15-30 and 30-995?

Method to represent damage levels 1,2,3 as 'Low', 'Medium' and 'High.

[50]:
```
def represent_damage_level(merged_data):
    # copy the dataframe
    merged_data = join_df.copy()
    # creating an additional columns as 'damage_grade_def' to store the␣
↪representation of damage levels.
    merged_data['damage_grade_def'] = np.where(merged_data.damage_grade==1,'(1)␣
↪Low',
    np.where(merged_data.damage_grade==2,'(2) Medium',np.where(merged_data.
↪damage_grade==3,'(3) High',0)))
    return merged_data
```

Method to plot the distribution of age across different damage levels.

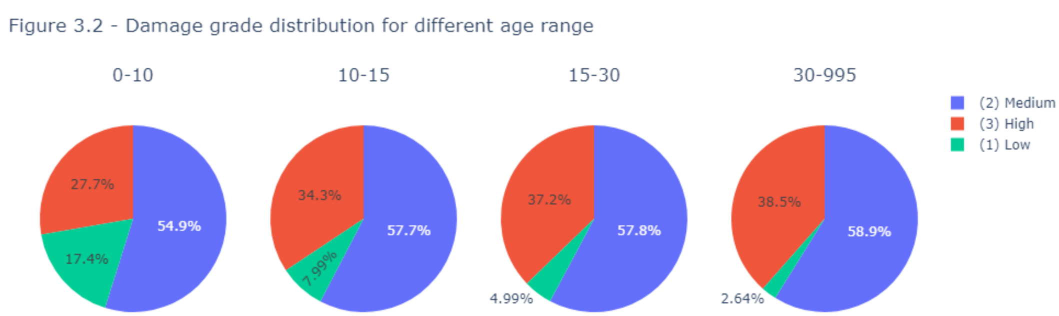
```
[51]: def plot1(join_df, age, rt, dgd):
          # represent damage levels as 1,2,3
          merged_data = represent_damage_level(join_df)
          # Bar plot within plotly express is used to plot the distribution
          fig = px.bar(merged_data.groupby(['age','damage_grade_def']).roof_type.
       ↪count().reset_index().rename(columns={'roof_type':'count'}),
          x="age", y="count", color="damage_grade_def", title="Figure 3.1 -␣
       ↪Distribution of the building age over damage grade",width=2500, height=500)
          # Updating X axis to have the age between 0 to 995.
          fig.update_xaxes(range=[0, 995])
          # changing the width of bars.
          for data in fig.data:
            data["width"] = 4.9
          img_bytes = pio.to_image(fig, format="png", engine="kaleido", width=1024,␣
       ↪height=560)
          return img_bytes
      display(Image(plot1(join_df, 'age', 'roof_type', 'damage_grade_def')))
```



Method to plot piecharts showing distribution of damage grade, for different age range.

```
[52]: def draw_subplotted_pie_chart(col, target_col):
          # represent damage levels as 1,2,3
          merged_data = represent_damage_level(join_df)
          # define labels with different age range.
          labels=['0-10','10-15','15-30','30-995']
          # cut the dataset using pandas.qcut
          merged_data['age_range'] = pd.qcut(merged_data.age,4,labels=labels)
```

```
    # figure containing subplots is defined.
    fig = make_subplots(rows=1, cols=4, specs=[[{'type':'domain'},
                                                {'type':'domain'},
                                                {'type':'domain'},
                                                {'type':'domain'}]],␣
↪subplot_titles = labels)
    # iterating labels using for loop to plot pie charts for different age␣
↪range.
    for i,lb in enumerate(labels):
        labeled = merged_data[merged_data[target_col]==lb]
        counted = pd.DataFrame(labeled.groupby(col)[col].count()).
↪rename(columns={col:'Count'}).reset_index()
        fig.add_trace(go.Pie(values=counted.Count, labels=counted[col],␣
↪name=lb),1,i+1)
    fig.update_layout(title_text= 'Figure 3.2 - Damage grade distribution for␣
↪different age range')
    img_bytes = pio.to_image(fig, format="png", engine="kaleido", width=1024,␣
↪height=420)
    return img_bytes
display(Image(draw_subplotted_pie_chart('damage_grade_def', 'age_range')))
```



7.7.2 Supporting methods

- Visualization is performed using Plotly
- Represented Histograms of Age using Stacked Bar plot
- Pie-chart to show the breakdown of Age (in %) from 0 to Extreme Values

7.7.3 Facts:

- Stacked barplot indicates that the maximum damage was posed to the buildings of age 10 with 4360 buildings damaged due to level 1 grade, 22370 buildings damaged with level 2 grade, and 12166 building damaged due to level 3 grade.

- Stacked barplot indicated that the buildings with lower age have had major impact as compared to the buildings of higher age.
- Stacked bar plot indicates that there were some historic buildings of age 995 which have mainly had level 2 damage and impacting 822 such buildings, level 3 damage has impacted 389 buildings, level 1 damage has impacted 179 buildings.
- Pie chart depicts that the percentage of damage for 'High level' damage has increased with the increase of Age Range. Percentage of damage is 27.7%, 34.3%, 37.2%, and 38.5% for the age range 0-10, 10-15, 15-30, and 30-995 respectively.
- Pie chart depicts that the percentage of damage for 'Medium level' damage has increased with the increase of Age Range. Percentage of damage is 54.9%, 57.7%, 57.8%, and 58.9% for the age range 0-10, 10-15, 15-30, and 30-995 respectively.
- Pie chart depicts that the percentage of damage for 'low level' damage has decreased with the increase of Age Range. Percentage of damage is 17.4%, 7.99%, 4.99%, and 2.64% for the age range 0-10, 10-15, 15-30, and 30-995 respectively.

7.7.4 Observations:

- Most of the buildings involved in the earthquake were in the age range of 0-50.
- New buildings with age as 0 have less number of buildings with 'High damage grade', whereas maximum buildings were damaged with Medium level of damage grade.
- Percentage of 'High level damage' is increasing with the increase of building Age.
- Percentage of 'Low level damage' is decreasing with the increase of building Age.
- Percentage of 'Medium level damage' is increasing with the increase of building Age.
- There were very few buildings between the age range of 120 to 994.

7.7.5 Answer to the Research Question:

Distribution of age with respect to damage level indicates that the damage to the buildings was higher for newer buildings, specially between age 0 to 50, whereas the damage has reduced significantly with the increase of age. In contrast, new build buildings have had less damage caused as compared to the buildings of age between 5 to 15. There were very few buildings between age range 150 to 994. In additon, there were 1390 historic buildings which were 995 years old and out of which 179 had low level impact, 822 had medium level impact and 389 had high level impact.

Below is the percentage of damage for different age range such as 0-10, 10-15, 15-30 and 30-995:-

- As per piechart, percentage damage for building with age range 0-10
  - Level 1 Damage: 17.4%
  - Level 2 Damage: 54.9%
  - Level 3 Damage: 27.7%
- As per piechart, percentage damage for building with age range 10-15
  - Level 1 Damage: 7.99%
  - Level 2 Damage: 57.7%
  - Level 3 Damage: 34.3%
- As per piechart, percentage damage for building with age range 15-30
  - Level 1 Damage: 4.99%

- Level 2 Damage: 57.8%
  - Level 3 Damage: 37.2%
- As per piechart, percentage damage for building with age range 30-995
  - Level 1 Damage: 2.64%
  - Level 2 Damage: 58.9%
  - Level 3 Damage: 38.5%

7.8 4.1 Research Question 4

7.8.1 4.1 What is the relationship between Area Percentage and Age?

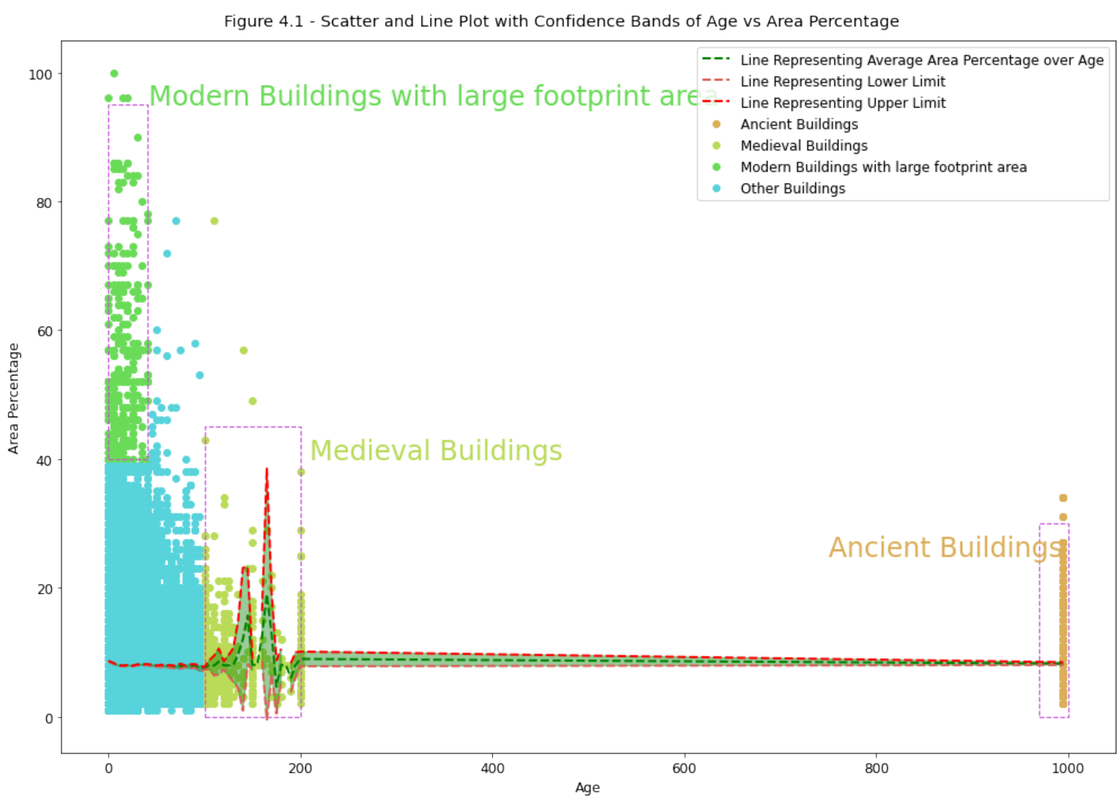
7.8.2 4.1 Scatter and Line Plot with Confidence Bands of Age vs Area Percentage

[53]:
```
fig, ax = plt.subplots(1, 1, figsize=(14,10))
colors = sns.color_palette("hls", 10)

def plot_area_percentage_vs_age(join_df):
    '''
    Plots Area Percentage and Age in a scatter plot
    @param join_df: Main DataFrame
    @return:
    '''
    # scatter plot of Ancient Buildings
    ancient_mask = join_df['age'] == 995
    ax.scatter(join_df.loc[ancient_mask].age, join_df.loc[ancient_mask].
 ↪area_percentage, color=colors[1], label='Ancient Buildings')

    # scatter plot of Medieval Buildings
    medieval_mask = (join_df['age'] >= 100) & (join_df['age'] <= 200)
    ax.scatter(join_df.loc[medieval_mask].age, join_df.loc[medieval_mask].
 ↪area_percentage, color=colors[2], label='Medieval Buildings')

    # scatter plot of Modern Buildings with Large Footprint Area
    modern_mask = (join_df['age'] <= 40) & (join_df['area_percentage'] >= 40)
    ax.scatter(join_df.loc[modern_mask].age, join_df.loc[modern_mask].
 ↪area_percentage, color=colors[3], label='Modern Buildings with large␣
 ↪footprint area')

    # scatter plot between Count of floors and Average height Percentage
    ax.scatter(join_df.loc[(np.logical_not(ancient_mask)) & (np.
 ↪logical_not(medieval_mask)) & (np.logical_not(modern_mask))].age,
               join_df.loc[(np.logical_not(ancient_mask)) & (np.
 ↪logical_not(medieval_mask)) & (np.logical_not(modern_mask))].area_percentage,
               color=colors[5], label='Other Buildings')

    # set legend
    ax.legend()
```

```
    # set title, xlabel and ylabel
    ax.set(xlabel="Age", ylabel="Area Percentage")
    fig.suptitle("Figure 4.1 - Scatter and Line Plot with Confidence Bands of␣
 ↪Age vs Area Percentage")
    # tight layout
    fig.tight_layout(pad=1.0)

def plot_average_line_representing_area_percentage_over_age(join_df):
    '''
    Plots Average Line of Area Percentage over Age
    @param join_df: Main DataFrame
    @return:
    '''
    # aggregation of area percentage with mean and standard error
    g = join_df.groupby('age')['area_percentage'].agg(['mean', 'sem'])
    # plot the average line
    ax.plot(g.index, g['mean'], color='green', label='Line Representing Average␣
 ↪Area Percentage over Age', ls='dashed', lw=2)
    # plot the lower limit of the average line
    ax.plot(g.index, g['mean']-1.96*g['sem'], color=colors[0], label='Line␣
 ↪Representing Lower Limit', ls='dashed', lw=2)
    # plot the upper limit of the average line
    ax.plot(g.index, g['mean']+1.96*g['sem'], color='red', label='Line␣
 ↪Representing Upper Limit', ls='dashed', lw=2)
    # fill the confidence intervals
    ax.fill_between(g.index, g['mean']+1.96*g['sem'], g['mean']-1.96*g['sem'],␣
 ↪edgecolor='g', facecolor='g', alpha=0.4)

    # display the legend
    ax.legend()

def plot_high_variance_area():
    '''
    Plots the High variance Area detected by the fluctuations in Area␣
 ↪Percentage values
    @return:
    '''
    # Loop over data points; create box from errors at each point
    high_variance_box = Rectangle((100, 0), 100, 45, fill=False, ls='dashed',␣
 ↪lw=1.15, color=colors[8])
    ax.add_patch(high_variance_box)
    ax.text(210, 40, "Medieval Buildings", fontsize=24, color=colors[2])
    ancient_box = Rectangle((970, 0), 30, 30, fill=False, ls='dashed', lw=1.15,␣
 ↪color=colors[8])
    ax.add_patch(ancient_box)
    ax.text(750, 25, "Ancient Buildings", fontsize=24, color=colors[1])
```

```
    modern_box = Rectangle((0, 40), 40, 55, fill=False, ls='dashed', lw=1.15,␣
↪color=colors[8])
    ax.add_patch(modern_box)
    ax.text(42, 95, "Modern Buildings with large footprint area", fontsize=24,␣
↪color=colors[3])

plot_area_percentage_vs_age(join_df)
plot_average_line_representing_area_percentage_over_age(join_df)
plot_high_variance_area()
fig.show()
```

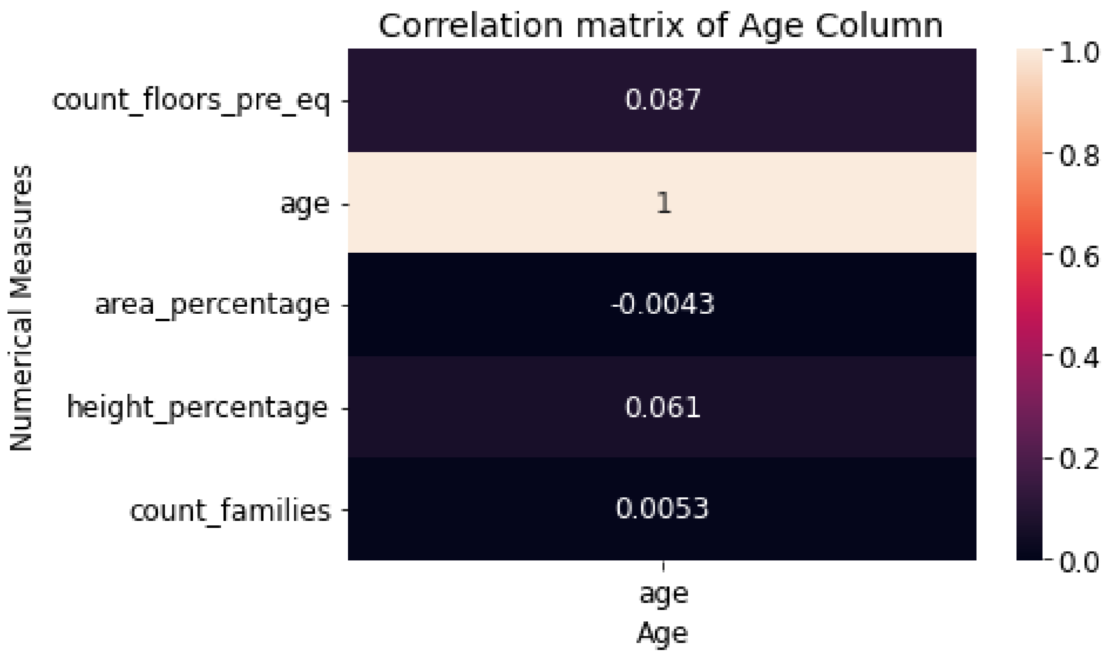
```
C:\Users\burse\AppData\Local\Temp/ipykernel_24064/1775364941.py:75: UserWarning:

Matplotlib is currently using module://matplotlib_inline.backend_inline, which
is a non-GUI backend, so cannot show the figure.
```



Correlation between Age and Area Percentage

[54]:
```
corr = join_df.loc[:, numerical_measures].corr()
hm = sns.heatmap(corr.iloc[:, [1]], annot = True)
```

```
hm.set(xlabel='Age', ylabel='Numerical Measures', title = "Correlation matrix␣
↪of Age Column")
plt.show()
```



7.8.3 Background:

From the Summary Statistics, Age and Area Percentage have relatively high variance among the Numerical Measures. Area Percentage and Height Percentage may be computed using LIDAR data. The high variances of Area and Age are explored further here.

