In [1]:
```
import pandas as pd
import seaborn as sns
from matplotlib import dates
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from scipy.ndimage import gaussian_filter1d
import re
```

In [38]:
```
#implementation from https://www.toptal.com/python/topic-modeling-python
#using LDA to create topic models from the wall descriptions

# from sklearn.feature_extraction.text import CountVectorizer
# from sklearn.feature_extraction.text import TfidfTransformer
# from sklearn.decomposition import LatentDirichletAllocation as LDA
# from nltk.corpus import stopwords
```

In [2]:
```
pd.options.display.max_columns = None
pd.options.display.max_rows = None
```

In [3]:
```
dataset = pd.read_csv('1VyIL3EA62oNab4whc59KOntE8nvYQsBN_alt_FILES/Scotland EP
```

In [4]:
```
#inspecting the dataset
dataset.head()
```

Out[4]:

	Property_UPRN	Postcode	POST_TOWN	Date of Assessment	Primary Energy Indicator (kWh/m²/year)	Total floor area (m²)	Current energy efficiency rating	Current energy efficiency rating band	Po Effi
0	1.001101e+09	EH4 5EZ	EDINBURGH	01/01/2021	375.0	94.0	53.0	E	
1	1.001951e+09	EH7 4HE	EDINBURGH	01/01/2021	250.0	175.0	66.0	D	
2	1.000996e+09	EH4 2DL	EDINBURGH	02/01/2021	403.0	72.0	61.0	D	
3	1.001257e+09	PH1 1SA	PERTH	02/01/2021	174.0	96.0	76.0	C	
4	1.235709e+09	G78 1QN	Glasgow	02/01/2021	145.0	58.0	79.0	C	

In [5]:
```
dataset.columns
```

Out[5]:
```
Index(['Property_UPRN', 'Postcode', 'POST_TOWN', 'Date of Assessment',
       'Primary Energy Indicator (kWh/m²/year)', 'Total floor area (m²)',
       'Current energy efficiency rating',
       'Current energy efficiency rating band',
       'Potential Energy Efficiency Rating',
       'Potential energy efficiency rating band',
       'Current Environmental Impact Rating',
       'Current Environmental Impact Rating Band',
       'Potential Environmental Impact Rating',
       'Potential Environmental Impact Rating Band',
       'CO2 Emissions Current Per Floor Area (kg.CO2/m²/yr)',
       'WALL_DESCRIPTION', 'WALL_ENERGY_EFF', 'ROOF_DESCRIPTION',
       'ROOF_ENERGY_EFF', 'FLOOR_DESCRIPTION', 'FLOOR_ENERGY_EFF',
       'FLOOR_ENV_EFF', 'WINDOWS_DESCRIPTION', 'WINDOWS_ENERGY_EFF',
       'WINDOWS_ENV_EFF', 'MAINHEAT_DESCRIPTION', 'MAINHEAT_ENERGY_EFF',
       'MAINHEAT_ENV_EFF', 'MAINHEATCONT_DESCRIPTION', 'MAINHEATC_ENERGY_EF
F',
       'MAINHEATC_ENV_EFF', 'HOT_WATER_ENERGY_EFF', 'HOT_WATER_ENV_EFF',
       'LIGHTING_DESCRIPTION', 'LIGHTING_ENERGY_EFF', 'LIGHTING_ENV_EFF',
       'Current Emissions (T.CO2/yr)',
       'Potential Reduction in Emissions (T.CO2/yr)',
       'Total current energy costs over 3 years (£)',
       'Current heating costs over 3 years (£)',
       'Potential heating costs over 3 years (£)',
       'Current hot water costs over 3 years (£)',
       'Potential hot water costs over 3 years (£)',
       'Current lighting costs over 3 years (£)',
       'Potential lighting costs over 3 years (£)',
       'Part 1 Construction Age Band', 'Built Form', 'Property Type'],
      dtype='object')
```

In [9]:
```
town_efficiency = dataset.groupby(['POST_TOWN'])['Current energy efficiency ra
```

In [10]:
```
#noticing that the same town can appear multiple times
town_efficiency
```

```
Polbeth                  88.190476
Pollock                  79.500000
Polmont                  61.500000
Poolewe                  29.250000
Port Glasgow             39.333333
Port Logan               55.000000
Portpatrick              72.000000
Portree                  68.972222
Portsonachan             41.250000
Portsoy                  35.000000
Prestonpans              76.000000
Prestwick                76.750000
Pumpherston              61.000000
QUARRIERS VILLAGE        71.500000
Quarriers Village        53.000000
RENFREW                  70.258103
ROGART                   45.052632
ROSEWELL                 78.764706
ROSLIN                   79.063725
ROTHIENORMAN             77.000000
```

In [11]:
```
dataset['POST_TOWN'] = dataset.POST_TOWN.astype(str).str.lower().str.strip()
```

In [12]:
```
len(dataset.POST_TOWN.unique())
```

Out[12]:
```
666
```

In [13]:
```
dataset['POST_TOWN'] = dataset['POST_TOWN'].str.replace('edingburgh','edinburg
```

In [14]:
```
len(dataset.POST_TOWN.unique())
```

Out[14]:
```
665
```

In [15]:
```
dataset.isnull().sum()
```

Out[15]:
```
Property_UPRN                                          0
Postcode                                               0
POST_TOWN                                              0
Date of Assessment                                     0
Primary Energy Indicator (kWh/m²/year)                 0
Total floor area (m²)                                  0
Current energy efficiency rating                       0
Current energy efficiency rating band                  0
Potential Energy Efficiency Rating                     0
Potential energy efficiency rating band                0
Current Environmental Impact Rating                    0
Current Environmental Impact Rating Band               0
Potential Environmental Impact Rating                  0
Potential Environmental Impact Rating Band             0
CO2 Emissions Current Per Floor Area (kg.CO2/m²/yr)    0
WALL_DESCRIPTION                                       0
WALL_ENERGY_EFF                                        0
ROOF_DESCRIPTION                                       0
ROOF_ENERGY_EFF                                    39235
```

In [16]:
```
town_efficiency = dataset.groupby(['POST_TOWN'])['Current energy efficiency ra
```

In [17]:
```
dataset['POST_TOWN'] = dataset['POST_TOWN'].str.rstrip()
```

In [18]:
```
len(dataset.POST_TOWN.unique())
```

Out[18]:
```
665
```

Part 1: Rankings

1 Rank Towns by Current Energy Efficiency Rating

In [19]:
```
town_efficiency = dataset.groupby(['POST_TOWN'])['Current energy efficiency ra
town_efficiency = town_efficiency.sort_values(by='Mean', ascending=False ).res
town_efficiency.head(5)
```

Out[19]:

	POST_TOWN	Mean
0	gartocharn	115.000000
1	bannockburn	111.750000
2	gatehouse of fleet	101.000000
3	north lanarkshire	97.117647
4	south aryshire	96.000000

2 Rank Towns by potential energy efficiency rating

In [20]:
```
town_potential_eff = dataset.groupby(['POST_TOWN'])['Potential Energy Efficien
town_potential_eff=  town_potential_eff.sort_values(by='Mean', ascending=False
town_potential_eff.head(5)
```

Out[20]:

	POST_TOWN	Mean
0	gatehouse of fleet	128.0
1	comrie	123.0
2	sanday	119.0
3	crainlarich	118.5
4	gartocharn	118.0

3 Rank Towns by current environmental impact rating

In [21]:
```
town_curr_env_impact = dataset.groupby(['POST_TOWN'])['Current Environmental I
town_curr_env_impact = town_curr_env_impact.sort_values(by='Mean', ascending=F
town_curr_env_impact.head(5)
```

Out[21]:

	POST_TOWN	Mean
0	gartocharn	113.000000
1	bannockburn	109.750000
2	north lanarkshire	98.705882
3	south aryshire	97.000000
4	ardfern	96.000000

3 Periods where houses were relatively more or less environmentally friendly than average

In [22]:
```
town_env_date = dataset.groupby(['POST_TOWN', 'Date of Assessment' ])['Current
town_env_date.head()
```

Out[22]:

	POST_TOWN	Date of Assessment	Mean
0	aberdeen	01/02/2021	64.818182
1	aberdeen	01/03/2021	70.153846
2	aberdeen	01/04/2021	69.333333
3	aberdeen	01/06/2021	72.755556
4	aberdeen	01/07/2021	68.844444

In [23]:
```
town_periods = town_env_date.merge(town_curr_env_impact, left_on='POST_TOWN',
town_periods.head()
```

Out[23]:

	POST_TOWN	Date of Assessment	Mean_x	Mean_y
0	aberdeen	01/02/2021	64.818182	68.005636
1	aberdeen	01/03/2021	70.153846	68.005636
2	aberdeen	01/04/2021	69.333333	68.005636
3	aberdeen	01/06/2021	72.755556	68.005636
4	aberdeen	01/07/2021	68.844444	68.005636

In [24]:
```
# more friendly

town_periods['isMoreEnvFriendly'] =  town_periods['Mean_x'] < town_periods['Me
# town_periods_more
```

In [25]:
```
town_periods['Date of Assessment'] = pd.to_datetime(town_periods['Date of Asse
```

```
/home/randell-crapy/miniconda3/envs/finance/lib/python3.10/site-packages/pa
ndas/core/tools/datetimes.py:1047: UserWarning: Parsing '13/01/2021' in DD/
MM/YYYY format. Provide format or specify infer_datetime_format=True for co
nsistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
/home/randell-crapy/miniconda3/envs/finance/lib/python3.10/site-packages/pa
ndas/core/tools/datetimes.py:1047: UserWarning: Parsing '13/02/2021' in DD/
MM/YYYY format. Provide format or specify infer_datetime_format=True for co
nsistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
/home/randell-crapy/miniconda3/envs/finance/lib/python3.10/site-packages/pa
ndas/core/tools/datetimes.py:1047: UserWarning: Parsing '13/04/2021' in DD/
MM/YYYY format. Provide format or specify infer_datetime_format=True for co
nsistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
/home/randell-crapy/miniconda3/envs/finance/lib/python3.10/site-packages/pa
ndas/core/tools/datetimes.py:1047: UserWarning: Parsing '13/05/2021' in DD/
MM/YYYY format. Provide format or specify infer_datetime_format=True for co
nsistent parsing.
```

In [26]:
```
town_periods = town_periods.sort_values(by=['Date of Assessment']).reset_index
```

In [27]:
```
town_periods.head()
```

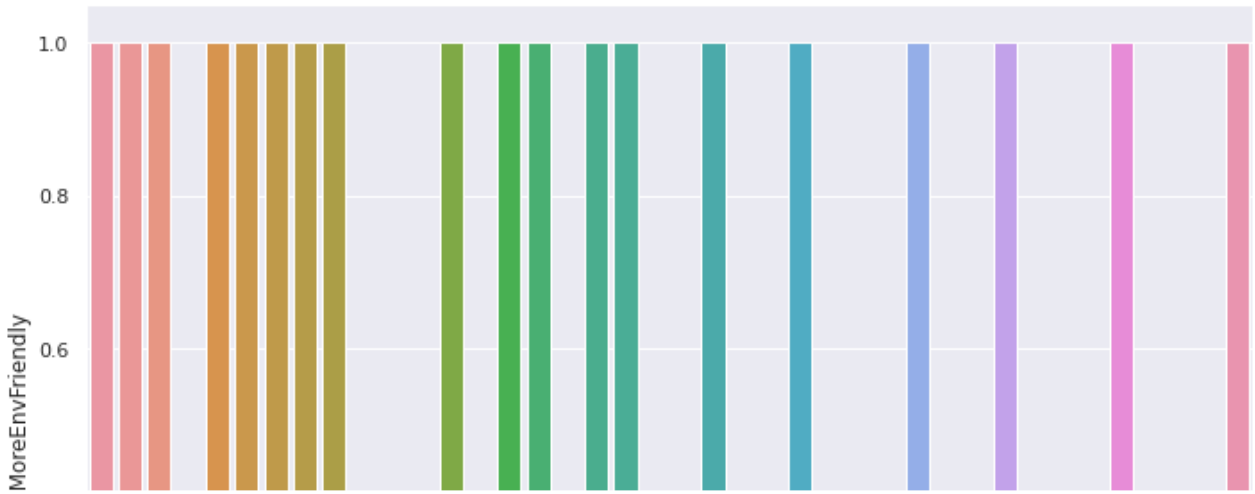
Out[27]:

	index	POST_TOWN	Date of Assessment	Mean_x	Mean_y	isMoreEnvFriendly
0	12489	edinburgh	2021-01-01	51.000000	69.462855	True
1	0	aberdeen	2021-01-02	64.818182	68.005636	True
2	10443	dumbarton	2021-01-02	63.000000	69.950456	True
3	23737	largs	2021-01-02	3.000000	69.317919	True
4	26695	millport	2021-01-02	37.000000	43.943182	True

In [31]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
```

In [32]:
```
start_date = 0
end_date = 5000
interval = 1
chosen_town = 'perth' # CHANGE THIS TO STUDY PERIODS FOR DIFFERENT TOWNS
to_viz = town_periods.sort_values(by=['Date of Assessment'])[town_periods['POS
# print(to_viz.head(30))
ax = sns.barplot(data=to_viz, x='Date of Assessment', y='isMoreEnvFriendly')
# ax.set_xticks(range(len(to_viz) // 5))  #, labels=range(2011, 2019)
ax.xaxis.set_major_locator(ticker.MultipleLocator(10))
ax.xaxis.set_major_formatter(dates.DateFormatter("%d-%b"))
```

```
/tmp/ipykernel_751989/86947112.py:5: UserWarning: Boolean Series key will b
e reindexed to match DataFrame index.
  to_viz = town_periods.sort_values(by=['Date of Assessment'])[town_periods
['POST_TOWN']  == chosen_town].loc[start_date:end_date: interval]
```



In [33]:
```
# less friendly

town_periods_more =  town_periods[town_periods['Mean_x'] > town_periods['Mean_
town_periods_more.head()
```

Out[33]:

	index	POST_TOWN	Date of Assessment	Mean_x	Mean_y	isMoreEnvFriendly
5	11289	dundee	2021-01-02	77.594595	66.440678	False
8	32306	shetland	2021-01-02	86.000000	48.292105	False
9	17416	hamilton	2021-01-02	70.714286	69.034586	False
10	25242	lochgelly	2021-01-02	77.500000	72.693913	False
13	35027	thurso	2021-01-02	88.000000	57.777778	False

In [34]:
```
town_env_date['Date of Assessment'] = pd.to_datetime(town_env_date['Date of As
```

In [35]:
```
gartocharn = town_env_date[town_env_date['POST_TOWN']=='gartocharn']
gartocharn
```

Out[35]:

	POST_TOWN	Date of Assessment	Mean
15324	gartocharn	2021-05-26	113.0

In [36]:
```
bannockburn = town_env_date[town_env_date['POST_TOWN']=='bannockburn']
bannockburn
```

Out[36]:

	POST_TOWN	Date of Assessment	Mean
3803	bannockburn	2021-06-28	109.75

In [37]:
```
gatehouse_of_fleet = town_env_date[town_env_date['POST_TOWN']=='gatehouse of f
gatehouse_of_fleet
```

Out[37]:

	POST_TOWN	Date of Assessment	Mean
15348	gatehouse of fleet	2021-08-19	83.0

In [38]:
```
north_lanarkshire = town_env_date[town_env_date['POST_TOWN']=='north lanarkshi
north_lanarkshire
```

Out[38]:

	POST_TOWN	Date of Assessment	Mean
28915	north lanarkshire	2021-11-11	99.1875
28916	north lanarkshire	2021-03-25	91.0000

In [39]:
```
south_aryshire = town_env_date[town_env_date['POST_TOWN']=='south aryshire']
south_aryshire
```

Out[39]:

	POST_TOWN	Date of Assessment	Mean
32811	south aryshire	2021-08-04	97.0

In [40]:
```
ardfern = town_env_date[town_env_date['POST_TOWN']=='ardfern']
ardfern
```

Out[40]:

	POST_TOWN	Date of Assessment	Mean
2354	ardfern	2021-12-14	96.0

In [41]:
```
measurements = dataset.groupby(['POST_TOWN'])['POST_TOWN'].count().reset_index
```

In [42]:
```
measurements = measurements.sort_values(by='count', ascending=False).reset_ind
measurements.head()
```

Out[42]:

	POST_TOWN	count
0	glasgow	37529
1	edinburgh	19276
2	aberdeen	9226
3	dundee	5782
4	paisley	3136

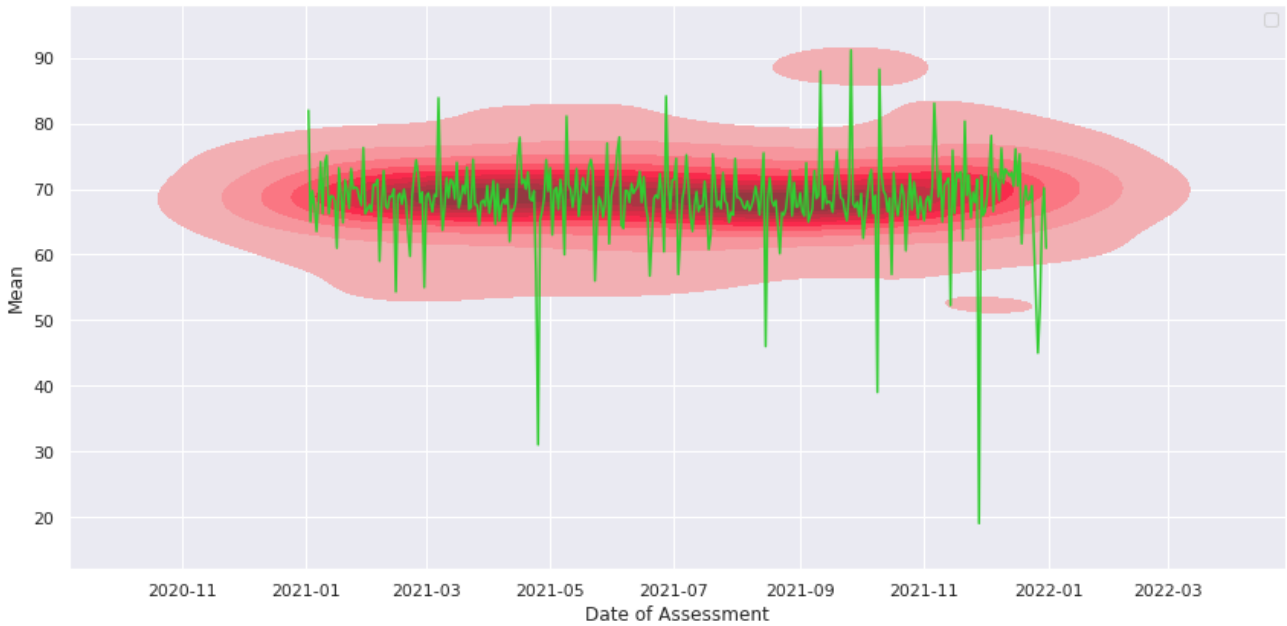
In [43]:
```
glasgow = town_env_date[town_env_date['POST_TOWN']=='glasgow']
glasgow
```

Out[43]:

	POST_TOWN	Date of Assessment	Mean
15486	glasgow	2021-02-01	67.567164
15487	glasgow	2021-03-01	68.902256
15488	glasgow	2021-04-01	66.358779
15489	glasgow	2021-05-01	73.200000
15490	glasgow	2021-06-01	70.469613
15491	glasgow	2021-07-01	71.487654
15492	glasgow	2021-08-01	68.750000
15493	glasgow	2021-09-01	69.426357
15494	glasgow	2021-10-01	69.291339
15495	glasgow	2021-11-01	65.218543
15496	glasgow	2021-12-01	66.600000

In [44]:
```
#Glasglow does not really show a trend.
fig, ax = plt.subplots(figsize=(12, 6))
sns.kdeplot(data=glasgow,
            color='crimson', x='Date of Assessment', y='Mean', fill=True, ax=a
sns.lineplot(data=glasgow,x='Date of Assessment', y='Mean', color='limegreen')
ax.legend()
plt.tight_layout()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argumen
t.
```



In [45]:
```
#Glasglow does not really show a trend.
fig, ax = plt.subplots(figsize=(12, 6))

Final_array_smooth = gaussian_filter1d(glasgow['Mean'].to_numpy(), sigma=20) #
glasgow['smooth_mean'] = Final_array_smooth
sns.lineplot(data=glasgow,x='Date of Assessment', y='Mean', color='crimson')
sns.lineplot(data=glasgow,x='Date of Assessment', y='smooth_mean', color='lime
ax.legend()
plt.tight_layout()
plt.show()
# Smoothing
```

