```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

pandas : It used for data manipulation and analysis. numPy : It is a powerful Python library for numerical computing. seaborn : It provides a high-level interface for creating attractive and informative statistical graphics. matplotlib.pyplot : It provides a MATLAB-like interface for creating basic plots and visualizations.

```
df=pd.read_csv("nuclear_explosions.csv")
```

pd.read_csv is used to read the data from the csv files df is the variable in which the data is getting stored from the csv files

```
df
```

	Location.Country	Location.Region	Data.Source	Location.Cordinates.La
0	USA	Alamogordo	DOE	
1	USA	Hiroshima	DOE	
2	USA	Nagasaki	DOE	
3	USA	Bikini	DOE	
4	USA	Bikini	DOE	
...	...	...	...	
2041	CHINA	Lop Nor	HFS	
2042	INDIA	Pokhran	HFS	
2043	INDIA	Pokhran	NRD	
2044	PAKIST	Chagai	HFS	
2045	PAKIST	Kharan	HFS	

2046 rows × 16 columns

Double-click (or enter) to edit

```
df.head()
```

	Location.Country	Location.Region	Data.Source	Location.Cordinates.Latit
0	USA	Alamogordo	DOE	3
1	USA	Hiroshima	DOE	3
2	USA	Nagasaki	DOE	3
3	USA	Bikini	DOE	1
4	USA	Bikini	DOE	1

head() --> function is used to read the starting 5 lines from the dataset

```
df.tail()
```

	Location.Country	Location.Region	Data.Source	Location.Cordinates.La
2041	CHINA	Lop Nor	HFS	
2042	INDIA	Pokhran	HFS	
2043	INDIA	Pokhran	NRD	
2044	PAKIST	Chagai	HFS	
2045	PAKIST	Kharan	HFS	

tail() --> tail() function is used to read the ending 5 lines from the dataset

```
df.sample(8)
```

	Location.Country	Location.Region	Data.Source	Location.Cordinates.La
1574	USSR	Bashki Russ	MTM	
95	USSR	Semi Kazakh	DOE	
252	USSR	Nz Russ	DOE	
406	USA	Nts	DOE	
1396	USA	Nts	DOE	
1502	FRANCE	Mururoa	WTN	
1304	USSR	Jakuts Russ	MTM	
2030	CHINA	Lop Nor	HFS	

sample(8) --> function is used to get the sample of 8 rows from the given dataset

```
df.dtypes
```

```
Location.Country                    object
Location.Region                     object
Data.Source                         object
Location.Cordinates.Latitude       float64
Location.Cordinates.Longitude      float64
Data.Magnitude.Body                float64
Data.Magnitude.Surface             float64
Location.Cordinates.Depth          float64
Data.Yeild.Lower                   float64
Data.Yeild.Upper                   float64
Data.Purpose                        object
Data.Name                           object
Data.Type                           object
Date.Day                             int64
Date.Month                           int64
Date.Year                            int64
dtype: object
```

dtypes --> It is used to get the datatype of the specified columns from the given datasets

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2046 entries, 0 to 2045
Data columns (total 16 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   Location.Country               2046 non-null   object 
 1   Location.Region                2046 non-null   object 
 2   Data.Source                    2046 non-null   object 
 3   Location.Cordinates.Latitude   2046 non-null   float64
 4   Location.Cordinates.Longitude  2046 non-null   float64
 5   Data.Magnitude.Body            2046 non-null   float64
 6   Data.Magnitude.Surface         2046 non-null   float64
 7   Location.Cordinates.Depth      2046 non-null   float64
 8   Data.Yeild.Lower               2046 non-null   float64
 9   Data.Yeild.Upper               2046 non-null   float64
 10  Data.Purpose                   2046 non-null   object 
 11  Data.Name                      2046 non-null   object 
 12  Data.Type                      2046 non-null   object 
 13  Date.Day                       2046 non-null   int64  
 14  Date.Month                     2046 non-null   int64  
 15  Date.Year                      2046 non-null   int64  
dtypes: float64(7), int64(3), object(6)
memory usage: 255.9+ KB
```

info() --> It is used to get the Information of about each and every columns It will also tell about the no. of the cells which are non-empty by displaying their number

```
df.count()
```

```
Location.Country                 2046
Location.Region                  2046
Data.Source                      2046
Location.Cordinates.Latitude     2046
Location.Cordinates.Longitude    2046
Data.Magnitude.Body              2046
Data.Magnitude.Surface           2046
Location.Cordinates.Depth        2046
Data.Yeild.Lower                 2046
Data.Yeild.Upper                 2046
Data.Purpose                     2046
Data.Name                        2046
Data.Type                        2046
Date.Day                         2046
Date.Month                       2046
Date.Year                        2046
dtype: int64
```

count() --> It is used to count the non-empty cells of each column It will display the number of that sooo...

```
df.shape
```

```
(2046, 16)
```

shape --> It is used to determine the dimensions or the size of an array or DataFrame. 1715 --> Rows 9 --> Columns

```
df.columns
```

```
Index(['Location.Country', 'Location.Region', 'Data.Source',
       'Location.Cordinates.Latitude', 'Location.Cordinates.Longitude',
       'Data.Magnitude.Body', 'Data.Magnitude.Surface',
       'Location.Cordinates.Depth', 'Data.Yeild.Lower',
'Data.Yeild.Upper',
       'Data.Purpose', 'Data.Name', 'Data.Type', 'Date.Day', 'Date.Month',
       'Date.Year'],
      dtype='object')
```

columns --> It will display the column names

```
df['Data.Magnitude.Surface'].mean()
```

```
0.35669599217986314
```

mean() --> It is used to find the mean of the required column data

```
df['Date.Year'].median()
```

```
1970.0
```

median() --> It is used to find the median of the required column data

```
df['Date.Year'].std()
```

```
10.372759916312438
```

std() --> It is used to find the Standard Deviation of the required column data

```
df['Date.Year'].mode()
```

```
0    1962
Name: Date.Year, dtype: int64
```

mode() --> It is used to find the mode of the required column data

```
df.describe()
```

