In [1]:
```
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# exploring the dataset
dataframe = pd.read_csv('city_level.csv')
```

In [3]:
```
dataframe
```

Out[3]:

	iso3c	region_id	country_name	income_id	city_name	additional_data_annual_budget_for_was
0	AFG	SAS	Afghanistan	LIC	Jalalabad	
1	AFG	SAS	Afghanistan	LIC	Kandahar	
2	AFG	SAS	Afghanistan	LIC	Mazar-E-Sharif	
3	AFG	SAS	Afghanistan	LIC	Kabul	
4	AFG	SAS	Afghanistan	LIC	HiratÂ	
...	...	...	...	...	...	
362	ZWE	SSF	Zimbabwe	LIC	HarareÂ	
363	ZWE	SSF	Zimbabwe	LIC	Gweru	
364	ZWE	SSF	Zimbabwe	LIC	Kariba	
365	ZWE	SSF	Zimbabwe	LIC	Masvingo City	
366	ZWE	SSF	Zimbabwe	LIC	SakubvaÂ	

367 rows × 113 columns

In [4]:
```
dataframe.head()
```

Out[4]:

	iso3c	region_id	country_name	income_id	city_name	additional_data_annual_budget_for_waste_
0	AFG	SAS	Afghanistan	LIC	Jalalabad	
1	AFG	SAS	Afghanistan	LIC	Kandahar	
2	AFG	SAS	Afghanistan	LIC	Mazar-E-Sharif	
3	AFG	SAS	Afghanistan	LIC	Kabul	
4	AFG	SAS	Afghanistan	LIC	HiratÂ	

5 rows × 113 columns

In [5]:
```
dataframe_details= pd.DataFrame(dataframe, columns=['iso3c', 'region_id', 'countr
```

In [6]:
```
dataframe_details
```

Out[6]:

	iso3c	region_id	country_name	city_name
0	AFG	SAS	Afghanistan	Jalalabad
1	AFG	SAS	Afghanistan	Kandahar
2	AFG	SAS	Afghanistan	Mazar-E-Sharif
3	AFG	SAS	Afghanistan	Kabul
4	AFG	SAS	Afghanistan	HiratÂ
...	...	...	...	...
362	ZWE	SSF	Zimbabwe	HarareÂ
363	ZWE	SSF	Zimbabwe	Gweru
364	ZWE	SSF	Zimbabwe	Kariba
365	ZWE	SSF	Zimbabwe	Masvingo City
366	ZWE	SSF	Zimbabwe	SakubvaÂ

367 rows × 4 columns

In [7]:
```
# select the needed fields
dataframe_trash = pd.DataFrame(dataframe,columns=['country_name', 'city_name', 'p
```

In [8]:
```
dataframe_trash
```

Out[8]:

	country_name	city_name	population_population_number_of_people	composition_food_organic_
0	Afghanistan	Jalalabad	326585.0	
1	Afghanistan	Kandahar	429000.0	
2	Afghanistan	Mazar-E-Sharif	635250.0	
3	Afghanistan	Kabul	3700000.0	
4	Afghanistan	HiratÂ	337000.0	
...	...	...	...	
362	Zimbabwe	HarareÂ	1379000.0	
363	Zimbabwe	Gweru	157865.0	
364	Zimbabwe	Kariba	26512.0	
365	Zimbabwe	Masvingo City	NaN	
366	Zimbabwe	SakubvaÂ	NaN	

367 rows × 12 columns

In [9]:
```
tion_food_organic_waste_percent', 'composition_paper_cardboard_percent', 'composit
tion_glass_percent', 'composition_metal_percent', 'composition_other_percent', 'c
```

In [10]:
```
# checking the dataframes
organic
```

Out[10]:

	country_name	city_name	population_population_number_of_people	composition_food_organic_
0	Afghanistan	Jalalabad	326585.0	
1	Afghanistan	Kandahar	429000.0	
2	Afghanistan	Mazar-E-Sharif	635250.0	
3	Afghanistan	Kabul	3700000.0	
4	Afghanistan	HiratÂ	337000.0	
...	...	...	...	
362	Zimbabwe	HarareÂ	1379000.0	
363	Zimbabwe	Gweru	157865.0	
364	Zimbabwe	Kariba	26512.0	
365	Zimbabwe	Masvingo City	NaN	
366	Zimbabwe	SakubvaÂ	NaN	

