Step 1: Data Loding and Understanding

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

import os

# hide warnings
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
# reading the csv files
cars= pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\cars_trucks_and_buses_per_1000_persons.csv")
co2=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\co2_emissions_tonnes_per_person.csv")
coal=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\coal_consumption_per_cap.csv")
ele_gen=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\electricity_generation_per_person.csv")
ele_use=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\electricity_use_per_person.csv")
forest=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\forest_coverage_percent.csv")
hydro=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\hydro_power_generation_per_person.csv")
income=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\income_per_person_gdppercapita_ppp_inflation_adjusted.csv")
industry=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\industry_percent_of_gdp.csv")
natural=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\natural_gas_production_per_person.csv")
oil_con=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\oil_consumption_per_cap.csv")
oil_pro=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\oil_production_per_person.csv")
year=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\yearly_co2_emissions_1000_tonnes.csv")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-3-ad36cf50ca52> in <module>
      1 # reading the csv files
----> 2 cars= pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\cars_trucks_and_buses_per_1000_persons.csv")
      3 co2=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\co2_emissions_tonnes_per_person.csv")
      4 coal=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\coal_consumption_per_cap.csv")
      5 ele_gen=pd.read_csv("C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\electricity_generation_per_person.csv")

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in parser_f(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_d
```

```
atetime_format, keep_date_col, date_parser, dayfirst, cache_dates, iterator, chunksize, c
ompression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapech
ar, comment, encoding, dialect, error_bad_lines, warn_bad_lines, delim_whitespace, low_me
mory, memory_map, float_precision)
    683         )
    684 
--> 685         return _read(filepath_or_buffer, kwds)
    686 
    687     parser_f.__name__ = name

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in _read(filepath_or_buffer, kwds)
    455 
    456     # Create the parser.
--> 457     parser = TextFileReader(fp_or_buf, **kwds)
    458 
    459     if chunksize or iterator:

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self, f, engine, **kwds)
    893             self.options["has_index_names"] = kwds["has_index_names"]
    894 
--> 895         self._make_engine(self.engine)
    896 
    897     def close(self):

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in _make_engine(self, engine)
   1133     def _make_engine(self, engine="c"):
   1134         if engine == "c":
-> 1135             self._engine = CParserWrapper(self.f, **self.options)
   1136         else:
   1137             if engine == "python":

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self, src, **kwds)
   1915         kwds["usecols"] = self.usecols
   1916 
-> 1917         self._reader = parsers.TextReader(src, **kwds)
   1918         self.unnamed_cols = self._reader.unnamed_cols
   1919 

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_source()

FileNotFoundError: [Errno 2] File b'C:\\Users\\NAMAN\\Desktop\\MLass\\co2_prediction\\car
s_trucks_and_buses_per_1000_persons.csv' does not exist: b'C:\\Users\\NAMAN\\Desktop\\MLa
ss\\co2_prediction\\cars_trucks_and_buses_per_1000_persons.csv'
```

In [ ]:

```
cars.head()
```

Out[ ]:

	geo	2002	2003	2004	2005	2006	2007
0	Afghanistan	NaN	NaN	NaN	NaN	NaN	22.8
1	Albania	73.0	NaN	85.0	87.5	97.3	102.0
2	Algeria	NaN	88.0	89.0	91.0	NaN	NaN
3	Angola	NaN	NaN	NaN	NaN	NaN	39.6
4	Argentina	NaN	NaN	NaN	NaN	NaN	314.0

In [ ]:

```
co2.head()
```

Out[ ]:

	geo	1800	1801	1802	1803	1804	1805	1806	1807	1808	...	2005	2006	2007	2008	2009	2010	2011
0	Afghanistan	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.0529	0.0637	0.0854	0.154	0.242	0.294	0.412

	geo	1800	1801	1802	1803	1804	1805	1806	1807	1808	...	2005	2006	2007	2008	2009	2010	2011
1	Albania	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1.3800	1.2800	1.3000	1.400	1.480	1.560	1.790
2	Algeria	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	3.2200	2.9900	3.1900	3.160	3.420	3.300	3.290
3	Andorra	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	7.3000	6.7500	6.5200	6.430	6.120	6.120	5.870
4	Angola	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.9800	1.1000	1.2000	1.180	1.230	1.240	1.250

5 rows × 216 columns

In [ ]:

```
coal.head()
```

Out[ ]:

	geo	1965	1966	1967	1968	1969	1970	1971	1972	1973	...	2007	2008	2009
0	Algeria	0.00554	0.00524	0.00389	0.0040	0.00495	0.0057	0.00154	0.0013	0.00146	...	0.02210	0.02170	0.01370
1	Argentina	0.03570	0.03690	0.03560	0.0282	0.03710	0.0409	0.03310	0.0291	0.03010	...	0.03050	0.03450	0.02340
2	Australia	1.53000	1.55000	1.55000	1.5500	1.57000	1.5500	1.53000	1.5700	1.61000	...	2.51000	2.57000	2.44000
3	Austria	0.69600	0.66000	0.62100	0.6100	0.59700	0.6390	0.58300	0.5270	0.52200	...	0.47000	0.45000	0.34400
4	Azerbaijan	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.00014	0.00079	0.00046

5 rows × 53 columns

In [ ]:

```
ele_gen.head()
```

Out[ ]:

	geo	1985	1986	1987	1988	1989	1990	1991	1992	1993	...	2007	2008	2009	2010	2011
0	Algeria	544.0	559.0	532.0	568.0	607.0	621.0	653.0	673	699	...	1090	1150	1220	1270	1440
1	Argentina	1490.0	1590.0	1660.0	1670.0	1580.0	1560.0	1620.0	1680	1810	...	2880	3190	3180	3210	3110
2	Australia	7860.0	8100.0	8360.0	8670.0	9020.0	9130.0	9150.0	9230	9350	...	11600	11500	11500	11300	11400
3	Austria	5850.0	5860.0	6610.0	6400.0	6530.0	6530.0	6620.0	6540	6670	...	7800	8030	8250	8450	7780
4	Azerbaijan	3110.0	3180.0	3320.0	3360.0	3270.0	3200.0	3170.0	2630	2520	...	2500	2450	2110	2070	2220

5 rows × 33 columns

Seeing above files we are sure that all of our files would have been read properly so no need to head all of them

We see that all of our columns are years and country name and we need only year 2014 and coury name so we will drop other years

In [ ]:

```
co22=co2[['geo','2014']]
coal2=coal[['geo','2014']]
ele_gen2=ele_gen[['geo','2014']]
ele_use2=ele_use[['geo','2014']]
forest2=forest[['geo','2014']]
income2=income[['geo','2014']]
industry2=industry[['geo','2014']]
natural2=natural[['geo','2014']]
oil_con2=oil_con[['geo','2014']]
oil_pro2=oil_pro[['geo','2014']]
```