```
/tmp/ipykernel_751989/1656999309.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  glasgow['smooth_mean'] = Final_array_smooth
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argumen
t.
```

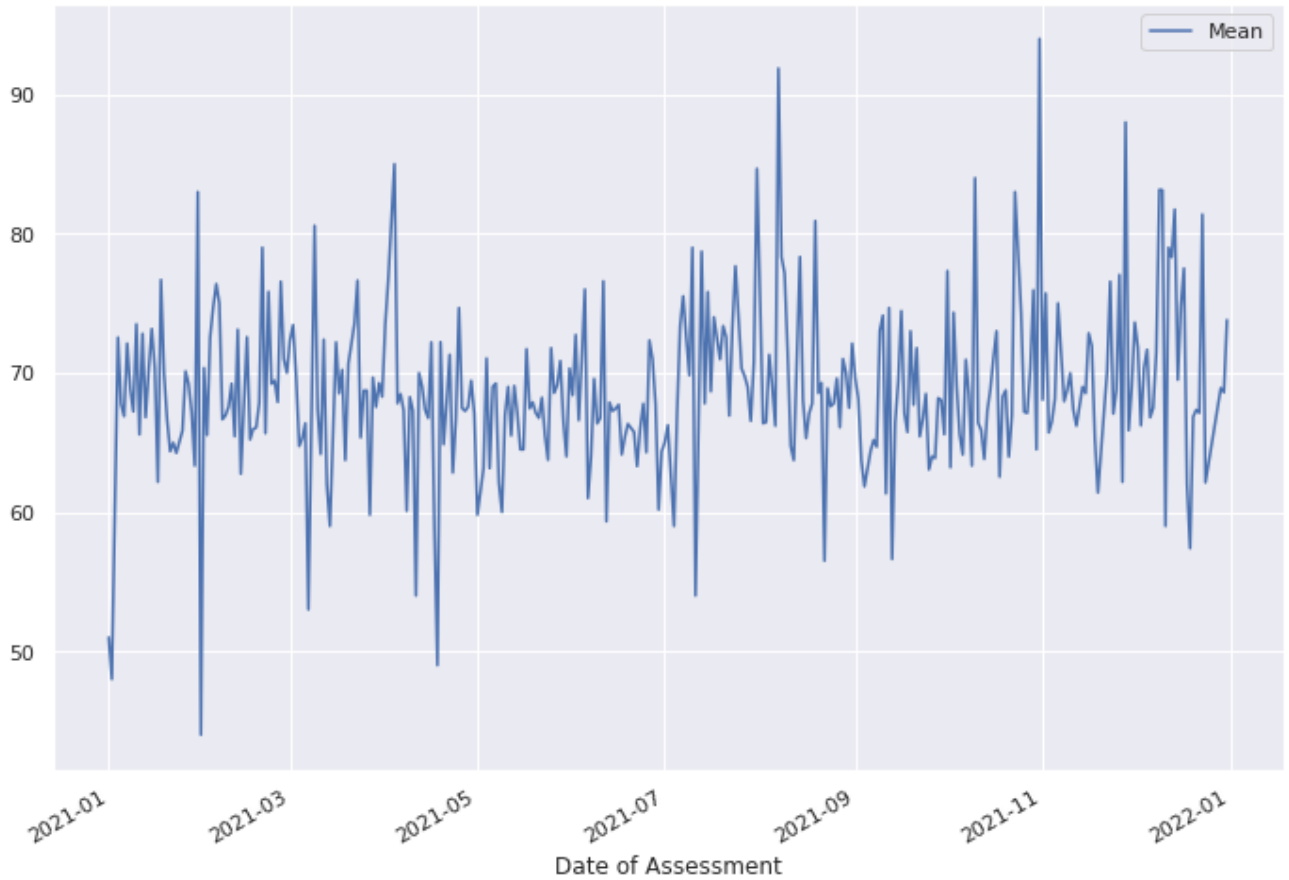
In [46]:
```
edinburgh = town_env_date[town_env_date['POST_TOWN']=='edinburgh']
edinburgh
```

Out[46]:

	POST_TOWN	Date of Assessment	Mean
12489	edinburgh	2021-01-01	51.000000
12490	edinburgh	2021-02-01	70.355932
12491	edinburgh	2021-03-01	72.357143
12492	edinburgh	2021-04-01	73.457447
12493	edinburgh	2021-05-01	59.800000
12494	edinburgh	2021-06-01	68.397590
12495	edinburgh	2021-07-01	65.000000
12496	edinburgh	2021-08-01	75.250000
12497	edinburgh	2021-09-01	69.628571
12498	edinburgh	2021-10-01	77.322581
12499	edinburgh	2021-11-01	68.066667

In [47]:
```
#Edinburgh seems to show larger impact at the end of the year
edinburgh.plot(x='Date of Assessment', y='Mean')
```

Out[47]:
```
<AxesSubplot: xlabel='Date of Assessment'>
```



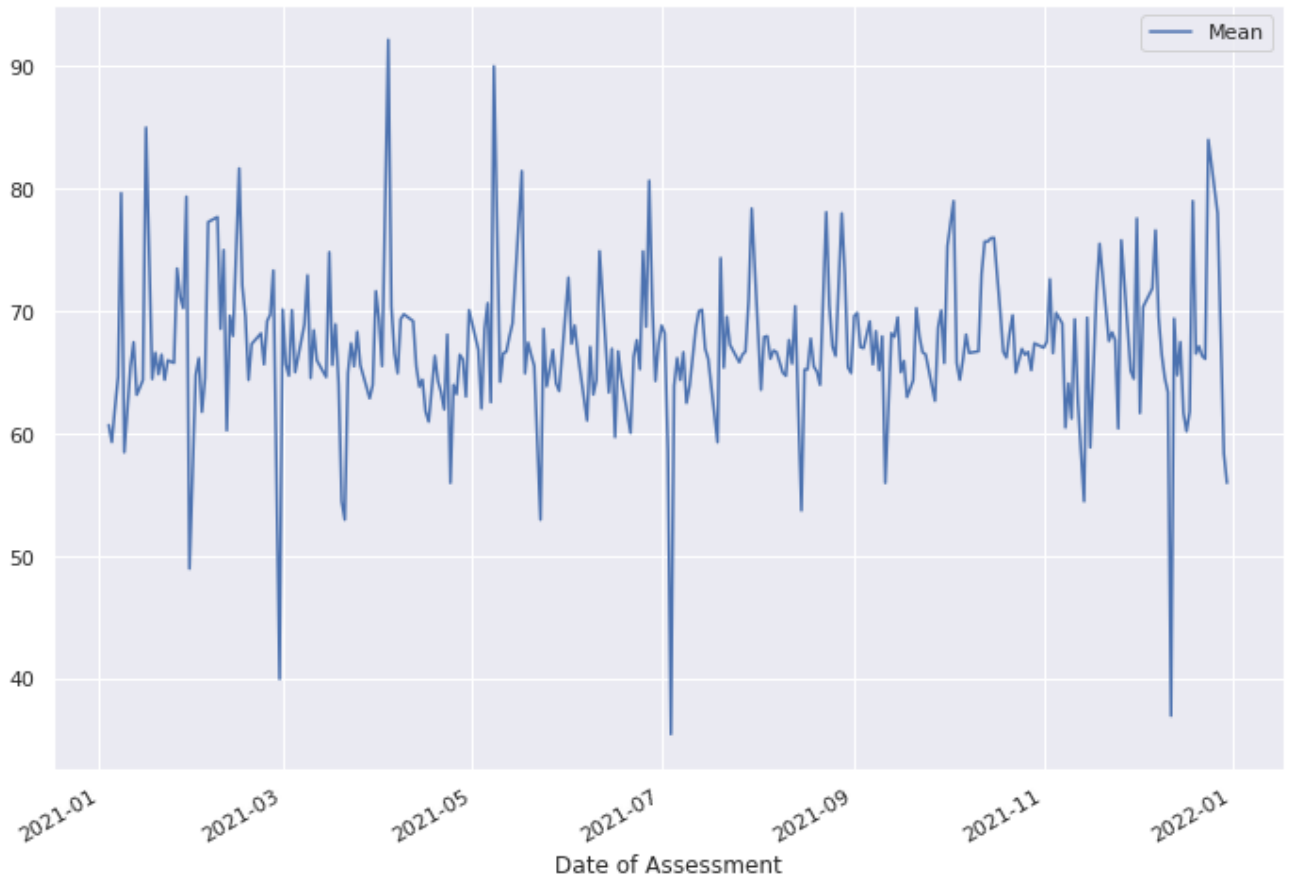
In [48]:
```
aberdeen = town_env_date[town_env_date['POST_TOWN']=='aberdeen']
aberdeen
```

Out[48]:

	POST_TOWN	Date of Assessment	Mean
0	aberdeen	2021-02-01	64.818182
1	aberdeen	2021-03-01	70.153846
2	aberdeen	2021-04-01	69.333333
3	aberdeen	2021-06-01	72.755556
4	aberdeen	2021-07-01	68.844444
5	aberdeen	2021-09-01	69.595238
6	aberdeen	2021-10-01	75.333333
7	aberdeen	2021-11-01	67.035714
8	aberdeen	2021-12-01	77.588710
9	aberdeen	2021-02-02	66.181818
10	aberdeen	2021-03-02	65.678571

In [49]:
```
#There does not seem to be a pattern for Aberdeen.
aberdeen.plot(x='Date of Assessment', y='Mean')
```

Out[49]:
```
<AxesSubplot: xlabel='Date of Assessment'>
```



4 Rank Towns by potential environmental impact rating 'Potential Environmental Impact Rating'

In [50]:
```
town_pot_env = dataset.groupby(['POST_TOWN'])['Potential Environmental Impact
town_pot_env = town_pot_env.sort_values(by='Mean', ascending=False).reset_inde
town_pot_env.head()
```

Out[50]:

	POST_TOWN	Mean
0	west plean	126.0
1	meigle	122.0
2	comrie	121.0
3	gartocharn	117.0
4	sanday	117.0

5 Rank Towns by Current Emissions (T.CO2/yr)

In [51]:
```
town_curr_em = dataset.groupby(['POST_TOWN'])['Current Emissions (T.CO2/yr)'].
town_curr_em = town_pot_env.sort_values(by='Mean', ascending=False).reset_inde
town_curr_em.head()
```

Out[51]:

	POST_TOWN	Mean
0	west plean	126.0
1	meigle	122.0
2	comrie	121.0
3	sanday	117.0
4	gartocharn	117.0

6 Rank Towns by Potential Reduction in Emissions (T.CO2/yr)

In [52]:
```
town_pot_red_em = dataset.groupby(['POST_TOWN'])['Potential Reduction in Emiss
town_pot_red_em = town_pot_env.sort_values(by='Mean', ascending=False).reset_i
town_pot_red_em.head()
```

Out[52]:

	POST_TOWN	Mean
0	west plean	126.0
1	meigle	122.0
2	comrie	121.0
3	sanday	117.0
4	gartocharn	117.0

7 Rank Towns by potential savings in heating costs (£) over three years

In [53]:
```
dataset['heat_savings'] = dataset['Current heating costs over 3 years (£)'] -
```

In [54]:
```
town_save_heating = dataset.groupby(['POST_TOWN'])['heat_savings'].mean().rese
town_save_heating = town_save_heating.sort_values(by='Mean', ascending=False).
town_save_heating.head()
```

Out[54]:

	POST_TOWN	Mean
0	bonawe	11085.0
1	east dunbartonshire	11016.0
2	morvern	7098.0
3	corsock	6639.0
4	findhorn	5922.0

8 Rank Towns by potential savings in hot water costs (£) over three years

In [55]:
```
dataset['hot_water_save'] = dataset['Current hot water costs over 3 years (£)'
town_hot_water_save = dataset.groupby(['POST_TOWN'])['hot_water_save'].mean().
town_hot_water_save = town_hot_water_save.sort_values(by='Mean', ascending=Fal
town_hot_water_save.head()
```

Out[55]:

	POST_TOWN	Mean
0	kincardine	2283.0
1	east dunbartonshire	1653.0
2	by maybole	1080.0
3	corsock	1056.0
4	burghead	978.0

9 Rank the top 5 wall descriptions (wall materials) by CO2 emissions current per floor area and wall energy efficiency

(create a single rating combining CO2 emissions and wall energy efficiency)

In [93]:
```
rating_map = {
    'Very Poor': 1,
    'Poor': 2,
    'Average': 3,
    'Good': 4,
    'Very Good': 5,
}
```

In [94]:
```
# Including assumed descriptions, experimented with and without and there wasn
def clean_string(a: str)-> str:
    return a.replace('(assumed)','').strip().replace('Â','')
```

In [95]:
```
def general_preprocess(df,description_col, EE_col):
    df = df.dropna(subset=[EE_col])
    df = df.dropna(subset=[description_col])
    df[description_col] = df[description_col].str.strip()
    df[EE_col] = df[EE_col].str.strip()
    energy_and_emission = []
    DELIMITER = "|"
    for idx, row in df.iterrows():
        if DELIMITER not in row[EE_col]:
            if('N/A' in row[EE_col]):
                continue
            energy_and_emission.append((clean_string(row[description_col]), cl
            continue
        wall_desc = row[description_col].split(DELIMITER)
        wall_eff = row[EE_col].split(DELIMITER)
        for i in range(len(wall_desc)):
            if('N/A' in wall_eff[i]):
                continue
            energy_and_emission.append((clean_string(wall_desc[i]),clean_strin
    ee_df = pd.DataFrame(energy_and_emission, columns=["description", "rating"
    ee_df = ee_df.dropna(subset=['description'])
    ee_df = ee_df.dropna(subset=['rating'])
    ee_df = ee_df[ee_df['rating'] != "nan"]
    ee_df['description'] = ee_df['description'].str.strip()
    ee_df['rating'] = ee_df['rating'].str.strip()
    ee_df['rating'].value_counts()
    just_rating = ee_df[['rating']]
    ee_df['rating'] = just_rating.applymap(lambda s: rating_map.get(s) if s in
    # Min Max Noramlise CO2
    CO2_col = ee_df['CO2']
    ee_df['CO2_norm'] = (CO2_col - CO2_col.min())/(CO2_col.max()-CO2_col.min()
#     print(ee_df['rating'].value_counts())
    ee_df['combi'] = ee_df['rating'] + ee_df['CO2_norm']
    return ee_df
```

In [96]:
```
energy_and_emission = []
DELIMITER = "|"
for idx, row in dataset.iterrows():
    if DELIMITER not in row.WALL_ENERGY_EFF:
        energy_and_emission.append((clean_string(row.WALL_DESCRIPTION), clean_
        continue
    wall_desc = row.WALL_DESCRIPTION.split(DELIMITER)
    wall_eff = row.WALL_ENERGY_EFF.split(DELIMITER)
    for i in range(len(wall_desc)):
        energy_and_emission.append((clean_string(wall_desc[i]),clean_string(wa
```

In [97]:
```
ee_df = pd.DataFrame(energy_and_emission, columns=["description", "rating","CO
```

In [98]:
```
ee_df['description'] = ee_df['description'].str.strip()
```

In [99]:
```
# Convert ratings to a number from 1 to 5
ee_df['rating'] = ee_df['rating'].str.strip()
ee_df['rating'].value_counts()
```

Out[99]:
```
Poor         70120
Good         65563
Average      48141
Very Good    42792
Very Poor    13413
Name: rating, dtype: int64
```

In [100]:
```
just_rating = ee_df[['rating']]
ee_df['rating'] = just_rating.applymap(lambda s: rating_map.get(s) if s in rat
```

In [101]:
```
ee_df.head()
```

Out[101]:

	description	rating	CO2
0	Cavity wall, as built, no insulation	2	66.0
1	Cavity wall, filled cavity	3	44.0
2	Cavity wall, as built, partial insulation	3	68.0
3	Cavity wall, filled cavity	3	68.0
4	Cavity wall, as built, insulated	4	31.0

In [102]:
```
# Min Max Noramlise CO2
CO2_col = ee_df['CO2']
# ee_df['CO2_norm'] = (CO2_col - CO2_col.mean())/CO2_col.std()
ee_df['CO2_norm'] = (CO2_col - CO2_col.min())/(CO2_col.max()-CO2_col.min()) *
```

In [103]:
```
# My combined metric would be out of 10, out of 5 for rating and out of 5 for
ee_df['combi'] = ee_df['rating'] + ee_df['CO2_norm']
```

In [104]:
```
wall_desc_ee = ee_df.groupby(['description'])['combi'].mean().reset_index(name
wall_desc_ee = wall_desc_ee.sort_values(by='Mean', ascending=True).reset_index
wall_desc_ee
```

Out[104]:

	description	Mean
0	Average thermal transmittance 1.70 W/m²K	2.223118
1	Average thermal transmittance 1.80 W/m²K	2.350806
2	Average thermal transmittance 1.63 W/m²K	2.424731
3	System built, as built, no insulation	2.696194
4	Timber frame, as built, no insulation	3.029699
5	Average thermal transmittance 1.06 W/m²K	3.213038
6	Average thermal transmittance 1.55 W/m²K	3.223118
7	Average thermal transmittance 1.40 W/m²K	3.256720
8	Average thermal transmittance 1.51 W/m²K	3.266801
9	Average thermal transmittance 1.31 W/m²K	3.280242
10	Average thermal transmittance 1.26 W/m²K	3.283602

10 Rank the top 5 roof descriptions by CO2 emissions current per floor area and wall energy efficiency

(create a single rating combining CO2 emissions and wall energy efficiency)

In [105]:
```
r_dataset = dataset.dropna(subset=["ROOF_ENERGY_EFF"])
r_dataset["ROOF_DESCRIPTION"] = r_dataset["ROOF_DESCRIPTION"].str.strip()
r_dataset["ROOF_ENERGY_EFF"] = r_dataset["ROOF_ENERGY_EFF"].str.strip()
# r_dataset = r_dataset[r_dataset["ROOF_ENERGY_EFF"].str.contains("N/A") == Fa
```

```
/tmp/ipykernel_751989/3450782831.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  r_dataset["ROOF_DESCRIPTION"] = r_dataset["ROOF_DESCRIPTION"].str.strip()
/tmp/ipykernel_751989/3450782831.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  r_dataset["ROOF_ENERGY_EFF"] = r_dataset["ROOF_ENERGY_EFF"].str.strip()
```

In [106]:
```
r_dataset.ROOF_ENERGY_EFF.value_counts()
```

Out[106]:
```
Good                          51723
Very Good                     33936
N/A                           11929
Very Poor                      9991
Average                        9120
Good | Good                    4940
Poor                           3070
Very Poor | Very Poor          2522
Good | Very Poor               2024
Average | Good                1460
Very Poor | Good               1314
Very Good | Good              1222
Good | Very Good                928
Very Poor | Poor               892
Good | Poor                    878
Average | Very Poor            787
Good | Average                 705
Poor | Good                    474
Very Good | Very Poor          448
```

In [107]:
```
r_energy_and_emission = []
DELIMITER = "|"
for idx, row in r_dataset.iterrows():
    if DELIMITER not in row.ROOF_ENERGY_EFF:
        if('N/A' in row.ROOF_ENERGY_EFF):
            continue
        energy_and_emission.append((clean_string(row.ROOF_DESCRIPTION), clean_

        continue
    r_desc = row.ROOF_DESCRIPTION.split(DELIMITER)
    r_eff = row.ROOF_ENERGY_EFF.split(DELIMITER)
    for i in range(len(r_desc)):
        if ('N/A' in r_eff[i]):
            continue
        r_energy_and_emission.append((clean_string(r_desc[i]),clean_string(r_e
```

In [108]:
```
ree_df = pd.DataFrame(r_energy_and_emission, columns=["description", "rating",
# Convert ratings to a number from 1 to 5
ree_df['description'] = ree_df['description'].str.strip()
ree_df['rating'] = ree_df['rating'].str.strip()
ree_df['rating'].value_counts()
just_rating = ree_df[['rating']]
ree_df['rating'] = just_rating.applymap(lambda s: rating_map.get(s) if s in ra
# Min Max Noramlise CO2
CO2_col = ree_df['CO2']
ree_df['CO2_norm'] = (CO2_col - CO2_col.min())/(CO2_col.max()-CO2_col.min()) *
ree_df['combi'] = ree_df['rating'] + ree_df['CO2_norm']
r_desc_ee = ree_df.groupby(['description'])['combi'].mean().reset_index(name='
r_desc_ee = r_desc_ee.sort_values(by='Mean', ascending=True).reset_index(drop=
r_desc_ee
```

Out[108]:

	description	Mean
0	Pitched, no insulation	2.029174
1	Flat, no insulation	2.043901
2	Pitched, 12 mm loft insulation	2.051051
3	Roof room(s), no insulation	2.125227
4	Pitched, limited insulation	2.221802
5	Flat, limited insulation	2.399630
6	Pitched, 0 mm loft insulation	2.469970
7	Pitched, 50 mm loft insulation	2.966181
8	Pitched, 25 mm loft insulation	3.002995
9	Roof room(s), ceiling insulated	3.302337
10	Roof room(s), limited insulation	3.329245
11	Pitched, 100 mm loft insulation	3.907899
12	Pitched, 75 mm loft insulation	3.929864
13	Pitched, insulated at rafters	4.156821
14	Flat, insulated	4.191999
15	Thatched	4.316441
16	Pitched, insulated	4.510150
17	Roof room(s), insulated	4.726006
18	Pitched, 270 mm loft insulation	4.775243
19	Pitched, 250 mm loft insulation	4.790899
20	Pitched, 200 mm loft insulation	4.819739
21	Roof room(s), thatched	4.833333
22	Pitched, 150 mm loft insulation	4.849895
23	Pitched, 400 mm loft insulation	5.520721
24	Pitched, 400+ mm loft insulation	5.654762
25	Pitched, 400+ mm loft insulation	5.726351
26	Pitched, 350 mm loft insulation	5.732421
27	(another dwelling above)	5.777027
28	Pitched, 300 mm loft insulation	5.783290
29	Thatched, with additional insulation	6.266892

Part 2: Algorithm Challenges

Algorithm Challenge 1: Build and algorithm to find correlations between CO2 emissions current per floor area vs wall description and wall energy efficiency

In [109]:
```
ee_df['description'].value_counts()
```

```
Timber frame, as built, no insulation                    2242
Average thermal transmittance 0.16 W/m²K                 2103
Sandstone or limestone, as built, partial insulation     1997
Sandstone or limestone, with internal insulation         1467
Average thermal transmittance 0.18 W/m²K                 1430
Sandstone or limestone, as built, insulated              1426
Average thermal transmittance 0.19 W/m²K                 1223
Average thermal transmittance 0.15 W/m²K                 1115
Solid brick, with external insulation                    1087
Average thermal transmittance 0.20 W/m²K                 1081
Granite or whinstone, with internal insulation            905
Granite or whinstone, as built, insulated                 778
Timber frame, with additional insulation                  619
Solid brick, with internal insulation                     611
Cavity wall, filled cavity and external insulation        564
Average thermal transmittance 0.14 W/m²K                  545
System built, as built, partial insulation                416
Cavity wall, with internal insulation                     387
System built, with internal insulation                    291
Granite or whinstone, as built, partial insulation        262
```

In [110]:
```
len(ee_df['description'].unique())
```

Out[110]:
```
120
```

Wall description can be classified as 2 kinds of data

1. Categorical like Cavity wall, as build, no insulation
2. Continuous like Average thermal transmittance 0.22 W/m²K

We'll start with continuous data first

In [111]:
```
just_test = dataset[dataset['WALL_DESCRIPTION'].str.contains('thermal transmit
```

In [112]:
```
# This shows wall descriptions with thermal transmittance values isn't a mix o
just_test.WALL_DESCRIPTION.value_counts()
```

Out[112]:
```
Average thermal transmittance 0.22 W/m²K      4659
Average thermal transmittance 0.21 W/m²K      3223
Average thermal transmittance 0.17 W/m²K      2477
Average thermal transmittance 0.16 W/m²K      1590
Average thermal transmittance 0.22 W/m²K      1284
Average thermal transmittance 0.18 W/m²K      1101
Average thermal transmittance 0.21 W/m²K      1016
Average thermal transmittance 0.15 W/m²K       960
Average thermal transmittance 0.19 W/m²K       908
Average thermal transmittance 0.20 W/m²K       868
Average thermal transmittance 0.17 W/m²K       825
Average thermal transmittance 0.16 W/m²K       464
Average thermal transmittance 0.14 W/m²K       435
Average thermal transmittance 0.18 W/m²K       329
Average thermal transmittance 0.19 W/m²K       300
Average thermal transmittance 0.20 W/m²K       206
Average thermal transmittance 0.13 W/m²K       177
Average thermal transmittance 0.23 W/m²K       176
Average thermal transmittance 0.15 W/m²K       153
```

In [113]:
```
# def get_thermal_transmittance(x):
#     description = x['description']
#     nums = re.findall(r'\d+',description)
#     print(nums)
#     return nums[0] if len(nums) > 0 else 0
```

In [114]:
```
thermal_df = ee_df[ee_df['description'].str.contains('thermal transmittance')]
```

In [115]:
```
# thermal_df['val'] = thermal_df.apply(get_thermal_transmittance, axis=1)
thermal_df['thermal_val'] = thermal_df['description'].str.extract(r'(\d+.\d+)'
```