	Location.Cordinates.Latitude	Location.Cordinates.Longitude	Data.Mag
count	2046.000000	2046.000000	
mean	35.462429	-36.015037	
std	23.352702	100.829355	
min	-49.500000	-169.320000	
25%	37.000000	-116.051500	
50%	37.100000	-116.000000	
75%	49.870000	78.000000	
max	75.100000	179.220000	

describe() --> It is used to describe the dataset into the count,mean,std,min,25%,50% (median),75%,max It will only describe and find the values of the dataset columns which are in float or integer

```
df["Location.Country"].value_counts()
```

```
USA       1031
USSR       714
FRANCE     208
UK          45
CHINA       43
INDIA        3
PAKIST       2
Name: Location.Country, dtype: int64
```

value_counts() --> It will count the no. of times the specfic column name present in the dataset

```
df["Location.Cordinates.Depth"].unique() #list of unique entries
```

```
array([-1.000e-01, -6.000e-01, -2.000e-01,  3.000e-02, -8.000e-02,
        0.000e+00, -3.500e-01, -4.000e-01, -5.000e-01, -7.000e-02,
       -3.000e-02, -4.500e-01, -1.000e-03,  1.000e-02, -2.500e-01,
       -1.150e+00, -2.000e+00, -8.000e-01,  2.000e-02, -1.200e+01,
       -1.500e+00, -6.800e+00,  3.000e-01, -2.800e+01,  2.000e-01,
        5.000e-02, -8.500e+01, -4.500e+01, -1.600e+02,  1.000e-01,
       -2.000e-03, -1.500e-01, -2.500e-02, -2.000e-02, -1.000e-02,
        4.500e-01,  2.500e-01,  1.500e-01, -4.000e+02,  6.000e-01,
        3.500e-01,  5.000e-01,  9.000e-01,  7.130e-01,  3.050e-01,
        4.120e-01,  7.650e-01,  7.320e-01,  6.370e-01,  9.120e-01,
        1.311e+00,  4.170e-01,  3.480e-01,  1.265e+00,  8.170e-01,
        3.200e-01,  7.160e-01,  1.451e+00,  6.400e-01,  6.550e-01,
        1.219e+00,  1.167e+00,  8.690e-01,  1.237e+00,  8.790e-01,
        7.800e-01,  3.170e-01,  4.270e-01,  3.310e-01,  6.900e-01,
        5.940e-01,  5.640e-01,  5.180e-01,  7.010e-01,  3.810e-01,
        5.300e-01,  3.850e-01,  3.720e-01,  6.680e-01,  6.580e-01,
        6.330e-01,  6.110e-01,  6.810e-01,  3.880e-01,  4.420e-01,
        5.760e-01,  5.420e-01,  6.890e-01,  3.260e-01,  5.790e-01,
        5.360e-01,  3.350e-01,  5.370e-01,  3.960e-01,  4.640e-01,
        2.290e-01,  2.050e-01,  3.690e-01,  2.710e-01,  3.510e-01,
        6.450e-01,  6.800e-01,  3.660e-01,  4.240e-01,  3.900e-01,
        5.730e-01,  3.540e-01,  3.230e-01,  3.410e-01,  2.040e-01,
        2.940e-01,  2.130e-01,  4.720e-01,  4.450e-01,  4.940e-01,
        6.510e-01,  3.570e-01,  5.700e-01,  2.890e-01,  4.000e-01,
        2.160e-01,  4.510e-01,  3.360e-01,  4.130e-01,  3.040e-01,
        3.430e-01,  5.330e-01,  2.650e-01,  3.840e-01,  6.250e-01,
        4.050e-01,  5.150e-01,  6.610e-01,  6.080e-01,  2.930e-01,
        3.320e-01,  4.150e-01])
```

.unique() --> It is used to count the unique enteries in that column

```
df.sort_values(["Date.Year"])
```

	Location.Country	Location.Region	Data.Source	Location.Cordinates.La
0	USA	Alamogordo	DOE	
1	USA	Hiroshima	DOE	
2	USA	Nagasaki	DOE	
3	USA	Bikini	DOE	
4	USA	Bikini	DOE	
...	...	...	...	
2041	CHINA	Lop Nor	HFS	
2042	INDIA	Pokhran	HFS	
2043	INDIA	Pokhran	NRD	
2044	PAKIST	Chagai	HFS	
2045	PAKIST	Kharan	HFS	

2046 rows × 16 columns

.sort_values() --> It will sort the values and provides u the data in the sorting manner If ascending of descending is not given then by default it will give u the data in the ascending manner/order only

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

```
Location.Country                  0
Location.Region                   0
Data.Source                       0
Location.Cordinates.Latitude      0
Location.Cordinates.Longitude     0
Data.Magnitude.Body               0
Data.Magnitude.Surface            0
Location.Cordinates.Depth         0
Data.Yeild.Lower                  0
Data.Yeild.Upper                  0
Data.Purpose                      0
Data.Name                         0
Data.Type                         0
Date.Day                          0
Date.Month                        0
Date.Year                         0
dtype: int64
```

isnull() --> It will Return u the no. of the empty cells in the row

```
df.drop(["Location.Cordinates.Latitude","Location.Cordinates.Longitude","Data.Ye
```

.drop() --> It is used to delete the specified columns As the no. of the empty cells in the column anzsic_descriptor2 is 105 and category is 179 and this both the columns are not useful to predict the data so deleting both the columns

```
df.head()
```

	Location.Country	Location.Region	Data.Source	Data.Magnitude.Body	Data
0	USA	Alamogordo	DOE	0.0	
1	USA	Hiroshima	DOE	0.0	
2	USA	Nagasaki	DOE	0.0	
3	USA	Bikini	DOE	0.0	
4	USA	Bikini	DOE	0.0	

Checking for the deleted columns

```
df.dropna(inplace=True)
```

dropna() --> It is used to delete the rows which have very less null values and then bring the data in the equal manner.