```
year2=year[['geo','2014']]
```

Reading above data we find that cars and hydro do not have column for 2014 so we will not be using these files

In [ ]:

```
# Now checking first 3 files for first 5 rows
co22.head()
```

Out[ ]:

	geo	2014
0	Afghanistan	0.299
1	Albania	1.960
2	Algeria	3.720
3	Andorra	5.830
4	Angola	1.290

In [ ]:

```
coal2.head()
```

Out[ ]:

	geo	2014
0	Algeria	0.00458
1	Argentina	0.03460
2	Australia	1.82000
3	Austria	0.34700
4	Azerbaijan	0.00017

In [ ]:

```
ele_gen2.head()
```

Out[ ]:

	geo	2014
0	Algeria	1640
1	Argentina	3290
2	Australia	10500
3	Austria	7540
4	Azerbaijan	2600

In [ ]:

```
# Checking for missing values
year2.isnull().sum()
```

Out[ ]:

```
geo     0
2014    0
dtype: int64
```

Hence we sucessfuly dropped waste columns.

Our next step will be to merge these files using geo as our common column

In [ ]:

```
df1 = pd.merge(co22, coal2, how='outer', on='geo')
df2 = pd.merge(df1,ele_gen2, how='outer', on='geo')
df3 = pd.merge(df2, ele_use2, how='outer', on='geo')
df4 = pd.merge(df3, forest2,  how='outer', on='geo')
df5 = pd.merge(df4, income2,  how='outer', on='geo')
df6 = pd.merge(df5, industry2,  how='outer', on='geo')
df7 = pd.merge(df6, natural2, how='outer', on='geo')
df8 = pd.merge(df7, oil_con2, how='outer', on='geo')
df9 = pd.merge(df8, oil_pro2, how='outer', on='geo')
df = pd.merge(df9, year2, how='outer', on='geo')
df.head()
```

Out[ ]:

	geo	2014_x	2014_y	2014_x	2014_y	2014_x	2014_y	2014_x	2014_y	2014_x	2014_y	2014
0	Afghanistan	0.299	NaN	NaN	NaN	2.07	1780.0	21.10	NaN	NaN	NaN	9810.0
1	Albania	1.960	NaN	NaN	2310.0	28.20	10700.0	21.50	NaN	NaN	NaN	5720.0
2	Algeria	3.720	0.00458	1640.0	1360.0	0.82	13500.0	42.30	1.92	0.452	1.76	145000.0
3	Andorra	5.830	NaN	NaN	NaN	34.00	44900.0	9.91	NaN	NaN	NaN	462.0
4	Angola	1.290	NaN	NaN	312.0	46.50	6260.0	NaN	NaN	NaN	3.08	34800.0

In [ ]:

```
df.columns
```

Out[ ]:

```
Index(['geo', '2014_x', '2014_y', '2014_x', '2014_y', '2014_x', '2014_y',
       '2014_x', '2014_y', '2014_x', '2014_y', '2014'],
      dtype='object')
```

In [ ]:

```
df.describe
```

Out[ ]:

```
<bound method NDFrame.describe of             geo  2014_x   2014_y  2014_x  2014_y  201
4_x   2014_y  2014_x  \
0    Afghanistan   0.299      NaN      NaN     NaN    2.07   1780.0   21.10
1        Albania   1.960      NaN      NaN  2310.0   28.20  10700.0   21.50
2        Algeria   3.720  0.00458  1640.0  1360.0    0.82  13500.0   42.30
3        Andorra   5.830      NaN      NaN     NaN   34.00  44900.0    9.91
4         Angola   1.290      NaN      NaN   312.0   46.50   6260.0     NaN
..           ...     ...      ...      ...     ...     ...      ...     ...
189        Yemen   0.865      NaN      NaN   216.0    1.04   3770.0   44.00
190       Zambia   0.288      NaN      NaN   707.0   65.70   3630.0   32.90
191     Zimbabwe   0.780      NaN      NaN   537.0   37.20   1910.0   22.50
192   San Marino     NaN      NaN      NaN     NaN    0.00  39100.0     NaN
193       Monaco     NaN      NaN      NaN     NaN     NaN  58300.0     NaN

     2014_y  2014_x  2014_y      2014
0       NaN     NaN     NaN    9810.0
1       NaN     NaN     NaN    5720.0
2      1.92   0.452   1.760  145000.0
3       NaN     NaN     NaN     462.0
4       NaN     NaN   3.080   34800.0
..      ...     ...     ...       ...
189    0.32     NaN   0.256   22700.0
190     NaN     NaN     NaN    4500.0
191     NaN     NaN     NaN   12000.0
```

```
192      NaN      NaN      NaN        NaN
193      NaN      NaN      NaN        NaN

[194 rows x 12 columns]>
```

In [ ]:

```
# Now renaming our columns
df.columns=[ "geo","co2","coal","ele_gen","ele_use","forest","income","industry","natural
","oil_con","oil_pro","year"]
df.head()
```

Out[ ]:

	geo	co2	coal	ele_gen	ele_use	forest	income	industry	natural	oil_con	oil_pro	year
0	Afghanistan	0.299	NaN	NaN	NaN	2.07	1780.0	21.10	NaN	NaN	NaN	9810.0
1	Albania	1.960	NaN	NaN	2310.0	28.20	10700.0	21.50	NaN	NaN	NaN	5720.0
2	Algeria	3.720	0.00458	1640.0	1360.0	0.82	13500.0	42.30	1.92	0.452	1.76	145000.0
3	Andorra	5.830	NaN	NaN	NaN	34.00	44900.0	9.91	NaN	NaN	NaN	462.0
4	Angola	1.290	NaN	NaN	312.0	46.50	6260.0	NaN	NaN	NaN	3.08	34800.0

In [ ]:

```
df.shape
```

Out[ ]:

```
(194, 12)
```

Step 2: Data Cleaning and Manipulation

In [ ]:

```
# Checking for missing values
df.isnull().sum()
```

Out[ ]:

```
geo            0
co2            2
coal         129
ele_gen      129
ele_use       57
forest         3
income         1
industry      11
natural      145
oil_con      129
oil_pro      145
year           2
dtype: int64
```

In [ ]:

```
#Checking the percentage of missing values
round(100*(df.isnull().sum()/len(df.index)), 2)
```

Out[ ]:

```
geo           0.00
co2           1.03
coal         66.49
ele_gen      66.49
ele_use      29.38
forest        1.55
income        0.52
industry      5.67
```