7.8.4 Facts:

- There is a high variance region of Area Percentage between 100 - 200 years old buildings, which have been identified as **Medieval Buildings**.
- There is another region consisting of 995 years old buildings, which have been identified as **Ancient Buildings**.
- **Modern Buildings with large footprint area** are those buildings which have 'age' less than 40 and 'area_percentage' greater than 40.
- **Medieval Buidings** are those buildings which have 'age' between 100 and 200, irrespective of the range of 'area_percentage'.
- **Ancient Buildings** are those buildings which have age equal to 995, irrespective of 'area_percentage'

7.8.5 Observations:

- Age and Area Percentage are slightly negatively correlated.

- The Pearson R correlation coefficient between Age and Area percentage is **-0.004323**.
- There are lot of buildings constructured after the Medieval Period and hence the change in the variance.

7.8.6 Answer to the Research Question:

- The confidence bands improve the detection of high variance regions.
- The scatter plot shown here denotes buildings with **large footprint area** are recently constructed and they may be **Modern Buildings**.
- The reason why the **Modern Buildings** collapsed is important for the analysis.
- The government can use this annotated data to identify the **materials used** and **best practices** of Modern buildings and why they collapsed during the earthquake.

7.8.7 4.2 Sub Analysis 1 - Research Question 4

The Collapse of Modern Buildings of Superstructures due to Seismic Vulnerability Factors (Post Seismic Codes Era) **Seismic Codes were developed in 1970s by Scientists which imply the reason for collapse of Modern Buildings is too important for the Analysis**

[55]:
```
# superstructure mask for dataframe
superstructure_mask = ((join_df['has_superstructure_adobe_mud'] == 1) |
                                                ␣
↪(join_df['has_superstructure_mud_mortar_stone'] == 1) |
                                    (join_df['has_superstructure_stone_flag'] ==␣
↪1) |
                                                ␣
↪(join_df['has_superstructure_cement_mortar_stone'] == 1) |
                                                ␣
↪(join_df['has_superstructure_mud_mortar_brick'] == 1) |
                                                ␣
↪(join_df['has_superstructure_cement_mortar_brick'] == 1) |
                                    (join_df['has_superstructure_timber'] == 1) |
                                    (join_df['has_superstructure_bamboo'] == 1) |
                                                ␣
↪(join_df['has_superstructure_rc_non_engineered'] == 1) |
                                    (join_df['has_superstructure_rc_engineered']␣
↪== 1) |
                                    (join_df['has_superstructure_other'] == 1))

# modern buildings with large footprint area mask for dataframe
modern_buildings_mask = (join_df['age'] <= 40) & (join_df['area_percentage'] >=␣
↪40)

# copy the original dataframe
cat_df = join_df.loc[superstructure_mask & modern_buildings_mask].copy()
# one hot encoding of categorical variables
```

```
categorical_df = pd.get_dummies(cat_df.loc[:, main_building_land_attributes +␣
↪sub_building_land_attributes + superstructure_attributes])
# run principal components analysis
X_pca, pc, evr = principal_components_analysis(categorical_df)
```

[56]:
```
print("Number of Modern Buildings in that region:", len(categorical_df))
```

```
Number of Modern Buildings in that region: 327
```

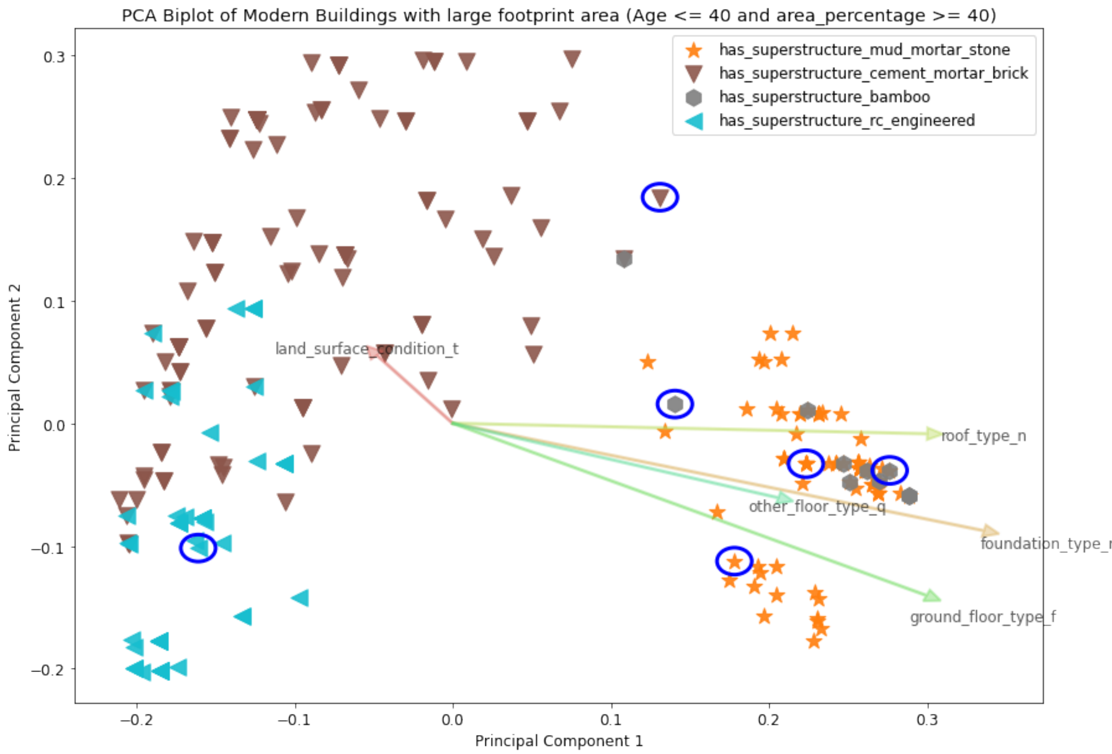
4.2 PCA Biplot of Superstructure Constructed Buildings for Modern Buildings (with large footprint area)

[57]:
```
def plot_scatter_plot_by_superstructures(X_pca, categorical_df, ax):
    # scale Principal component 1
    scalex = 0.5 / (X_pca[:,0].max() - X_pca[:,0].min())
    # scale Principal component 2
    scaley = 0.5 / (X_pca[:,1].max() - X_pca[:,1].min())
    # 10 colors color pallette
    new_palette = np.array(sns.color_palette(palette=None, n_colors=11))

    # Top 8 Building Types with greatest damage count
    damage_index = (cat_df['area_percentage'] >= 80) & (cat_df['age'] <= 10) &␣
↪((cat_df['damage_grade'] == 3) | (cat_df['damage_grade'] == 2))

    # marking which among the superstructures are of zero age and have greatest␣
↪damage count (impact)
    rows = X_pca[damage_index]
    for row in rows[:, :2]:
        x,y = row[0] * scalex, row[1] * scaley
        ax.add_patch(Circle((x, y), 0.011, fill=False, color='blue', lw=3))
    # scatter plot of X_pca over all superstructures (mud mortar stone)
    ax.scatter(X_pca[categorical_df['has_superstructure_mud_mortar_stone'] ==␣
↪1,0] * scalex, X_pca[categorical_df['has_superstructure_mud_mortar_stone']␣
↪== 1,1] * scaley, color=new_palette[1],␣
↪label='has_superstructure_mud_mortar_stone', alpha=0.9, s=180, marker='*')
    # scatter plot of X_pca over all superstructures (cement mortar brick)
    ax.scatter(X_pca[categorical_df['has_superstructure_cement_mortar_brick']␣
↪== 1,0] * scalex,␣
↪X_pca[categorical_df['has_superstructure_cement_mortar_brick'] == 1,1] *␣
↪scaley, color=new_palette[5],␣
↪label='has_superstructure_cement_mortar_brick', alpha=0.9, s=180, marker='v')
    # scatter plot of X_pca over all superstructures (bamboo)
    ax.scatter(X_pca[categorical_df['has_superstructure_bamboo'] == 1,0] *␣
↪scalex, X_pca[categorical_df['has_superstructure_bamboo'] == 1,1] * scaley,␣
↪color=new_palette[7], label='has_superstructure_bamboo', alpha=0.9, s=180,␣
↪marker='h')
    # scatter plot of X_pca over all superstructures (rc engineered)
```

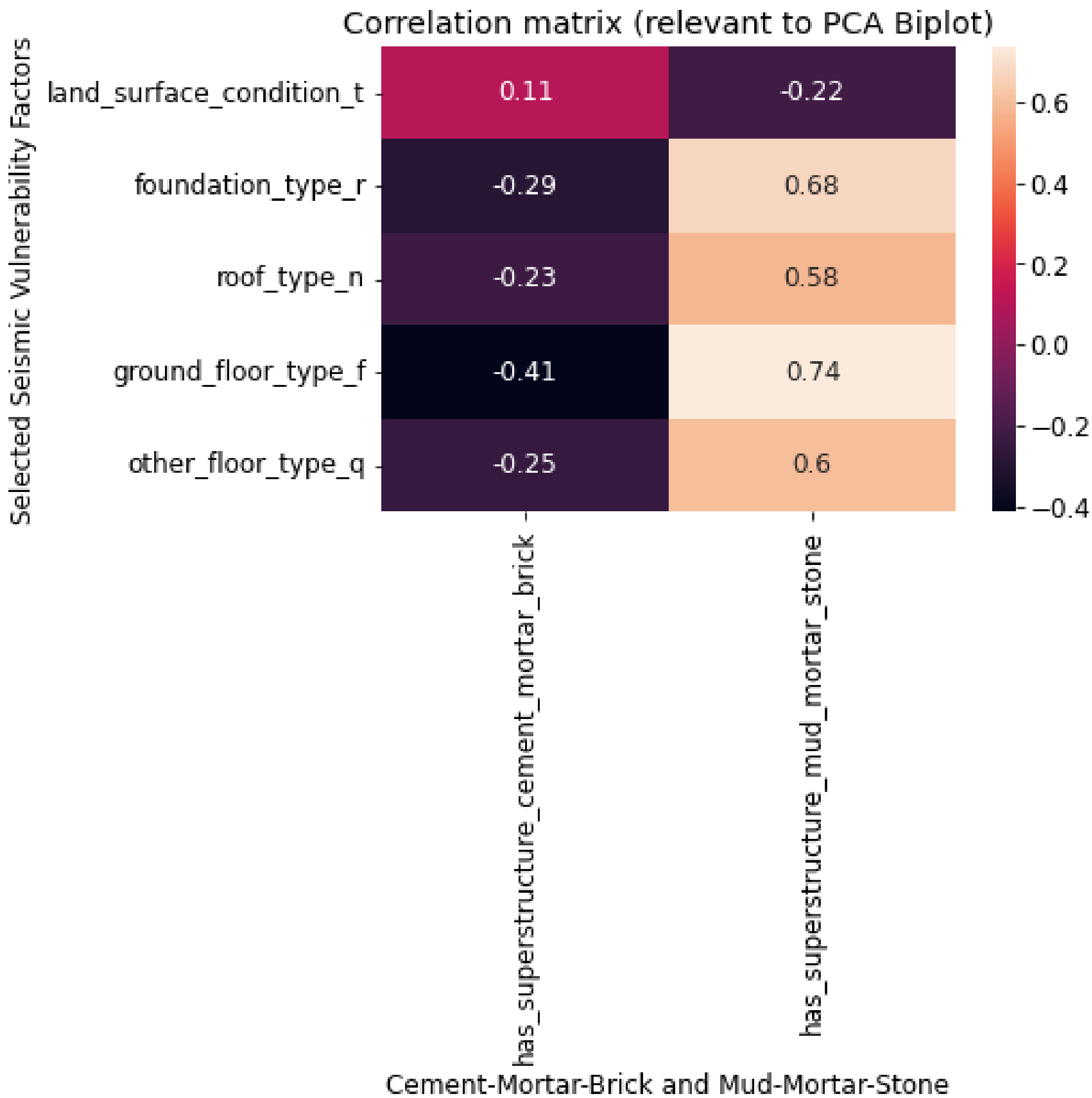
```
    ax.scatter(X_pca[categorical_df['has_superstructure_rc_engineered'] == 1,0]␣
↪* scalex, X_pca[categorical_df['has_superstructure_rc_engineered'] == 1,1] *␣
↪scaley, color=new_palette[9], label='has_superstructure_rc_engineered',␣
↪alpha=0.9, s=180, marker='<')
    # set the legend
    ax.legend(loc='best')
    # set title and labels
    ax.set(xlabel="Principal Component 1", ylabel="Principal Component 2",␣
↪title="PCA Biplot of Modern Buildings with large footprint area (Age <= 40␣
↪and area_percentage >= 40)")

# make a subplot
fig, ax = plt.subplots(1,1, figsize=(14,10))
# scatter plot of Principal Components classified by superstructures
plot_scatter_plot_by_superstructures(X_pca, categorical_df, ax)
# loadings plot with selected vectors - most seismic vulnerability factors
plot_loadings_plot(plt, X_pca, categorical_df, ax,␣
↪eigen_vectors=(36,39,42,11,17))
# super title
fig.suptitle("Figure 4.2 - Modern Buildings marked in Circle with High Area␣
↪(>=80) and Low Age (<=10)")
# show the figure
fig.show()
```

```
C:\Users\burse\AppData\Local\Temp/ipykernel_24064/1444896409.py:39: UserWarning:

Matplotlib is currently using module://matplotlib_inline.backend_inline, which
is a non-GUI backend, so cannot show the figure.
```

Figure 4.2 - Modern Buildings marked in Circle with High Area (>=80) and Low Age (<=10)




```
[58]: # the selected seismic vulnerabilty factors for PCA Biplot
selected_seismic_vulnerability_factors = [
    'land_surface_condition_t', 'foundation_type_r', 'roof_type_n',␣
 ↪'ground_floor_type_f', 'other_floor_type_q'
]
# correlation analysis of only those buildings with superstructures
corr = pd.get_dummies(join_df.loc[superstructure_mask & modern_buildings_mask]).
 ↪loc[:, ['has_superstructure_cement_mortar_brick',
                    'has_superstructure_mud_mortar_stone']
                    + selected_seismic_vulnerability_factors].corr().iloc[2:, :2]
hm = sns.heatmap(corr, annot = True)
hm.set(xlabel='Cement-Mortar-Brick and Mud-Mortar-Stone', ylabel='Selected␣
 ↪Seismic Vulnerability Factors', title = "Correlation matrix (relevant to PCA␣
 ↪Biplot)")
plt.show()
```



Correlation Table supporting PCA Biplot of Superstructures and Selected Seismic Vulnerability Factors:

	has_superstructure_cement_mortar_brick	has_superstructure_mud_mortar_stone
land_surface_condition_t	0.107343	-0.220169
foundation_type_r	-0.294016	0.678974
roof_type_n	-0.233351	0.584432
other_floor_type_q	-0.414669	0.737560
ground_floor_type_f	-0.251541	0.598873

Understanding the Visualization

- The arrows denote the eigen vectors of the PCA Analysis. The angle between arrows imply correlation between the eigen vectors.
- If the arrows are over a particular region, then the eigen vectors are correlated to those points in that region.
- We can see that the LSC (t) and Cement Mortar Brick are more correlated than other superstructures
- PCA Biplot reveals the relationship between the Scatter plot of reduced dimensions and the eigen vectors

7.8.8 Background:

- How the seismic vulnerability factors impacted the collapse of Modern Buildings (with large footprint area) whose construction may have been flawless?
- Based on materials used for construction (superstructures), what can be deduced?