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

```
Location.Country             0
Location.Region              0
Data.Source                  0
Data.Magnitude.Body          0
Data.Magnitude.Surface       0
Location.Cordinates.Depth    0
Data.Yeild.Lower             0
Data.Purpose                 0
Data.Name                    0
Data.Type                    0
Date.Day                     0
Date.Month                   0
Date.Year                    0
dtype: int64
```

There is no null cell in the rows

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2046 entries, 0 to 2045
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Location.Country           2046 non-null   object 
 1   Location.Region            2046 non-null   object 
 2   Data.Source                2046 non-null   object 
 3   Data.Magnitude.Body        2046 non-null   float64
 4   Data.Magnitude.Surface     2046 non-null   float64
 5   Location.Cordinates.Depth  2046 non-null   float64
 6   Data.Yeild.Lower           2046 non-null   float64
 7   Data.Purpose               2046 non-null   object 
 8   Data.Name                  2046 non-null   object 
 9   Data.Type                  2046 non-null   object 
 10  Date.Day                   2046 non-null   int64  
 11  Date.Month                 2046 non-null   int64  
 12  Date.Year                  2046 non-null   int64  
dtypes: float64(4), int64(3), object(6)
memory usage: 207.9+ KB
```

Double-click (or enter) to edit

DETECTING OUTLIER

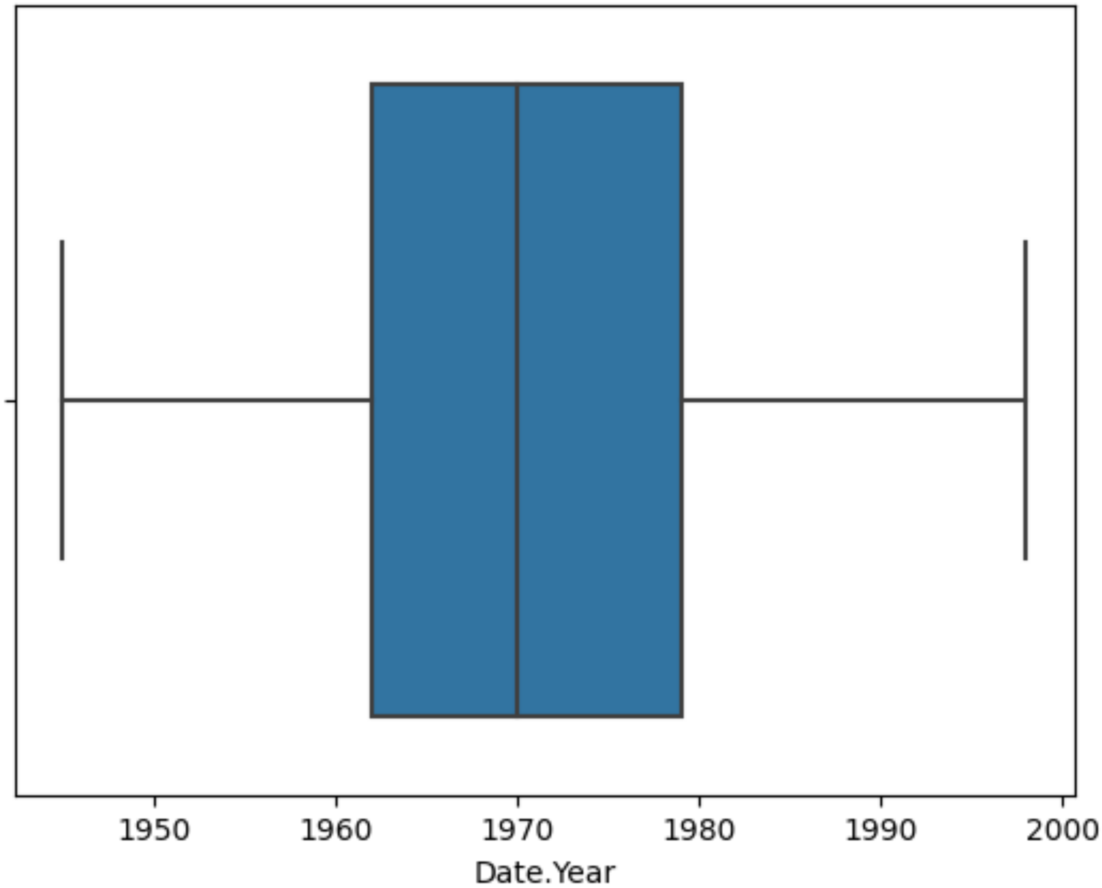
Outlier --> It is an extreme value that falls far outside the typical range of values in a dataset.

It is a **Univarient Analysis**

Bcz we are analyzing using the single variable