```
natural      74.74
oil_con      66.49
oil_pro      74.74
year          1.03
dtype: float64
```

As we can see:

coal 66.49

ele_gen 66.49

natural 74.74

oil_con 66.49

oil_pro 74.74

These have more than 50% of missing values. So there is no need to consider them.

Whereas,

co2 1.03

forest 1.55

income 0.52

industry 5.67

year 1.03

ele_use 29.38

So, we will be filling it with mean.

In [ ]:

```
# Dropping useless columns
df = df.drop(['coal','ele_gen','natural','oil_con','oil_pro'], axis=1)
```

In [ ]:

```
df['co2'].mean()
```

Out[ ]:

```
4.44008489583333
```

In [ ]:

```
df['co2'].fillna(value = (df['co2'].mean()), inplace=True)
```

In [ ]:

```
df['forest'].mean()
```

Out[ ]:

```
31.907068062827214
```

In [ ]:

```
df['forest'].fillna(value = (df['forest'].mean()), inplace=True)
```

In [ ]:

```
df['income'].mean()
```

Out[ ]:

```
17210.39896373057
```

In [ ]:

```
df['income'].fillna(value = (df['income'].mean()), inplace=True)
```

In [ ]:

```
df['industry'].mean()
```

Out[ ]:

```
26.761092896174873
```

In [ ]:

```
df['industry'].fillna(value = (df['industry'].mean()), inplace=True)
```

In [ ]:

```
df['year'].mean()
```

Out[ ]:

```
175992.54166666666
```

In [ ]:

```
df['year'].fillna(value = (df['year'].mean()), inplace=True)
```

In [ ]:

```
df['ele_use'].mean()
```

Out[ ]:

```
4253.62189781022
```

In [ ]:

```
df['ele_use'].fillna(value = (df['ele_use'].mean()), inplace=True)
```

In [ ]:

```
df=df.dropna()
```

In [ ]:

```
df.isnull().sum()
```

Out[ ]:

```
geo         0
co2         0
ele_use     0
forest      0
income      0
industry    0
year        0
dtype: int64
```

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 194 entries, 0 to 193
Data columns (total 7 columns):
geo         194 non-null object
co2         194 non-null float64
```

```
ele_use       194 non-null float64
forest        194 non-null float64
income        194 non-null float64
industry      194 non-null float64
year          194 non-null float64
dtypes: float64(6), object(1)
memory usage: 12.1+ KB
```

In [ ]:

```
df.columns
```

Out[ ]:

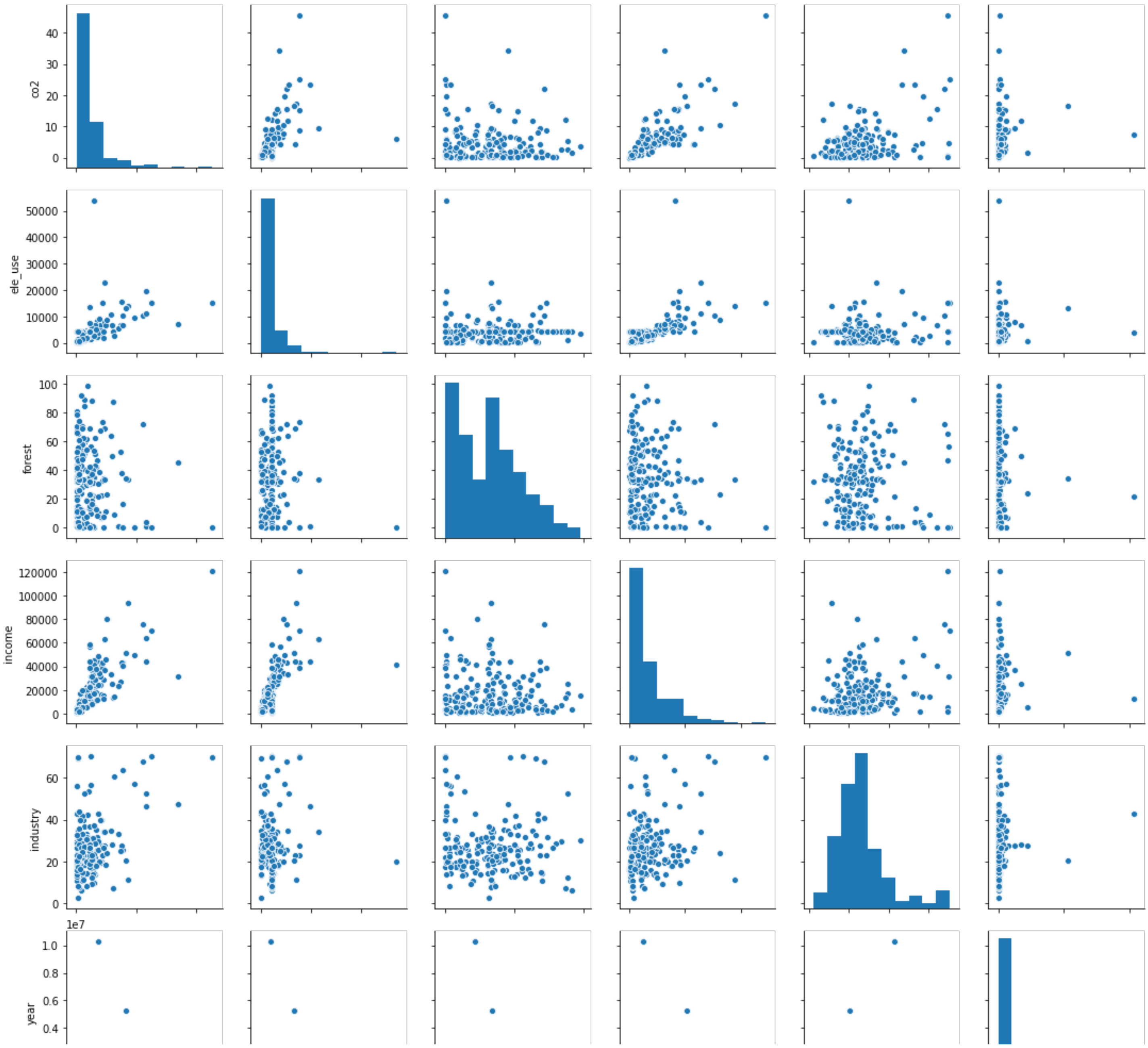
```
Index(['geo', 'co2', 'ele_use', 'forest', 'income', 'industry', 'year'], dtype='object')
```

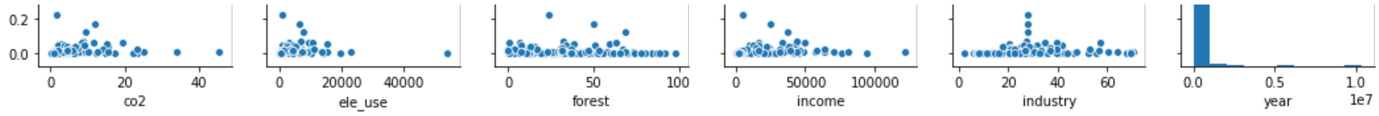
Hence, we managed all the missing values and manipulated the data.

