```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

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

```
dataset = pd.read_csv('/content/drive/MyDrive/Asteroid_Updated.csv')
```

```
<ipython-input-4-b63ae7b9b3e7>:1: DtypeWarning: Columns (0,10,15,16,23,24) hav
  dataset = pd.read_csv('/content/drive/MyDrive/Asteroid_Updated.csv')
```

```
print(dataset.head())
print(dataset.info())
dataset.isnull().sum()
```

```
condition_code       867
n_obs_used             0
H                   2689
neo                    6
pha                16442
diameter          702078
extent            839696
albedo            703305
rot_per           820918
GM                839700
BV                838693
UB                838735
IR                839713
spec_B            838048
spec_T            838734
G                 839595
moid               16442
class                  0
n                      2
per                    6
ma                     8
dtype: int64
```

```
dataset = dataset.drop(["extent","rot_per","GM","BV","UB","IR","spec_B","spec_T","(
```

```
dataset = dataset.drop(["name"],axis=1)
print(dataset)
dataset.isnull().sum()
```

```
4       2.574249  0.191095   5.366988  141.576605  358.687607  2.082324
...          ...       ...        ...         ...         ...       ...
839709  2.812945  0.664688   4.695700  183.310012  234.618352  0.943214
839710  2.645238  0.259376  12.574937    1.620020  339.568072  1.959126
839711  2.373137  0.202053   0.732484  176.499082  198.026527  1.893638
839712  2.260404  0.258348   9.661947  204.512448  148.496988  1.676433
839713  2.546442  0.287672   5.356238   70.709555  273.483265  1.813901

              ad     per_y  data_arc condition_code  ...       H  neo pha  \
0       2.979647  4.608202    8822.0              0  ...   3.340    N   N
1       3.411067  4.616444   72318.0              0  ...   4.130    N   N
2       3.354967  4.360814   72684.0              0  ...   5.330    N   N
3       2.570926  3.628837   24288.0              0  ...   3.200    N   N
4       3.066174  4.130323   63507.0              0  ...   6.850    N   N
...          ...       ...       ...            ...  ...     ...  ...  ..
839709  4.682676  4.717914   17298.0              0  ...  20.400    Y   Y
839710  3.331350  4.302346      16.0              9  ...  17.507    N   N
839711  2.852636  3.655884       5.0              9  ...  18.071    N   N
839712  2.844376  3.398501      10.0              9  ...  18.060    N   N
839713  3.278983  4.063580      11.0              9  ...  17.406    N   N

       diameter  albedo      moid  class         n          per         ma
0         939.4  0.0900  1.594780    MBA  0.213885  1683.145708  77.372096
```

```
839709      NaN     NaN  0.032397    APO  0.208911  1723.217927  156.905910
839710      NaN     NaN  0.956145    MBA  0.229090  1571.431965   13.366251
839711      NaN     NaN  0.893896    MBA  0.269600  1335.311579  355.351127
839712      NaN     NaN  0.680220    MBA  0.290018  1241.302609   15.320134
839713      NaN     NaN  0.815280    MBA  0.242551  1484.222588   20.432959

[839714 rows x 21 columns]
a                      2
e                      0
i                      0
om                     0
w                      0
q                      0
ad                     6
per_y                  1
data_arc           15474
condition_code       867
n_obs_used             0
H                   2689
neo                    6
pha                16442
diameter          702078
albedo            703305
moid               16442
class                  0
n                      2
per                    6
ma                     8
```

```
dataset["a"].fillna(dataset["a"].mean(),inplace=True)
dataset["ad"].fillna(dataset["ad"].mean(),inplace=True)
dataset["per_y"].fillna(dataset["per_y"].mean(),inplace=True)
dataset["n"].fillna(dataset["n"].mean(),inplace=True)
dataset["per"].fillna(dataset["per"].mean(),inplace=True)
dataset["ma"].fillna(dataset["ma"].mean(),inplace=True)
dataset.isnull().sum()
```

```
a                      0
e                      0
i                      0
om                     0
w                      0
q                      0
ad                     0
per_y                  0
data_arc           15474
condition_code       867
n_obs_used             0
H                   2689
neo                    6
pha                16442
diameter          702078
albedo            703305
moid               16442
class                  0
n                      0
per                    0
```

```
ma                      0
dtype: int64
```

```
y=dataset["neo"]
y.value_counts()
```

```
N    818308
Y     21400
Name: neo, dtype: int64
```

```
dataset['neo'] = dataset['neo'].map( {'N': -1, 'Y':1} )
```

```
y=dataset["neo"]
y.value_counts()
```

```
-1.0    818308
 1.0     21400
Name: neo, dtype: int64
```

```
y=dataset["pha"]
y.value_counts()
```

```
N    821257
Y      2015
Name: pha, dtype: int64
```

```
dataset['pha'] = dataset['pha'].map( {'N': -1, 'Y':1} )
```

```
y=dataset["pha"]
y.value_counts()
```

```
-1.0    821257
 1.0      2015
Name: pha, dtype: int64
```

```
y=dataset["condition_code"]
y.value_counts()
```

```
0      540392
0       95711
9       23942
1       22193
5       19766
6       17103
7       15556
8       15474
4       15173
2       14541
1       10568
3        9430
9.0      7224
6.0      5804
2        5563
```

```
5        5336
7.0      4946
8.0      4347
3        3133
4        2490
E         154
D           1
Name: condition_code, dtype: int64
```

```
dataset=dataset[dataset['condition_code']!='E']
```

```
y=dataset["condition_code"]
y.value_counts()
```

```
0      540392
0       95711
9       23942
1       22193
5       19766
6       17103
7       15556
8       15474
4       15173
2       14541
1       10568
3        9430
9.0      7224
6.0      5804
2        5563
5        5336
7.0      4946
8.0      4347
3        3133
4        2490
D           1
Name: condition_code, dtype: int64
```

```
dataset=dataset[dataset['condition_code']!='D']
dataset
```

	a	e	i	om	w	q	ad	pe
0	2.769165	0.076009	10.594067	80.305532	73.597694	2.558684	2.979647	4.608
1	2.772466	0.230337	34.836234	173.080063	310.048857	2.133865	3.411067	4.616
2	2.669150	0.256942	12.988919	169.852760	248.138626	1.983332	3.354967	4.360
3	2.361418	0.088721	7.141771	103.810804	150.728541	2.151909	2.570926	3.628
4	2.574249	0.191095	5.366988	141.576605	358.687607	2.082324	3.066174	4.130

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 839559 entries, 0 to 839713
Data columns (total 21 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   a               839559 non-null  float64
 1   e               839559 non-null  float64
 2   i               839559 non-null  float64
 3   om              839559 non-null  float64
 4   w               839559 non-null  float64
 5   q               839559 non-null  float64
 6   ad              839559 non-null  float64
 7   per_y           839559 non-null  float64
 8   data_arc        824085 non-null  float64
 9   condition_code  838692 non-null  object 
 10  n_obs_used      839559 non-null  int64  
 11  H               836870 non-null  float64
 12  neo             839553 non-null  float64
 13  pha             823272 non-null  float64
 14  diameter        137636 non-null  object 
 15  albedo          136409 non-null  float64
 16  moid            823272 non-null  float64
 17  class           839559 non-null  object 
 18  n               839559 non-null  float64
 19  per             839559 non-null  float64
 20  ma              839559 non-null  float64
dtypes: float64(17), int64(1), object(3)
memory usage: 140.9+ MB
```

```
y=dataset["condition_code"]
y.value_counts()
```

```
0      540392
0       95711
9       23942
1       22193
5       19766
6       17103
7       15556
8       15474
4       15173
2       14541
1       10568
3        9430
9.0      7224
```

```
6.0       5804
2         5563
5         5336
7.0       4946
8.0       4347
3         3133
4         2490
Name: condition_code, dtype: int64
```

```
dataset.corr()
```

```
<ipython-input-21-c187c74d1e71>:1: FutureWarning: The default value of numeri
  dataset.corr()
```

	a	e	i	om	w	q	ad
a	1.000000	-0.007132	-0.018973	0.000155	-0.001283	0.025003	0.058672
e	-0.007132	1.000000	0.132792	0.005204	0.007978	-0.115409	0.086631
i	-0.018973	0.132792	1.000000	-0.015800	-0.000701	0.032010	0.071985
om	0.000155	0.005204	-0.015800	1.000000	-0.132300	-0.010910	-0.002044
w	-0.001283	0.007978	-0.000701	-0.132300	1.000000	-0.003412	-0.000389
q	0.025003	-0.115409	0.032010	-0.010910	-0.003412	1.000000	0.306933
ad	0.058672	0.086631	0.071985	-0.002044	-0.000389	0.306933	1.000000
per_y	0.052980	0.043546	0.040328	-0.000796	-0.000667	0.109765	0.931741
data_arc	0.000106	-0.149614	-0.140710	0.001744	-0.005363	-0.028362	-0.021425
n_obs_used	-0.000409	-0.079494	-0.071632	-0.005431	0.003743	-0.029189	-0.014504
H	-0.008793	0.342410	-0.098055	0.002186	-0.004289	-0.436410	-0.124646
neo	-0.026189	0.498814	0.088954	0.007088	0.001343	-0.107873	-0.009033
pha	-0.000415	0.197500	0.036250	0.002641	0.000020	-0.036172	-0.002201
albedo	-0.110227	-0.019379	-0.089775	0.000736	-0.003063	-0.262726	-0.069225
moid	0.024907	-0.105112	0.040909	-0.011062	-0.003382	0.999742	0.307557
n	-0.005133	0.201315	0.000080	0.008957	0.003678	-0.327805	-0.098147
per	0.052948	0.043548	0.040331	-0.000796	-0.000667	0.109765	0.931741
ma	0.001656	-0.015434	0.007044	0.000608	-0.007585	-0.004538	-0.006125

```
dataset['class'] = dataset['class'].map( {'IEO':0, 'AST':1, 'ATE':2, 'APO':3, 'AMO'

y=dataset["class"]
y.value_counts()
```

```
6     747292
8      24712
5      17547
7      17341
3      11759
4       8020
11      7383
13      3308
2       1601
12       486
1         84
0         20
10         4
9          2
Name: class, dtype: int64
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 839559 entries, 0 to 839713
Data columns (total 21 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   a               839559 non-null  float64
 1   e               839559 non-null  float64
 2   i               839559 non-null  float64
 3   om              839559 non-null  float64
 4   w               839559 non-null  float64
 5   q               839559 non-null  float64
 6   ad              839559 non-null  float64
 7   per_y           839559 non-null  float64
 8   data_arc        824085 non-null  float64
 9   condition_code  838692 non-null  object 
 10  n_obs_used      839559 non-null  int64  
 11  H               836870 non-null  float64
 12  neo             839553 non-null  float64
 13  pha             823272 non-null  float64
 14  diameter        137636 non-null  object 
 15  albedo          136409 non-null  float64
 16  moid            823272 non-null  float64
 17  class           839559 non-null  int64  
 18  n               839559 non-null  float64
 19  per             839559 non-null  float64
 20  ma              839559 non-null  float64
dtypes: float64(17), int64(2), object(2)
memory usage: 140.9+ MB
```

```
dataset = dataset.drop(["condition_code"],axis=1)
```

```
dataset
```