```
sns.boxplot(x=df['Date.Year'])
```

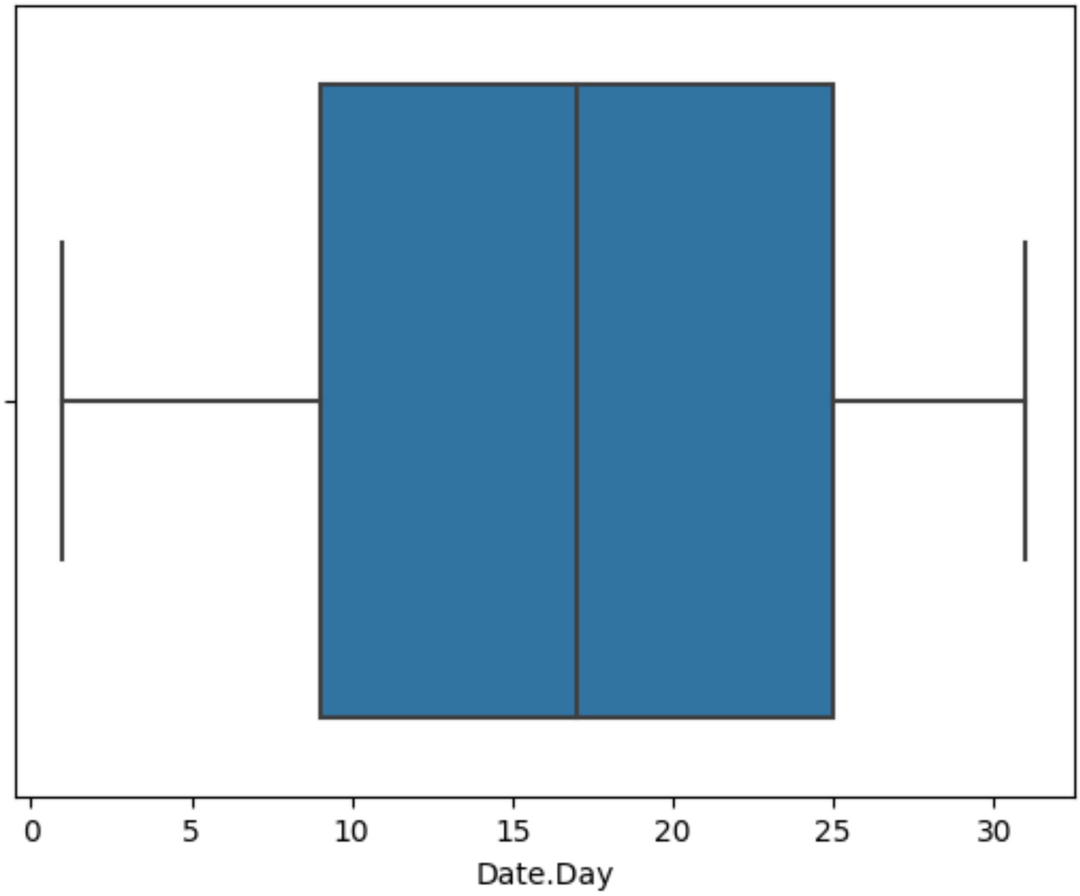
```
<Axes: xlabel='Date.Year'>
```



There is no outlier in the year column

```
sns.boxplot(x=df['Date.Day'])
```

```
<Axes: xlabel='Date.Day'>
```



There are outlier in the data_val column

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
Data.Magnitude.Body           5.1
Data.Magnitude.Surface        0.0
Location.Cordinates.Depth     0.0
Data.Yeild.Lower             20.0
Date.Day                     16.0
Date.Month                    5.0
Date.Year                    17.0
dtype: float64
<ipython-input-41-4cd70db7bbb2>:1: FutureWarning: The default value of nume
  Q1 = df.quantile(0.25)
<ipython-input-41-4cd70db7bbb2>:2: FutureWarning: The default value of nume
  Q3 = df.quantile(0.75)
```

This Calculates Interquartile Range (IQR) for each column in 25 % and 75 % and then it will substract to get the 50% of Interquartile Range (IQR)

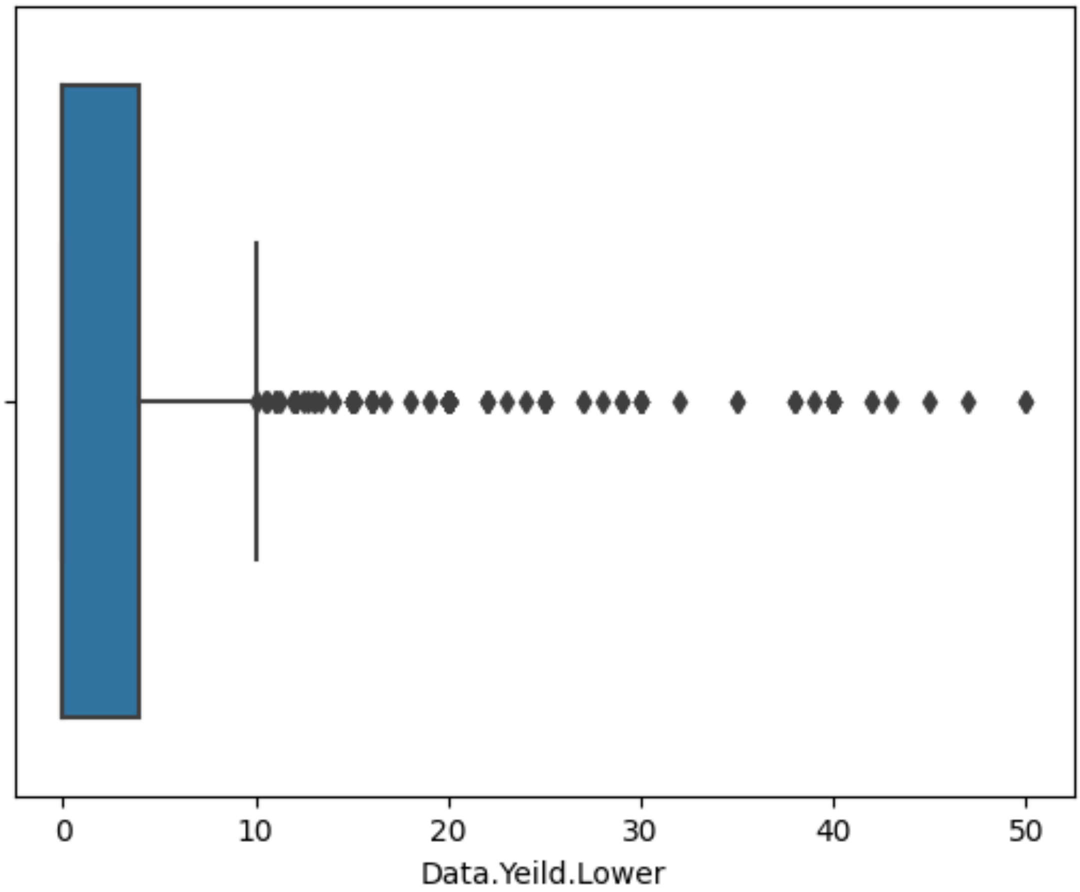
```
df = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
df.shape
```

```
<ipython-input-42-f4e1682787c4>:1: FutureWarning: Automatic reindexing on D
  df = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
(1420, 13)
```