Step 3: Data Visualisation

In [ ]:

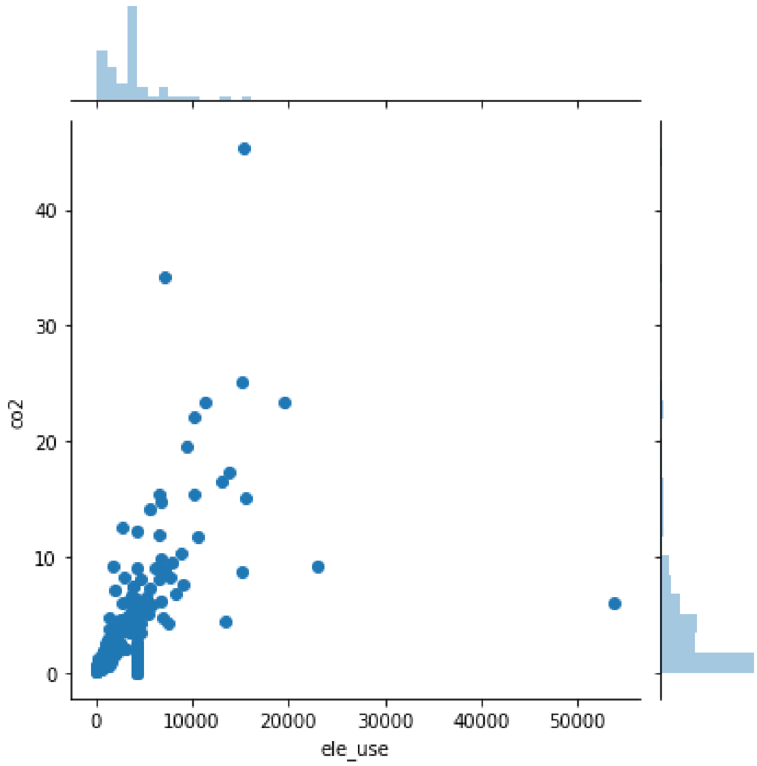
```
# Visualizing our data
sns.pairplot(df)
plt.show()
```