367 rows × 7 columns

In [11]:
```
inorganic
```

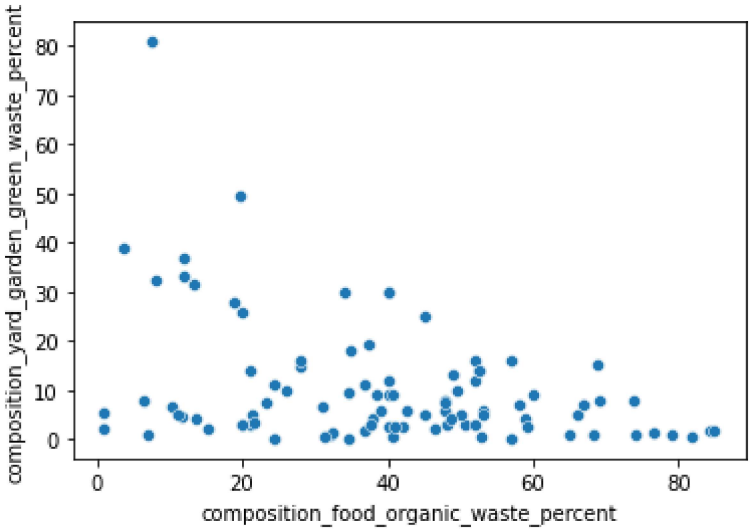
Out[11]:

	country_name	city_name	population_population_number_of_people	composition_glass_percent
0	Afghanistan	Jalalabad	326585.0	1.1175
1	Afghanistan	Kandahar	429000.0	2.3110
2	Afghanistan	Mazar-E-Sharif	635250.0	1.3200
3	Afghanistan	Kabul	3700000.0	1.0000
4	Afghanistan	HiratÂ	337000.0	1.4700
...	...	...	...	...
362	Zimbabwe	HarareÂ	1379000.0	6.4500
363	Zimbabwe	Gweru	157865.0	3.0000
364	Zimbabwe	Kariba	26512.0	2.0000
365	Zimbabwe	Masvingo City	NaN	4.0000
366	Zimbabwe	SakubvaÂ	NaN	5.0000

367 rows × 8 columns

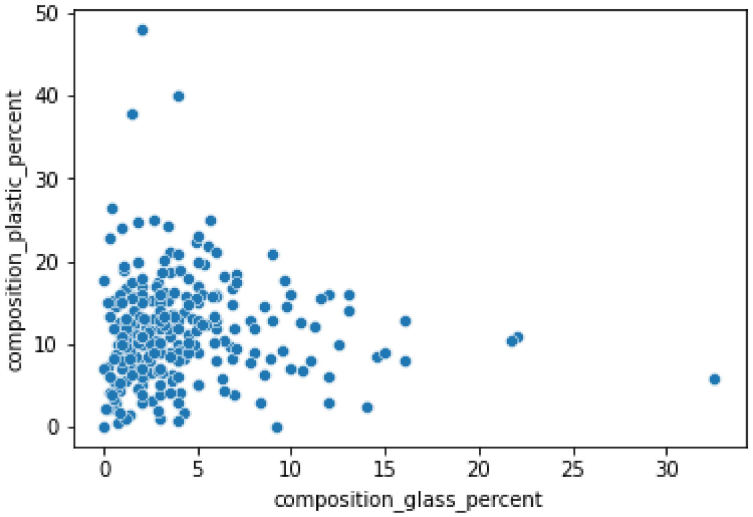
In [12]:
```
# plotting scatter plot for the organic data
import seaborn as sns

sns.scatterplot(data=organic, x="composition_food_organic_waste_percent", y="comp
```

Out[12]:
```
<AxesSubplot:xlabel='composition_food_organic_waste_percent', ylabel='compositi
on_yard_garden_green_waste_percent'>
```



In [13]:
```
sns.scatterplot(data=inorganic, x="composition_glass_percent", y="composition_pla
```