```
/tmp/ipykernel_751989/3705620401.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  thermal_df['thermal_val'] = thermal_df['description'].str.extract(r'(\d+.\d
+)').astype('float')
```

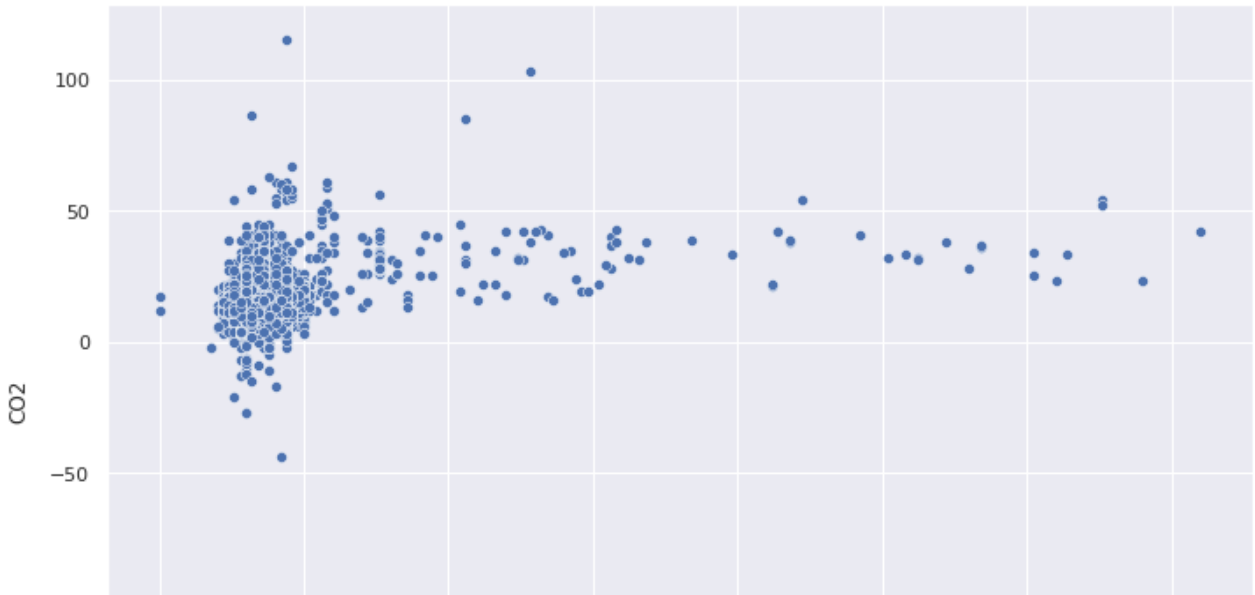
In [116]:
```
thermal_df.head(10)
```

Out[116]:

	description	rating	CO2	CO2_norm	combi	thermal_val
121	Average thermal transmittance 0.20 W/m²K	5	17.0	1.182796	6.182796	0.20
122	Average thermal transmittance 0.21 W/m²K	5	12.0	1.149194	6.149194	0.21
123	Average thermal transmittance 0.22 W/m²K	5	12.0	1.149194	6.149194	0.22
124	Average thermal transmittance 0.21 W/m²K	5	11.0	1.142473	6.142473	0.21
125	Average thermal transmittance 0.21 W/m²K	5	13.0	1.155914	6.155914	0.21
126	Average thermal transmittance 0.25 W/m²K	5	8.0	1.122312	6.122312	0.25
127	Average thermal transmittance 0.25 W/m²K	5	8.0	1.122312	6.122312	0.25
131	Average thermal transmittance 0.25 W/m²K	5	8.0	1.122312	6.122312	0.25
132	Average thermal transmittance 0.25 W/m²K	5	8.0	1.122312	6.122312	0.25
133	Average thermal transmittance 0.22 W/m²K	5	6.0	1.108871	6.108871	0.22

In [117]:
```
sns.scatterplot(data=thermal_df, x='thermal_val',y='CO2')
```

Out[117]:
```
<AxesSubplot: xlabel='thermal_val', ylabel='CO2'>
```



Generally there doesn't seem to be a correlation between thermal transmittance and CO2 emissions.

Let's take a look wrt their energy rating

In [118]:
```
desired_rating = 2
filtered_thermal_df = thermal_df[thermal_df['rating'] == desired_rating]
filtered_thermal_df.head()
```

Out[118]:

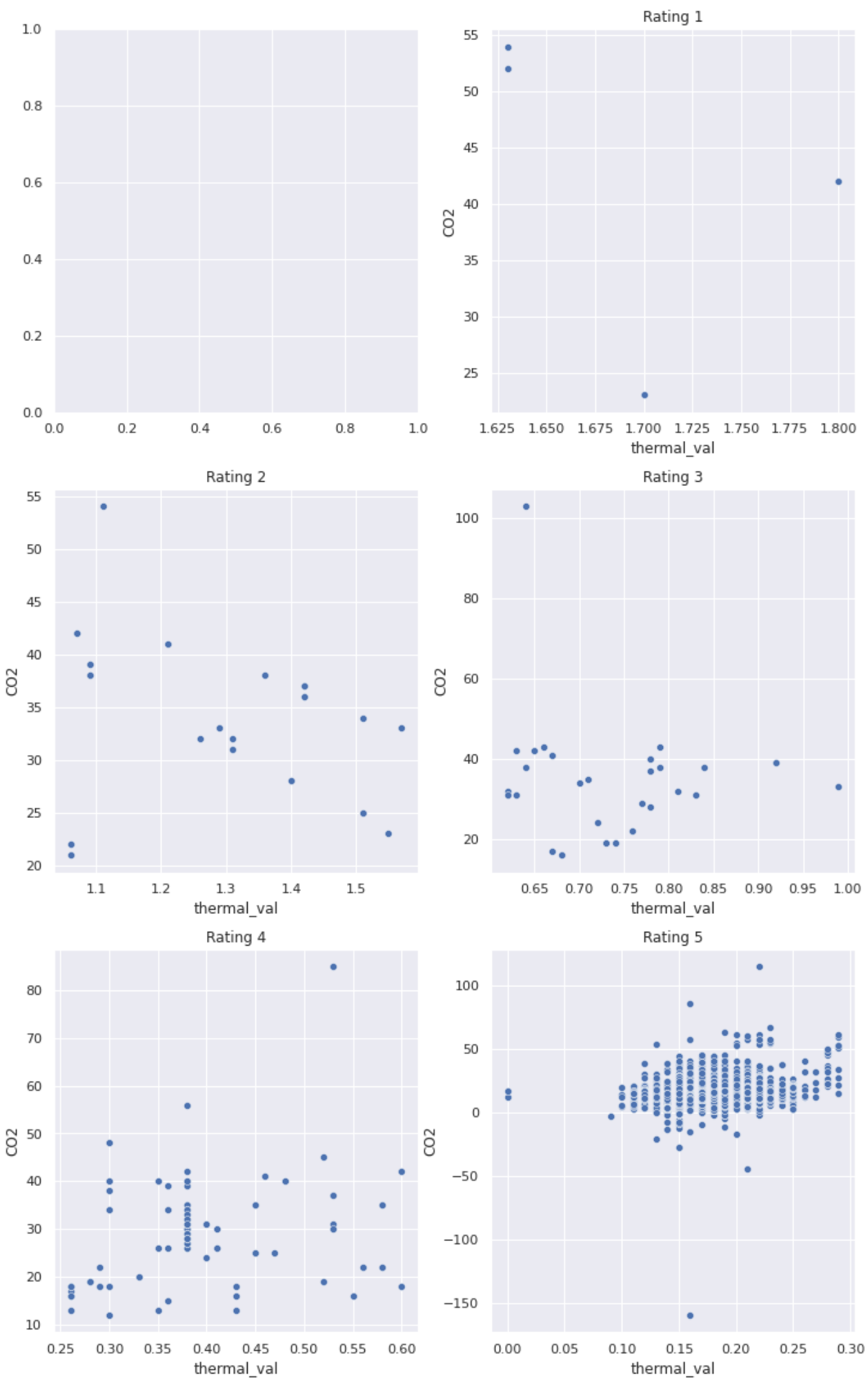
	description	rating	CO2	CO2_norm	combi	thermal_val
10258	Average thermal transmittance 1.09 W/m²K	2	38.0	1.323925	3.323925	1.09
10268	Average thermal transmittance 1.09 W/m²K	2	39.0	1.330645	3.330645	1.09
10269	Average thermal transmittance 1.09 W/m²K	2	38.0	1.323925	3.323925	1.09
10303	Average thermal transmittance 1.09 W/m²K	2	39.0	1.330645	3.330645	1.09
71333	Average thermal transmittance 1.51 W/m²K	2	25.0	1.236559	3.236559	1.51

In [119]:
```
# Number of each rating
print(len(filtered_thermal_df))
```

```
24
```

In [120]:

```
fig, ax = plt.subplots(3, 2)
fig.set_size_inches(12, 20)
for i in range(1,6):
    desired_rating = i
    filtered_thermal_df = thermal_df[thermal_df['rating'] == desired_rating]
    ax[i // 2, i % 2].set_title(f'Rating {desired_rating}')
#     ax[i // 2, i % 2].plot(filtered_thermal_df['thermal_val'],filtered_therm
    sns.scatterplot(ax=ax[i // 2, i % 2], data=filtered_thermal_df, x='thermal
```



Again there isn't any correlation between thermal transmittance and CO2 emissions wrt their energy

ratings.

However, we can at least observe that thermal transmittance is lower as energy ratings are better rated.

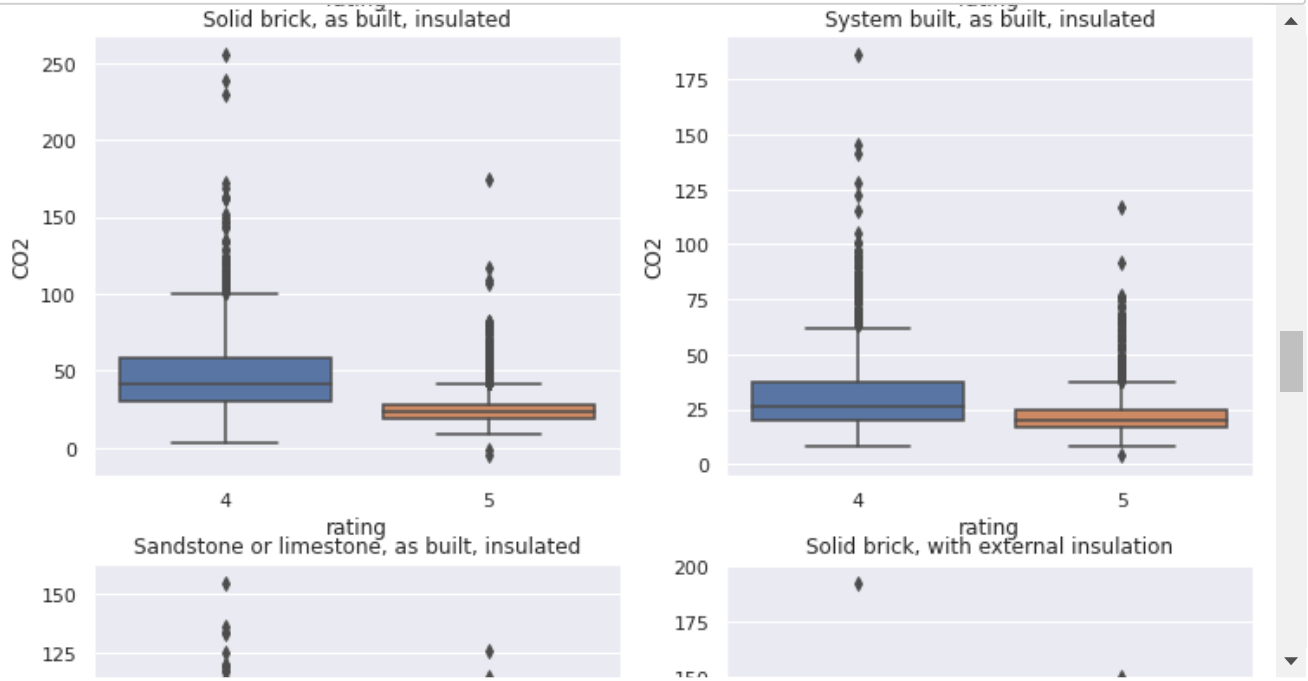
Let's now investigate if wall descriptions of same categorical value are clustered together in terms of C02 emissions

In [121]:
```
import math
```

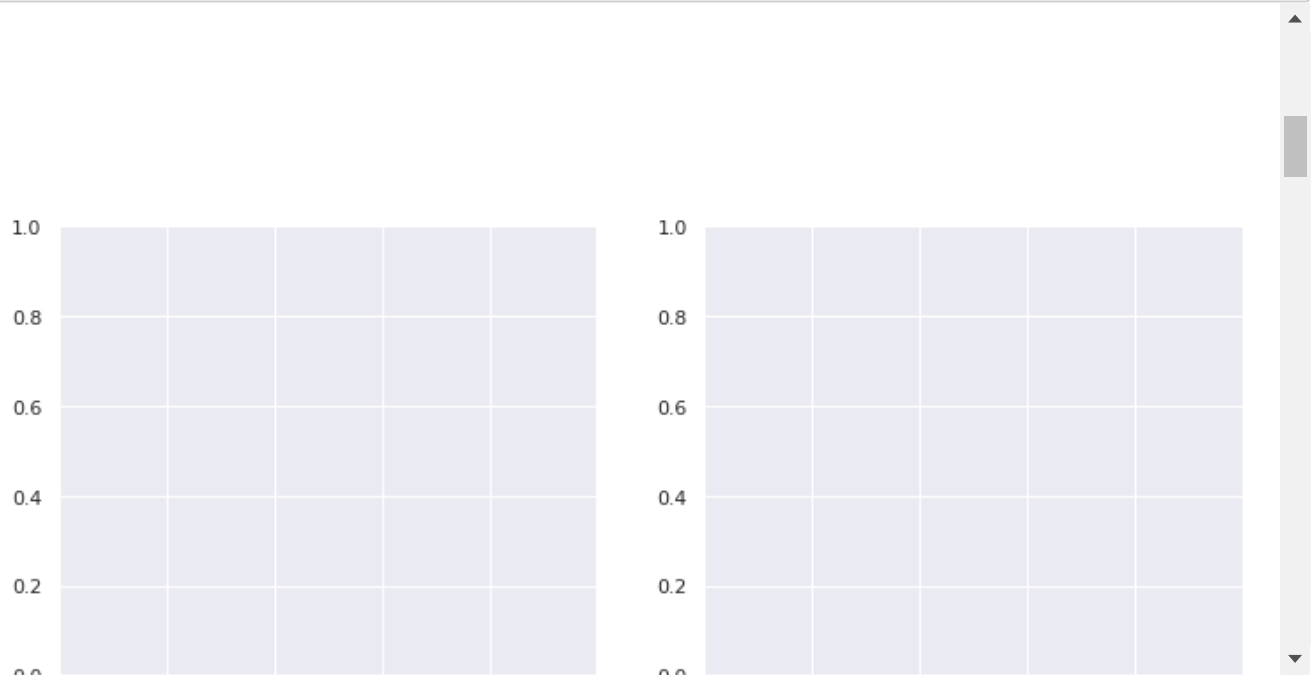
In [122]:
```
others_df = ee_df[ee_df['description'].str.contains('thermal transmittance') =
```

In [123]:
```
unique_descriptions = others_df['description'].unique()
```

In [125]:
```
fig, ax = plt.subplots(math.ceil(len(unique_descriptions) / 2), 2)
fig.set_size_inches(12, 100)
for i in range(len(unique_descriptions)):
    cur = unique_descriptions[i]
    filtered_others_df = others_df[others_df['description'] == cur]
    ax[i // 2, i % 2].set_title(f'{cur}')
    sns.boxplot(ax=ax[i // 2, i % 2], data=filtered_others_df, x='rating',y='C
```



In [126]:
```
fig, ax = plt.subplots(math.ceil(len(unique_descriptions) / 2), 2)
fig.set_size_inches(12, 100)
for i in range(len(unique_description_df)):
    cur = unique_descriptions[i]
    filtered_others_df = others_df[others_df['description'] == cur]
    ax[i // 2, i % 2].set_title(f'{cur}')
    sns.scatterplot(ax=ax[i // 2, i % 2], data=filtered_others_df, x='rating',
```



The above plots shows a range of CO2 values a specific category of wall descriptions can take, with some rating specific clustering, variances, rough expected min and max CO2 emission values.

Algorithm Challenge 2: Build and algorithm to find correlations between CO2 emissions current per floor area vs roof description and roof energy efficiency

In [127]:
```
ree_df['description'].value_counts()
# seems like loft insulation only occurs on pitched roofs
```

Out[127]:
```
Roof room(s), insulated                  6829
Pitched, no insulation                   6341
Pitched, insulated                       6185
Roof room(s), no insulation              4447
Pitched, 200 mm loft insulation          3761
Roof room(s), ceiling insulated          3378
Pitched, 300 mm loft insulation          3136
Pitched, 150 mm loft insulation          2807
Pitched, 250 mm loft insulation          2737
Flat, insulated                          2345
Pitched, 100 mm loft insulation          2219
Flat, limited insulation                 2197
Pitched, 270 mm loft insulation          1875
Pitched, limited insulation              1858
Flat, no insulation                      1396
Roof room(s), limited insulation         1007
Pitched, 50 mm loft insulation            443
Pitched, insulated at rafters             413
Pitched, 350 mm loft insulation           313
Pitched, 75 mm loft insulation            299
Pitched, 400+ mm loft insulation          252
Pitched, 25 mm loft insulation             91
Pitched, 400 mm loft insulation            25
Pitched, 12 mm loft insulation             24
Pitched, 0 mm loft insulation               6
Thatched                                    4
(another dwelling above)                    4
Thatched, with additional insulation        2
Pitched, 400+  mm loft insulation           2
Roof room(s), thatched                      1
Name: description, dtype: int64
```

Roof description can also be classified as 2 kinds of data

1. Categorical like Roof room(s), no insulation
2. Continuous values loft insulation e.g. 200mm

Similarly, we'll start with continuous data first

In [128]:
```
loft_df = ree_df[ree_df['description'].str.contains('loft insulation')]
loft_df['thickness'] = loft_df['description'].str.extract(r'(\d+)').astype('fl
```

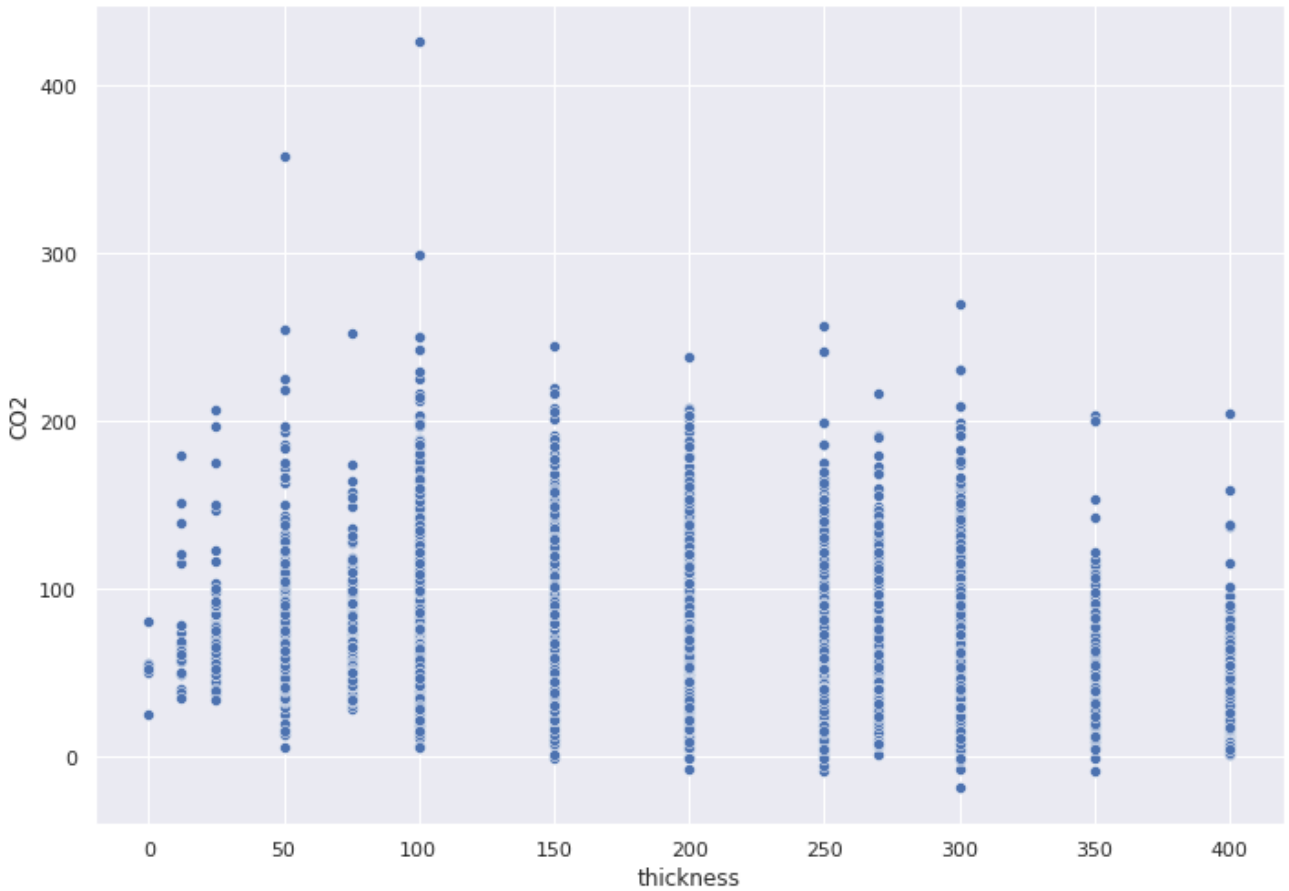
```
/tmp/ipykernel_751989/3337928212.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  loft_df['thickness'] = loft_df['description'].str.extract(r'(\d+)').astype
('float')
```

In [129]:
```
loft_df.thickness.value_counts()
```

Out[129]:
```
200.0    3761
300.0    3136
150.0    2807
250.0    2737
100.0    2219
270.0    1875
50.0      443
350.0     313
75.0      299
400.0     279
25.0       91
12.0       24
0.0         6
Name: thickness, dtype: int64
```

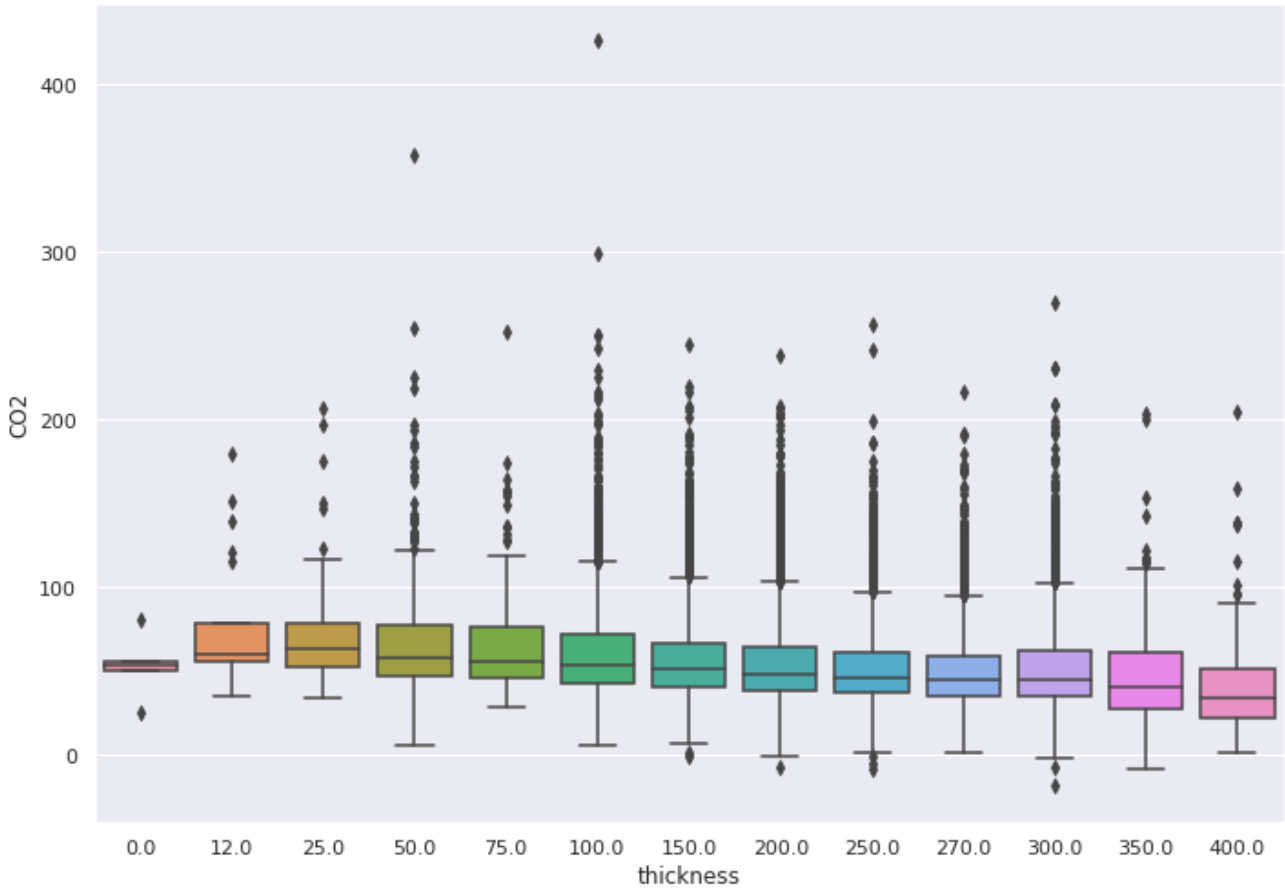
In [130]:
```
sns.scatterplot(data=loft_df, x='thickness',y='CO2')
```

Out[130]:
```
<AxesSubplot: xlabel='thickness', ylabel='CO2'>
```



In [131]:
```
sns.boxplot(data=loft_df, x='thickness',y='CO2')
```

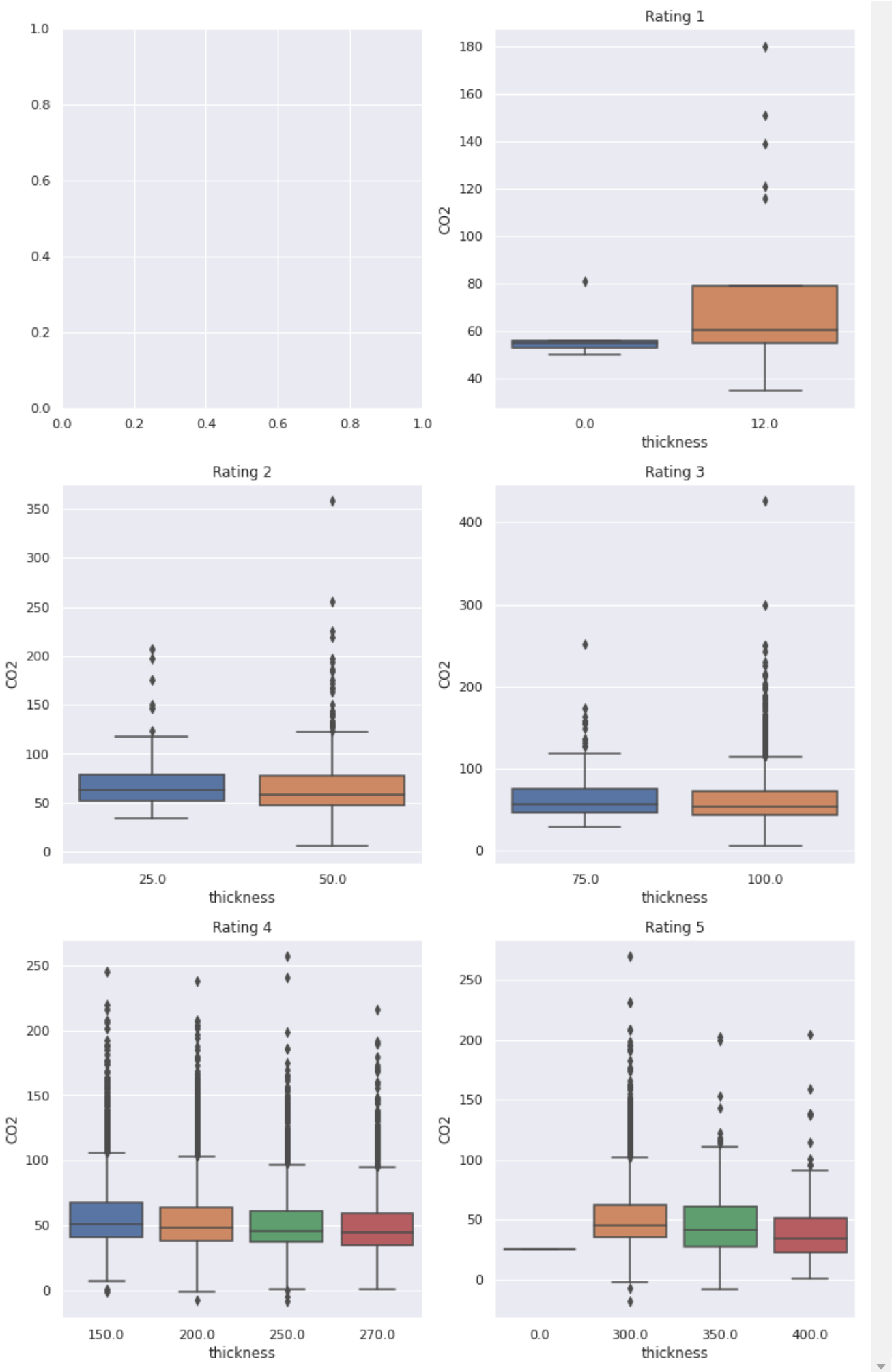
Out[131]:
```
<AxesSubplot: xlabel='thickness', ylabel='CO2'>
```



Both boxplot and scatter plot shows that there's no correlation. Now wrt to rating

In [132]:

```
fig, ax = plt.subplots(3, 2)
fig.set_size_inches(12, 20)
for i in range(1,6):
    desired_rating = i
    filtered_loft_df = loft_df[loft_df['rating'] == desired_rating]
    ax[i // 2, i % 2].set_title(f'Rating {desired_rating}')
    sns.boxplot(ax=ax[i // 2, i % 2], data=filtered_loft_df, x='thickness',y='
```



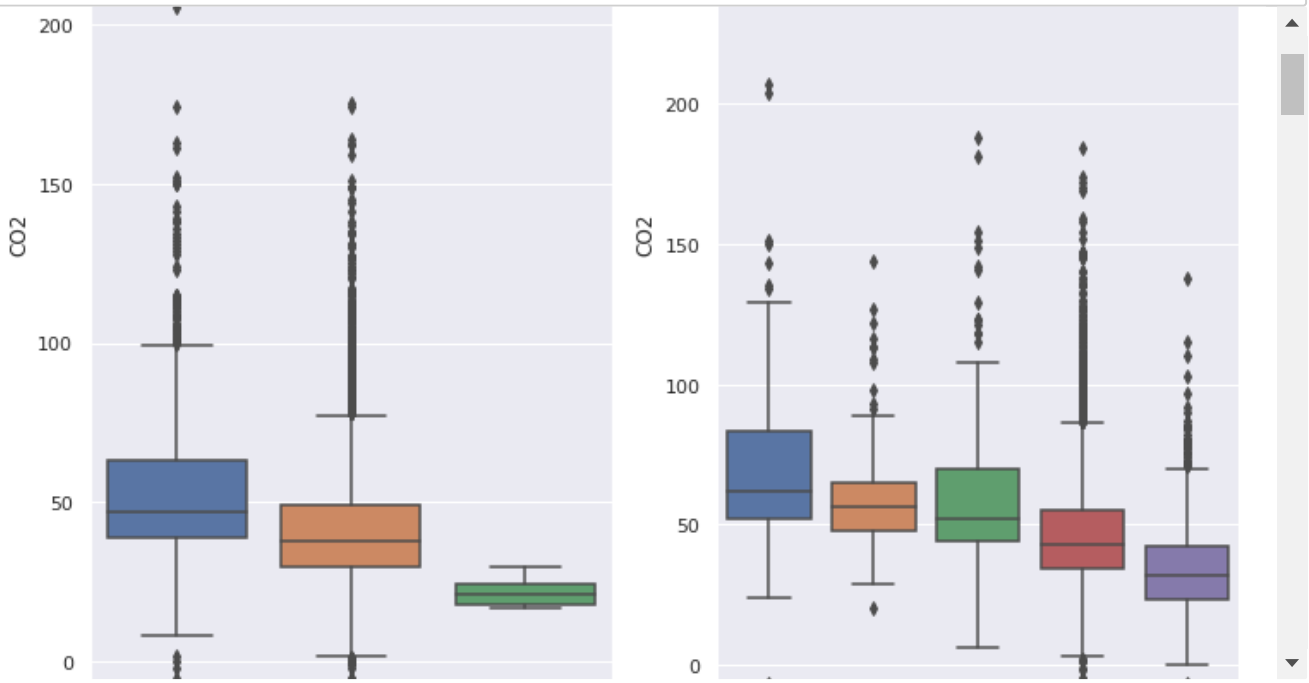
For most of the plots, it shows that within a rating, as thickness increases, mean CO2 decreases.

We can also see that the thickness is generally higher as the rating is higher.

In [133]:
```
r_others_df = ree_df[ree_df['description'].str.contains('loft insulation') ==
```

In [134]:
```
unique_descriptions = r_others_df['description'].unique()
```

In [135]:
```
fig, ax = plt.subplots(math.ceil(len(unique_descriptions) / 2), 2)
fig.set_size_inches(12, 100)
for i in range(len(unique_descriptions)):
    cur = unique_descriptions[i]
    filtered_r_others_df = r_others_df[r_others_df['description'] == cur]
    ax[i // 2, i % 2].set_title(f'{cur}')
    sns.boxplot(ax=ax[i // 2, i % 2], data=filtered_r_others_df, x='rating',y=
```



Also for most of the roof categories, higher rating means generally lower mean CO2 ratings

Generally, the range of CO2 values across the categories are roughly the same besides the outliers.

Algorithm Challenge 3: Build and algorithm to find correlations between construction age band vs current energy efficiency and current emissions (T.CO2/yr)

The approach to this is to create a linear regression model using the age of the building and to find out the relationship between current energy efficiency and current emissions.

In [136]:
```
dataset['Part 1 Construction Age Band'].value_counts()
```

Out[136]:
```
before 1919     36103
1950-1964       27750
1965-1975       24157
1930-1949       14222
1984-1991       10235
2003-2007        9503
1976-1983        9150
1992-1998        9076
1999-2002        5891
2008 onwards     5082
1919-1929        3898
Name: Part 1 Construction Age Band, dtype: int64
```

In [137]:
```
age_of_build = dataset.groupby(['Part 1 Construction Age Band'])['Current ener
```

```
/tmp/ipykernel_751989/1828837144.py:1: FutureWarning: Indexing with multiple
keys (implicitly converted to a tuple of keys) will be deprecated, use a list
instead.
  age_of_build = dataset.groupby(['Part 1 Construction Age Band'])['Current e
nergy efficiency rating','Current Emissions (T.CO2/yr)'].mean().reset_index()
```

In [138]:
```
age_of_build = age_of_build.dropna(subset=['Part 1 Construction Age Band'])
```

In [139]:
```
age_of_build['Part 1 Construction Age Band'] = age_of_build['Part 1 Constructi
age_of_build['Part 1 Construction Age Band'] = age_of_build['Part 1 Constructi
```

In [140]:
```
age_of_build.head(20)
```

Out[140]:

	Part 1 Construction Age Band	Current energy efficiency rating	Current Emissions (T.CO2/yr)
0	1919-1929	62.244228	4.805747
1	1930-1949	64.988258	4.264428
2	1950-1964	65.663387	3.973697
3	1965-1975	66.230327	4.079704
4	1976-1983	67.528634	3.858721
5	1984-1991	69.085491	3.484475
6	1992-1998	71.311040	3.473204
7	1999-2002	72.708708	3.404447
8	2003-2007	76.565927	2.898558
9	2008-2022	78.443133	2.598642
10	1850-1919	59.158740	5.644251

In [141]:
```
start_end_df = age_of_build['Part 1 Construction Age Band'].str.extract(r'((\d
```

In [142]:
```
start_end_df
```

Out[142]:

	0	1	2
0	1919-1929	1919	1929
1	1930-1949	1930	1949
2	1950-1964	1950	1964
3	1965-1975	1965	1975
4	1976-1983	1976	1983
5	1984-1991	1984	1991
6	1992-1998	1992	1998
7	1999-2002	1999	2002
8	2003-2007	2003	2007
9	2008-2022	2008	2022
10	1850-1919	1850	1919

In [143]:
```
def get_mid_year(x):
#     print(x)
    start = int(x[1])
    end = int(x[2])
    return start + (end - start) // 2
```

In [144]:
```
age_of_build_start_end = pd.merge(start_end_df, age_of_build,left_on=0, right_
age_of_build_start_end['mid_year'] = age_of_build_start_end.apply(get_mid_year
age_of_build_start_end = age_of_build_start_end.sort_values(by='mid_year', asc
```

In [145]:
```
age_of_build_start_end['est_building_age'] = 2022-age_of_build_start_end['mid_
```

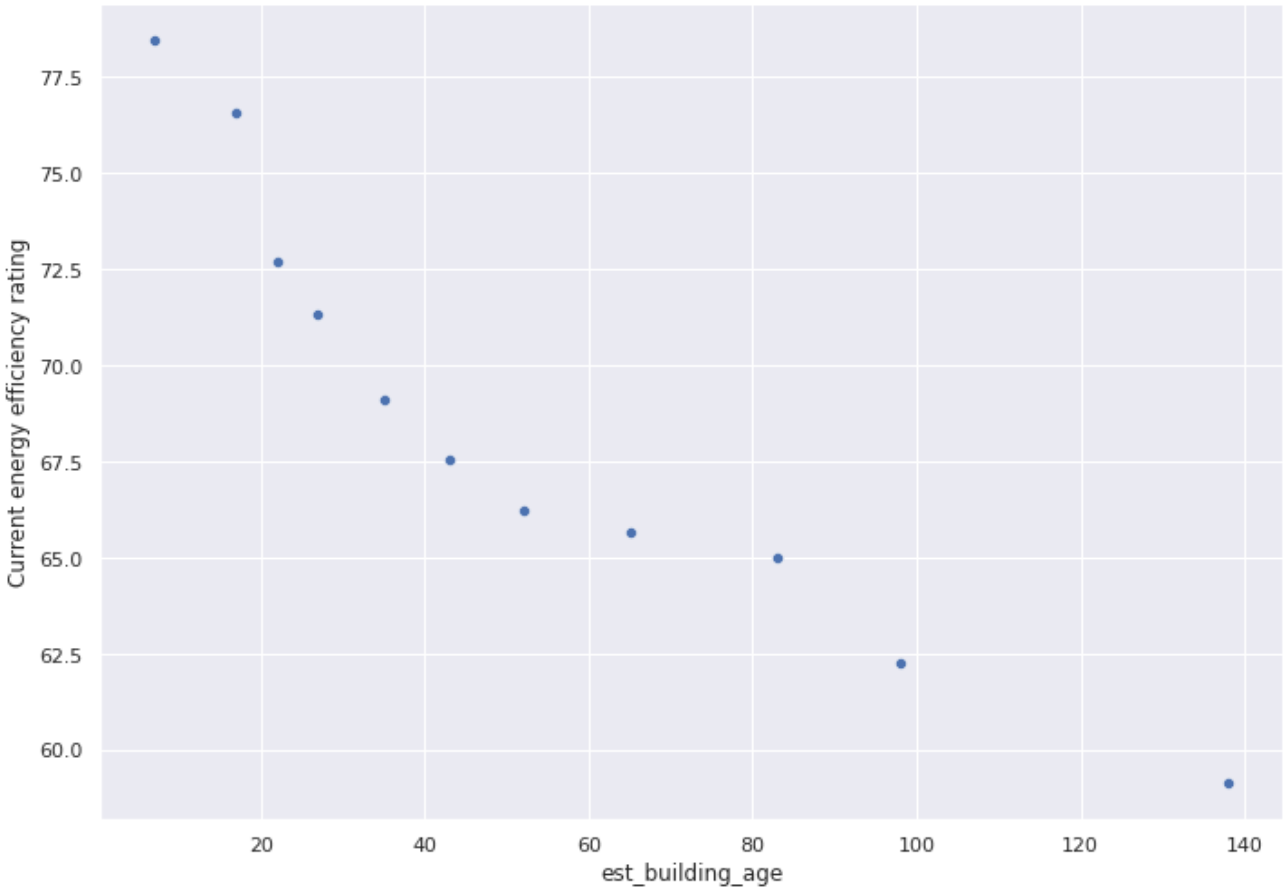
In [146]:
```
age_of_build_start_end
```

Out[146]:

	1	2	Part 1 Construction Age Band	Current energy efficiency rating	Current Emissions (T.CO2/yr)	mid_year	est_building_age
0	1850	1919	1850-1919	59.158740	5.644251	1884	138
1	1919	1929	1919-1929	62.244228	4.805747	1924	98
2	1930	1949	1930-1949	64.988258	4.264428	1939	83
3	1950	1964	1950-1964	65.663387	3.973697	1957	65
4	1965	1975	1965-1975	66.230327	4.079704	1970	52
5	1976	1983	1976-1983	67.528634	3.858721	1979	43
6	1984	1991	1984-1991	69.085491	3.484475	1987	35
7	1992	1998	1992-1998	71.311040	3.473204	1995	27
8	1999	2002	1999-2002	72.708708	3.404447	2000	22
9	2003	2007	2003-2007	76.565927	2.898558	2005	17
10	2008	2022	2008-2022	78.443133	2.598642	2015	7

In [147]:
```
sns.scatterplot(data=age_of_build_start_end, x='est_building_age', y='Current
```

Out[147]:
```
<AxesSubplot: xlabel='est_building_age', ylabel='Current energy efficiency ra
ting'>
```

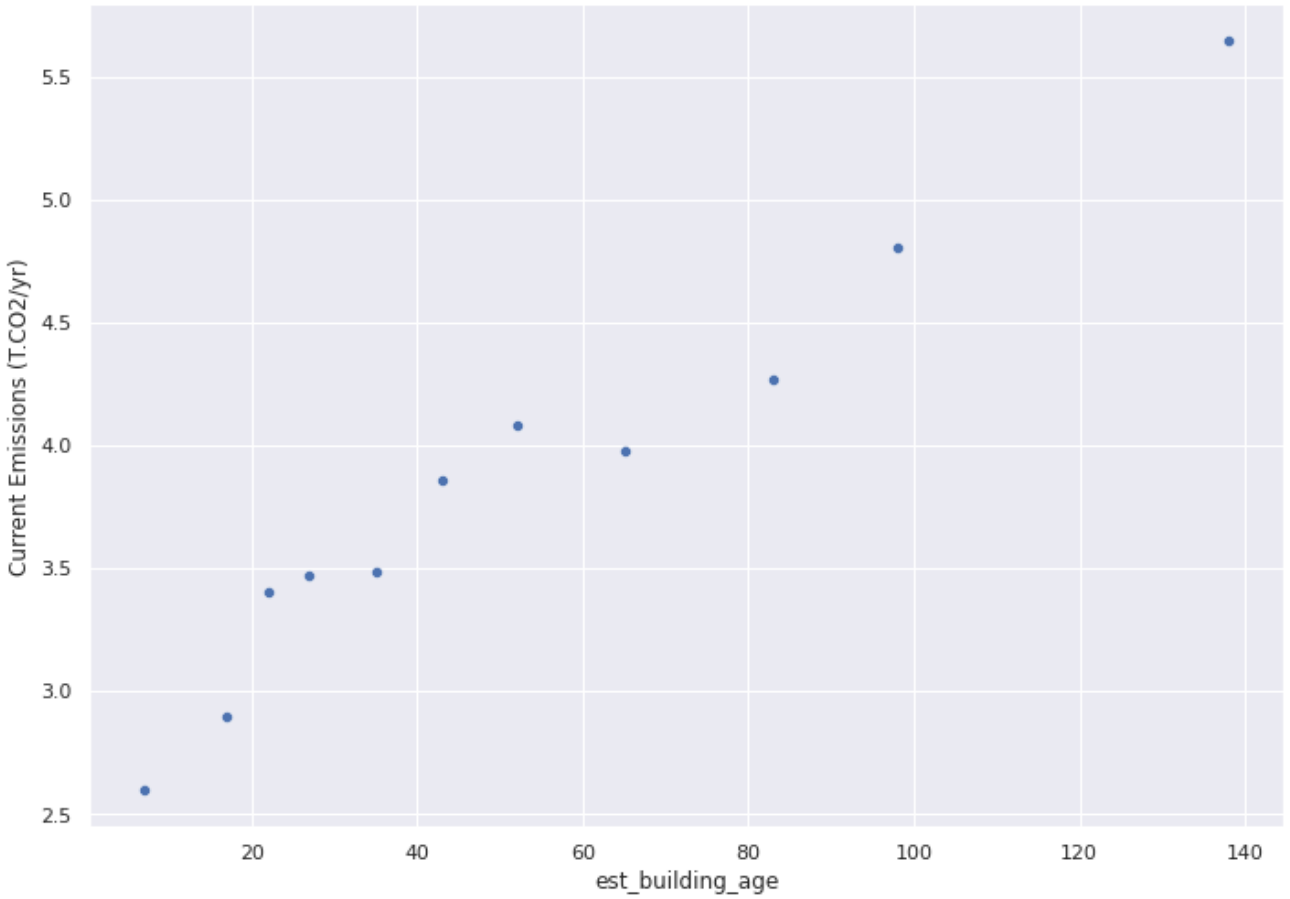


Visually, this shows a strong correlation between the energy efficiecny rating and the building's age and hence the Construction Age Band.

Older the building, the lower the energy efficiency rating

In [148]:
```
sns.scatterplot(data=age_of_build_start_end, x='est_building_age', y='Current
```

Out[148]:
```
<AxesSubplot: xlabel='est_building_age', ylabel='Current Emissions (T.CO2/y
r)'>
```



Older the building, the higher the emissions of CO2 yearly.

Being visually strongly negatively correlated with current energy efficiency shows that the data is pretty reliable.

Thus, we can attempt to build an algorithm that predicts the expected energy efficiencies and and CO2 emissions

In [149]:
```
from sklearn.linear_model import LinearRegression
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_validate
```

In [150]:
```
dependent_var = 'est_building_age'
```

In [151]:
```
# Process main dataset
part3_df = dataset[['Property_UPRN','Postcode','POST_TOWN', 'Part 1 Constructi
```

In [152]:
```
part3_df.head()
```

Out[152]:

	Property_UPRN	Postcode	POST_TOWN	Part 1 Construction Age Band	Current energy efficiency rating	Current Emissions (T.CO2/yr)
0	1.001101e+09	EH4 5EZ	edinburgh	1930-1949	53.0	6.2
1	1.001951e+09	EH7 4HE	edinburgh	1919-1929	66.0	7.7
2	1.000996e+09	EH4 2DL	edinburgh	1965-1975	61.0	4.9
3	1.001257e+09	PH1 1SA	perth	1999-2002	76.0	2.9
4	1.235709e+09	G78 1QN	glasgow	before 1919	79.0	1.5

In [153]:
```
part3_df['Part 1 Construction Age Band'] = part3_df['Part 1 Construction Age B
part3_df['Part 1 Construction Age Band'] = part3_df['Part 1 Construction Age B
```