7.8.9 Facts:

- Most frequently occuring Building/land Characteristics are taken for analysis using PCA Biplot.
- Scatter plot of points involving Superstructures and Modern Buildings (with large footprint area) are taken into consideration.
- Buildings marked in Circle indicate High Area (>=80) and Low Age (<=10)

7.8.10 Observations:

- GFT (Ground Floor Type), FT (Foundation Type), RT (Roof Type), OFT (Other Floor Type) are aligned opposite to LSC (Land Surface Condition).
- **Brown coloured points (Triangles)** are representing **Cement Mortar Brick**, and it suggests Cement Mortar Brick is more related to **the Land Surface Condition (t)** as it explains more variance over that region.
- **Orange coloured points (Stars)** represent **Mud Mortar Stone** and the eigen vectors indicate Mud Mortar Stone is more related to **the GFT (f), OFT (q), RT (n) and FT (r)**.
- **Marked Blue Circle Indicators** are more correlated with **GFT (f), OFT (q), RT (n), FT (r), rather than LSC (t)**

7.8.11 4.3 Sub Analysis 2 - Research Question 4

7.8.12 The Collapse of Modern Buildings for Secondary Use due to Seismic Vulnerability

7.8.13 - Building and Land Characteristics for Secondary Use Modern Buildings with Large Footprint Area

[59]:
```
# secondary usage mask for dataframe
secondary_usage_mask = ((join_df['has_secondary_use_use_police'] == 1) |
             (join_df['has_secondary_use_gov_office'] == 1) |
             (join_df['has_secondary_use_institution'] == 1) |
             (join_df['has_secondary_use_health_post'] == 1) |
             (join_df['has_secondary_use_rental'] == 1) |
```

```
            (join_df['has_secondary_use_agriculture'] == 1) |
            (join_df['has_secondary_use_hotel'] == 1) |
            (join_df['has_secondary_use_industry'] == 1) |
            (join_df['has_secondary_use_school'] == 1) |
            (join_df['has_secondary_use_other'] == 1))

# modern buildings with large footprint area mask for dataframe
modern_buildings_mask = (join_df['age'] <= 40) & (join_df['area_percentage'] >=␣
↪40)

# copy the original dataframe
categorical_df = join_df.loc[secondary_usage_mask & modern_buildings_mask].
↪copy()
# one hot encoding of categorical variables
categorical_df = pd.get_dummies(categorical_df.loc[:,␣
↪main_building_land_attributes + sub_building_land_attributes +␣
↪secondary_usage_attributes + ['damage_grade']])
# run principal components analysis
X_pca, pc, evr = principal_components_analysis(categorical_df)
```

[60]:
```
print("Number of Modern Buildings in that region:", len(categorical_df))
```

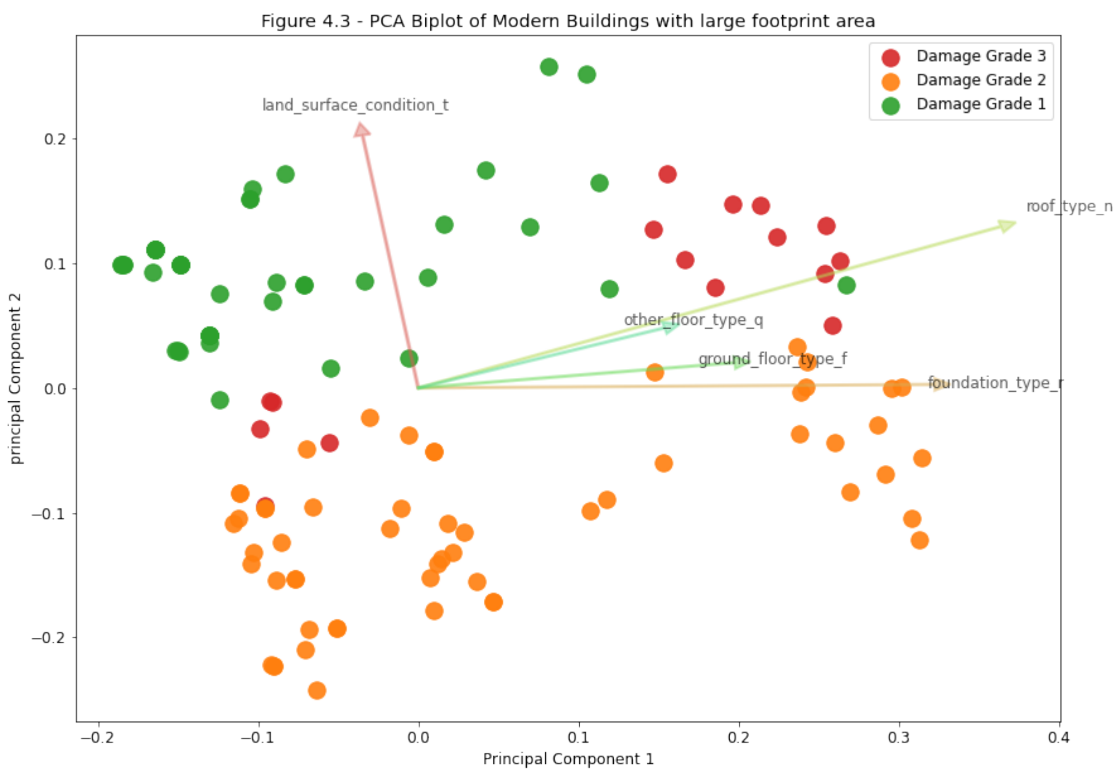
```
Number of Modern Buildings in that region: 122
```

4.3 PCA Biplot of Secondary Use Buildings of only Modern Buildings (with large footprint area)

[61]:
```
def plot_scatter_plot_by_secondary_use(X_pca, categorical_df, ax):
    # scale Principal component 1
    scalex = 0.5 / (X_pca[:,0].max() - X_pca[:,0].min())
    # scale Principal component 2
    scaley = 0.5 / (X_pca[:,1].max() - X_pca[:,1].min())
    # 10 colors color pallette
    new_palette = np.array(sns.color_palette(palette=None, n_colors=11))

    # scatter plot
    ax.scatter(X_pca[categorical_df['damage_grade_3'] == 1,0] * scalex,␣
↪X_pca[categorical_df['damage_grade_3'] == 1,1] * scaley,␣
↪color=new_palette[3], label='Damage Grade 3', alpha=0.9, s=180)
    ax.scatter(X_pca[categorical_df['damage_grade_2'] == 1,0] * scalex,␣
↪X_pca[categorical_df['damage_grade_2'] == 1,1] * scaley,␣
↪color=new_palette[1], label='Damage Grade 2', alpha=0.9, s=180)
    ax.scatter(X_pca[categorical_df['damage_grade_1'] == 1,0] * scalex,␣
↪X_pca[categorical_df['damage_grade_1'] == 1,1] * scaley,␣
↪color=new_palette[2], label='Damage Grade 1', alpha=0.9, s=180)

    # set legend
```

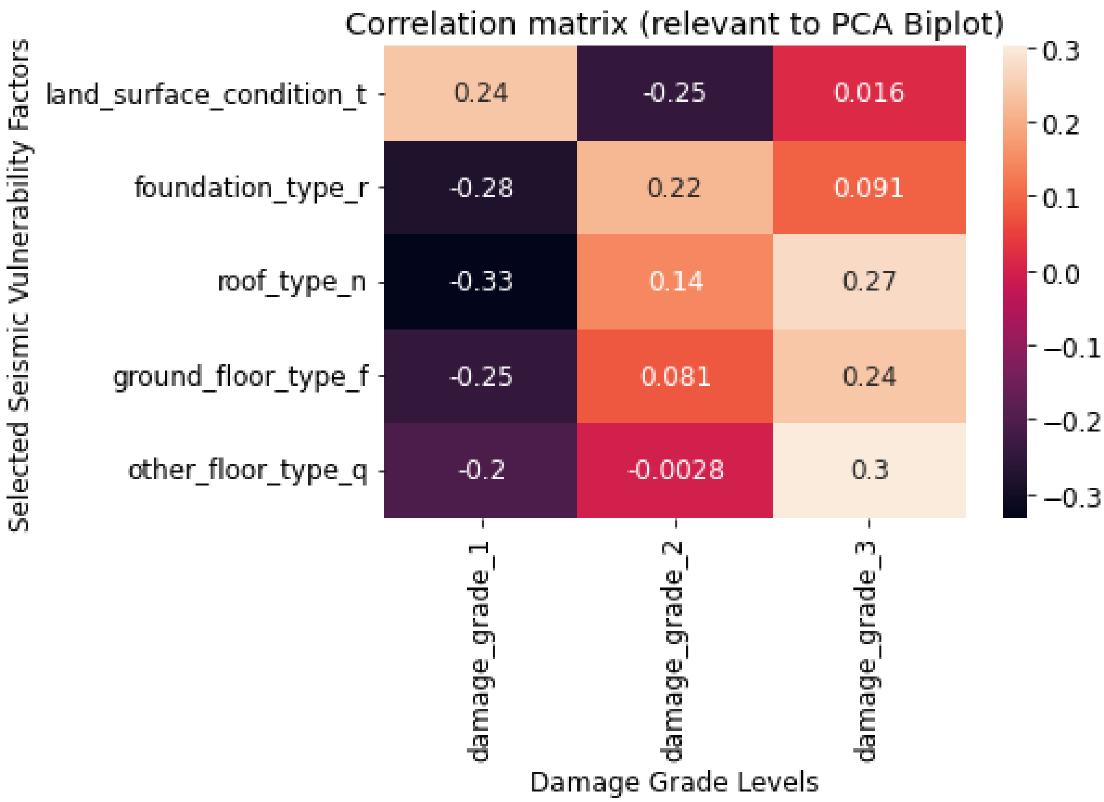
```
    ax.legend(loc='best')
    # set title and labels
    ax.set(xlabel="Principal Component 1", ylabel="principal Component 2",␣
↪title="Figure 4.3 - PCA Biplot of Modern Buildings with large footprint␣
↪area")

fig, ax = plt.subplots(1,1, figsize=(14,10))
plot_scatter_plot_by_secondary_use(X_pca, categorical_df, ax)
plot_loadings_plot(plt, X_pca, categorical_df, ax,␣
↪eigen_vectors=(36,39,42,11,17))
fig.show()
```

```
C:\Users\burse\AppData\Local\Temp/ipykernel_24064/2562615591.py:22: UserWarning:

Matplotlib is currently using module://matplotlib_inline.backend_inline, which
is a non-GUI backend, so cannot show the figure.
```



[62]:
```
# the selected seismic vulnerabilty factors for PCA Biplot
selected_seismic_vulnerability_factors = [
    'land_surface_condition_t', 'foundation_type_r', 'roof_type_n',␣
↪'ground_floor_type_f', 'other_floor_type_q'
```

```
]
# correlation analysis of only those buildings with superstructures
corr = pd.get_dummies(join_df.loc[secondary_usage_mask &␣
 ↪modern_buildings_mask]).loc[:, ['damage_grade_1',
                  'damage_grade_2', 'damage_grade_3']
                  + selected_seismic_vulnerability_factors].corr().iloc[3:, :3]
hm = sns.heatmap(corr, annot = True)
hm.set(xlabel='Damage Grade Levels', ylabel='Selected Seismic Vulnerability␣
 ↪Factors', title = "Correlation matrix (relevant to PCA Biplot)")
plt.show()
```



Correlation Table supporting PCA Biplot of Damage Grade and Selected Seismic Vulnerability Factors

	damage_grade_1	damage_grade_2	damage_grade_3
land_surface_condition_t	0.239057	-0.246834	0.016093
foundation_type_r	-0.282319	0.216711	0.091404
roof_type_n	-0.332262	0.143721	0.271917
ground_floor_type_f	-0.247743	0.081219	0.241045
other_floor_type_q	-0.204143	-0.002824	0.301591

Understanding the Visualization

- The arrows denote the eigen vectors of the PCA Analysis. The angle between arrows imply correlation between the eigen vectors.
- If the arrows are over a particular region, then the eigen vectors are correlated to those points in that region.
- We can see that the LSC (t) and Damage Grade 1 are more correlated than other Damage Grade Levels.
- PCA Biplot reveals the relationship between the Scatter plot of reduced dimensions and the eigen vectors

7.8.14 Background:

- How did the Damage Grade Impact the Seismic Vulnerability Factors for Modern Buildings (with large footprint area)?
- Based on best practices of Secondary Use Buildings, what can be deduced?

7.8.15 Facts:

- Most frequently occuring Building/land Characteristics are taken for analysis using PCA Biplot.
- Scatter plot of points involving Secondary Use and Modern Buildings (with large footprint area) are taken into consideration.

7.8.16 Observations:

- Land Surface Condition (LSC) ‘t’ is mostly related to the Damage Grade 1.
- The other factors - Roof Type (n), Foundation Type (r), Ground Floor Type (f), Other Floor Type (q) are mostly related to Damage Grade 2 and 3 only.