Out[13]:
```
<AxesSubplot:xlabel='composition_glass_percent', ylabel='composition_plastic_pe
rcent'>
```



In [86]:
```
# creating the classifier model
#Based on the findings we have, the data will be categorised into two. Decomposer

organic1 = pd.DataFrame(dataframe_trash, columns=['composition_food_organic_waste
organic2 = pd.DataFrame(dataframe_trash, columns=['composition_plastic_percent'])

# Delete the NAN values for the model to work
organic_1 = organic1.dropna()
organic_2 = organic2.dropna()
```

In [87]:
```
organic_1.shape
```

Out[87]: (291, 1)

In [88]:
```
organic_2.shape
```

Out[88]: (283, 1)

In [89]:
```
organic_3 = organic_1.iloc[0:283, :]

organic_3
```

Out[89]:

	composition_food_organic_waste_percent
0	1.0455
1	11.6000
2	50.5000
3	13.5000
4	52.9400
...	...
354	42.6000
355	42.5000
356	40.0000
357	40.0000
358	60.0000

283 rows × 1 columns

In [90]:
```
organic_4 = organic_2.values.reshape(-1, 1)

organic_4
```

Out[90]:
```
array([[5.6455e+00],
       [9.6430e+00],
       [1.0640e+01],
       [1.1000e+01],
       [1.2060e+01],
       [1.2000e+01],
       [1.1000e+01],
       [1.1100e+01],
       [1.9000e+01],
       [2.4230e+01],
       [1.4000e+01],
       [1.1680e+01],
       [1.6000e+01],
       [1.5750e+01],
       [9.7500e+00],
       [1.2800e+01],
       [8.2000e+00],
       [1.5330e+01],
       [4.0000e+00],
```

In [91]:
```
# Model Building part of the code
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
```

In [92]:
```
from sklearn import preprocessing
from sklearn import utils

#convert y values to categorical values
recy = organic_4

lab = preprocessing.LabelEncoder()
recy_transformed = lab.fit_transform(recy)

#view transformed values
print(recy_transformed)
```

```
[ 33  70  81  82  94  93  82  83 156 172 117  89 138 134  71 103  55 127
  20  14  79  82  24  74  13  49  26  22  21 168  12  75  59 104 163 126
 121 164  74  96 113  25  82  50 145  74 102  53  98  38 157  67  16  74
 100  53 145   7  53  62 144  80 118  19  55  79 117   1  39  30 101  31
   5  20  51   5  46  74  18  11 135 147  53 143 174  37  82 104  79 147
  29 169   0 116 145  93  90  53  86 167 129 130 165 109 124   5  60 166
  28 128  93  93 114  10   9 138  99 139  74  47  64  28  48  17  85 104
 115  53  47 176 100 138  91 111 160 177  61 125  56 140 106  47  37  41
  36  93 104 145  82 153  80  38  15  20  80  65  72 159  62  82  47  63
  28  83  74  28  45 107 155 141  99  27  98 138 124  53 133 150  74 163
 173 138   8 124  20 154  62  42  58  40 117  53  28 112  73 149 119 162
 132  43  32 142  54  92 108  84 178  40   2 103  93  88 124  69 180 152
 120 104  97  23  78  93  62  68 104  35 104   6 138 110 158   4  82 138
 123  95 161  34  62   3  77  44 175  74  15 150 111 111  76  82  62 137
 131  66   8  62 171   7 100  47 136  87  59 138  52 105 122  74  37  15
 148 146  57  47  47  82  63  59 151  77  82 179 170]

C:\Users\Miss Gloriah\anaconda3\lib\site-packages\sklearn\preprocessing\_label.
py:115: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using rave
l().
  y = column_or_1d(y, warn=True)
```

In [93]:
```
# Creating the training and test data
decomp = organic_3

decomp_train, decomp_test, recy_transformed_train, recy_transformed_test = train_
```

In [94]:
```
from xgboost import XGBRegressor
xgb_model = XGBRegressor(random_state = 96)
```

In [95]:
```
search_space ={'n_estimators': [100,200,500],
               "max_depth": [3, 6, 9],
               "gamma": [0, 0.01, 0.1],
               "learning_rate": [0.001, 0.01, 0.1, 1]
}
```