	a	e	i	om	w	q	ad	pe:
0	2.769165	0.076009	10.594067	80.305532	73.597694	2.558684	2.979647	4.608
1	2.772466	0.230337	34.836234	173.080063	310.048857	2.133865	3.411067	4.616
2	2.669150	0.256942	12.988919	169.852760	248.138626	1.983332	3.354967	4.360
3	2.361418	0.088721	7.141771	103.810804	150.728541	2.151909	2.570926	3.628
4	2.574249	0.191095	5.366988	141.576605	358.687607	2.082324	3.066174	4.130
...	...	...	...	...	...	...	...	
839709	2.812945	0.664688	4.695700	183.310012	234.618352	0.943214	4.682676	4.717
839710	2.645238	0.259376	12.574937	1.620020	339.568072	1.959126	3.331350	4.302
839711	2.373137	0.202053	0.732484	176.499082	198.026527	1.893638	2.852636	3.655
839712	2.260404	0.258348	9.661947	204.512448	148.496988	1.676433	2.844376	3.398
839713	2.546442	0.287672	5.356238	70.709555	273.483265	1.813901	3.278983	4.063

```
dataset.corr()
```

```
<ipython-input-25-c187c74d1e71>:1: FutureWarning: The default value of numeri
  dataset.corr()
```

	a	e	i	om	w	q	ad
a	1.000000	-0.007132	-0.018973	0.000155	-0.001283	0.025003	0.058672
e	-0.007132	1.000000	0.132792	0.005204	0.007978	-0.115409	0.086631

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 839559 entries, 0 to 839713
Data columns (total 20 columns):
 #   Column      Non-Null Count   Dtype  
---  ------      --------------   -----  
 0   a           839559 non-null  float64
 1   e           839559 non-null  float64
 2   i           839559 non-null  float64
 3   om          839559 non-null  float64
 4   w           839559 non-null  float64
 5   q           839559 non-null  float64
 6   ad          839559 non-null  float64
 7   per_y       839559 non-null  float64
 8   data_arc    824085 non-null  float64
 9   n_obs_used  839559 non-null  int64  
 10  H           836870 non-null  float64
 11  neo         839553 non-null  float64
 12  pha         823272 non-null  float64
 13  diameter    137636 non-null  object 
 14  albedo      136409 non-null  float64
 15  moid        823272 non-null  float64
 16  class       839559 non-null  int64  
 17  n           839559 non-null  float64
 18  per         839559 non-null  float64
 19  ma          839559 non-null  float64
dtypes: float64(17), int64(2), object(1)
memory usage: 134.5+ MB
```

ma	0.001656	-0.015434	0.007044	0.000608	-0.007585	-0.004538	-0.006125

```
dataset.isnull().sum()
```

```
a                 0
e                 0
i                 0
om                0
w                 0
q                 0
ad                0
per_y             0
data_arc      15474
n_obs_used        0
H              2689
neo               6
pha           16287
diameter     701923
albedo       703150
moid          16287
class             0
n                 0
per               0
```

```
ma               0
dtype: int64
```

```
dataset["data_arc"].fillna(dataset.groupby("n_obs_used")["data_arc"].transform("mea
```

```
dataset["moid"].fillna(dataset.groupby("q")["moid"].transform("mean"),inplace=True)
```

```
dataset["neo"].fillna(dataset["neo"].mean(),inplace=True)
```

```
dataset["H"].fillna(dataset.groupby("n")["H"].transform("mean"),inplace=True)
```

```
dataset.isnull().sum()
```

```
a                0
e                0
i                0
om               0
w                0
q                0
ad               0
per_y            0
data_arc         0
n_obs_used       0
H             2689
neo              0
pha          16287
diameter    701923
albedo      703150
moid         16287
class            0
n                0
per              0
ma               0
dtype: int64
```

```
dataset["pha"].fillna(dataset["pha"].mode(),inplace=True)
```

```
dataset["moid"].fillna(dataset["moid"].mean(),inplace=True)
```

```
dataset["H"].fillna(dataset["H"].mean(),inplace=True)
```

```
dataset.isnull().sum()
```

```
a                0
e                0
i                0
om               0
w                0
q                0
ad               0
per_y            0
data_arc         0
n_obs_used       0
H                0
neo              0
pha          16287
```

```
diameter      701923
albedo        703150
moid               0
class              0
n                  0
per                0
ma                 0
dtype: int64
```

```
y=dataset["pha"]
y.value_counts()
```

```
-1.0    821257
 1.0      2015
Name: pha, dtype: int64
```

```
dataset["pha"]=dataset["pha"].fillna(-1)
```

```
dataset.isnull().sum()
```

```
a                  0
e                  0
i                  0
om                 0
w                  0
q                  0
ad                 0
per_y              0
data_arc           0
n_obs_used         0
H                  0
neo                0
pha                0
diameter      701923
albedo        703150
moid               0
class              0
n                  0
per                0
ma                 0
dtype: int64
```

```
dataset=dataset.dropna()
```

```
dataset.isnull().sum()
```

```
a             0
e             0
i             0
om            0
w             0
q             0
ad            0
per_y         0
data_arc      0
n_obs_used    0
```

```
H            0
neo          0
pha          0
diameter     0
albedo       0
moid         0
class        0
n            0
per          0
ma           0
dtype: int64
```

```
dataset.info()
```

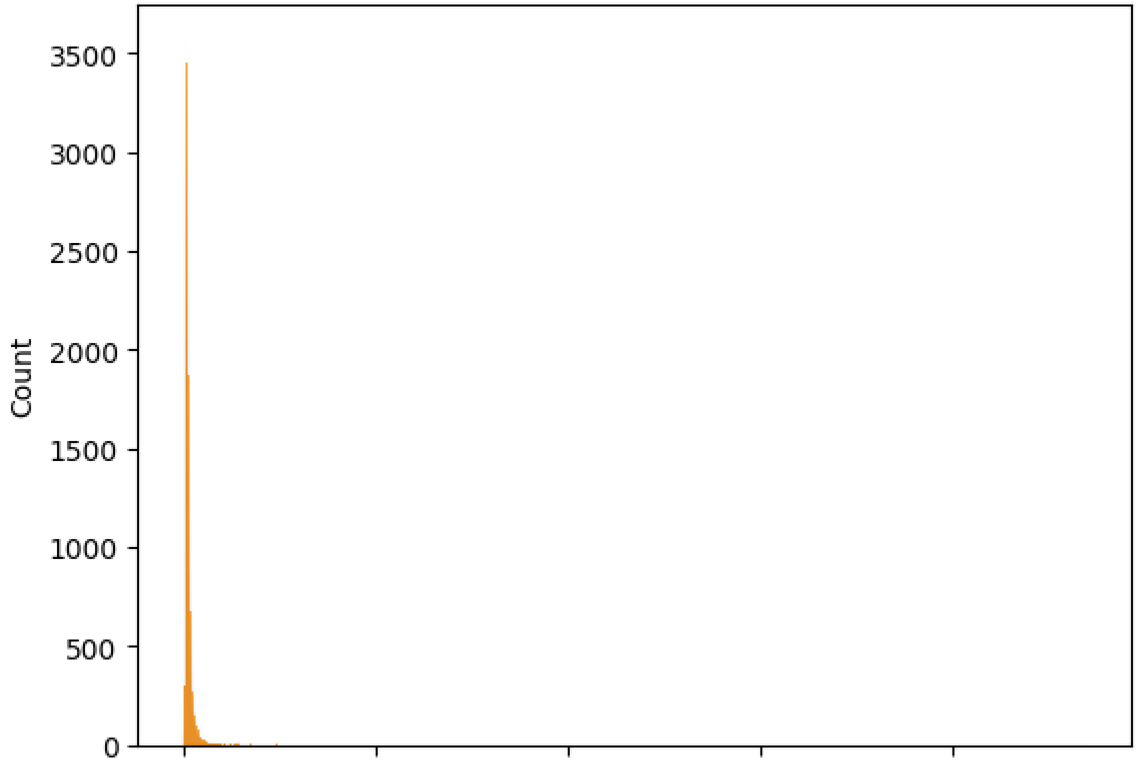
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 136406 entries, 0 to 810375
Data columns (total 20 columns):
 #   Column      Non-Null Count   Dtype  
---  ------      --------------   -----  
 0   a           136406 non-null  float64
 1   e           136406 non-null  float64
 2   i           136406 non-null  float64
 3   om          136406 non-null  float64
 4   w           136406 non-null  float64
 5   q           136406 non-null  float64
 6   ad          136406 non-null  float64
 7   per_y       136406 non-null  float64
 8   data_arc    136406 non-null  float64
 9   n_obs_used  136406 non-null  int64  
 10  H           136406 non-null  float64
 11  neo         136406 non-null  float64
 12  pha         136406 non-null  float64
 13  diameter    136406 non-null  object 
 14  albedo      136406 non-null  float64
 15  moid        136406 non-null  float64
 16  class       136406 non-null  int64  
 17  n           136406 non-null  float64
 18  per         136406 non-null  float64
 19  ma          136406 non-null  float64
dtypes: float64(17), int64(2), object(1)
memory usage: 21.9+ MB
```

```
dataset["diameter"] = dataset["diameter"].astype(float)
```

```
<ipython-input-34-462e321415db>: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
  dataset["diameter"] = dataset["diameter"].astype(float)
```