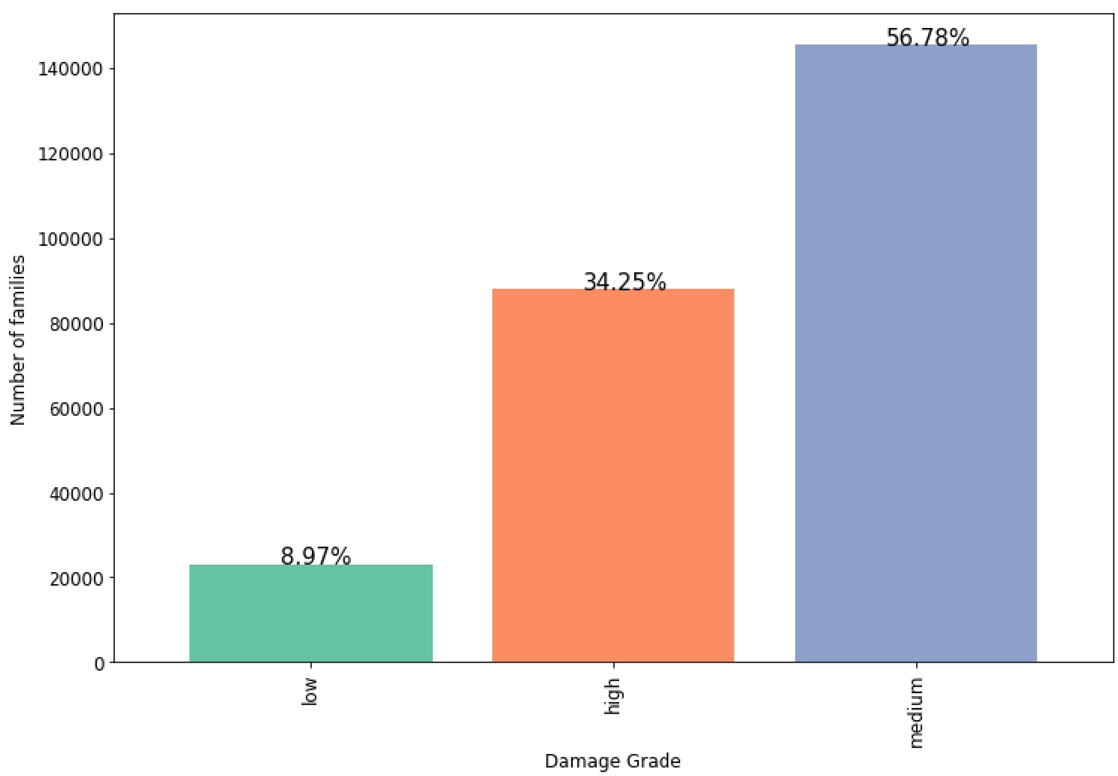
7.9 5.1 Research Question 5

7.9.1 How are families affected due to earthquakes?

[63]:
```
def plot_families_earthquake(colors):
    # copy of original dataset
    temp_df = join_df.copy()
    # map from 1,2,3 to low,medium,high
    temp_df["damage_grade"] = temp_df["damage_grade"].map(
        {1: "low", 2: "medium", 3: "high"}
    )
    # setting figure size
    fig = plt.figure(figsize=(12,8))
    fig.tight_layout(pad=1.0)
    # pandas plotting bar plot
    ax=temp_df.groupby("damage_grade")["count_families"].sum().sort_values().
 ↪plot.bar(color=colors[:3], width=0.8)
    # calculating the height of bars
    totals = []
```

```python
    for i in ax.patches:
        totals.append(i.get_height())
    total = sum(totals)
    # setting the percentage values on top of each bar
    for i in ax.patches:
        # get_x pulls left or right; get_height pushes up or down
        ax.text(i.get_x()+.30, i.get_height(),
                str(round((i.get_height()/total)*100, 2))+'%', fontsize=15,
                color='black')
    # setting title and labels
    fig.suptitle("Figure 5.1 - Families Affected due to earthquake")
    plt.ylabel("Number of families")
    plt.xlabel("Damage Grade")

colors = sns.color_palette("Set2", n_colors=10)
plot_families_earthquake(colors)
```

Figure 5.1 - Families Affected due to earthquake



7.9.2 Visualization

- Visualization is performed by using Pandas data frame

- The X-axis represents the Damage grade
- The Y-axis represents the Number of Families

7.9.3 Facts:

- There are total 256418 families in the given dataset.
- There are 3 damage grade levels.1 represent low damage grade,2 represent medium damage grade and 3 represent high damage grade

7.9.4 Observations:

Out of 256418 familes: - 22991 families are affected by low damage grade level which constitutes of 8.97% - 145593 families are affected by medium damage grade which constitutes of 56.78% - 87834 families are affected by high damage grade which constitutes of 34.25%

7.9.5 Answer to the Research Question:

- Most number of familes are affected my medium damage grade
- Least number of families are affected by low damage grade

7.10 6.1 Research Question 6 (MAIN PLOT)

7.10.1 If a sample is taken from the population, then which Other Floor Type category will show relatively higher Average Height Percentage?

Scatter Plot/Histograms of Other Floor Type (X-axis) and Ground Floor Type (color) with Height Percentage (Y-axis)

[64]:
```
# setup the gridspec 2,2 with one main plot and 2 side plots on x and y axes␣
↪respectively
result = setup_gridspec__one_main__two_side_subplots(plt)
# gridspec
gs = result["gridspec"]
# axis
ax = result["ax"]
# axis on top parallel to x-axis
axx = result["axx"]
# axis on the side parallel to y-axis
axy = result["axy"]
# figure of the plot
fig = result["fig"]

def plot_scatter_bubble_numerical_vs_categorical_bar_hist_grid_no_slice(x_attr,␣
↪y_attr, dimension, xlabel, ylabel, df, ax, ax_histx, ax_histy):
    # define central tendency based aggregation functions
    age_df = join_df.loc[:, [x_attr, y_attr, dimension]].groupby(by=[x_attr,␣
↪y_attr]).mean()
    # get ground floor type from index
    ground_floor_type = age_df.index.get_level_values(0)
    # get OFT from index
```

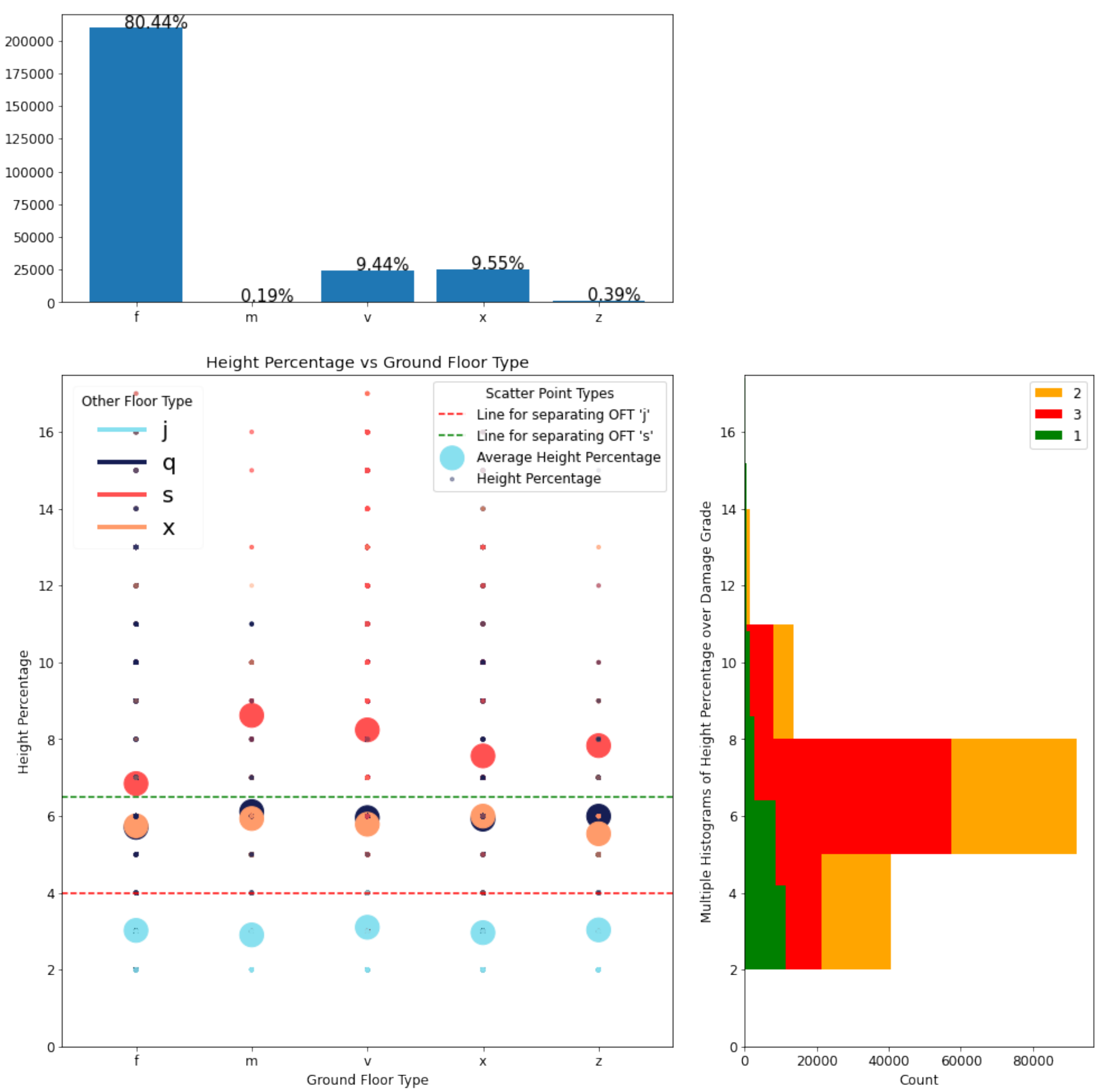
```
    other_floor_type = age_df.index.get_level_values(1)
    # set unique colors as per specification
    unique_colors = ['#88E0EF', '#161E54', '#FF5151', '#FF9B6A']

    assert len(np.unique(ground_floor_type)) == 5, "Test #1 failed"
    assert len(np.unique(other_floor_type)) == 4, "Test #2 Failed"
    assert len(unique_colors) == 4, "Test #3 Failed"
    assert unique_colors == ['#88E0EF', '#161E54', '#FF5151', '#FF9B6A'], "Test␣
↪#4 Failed"

    # set xticks and xtick labels
    ax.set_xticks(range(0,len(np.unique(ground_floor_type))))
    ax.set_xticklabels(np.unique(ground_floor_type))

    # test xticklabels
    assert [t.get_text() for t in ax.get_xticklabels()] == np.
↪unique(ground_floor_type).tolist(), "Test #5 Failed"

    # set colors dictionary
    colors = dict(zip(np.unique(other_floor_type), unique_colors))
    # scatter plot 1 for averaged values
    ax.scatter(x=ground_floor_type, y=age_df[dimension], c=[colors[of] for of␣
↪in other_floor_type], marker='o', s=450, label="Average Height Percentage")
    # scatter plot 2 for actual values
    ax.scatter(x=join_df[x_attr], y=join_df[dimension], c=[colors[of] for of in␣
↪join_df[y_attr].values.tolist()], label="Height Percentage", s=10, alpha=0.4)
    # create custom legend, by creating custom lines
    custom_lines = [Line2D([0], [0], color=colors[of], lw=4) for of in np.
↪unique(other_floor_type)]
    # create legend using custom lines
    legend1 = ax.legend(custom_lines, np.unique(other_floor_type), loc="upper␣
↪left", title="Other Floor Type", framealpha=0.1, fontsize=20)
    # add legend to axis
    ax.add_artist(legend1)
    # set labels and titles
    ax.set(xlabel=xlabel, ylabel=ylabel)
    ax.set_title("{ylabel} vs {xlabel}".format(xlabel=xlabel, ylabel=ylabel))

    # set horizontal lines separating bottom portion
    ax.axhline(y=4, xmin=0, xmax=4, ls='dashed', color='red', label="Line for␣
↪separating OFT 'j'")
    # set horizontal lines separating top portion
    ax.axhline(y=6.5, xmin=0, xmax=4, ls='dashed', color='green', label="Line␣
↪for separating OFT 's'")
    # set the legend
    ax.legend(title="Scatter Point Types")
```

```
    # get value counts of ground floor type
    counter_i = df.loc[:, [x_attr]].value_counts()
    # plot bar plot
    ax_histx.bar(counter_i.index.get_level_values(0), counter_i.values)

    # percentages for value counts
    totals = []
    for i in ax_histx.patches:
        totals.append(i.get_height())
    total = sum(totals)
    # setting the percentage values on top of each bar
    for i in ax_histx.patches:
        # get_x pulls left or right; get_height pushes up or down
        ax_histx.text(i.get_x()+.30, i.get_height(),
                str(round((i.get_height()/total)*100, 2))+'%', fontsize=15,
                color='black')

    # set xticks and xtick labels
    ax_histx.set_xticks(range(0,len(np.unique(ground_floor_type))))
    ax_histx.set_xticklabels(np.unique(ground_floor_type))

    # test for xticklabels
    assert [t.get_text() for t in ax_histx.get_xticklabels()] == np.
 ↪unique(ground_floor_type).tolist(), "Test #6 Failed"

    # get histograms for damage grade 1,2,3
    hist_y1 = join_df.loc[join_df['damage_grade'] == 1][dimension]
    hist_y2 = join_df.loc[join_df['damage_grade'] == 2][dimension]
    hist_y3 = join_df.loc[join_df['damage_grade'] == 3][dimension]
    # plot histograms for 1,2,3 respectively
    ax_histy.hist(hist_y2, orientation='horizontal', label='2', color='orange')
    ax_histy.hist(hist_y3, orientation='horizontal', label='3', color='red')
    ax_histy.hist(hist_y1, orientation='horizontal', label='1', color='green')
    # set legend for side subplot (y-axis)
    ax_histy.legend()
    # set labels
    ax_histy.set(xlabel='Count', ylabel="Multiple Histograms of {ylabel} over␣
 ↪Damage Grade".format(ylabel=ylabel))
    ax_histy.set_ylim((0,17.5))

    # set super title for figure
    fig.suptitle("Figure 6.1 - Average Height Percentage vs Ground Floor Type␣
 ↪Distinguished by Other Floor Type (color) and Quantity of values (size)")

plot_scatter_bubble_numerical_vs_categorical_bar_hist_grid_no_slice('ground_floor_type',␣
 ↪'other_floor_type', 'height_percentage',
```

```
                                                                    'Ground␣
↪Floor Type', 'Height Percentage', join_df, ax=ax, ax_histx=axx, ax_histy=axy)

fig.show()
```

```
C:\Users\burse\AppData\Local\Temp/ipykernel_24064/2449125110.py:103:
UserWarning:

Matplotlib is currently using module://matplotlib_inline.backend_inline, which
is a non-GUI backend, so cannot show the figure.
```

Figure 6.1 - Average Height Percentage vs Ground Floor Type Distinguished by Other Floor Type (color) and Quantity of values (size)



7.10.2 Background

- In an earthquake-affected site, if a building inspector visits the site, then can he establish if the sample of population he has taken will have on an average higher height percentage for OFT 's' compared to OFT 'j'
- The impact of Height Percentage on Tower-like Buildings has been established in Figure 6.3
- There is a pattern between OFT and GFT as it is known by the irregularity of buildings' design.
- Exploration of OFT and GFT vs Height Percentage is a criteria to conclude on different floor types that have undergone damage.

7.10.3 Facts:

- The scatter plot of height vs Ground Floor Type is plotted
- Additionally, the average height is plotted with a larger size dimension
- The side histograms indicate the distribution of Height Percentage over Damage Grade
- The top bar chart (histogram) shows frequency of occurence of GFT types in the dataset
- The colors indicate the different OFT types
- GFT (f) occurs 80.44% of times in the dataset, followed by 9.55% for GFT (x), 9.44% for GFT (v), 0.39% for GFT (z) and 0.19% for GFT (m)

7.10.4 Observations:

- As per Figure 6.1, The Average Height percentage for population is higher for OFT 's' (in fact highest) than OFT 'j' (which is lowest)
- The distribution of Ground Floor Type has been discussed in most commonly occuring Seismic Vulnerability Factors, which says GFT 'f' is the most frequent
- The Histogram of Height Percentage is majorly over **2 to 10 Height Percentage** which is also seen in the Scatter plot between Height Percentage and Count of Floors
- There is a clear line separating OFT 'j', OFT 's' and other OFT typesThere is a clear line separating OFT 'j', OFT 's' and other OFT types
- OFT 's' and OFT 'j' show an average height percentage as higher and lower set of values respectively for each Ground Floor Type (GFT)
- There is a **pattern for such irregularity in the designs** because it can be established that **All buildings** have **higher Average Height Percentage** for 's' compared to 'j'.
- As per Figure 6.3, on the contrary, **Tower-like buildings** have **lower Average height percentage** for 's' compared to 'j'.
- This fact leads to the pattern of irregularity in the buildings' design which may be due to design criteria or bad practices of designs.

7.10.5 Answer to the Research Question:

- **OFT 's' has higher Average Height Percentage**, **OFT 'j' has lower Average Height Percentage**.
- A statistical test in Student's T-test has been shown to prove that there is no significant difference between mean of the population and the mean of the sample and the null hypothesis is true.
- A building inspector who comes to the site **cannot easily establish the OFT 'j' as the lowest or OFT 's' as the highest height** for buildings taken on an average even though

on an average they are separated as per the lines.