(df < (Q1 - 1.5 * IQR)): This part of the expression checks if any value in the DataFrame is less than the lower bound (Q1 - 1.5 * IQR). (df > (Q3 + 1.5 * IQR)): This part of the expression checks if any value in the DataFrame is greater than the upper bound (Q3 + 1.5 * IQR). The | operator is used to combine these two conditions with an OR operation.

```
sns.boxplot(x=df['Data.Yeild.Lower'])
```

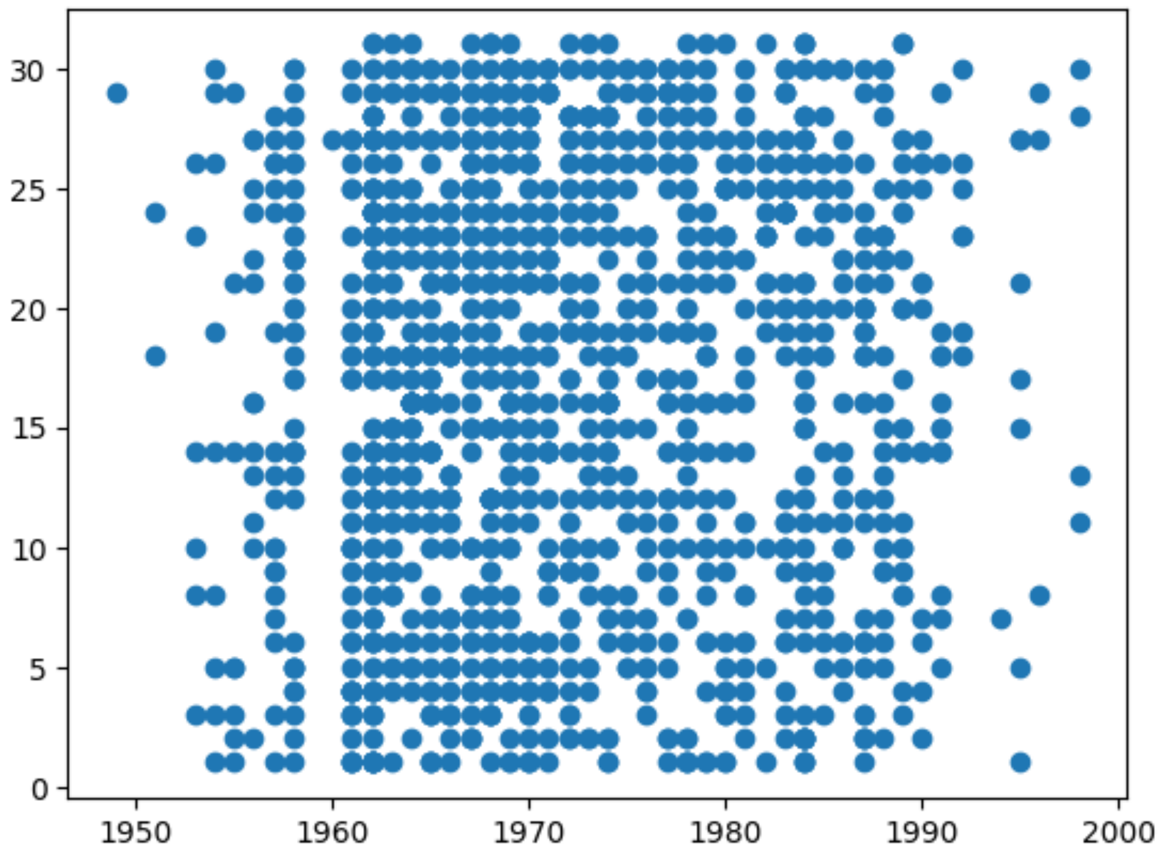
```
<Axes: xlabel='Data.Yeild.Lower'>
```



EDA (Exploratory Data Analysis) It is used to understand the main characteristics, patterns, and insights hidden in the data.

```
plt.scatter(df['Date.Year'],df['Date.Day'])
```

```
<matplotlib.collections.PathCollection at 0x781f5442bdf0>
```



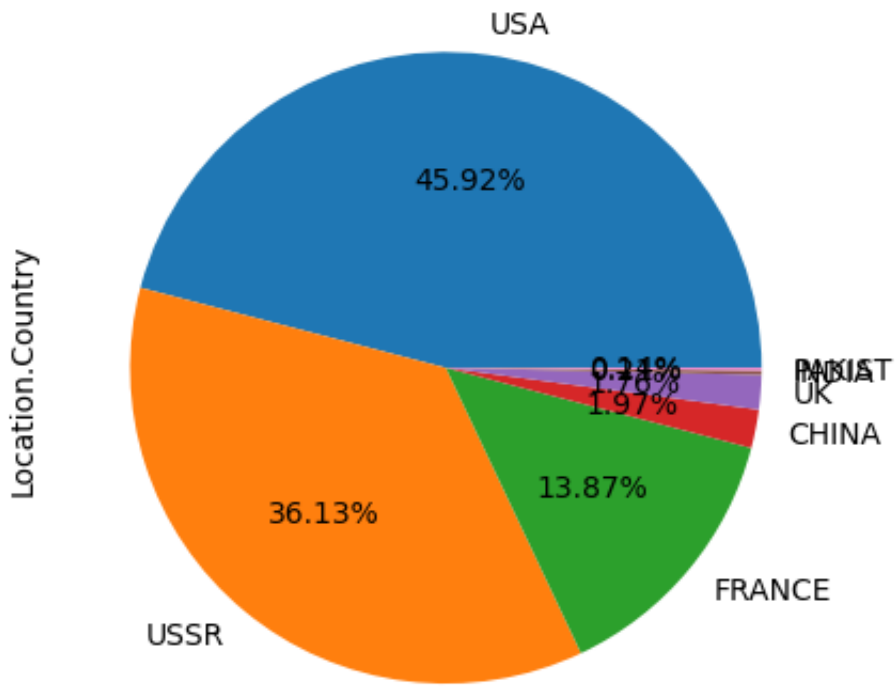
Scatter PlotT It is Bivarent analysis because the 2 variables that is x and y are used. It represents data points as individual dots on a two-dimensional plane, with one variable on the x-axis and the other variable on the y-axis. Each dot represents an observation or data point. In this also year is the x axis whereas the data_val is the y axis

```
pie=df['Location.Country'].value_counts()
pie
```

```
USA       652
USSR      513
FRANCE    197
CHINA      28
UK         25
INDIA       3
PAKIST      2
Name: Location.Country, dtype: int64
```