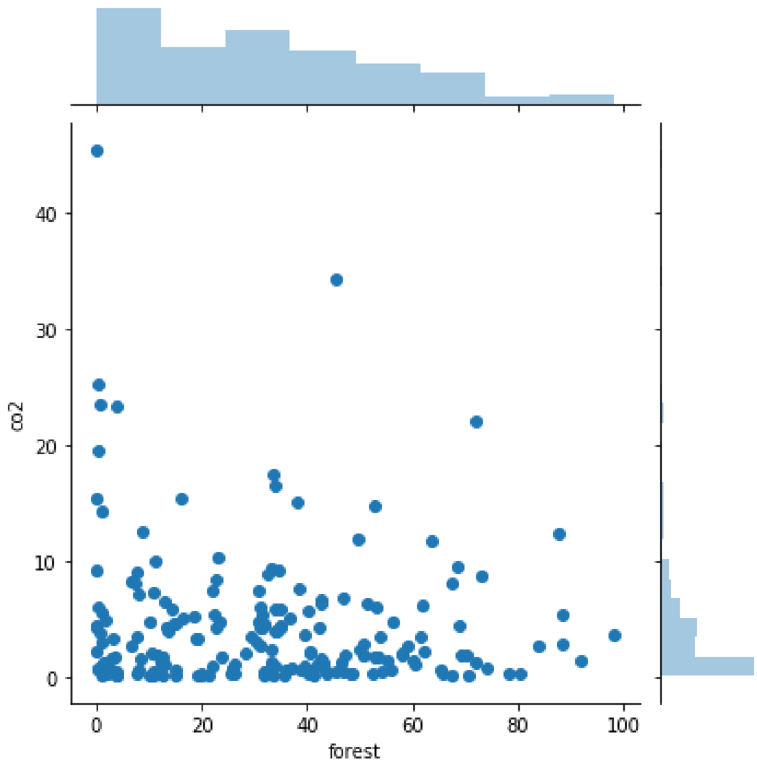


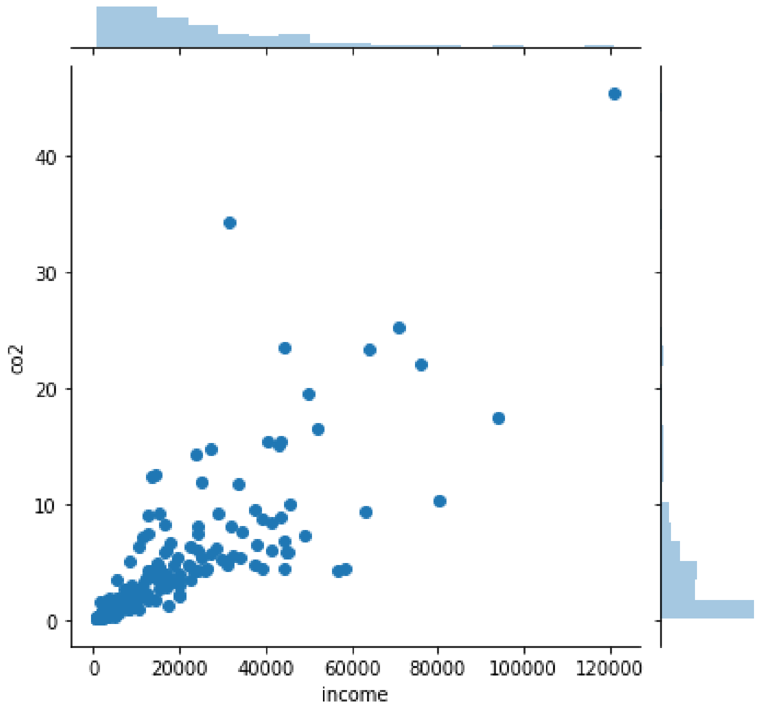


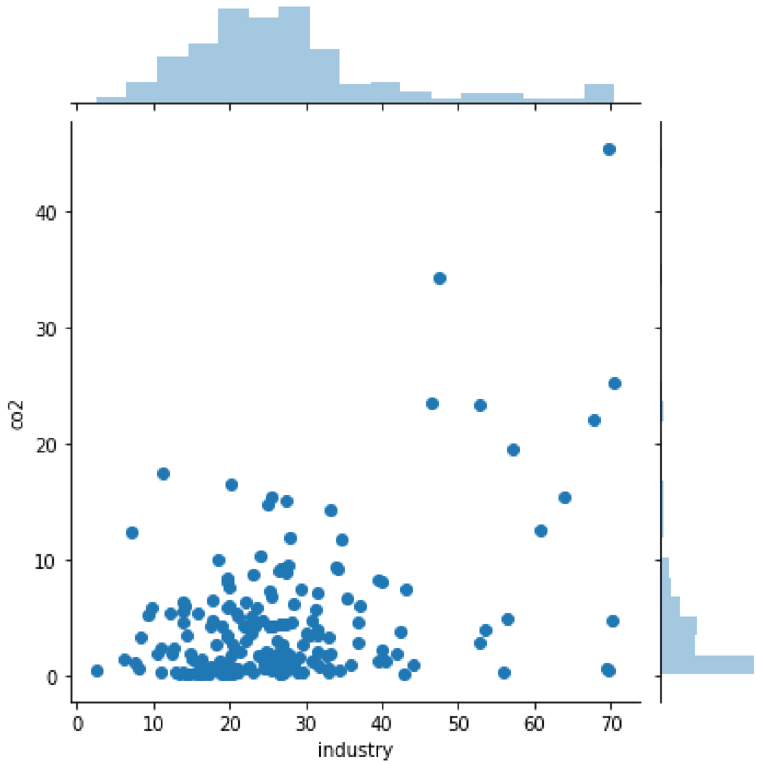
In [ ]:

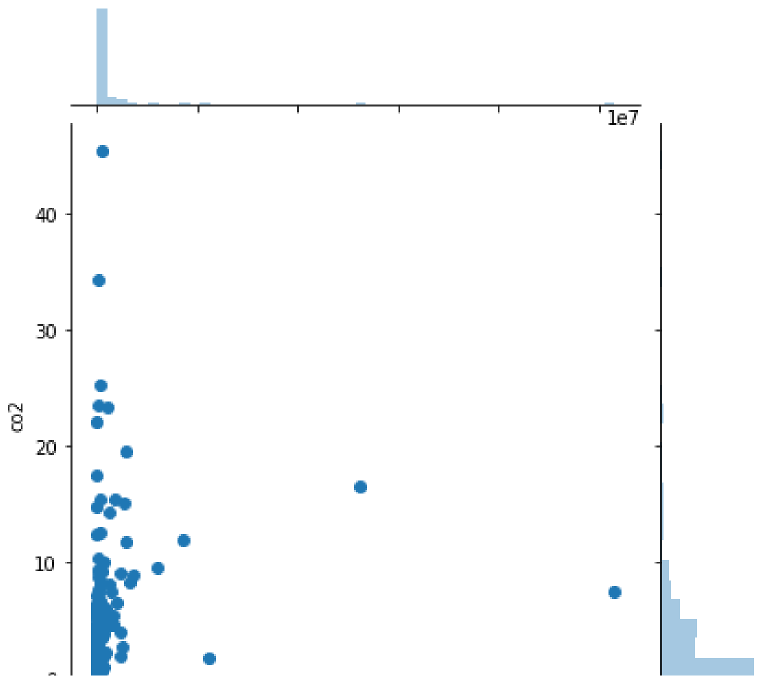
```
sns.jointplot('ele_use', 'co2', df)
plt.show()
sns.jointplot('forest', 'co2', df)
plt.show()
sns.jointplot('income', 'co2', df)
plt.show()
sns.jointplot('industry', 'co2', df)
plt.show()
sns.jointplot('year', 'co2', df)
plt.show()
```

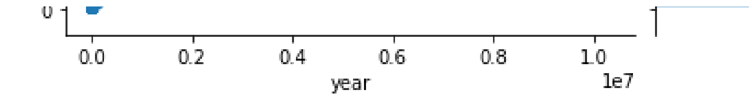





co2
income
0
10
20
30
40
20000
40000
60000
80000
100000
120000


co2
industry
0
10
20
30
40
50
60
70


co2
1e7
0
10
20
30
40



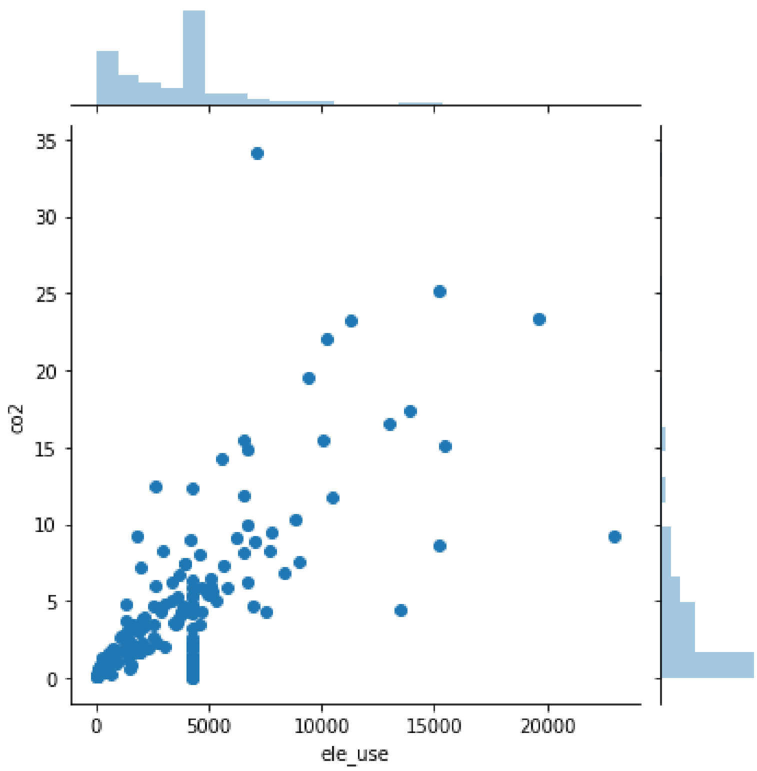
Now treating the outliers

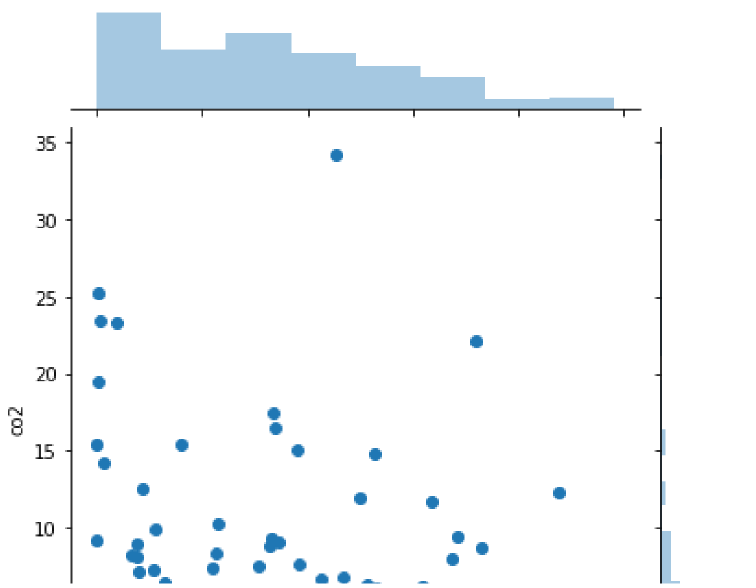
In [ ]:

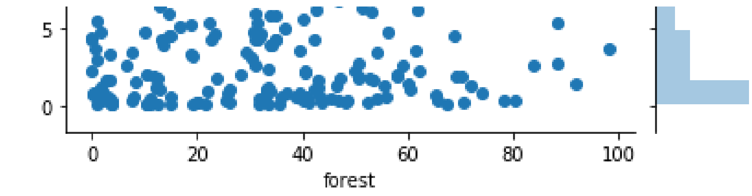
```
#treating outliers
df=df[(df.ele_use<50000)]
df=df[(df.income<100000)]
```

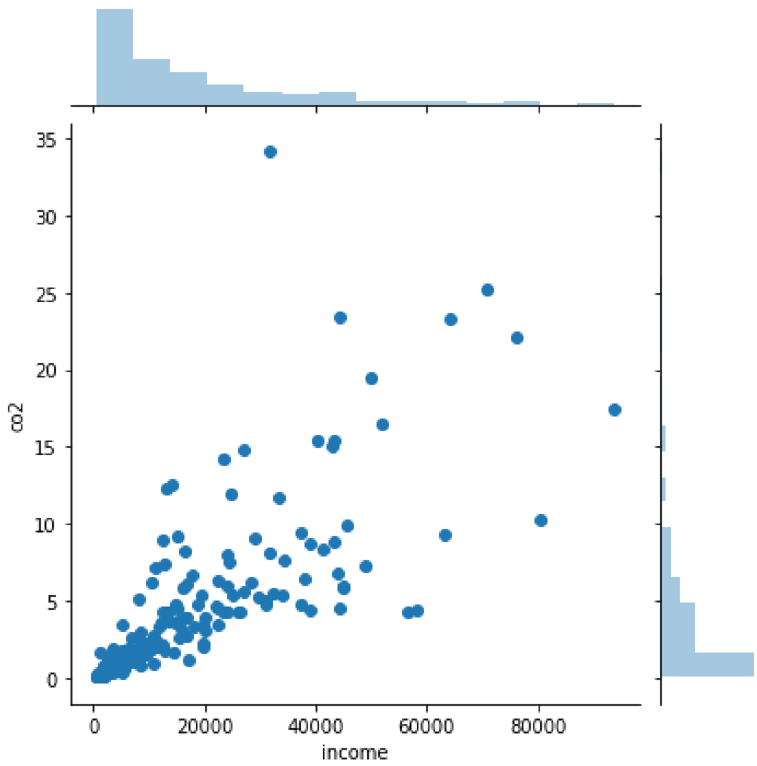
In [ ]:

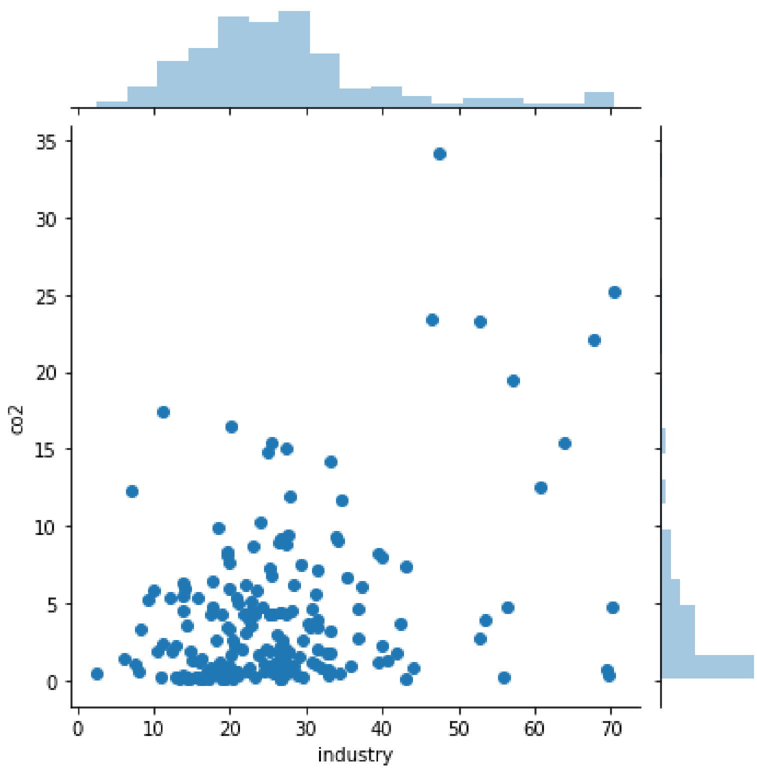
```
sns.jointplot('ele_use', 'co2', df)
plt.show()
sns.jointplot('forest', 'co2', df)
plt.show()
sns.jointplot('income', 'co2', df)
plt.show()
sns.jointplot('industry', 'co2', df)
plt.show()
sns.jointplot('year', 'co2', df)
plt.show()
```