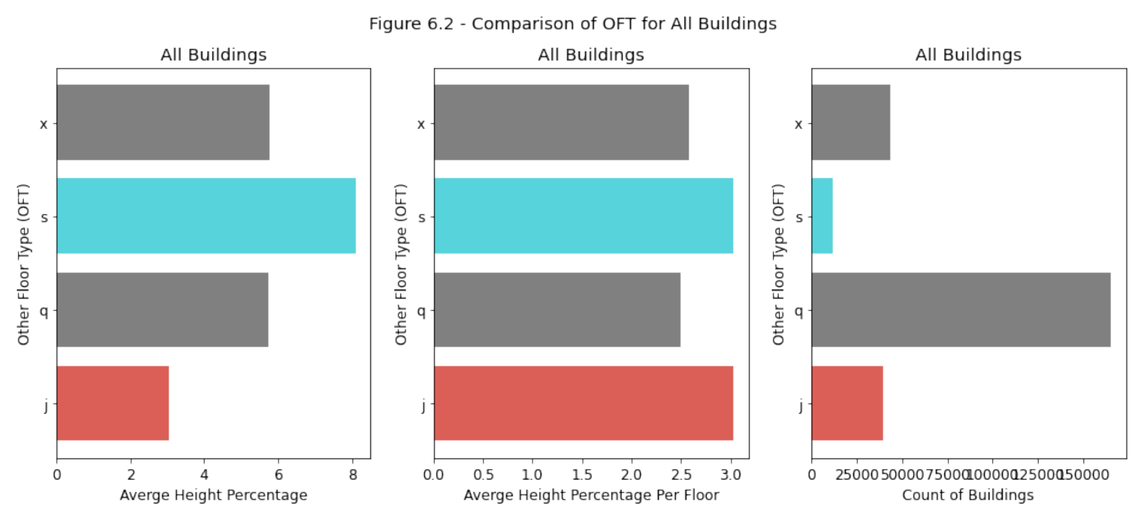
- The building inspector **has to check the side histograms of Average Height Percentage** over Damage Grade 2 and Damage Grade 3 and if the height falls **within the histograms (2 to 10 % Height)**, then he may be able to establish the OFT for that building given Tower-like buildings are excluded.

7.11 6.2 Sub-Visualizations of RQ6

7.11.1 Analysis on Effect of Height with 'j': Bar Chart of Height with 'j' OFT and Damage Grade

[65]:
```
# plot the average height against OFT
def plot_average_height_against_OFT(ax, colors):
    # set the data
    data = join_df.loc[:, ['other_floor_type', 'height_percentage']].
 ↪groupby('other_floor_type', as_index=False).mean()
    # plot bar horizontal
    ax.barh(data.other_floor_type, data.height_percentage, color=[colors[0],␣
 ↪'gray', colors[2], 'gray'])
    # set labels
    ax.set_xlabel("Averge Height Percentage")
    ax.set_ylabel("Other Floor Type (OFT)")
    # set title
    ax.set_title("All Buildings")

# plot the Average Height Percentage Per Floor Against OFT
def plot_average_height_per_floor_against_OFT(ax, colors):
    # set the data
    data = join_df.loc[:, ['other_floor_type', 'height_percentage',␣
 ↪'count_floors_pre_eq']]
    # set a new height per floor attribute
    data['height_per_floor'] = data['height_percentage'] /␣
 ↪data['count_floors_pre_eq']
    data = data[["other_floor_type", "height_per_floor"]].
 ↪groupby('other_floor_type', as_index=False).mean()
    # set bar horizontal plot
    ax.barh(data.other_floor_type, data.height_per_floor, color=[colors[0],␣
 ↪'gray', colors[2], 'gray'])
    # set labels
    ax.set_xlabel("Averge Height Percentage Per Floor")
    ax.set_ylabel("Other Floor Type (OFT)")
    # set title
    ax.set_title("All Buildings")

# plot the count against OFT
def plot_count_against_OFT(ax, colors):
    # set the data
```

```python
    data = join_df.loc[:, ['other_floor_type', 'damage_grade']].groupby('other_floor_type', as_index=False).count()
    # horizontal bar plot
    ax.barh(data.other_floor_type, data.damage_grade, color=[colors[0], 'gray', colors[2], 'gray'])
    # set labels
    ax.set_xlabel("Count of Buildings")
    ax.set_ylabel("Other Floor Type (OFT)")
    # set title
    ax.set_title("All Buildings")

# make a subplot
fig, ax = plt.subplots(1, 3, figsize=(16,6))
# set colors
colors = sns.color_palette("hls", 4)
# set super title
fig.suptitle("Figure 6.2 - Comparison of OFT for All Buildings")
# call plot function
plot_average_height_against_OFT(ax[0], colors)
# call plot function
plot_average_height_per_floor_against_OFT(ax[1], colors)
# call plot function
plot_count_against_OFT(ax[2], colors)
```



7.11.2 Background:

- How does Height per Floor and Height make a difference in the collapse of buildings?

7.11.3 Facts:

- Some floors are built for specific purpose.

- Let us take Height into perspective.
- According to the Literature Review, Greater the Height, greater are the vibrations. Hence a floor that tends to vibrates upon earthquake must be constructed with lower height to reduce vibrations on walls.
- Floor 'j' has lesser height when compared to others on an average for **All buildings**. (Please refer to Figure 6.1)

7.11.4 Observations:

- The 'j' OFT has the least average height percentage and the 's' OFT has highest average height percentage
- The Average Height Percentage Per Floor for 'j' OFT has increased to the level of 's' OFT.
- The 'j' OFT has higher Damage Impact than the 's' OFT.
- Could characteristics of 'j' have led to damage on an overall perspective? Let us consider the collapse of Towers?

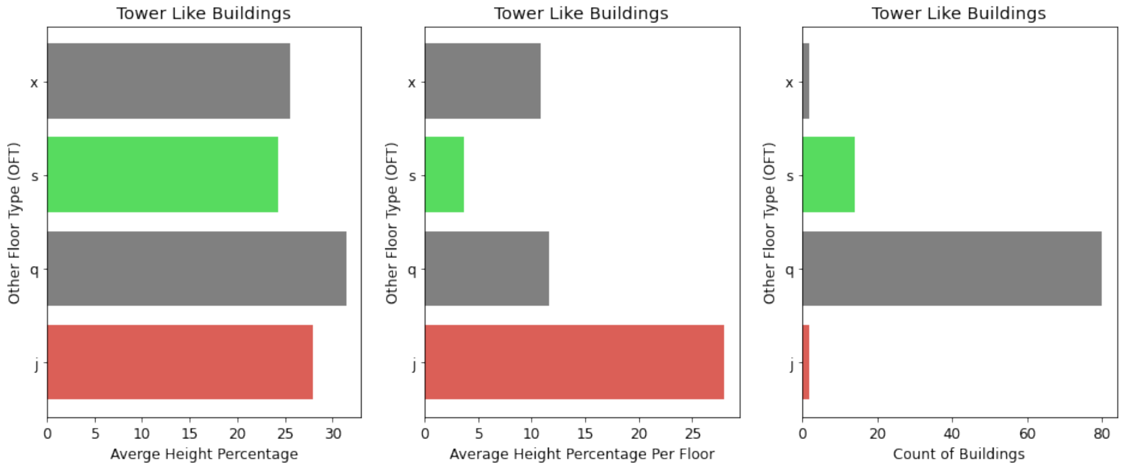
7.12 6.3 Sub-Visualizations of RQ6

7.12.1 Analysis on Effect of Height with 'j': Bar Chart of Height with OFT and Damage Grade for (Tower-like Buildings)

[66]:
```
# mask for tower like buildings
tower_like_mask = (join_df['height_percentage'] >= 23)

# plot OFT vs Average Height Per Floor
def plot_OFT_Average_Height_Per_Floor_dominant(ax, colors):
    # set the data
    data = join_df.loc[tower_like_mask, ['other_floor_type',␣
 ↪'height_percentage', 'count_floors_pre_eq']]
    # set height per floor additional attribute for visualization
    data['height_per_floor'] = data['height_percentage'] /␣
 ↪data['count_floors_pre_eq']
    # group by other floor type
    data = data[["other_floor_type", "height_per_floor"]].
 ↪groupby('other_floor_type', as_index=False).mean().
 ↪sort_values(by=['other_floor_type'])
    # plot bar horizontal with OFT vs Height Per Floor
    ax.barh(data.other_floor_type, data.height_per_floor, color=[colors[0],␣
 ↪'gray', colors[2], 'gray'])
    # set labels
    ax.set_xlabel("Average Height Percentage Per Floor")
    ax.set_ylabel("Other Floor Type (OFT)")
    # set title
    ax.set_title("Tower Like Buildings")

# plot OFT vs Average
def plot_OFT_Average_Dominant(ax, colors):
    # set the data
```

```
    data = join_df.loc[tower_like_mask, ['other_floor_type',␣
↪'height_percentage']].groupby('other_floor_type', as_index=False).mean().
↪sort_values(by=['other_floor_type'])
    # plot bar horizontal of OFT vs Height
    ax.barh(data.other_floor_type, data.height_percentage, color=[colors[0],␣
↪'gray', colors[2], 'gray'])
    # set labels
    ax.set_xlabel("Averge Height Percentage")
    ax.set_ylabel("Other Floor Type (OFT)")
    # set title
    ax.set_title("Tower Like Buildings")

# plot the count against OFT
def plot_OFT_Count_dominant(ax, colors):
    # set the data
    data = join_df.loc[tower_like_mask, ['other_floor_type', 'damage_grade']].
↪groupby('other_floor_type', as_index=False).count()
    # plot horizontal bar plot of OFT vs Damage Grade
    ax.barh(data.other_floor_type, data.damage_grade, color=[colors[0], 'gray',␣
↪colors[2], 'gray'])
    # set labels
    ax.set_xlabel("Count of Buildings")
    ax.set_ylabel("Other Floor Type (OFT)")
    # set title
    ax.set_title("Tower Like Buildings")

# make a subplot
fig, ax = plt.subplots(1,3,figsize=(16,6))
# set the colors
colors = sns.color_palette("hls", 6)
# set the super title
fig.suptitle("Figure 6.3 - Comparison of OFT for Tower-like Buildings")
# plot OFT vs Average
plot_OFT_Average_Dominant(ax[0], colors)
# plot OFT vs Average Height per Floor
plot_OFT_Average_Height_Per_Floor_dominant(ax[1], colors)
# plot OFT vs Damage Grade Count
plot_OFT_Count_dominant(ax[2], colors)
```

Figure 6.3 - Comparison of OFT for Tower-like Buildings



7.12.2 Background:

- The **Tower-like buildings** are those with average height percentage greater or equal to 23.

7.12.3 Facts:

- 's' OFT has a lower Average Height Percentage than 'j' OFT.
- 'j' OFT has an increased Height Per Floor than 's' OFT.
- When the **Tower-like Buildings** are taken into consideration, 'j' OFT has **very less amount of buildings** damaged.

7.12.4 Observations:

- **Arguably, it could be said that 'j' contributed to the collapse of buildings due to its Height Per Floor on an overall perspective.**
- **On the contrary, It could also be said that the damage impact is too low to conclude on 'j' OFT's involvement with Tower-like Buildings.**

7.13 6.4 Sub Answer of RQ6

7.13.1 Establishing the Mean of Sample picked is same as Mean of Population

[67]:
```
# function to conduct the t-test
def analyze_alpha(averages, null_hypothesis_mean=0.0):
    """
    Perform a t-test with the null hypothesis being that the expected mean␣
 ↪return is zero.

    Parameters
    ----------
```

```
    Returns
    -------
    t_value
        T-statistic from t-test
    p_value
        Corresponding p-value
    """
    # is it a two-tailed distribution
    two_tailed = False
    # one-sided when tow-tailed is False
    one_sided = True if two_tailed == False else False
    # mode 1 or mode 2 (no of tails)
    mode = 1 if two_tailed else 2

    # scipy.stats t-test with null hypothesis mean
    t_value, p_value = stats.ttest_1samp(averages, null_hypothesis_mean)

    # t-value and p-value divided by mode
    return t_value, p_value / mode

def evaluate_gft_oft_t_test_inference_by_simulation():
    p_s = []
    # 10 iterations
    for i in range(10):
        averages = []
        # 100 simulations of samples
        for i in range(100):
            # 100 buildings in a particular geographical region
            sample_df = join_df.sample(100)
            # check if rows with GFT == f and OFT == j exist in the sample of␣
 ↪100 rows
            if len(sample_df.loc[(sample_df['ground_floor_type'] == 'f') &
                                 (sample_df['other_floor_type'] == 'j')]):
                # append averages
                averages.append(sample_df.loc[:, ['ground_floor_type',␣
 ↪'other_floor_type', 'height_percentage']]
                                .groupby(['ground_floor_type',␣
 ↪'other_floor_type']).mean()
                            .loc[('f', 'j')].height_percentage)
        # calculate net average
        net_average = \
        join_df.loc[:, ['ground_floor_type', 'other_floor_type',␣
 ↪'height_percentage']]\
    .groupby(['ground_floor_type', 'other_floor_type']).mean().loc[('f', 'j')].
 ↪height_percentage
        # conduct the t-test for 100 simulations of averages with 100 sample␣
 ↪data set rows
```

```python
        t, p = analyze_alpha(averages, null_hypothesis_mean=net_average)
        # append the probability
        p_s.append(p)
        # print the t-test and p-value conducted for each iteration
        print("t_test value = ", t, " p_value = ", p)
    # print Average p-value over 10 iterations
    print("Average p-value: ", np.mean(p_s))

# call the function to evaluate t-test
evaluate_gft_oft_t_test_inference_by_simulation()
```

```
t_test value =  -0.8439197937241771  p_value =  0.2003744162994993
t_test value =  -1.3297867128519507  p_value =  0.09332216331951876
t_test value =  0.1911504014526544  p_value =  0.42439961675292515
t_test value =  1.2456691931148942  p_value =  0.10791295040886673
t_test value =  0.4385595491765615  p_value =  0.3309681347294618
t_test value =  1.2163531882849246  p_value =  0.11337126327382266
t_test value =  -1.0404230423644365  p_value =  0.15033882935253182
t_test value =  -1.1238008056516626  p_value =  0.13190759875522895
t_test value =  0.14533081023235264  p_value =  0.4423725855341905
t_test value =  0.23169646767592922  p_value =  0.4086260042067036
Average p-value:  0.24035935626327493
```

7.13.2 Background:

- Inference by simulation is required to answer the Research Question
- A significance value of <= 0.05 will imply that there is significant difference between mean of population and mean of sample

7.13.3 Facts:

- A sample of 100 buildings are taken at a single simulation.
- The averages of 100 samples over 100 simulations are recorded and compared against the null hypothesis mean.
- 10 iteratons of such simulations are shown in this t-test simulation

7.13.4 Observations:

- All the p-value of every t-test simulation exceeds, p >= 0.05
- The average p-value of all simulations is about **0.20 to 0.30**

7.13.5 Answer to Research Question:

- There is no significance obtained in comparing the mean of population and mean of sample by difference of means using Student T-test
- This implies the sample mean may be the same as the population mean.
- **The building inspector can investigate the building based on the statistics results on Average Height Percentage and suggest if the mean is similar to the population mean**

7.14 7.1 Research Question 7 (MAIN PLOT)

7.14.1 If a sample of RC Engineered Superstructures is taken from the population, on an average for the Foundation Type 'u', will Age be relatively higher compared to other Foundation Types?