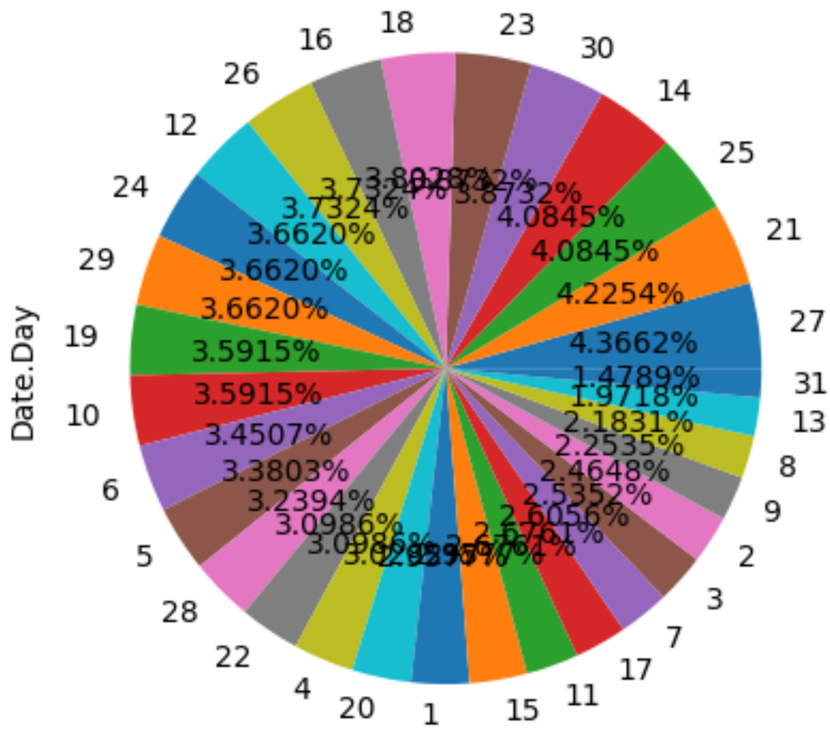
```
pie.plot(kind="pie",autopct="%.2f%%")
```

```
<Axes: ylabel='Location.Country'>
```




```
df["Date.Day"].value_counts().plot(kind="pie",autopct="%.4f%%")
```

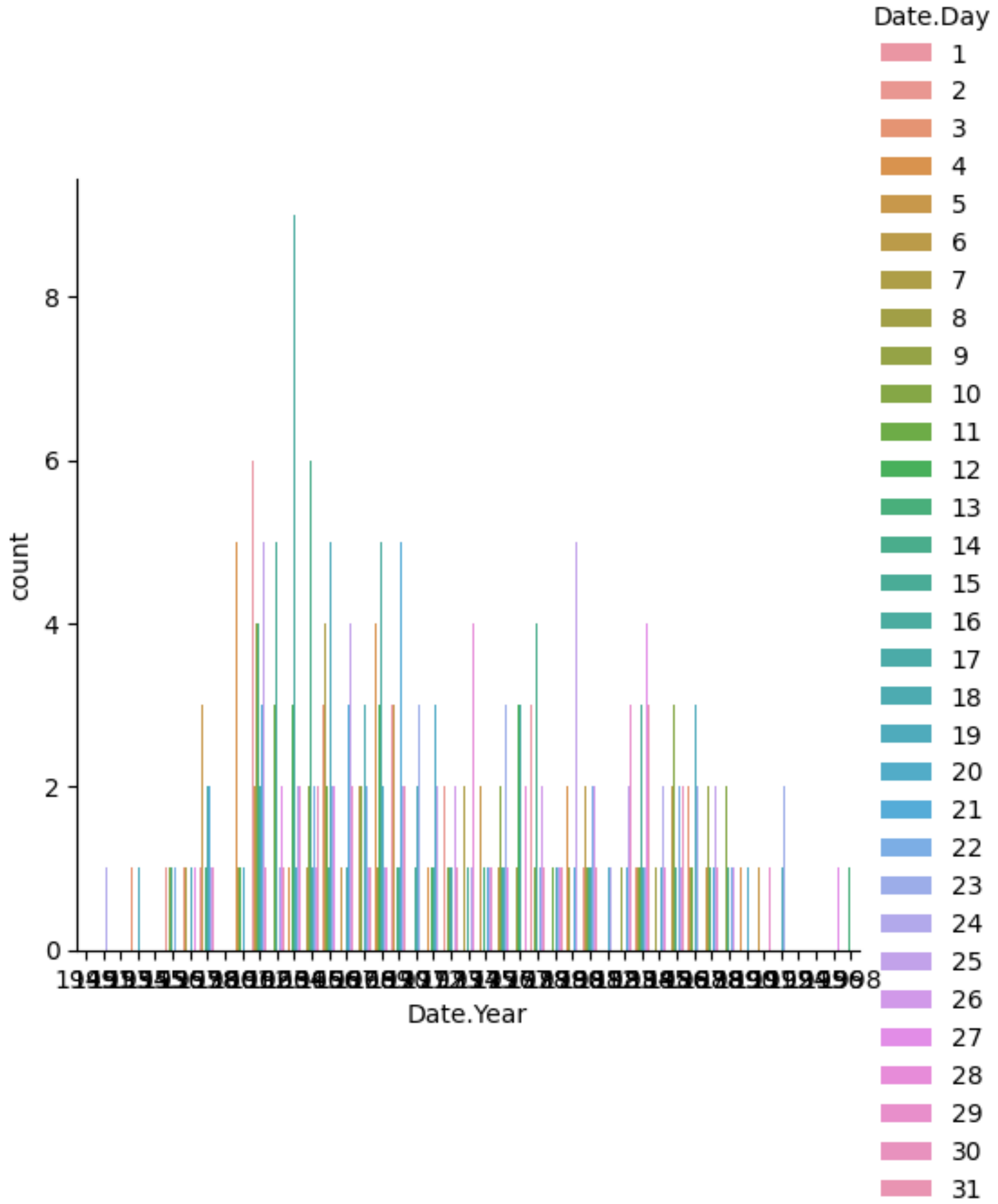
```
<Axes: ylabel='Date.Day'>
```



Pie Chart It is univarient analysis , as single variable is used. In this region and gas can show the region in the pie format. It is called a "pie chart" because the chart resembles a pie that is divided into slices, with each slice representing a particular category or data point. The size of each slice corresponds to the proportion or percentage of the whole that each category represents.

```
sns.catplot(x ='Date.Year', hue ='Date.Day',kind ='count', data = df)
```

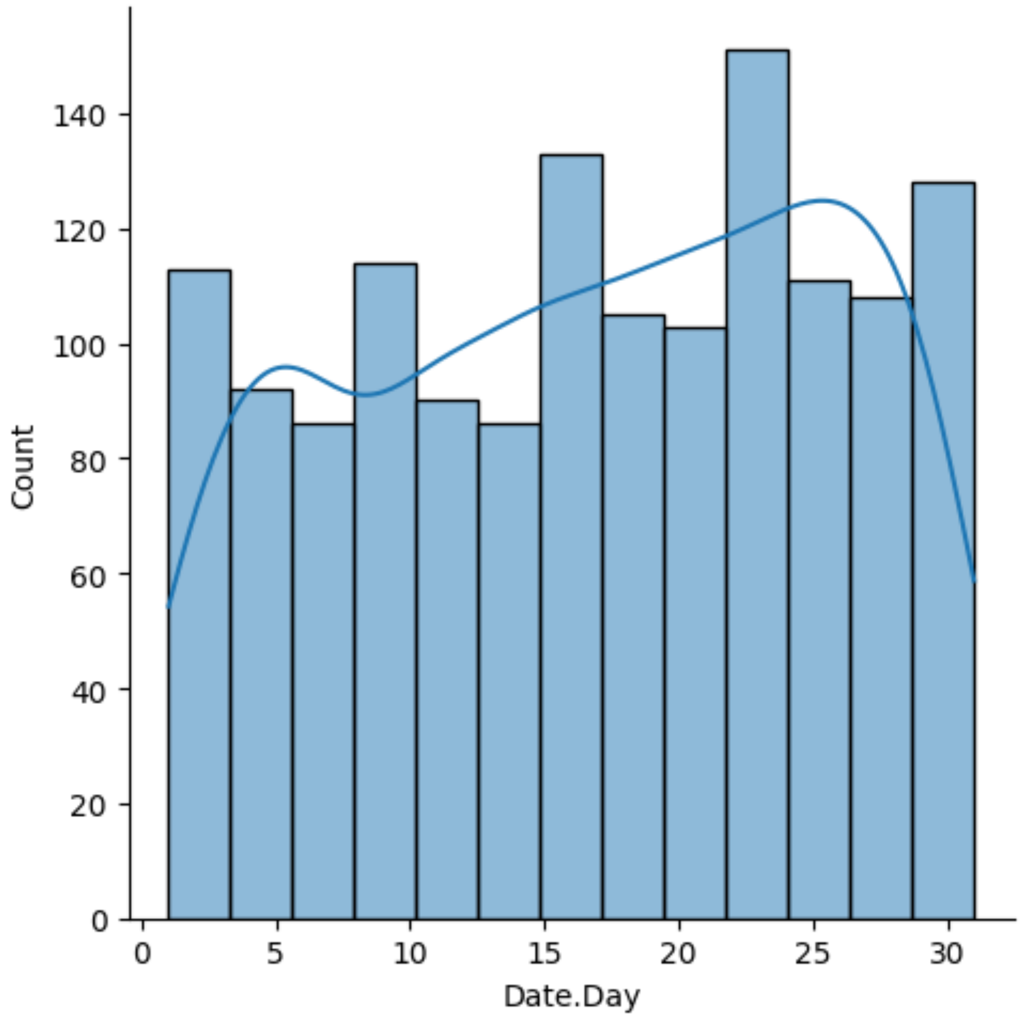
```
<seaborn.axisgrid.FacetGrid at 0x781f5436a5f0>
```



Count Plot In the Countplots height of each bar represents the number of occurrences of each category in the dataset. Countplots are particularly useful for visualizing the frequency of different categories and identifying the most common or least common categories in the data. In this the number of the gases released in each year.

```
sns.displot(df['Date.Day'],kde=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x781f51d12020>
```

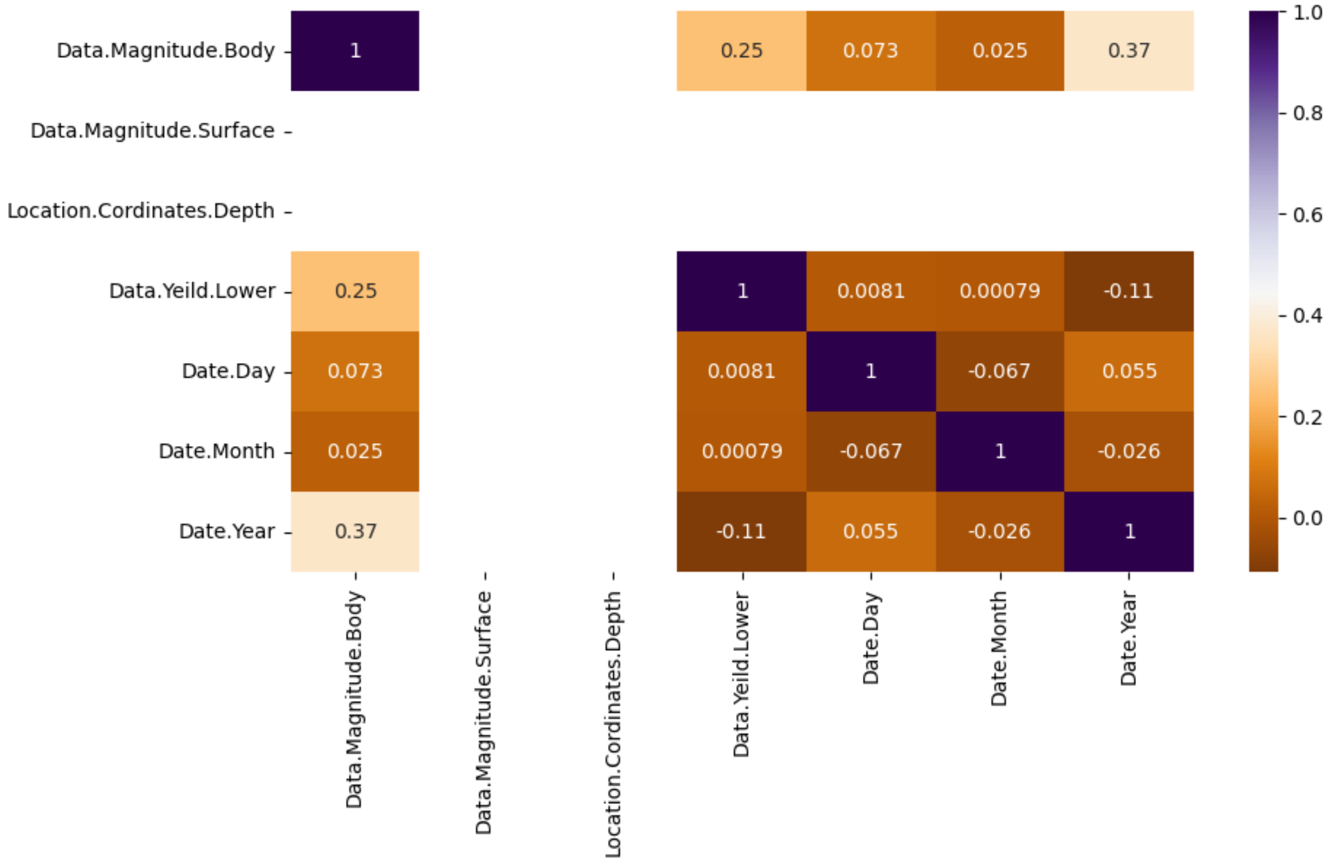


DisPlot displot is used to create a histogram to visualize the distribution of a numerical variable. It is used to create a KDE plot to visualize the estimated probability density function of a numerical variable. KDE plots show the smoothed continuous representation of the data distribution.

```
plt.figure(figsize=(10,5))
c= df.corr()
sns.heatmap(c,cmap="PuOr",annot=True)
c
```

```
<ipython-input-54-38bf0624e2ab>:2: FutureWarning: The default value of nume
  c= df.corr()
```

	Data.Magnitude.Body	Data.Magnitude.Surface	Locatio
Data.Magnitude.Body	1.000000	NaN	
Data.Magnitude.Surface	NaN	NaN	
Location.Cordinates.Depth	NaN	NaN	
Data.Yeild.Lower	0.246964	NaN	
Date.Day	0.073483	NaN	
Date.Month	0.024775	NaN	
Date.Year	0.367198	NaN	



In machine learning, splitting the data into X and Y components serves the purpose of preparing the data for model training and evaluation. The X and Y components represent the features (input) and the corresponding target variable (output) for the learning process. This separation is crucial for several reasons: Training the Model: In supervised machine learning, we have a dataset with input-output pairs. The X-axis contains the input features (often denoted as X or features matrix), and the Y-axis contains the corresponding target values (often denoted as Y or target vector). During model training, the algorithm learns from the patterns in the input-output relationships to make predictions. Feature Extraction: The X-axis includes the features or attributes that are used to describe the data instances. Each row in the X-axis represents an individual data point, and each column corresponds to a specific feature. Feature extraction and selection are essential steps in machine learning, and separating features from the target variable allows you to apply different preprocessing steps to them independently. Model Evaluation: After training a machine learning model, it needs to be evaluated to assess its performance and generalization ability on unseen data. For evaluation, the model is provided with new input features (X) to make predictions, and the predictions are compared with the corresponding target values (Y) to calculate performance metrics like accuracy, mean squared error, etc. Avoiding Data Leakage: Data leakage is a situation where information from the target variable is inadvertently included in the features during model training. This can lead to overfitting, where the model performs well on the training data but poorly on new, unseen data. By keeping the target variable (Y) separate from the features (X) during training, we prevent data leakage and ensure the model's ability to generalize to new data. Cross-Validation: When performing cross-validation, where the dataset is split into multiple subsets for training and testing, keeping X and Y separate is essential. This helps ensure that during each fold of cross-validation, the model doesn't get access to the target variable of the test set, preventing any data leakage.

```
x=df.iloc[:,0:5]
x
```

	Location.Country	Location.Region	Data.Source	Data.Magnitude.Body	D
8	USSR	Semi Kazakh	DOE	0.0	
18	USSR	Semi Kazakh	DOE	0.0	
19	USSR	Semi Kazakh	DOE	0.0	
50	USSR	Semi Kazakh	DOE	0.0	
51	USSR	Semi Kazakh	MTM	0.0	
...	...	...	...	...	
2041	CHINA	Lop Nor	HFS	5.3	
2042	INDIA	Pokhran	HFS	5.3	
2043	INDIA	Pokhran	NRD	0.0	
2044	PAKIST	Chagai	HFS	0.0	
2045	PAKIST	Kharan	HFS	5.0	

1420 rows × 5 columns

In x axis we r including all the columns except the targeted row which will be the y axis

```
y=df.iloc[:,5]
y
```

```
8       0.0
18      0.0
19      0.0
50      0.0
51      0.0
       ...
2041    0.0
2042    0.0
2043    0.0
2044    0.0
2045    0.0
Name: Location.Cordinates.Depth, Length: 1420, dtype: float64
```

In y axis we r including the targeted row only i.e data_val row Which will going to calculate and give us the value of the data i.e how much greenhouse gass emmision would be produced by each region in the year