```
/tmp/ipykernel_751989/1629447571.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  part3_df['Part 1 Construction Age Band'] = part3_df['Part 1 Construction Ag
e Band'].str.replace('2008 onwards', '2008-2022')
/tmp/ipykernel_751989/1629447571.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  part3_df['Part 1 Construction Age Band'] = part3_df['Part 1 Construction Ag
e Band'].str.replace('before 1919', '1850-1919')
```

In [154]:
```
part3_df = pd.merge(start_end_df, part3_df,left_on=0, right_on='Part 1 Constru
part3_df['mid_year'] = part3_df.apply(get_mid_year,axis=1)
part3_df = part3_df.sort_values(by='mid_year', ascending=True).reset_index(dro
part3_df['est_building_age'] = 2022-part3_df['mid_year']
```

In [155]:
```
part3_df.head()
```

Out[155]:

	1	2	Property_UPRN	Postcode	POST_TOWN	Part 1 Construction Age Band	Current energy efficiency rating	Current Emissions (T.CO2/yr)	mid_year
0	1850	1919	1.001135e+09	EH4 1QA	edinburgh	1850-1919	71.0	5.5	1884
1	1850	1919	1.000055e+09	G12 9YD	glasgow	1850-1919	78.0	3.3	1884
2	1850	1919	1.000062e+09	G42 9RH	glasgow	1850-1919	13.0	6.5	1884
3	1850	1919	1.000098e+09	ML11 9DU	lanark	1850-1919	56.0	3.7	1884
4	1850	1919	1.000170e+09	EH21 6ER	musselburgh	1850-1919	67.0	3.2	1884

In [156]:
```
target_col = 'Current Emissions (T.CO2/yr)'
x = part3_df[dependent_var].to_numpy().reshape(-1, 1)
y = part3_df[target_col].to_numpy()
```

In [157]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20, shuf
```

In [158]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=
```

In [159]:
```
# Create two models: Quadratic and linear regression
polyreg = make_pipeline(PolynomialFeatures(2), LinearRegression(fit_intercept=
linreg = LinearRegression()
```

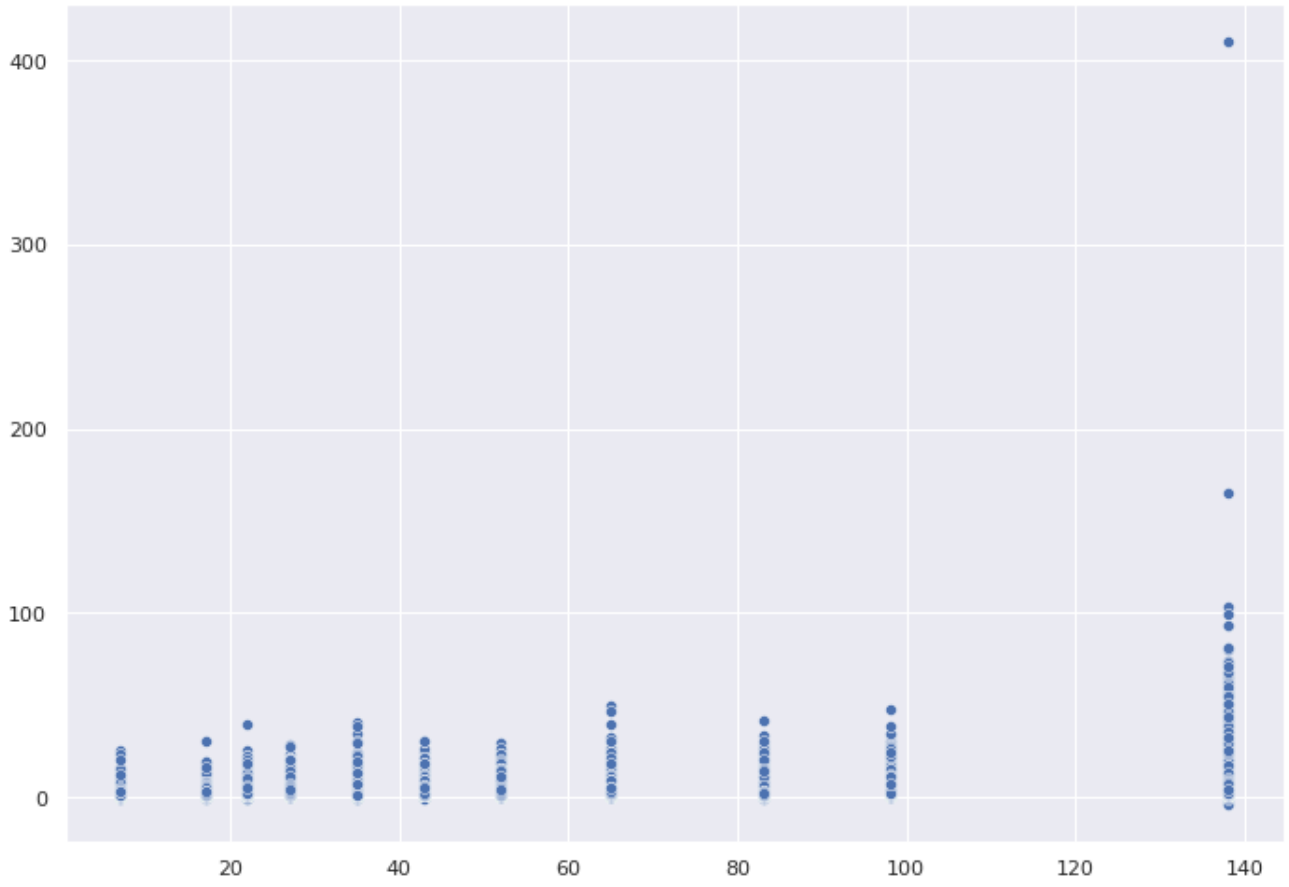
In [160]:
```
scoring = "neg_root_mean_squared_error"
polyscores = cross_validate(polyreg, X_train, y_train, scoring=scoring, return
linscores = cross_validate(linreg, X_train, y_train, scoring=scoring, return_e
```

In [161]:
```
# Which one is better? Linear and polynomial
print(linscores["test_score"].mean())
print(polyscores["test_score"].mean())
print(linscores["test_score"].mean() - polyscores["test_score"].mean())
```