Plot of Age and Damage Grade vs Foundation Type Sliced by RC Engineered Superstructures Distinguished by Amount of Buildings Damaged

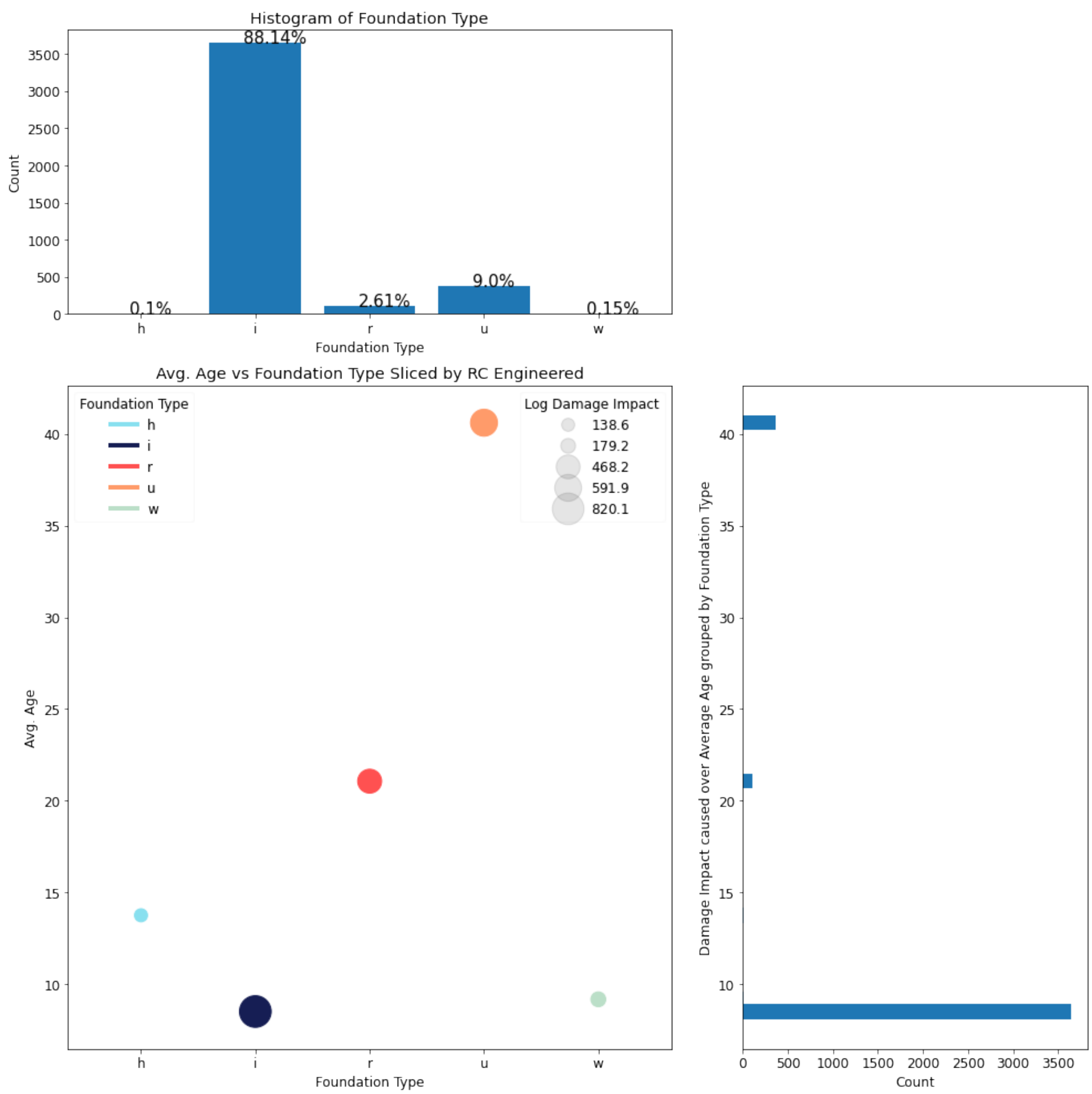
[68]:
```
# setup the gridspec 2,2 with one main plot and 2 side plots on x and y axes␣
↪respectively
result = setup_gridspec__one_main__two_side_subplots(plt)
# gridspec
gs = result["gridspec"]
# axis
ax = result["ax"]
# axis on top parallel to x-axis
axx = result["axx"]
# axis on the side parallel to y-axis
axy = result["axy"]
# figure of the plot
fig = result["fig"]

def plot_scatter_bubble_numerical_vs_categorical_bar_bar_sliced(sizes_tuple,␣
↪xlabel, ylabel, sliced_by, slice_idx,
                                                                            ␣
↪categorical_dimension, numerical_dimension, df, ax, ax_histx, ax_histy):
    # unique colors used in the graph
    unique_colors = ['#88E0EF', '#161E54', '#FF5151', '#FF9B6A', '#BBDFC8']
    # count of categorical
    var_df = df.loc[slice_idx, [numerical_dimension, categorical_dimension]].
↪groupby(by=[categorical_dimension], as_index=False).count()
    # mean of numerical dimension over categorical
    age_df = df.loc[slice_idx, [numerical_dimension, categorical_dimension]].
↪groupby(by=[categorical_dimension], as_index=False).mean()
    # set index to categorical dimension
    age_df.index = age_df[categorical_dimension]

    assert len(unique_colors) == 5, "Test #1 Failed"
    assert len(var_df[categorical_dimension]) == 5, "Test #2 Failed"
    assert len(age_df[categorical_dimension]) == 5, "Test #3 Failed"

    # set xticks and xtick labels
    ax.set_xticks(range(0,len(age_df[categorical_dimension])))
    ax.set_xticklabels(age_df[categorical_dimension].values)

    # tests for xticklabels
    assert [t.get_text() for t in ax.get_xticklabels()] ==␣
↪age_df[categorical_dimension].values.tolist(), "Test #4 Failed"
```

```
    # value counts for categorical dimension
    counter_i = df.loc[slice_idx, [categorical_dimension]].value_counts().
↪sort_index(ascending=True)
    colors = dict(zip(age_df.index.values, unique_colors))
    # scatter plot for Numerical vs Categorical
    scatter = ax.scatter(age_df.index.values, age_df[numerical_dimension],␣
↪c=[colors[ft] for ft in colors],␣
↪s=sizes_tuple[0](var_df[numerical_dimension])*sizes_tuple[1])
    # set labels and title
    ax.set(xlabel=xlabel, ylabel=ylabel)
    ax.set_title("{ylabel} vs {xlabel} Sliced by {sliced_by}".
↪format(xlabel=xlabel, ylabel=ylabel, sliced_by=sliced_by))
    # add custom legend
    custom_lines = [Line2D([0], [0], color=colors[dim], lw=4) for dim in age_df.
↪index.values]
    legend1 = ax.legend(custom_lines, age_df.index.values, loc="upper left",␣
↪title="Foundation Type", framealpha=0.1)
    ax.add_artist(legend1)
    # add legend for log size
    handles, labels = scatter.legend_elements(prop="sizes", alpha=0.1)
    legend2 = ax.legend(handles, labels, loc="upper right", title="Log Damage␣
↪Impact", framealpha=0.1)
    # plot bar plot of value counts (top bar plot)
    ax_histx.bar(counter_i.index.get_level_values(0), counter_i.values)
    ax_histx.set(xlabel=xlabel, ylabel='Count')
    ax_histx.set_title("Histogram of {xlabel}".format(xlabel=xlabel))

    # percentages for value counts
    totals = []
    for i in ax_histx.patches:
        totals.append(i.get_height())
    total = sum(totals)
    # setting the percentage values on top of each bar
    for i in ax_histx.patches:
        # get_x pulls left or right; get_height pushes up or down
        ax_histx.text(i.get_x()+.30, i.get_height(),
               str(round((i.get_height()/total)*100, 2))+'%', fontsize=15,
               color='black')

    # set xticks and xtick labels
    ax_histx.set_xticks(range(0,len(age_df[categorical_dimension])))
    ax_histx.set_xticklabels(age_df[categorical_dimension].values)

    # tests for xticklabels
```

```
    assert [t.get_text() for t in ax_histx.get_xticklabels()] ==␣
↪age_df[categorical_dimension].values.tolist(), "Test #5 Failed"

    # plot bar horizontal plot of numerical and count (side bar plot)
    ax_histy.barh(age_df[numerical_dimension], var_df[numerical_dimension])
    ax_histy.set(xlabel='Count', ylabel="Damage Impact caused over Average Age␣
↪grouped by Foundation Type")

    # set super title
    fig.suptitle("Figure 7.1 - Age vs Foundation Type Sliced by RC Engineered␣
↪Superstructures Distinguished by Amount of Buildings Damaged", y=0.95)

# categorical_dimension = 'foundation_type', numerical_dimension = 'age', Age␣
↪vs FT plot
plot_scatter_bubble_numerical_vs_categorical_bar_bar_sliced(sizes_tuple=(np.
↪log, 1e2), xlabel='Foundation Type', ylabel='Avg. Age',
                                                             sliced_by='RC␣
↪Engineered', slice_idx=join_df['has_superstructure_rc_engineered'] == 1,
                                                            ␣
↪categorical_dimension='foundation_type', numerical_dimension='age',␣
↪df=join_df, ax=ax, ax_histx=axx,
                                                             ax_histy=axy)

fig.show()
```

```
C:\Users\burse\AppData\Local\Temp/ipykernel_24064/211334328.py:88: UserWarning:

Matplotlib is currently using module://matplotlib_inline.backend_inline, which
is a non-GUI backend, so cannot show the figure.
```

Figure 7.1 - Age vs Foundation Type Sliced by RC Engineered Superstructures Distinguished by Amount of Buildings Damaged



7.14.2 Background:

- In an earthquake-affected site, if a building inspector visits the site, then can he establish that the Average Age of RC Engineered Buildings will be higher for the Foundation Type (u) compared to other Foundation types.
- The relation between Height Percentage and Foundation Types have been established.

7.14.3 Facts:

- The Average Age for RC Engineered Superstructures is high for buildings with FT (u).
- For the dataset sliced by Superstructures, 88.14% of times FT (i) occurs in the slice, followed by FT (u) 9%, then FT (r) 2.61%, FT (w) 0.15% and FT (h) 0.1%

7.14.4 Observations:

- In the Scatter plot several buildings of FT 'u' deteriorated due to age (about 60 / 327 Modern Buildings)
- FT (u) is definitely indicative of Age and deterioration due to age.

7.14.5 Answer to Research Question:

- The answer to research question using Scatter plot (Sub Visualization of RQ7) indicates the **Average Age for FT (u) is higher than FT (i)** for 327 Modern Buildings
- The research question RQ7 - Main Plot also indicates that the **Average Age for FT (u) is higher than any other FT types**, but by how much is not known.
- A building inspector visiting the site can establish that: if the superstructure is **RC Engineered** and then the **Age is high (exactly 40 and above)**, with **45.1113 % to 45.4547 % certainty** the FT will be 'u'. (Please see the calculation in 7.3 Sub Answer of RQ7 - below)

7.15 7.2 Sub Visualizations of RQ7

7.15.1 Modern/Medieval/Ancient Buildings - Bar chart of Average Age vs Height for different Foundation Types

[69]:
```
def plot_buildings_histogram_foundation_type(ax1, ax2, colors, kind='medieval'):
    if kind == 'medieval':
        # medieval buildings mask
        buildings_mask = (join_df['age'] >= 100) & (join_df['age'] <= 200) &␣
↪(join_df['area_percentage'] >= 0) & (join_df['area_percentage'] <= 45)
        # set title and labels
        ax1.set(ylabel="Foundation Type", xlabel="Count", title="Medieval␣
↪Buildings")
        ax2.set(ylabel="Foundation Type", xlabel="Average Age", title="Medieval␣
↪Buildings")
    elif kind == 'modern':
        # modern buildings with large footprint area mask for dataframe
        buildings_mask = (join_df['age'] <= 40) & (join_df['area_percentage']␣
↪>= 40)
        # set title and labels
        ax1.set(ylabel="Foundation Type", xlabel="Count", title="Modern␣
↪Buildings with Large Footprint Area")
        ax2.set(ylabel="Foundation Type", xlabel="Average Age", title="Modern␣
↪Buildings with Large Footprint Area")
    elif kind == 'ancient':
        # ancient buildings
        buildings_mask = (join_df['age'] == 995)
        # set title and labels
        ax1.set(ylabel="Foundation Type", xlabel="Count", title="Ancient␣
↪Buildings")
```

```
        ax2.set(ylabel="Foundation Type", xlabel="Average Age", title="Ancient␣
↪Buildings")

    # count of buildings
    buildings_count = join_df.loc[buildings_mask, ['age', 'foundation_type']].
↪groupby('foundation_type', as_index=False).count()
    # average age of buildings
    buildings_age = join_df.loc[buildings_mask, ['age', 'foundation_type']].
↪groupby('foundation_type', as_index=False).mean()
    # average height of buildings
    buildings_height = join_df.loc[buildings_mask, ['height_percentage',␣
↪'foundation_type']].groupby('foundation_type', as_index=False).mean()
    # plot bar plot of foundation type and count of buildings
    ax1.barh(buildings_count.foundation_type, buildings_count.age, height=0.8,␣
↪color=colors)
    # plot bar plot of foundation type and average age
    ax2.barh(buildings_age.foundation_type, buildings_age.age, height=0.8,␣
↪color=colors)
    # inver axes for modern buildings
    if kind == 'modern':
        ax1.invert_xaxis()
        ax2.invert_xaxis()
        ax1.invert_yaxis()
        ax2.invert_yaxis()
    else:
        ax1.invert_yaxis()
        ax2.invert_yaxis()

# make a subplot
fig, ax = plt.subplots(2, 3, figsize=(16,9))
# tight layout
fig.tight_layout(pad=5.0)
# set the colors
colors = sns.color_palette("crest", 5)
# set the super title
fig.suptitle("Figure 7.2 - Modern (age <= 40 and area_percentage >= 40) /␣
↪Medieval (100 <= age <= 200) / Ancient Buildings (age == 995)")
# plot histogram (value counts) and average age for medieval / modern and␣
↪ancient buildings
plot_buildings_histogram_foundation_type(ax[0,1], ax[1,1], colors,␣
↪kind='medieval')
plot_buildings_histogram_foundation_type(ax[0,0], ax[1,0], colors,␣
↪kind='modern')
plot_buildings_histogram_foundation_type(ax[0,2], ax[1,2], colors,␣
↪kind='ancient')
```

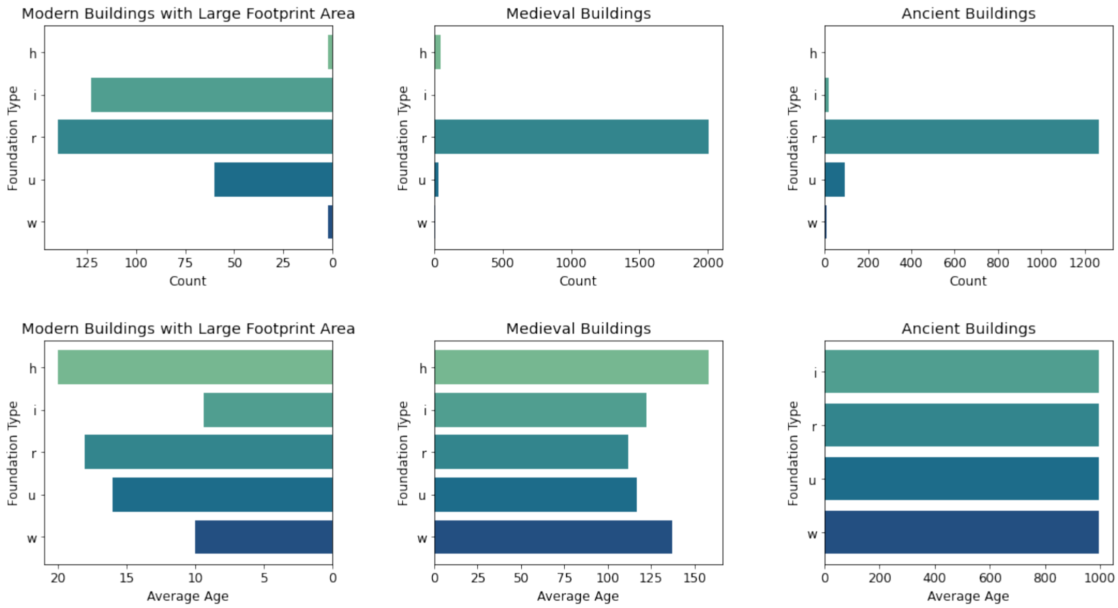


Figure 7.2 - Modern (age <= 40 and area_percentage >= 40) / Medieval (100 <= age <= 200) / Ancient Buildings (age == 995)

7.15.2 Background:

- The Age distribution as shown in RQ3 has a lot of buildings between 10 and 20 whereas the extreme values lie at 995 years old.
- The Scatter plot in RQ4 differentiates between Ancient, Medieval and Modern Buildings
- Modern (age range)
- medieval (age range)

7.15.3 Observations:

- Considering most frequently occuring factor overall ‘r’ is the highest, and contribution of ‘u’ (overall) is very small.
- In this case (Modern Buildings), ‘u’ shows a significant hike indicating damage may be due to ‘u’ FT.
- For Medieval and Ancient Buildings, the ‘r’ Foundation Type domainates in Count
- For Modern and Medieval Buildings, the average Age is higher for ‘h’ Foundation Type

7.16 7.3 Sub Answer of RQ7

7.16.1 Establishing the certainty with which the Average Age will be high (40 and above) for Foundation Type (u)

Categorical Variable coming from a Multinomial Distribution **Full Dataset:** All 260601 records of data from which the Standard deviation is calculated

Superstructures (Actual): All Superstructure records inside the dataset

Sample Data (When the Age is High, >= 40): Our sample taken from superstructures when Age is 40 and above

Standard deviation of Population (Based on Population Proportion):

$\sqrt{p(1-p)}$

```
- p is the probability of occurence of that categorical variable
```

Standard Error of Sample (when Age is High), Based on Sample Proportion (ref: Estimated Standard Deviation):

$\sqrt{\frac{p(1-p)}{n_{superstructures.and.age>=40}}}$

Confidence Interval of Sample:

$Count +/- 1.96 * Std.Error.of.Sample$

	Full Dataset	Superstructures (Actual)	Sample (Age is High, >= 40)
Number of entries	260601	4133	53
Foundation Type (u)	14260 / 260601	372 / 4133	24 / 53
FT (u) Probability	0.05472	0.09001	0.45283
Standard Deviation	0.2274		
Standard Error		0.00353769	0.0464178
Confidence Interval		372 + / - 0.828011579821664	24 + / - 0.0909789
Certainty			**45.1113 % to 45.4547 %**

The certainty with which the Building Inspector can say the Average Age is High is: 45 %

7.16.2 Answer to Research Question:

- When the average age is high (>= 40), then the building inspector can suggest that about 45% of times the assumption on higher average age for FT (u) is correct for RC Engineered Superstructures.