In [96]:
```
from sklearn.model_selection import GridSearchCV

GS = GridSearchCV(estimator = xgb_model,
                  param_grid = search_space,
                  scoring = ["r2", "neg_root_mean_squared_error"],
                  refit = 'r2',
                  cv = 5,
                  verbose = 4)
```

In [97]:
```
GS.fit(decomp_train, recy_transformed_train)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
[CV 1/5] END gamma=0, learning_rate=0.001, max_depth=3, n_estimators=100; neg
_root_mean_squared_error: (test=-89.375) r2: (test=-2.424) total time=   0.0s
[CV 2/5] END gamma=0, learning_rate=0.001, max_depth=3, n_estimators=100; neg
_root_mean_squared_error: (test=-81.991) r2: (test=-1.987) total time=   0.0s
[CV 3/5] END gamma=0, learning_rate=0.001, max_depth=3, n_estimators=100; neg
_root_mean_squared_error: (test=-104.890) r2: (test=-3.114) total time=   0.0
s

C:\Users\Miss Gloriah\anaconda3\lib\site-packages\xgboost\data.py:250: Future
Warning: pandas.Int64Index is deprecated and will be removed from pandas in a
future version. Use pandas.Index with the appropriate dtype instead.
  elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
C:\Users\Miss Gloriah\anaconda3\lib\site-packages\xgboost\data.py:250: Future
Warning: pandas.Int64Index is deprecated and will be removed from pandas in a
future version. Use pandas.Index with the appropriate dtype instead.
  elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
C:\Users\Miss Gloriah\anaconda3\lib\site-packages\xgboost\data.py:250: Future
Warning: pandas.Int64Index is deprecated and will be removed from pandas in a
future version. Use pandas.Index with the appropriate dtype instead.
```

In [98]:
```
df = pd.DataFrame(GS.cv_results_)
df = df.sort_values('rank_test_r2')
df.to_csv('GS Results.csv')
```

In [99]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [100]:
```
tree_model= DecisionTreeClassifier(criterion='entropy')
tree_model.fit(decomp_train, recy_transformed_train)
```

Out[100]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [101]:
```
# finding the score of the model
tree_model.score(decomp_test, recy_transformed_test)
```

Out[101]:
```
0.017543859649122806
```

In [102]:
```
from sklearn import tree
fig=plt.figure(figsize=(25,15))
tree.plot_tree(tree_model,filled=True,rounded=True,
feature_names=organic, class_names=["composition_food_organic_waste_percent", "co
```

```
   3504     return self._getitem_multilevel(key)
-> 3505 indexer = self.columns.get_loc(key)
   3506 if is_integer(indexer):
   3507     indexer = [indexer]

File ~\anaconda3\lib\site-packages\pandas\core\indexes\base.py:3623, in Inde
x.get_loc(self, key, method, tolerance)
   3621     return self._engine.get_loc(casted_key)
   3622 except KeyError as err:
-> 3623     raise KeyError(key) from err
   3624 except TypeError:
   3625     # If we have a listlike key, _check_indexing_error will raise
   3626     #  InvalidIndexError. Otherwise we fall through and re-raise
   3627     #  the TypeError.
   3628     self._check_indexing_error(key)

KeyError: 0
```

In [103]:
```
# Logistic Regression
model = LogisticRegression()
```

In [104]:
```
model.fit(decomp_train, recy_transformed_train)
```

```
C:\Users\Miss Gloriah\anaconda3\lib\site-packages\sklearn\linear_model\_logisti
c.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-
learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on)
  n_iter_i = _check_optimize_result(
```

Out[104]:
```
LogisticRegression()
```

In [105]:
```
model.score(decomp_test, recy_transformed_test)
```

Out[105]:
```
0.05263157894736842
```

In [106]:
```
# Linear Regression
from sklearn.linear_model import LinearRegression
modell = LinearRegression()
```

In [107]:
```
modell.fit(decomp_train, recy_transformed_train)
```

Out[107]:
```
LinearRegression()
```

In [108]:
```
modell.score(decomp_test, recy_transformed_test)
```

Out[108]:
```
-0.08867506162880368
```

In [ ]:
```

```