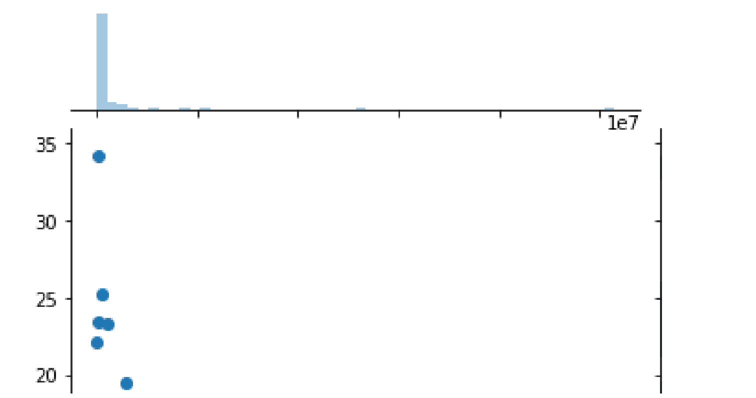


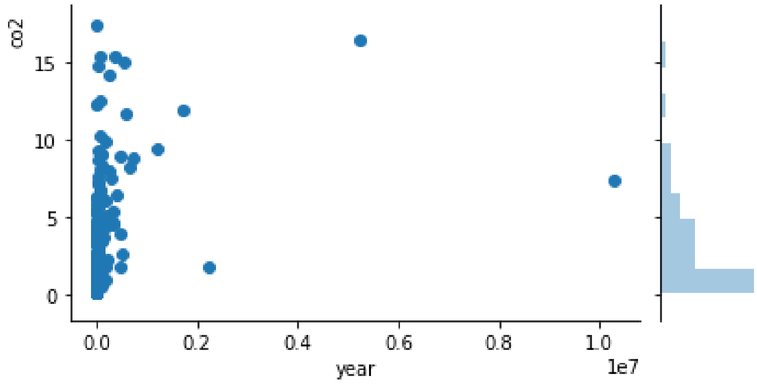



forest


co2
income

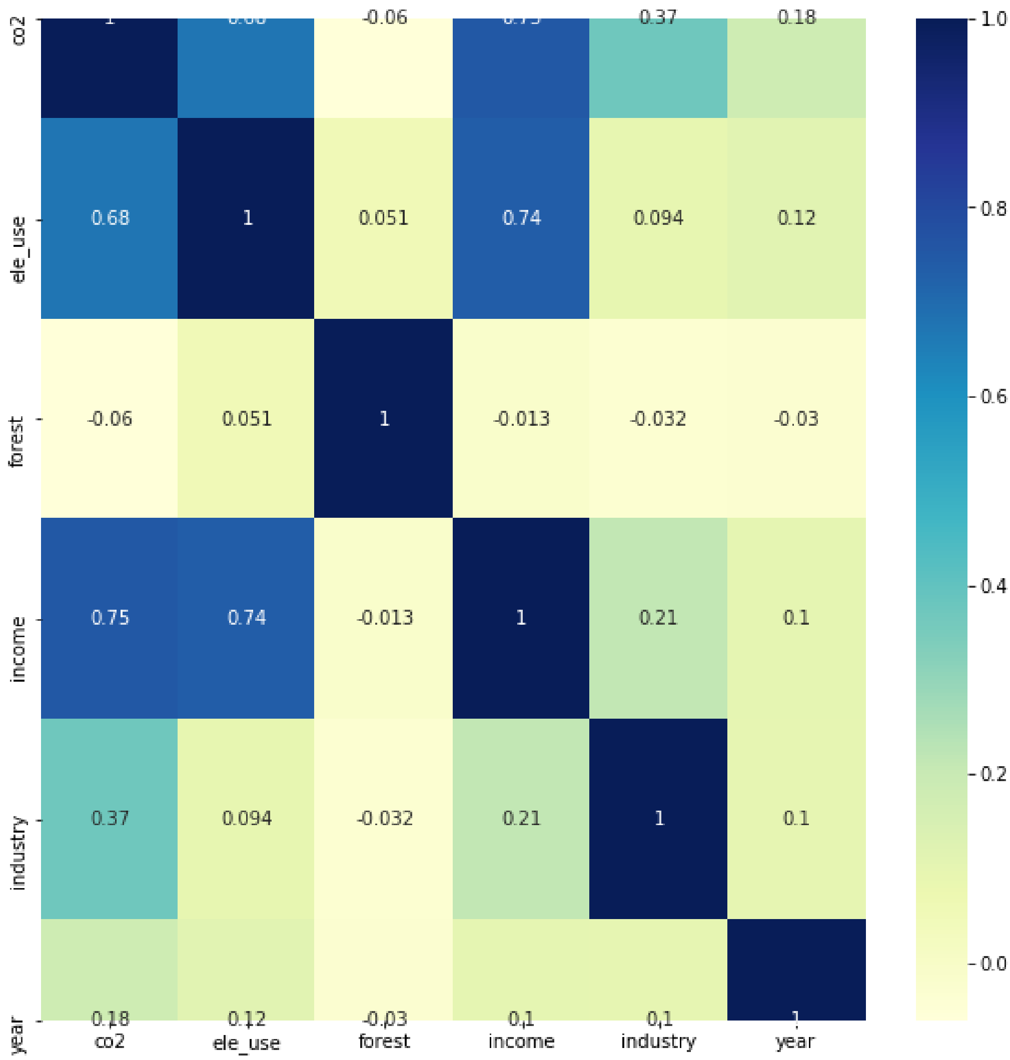

co2
industry


1e7



In [ ]:

```
# Now checking the correlation of our data
#Checking the correlation of our target variable with other variables
plt.figure(figsize = (10, 10))
sns.heatmap(df.corr(), annot = True, cmap="YlGnBu")
plt.show()
```



Step 4: Data Prepration

In [ ]:

```
# Splitting our data into target and independent variables
x= df.drop(['geo','co2'], axis=1)
x.head()
```

Out[ ]:

	ele_use	forest	income	industry	year
0	4253.621898	2.07	1780.0	21.100000	9810.0

	ele_use	forest	income	industry	year
1	2310.000000	28.20	10700.0	21.500000	5720.0
2	1360.000000	0.82	13500.0	42.300000	145000.0
3	4253.621898	34.00	44900.0	9.910000	462.0
4	312.000000	46.50	6260.0	26.761093	34800.0

In [ ]:

```
y=df['co2']
y.head()
```

Out[ ]:

```
0    0.299
1    1.960
2    3.720
3    5.830
4    1.290
Name: co2, dtype: float64
```

In [ ]:

```
# scaling the features
from sklearn.preprocessing import scale

# storing column names in cols, since column names are (annoyingly) lost after
# scaling (the df is converted to a numpy array)
cols = x.columns
x = pd.DataFrame(scale(x))
x.columns = cols
x.columns
```

Out[ ]:

```
Index(['ele_use', 'forest', 'income', 'industry', 'year'], dtype='object')
```

In [ ]:

```
# split into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y,
                                                    train_size=0.7,
                                                    test_size = 0.3, random_state=100)
```

Step 5:Model Building and Evaluation

Using lasso regression

In [ ]:

```
lasso = Lasso()

# list of alphas to tune
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1,
 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50, 100, 500, 1000 ]}

# cross validation
folds = 5
model_cv = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 140 out of 140 | elapsed:    0.4s finished
```

Out[ ]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=Lasso(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=1000, normalize=False, positive=False,
                             precompute=False, random_state=None,
                             selection='cyclic', tol=0.0001, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
                                   4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50,
                                   100, 500, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='neg_mean_absolute_error', verbose=1)
```

In [ ]:

```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results.head()
```

Out[ ]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	s
0	0.001995	0.000630	0.001196	0.000400	0.0001	{'alpha': 0.0001}	-1.623682	-3.279831	
1	0.001799	0.000401	0.000802	0.000401	0.001	{'alpha': 0.001}	-1.623824	-3.275315	
2	0.001601	0.000494	0.000594	0.000485	0.01	{'alpha': 0.01}	-1.625448	-3.229997	
3	0.001203	0.000396	0.001010	0.000017	0.05	{'alpha': 0.05}	-1.632663	-3.033372	
4	0.001590	0.000497	0.000604	0.000493	0.1	{'alpha': 0.1}	-1.643105	-2.799111	

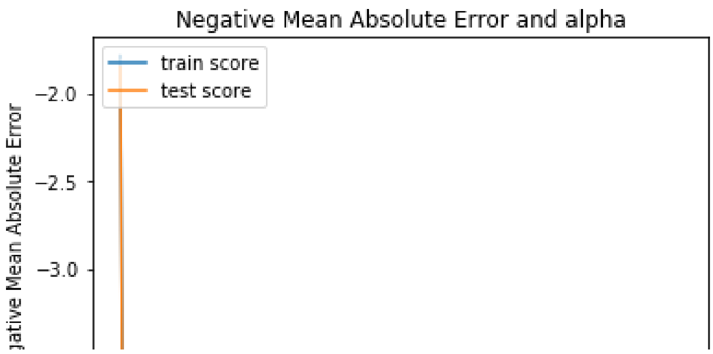
5 rows × 21 columns

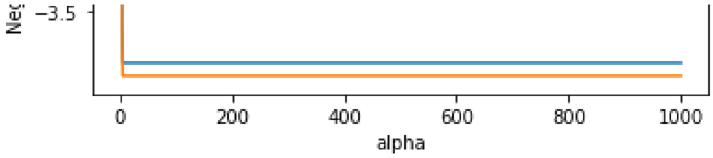
In [ ]:

```
# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('float32')

# plotting
plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'])
plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'])
plt.xlabel('alpha')
plt.ylabel('Negative Mean Absolute Error')

plt.title("Negative Mean Absolute Error and alpha")
plt.legend(['train score', 'test score'], loc='upper left')
plt.show()
```





In [ ]:

```
cv_results
```

Out[ ]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score
0	0.001995	0.000630	0.001196	0.000400	0.0001	{'alpha': 0.0001}	-1.623682	-3.279831
1	0.001799	0.000401	0.000802	0.000401	0.0010	{'alpha': 0.001}	-1.623824	-3.275315
2	0.001601	0.000494	0.000594	0.000485	0.0100	{'alpha': 0.01}	-1.625448	-3.229997
3	0.001203	0.000396	0.001010	0.000017	0.0500	{'alpha': 0.05}	-1.632663	-3.033372
4	0.001590	0.000497	0.000604	0.000493	0.1000	{'alpha': 0.1}	-1.643105	-2.799111
5	0.001787	0.000393	0.000800	0.000400	0.2000	{'alpha': 0.2}	-1.664726	-2.788482
6	0.001987	0.000622	0.000793	0.000396	0.3000	{'alpha': 0.3}	-1.686347	-2.820155
7	0.001795	0.000974	0.001003	0.000013	0.4000	{'alpha': 0.4}	-1.712886	-2.881626
8	0.001773	0.000404	0.000408	0.000500	0.5000	{'alpha': 0.5}	-1.749590	-2.914875
9	0.001569	0.000490	0.000806	0.000403	0.6000	{'alpha': 0.6}	-1.786293	-2.930292
10	0.001578	0.000486	0.000811	0.000406	0.7000	{'alpha': 0.7}	-1.824228	-2.952786
11	0.002196	0.000734	0.000804	0.000402	0.8000	{'alpha': 0.8}	-1.871696	-2.978052
12	0.001190	0.000403	0.000997	0.000028	0.9000	{'alpha': 0.9}	-1.938238	-3.003319
13	0.001595	0.000495	0.000605	0.000494	1.0000	{'alpha': 1.0}	-2.016176	-3.028585
14	0.001980	0.000024	0.000197	0.000394	2.0000	{'alpha': 2.0}	-2.866689	-3.306513
15	0.001789	0.000398	0.000398	0.000487	3.0000	{'alpha': 3.0}	-3.497085	-3.653909
16	0.002203	0.000411	0.000204	0.000408	4.0000	{'alpha': 4.0}	-4.156454	-4.062773
17	0.001213	0.000406	0.000784	0.000392	5.0000	{'alpha': 5.0}	-4.156454	-4.205025
18	0.001397	0.000478	0.000799	0.000399	6.0000	{'alpha': 6.0}	-4.156454	-4.205025
19	0.001983	0.000899	0.001214	0.000393	7.0000	{'alpha': 7.0}	-4.156454	-4.205025
20	0.001985	0.000618	0.001191	0.000740	8.0000	{'alpha': 8.0}	-4.156454	-4.205025
21	0.001990	0.000627	0.000801	0.000747	9.0000	{'alpha': 9.0}	-4.156454	-4.205025
22	0.001601	0.000508	0.001004	0.000017	10.0000	{'alpha': 10.0}	-4.156454	-4.205025

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	s
23	0.001785	0.000746	0.001005	0.000011	20.0000	{'alpha': 20}	-4.156454	-4.205025	
24	0.001614	0.000494	0.000794	0.000397	50.0000	{'alpha': 50}	-4.156454	-4.205025	
25	0.002008	0.000631	0.000399	0.000488	100.0000	{'alpha': 100}	-4.156454	-4.205025	
26	0.001581	0.000479	0.000803	0.000402	500.0000	{'alpha': 500}	-4.156454	-4.205025	
27	0.001388	0.000480	0.000793	0.000397	1000.0000	{'alpha': 1000}	-4.156454	-4.205025	

28 rows × 21 columns

In [ ]:

```
lasso = Lasso(alpha = 0.1)
lasso.fit(X_train, y_train)
Y_pred1 = lasso.predict(X_train)

#Printing Lasso Coefficients
print('Lasso Coefficients',lasso.coef_,sep='\n')

# Calculate Mean Squared Error
mean_squared_error = np.mean((Y_pred1 - y_train)**2)
print("Mean squared error on train set", mean_squared_error)

Y_pred2 = lasso.predict(X_test)
mean_squared_error1 = np.mean((Y_pred2 - y_test)**2)

print("Mean squared error on test set", mean_squared_error1)
```

```
Lasso Coefficients
[ 2.508523   -0.29428438  2.02843619  1.34395783  0.20211723]
Mean squared error on train set 9.293386945837502
Mean squared error on test set 10.99298676683838
```

Mean square error on test set almost equal train set