7.17 8.1 Research Question 8

7.17.1 If a sample of hotels are taken from the population, what Foundation Type will have a relatively higher Average Area Percentage?

Plot of Area Percentage and Damage Grade vs Foundation Type Sliced by Hotel Distinguished by Amount of Buildings Damaged

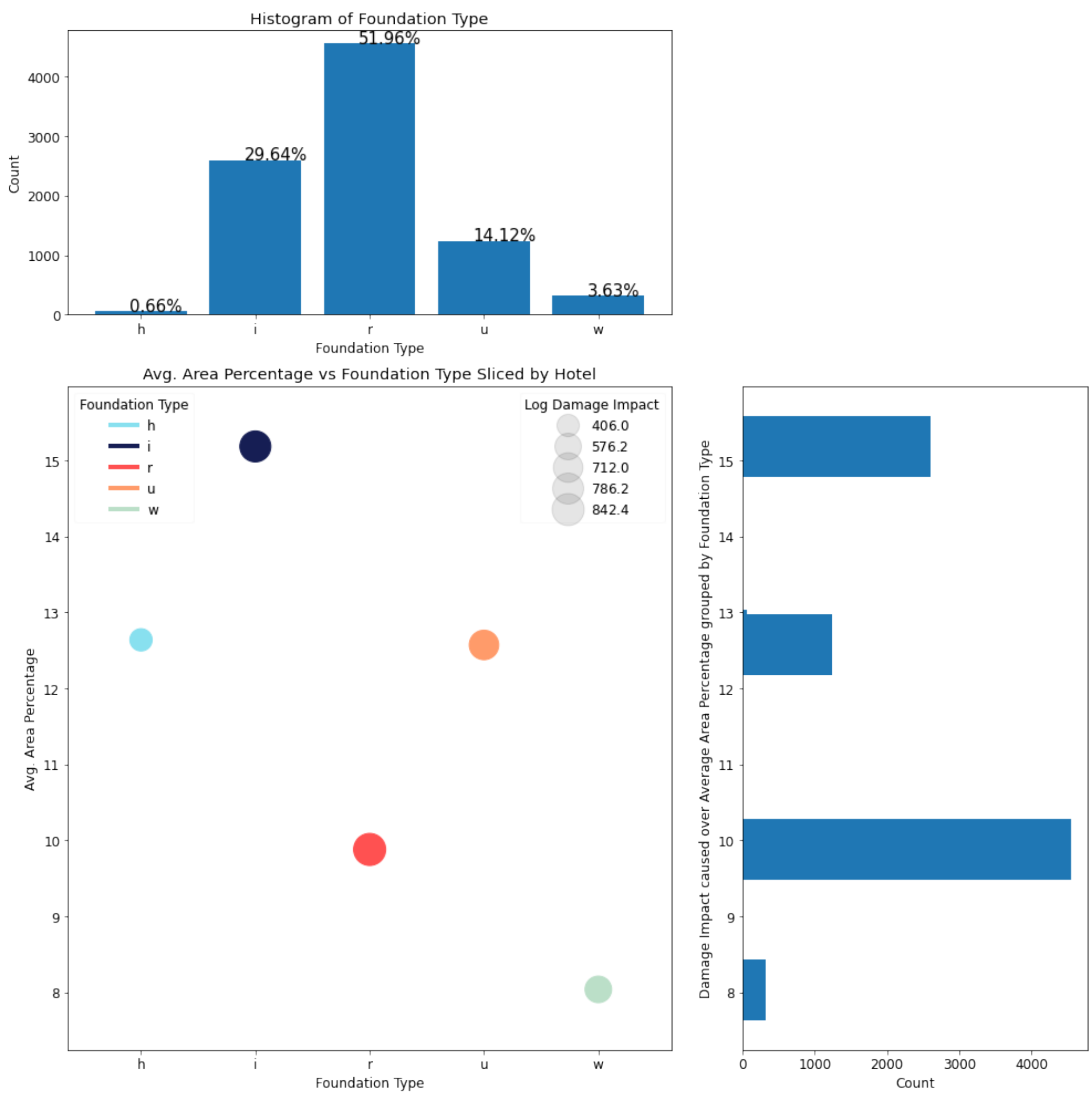
[70]:
```
# setup the gridspec 2,2 with one main plot and 2 side plots on x and y axes␣
↪respectively
result = setup_gridspec__one_main__two_side_subplots(plt)
# gridspec
gs = result["gridspec"]
# axis
ax = result["ax"]
# axis on top parallel to x-axis
axx = result["axx"]
# axis on the side parallel to y-axis
```

```
axy = result["axy"]
# figure of the plot
fig = result["fig"]

# plot scatter / bubble plot of numerical vs categorical with bar graphs on the␣
↪sides
def plot_scatter_bubble_numerical_vs_categorical_bar_hist_sliced(sizes_tuple,␣
↪xlabel, ylabel, sliced_by, slice_idx,                                        ␣
↪categorical_dimension, numerical_dimension, df, ax, ax_histx, ax_histy):
    # set unique colors
    unique_colors = ['#88E0EF', '#161E54', '#FF5151', '#FF9B6A', '#BBDFC8']
    var_df = df.loc[slice_idx, [numerical_dimension, categorical_dimension]].
↪groupby(by=[categorical_dimension], as_index=False).count()
    age_df = df.loc[slice_idx, [numerical_dimension, categorical_dimension]].
↪groupby(by=[categorical_dimension], as_index=False).mean()
    age_df.index = age_df[categorical_dimension]

    # test against the unique colors
    assert len(unique_colors) == 5, "Test #1 Failed"
    # test length of unique categorical dimension
    assert len(var_df[categorical_dimension]) == 5, "Test #2 Failed"
    # test age_df dataframe for unique categorical dimension
    assert len(age_df[categorical_dimension]) == 5, "Test #3 Failed"

    # set xticks and xtick labels
    ax.set_xticks(range(0,len(age_df[categorical_dimension])))
    ax.set_xticklabels(age_df[categorical_dimension].values)

    # test for xtick labels
    assert [t.get_text() for t in ax.get_xticklabels()] ==␣
↪age_df[categorical_dimension].values.tolist(), "Test #4 Failed"

    # get value counts
    counter_i = df.loc[slice_idx, [categorical_dimension]].value_counts().
↪sort_index(ascending=True)
    # set colors by age index
    colors = dict(zip(age_df.index.values, unique_colors))
    # scatter plot of categorical dimension and numerical dimension
    scatter = ax.scatter(age_df.index.values, age_df[numerical_dimension],␣
↪c=[colors[ft] for ft in colors],␣
↪s=sizes_tuple[0](var_df[numerical_dimension])*sizes_tuple[1])
    # set labels
    ax.set(xlabel=xlabel, ylabel=ylabel)
    # set title
```

```
    ax.set_title("{ylabel} vs {xlabel} Sliced by {sliced_by}".
↪format(xlabel=xlabel, ylabel=ylabel, sliced_by=sliced_by))
    # add custom legend
    custom_lines = [Line2D([0], [0], color=colors[dim], lw=4) for dim in age_df.
↪index.values]
    legend1 = ax.legend(custom_lines, age_df.index.values, loc="upper left",␣
↪title="Foundation Type", framealpha=0.1)
    ax.add_artist(legend1)
    # add size legend
    handles, labels = scatter.legend_elements(prop="sizes", alpha=0.1)
    legend2 = ax.legend(handles, labels, loc="upper right", title="Log Damage␣
↪Impact", framealpha=0.1)
    # plot bar plot of value counts
    ax_histx.bar(counter_i.index.get_level_values(0), counter_i.values)
    # set labels and title
    ax_histx.set(xlabel=xlabel, ylabel='Count')
    ax_histx.set_title("Histogram of {xlabel}".format(xlabel=xlabel))

    # percentages for value counts
    totals = []
    for i in ax_histx.patches:
        totals.append(i.get_height())
    total = sum(totals)
    # setting the percentage values on top of each bar
    for i in ax_histx.patches:
        # get_x pulls left or right; get_height pushes up or down
        ax_histx.text(i.get_x()+.30, i.get_height(),
                str(round((i.get_height()/total)*100, 2))+'%', fontsize=15,
                color='black')

    # set xticks and xtick labels
    ax_histx.set_xticks(range(0,len(age_df[categorical_dimension])))
    ax_histx.set_xticklabels(age_df[categorical_dimension].values)

    # test for xticklabels
    assert [t.get_text() for t in ax_histx.get_xticklabels()] ==␣
↪age_df[categorical_dimension].values.tolist(), "Test #5 Failed"

    # plot bar plot horizontal with count plot for numerical dimension
    ax_histy.barh(age_df[numerical_dimension], var_df[numerical_dimension])
    ax_histy.set(xlabel='Count', ylabel="Damage Impact caused over Average Area␣
↪Percentage grouped by Foundation Type")

    # set super title
    fig.suptitle("Figure 8.1 - Area Percentage vs Foundation Type Sliced by␣
↪Hotels Distinguished by Amount of Buildings Damaged", y=0.95)
```

```
# call the plot function
plot_scatter_bubble_numerical_vs_categorical_bar_hist_sliced(sizes_tuple=(np.
 ↪log, 1e2), xlabel='Foundation Type', ylabel='Avg. Area Percentage',
                                                              sliced_by='Hotel',␣
 ↪slice_idx=join_df['has_secondary_use_hotel'] == 1,
                                                             ␣
 ↪categorical_dimension='foundation_type',␣
 ↪numerical_dimension='area_percentage', df=join_df, ax=ax, ax_histx=axx,
                                                              ax_histy=axy)
```

Figure 8.1 - Area Percentage vs Foundation Type Sliced by Hotels Distinguished by Amount of Buildings Damaged



7.17.2 Background:

- In an earthquake-affected site, if a building inspector inspects the site, and found out that the footprint area (area percentage) is high for the collapsed building, then can he establish whether the Foundation Type will be 'i'
- The relation between Count of Families and Area Percentage has been established.

7.17.3 Facts:

- Hotel Buildings are found to have greater amount of buildings for 'r' FT compared to 'i' FT which can be observed from the top histogram bar plot of Foundation Types
- Large Hotel Buildings and Small Hotel Buildings have been taken for analysis in the Histograms
- The distribution of 'i' over Small and Large are unknown, hence a Histogram is used to show the difference
- In the Hotels data (sliced by Hotels), about 51.96% of times FT (r) occurs in the slice, followed by 29.64% for FT (i), 14.12% for FT (u), 3.63% for FT (w) and 0.66% for FT (h)

7.17.4 Observations:

- The Histogram reveals that 'i' FT is seen largely in Large Hotel Buildings and it is because of that the overall average is highest in above Scatter / Bubble Plot
- The small value for 'w' in the side histogram is not visible due to a scale, and that has been adjusted as Log Damage Impact in the Scatter / Bubble plot above

7.17.5 Answer to Research Question:

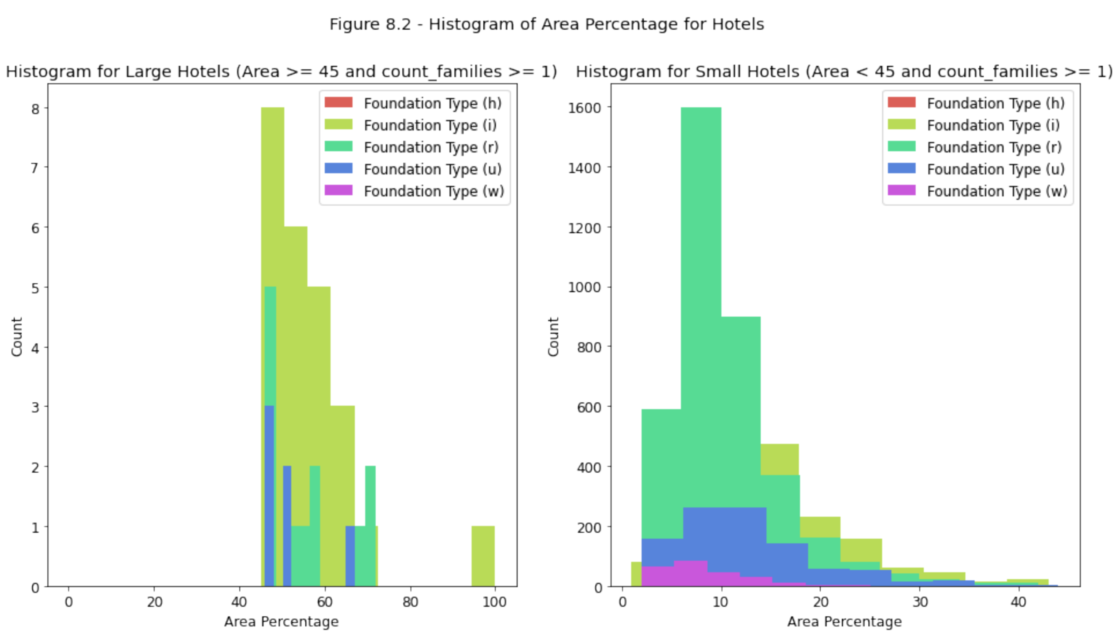
- **The building inspector** will be able to assume the mean of the sample is same as the mean of the population, by inference by simulation
- **The building inspector** would be able to say that the Average height of FT (i) will be high with **42 - 44% certainty**

7.18 8.2 Sub Visualization of RQ8

7.18.1 Histogram of Area Percentage for Large Hotels and Small Hotels

[71]:
```python
# Plot Histogram of Large Hotels and Small Hotels over Foundation Types
def plot_hotels_foundation_types_average_age(ax1, colors, kind='large_hotels'):
    if kind == 'large_hotels':
        hotels_mask = (join_df['has_secondary_use_hotel'] == 1) &␣
↪(join_df['area_percentage'] >= 45) & (join_df['count_families'] >= 1)
        ax1.set(xlabel="Area Percentage", ylabel="Count", title="Histogram for␣
↪Large Hotels (Area >= 45 and count_families >= 1)")
    elif kind == 'small_hotels':
        hotels_mask = (join_df['has_secondary_use_hotel'] == 1) &␣
↪(join_df['area_percentage'] < 45) & (join_df['count_families'] >= 1)
        ax1.set(xlabel="Area Percentage", ylabel="Count", title="Histogram for␣
↪Small Hotels (Area < 45 and count_families >= 1)")
    # large hotels mask or small hotels mask with the dataframe
```

```
    hotels = join_df.loc[hotels_mask, ['area_percentage', 'foundation_type']]
    # histogram of 'h'
    ax1.hist(hotels[hotels['foundation_type'] == 'h'].area_percentage,␣
↪color=colors[0], label="Foundation Type (h)")
    # histogram of 'i'
    ax1.hist(hotels[hotels['foundation_type'] == 'i'].area_percentage,␣
↪color=colors[1], label="Foundation Type (i)")
    # histogram of 'r'
    ax1.hist(hotels[hotels['foundation_type'] == 'r'].area_percentage,␣
↪color=colors[2], label="Foundation Type (r)")
    # histogram of 'u'
    ax1.hist(hotels[hotels['foundation_type'] == 'u'].area_percentage,␣
↪color=colors[3], label="Foundation Type (u)")
    # histogram of 'w'
    ax1.hist(hotels[hotels['foundation_type'] == 'w'].area_percentage,␣
↪color=colors[4], label="Foundation Type (w)")
    # set the legend
    ax1.legend()

# make a subplot
fig, ax = plt.subplots(1,2,figsize=(16,8))
# set colors
colors = sns.color_palette("hls", 5)
# set super title
fig.suptitle("Figure 8.2 - Histogram of Area Percentage for Hotels")
# plot in first axis
plot_hotels_foundation_types_average_age(ax[0], colors, kind='large_hotels')
# plot in second axis
plot_hotels_foundation_types_average_age(ax[1], colors, kind='small_hotels')
```



7.18.2 Background:

- Large Hotel Buildings and Small Hotel Buildings have varying Foundation Types.
- This will give insights into the distribution of Area Percentage over Foundation Types

7.18.3 Facts:

- For Large Hotels, 'i' has a more taller distribution with greater Area Percentage
- For Small Hotels, 'r' has a more larger distribution with lesser Area Percentage compared to Large Hotels

7.18.4 Observations:

- 'i' shows more overall average area percentage because Large Hotels are mostly constructed with 'i' Foundation Type
- 'r' shows lesser overall average area percentage because only Small Hotels dominate with 'r' Foundation Type
- As per Figure 8.1, the 'i' FT has contributed to the greater Area because of Large Hotels.
- Even though 'r' FT is dominating in Small Hotel Buildings, the influence of Height on the Average is less when compared to 'i' FT for Large Hotels.