```
-3.598434801556503
-3.5939137743194225
-0.004521027237080588
```

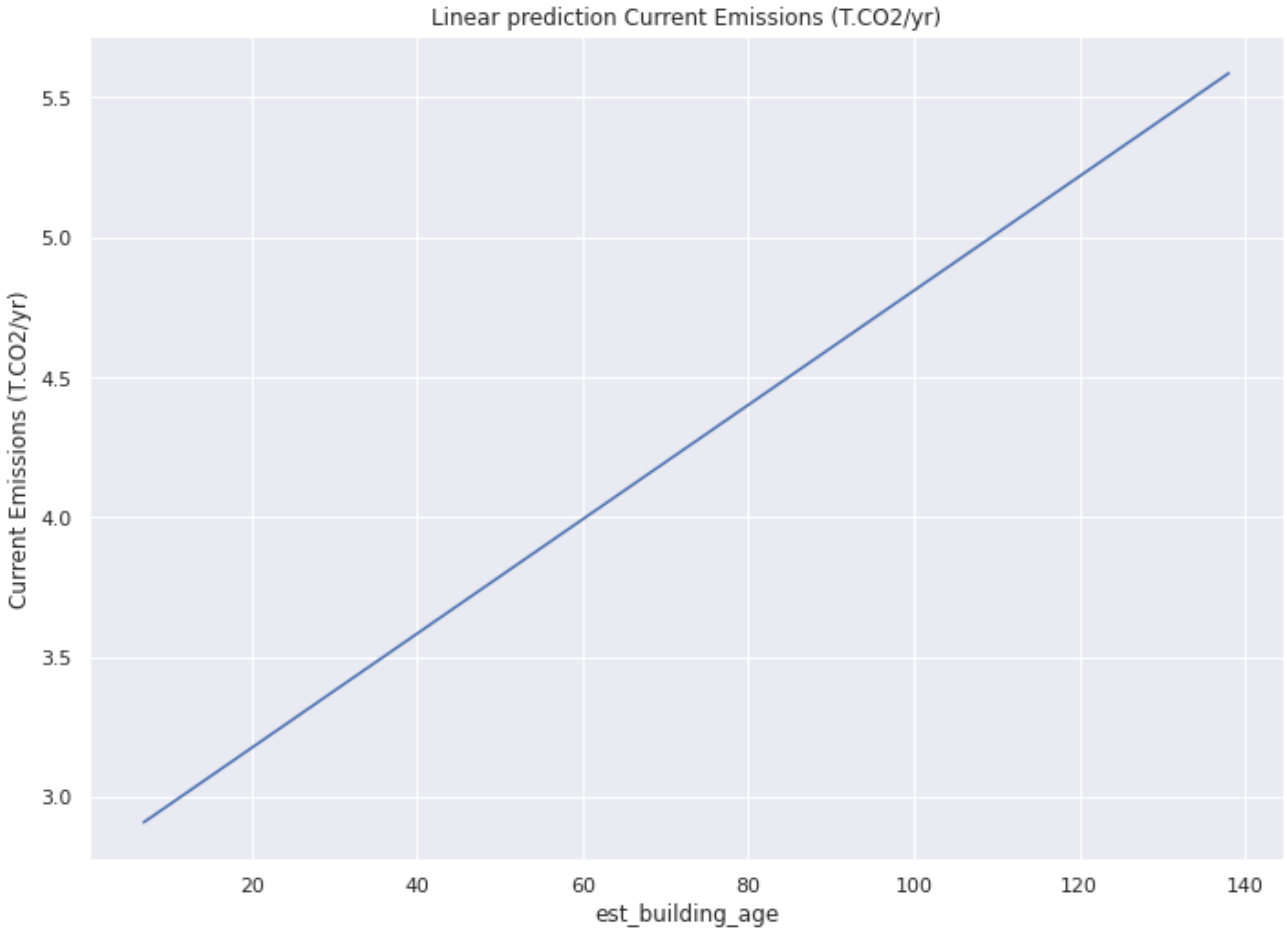
In [162]:
```
sns.scatterplot(x=part3_df[dependent_var].to_numpy(), y= y)
```

Out[162]: <AxesSubplot: >



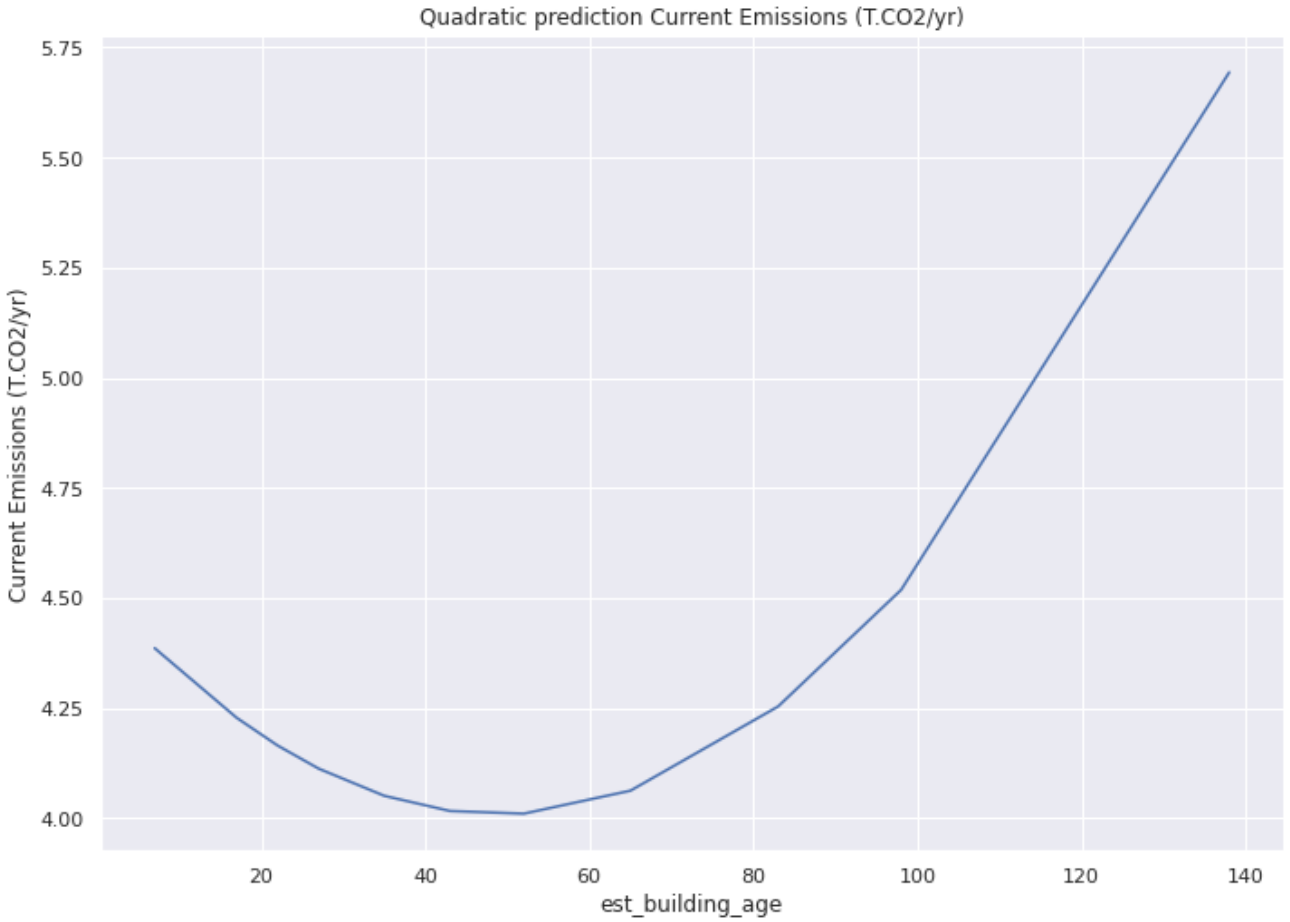
In [163]:
```
plt.title(f"Linear prediction {target_col}")
plt.xlabel(dependent_var)
plt.ylabel(target_col)
plt.plot(x, linscores["estimator"][0].predict(x))
```

Out[163]:
```
[<matplotlib.lines.Line2D at 0x7fca166c1630>]
```



In [164]:
```
plt.title(f"Quadratic prediction {target_col}")
plt.xlabel(dependent_var)
plt.ylabel(target_col)
plt.plot(x, polyscores["estimator"][0].predict(x))
```

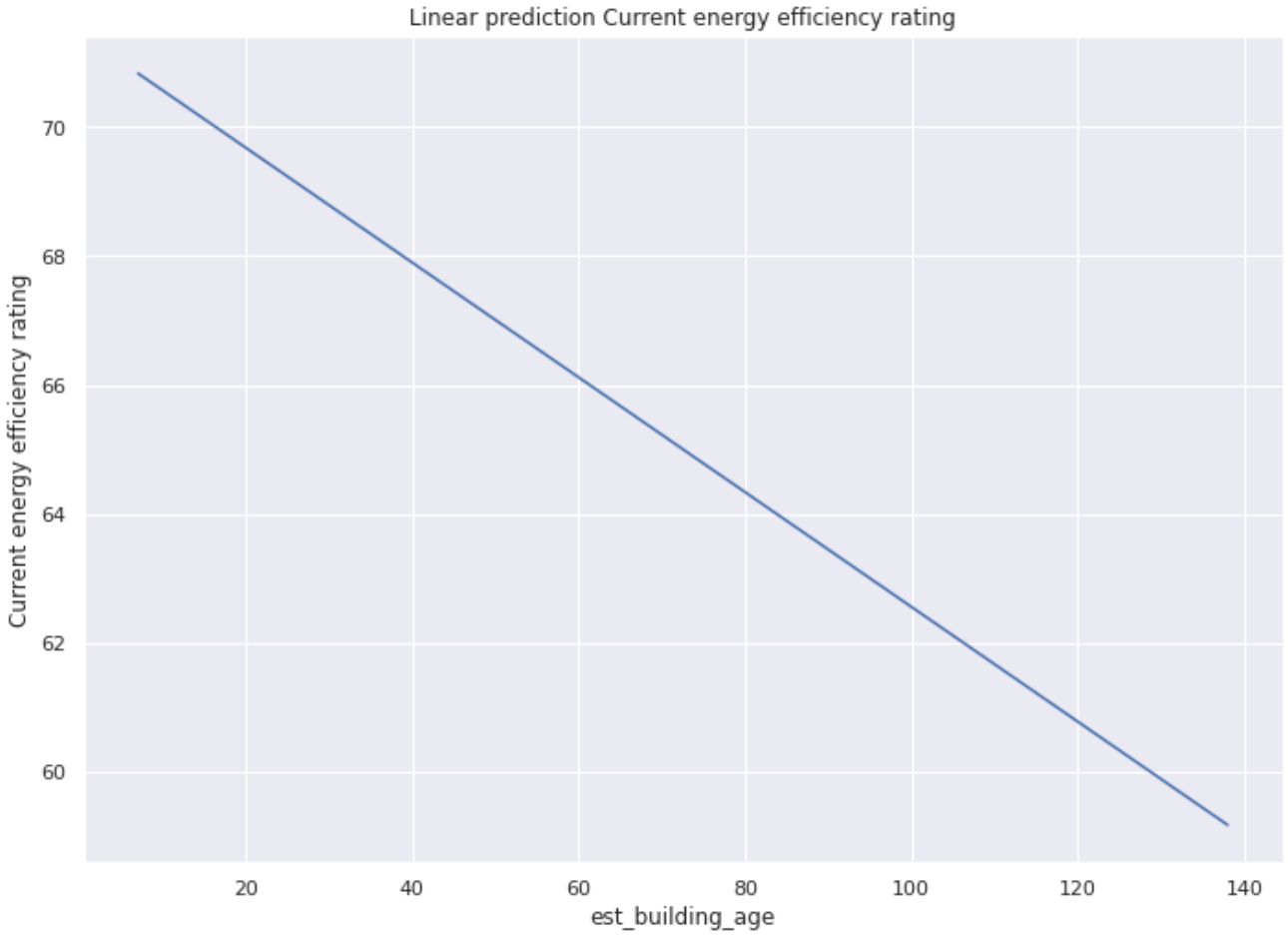
Out[164]:
```
[<matplotlib.lines.Line2D at 0x7fca16442020>]
```



In [165]:
```
target_col = 'Current energy efficiency rating'
x = part3_df[dependent_var].to_numpy().reshape(-1, 1)
y = part3_df[target_col].to_numpy()
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20, shuf
# Create two models: Quadratic and linear regression
polyreg = make_pipeline(PolynomialFeatures(2), LinearRegression(fit_intercept=
linreg = LinearRegression()
scoring = "neg_root_mean_squared_error"
polyscores = cross_validate(polyreg, X_train, y_train, scoring=scoring, return
linscores = cross_validate(linreg, X_train, y_train, scoring=scoring, return_e
```

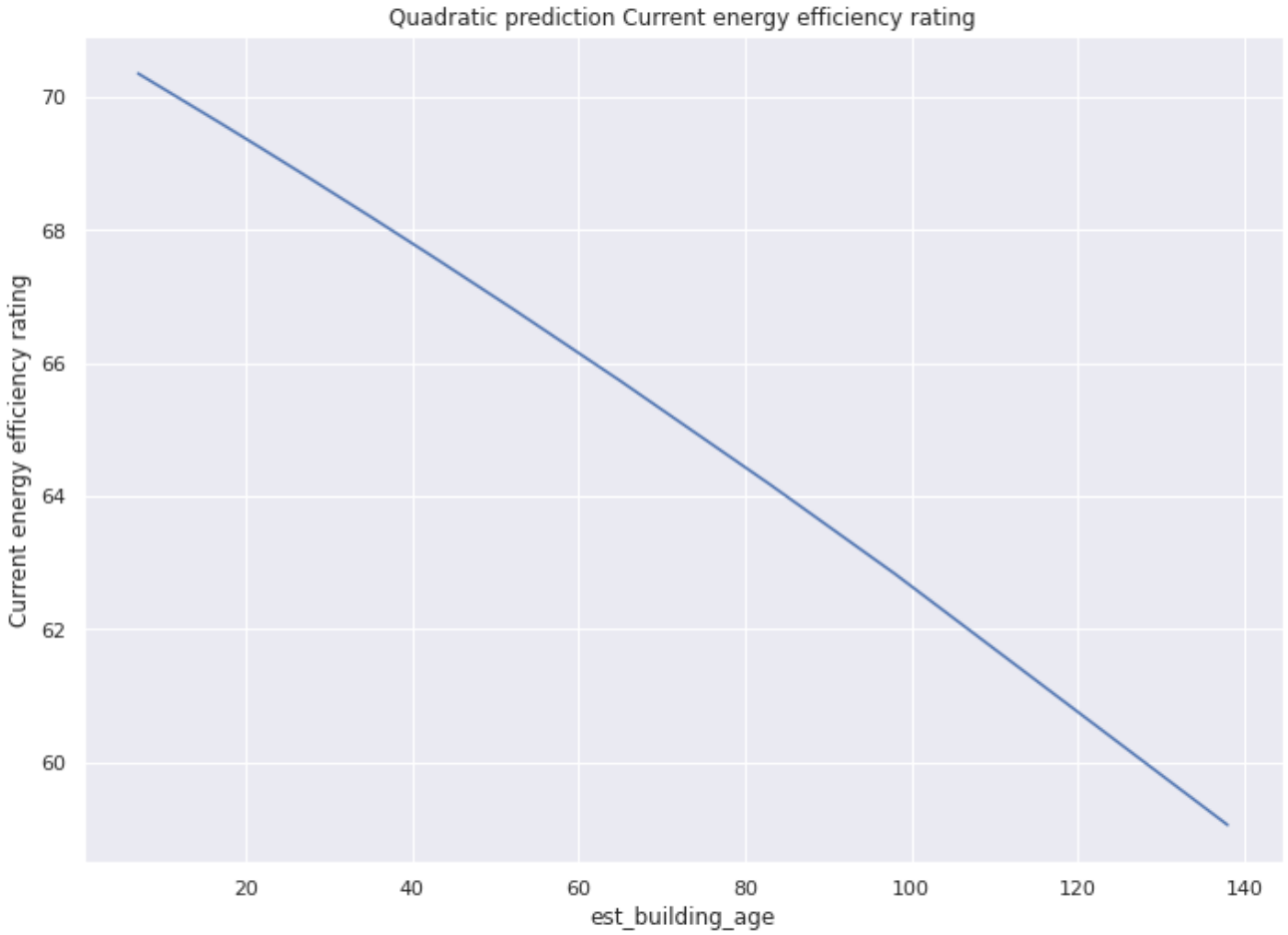
In [166]:
```
plt.title(f"Linear prediction {target_col}")
plt.xlabel(dependent_var)
plt.ylabel(target_col)
plt.plot(x, linscores["estimator"][0].predict(x))
```

Out[166]:
```
[<matplotlib.lines.Line2D at 0x7fca145d95a0>]
```



In [167]:
```
plt.title(f"Quadratic prediction {target_col}")
plt.xlabel(dependent_var)
plt.ylabel(target_col)
plt.plot(x, polyscores["estimator"][0].predict(x))
```

Out[167]: [<matplotlib.lines.Line2D at 0x7fca1a451750>]



For energy efficiency, the older the building, energy efficiency (higher the better) declines by 0.167 units. Also, the older the building, the higher the

emissions (lower the better), by 0.028 units.

As expected, the newer the building, the better the energy efficiency; the newer the building, emissions are lower.

Algorithm Challenge 4 Build an algorithm that takes as input the characteristics of a building (any field of the dataset) and outputs recommendations on the elements of the house to be modified to improve its energy performance - (15 points)

Input for 4: All fields besides target

Input for 5: All fields besides target and costs fields

Input for 6: All fields besides target

Target for 4: elements of the house to be modified to improve its energy performance

Target for 5: Total current energy costs over 3 years (£)

Target for 6: elements of the house should be modified to most effectively decrease the total energy cost of the building over a 3-year period

Challenge 4 aims to recommend to the user probably the best sets of features a building should have in order to achieve maximal energy performance.

For example, we can see that some wall descriptions are associated with specific wall energy efficiencies, so we can recommend the best rated wall descriptions (Good and Very Good)

We'll assume that any recommendations is "feasible" in the current year 2022, so it doesn't have to be recommended based on the construction age band.

Roofs

Based on the algorithm 2, greater loft insulation is correlated with better ROOF_ENERGY_EFF, regardless of whether roof is pitched or thatched, or roof room(s).

For decent rating of good/very good, loft insulation should be at least 150mm and Average thermal transmittance at most 0.3

In [259]:
```
ree_df = general_preprocess(dataset, 'ROOF_DESCRIPTION', 'ROOF_ENERGY_EFF')
```

In [260]:
```
ree_df.head()
```

Out[260]:

	description	rating	CO2	CO2_norm	combi
0	Pitched, 25 mm loft insulation	2	66.0	1.512097	3.512097
1	Pitched, insulated	3	44.0	1.364247	4.364247
2	Roof room(s), insulated	4	44.0	1.364247	5.364247
3	Roof room(s), no insulation	1	44.0	1.364247	2.364247
4	Pitched, 250 mm loft insulation	4	31.0	1.276882	5.276882

In [265]:
```
# sorted_ree_df = ree_df.sort_values(by='rating', ascending=False).reset_index
r_desc_rating = ree_df.groupby(['description'])['rating'].mean().reset_index(n
r_desc_rating = r_desc_rating.sort_values(by='Mean', ascending=False).reset_in
mean_ROOF_rate = r_desc_rating.copy()
```

In [266]:
```
mean_ROOF_rate.head(50)
```

Out[266]:

	description	Mean
0	(another dwelling above)	5.000000
1	Average thermal transmittance 0.12 W/m?K	5.000000
2	Pitched, 350 mm loft insulation	5.000000
3	Pitched, 400 mm loft insulation	5.000000
4	Pitched, 400+ mm loft insulation	5.000000
5	Pitched, 400+ mm loft insulation	5.000000
6	Average thermal transmittance 0.14 W/m²K	5.000000
7	Average thermal transmittance 0.14 W/m?K	5.000000
8	Average thermal transmittance 0.13 W/m²K	5.000000
9	Average thermal transmittance 0.13 W/m?K	5.000000
10	Average thermal transmittance 0.12 W/m²K	5.000000
11	Average thermal transmittance 0.11 W/m²K	5.000000
12	Average thermal transmittance 0.05 W/m²K	5.000000
13	Average thermal transmittance 0.11 W/m?K	5.000000
14	Average thermal transmittance 0.10 W/m²K	5.000000
15	Average thermal transmittance 0.1 W/m²K	5.000000
16	Average thermal transmittance 0.09 W/m²K	5.000000
17	Average thermal transmittance 0.09 W/m?K	5.000000
18	Average thermal transmittance 0.08 W/m²K	5.000000
19	Average thermal transmittance 0.08 W/m?K	5.000000
20	Average thermal transmittance 0.07 W/m²K	5.000000
21	Average thermal transmittance 0.06 W/m²K	5.000000
22	Pitched, 300 mm loft insulation	5.000000
23	Thatched, with additional insulation	4.818182
24	Average thermal transmittance 0.21 W/m²K	4.000000
25	Pitched, 150 mm loft insulation	4.000000
26	Roof room(s), thatched	4.000000
27	Average thermal transmittance 0.15 W/m²K	4.000000
28	Average thermal transmittance 0.16 W/m²K	4.000000
29	Average thermal transmittance 0.17 W/m²K	4.000000
30	Average thermal transmittance 0.18 W/m²K	4.000000
31	Average thermal transmittance 0.19 W/m²K	4.000000
32	Pitched, 270 mm loft insulation	4.000000
33	Pitched, 250 mm loft insulation	4.000000
34	Average thermal transmittance 0.22 W/m²K	4.000000
35	Pitched, 200 mm loft insulation	4.000000
36	Average thermal transmittance 0.29 W/m²K	4.000000
37	Average thermal transmittance 0.23 W/m²K	4.000000

	description	Mean
38	Average thermal transmittance 0.24 W/m²K	4.000000
39	Average thermal transmittance 0.25 W/m²K	4.000000
40	Average thermal transmittance 0.20 W/m²K	4.000000
41	Roof room(s), insulated	3.995510
42	Pitched, insulated	3.800125
43	Flat, insulated	3.508031
44	Thatched	3.400000
45	Pitched, insulated at rafters	3.349931
46	Average thermal transmittance 0.32 W/m²K	3.000000
47	Pitched, 75 mm loft insulation	3.000000
48	Average thermal transmittance 0.31 W/m²K	3.000000
49	Average thermal transmittance 0.38 W/m²K	3.000000

Walls

Based on the algorithm 1, energy ratings are better rated as wall thermal transmittance is lower

Cavity wall that's filled / has insulation is better rated than with no insulation. Other materials also exhibit the same insulation correlation.

Internal and external insulation has no siginificant difference but generally internal insulation is better than external insulation.

Thus based on the below countplots, we can see wall insulation has correspondingly high energy ratings

For a decent rating, average thermal transmittance should be at least 0.35 W/m²K

In [244]:
```
# sorted_ree_df = ree_df.sort_values(by='rating', ascending=False).reset_index
e_desc_rating = ee_df.groupby(['description'])['rating'].mean().reset_index(na
e_desc_rating = e_desc_rating.sort_values(by='Mean', ascending=False).reset_in
mean_WALL_rate = e_desc_rating.copy()
```

In [172]:
```
e_desc_rating.head(100)
```

84	Average thermal transmittance 0.71 W/m²K	3.000000
85	Average thermal transmittance 0.70 W/m²K	3.000000
86	Average thermal transmittance 0.68 W/m²K	3.000000
87	Average thermal transmittance 0.67 W/m²K	3.000000
88	Average thermal transmittance 0.66 W/m²K	3.000000
89	Average thermal transmittance 0.65 W/m²K	3.000000
90	Average thermal transmittance 0.64 W/m²K	3.000000
91	Average thermal transmittance 0.63 W/m²K	3.000000
92	Average thermal transmittance 0.62 W/m²K	3.000000
93	Average thermal transmittance 0.77 W/m²K	3.000000
94	Solid brick, as built, partial insulation	2.999867
95	Sandstone or limestone, as built, partial insu...	2.999499
96	Park home wall, as built	2.810345

In [173]:
```
ee_others_df = ee_df[ee_df['description'].str.contains('thermal transmittance'
unique_descriptions = ee_others_df['description'].unique()
```

In [174]:
```
ee_others_df.head()
```

Out[174]:

	description	rating	CO2	CO2_norm	combi
0	Cavity wall, as built, no insulation	2	66.0	1.512097	3.512097
1	Cavity wall, filled cavity	3	44.0	1.364247	4.364247
2	Cavity wall, as built, partial insulation	3	68.0	1.525538	4.525538
3	Cavity wall, filled cavity	3	68.0	1.525538	4.525538
4	Cavity wall, as built, insulated	4	31.0	1.276882	5.276882

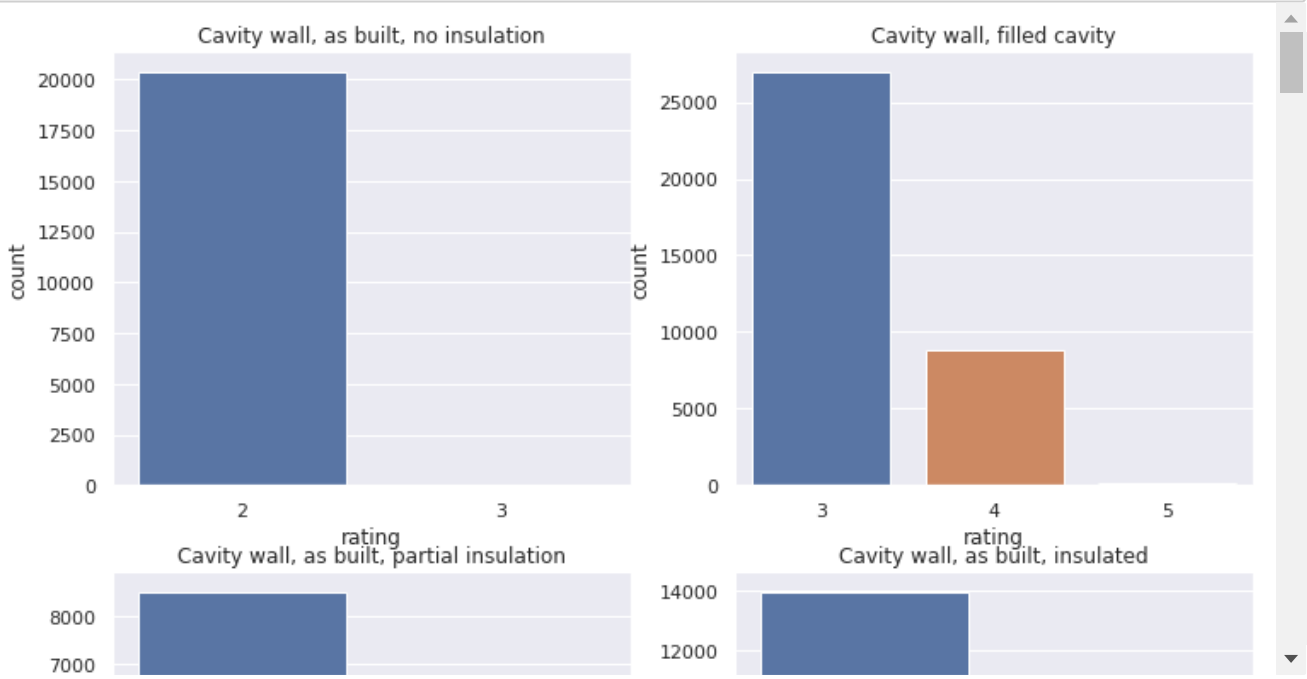
In [175]:
```
import math
```

In [176]:
```
ee_others_df[ee_others_df['description'] == 'Cavity wall, as built, no insulat
```

Out[176]:

rating	CO2	CO2_norm	combi
2	58.990701	1.464991	3.464991
3	44.000000	1.364247	4.364247

In [177]:
```
# These diagrams show which wall descriptions have greater occurance of high r
fig, ax = plt.subplots(math.ceil(len(unique_descriptions) / 2), 2)
fig.set_size_inches(12, 100)
for i in range(len(unique_descriptions)):
    cur = unique_descriptions[i]
    filtered_others_df = ee_others_df[ee_others_df['description'] == cur]
    ax[i // 2, i % 2].set_title(f'{cur}')
    sns.countplot(ax=ax[i // 2, i % 2], data=filtered_others_df, x='rating')
```



In [178]:
```
df = ee_others_df
```

In [179]:
```
df_desc_rating = df.groupby(['description'])['rating'].mean().reset_index(name
df_desc_rating = df_desc_rating.sort_values(by='Mean', ascending=False).reset_
mean_WALL_CAT_rate = df_desc_rating.copy()
df_desc_rating.iloc[0:70]
```

Out[179]:

	description	Mean
0	Cavity wall, filled cavity and external insula...	4.751773
1	Cavity wall, filled cavity and internal insula...	4.731343
2	System built, as built, insulated	4.441198
3	Solid brick, with internal insulation	4.387889
4	Solid brick, as built, insulated	4.362207
5	Timber frame, as built, insulated	4.351770
6	Granite or whinstone, as built, insulated	4.335476
7	Sandstone or limestone, as built, insulated	4.333100
8	Cavity wall, with internal insulation	4.320413
9	Granite or whinstone, with internal insulation	4.267403
10	Sandstone or limestone, with internal insulation	4.234492
11	Timber frame, with additional insulation	4.195477
12	Cavity wall, as built, insulated	4.161397
13	Granite or whinstone, with external insulation	4.137931
14	Solid brick, with external insulation	4.124195
15	System built, with internal insulation	4.120275
16	System built, with external insulation	4.087906
17	Cavity wall, with external insulation	4.070526
18	Sandstone or limestone, with external insulation	4.038961
19	Cob, as built	3.565217
20	Cavity wall, filled cavity	3.250852
21	Park home wall, with external insulation	3.166667
22	Cavity wall, as built, partial insulation	3.001178
23	Park home wall, with internal insulation	3.000000
24	Granite or whinstone, as built, partial insula...	3.000000
25	Timber frame, as built, partial insulation	3.000000
26	System built, as built, partial insulation	3.000000
27	Solid brick, as built, partial insulation	2.999867
28	Sandstone or limestone, as built, partial insu...	2.999499
29	Park home wall, as built	2.810345
30	Cavity wall, as built, no insulation	2.000639
31	Solid brick, as built, no insulation	1.903434
32	Sandstone or limestone, as built, no insulation	1.849661
33	Granite or whinstone, as built, no insulation	1.796091
34	Timber frame, as built, no insulation	1.488403
35	System built, as built, no insulation	1.251495

FLOOR_DESCRIPTION

Same as walls

Should have thermmal transmittance of less than 0.3 for good or better rating

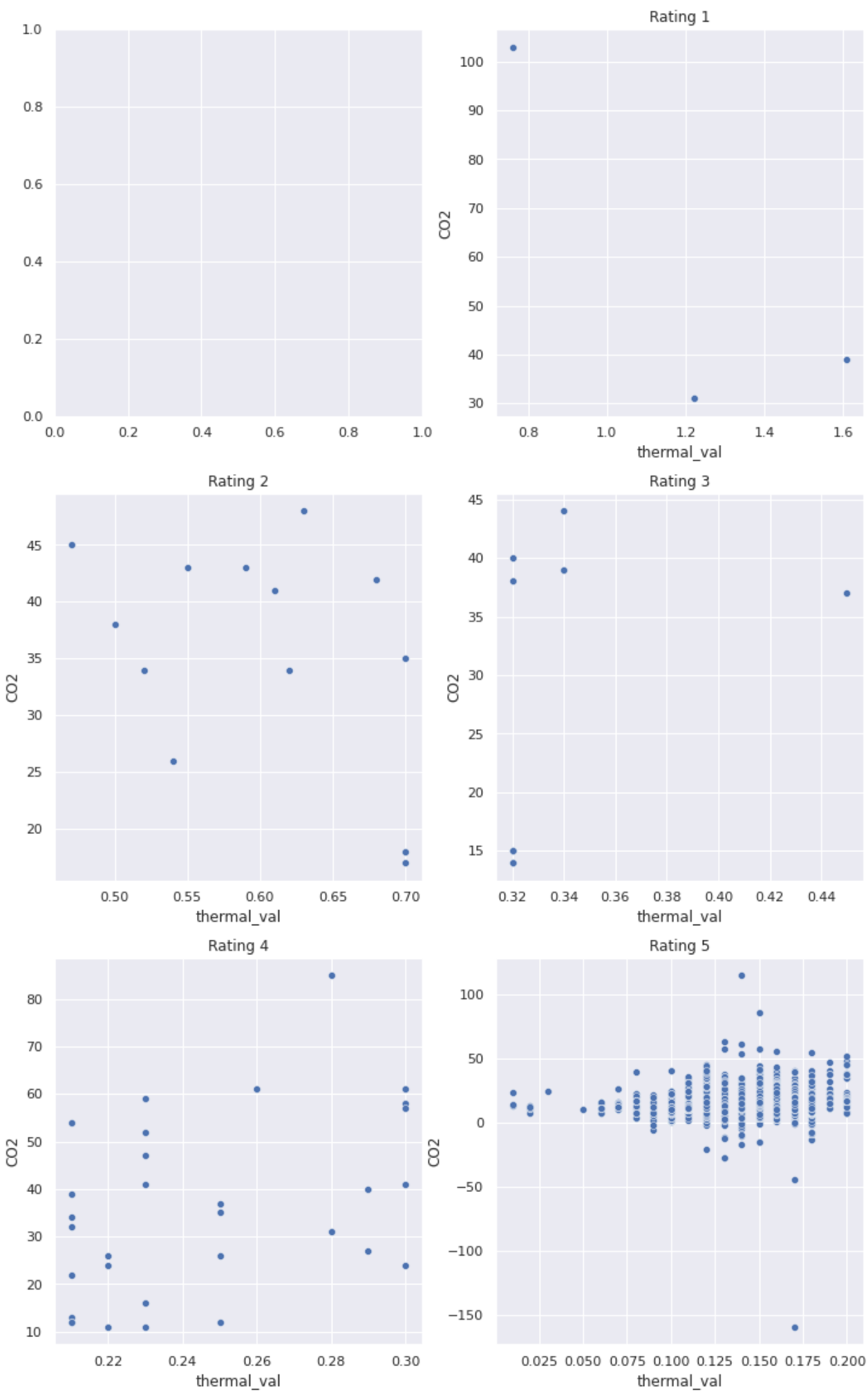
In [180]:
```
# dataset['FLOOR_ENERGY_EFF'] = dataset['FLOOR_ENERGY_EFF'].astype('str')
```

In [181]:
```
f_ee_df = general_preprocess(dataset, 'FLOOR_DESCRIPTION', 'FLOOR_ENERGY_EFF')
```

In [182]:
```
f_ee_df['thermal_val'] = f_ee_df['description'].str.extract(r'(\d+.\d+)').asty
```

In [183]:

```
fig, ax = plt.subplots(3, 2)
fig.set_size_inches(12, 20)
for i in range(1,6):
    desired_rating = i
    filtered_thermal_df = f_ee_df[f_ee_df['rating'] == desired_rating]
    ax[i // 2, i % 2].set_title(f'Rating {desired_rating}')
    sns.scatterplot(ax=ax[i // 2, i % 2], data=filtered_thermal_df, x='thermal
```



In [184]:
```
df = f_ee_df
```

In [185]:
```
df_desc_rating = df.groupby(['description'])['rating'].mean().reset_index(name
df_desc_rating = df_desc_rating.sort_values(by='Mean', ascending=False).reset_
mean_FLOOR_rate = df_desc_rating.copy()
df_desc_rating.iloc[0:70]
```

Out[185]:

	description	Mean
0	Average thermal transmittance 0.01 W/m²K	5.0
1	Average thermal transmittance 0.13 W/m²K	5.0
2	Average thermal transmittance 0.02 W/m²K	5.0
3	Average thermal transmittance 0.20 W/m²K	5.0
4	Average thermal transmittance 0.19 W/m²K	5.0
5	Average thermal transmittance 0.18 W/m²K	5.0
6	Average thermal transmittance 0.18 W/m?K	5.0
7	Average thermal transmittance 0.17 W/m²K	5.0
8	Average thermal transmittance 0.17 W/m?K	5.0
9	Average thermal transmittance 0.16 W/m²K	5.0
10	Average thermal transmittance 0.15 W/m²K	5.0
11	Average thermal transmittance 0.15 W/m?K	5.0
12	Average thermal transmittance 0.14 W/m²K	5.0
13	Average thermal transmittance 0.13 W/m?K	5.0
14	Average thermal transmittance 0.12 W/m²K	5.0
15	Average thermal transmittance 0.11 W/m²K	5.0
16	Average thermal transmittance 0.10 W/m²K	5.0
17	Average thermal transmittance 0.09 W/m²K	5.0
18	Average thermal transmittance 0.08 W/m²K	5.0
19	Average thermal transmittance 0.07 W/m²K	5.0
20	Average thermal transmittance 0.06 W/m²K	5.0
21	Average thermal transmittance 0.05 W/m²K	5.0
22	Average thermal transmittance 0.03 W/m²K	5.0
23	Average thermal transmittance 0.26 W/m²K	4.0
24	Average thermal transmittance 0.30 W/m²K	4.0
25	Average thermal transmittance 0.29 W/m²K	4.0
26	Average thermal transmittance 0.28 W/m²K	4.0
27	Average thermal transmittance 0.22 W/m²K	4.0
28	Average thermal transmittance 0.25 W/m²K	4.0
29	Average thermal transmittance 0.21 W/m²K	4.0
30	Average thermal transmittance 0.23 W/m²K	4.0
31	Average thermal transmittance 0.32 W/m²K	3.0
32	Average thermal transmittance 0.34 W/m²K	3.0
33	Average thermal transmittance 0.45 W/m²K	3.0
34	Average thermal transmittance 0.61 W/m²K	2.0
35	Average thermal transmittance 0.70 W/m²K	2.0

	description	Mean
36	Average thermal transmittance 0.68 W/m²K	2.0
37	Average thermal transmittance 0.63 W/m²K	2.0
38	Average thermal transmittance 0.62 W/m²K	2.0
39	Average thermal transmittance 0.55 W/m²K	2.0
40	Average thermal transmittance 0.59 W/m²K	2.0
41	Average thermal transmittance 0.54 W/m²K	2.0
42	Average thermal transmittance 0.52 W/m²K	2.0
43	Average thermal transmittance 0.50 W/m²K	2.0
44	Average thermal transmittance 0.47 W/m²K	2.0
45	Average thermal transmittance 0.76 W/m²K	1.0
46	Average thermal transmittance 1.22 W/m²K	1.0
47	Average thermal transmittance 1.61 W/m²K	1.0

LIGHTING_DESCRIPTION

Pretty understood that we should be the lowest possible energy lighting.

For a good / very good light energy efficiency rating, it should be Low energy lighting in AT LEAST 45% of fixed outlets

In [186]:
```
light_ee_df = general_preprocess(dataset, 'LIGHTING_DESCRIPTION', 'LIGHTING_EN
```

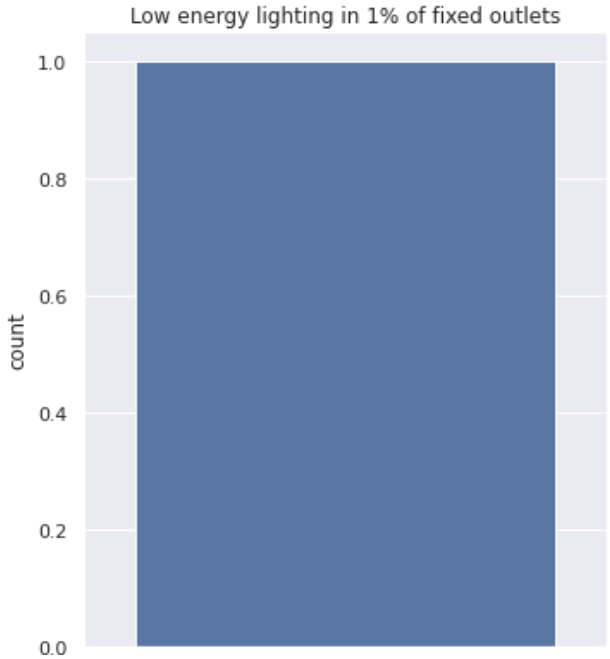
In [187]:
```
light_ee_df.head()
```

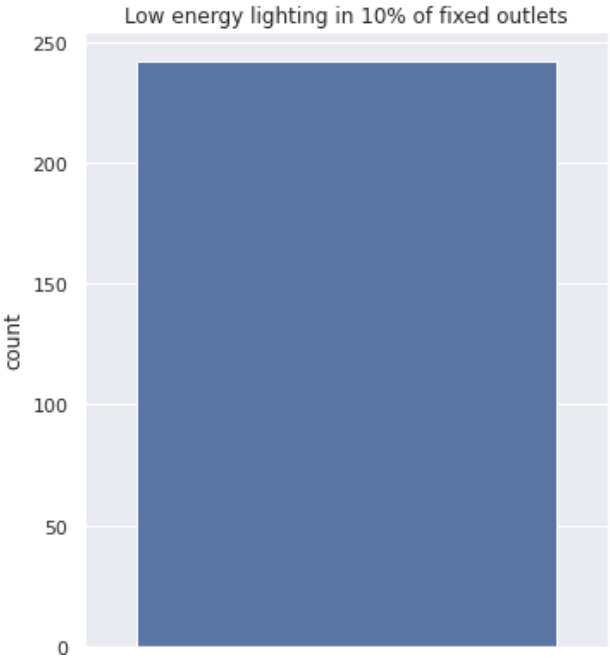
Out[187]:

	description	rating	CO2	CO2_norm	combi
0	Low energy lighting in all fixed outlets	5	66.0	1.512097	6.512097
1	Low energy lighting in all fixed outlets	5	44.0	1.364247	6.364247
2	Low energy lighting in 50% of fixed outlets	4	68.0	1.525538	5.525538
3	Low energy lighting in all fixed outlets	5	31.0	1.276882	6.276882
4	Low energy lighting in 90% of fixed outlets	5	26.0	1.243280	6.243280

In [188]:
```
df = light_ee_df
```

In [189]:
```
unique_descriptions = sorted(df['description'].unique())
fig, ax = plt.subplots(math.ceil(len(unique_descriptions) / 2), 2)
fig.set_size_inches(12, 400)
for i in range(len(unique_descriptions)):
    cur = unique_descriptions[i]
    filtered_others_df = df[df['description'] == cur]
    ax[i // 2, i % 2].set_title(f'{cur}')
    sns.countplot(ax=ax[i // 2, i % 2], data=filtered_others_df, x='rating')
```





In [190]:
```
df_desc_rating = df.groupby(['description'])['rating'].mean().reset_index(name
df_desc_rating = df_desc_rating.sort_values(by='Mean', ascending=False).reset_
mean_LIGHTING_rate = df_desc_rating.copy()
df_desc_rating.iloc[40:70]
```

Out[190]:

	description	Mean
40	Low energy lighting in 63% of fixed outlets	4.0
41	Low energy lighting in 64% of fixed outlets	4.0
42	Low energy lighting in 65% of fixed outlets	4.0
43	Low energy lighting in 67% of fixed outlets	4.0
44	Low energy lighting in 68% of fixed outlets	4.0
45	Low energy lighting in 69% of fixed outlets	4.0
46	Low energy lighting in 66% of fixed outlets	4.0
47	Low energy lighting in 56% of fixed outlets	4.0
48	Low energy lighting in 52% of fixed outlets	4.0
49	Low energy lighting in 47% of fixed outlets	4.0
50	Low energy lighting in 50% of fixed outlets	4.0
51	Low energy lighting in 45% of fixed outlets	4.0
52	Low energy lighting in 46% of fixed outlets	4.0
53	Low energy lighting in 51% of fixed outlets	4.0
54	Low energy lighting in 48% of fixed outlets	4.0
55	Low energy lighting in 49% of fixed outlets	4.0
56	Low energy lighting in 35% of fixed outlets	3.0
57	Low energy lighting in 25% of fixed outlets	3.0
58	Low energy lighting in 26% of fixed outlets	3.0
59	Low energy lighting in 28% of fixed outlets	3.0
60	Low energy lighting in 29% of fixed outlets	3.0
61	Low energy lighting in 31% of fixed outlets	3.0
62	Low energy lighting in 32% of fixed outlets	3.0
63	Low energy lighting in 33% of fixed outlets	3.0
64	Low energy lighting in 34% of fixed outlets	3.0
65	Low energy lighting in 27% of fixed outlets	3.0
66	Low energy lighting in 36% of fixed outlets	3.0
67	Low energy lighting in 42% of fixed outlets	3.0
68	Low energy lighting in 37% of fixed outlets	3.0
69	Low energy lighting in 44% of fixed outlets	3.0

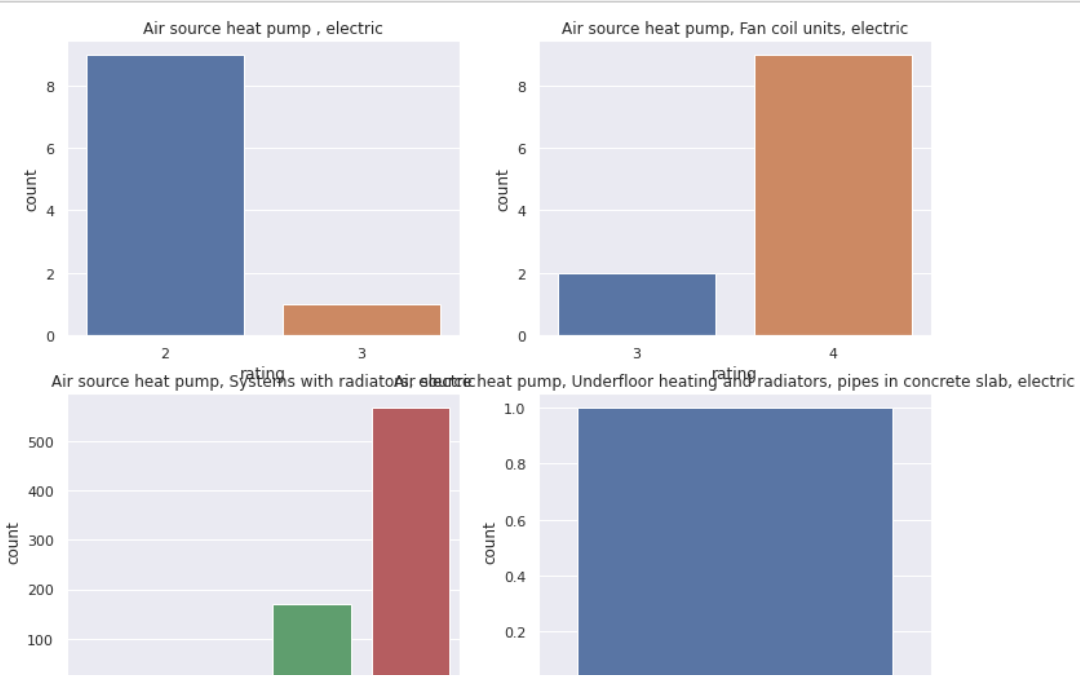
MAINHEAT_DESCRIPTION

Insulation matters similar to the other physical building features

In [191]:
```
mh_ee_df = general_preprocess(dataset, 'MAINHEAT_DESCRIPTION', 'MAINHEAT_ENERG
```

In [192]:
```
df = mh_ee_df
```

In [193]:
```
unique_descriptions = sorted(df['description'].unique())
fig, ax = plt.subplots(math.ceil(len(unique_descriptions) / 2), 2)
fig.set_size_inches(12, 200)
for i in range(len(unique_descriptions)):
    cur = unique_descriptions[i]
    filtered_others_df = df[df['description'] == cur]
    ax[i // 2, i % 2].set_title(f'{cur}')
    sns.countplot(ax=ax[i // 2, i % 2], data=filtered_others_df, x='rating')
```



In [194]:
```
df_desc_rating = df.groupby(['description'])['rating'].mean().reset_index(name
df_desc_rating = df_desc_rating.sort_values(by='Mean', ascending=False).reset_
mean_MAINHEAT_rate = df_desc_rating.copy()
df_desc_rating.head(20)
```

Out[194]:

	description	Mean
0	Ground source heat pump, Underfloor heating, p...	5.000000
1	Air source heat pump, Underfloor heating and r...	5.000000
2	Air source heat pump, Underfloor heating, pipe...	5.000000
3	Air source heat pump, Underfloor heating, pipe...	5.000000
4	Ground source heat pump, Underfloor heating, p...	5.000000
5	Ground source heat pump, Underfloor heating, p...	5.000000
6	Ground source heat pump, Underfloor heating an...	5.000000
7	Ground source heat pump, Systems with radiator...	5.000000
8	Air source heat pump, Underfloor heating and r...	4.967742
9	Air source heat pump, Underfloor heating, pipe...	4.925926
10	Air source heat pump, Systems with radiators, ...	4.742323
11	Community scheme	4.033659
12	Boiler & underfloor, mains gas	4.000000
13	Boiler, mains gas	4.000000
14	Boiler And underfloor heating, mains gas	4.000000
15	Warm air, mains gas	4.000000
16	Ground source heat pump, fan coil units, electric	4.000000
17	Air source heat pump, Underfloor heating and r...	4.000000
18	Air source heat pump, radiators, mains gas	4.000000
19	Boiler and radiators, mains gas	3.999566

In [195]:
```
df_desc_rating.description.value_counts()
```

Out[195]:
```
Ground source heat pump, Underfloor heating, pipes in screed above insulatio
n, electric                  1
Air source heat pump, Underfloor heating and radiators, pipes in insulated ti
mber floor, electric         1
Boiler and underfloor heating, wood pellets
1
Room heaters, oil
1
Room heaters, smokeless fuel
1
Room heaters, wood pellets
1
Boiler &amp;amp; underfloor, LPG
1
Warm air, oil
1
Boiler and radiators, wood logs
1
Air source heat pump , electric
1
Electric storage heaters, radiators
1
Boiler and radiators, B30K
1
Boiler and radiators, wood pellets
1
Warm air, Electricaire
1
Air source heat pump, warm air, electric
1
Boiler and radiators, coal
1
Room heaters, mains gas
1
Boiler and radiators, dual fuel (mineral and wood)
1
Water source heat pump, underfloor, electric
1
Boiler and radiators,
1
Boiler and underfloor heating, B30K
1
Boiler and radiators, LPG
1
Room heaters, coal
1
Room heaters, bottled LPG
1
Electric ceiling heating
1
Boiler and radiators, bottled LPG
1
Warm air, LPG
1
No system present: electric heaters assumed
1
Portable electric heaters assumed for most rooms
1
Room heaters, electric
1
Room heaters, anthracite
```

1
Room heaters, wood logs
1
Room heaters, LPG
1
Boiler and radiators, electric
1
Room heaters, dual fuel (mineral and wood)
1
Boiler and radiators, smokeless fuel
1
Boiler and underfloor heating, electric
1
Electric underfloor heating
1
Boiler and underfloor heating, LPG
1
Boiler and underfloor heating, wood logs
1
Boiler and radiators, anthracite
1
Electric storage heaters
1
Air source heat pump, Systems with radiators, electric
1
Air source heat pump, Underfloor heating and radiators, pipes in concrete sla
b, electric 1
Ground source heat pump, fan coil units, electric
1
Warm air, mains gas
1
Boiler And underfloor heating, mains gas
1
Boiler, mains gas
1
Boiler & underfloor, mains gas
1
Community scheme
1
Air source heat pump, Underfloor heating, pipes in screed above insulation, e
lectric 1
Boiler and radiators, mains gas
1
Air source heat pump, Underfloor heating and radiators, pipes in screed above
insulation, electric 1
Ground source heat pump, Systems with radiators, electric
1
Ground source heat pump, Underfloor heating and radiators, pipes in screed ab
ove insulation, electric 1
Ground source heat pump, Underfloor heating, pipes in concrete slab, electric
1
Ground source heat pump, Underfloor heating, pipes in insulated timber floor,
electric 1
Air source heat pump, Underfloor heating, pipes in insulated timber floor, el
ectric 1
Air source heat pump, Underfloor heating, pipes in concrete slab, electric
1
Air source heat pump, radiators, mains gas
1
Boiler and underfloor heating, mains gas
1
Boiler and radiators, oil
1

```
Ground source heat pump, warm air, electric
1
Boiler and underfloor heating, oil
1
Air source heat pump, fan coil units, electric
1
Boiler, oil
1
Boiler And radiators, wood pellets
1
Electric ceiling heating, electric
1
Ground source heat pump , electric
1
Water source heat pump, radiators, electric
1
Boiler &amp;amp; underfloor, oil
1
Air source heat pump, underfloor, electric
1
Boiler &amp;amp; underfloor, wood pellets
1
Boiler and radiators, wood chips
1
Water source heat pump, warm air, electric
1
Ground source heat pump, radiators, electric
1
Ground source heat pump, underfloor, electric
1
Air source heat pump, radiators, electric
1
Air source heat pump, Fan coil units, electric
1
Boiler &amp;amp; underfloor, electric
1
Name: description, dtype: int64
```

MAINHEATCONT_DESCRIPTION

Flat rate charging seems to correlate with bad heating

Time, temperature zone control, use of thermostats lead to good ratings

In [196]:
```
dataset.head()
```

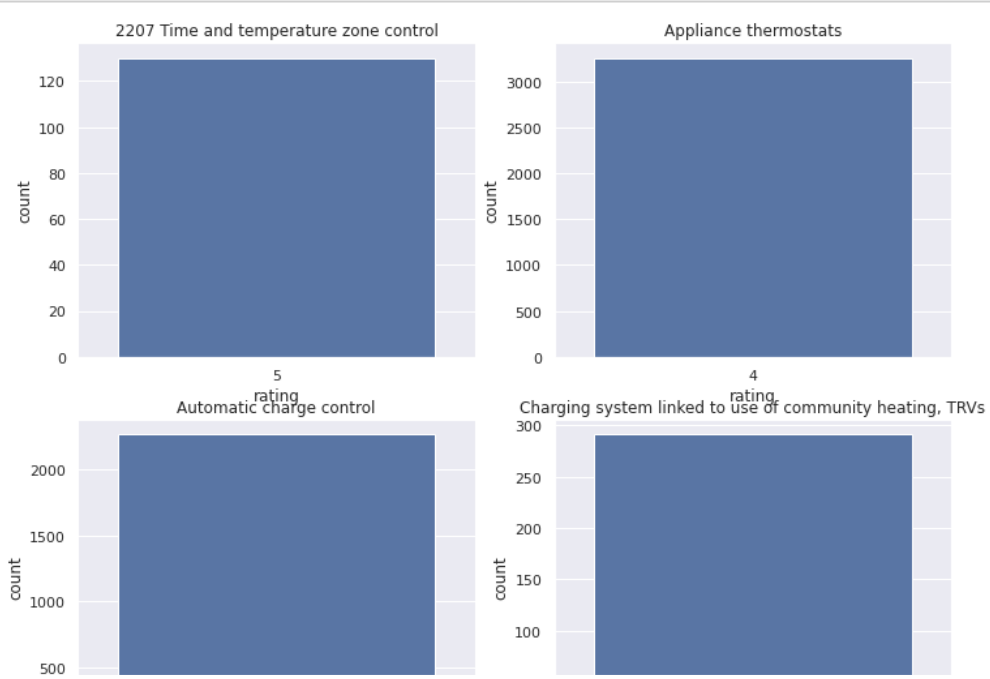
Out[196]:

	Property_UPRN	Postcode	POST_TOWN	Date of Assessment	Primary Energy Indicator (kWh/m²/year)	Total floor area (m²)	Current energy efficiency rating	Current energy efficiency rating band	Po Effi
0	1.001101e+09	EH4 5EZ	edinburgh	01/01/2021	375.0	94.0	53.0	E	
1	1.001951e+09	EH7 4HE	edinburgh	01/01/2021	250.0	175.0	66.0	D	
2	1.000996e+09	EH4 2DL	edinburgh	02/01/2021	403.0	72.0	61.0	D	
3	1.001257e+09	PH1 1SA	perth	02/01/2021	174.0	96.0	76.0	C	
4	1.235709e+09	G78 1QN	glasgow	02/01/2021	145.0	58.0	79.0	C	

In [197]:
```
hw_ee_df = general_preprocess(dataset, 'MAINHEATCONT_DESCRIPTION', 'MAINHEATC_
```

In [198]:
```
df = hw_ee_df
```

In [199]:
```
unique_descriptions = sorted(df['description'].unique())
fig, ax = plt.subplots(math.ceil(len(unique_descriptions) / 2), 2)
fig.set_size_inches(12, 100)
for i in range(len(unique_descriptions)):
    cur = unique_descriptions[i]
    filtered_others_df = df[df['description'] == cur]
    ax[i // 2, i % 2].set_title(f'{cur}')
    sns.countplot(ax=ax[i // 2, i % 2], data=filtered_others_df, x='rating')
```



In [200]:
```
df_desc_rating = df.groupby(['description'])['rating'].mean().reset_index(name
df_desc_rating = df_desc_rating.sort_values(by='Mean', ascending=False).reset_
mean_MAINHEATCONT_rate = df_desc_rating.copy()
df_desc_rating.head(20)
```

Out[200]:

	description	Mean
0	2207 Time and temperature zone control	5.000000
1	Time and temperature zone control	5.000000
2	Programmer and at least two room thermostats	4.012059
3	Programmer, room thermostat and TRVs	4.002702
4	Programmer and room thermostats	4.000000
5	Charging system linked to use of community hea...	4.000000
6	Charging system linked to use of community hea...	4.000000
7	Charging system linked to use of community hea...	4.000000
8	Controls for high heat retention storage heaters	4.000000
9	Programmer and appliance thermostats	4.000000
10	Temperature zone control	4.000000
11	Charging system linked to use of community hea...	4.000000
12	Programmer, TRVs and boiler energy manager	4.000000
13	Appliance thermostats	4.000000
14	Room thermostats only	4.000000
15	Programmer, TRVs and flow switch	3.285714
16	Programmer and room thermostat	3.069886
17	Programmer, TRVs and bypass	3.000000
18	Automatic charge control	3.000000
19	Thermostat and programmer	3.000000

WINDOWS_DESCRIPTION

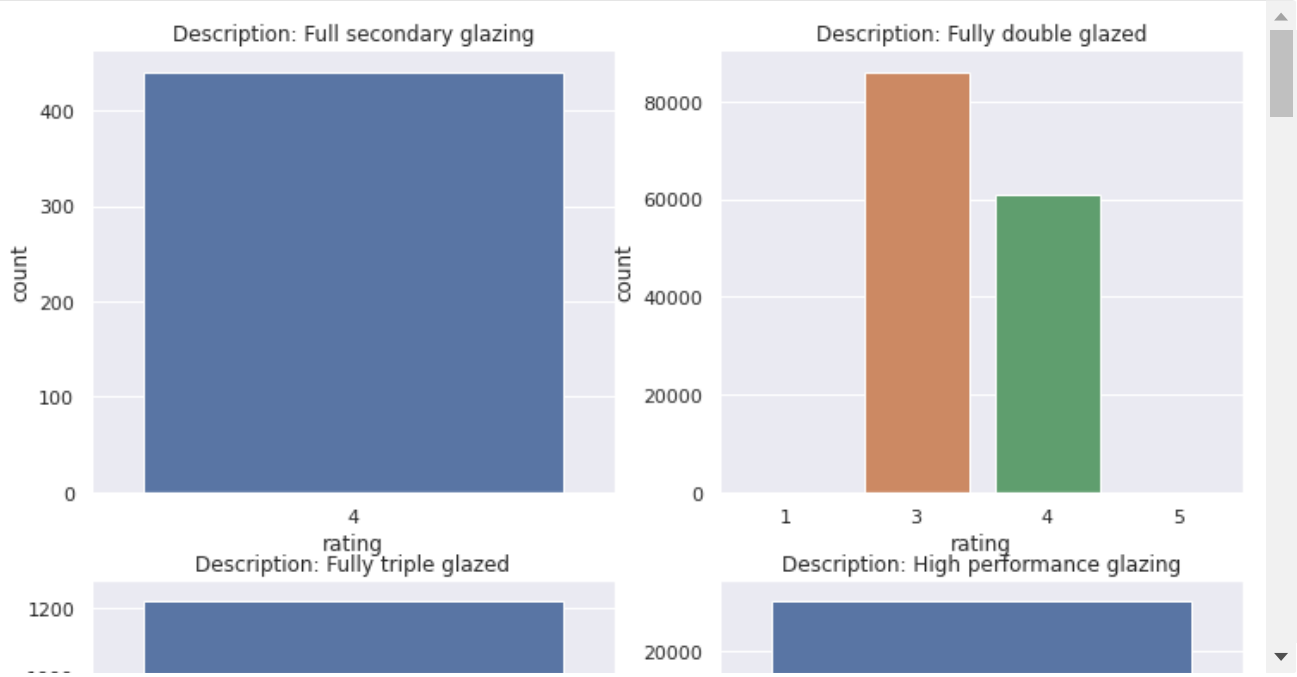
The number of glazing AND the degree of glazing matters. More full the more efficient.

Use of secondary glazing leads to mixed results, probably dependent mainly on base glazing.

Thus we should recommend more layers of glazing that are mostly or all full, and recommend secondary glazing if not present.

In [201]:
```
w_ee_df = general_preprocess(dataset, 'WINDOWS_DESCRIPTION', 'WINDOWS_ENERGY_E
```

In [202]:
```
df = w_ee_df
unique_descriptions = sorted(df['description'].unique())
fig, ax = plt.subplots(math.ceil(len(unique_descriptions) / 2), 2)
fig.set_size_inches(12, 50)
for i in range(len(unique_descriptions)):
    cur = unique_descriptions[i]
    filtered_others_df = df[df['description'] == cur]
    ax[i // 2, i % 2].set_title(f'{cur}')
    sns.countplot(ax=ax[i // 2, i % 2], data=filtered_others_df, x='rating')
```



In [203]:
```
df_desc_rating = df.groupby(['description'])['rating'].mean().reset_index(name
df_desc_rating = df_desc_rating.sort_values(by='Mean', ascending=False).reset_
mean_window_rate = df_desc_rating.copy()
df_desc_rating.head(20)
```

Out[203]:

	description	Mean
0	Description: High performance glazing	5.000000
1	Description: Full secondary glazing	4.000000
2	Description: Fully triple glazed	4.000000
3	Description: Mostly triple glazing	4.000000
4	Description: Fully double glazed	3.413986
5	Description: Multiple glazing throughout	3.279412
6	Description: Mostly multiple glazing	3.272727
7	Description: Mostly double glazing	3.167067
8	Description: Mostly secondary glazing	3.010870
9	Description: Partial triple glazing	2.700000
10	Description: Partial multiple glazing	2.470588
11	Description: Partial secondary glazing	2.330645
12	Description: Partial double glazing	2.214312
13	Description: Some triple glazing	2.000000
14	Description: Some secondary glazing	1.758242
15	Description: Some double glazing	1.435699
16	Description: Some multiple glazing	1.333333
17	Description: Single glazed	1.001064

Challenge 4 conclusion

Due to lack of time, I may not be able to implement the algorithm.

The idea is to iterate through the key characteristics of the property (i.e. walls, roof, floor, windows, main heat) and find their mean ratings (e.g. Full secondary glazing has mean rating of 4)

If it's less than 4, suggest features that whose mean ratings are 4 or more.

If this feature is a continuous value, recommend the minimum values as highlighted throughout this section

Algorithm Challenge 5: Build an algorithm that takes as input the characteristics of a building (any field of the dataset except those related to costs) and outputs the total cost of energy of the building over a 3-year period - (15 points)

Continuing from the insights of Challenge 4, we'll make use of the physical structures of a property to predict the total output cost of energy of building over 3 year period.

"Current energy cost rating (EER or ‘SAP rating’) for the building which is calculated using both the energy efficiency of the building and the cost of fuels used."

Since this metric uses costs of fuelds, we should refrain from using it. This goes for the other physical-feature specific rating, especially since some contain N/A values

Methodology

We'll replace the categorical data of each physical feature with a very broad mean value derived from mainly challenge 4

In [204]:
```
dataset.head()
```

Out[204]:

	Property_UPRN	Postcode	POST_TOWN	Date of Assessment	Primary Energy Indicator (kWh/m²/year)	Total floor area (m²)	Current energy efficiency rating	Current energy efficiency rating band	Po Effi
0	1.001101e+09	EH4 5EZ	edinburgh	01/01/2021	375.0	94.0	53.0	E	
1	1.001951e+09	EH7 4HE	edinburgh	01/01/2021	250.0	175.0	66.0	D	
2	1.000996e+09	EH4 2DL	edinburgh	02/01/2021	403.0	72.0	61.0	D	
3	1.001257e+09	PH1 1SA	perth	02/01/2021	174.0	96.0	76.0	C	
4	1.235709e+09	G78 1QN	glasgow	02/01/2021	145.0	58.0	79.0	C	

In [205]:
```
dataset['Property Type'].value_counts()
```

Out[205]:
```
House          81519
Flat           79909
Bungalow       19918
Maisonette      3603
Park home         90
Name: Property Type, dtype: int64
```

In [206]:
```
property_desc_rating = dataset.groupby(['Property Type'])['Current energy effi
property_desc_rating = property_desc_rating.sort_values(by='Mean', ascending=F
```

In [207]:
```
property_desc_rating.head(20) #Rather significant
```

Out[207]:

	Property Type	Mean
0	Flat	70.891489
1	House	69.238276
2	Maisonette	66.748543
3	Bungalow	62.687669
4	Park home	55.222222

In [208]:
```
built_desc_rating = dataset.groupby(['Built Form'])['Current energy efficiency
built_desc_rating = built_desc_rating.sort_values(by='Mean', ascending=False).
```

In [209]:
```
built_desc_rating.head(20) # Insignificant
```

Out[209]:

	Built Form	Mean
0	Enclosed Mid-Terrace	70.434824
1	Mid-Terrace	69.924877
2	End-Terrace	69.715773
3	Enclosed End-Terrace	69.627995
4	Semi-Detached	68.619501
5	Detached	68.521267

In [217]:
```
property_desc_rating[property_desc_rating['Property Type'] == 'Flat'].to_numpy
```

Out[217]:
```
70.89148906881577
```

In [210]:
```
final_dataset = dataset.copy()
```

In [219]:
```
def apply_property(x):
    cur = x['Property Type']
#     print(f"cur: {cur}")
    expected = property_desc_rating[property_desc_rating['Property Type'] == c
    return expected
final_dataset['property_mean'] = final_dataset.apply(apply_property,axis=1)
```

In [220]:
```
final_dataset['property_mean'].value_counts()
```

Out[220]:
```
69.238276    81519
70.891489    79909
62.687669    19918
66.748543     3603
55.222222       90
Name: property_mean, dtype: int64
```

In [223]:
```
mean_window_rate.head()
```

Out[223]:

	description	Mean
0	Description: High performance glazing	5.000000
1	Description: Full secondary glazing	4.000000
2	Description: Fully triple glazed	4.000000
3	Description: Mostly triple glazing	4.000000
4	Description: Fully double glazed	3.413986

In [224]:
```
final_dataset.head()
```

Out[224]:

Potential Environmental Impact Rating Band	CO2 Emissions Current Per Floor Area (kg.CO2/m²/yr)	WALL_DESCRIPTION	WALL_ENERGY_EFF	ROOF_DESCRIPTION	ROOF_ENE
B	66.0	Cavity wall, as built, no insulation (assumed)	Poor	Pitched, 25 mm loft insulation	
C	44.0	Cavity wall, filled cavity	Average	Pitched, insulated (assumed) | Roof room(s), i...	Average |
D	68.0	Cavity wall, as built, partial insulation (ass...	Average | Average	(another dwelling above)	
B	31.0	Cavity wall, as built, insulated (assumed)	Good	Pitched, 250 mm loft insulation	
B	26.0	Sandstone or limestone, with internal insulati...	Good | Good	Pitched, 200 mm loft insulation	

In [240]:
```
def apply_window(x):
    col = 'WINDOWS_DESCRIPTION'
    cur = x[col]
    expected = mean_window_rate[mean_window_rate['description'] == cur].to_num
    return expected
final_dataset = final_dataset.dropna(subset=['WINDOWS_DESCRIPTION'])
final_dataset['WINDOWS_DESCRIPTION'] = final_dataset['WINDOWS_DESCRIPTION'].ap
final_dataset['window_mean'] = final_dataset.apply(apply_window,axis=1)
```

In [251]:
```
def apply_wall(x):
    col = 'WALL_DESCRIPTION'
    mean_df = mean_WALL_rate
    cur = x[col]
#     print(cur)
    cur = cur.split('|')
#     print(cur)
    expected = 0 #mean_window_rate[mean_window_rate['description'] == cur].to_
    for x in cur:
        x = clean_string(x)
        expected += mean_df[mean_df['description'] == x].to_numpy()[0][1]
    return expected / len(cur)
final_dataset = final_dataset.dropna(subset=['WALL_DESCRIPTION'])
final_dataset['WALL_DESCRIPTION'] = final_dataset['WALL_DESCRIPTION'].apply(cl
final_dataset['wall_mean'] = final_dataset.apply(apply_wall,axis=1)
```