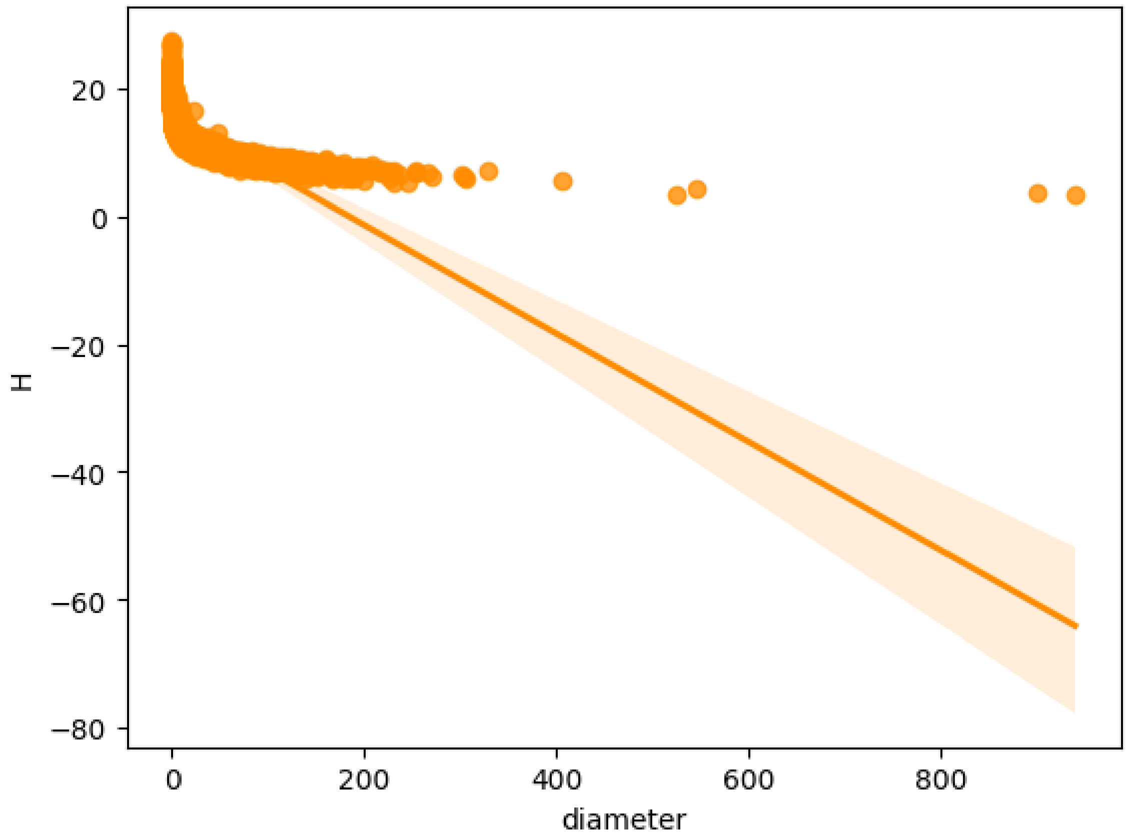
```
sns.histplot(dataset["diameter"],color='darkorange')
```

```
<Axes: xlabel='diameter', ylabel='Count'>
```



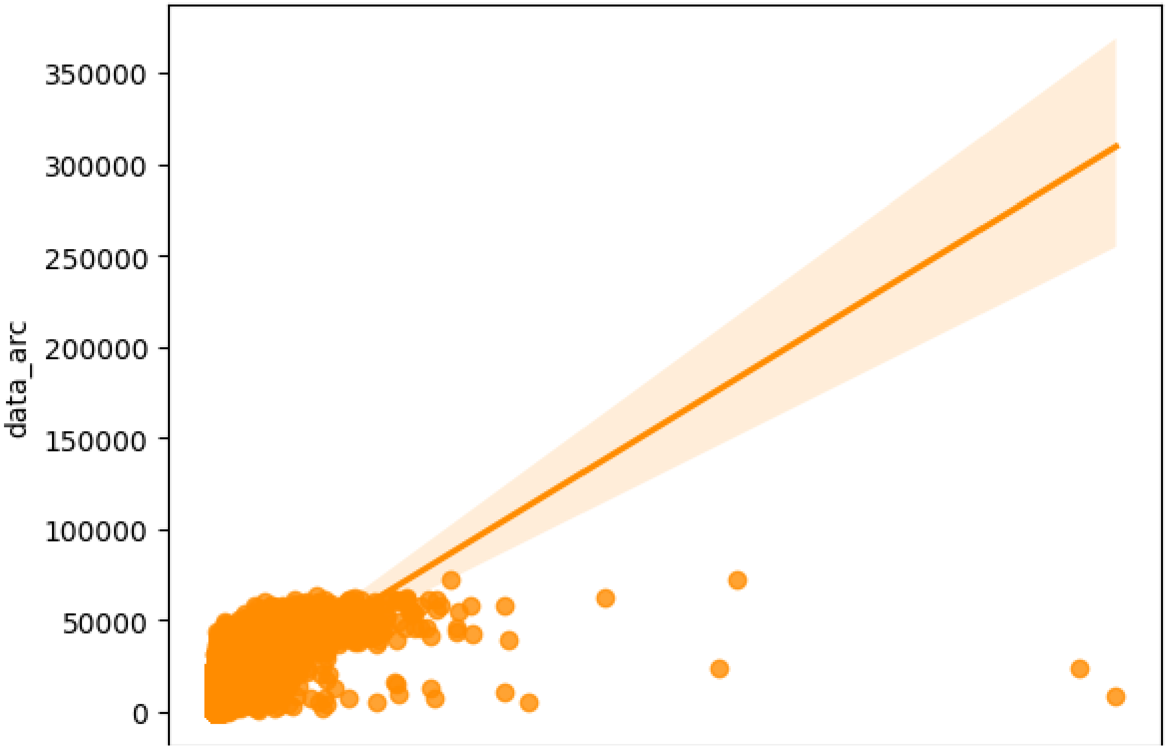

```
sns.regplot(dataset,x=dataset['diameter'],y=dataset['H'],color='darkorange')
```

```
<Axes: xlabel='diameter', ylabel='H'>
```




```
sns.regplot(dataset,x=dataset['diameter'],y=dataset['data_arc'],color='darkorange')
```

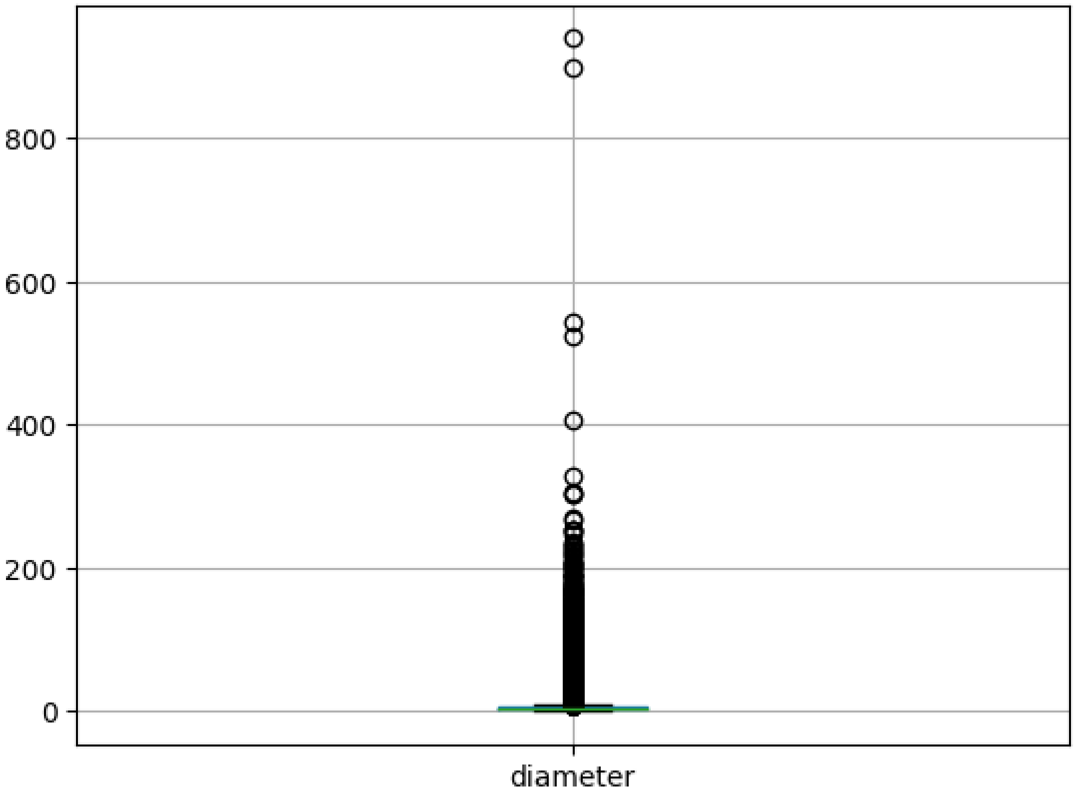
```
<Axes: xlabel='diameter', ylabel='data_arc'>
```




```
sns.distplot(dataset['diameter'].dropna())
```

```
<ipython-input-38-2506428ea76d>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.
```

```
figure=dataset.boxplot(column="diameter")
```




```
dataset['diameter'].describe()
```

```
count    136406.000000
mean          5.505299
std           9.422372
min           0.008000
25%           2.780000
50%           3.970000
75%           5.764000
max         939.400000
Name: diameter, dtype: float64
```

```
IQR=dataset.diameter.quantile(0.75)-dataset.diameter.quantile(0.25)
```

```
lower_bridge=dataset['diameter'].quantile(0.25)-(IQR*1.5)
upper_bridge=dataset['diameter'].quantile(0.75)+(IQR*1.5)
print(lower_bridge), print(upper_bridge)
```

```
-1.6960000000000001
10.240000000000002
(None, None)
```

```
lower_bridge=dataset['diameter'].quantile(0.25)-(IQR*3)
upper_bridge=dataset['diameter'].quantile(0.75)+(IQR*3)
print(lower_bridge), print(upper_bridge)
```

```
-6.172000000000002
14.716000000000001
(None, None)
```

```
upper_bridge=dataset['diameter'].quantile(0.9)+(IQR*3)
print(upper_bridge)
```

```
17.6015
```

```
upper_bridge=dataset['diameter'].quantile(0.95)
print(upper_bridge)
```

```
12.13675
```

```
upper_bridge=dataset['diameter'].quantile(0.95)+(IQR*3)
print(upper_bridge)
```

```
21.08875
```

```
upper_bridge=dataset['diameter'].quantile(0.99)+(IQR*3)
print(upper_bridge)
```

```
40.90720000000066
```

```
dataset=dataset[dataset['diameter']<=50]
```

```
dataset['diameter'].describe()
```

```
count    135645.000000
mean          4.990452
std           4.158836
min           0.008000
25%           2.774000
50%           3.956000
75%           5.720000
max          49.990000
Name: diameter, dtype: float64
```

```
sns.distplot(dataset['diameter'].dropna())
```

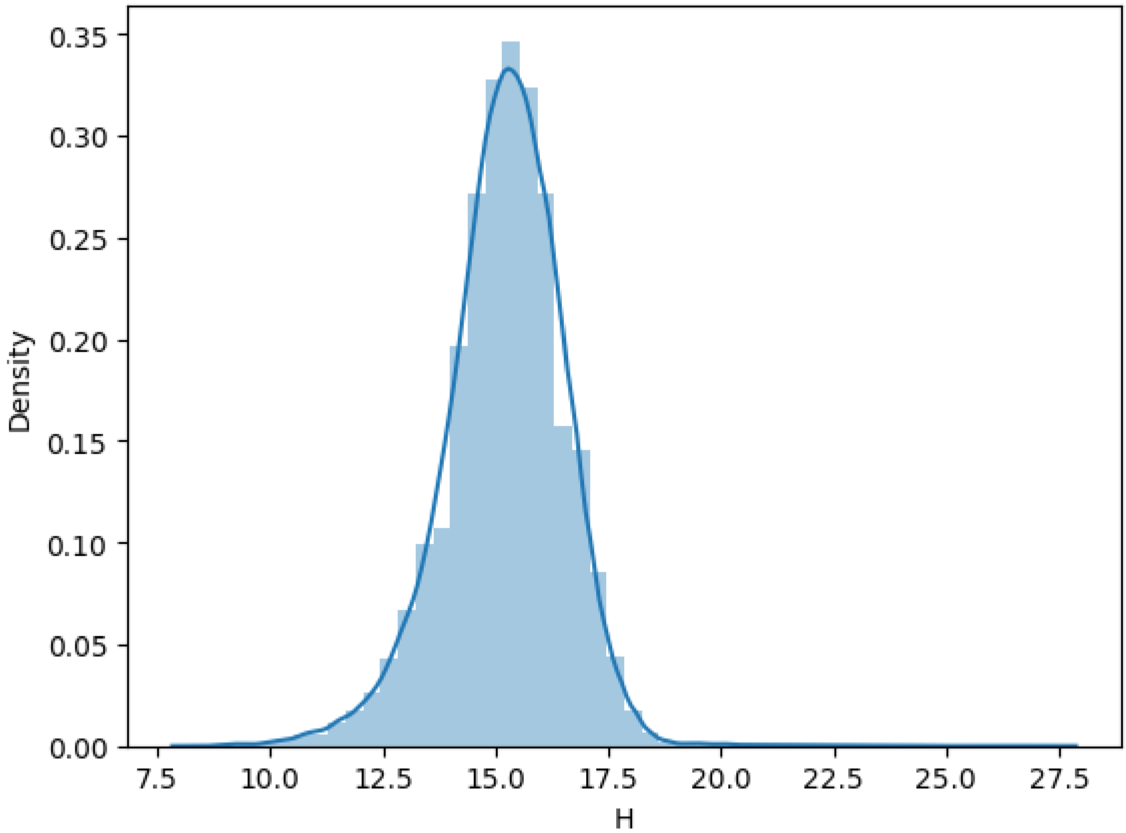
```
<ipython-input-48-5e422195445d>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['diameter'].dropna())
<Axes: xlabel='diameter', ylabel='Density'>
```


```
dataset.corr()
```

	a	e	i	om	w	q	ad
a	1.000000	0.014880	0.144898	-0.000902	-0.002813	0.320651	0.987461
e	0.014880	1.000000	0.145619	-0.000673	0.012635	-0.563526	0.109407
i	0.144898	0.145619	1.000000	-0.013451	-0.004982	0.080945	0.137335
om	-0.000902	-0.000673	-0.013451	1.000000	-0.107288	-0.004408	-0.000205
w	-0.002813	0.012635	-0.004982	-0.107288	1.000000	-0.008622	-0.001491

```
sns.distplot(dataset['H'].dropna())
```

```
<ipython-input-50-f49ea5c28868>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['H'].dropna())
<Axes: xlabel='H', ylabel='Density'>
```



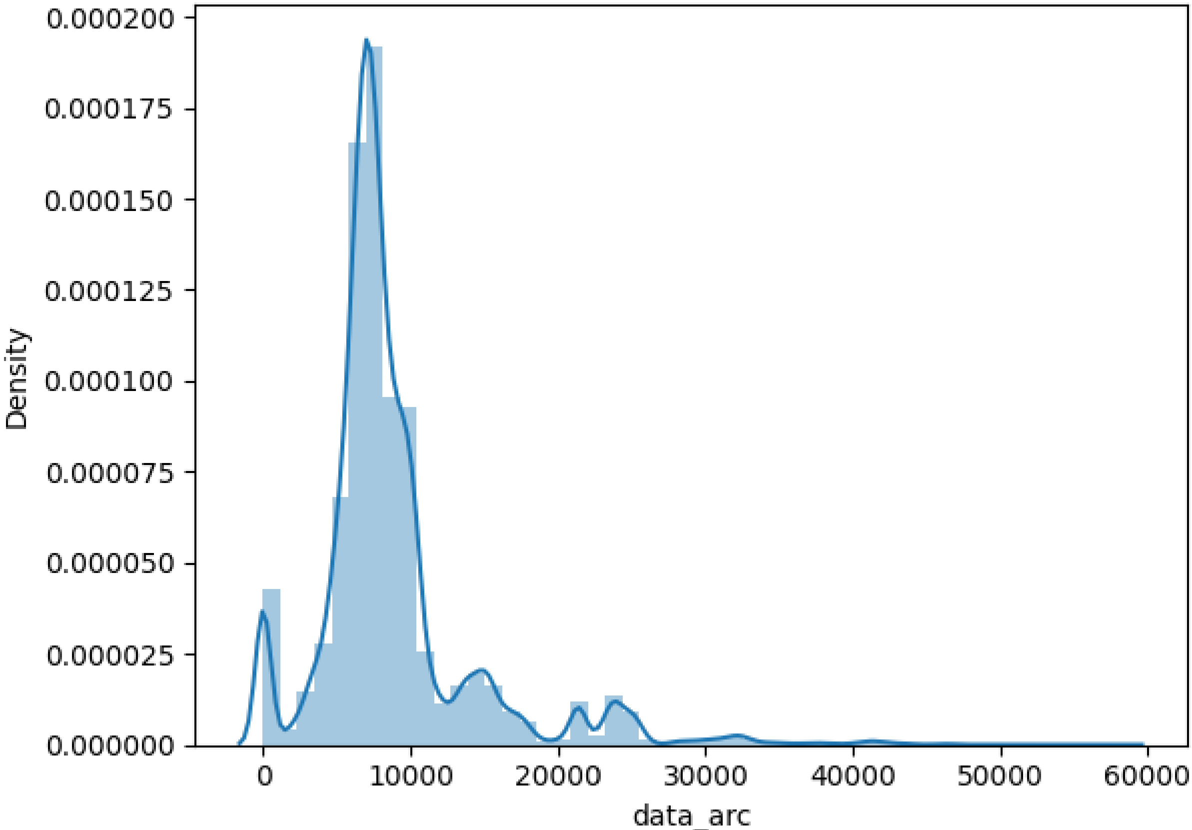

```
uppper_boundary=dataset['H'].mean() + 3* dataset['H'].std()
lower_boundary=dataset['H'].mean() - 3* dataset['H'].std()
print(lower_boundary), print(uppper_boundary),print(dataset['H'].mean())
```

```
11.23180588676272
19.183777394499074
```

```
15.207791640630896
(None, None, None)
```

```
sns.distplot(dataset['data_arc'].dropna())
```

```
<ipython-input-52-10b242f03382>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['data_arc'].dropna())
<Axes: xlabel='data_arc', ylabel='Density'>
```




```
dataset['data_arc'].describe()
```

```
count    135645.000000
mean       8837.831326
std        5646.415202
min           1.000000
25%        6302.000000
50%        7579.000000
75%        9714.000000
max       58007.000000
Name: data_arc, dtype: float64
```

```
IQR=dataset.data_arc.quantile(0.75)-dataset.data_arc.quantile(0.25)
lower_bridge=dataset['data_arc'].quantile(0.25)-(IQR*1.5)
upper_bridge=dataset['data_arc'].quantile(0.75)+(IQR*1.5)
print(lower_bridge), print(upper_bridge)
```

```
1184.0
14832.0
(None, None)
```

```
IQR=dataset.data_arc.quantile(0.75)-dataset.data_arc.quantile(0.25)
lower_bridge=dataset['data_arc'].quantile(0.25)-(IQR*3)
upper_bridge=dataset['data_arc'].quantile(0.75)+(IQR*3)
print(lower_bridge), print(upper_bridge)
```

```
-3934.0
19950.0
(None, None)
```

```
upper_bridge=dataset['data_arc'].quantile(0.95)
print(upper_bridge)
```

```
21558.0
```

```
dataset=dataset[dataset['data_arc']<=35000]
```

```
dataset['data_arc'].describe()
```

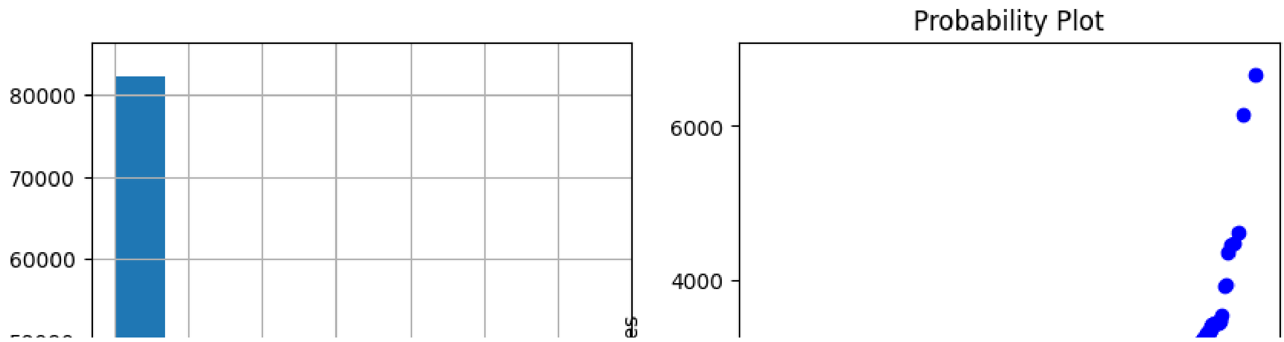
```
count    134974.000000
mean       8677.993378
std        5178.000757
min           1.000000
25%        6297.000000
50%        7567.000000
75%        9662.750000
max       34994.000000
Name: data_arc, dtype: float64
```

```
sns.distplot(dataset['data_arc'].dropna())
```

```
<ipython-input-58-10b242f03382>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['data_arc'].dropna())
<Axes: xlabel='data_arc', ylabel='Density'>
```


```
import scipy.stats as stat
import pylab

def plot_data(df,feature):
    plt.figure(figsize=(10,6))
    plt.subplot(1,2,1)
    df[feature].hist()
    plt.subplot(1,2,2)
    stat.probplot(df[feature],dist='norm',plot=pylab)
    plt.show()
```

```
plot_data(dataset,'n_obs_used')
```

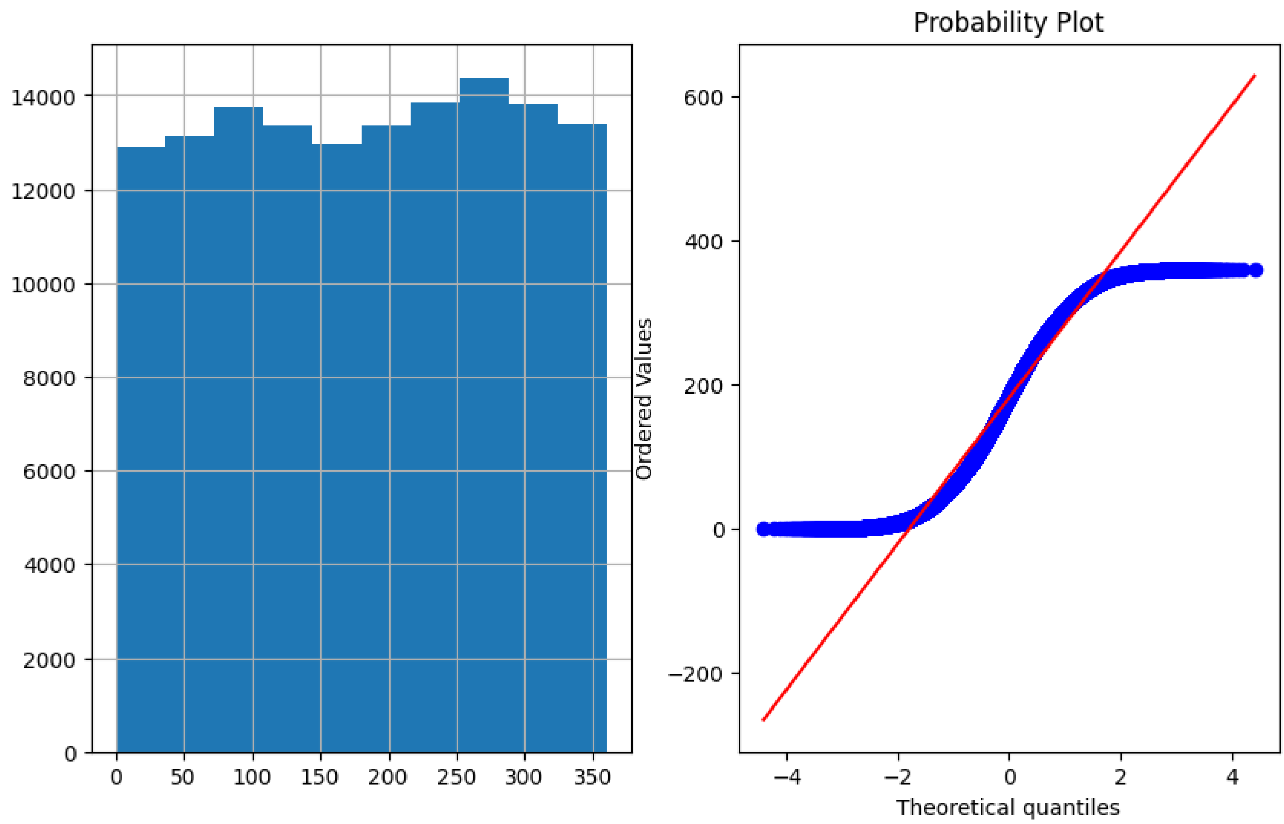



```
df=dataset
```

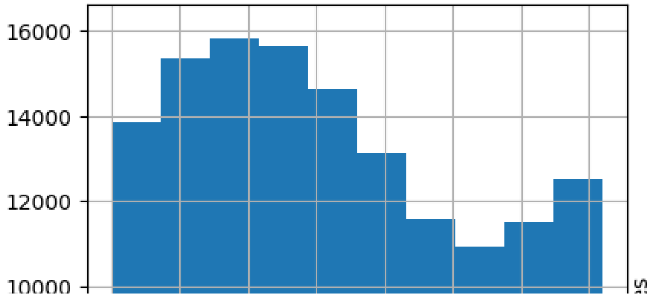

```
df.corr()
```

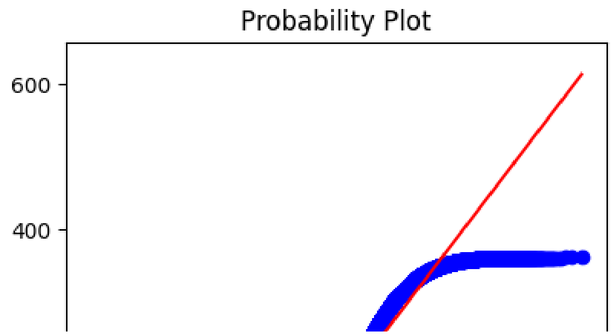
	a	e	i	om	w	q	ad	
a	1.000000	0.014988	0.144930	-0.000861	-0.002853	0.320396	0.987498	(
e	0.014988	1.000000	0.145365	-0.000756	0.012690	-0.563385	0.109349	(
i	0.144930	0.145365	1.000000	-0.013512	-0.005210	0.081016	0.137364	(
om	-0.000861	-0.000756	-0.013512	1.000000	-0.106952	-0.004323	-0.000177	(
w	-0.002853	0.012690	-0.005210	-0.106952	1.000000	-0.008762	-0.001511	-(
q	0.320396	-0.563385	0.081016	-0.004323	-0.008762	1.000000	0.167068	(
ad	0.987498	0.109349	0.137364	-0.000177	-0.001511	0.167068	1.000000	(
per_y	0.949897	0.046063	0.094526	0.000383	-0.001680	0.087718	0.974068	:
data_arc	-0.027342	-0.034244	-0.227405	-0.002600	-0.003843	-0.033963	-0.022806	-(
n_obs_used	-0.055632	-0.080954	-0.231727	-0.025505	0.012039	-0.100913	-0.041110	-(
H	-0.128115	0.216437	-0.032448	0.004731	-0.009857	-0.398973	-0.066953	-(
neo	-0.052235	0.339667	0.101925	0.003661	0.001053	-0.248185	-0.013068	-(
pha	-0.030848	0.166516	0.026561	0.000740	-0.003003	-0.131067	-0.010297	-(
diameter	0.205111	-0.117310	0.101101	-0.000697	0.002788	0.504440	0.129541	(
albedo	-0.110504	-0.020083	-0.088626	0.000505	-0.003059	-0.278586	-0.068656	-(
moid	0.323534	-0.538755	0.123202	-0.005188	-0.008660	0.996368	0.170939	(
class	0.266951	-0.171787	0.064119	-0.000651	-0.001260	0.668708	0.166570	(
n	-0.275388	0.189241	-0.110307	0.008068	0.003543	-0.744233	-0.162783	-(
per	0.949897	0.046063	0.094526	0.000383	-0.001680	0.087718	0.974068	:
ma	0.016368	-0.017401	0.014522	-0.003259	0.001364	0.078613	0.003954	-(

```
plot_data(dataset,'w')
```




```
plot_data(dataset,'om')
```

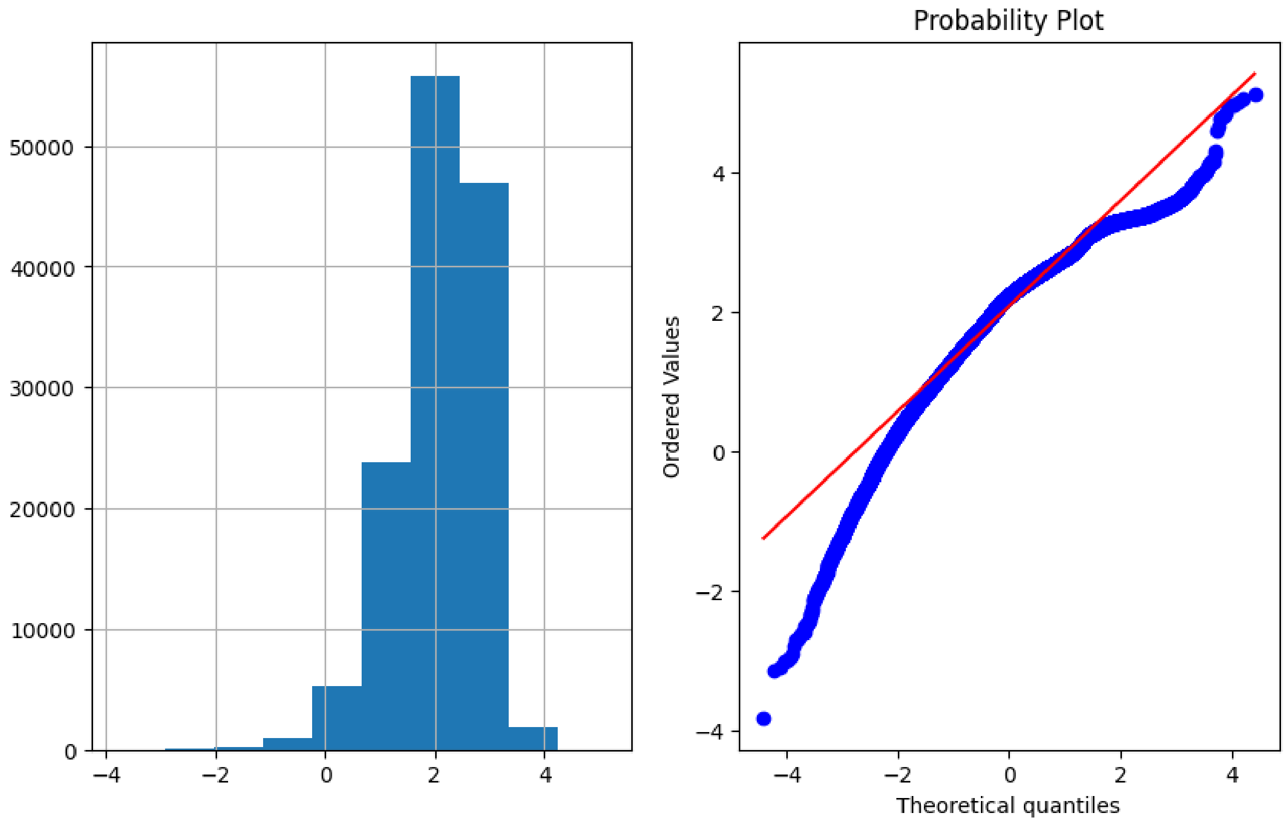





```
df['a/ad']=(df['om'])/(df['w'])
```

```
df.corr()
```

```
df['e_log']=np.log(df['i'])
plot_data(df,'e_log')
```

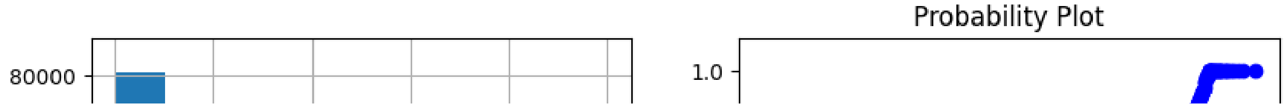


ma	0.016368	-0.017401	0.014522	-0.003259	0.001364	0.078613	0.003954	-(

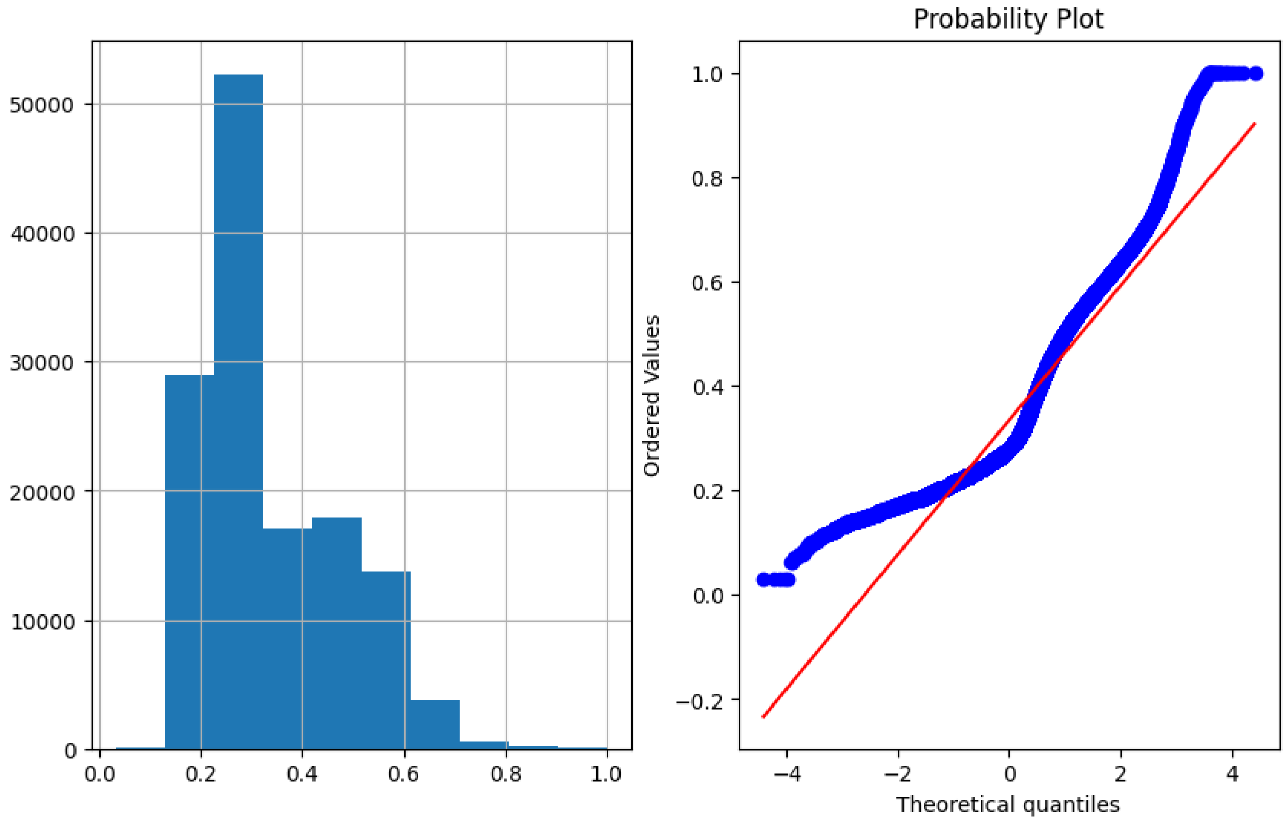
```
df.corr()
```

	a	e	i	om	w	q	ad
a	1.000000	0.014988	0.144930	-0.000861	-0.002853	0.320396	0.987498
e	0.014988	1.000000	0.145365	-0.000756	0.012690	-0.563385	0.109349
i	0.144930	0.145365	1.000000	-0.013512	-0.005210	0.081016	0.137364
om	-0.000861	-0.000756	-0.013512	1.000000	-0.106952	-0.004323	-0.000177
w	-0.002853	0.012690	-0.005210	-0.106952	1.000000	-0.008762	-0.001511
q	0.320396	-0.563385	0.081016	-0.004323	-0.008762	1.000000	0.167068
ad	0.987498	0.109349	0.137364	-0.000177	-0.001511	0.167068	1.000000
per_y	0.949897	0.046063	0.094526	0.000383	-0.001680	0.087718	0.974068
data_arc	-0.027342	-0.034244	-0.227405	-0.002600	-0.003843	-0.033963	-0.022806
n_obs_used	-0.055632	-0.080954	-0.231727	-0.025505	0.012039	-0.100913	-0.041110
H	-0.128115	0.216437	-0.032448	0.004731	-0.009857	-0.398973	-0.066953
neo	-0.052235	0.339667	0.101925	0.003661	0.001053	-0.248185	-0.013068
pha	-0.030848	0.166516	0.026561	0.000740	-0.003003	-0.131067	-0.010297
diameter	0.205111	-0.117310	0.101101	-0.000697	0.002788	0.504440	0.129541
albedo	-0.110504	-0.020083	-0.088626	0.000505	-0.003059	-0.278586	-0.068656

```
plot_data(df,'albedo')
```




```
df['albedo_sqaure']=df.albedo**(1/2)
plot_data(df,'albedo_sqaure')
```




```
df.corr()['diameter']
```

```
a              0.205111
e             -0.117310
i              0.101101
om            -0.000697
w              0.002788
q              0.504440
ad             0.129541
per_y          0.068389
data_arc       0.439798
n_obs_used     0.456118
H             -0.722641
neo           -0.075739
pha           -0.040669
diameter       1.000000
albedo        -0.215648
moid           0.508903
class          0.422067
```

```
n                -0.404766
per               0.068389
ma                0.023790
a/ad             -0.001211
e_log             0.079090
albedo_sqaure    -0.214760
Name: diameter, dtype: float64
```

```
df = df.drop('e_log',axis=1)
```

```
# df = df.drop(["a/ad"],axis=1)
# df = df.drop(["n_obs_used_log"],axis=1)
```

```
df.corr()['diameter']
```

```
a                 0.205111
e                -0.117310
i                 0.101101
om               -0.000697
w                 0.002788
q                 0.504440
ad                0.129541
per_y             0.068389
data_arc          0.439798
n_obs_used        0.456118
H                -0.722641
neo              -0.075739
pha              -0.040669
diameter          1.000000
albedo           -0.215648
moid              0.508903
class             0.422067
n                -0.404766
per               0.068389
ma                0.023790
albedo_sqaure    -0.214760
Name: diameter, dtype: float64
```

```
df=df.assign(feature3=lambda x:(x.a)*np.sqrt(1-((x.e)**2)))
```

```
df.corr()['diameter']
```

```
a                 0.205111
e                -0.117310
i                 0.101101
om               -0.000697
w                 0.002788
q                 0.504440
ad                0.129541
per_y             0.068389
data_arc          0.439798
n_obs_used        0.456118
H                -0.722641
neo              -0.075739
pha              -0.040669
diameter          1.000000
```

```
albedo           -0.215648
moid              0.508903
class             0.422067
n                -0.404766
per               0.068389
ma                0.023790
albedo_sqaure    -0.214760
feature3          0.470327
Name: diameter, dtype: float64
```

```
dataset=df
```

```
X = dataset.drop("diameter",axis=1)
y = dataset["diameter"]
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(X)
```

```
▾ StandardScaler
StandardScaler()
```

```
sd_data=sc.transform(X)
sd_data = pd.DataFrame(sd_data)
sd_data
```

	0	1	2	3	4	5	6	
0	-0.307984	1.446014	-0.398851	0.857915	-0.997616	-1.367255	-0.093037	-0.049
1	-0.251253	-1.083786	-1.279787	0.698198	-0.288212	-0.242468	-0.221160	-0.041
2	-0.379703	0.291596	-1.158819	-0.079231	-0.087364	-1.116567	-0.209398	-0.058
3	-0.046260	0.191728	-1.140800	0.780463	-1.545652	-0.204760	-0.014075	-0.012
4	0.023274	1.413581	0.631120	0.775606	-0.027749	-0.582756	0.121197	-0.002
...	...	...	...	...	...	...	...	
134969	0.231115	3.689374	2.708688	-0.528136	-0.435022	-1.256660	0.449662	0.029
134970	0.241299	0.177313	2.470409	1.355971	-1.565995	0.557814	0.158324	0.030
134971	-0.174615	-0.898043	0.189396	0.745104	-0.113978	-0.093933	-0.166111	-0.030
134972	0.161237	1.839021	0.610655	1.695603	1.552252	-0.470517	0.246114	0.018
134973	-0.262052	-0.471650	-0.850320	-0.209627	-1.448138	-0.512830	-0.187410	-0.042

134974 rows × 21 columns

```
from sklearn.model_selection import train_test_split
X_train, X_rem, y_train, y_rem = train_test_split(X,y, train_size=0.8,shuffle=True)
X_valid, X_test, y_valid, y_test = train_test_split(X_rem,y_rem, test_size=0.5,shuf


from xgboost import XGBRegressor

model = XGBRegressor()


model.fit(
    X_train,
    y_train,
    eval_metric="rmse",
    eval_set=[(X_train, y_train), (X_valid, y_valid)],
    verbose=True)
```

```
  warnings.warn(
[0]	validation_0-rmse:4.11467	validation_1-rmse:4.14905
[1]	validation_0-rmse:2.93717	validation_1-rmse:2.97097
[2]	validation_0-rmse:2.11821	validation_1-rmse:2.16468
[3]	validation_0-rmse:1.55896	validation_1-rmse:1.61022
[4]	validation_0-rmse:1.18055	validation_1-rmse:1.23824
[5]	validation_0-rmse:0.93486	validation_1-rmse:0.99866
[6]	validation_0-rmse:0.78135	validation_1-rmse:0.85137
[7]	validation_0-rmse:0.68856	validation_1-rmse:0.76397
[8]	validation_0-rmse:0.63508	validation_1-rmse:0.71298
[9]	validation_0-rmse:0.60393	validation_1-rmse:0.68536
[10]	validation_0-rmse:0.58521	validation_1-rmse:0.66892
[11]	validation_0-rmse:0.57283	validation_1-rmse:0.65926
[12]	validation_0-rmse:0.56533	validation_1-rmse:0.65103
[13]	validation_0-rmse:0.55999	validation_1-rmse:0.64824
[14]	validation_0-rmse:0.55550	validation_1-rmse:0.64543
[15]	validation_0-rmse:0.55160	validation_1-rmse:0.64442
[16]	validation_0-rmse:0.54889	validation_1-rmse:0.64288
[17]	validation_0-rmse:0.54657	validation_1-rmse:0.64118
[18]	validation_0-rmse:0.54351	validation_1-rmse:0.63976
[19]	validation_0-rmse:0.54132	validation_1-rmse:0.63885
[20]	validation_0-rmse:0.53900	validation_1-rmse:0.63878
[21]	validation_0-rmse:0.53695	validation_1-rmse:0.63797
[22]	validation_0-rmse:0.53562	validation_1-rmse:0.63767
[23]	validation_0-rmse:0.53419	validation_1-rmse:0.63645
[24]	validation_0-rmse:0.53331	validation_1-rmse:0.63635
[25]	validation_0-rmse:0.53214	validation_1-rmse:0.63660
[26]	validation_0-rmse:0.53046	validation_1-rmse:0.63547
[27]	validation_0-rmse:0.52839	validation_1-rmse:0.63490
[28]	validation_0-rmse:0.52755	validation_1-rmse:0.63518
[29]	validation_0-rmse:0.52716	validation_1-rmse:0.63513
[30]	validation_0-rmse:0.52412	validation_1-rmse:0.63672
[31]	validation_0-rmse:0.52373	validation_1-rmse:0.63689
[32]	validation_0-rmse:0.52284	validation_1-rmse:0.63648
[33]	validation_0-rmse:0.52191	validation_1-rmse:0.63699
[34]	validation_0-rmse:0.51929	validation_1-rmse:0.63697
[35]	validation_0-rmse:0.51766	validation_1-rmse:0.63708
[36]	validation_0-rmse:0.51701	validation_1-rmse:0.63667
[37]	validation_0-rmse:0.51480	validation_1-rmse:0.63581
[38]	validation_0-rmse:0.51330	validation_1-rmse:0.63512
[39]	validation_0-rmse:0.51214	validation_1-rmse:0.63484
[40]	validation_0-rmse:0.50987	validation_1-rmse:0.63390
[41]	validation_0-rmse:0.50759	validation_1-rmse:0.63382
[42]	validation_0-rmse:0.50552	validation_1-rmse:0.63422
[43]	validation_0-rmse:0.50434	validation_1-rmse:0.63391
[44]	validation_0-rmse:0.50371	validation_1-rmse:0.63397
[45]	validation_0-rmse:0.50183	validation_1-rmse:0.63547
[46]	validation_0-rmse:0.50048	validation_1-rmse:0.63535
[47]	validation_0-rmse:0.50006	validation_1-rmse:0.63532
[48]	validation_0-rmse:0.49871	validation_1-rmse:0.63563
[49]	validation_0-rmse:0.49860	validation_1-rmse:0.63553
[50]	validation_0-rmse:0.49626	validation_1-rmse:0.63617
[51]	validation_0-rmse:0.49494	validation_1-rmse:0.63666
[52]	validation_0-rmse:0.49359	validation_1-rmse:0.63666
[53]	validation_0-rmse:0.49189	validation_1-rmse:0.63717
[54]	validation_0-rmse:0.49049	validation_1-rmse:0.63648
[55]	validation_0-rmse:0.48991	validation_1-rmse:0.63629
```

```
[56]	validation_0-rmse:0.48875	validation_1-rmse:0.63589
```

```
pred =  model.predict(X_test)
```

```
from sklearn.metrics import mean_squared_error
```

```
mean_squared_error(pred,y_test)
```

```
0.4822733426252814
```

```
[65]	validation_0-rmse:0.48071	validation_1-rmse:0.63710
```

```
from sklearn.metrics import r2_score
```

```
r2_score(pred,y_test)
```

```
0.9657065075233509
```

```
[72]	validation_0-rmse:0.47363	validation_1-rmse:0.63584
```

```
model2 = XGBRegressor(learning_rate=0.05)
```

```
model2.fit(
    X_train,
    y_train,
    eval_metric="rmse",
    eval_set=[(X_train, y_train), (X_valid, y_valid)],
    verbose=True)
```

```
  warnings.warn(
[0]	validation_0-rmse:5.52032	validation_1-rmse:5.55545
[1]	validation_0-rmse:5.25197	validation_1-rmse:5.28661
[2]	validation_0-rmse:4.99721	validation_1-rmse:5.03183
[3]	validation_0-rmse:4.75534	validation_1-rmse:4.78952
[4]	validation_0-rmse:4.52551	validation_1-rmse:4.55982
[5]	validation_0-rmse:4.30717	validation_1-rmse:4.34161
[6]	validation_0-rmse:4.10002	validation_1-rmse:4.13379
[7]	validation_0-rmse:3.90334	validation_1-rmse:3.93834
[8]	validation_0-rmse:3.71673	validation_1-rmse:3.75132
[9]	validation_0-rmse:3.53957	validation_1-rmse:3.57373
[10]	validation_0-rmse:3.37109	validation_1-rmse:3.40572
[11]	validation_0-rmse:3.21131	validation_1-rmse:3.24711
[12]	validation_0-rmse:3.05984	validation_1-rmse:3.09559
[13]	validation_0-rmse:2.91622	validation_1-rmse:2.95293
[14]	validation_0-rmse:2.77935	validation_1-rmse:2.81772
[15]	validation_0-rmse:2.64995	validation_1-rmse:2.68936
[16]	validation_0-rmse:2.52690	validation_1-rmse:2.56689
[17]	validation_0-rmse:2.41064	validation_1-rmse:2.45224
[18]	validation_0-rmse:2.30001	validation_1-rmse:2.34139
[19]	validation_0-rmse:2.19556	validation_1-rmse:2.23880
[20]	validation_0-rmse:2.09611	validation_1-rmse:2.14088
[21]	validation_0-rmse:2.00228	validation_1-rmse:2.04826
[22]	validation_0-rmse:1.91294	validation_1-rmse:1.95967
[23]	validation_0-rmse:1.82861	validation_1-rmse:1.87646
[24]	validation_0-rmse:1.74872	validation_1-rmse:1.79746
[25]	validation_0-rmse:1.67324	validation_1-rmse:1.72294
[26]	validation_0-rmse:1.60191	validation_1-rmse:1.65309
[27]	validation_0-rmse:1.53428	validation_1-rmse:1.58648
[28]	validation_0-rmse:1.47052	validation_1-rmse:1.52418
[29]	validation_0-rmse:1.41031	validation_1-rmse:1.46525
[30]	validation_0-rmse:1.35387	validation_1-rmse:1.40988
[31]	validation_0-rmse:1.30023	validation_1-rmse:1.35666
[32]	validation_0-rmse:1.24982	validation_1-rmse:1.30711
[33]	validation_0-rmse:1.20224	validation_1-rmse:1.26095
[34]	validation_0-rmse:1.15752	validation_1-rmse:1.21747
[35]	validation_0-rmse:1.11549	validation_1-rmse:1.17650
[36]	validation_0-rmse:1.07609	validation_1-rmse:1.13844
[37]	validation_0-rmse:1.03894	validation_1-rmse:1.10214
[38]	validation_0-rmse:1.00422	validation_1-rmse:1.06831
[39]	validation_0-rmse:0.97164	validation_1-rmse:1.03670
[40]	validation_0-rmse:0.94119	validation_1-rmse:1.00752
[41]	validation_0-rmse:0.91260	validation_1-rmse:0.98019
[42]	validation_0-rmse:0.88598	validation_1-rmse:0.95426
[43]	validation_0-rmse:0.86118	validation_1-rmse:0.93094
[44]	validation_0-rmse:0.83795	validation_1-rmse:0.90844
[45]	validation_0-rmse:0.81623	validation_1-rmse:0.88769
[46]	validation_0-rmse:0.79600	validation_1-rmse:0.86832
[47]	validation_0-rmse:0.77720	validation_1-rmse:0.85065
[48]	validation_0-rmse:0.75962	validation_1-rmse:0.83397
[49]	validation_0-rmse:0.74341	validation_1-rmse:0.81825
[50]	validation_0-rmse:0.72819	validation_1-rmse:0.80398
[51]	validation_0-rmse:0.71425	validation_1-rmse:0.79077
[52]	validation_0-rmse:0.70125	validation_1-rmse:0.77858
[53]	validation_0-rmse:0.68904	validation_1-rmse:0.76715
[54]	validation_0-rmse:0.67789	validation_1-rmse:0.75663
[55]	validation_0-rmse:0.66748	validation_1-rmse:0.74697
```

```
[56]	validation_0-rmse:0.65792	validation_1-rmse:0.73751
```

```
pred2 =  model2.predict(X_test)
```

```
mean_squared_error(pred2,y_test)
```

```
0.48202692540488873
[63]	validation_0-rmse:0.60826	validation_1-rmse:0.69159
```

```
r2_score(pred2,y_test)
```

```
0.9647207279898601
[68]	validation_0-rmse:0.58566	validation_1-rmse:0.67247
```

```
from sklearn.svm import SVR
svr = SVR(C=1.0, epsilon=0.2)
svr.fit(X_train, y_train)
```

```
▾ SVR
SVR(epsilon=0.2)
```

```
[77]	validation_0-rmse:0.56094	validation_1-rmse:0.65080
```

```
pred3 = svr.predict(X_test)
```

```
mean_squared_error(pred3,y_test)
```

```
5.096299185338949
[83]	validation_0-rmse:0.55048	validation_1-rmse:0.64314
```

```
r2_score(pred3,y_test)
```

```
0.25714575007025553
[88]	validation_0-rmse:0.54384	validation_1-rmse:0.63783
```

```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=0)
regressor.fit(X_train, y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor(random_state=0)
```

```
[97]	validation_0-rmse:0.53536	validation_1-rmse:0.63318
```

```
pred4 = regressor.predict(X_test)
```

```
mean_squared_error(pred4,y_test)
```

```
0.8255055720847533
             colsample_bytree=None, early_stopping_rounds=None,
```

```
r2_score(pred4,y_test)
```

```
0.9427296704965822
             max_delta_step=None, max_depth=None, max_leaves=None,
```

```
from sklearn.ensemble import BaggingRegressor
regr = BaggingRegressor(base_estimator=DecisionTreeRegressor(),n_estimators=100, ra
regr.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/ensemble/_base.py:166: FutureWa
  warnings.warn(
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  4.4min finished
```

▸ **BaggingRegressor**

▸ **base_estimator: DecisionTreeRegressor**

▸ DecisionTreeRegressor

```
regr.oob_score_
```

```
0.9743095970686392
```

```
pred5 = regr.predict(X_test)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.1s finished
```

```
mean_squared_error(pred5,y_test)
```

```
0.4894245185597643
```

```
r2_score(pred5,y_test)
```

```
0.9648656508410457
```

```
from sklearn.ensemble import GradientBoostingRegressor
reg2 = GradientBoostingRegressor(random_state=0)
reg2.fit(X_train, y_train)
```

▾ GradientBoostingRegressor

GradientBoostingRegressor(random_state=0)

```
pred6 = reg2.predict(X_test)
```

```
mean_squared_error(pred6,y_test)
```

```
0.4959430840312019
```

```
r2_score(pred6,y_test)
```

```
0.9639106172846329
```

```
from tensorflow import keras


def build_model(n_h=1,n_n=30,lr=3e-3,input_shape=X_train.shape[1:]):
    ann = keras.models.Sequential()
    options = {"input_shape": input_shape}
    for layer in range(n_h):
        ann.add(keras.layers.Dense(n_n, activation="relu", **options))
        options = {}
    ann.add(keras.layers.Dense(1,activation="linear", **options))
    lr_adp = keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=lr,c
    optimizer1 = keras.optimizers.SGD(lr_adp,momentum=0.9)
    ann.compile(loss='mean_absolute_error', optimizer=optimizer1, metrics=['mean_ab
    return ann


keras_reg = keras.wrappers.scikit_learn.KerasRegressor(build_model)
```

```
<ipython-input-103-651c14c6d32f>:1: DeprecationWarning: KerasRegressor is dep
  keras_reg = keras.wrappers.scikit_learn.KerasRegressor(build_model)
```

```
keras_reg.fit(X_train,y_train,epochs=100,
             validation_data=(X_valid,y_valid))
```

```
Epoch 89/100
3375/3375 [==============================] - 9s 3ms/step - loss: 2.1659 - me
Epoch 90/100
3375/3375 [==============================] - 9s 3ms/step - loss: 2.1659 - me
Epoch 91/100
3375/3375 [==============================] - 9s 3ms/step - loss: 2.1659 - me
Epoch 92/100
3375/3375 [==============================] - 8s 2ms/step - loss: 2.1659 - me
Epoch 93/100
3375/3375 [==============================] - 10s 3ms/step - loss: 2.1659 - m
Epoch 94/100
3375/3375 [==============================] - 7s 2ms/step - loss: 2.1659 - me
Epoch 95/100
3375/3375 [==============================] - 10s 3ms/step - loss: 2.1659 - m
Epoch 96/100
3375/3375 [==============================] - 7s 2ms/step - loss: 2.1659 - me
Epoch 97/100
3375/3375 [==============================] - 9s 3ms/step - loss: 2.1659 - me
Epoch 98/100
3375/3375 [==============================] - 8s 2ms/step - loss: 2.1659 - me
Epoch 99/100
3375/3375 [==============================] - 10s 3ms/step - loss: 2.1659 - m
Epoch 100/100
3375/3375 [==============================] - 7s 2ms/step - loss: 2.1659 - me
<keras.callbacks.History at 0x7fdb3874d5e0>
```

```
pred7 = keras_reg.predict(X_test)
```

```
422/422 [==============================] - 1s 2ms/step
```

```
mean_squared_error(pred7,y_test)
```

```
15.28869306463847
```

```
r2_score(pred7,y_test)
```

```
-268961532769091.6
```

```
from sklearn.model_selection import GridSearchCV
parameters = {'n_estimators':[90, 110]}
clf = GridSearchCV(regr, parameters)
```

```
import warnings
warnings.filterwarnings("ignore")
```

```
clf.fit(X_train, y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.9min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  3.0min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.9min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.9min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.9min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  3.6min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  3.5min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.7s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  3.6min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  3.6min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  3.5min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  4.7min finished
```

```
clf.best_params_
```

```
{'n_estimators': 110}
```

▸ DecisionTreeRegressor

```
final_model = clf.best_estimator_
```

```
pred8 = final_model.predict(X_test)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.1s finished
```