7.19 8.3 Sub Answer of RQ8

7.19.1 Calculating the Certainty with which Average Area Percentage of FT (i) will be high

Categorical Variable coming from a Multinomial Distribution **Full Dataset:** All 260601 records of data from which the Standard deviation is calculated

Hotels (Actual): All Hotel records inside the dataset

Sample Data (When the Area Percentage is High, >= 16): Our sample taken from hotels when Area is 16 and above

Standard deviation:

$\sqrt{p(1-p)}$

`- p is the probability of occurence of that categorical variable`

Standard Error of Hotels (Sample Proportion) or ref: Estimated Standard Deviation:

$\sqrt{\frac{p(1-p)}{n_{hotels.with.area>=16}}}$

Confidence Interval of Sample:

$Count + / - 1.96 * Std.Error.of.Sample$

	Full Dataset	**Hotels (Actual)**	**Sample (Area Percentage is High)**
Number of entries	260601	8763	1872

	Full Dataset	Hotels (Actual)	Sample (Area Percentage is High)
Foundation Type (i)	10579 / 260601	2597 / 4133	972 / 1872
FT (i) Probability	0.040595	0.628357	0.519231
Standard Deviation	0.19735		
Standard Error		0.0021082	0.0045613
Confidence Interval		2597 + / - 0.0041321	972 + / - 0.00894
Certainty			**51.9231%**

7.19.2 The certainty with which the Building Inspector can say the Average Area for FT (i) is High (>= 40) is: 51.9 %

7.19.3 Answer to Research Question:

- When the average area percentage is high (>= 16), then the building inspector can suggest that about 51.9% of times the assumption on higher average area percentage for FT (i) is correct for Hotel Buildings.

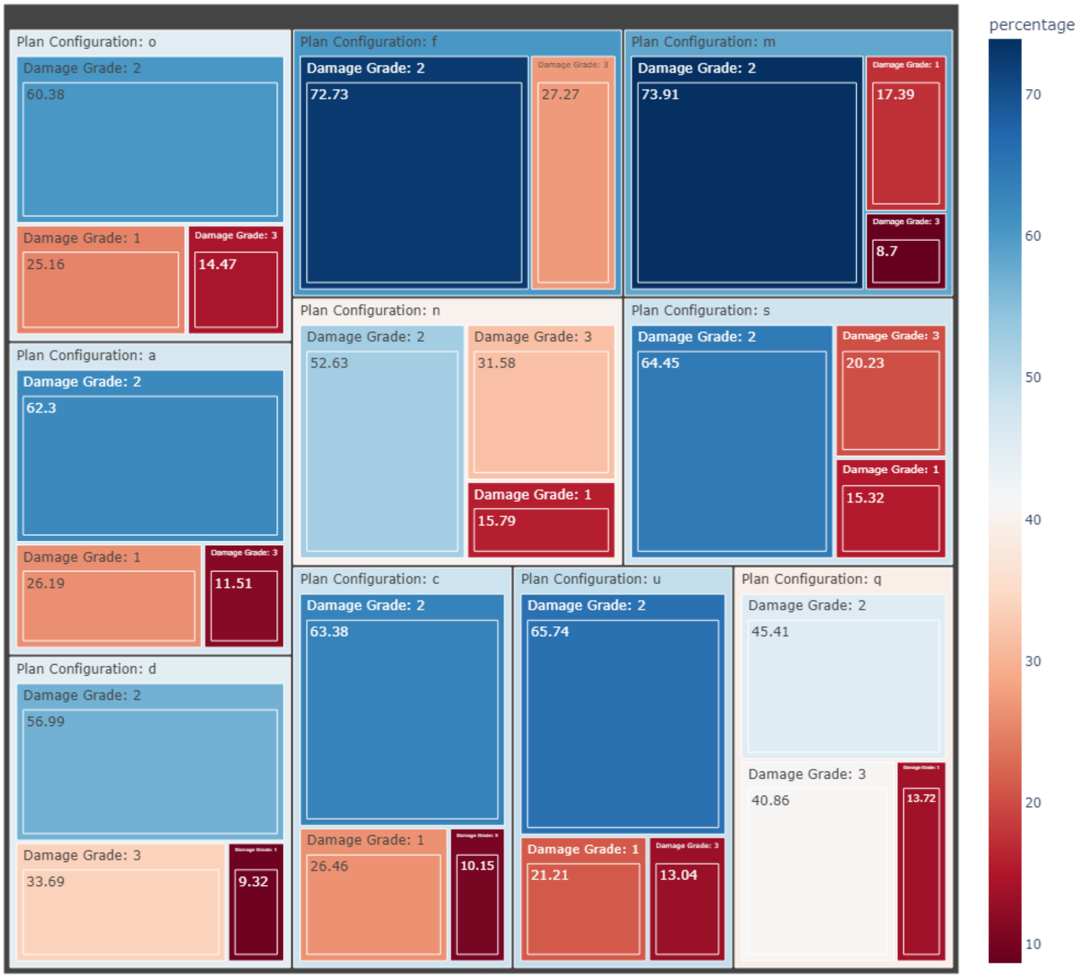
7.20 9.1 Research Question 9

7.20.1 What is the Damage Grade Distribution for each Plan Configuration?

[72]:
```
def preprocess_plan_configuration_vs_damage_grade():
    # groupby plan_configuration and damage_grade
    plan_configuration = join_df.groupby(['plan_configuration','damage_grade']).
↪size().reset_index(name='percentage')
    # set indices
    plan_configuration = plan_configuration.set_index(['plan_configuration',␣
↪'damage_grade'])
    # apply percentages for calculating within groups damage grade %
    plan_configuration = plan_configuration.groupby(level=0).apply(lambda x:␣
↪round(100 * x / float(x.sum()),2)).reset_index()
    # converting to string type
    plan_configuration = plan_configuration.astype({'plan_configuration':␣
↪'string', 'damage_grade': 'string'})
    # renaming the columns
    plan_configuration['plan_configuration'] = "Plan Configuration: " +␣
↪plan_configuration['plan_configuration']
    # renaming the damage grde column
    plan_configuration['damage_grade'] = "Damage Grade: " +␣
↪plan_configuration['damage_grade']
    # dropping zeros from the plan_configuration dataframe (happened due to␣
↪category type conversion)
    plan_configuration.
↪drop(labels=plan_configuration[plan_configuration['percentage'] == 0].index,␣
↪inplace=True)

    return plan_configuration
```

```
def plot_plan_configuration_vs_damage_grade(plan_configuration):
    # plot a treemap plot
    fig = px.treemap(plan_configuration,␣
↪path=['plan_configuration','damage_grade', 'percentage'],␣
↪values='percentage',
                 color='percentage', hover_data=['damage_grade'],
                 color_continuous_scale='RdBu')
    # create title text, update layout
    fig.update_layout(margin = dict(t=50, l=25, r=25, b=25), title_text='Figure␣
↪9.1 - Damage Grade Distribution for each Plan Configuration')
    # obtaining image bytes using kaleido
    img_bytes = pio.to_image(fig, format="png", engine="kaleido", width=1024,␣
↪height=960)
    return img_bytes

# call function to get plan configuration
plan_configuration = preprocess_plan_configuration_vs_damage_grade()
# display the Bytes Image
display(Image(plot_plan_configuration_vs_damage_grade(plan_configuration)))
```

Figure 9.1 - Damage Grade Distribution for each Plan Configuration



7.20.2 Visualization:

- Visualization is performed using plotly
- Treemap is used to show the plan configurations and damage grade

7.20.3 Facts:

- There are 10 types of plan configurations in the dataset.
- There are 3 types of damage grade levels in the dataset.
- The plan configurations are ‘a’,‘c’,‘d’,‘f’,‘m’,‘n’,‘o’,‘q’,‘s’,‘u’.
- The damage grades are 1,2,3. 1 represents low damage,2 represents medium damage grade and 3 represent high damage grade.

7.20.4 Observations:

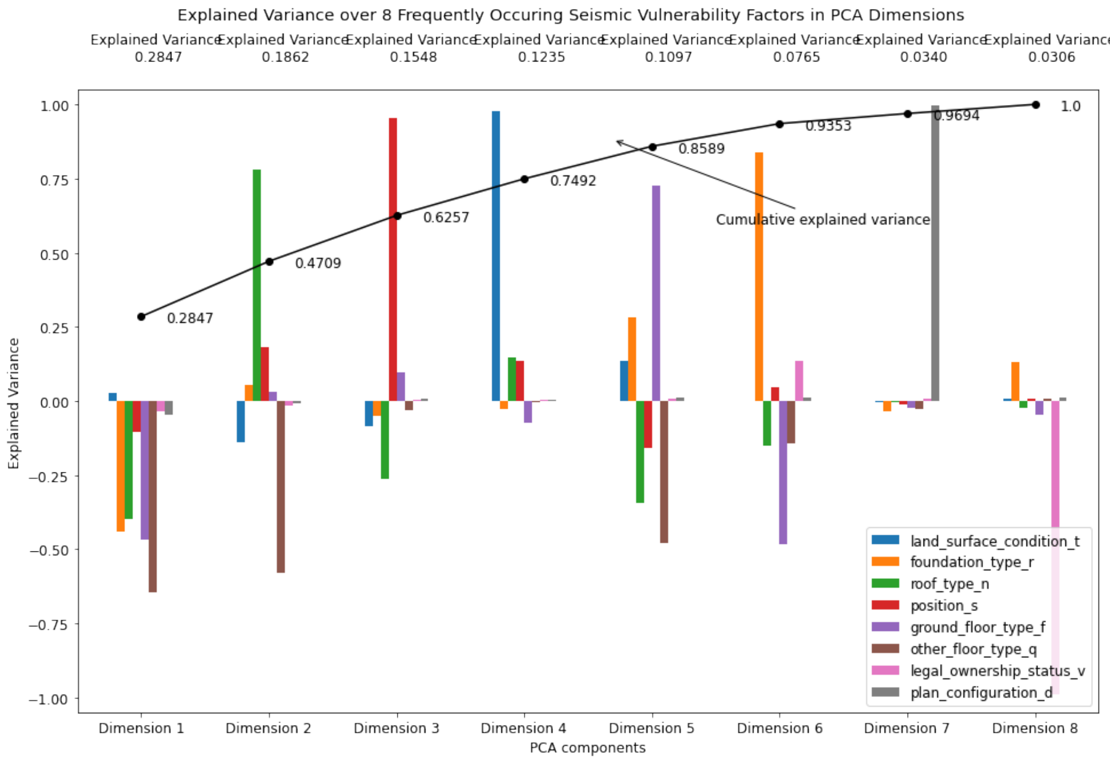
- All the plan configurations have damage grade level 2 as the highest percentage
- Plan configuration 'f' buildings doesnot have damage grade level 1
- Plan configurations 'm','c','f','o','a','u' have lowest percentage in damage grade level 3
- Plan configurations 'q','d','n','s' have lowest percentage in damage grade level 1
- Out of all plan configurations 'm' has highest percentage(74) in damage grade level 2
- Out of all plan configurations 'm' has lowest percentage(9) in damage grade level 3

7.21 PCA Components Plot

[73]:
```
def pca_results(data, pc, evr):
    '''
    Create a DataFrame of the PCA results
    Includes dimension feature weights and explained variance
    Visualizes the PCA results
    '''

    # Dimension indexing
    dimensions = dimensions = ['Dimension {}'.format(i) for i in␣
↪range(1,len(pc)+1)]

    # PCA components
    components = pd.DataFrame(np.round(pc, 4), columns = data.keys())
    components.index = dimensions

    # PCA explained variance
    ratios = evr.reshape(len(pc), 1)
    variance_ratios = pd.DataFrame(np.round(ratios, 4), columns = ['Explained␣
↪Variance'])
    variance_ratios.index = dimensions

    # Create a bar plot visualization
    fig, ax = plt.subplots(figsize = (16,10))

    fig.suptitle("Explained Variance over 8 Frequently Occuring Seismic␣
↪Vulnerability Factors in PCA Dimensions")

    # Plot the feature weights as a function of the components
    components.plot(ax = ax, kind = 'bar');
    ax.set_ylabel("Feature Weights")
    ax.set_xticklabels(dimensions, rotation=0)

    # Display the explained variance ratios
    for i, ev in enumerate(evr):
```

```
        ax.text(i-0.40, ax.get_ylim()[1] + 0.05, "Explained Variance\n
 ↪%.4f"%(ev))

    # Return a concatenated DataFrame
    return pd.concat([variance_ratios, components], axis = 1)
```

[74]:
```
most_seismic_vulnerability_factors = [
    'land_surface_condition_t', 'foundation_type_r', 'roof_type_n',
 ↪'position_s',
    'ground_floor_type_f', 'other_floor_type_q', 'legal_ownership_status_v',
 ↪'plan_configuration_d'
]
def plot_pca_components_explained_variance():
    # copy the original dataframe
    cat_df = pd.get_dummies(join_df.loc[:, main_building_land_attributes +
 ↪sub_building_land_attributes])
    # taking only most seismic vulnerability factors
    cat_df = cat_df[most_seismic_vulnerability_factors]
    # run pca analysis on the dataframe
    # transformed_matrix, principal_components, explained_variance_ratio
    X_pca, pc, evr = principal_components_analysis(cat_df)

    results = pca_results(cat_df, pc, evr)
    x = np.arange(0,8)

    # plot the cumulative variance
    plt.plot(x, np.cumsum(evr), '-o', color='black')

    # plot styling
    plt.ylim(-1.05, 1.05)

    plt.annotate('Cumulative explained variance', xy=(3.7, .88),
 ↪arrowprops=dict(arrowstyle='->'), xytext=(4.5, .6))

    for i,j in zip(x, np.cumsum(evr)):
        plt.annotate(str(j.round(4)),xy=(i+.2,j-.02))

    # xticks
    plt.xticks(range(0,8))
    # set labels
    plt.xlabel('PCA components')
    plt.ylabel('Explained Variance')
    plt.show()

plot_pca_components_explained_variance()
```



7.21.1 PCA Components Explanation

- **Dimension 1 relates to Irregularity in Design and Best Practices, as greatest components OFT_q and GFT_f are aligned**
- **Dimension 2 relates to Damage in favour of RT_n and against OFT_q**
- **Dimension 3 could be representing the Orientation of the Building due to Position_s**
- **Dimension 4 relates to Construction/Damage in favour of (LSC-t) / Terrain Surfaces**
- **Dimension 5 relates to Construction by Floating GFT/Terrain/Raft and against OFT_q**
- **Dimension 6 relates to Construction in favour of Raft (r) Foundation Type and against GFT (f)**