```
Cavity wall, as built, no insulation
System built, as built, no insulation
Timber frame, as built, insulated
Cavity wall, as built, insulated
Granite or whinstone, as built, no insulation
Timber frame, as built, insulated
Cavity wall, as built, insulated
Sandstone or limestone, as built, no insulation
Cavity wall, filled cavity
Solid brick, as built, no insulation
Timber frame, as built, insulated
Cavity wall, filled cavity
Cavity wall, filled cavity
Granite or whinstone, as built, no insulation
Timber frame, as built, insulated
Cavity wall, filled cavity
Cavity wall, filled cavity
Cavity wall, filled cavity
System built, with external insulation
Cavity wall, filled cavity
```

In [270]:
```
def apply_roof(x):
    col = 'ROOF_DESCRIPTION'
    mean_df = mean_ROOF_rate
#     print(x)
    cur = x[col]
#     print(cur)
    cur = cur.split('|')
#     print(cur)
    expected = 0 #mean_window_rate[mean_window_rate['description'] == cur].to_
    for x in cur:
        x = clean_string(x)
#         print(x)
        temp = mean_df[mean_df['description'] == x]
        if(len(temp) == 0):
            expected += 0
            continue
        expected += mean_df[mean_df['description'] == x].to_numpy()[0][1]
    return expected / len(cur)
final_dataset = final_dataset.dropna(subset=['ROOF_DESCRIPTION'])
final_dataset['ROOF_DESCRIPTION'] = final_dataset['ROOF_DESCRIPTION'].apply(cl
final_dataset['roof_mean'] = final_dataset.apply(apply_roof,axis=1)
```

In [271]:
```
def apply_mainheat(x):
    col = 'MAINHEAT_DESCRIPTION'
    mean_df = mean_MAINHEAT_rate
    cur = x[col]
#     print(cur)
    cur = cur.split('|')
#     print(cur)
    expected = 0 #mean_window_rate[mean_window_rate['description'] == cur].to_
    for x in cur:
        x = clean_string(x)
        expected += mean_df[mean_df['description'] == x].to_numpy()[0][1]
    return expected / len(cur)
final_dataset = final_dataset.dropna(subset=['MAINHEAT_DESCRIPTION'])
final_dataset['MAINHEAT_DESCRIPTION'] = final_dataset['MAINHEAT_DESCRIPTION'].
final_dataset['mainheat_mean'] = final_dataset.apply(apply_wall,axis=1)
```

```
Timber frame, as built, insulated
Cavity wall, as built, insulated
Sandstone or limestone, as built, no insulation
System built, as built, no insulation
Cavity wall, as built, insulated
Cavity wall, filled cavity
System built, with external insulation
System built, with external insulation
System built, with external insulation
System built, with external insulation
System built, with external insulation
System built, with external insulation
System built, with external insulation
System built, with external insulation
System built, with external insulation
System built, with external insulation
Sandstone or limestone, as built, no insulation
Solid brick, as built, no insulation
System built, with external insulation
Cavity wall, filled cavity
```

In [275]:
```
final_dataset.head()
```

Out[275]:

:FF	LIGHTING_ENV_EFF	Current Emissions (T.CO2/yr)	Potential Reduction in Emissions (T.CO2/yr)	Total current energy costs over 3 years (£)	Current heating costs over 3 years (£)	Potential heating costs over 3 years (£)	Current hot water costs over 3 years (£)	Potential hot water costs over 3 years (£)	Current lighting costs over 3 years (£)	P y
ood	Very Good	6.2	4.2	3789.0	2922.0	1548.0	645.0	219.0	222.0	
ood	Very Good	7.7	2.8	4635.0	4068.0	3015.0	246.0	246.0	321.0	
ood	Good	4.9	1.6	3570.0	2226.0	1191.0	1038.0	564.0	306.0	
ood	Very Good	2.9	0.9	2049.0	1554.0	1554.0	258.0	177.0	237.0	
ood	Very Good	1.5	0.0	1212.0	828.0	828.0	216.0	216.0	168.0	

In [290]:
```
target_col = 'Total current energy costs over 3 years (£)'
input_cols = ['mainheat_mean', 'roof_mean','wall_mean','window_mean','property
all_cols = input_cols
all_cols.append(target_col)
```

In [291]:
```
# Filter for only the mean features, consider normalising them later
final_dataset_means = final_dataset[all_cols]
```

In [288]:
```
(CO2_col - CO2_col.min())/(CO2_col.max()-CO2_col.min())
```

```
9         0.180180
10        0.180180
11        0.162162
12        0.162162
13        0.058559
14        0.058559
15        0.099099
16        0.099099
17        0.218468
18        0.218468
19        0.182432
20        0.182432
21        0.204955
22        0.204955
23        0.159910
24        0.159910
25        0.184685
26        0.184685
27        0.155405
28        0.155405
```

In [292]:
```
for i in input_cols:
    cur = final_dataset_means[i]
    final_dataset_means[i] = (cur - cur.min())/(cur.max()-cur.min())
final_dataset_means.head(10)
```

```
/tmp/ipykernel_751989/3603412996.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  final_dataset_means[i] = (cur - cur.min())/(cur.max()-cur.min())
```

Out[292]:

	mainheat_mean	roof_mean	wall_mean	window_mean	property_mean	Total current energy costs over 3 years (£)
0	0.250160	0.400000	0.250160	0.603391	0.894493	0.019346
1	0.562713	0.588534	0.562713	0.603391	0.894493	0.024054
2	0.531504	1.000000	0.531504	0.000000	1.000000	0.018128
3	0.790349	0.800000	0.790349	0.603391	0.894493	0.009665
4	0.827798	0.800000	0.827798	0.603391	1.000000	0.005008
5	0.562713	0.600000	0.562713	0.603391	0.476439	0.024721
6	0.562713	0.799551	0.562713	0.603391	0.894493	0.014339
7	0.771976	0.800000	0.771976	0.603391	0.894493	0.008062
8	0.533505	0.800000	0.533505	0.603391	0.894493	0.014038
9	0.837942	1.000000	0.837942	0.603391	1.000000	0.004891

In [296]:
```
x = final_dataset_means[input_cols].to_numpy()#.reshape(-1, 1)
y = final_dataset_means[target_col].to_numpy()
```

In [297]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20, shuf
```

In [300]:
```
# Create two models: Quadratic and linear regression
polyreg = make_pipeline(PolynomialFeatures(2), LinearRegression(fit_intercept=
linreg = LinearRegression()
scoring = "neg_root_mean_squared_error"
polyscores = cross_validate(polyreg, x, y, scoring=scoring, return_estimator=T
linscores = cross_validate(linreg, x, y, scoring=scoring, return_estimator=Tru
```

In [301]:
```
# Which one is better? Linear and polynomial
print(linscores["test_score"].mean())
print(polyscores["test_score"].mean())
print(linscores["test_score"].mean() - polyscores["test_score"].mean())
```

```
-1.1306940979024587e-16
-8.208992290962625e-17
-3.097948688061962e-17
```

Challenge 5 conclusion

Such feature engineering using means and linear regression has proven to be able to predict fairly well the total energy cost of building over 3 year period|

Algorithm Challenge 6: Build an algorithm that takes as input the characteristics of a building (any field in the dataset) and outputs recommendations on which elements of the house should be modified to most effectively decrease the total energy cost of the building over a 3-year period

In [302]:
```
semi_filtered_df = dataset[(dataset['Current energy efficiency rating'] < 150)
```

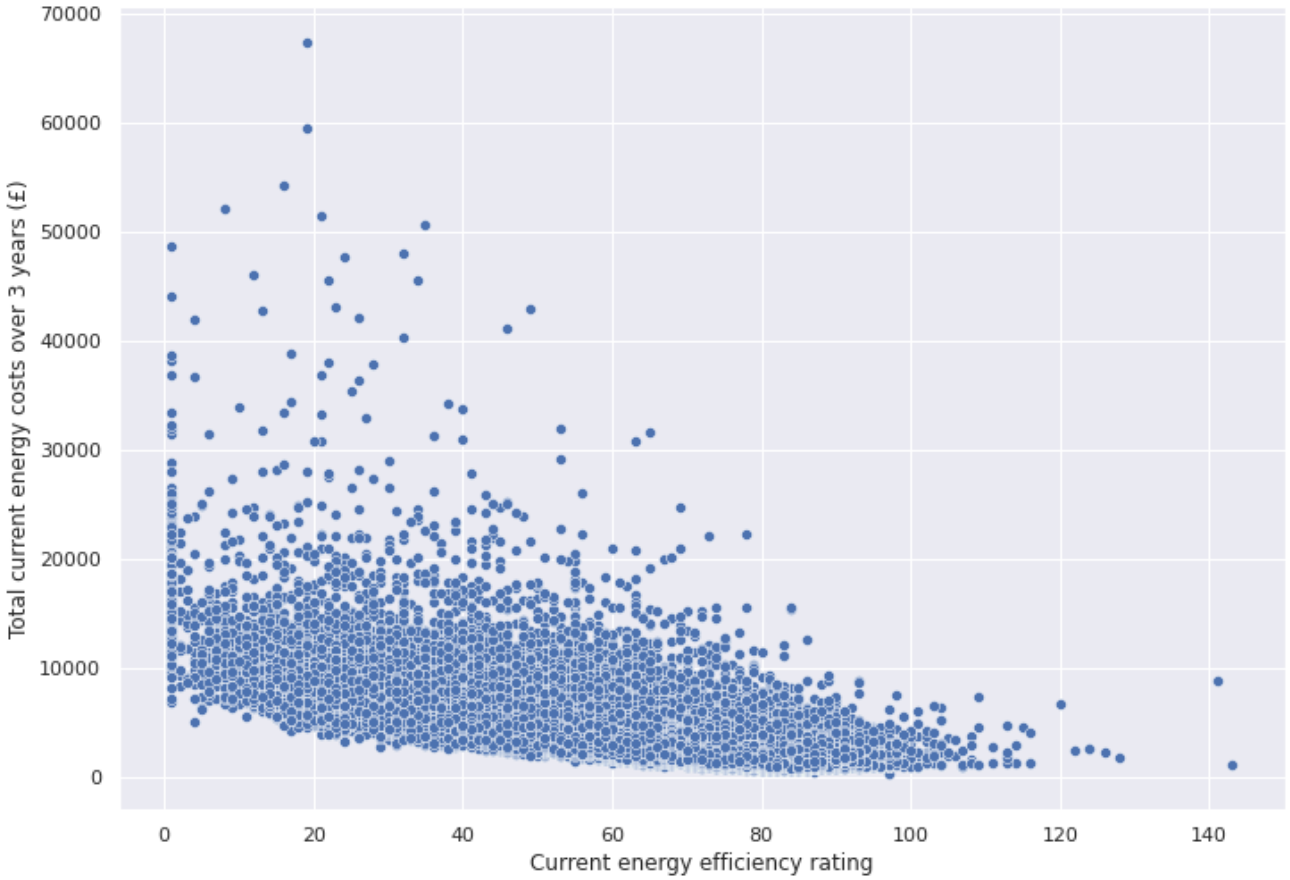
In [303]:
```
semi_filtered_df.head()
```

Out[303]:

	Property_UPRN	Postcode	POST_TOWN	Date of Assessment	Primary Energy Indicator (kWh/m²/year)	Total floor area (m²)	Current energy efficiency rating	Current energy efficiency rating band	Po E Effi
0	1.001101e+09	EH4 5EZ	edinburgh	01/01/2021	375.0	94.0	53.0	E	
1	1.001951e+09	EH7 4HE	edinburgh	01/01/2021	250.0	175.0	66.0	D	
2	1.000996e+09	EH4 2DL	edinburgh	02/01/2021	403.0	72.0	61.0	D	
3	1.001257e+09	PH1 1SA	perth	02/01/2021	174.0	96.0	76.0	C	
4	1.235709e+09	G78 1QN	glasgow	02/01/2021	145.0	58.0	79.0	C	

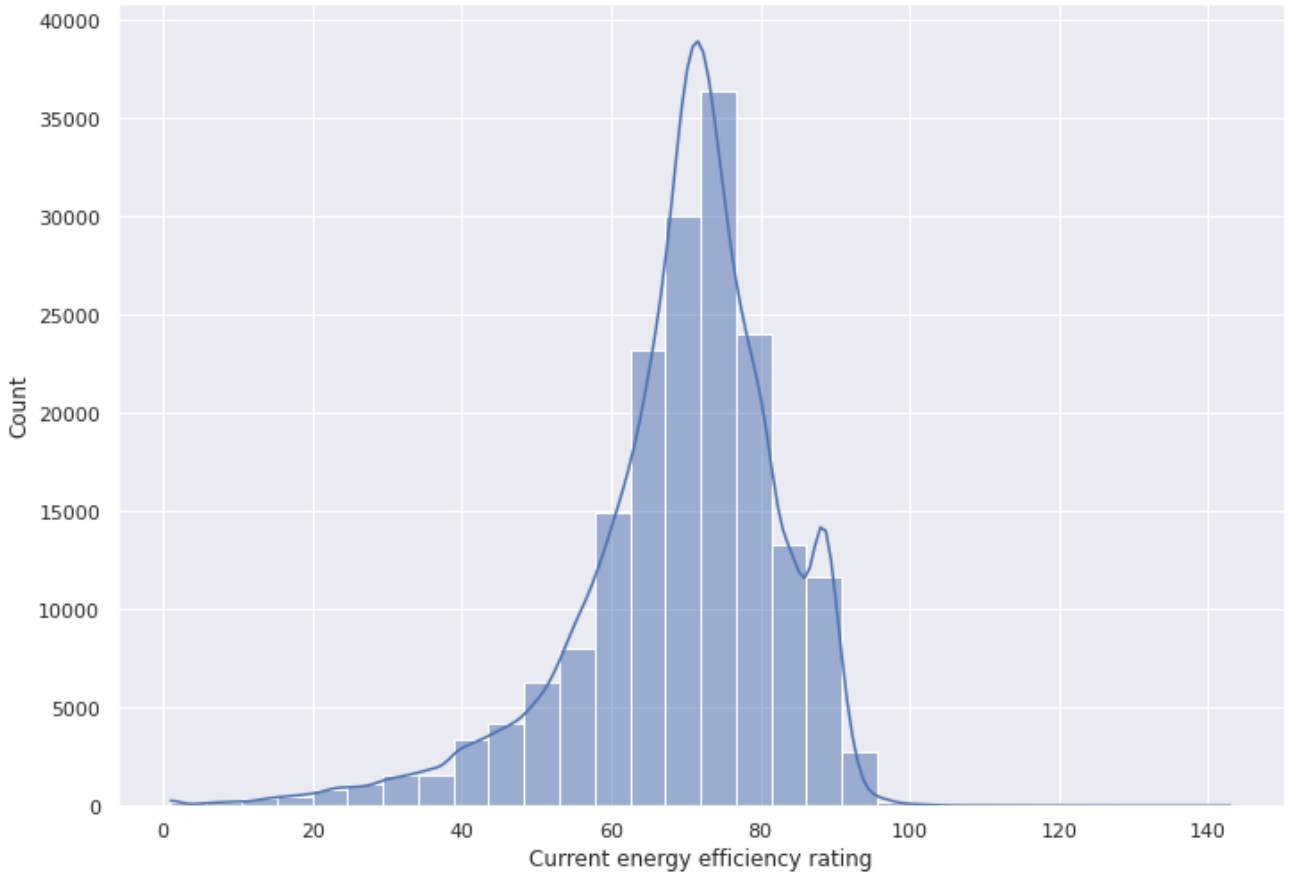
In [304]:
```
sns.scatterplot(data=semi_filtered_df, x='Current energy efficiency rating',y=
```

Out[304]:
```
<AxesSubplot: xlabel='Current energy efficiency rating', ylabel='Total curren
t energy costs over 3 years (£)'>
```



In [305]:
```
sns.histplot(data=semi_filtered_df, x="Current energy efficiency rating", bins
```

Out[305]:
```
<AxesSubplot: xlabel='Current energy efficiency rating', ylabel='Count'>
```



The above plots shows that there is a slight negative correlation between current energy efficiency rating and total current energy costs over 3 years. The spread of the total current energy costs is shown to decrease as energy efficiency ratings increase. (e.g. even when properties with 0-400 energy effiency rating has a low count, the spread if enourmous compared to the 60 onwards range which have a much higher count)

We can then conclude that if we want a HIGHER chance of the total current energy costs over 3 years to relatively low, the current energy efficiency rating should be as high as possible.

An appropriate algorithm would build on the results of 1, 2, and 4 to provide guidance on how to boost energy performance as defined by energy efficiency. That's because both this challenge and challenge 4 tackle the same underlying principle of energy conservation based on current energy efficiency

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: