XGBoost Regression

XgBoost is an ensemble learner.

XGBoost regression or Extreme Gradient Boosting is supervised learning technique which uses Decision Trees and and Gradient Boosting method. It contains loss function and regularized parameters which gives ot name Extreme.

It tells about the difference between actual values and predicted values, i.e how far the model results are from the real values.

Methodology

1. Import all the necessary libraries for the task.

2. Read Data into the DataFrame.

3. Understand Data by using methods like hea() and info().

4. Cleaning Data or Preprocessing on DataSet.

5. Visualizing Data using Scatterplots, histograms and Heatmaps.

6. Splitting Data into features and Target varieable.

7. Splitting Data into Training and Testing Data.

8. Training model using XGBoost Regressor on training Data.

9. Predicting values on Testing Data.

10. Calculatin RMSE and R2 Score on Testing Data to see how well model preformed.m

Importing Libraries

In [1]:

```
import pandas as pd
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
import xgboost as xgb
%matplotlib inline
```

Reading Dataset From .csv file as Pd Data Frame

In [2]:

```
DataSet = pd.read_csv("Asteroid_Updated.csv")
```

```
C:\Users\nabhr\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3057: DtypeWarning: Columns (0,10,15,16,23,24) have mix
ed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Printing first 10 Rows of Data Set to cunderstand the data.

In [3]:

```
DataSet.head(10)
```

Out[3]:

	name	a	e	i	om	w	q	ad	per_y	data_arc	...	UB	IR	spec_B	spec_T	G	moid	class	n	p
0	Ceres	2.769165	0.076009	10.594067	80.305532	73.597694	2.558684	2.979647	4.608202	8822.0	...	0.426	NaN	C	G	0.12	1.594780	MBA	0.213885	1683.1457
1	Pallas	2.772466	0.230337	34.836234	173.080063	310.048857	2.133865	3.411067	4.616444	72318.0	...	0.284	NaN	B	B	0.11	1.233240	MBA	0.213503	1686.1559
2	Juno	2.669150	0.256942	12.988919	169.852760	248.138626	1.983332	3.354967	4.360814	72684.0	...	0.433	NaN	Sk	S	0.32	1.034540	MBA	0.226019	1592.7872
3	Vesta	2.361418	0.088721	7.141771	103.810804	150.728541	2.151909	2.570926	3.628837	24288.0	...	0.492	NaN	V	V	0.32	1.139480	MBA	0.271609	1325.4327
4	Astraea	2.574249	0.191095	5.366988	141.576605	358.687607	2.082324	3.066174	4.130323	63507.0	...	0.411	NaN	S	S	NaN	1.095890	MBA	0.238632	1508.6004
5	Hebe	2.425160	0.203007	14.737901	138.640203	239.807490	1.932835	2.917485	3.776755	62329.0	...	0.399	NaN	S	S	0.24	0.973965	MBA	0.260972	1379.4597
6	Iris	2.385334	0.231206	5.523651	259.563231	145.265106	1.833831	2.936837	3.684105	62452.0	...	0.484	NaN	S	S	NaN	0.846100	MBA	0.267535	1345.6191
7	Flora	2.201764	0.156499	5.886955	110.889330	285.287462	1.857190	2.546339	3.267115	62655.0	...	0.489	NaN	NaN	S	0.28	0.874176	MBA	0.301681	1193.3137
8	Metis	2.385637	0.123114	5.576816	68.908577	6.417369	2.091931	2.679342	3.684806	61821.0	...	0.496	NaN	NaN	S	0.17	1.106910	MBA	0.267484	1345.8753
9	Hygiea	3.141539	0.112461	3.831560	283.202167	312.315206	2.788240	3.494839	5.568291	62175.0	...	0.351	NaN	C	C	NaN	1.778390	MBA	0.177007	2033.8182

10 rows × 31 columns

Analysing Data Coloumns and it's types

In [4]:

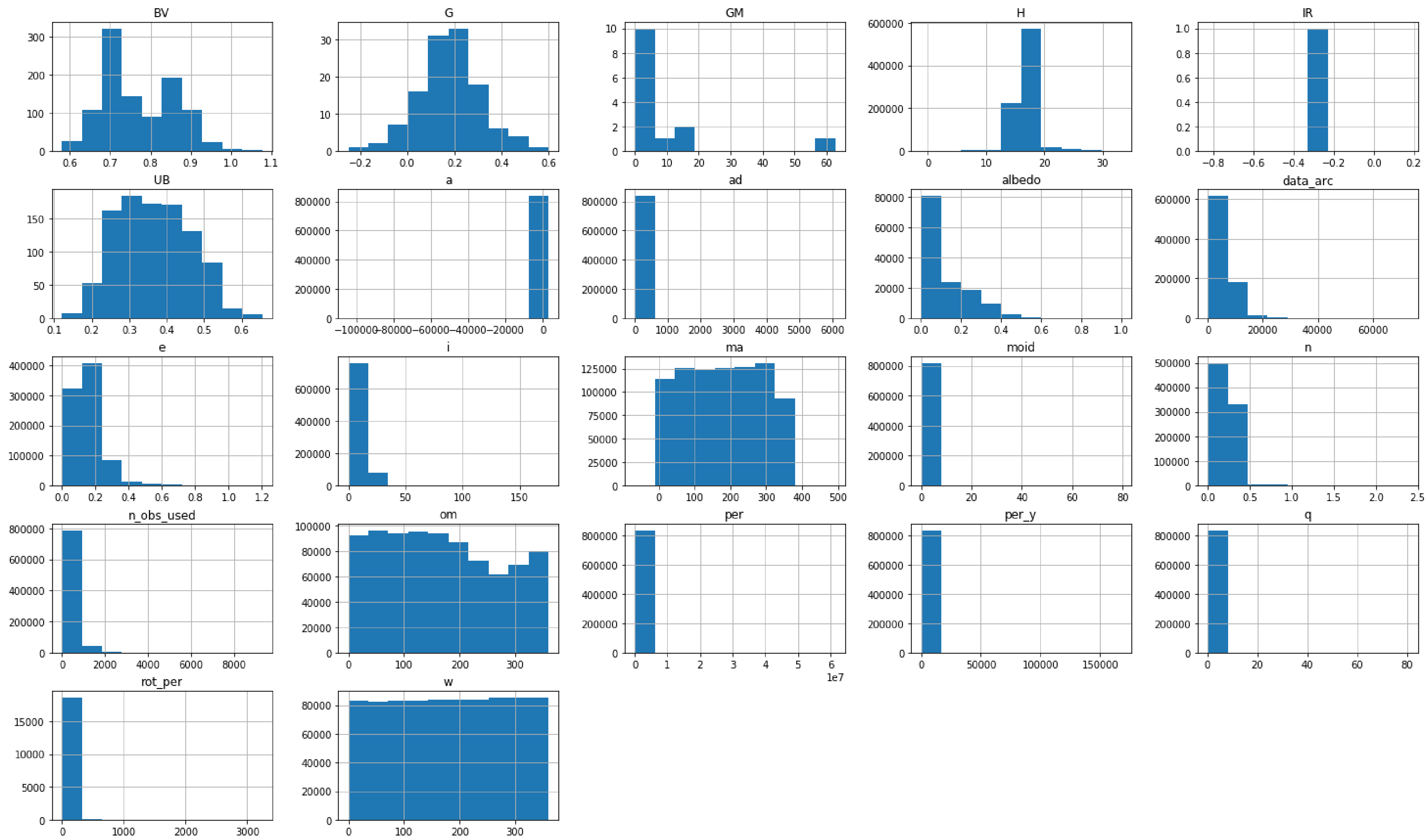
```
DataSet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 839714 entries, 0 to 839713
Data columns (total 31 columns):
name              21967 non-null object
a                 839712 non-null float64
e                 839714 non-null float64
i                 839714 non-null float64
om                839714 non-null float64
w                 839714 non-null float64
q                 839714 non-null float64
ad                839708 non-null float64
per_y             839713 non-null float64
data_arc          824240 non-null float64
condition_code    838847 non-null object
n_obs_used        839714 non-null int64
H                 837025 non-null float64
neo               839708 non-null object
pha               823272 non-null object
diameter          137636 non-null object
extent            18 non-null object
albedo            136409 non-null float64
rot_per           18796 non-null float64
GM                14 non-null float64
BV                1021 non-null float64
UB                979 non-null float64
IR                1 non-null float64
spec_B            1666 non-null object
spec_T            980 non-null object
G                 119 non-null float64
moid              823272 non-null float64
class             839714 non-null object
n                 839712 non-null float64
per               839708 non-null float64
ma                839706 non-null float64
dtypes: float64(21), int64(1), object(9)
memory usage: 198.6+ MB
```

Visualizing Data and analyzing for irregularities

In [5]:

```
DataSet.hist(figsize = (25,15))
plt.show()
```



Since there are many coloumns which have very less non-null values. We can drop all those coloumns which have non-null values much less than total number of values.

In [6]:

```
DataSet = DataSet.drop('IR', axis = 1)
```

```
DataSet = DataSet.drop('UB', axis = 1)
DataSet = DataSet.drop('G', axis = 1)
DataSet = DataSet.drop('GM', axis = 1)
DataSet = DataSet.drop('extent', axis = 1)
DataSet = DataSet.drop('BV', axis = 1)
DataSet = DataSet.drop('spec_B', axis = 1)
DataSet = DataSet.drop('spec_T', axis = 1)
DataSet = DataSet.drop('rot_per', axis = 1)
DataSet = DataSet.drop('name', axis = 1)
```

Analysing Data Coloumn Again and diving deep itno data cleaning and processing

In [7]:

```
DataSet.shape
DataSet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 839714 entries, 0 to 839713
Data columns (total 21 columns):
a                 839712 non-null float64
e                 839714 non-null float64
i                 839714 non-null float64
om                839714 non-null float64
w                 839714 non-null float64
q                 839714 non-null float64
ad                839708 non-null float64
per_y             839713 non-null float64
data_arc          824240 non-null float64
condition_code    838847 non-null object
n_obs_used        839714 non-null int64
H                 837025 non-null float64
neo               839708 non-null object
pha               823272 non-null object
diameter          137636 non-null object
albedo            136409 non-null float64
moid              823272 non-null float64
class             839714 non-null object
n                 839712 non-null float64
per               839708 non-null float64
ma                839706 non-null float64
dtypes: float64(15), int64(1), object(5)
memory usage: 134.5+ MB
```

As analysed from info, Dataset still has rows with null entry, se lets Delete those rows which have any entries as NA values.

In [8]:

```
DataSet = DataSet.dropna(axis = 'index', how = 'any')
```

In [9]:

```
DataSet.shape
```

Out[9]:

```
(136005, 21)
```

Now checking the rows with object values.

In [10]:

```
for col in DataSet.columns:
    if(DataSet[col].dtype == object):
        A = DataSet[col].unique()
        print(col, A)
```

```
condition_code [0 1 3 2 '0' '1' '2' '4' '5' '9' '3' '7' 5.0 6.0 4.0 7.0 9.0 8.0 '8' '6']
neo ['N' 'Y']
pha ['N' 'Y']
diameter ['939.4' '545' '246.596' ... 0.122 0.6509999999999999 1.077]
class ['MBA' 'OMB' 'MCA' 'AMO' 'IMB' 'TJN' 'CEN' 'APO' 'ATE' 'AST' 'TNO']
```

Dropping those columns with very large number of unique values or very small number of unique values

In [11]:

```
DataSet = DataSet.drop('condition_code', axis = 1)
DataSet = DataSet.drop('neo', axis = 1)
DataSet = DataSet.drop('pha', axis = 1)
DataSet = DataSet.drop('class', axis = 1)
```

Since diameter column only has in values as strings, lets convert diameter and n_obs_used column to float.

In [12]:

```
DataSet['diameter'] = DataSet['diameter'].astype(float)
DataSet['n_obs_used'] = DataSet['n_obs_used'].astype(float)
```

In [13]:

```
DataSet.info()
```

```
DataSet.shape
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 136005 entries, 0 to 810375
Data columns (total 17 columns):
a             136005 non-null float64
e             136005 non-null float64
i             136005 non-null float64
om            136005 non-null float64
w             136005 non-null float64
q             136005 non-null float64
ad            136005 non-null float64
per_y         136005 non-null float64
data_arc      136005 non-null float64
n_obs_used    136005 non-null float64
H             136005 non-null float64
diameter      136005 non-null float64
albedo        136005 non-null float64
moid          136005 non-null float64
n             136005 non-null float64
per           136005 non-null float64
ma            136005 non-null float64
dtypes: float64(17)
memory usage: 18.7 MB
```

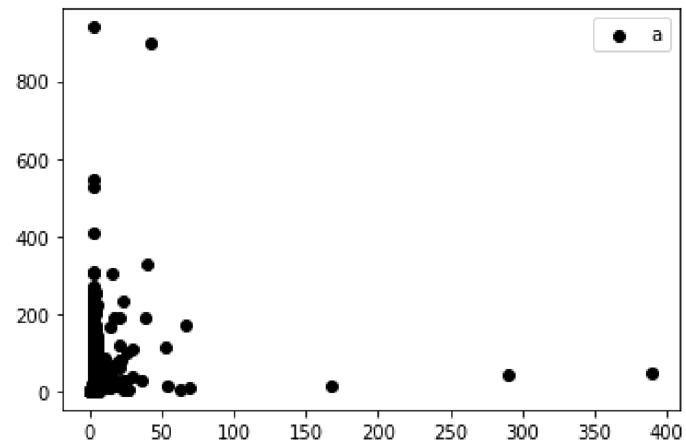
Out[13]:

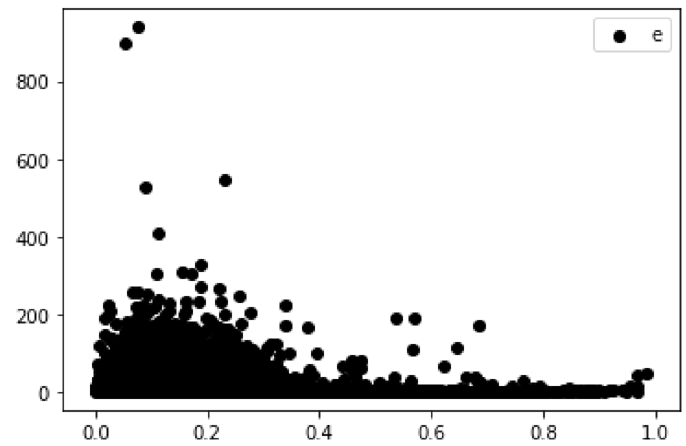
```
(136005, 17)
```

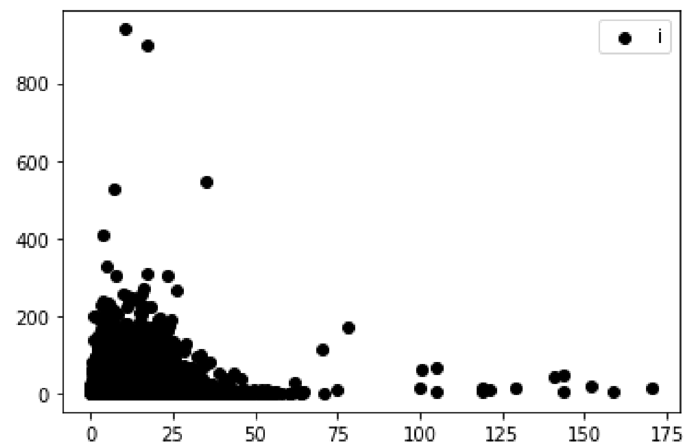
Plotting Scatter Plots.

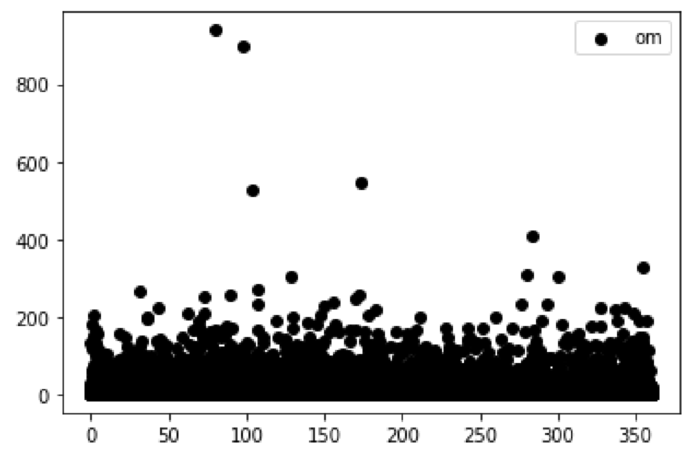
In [14]:

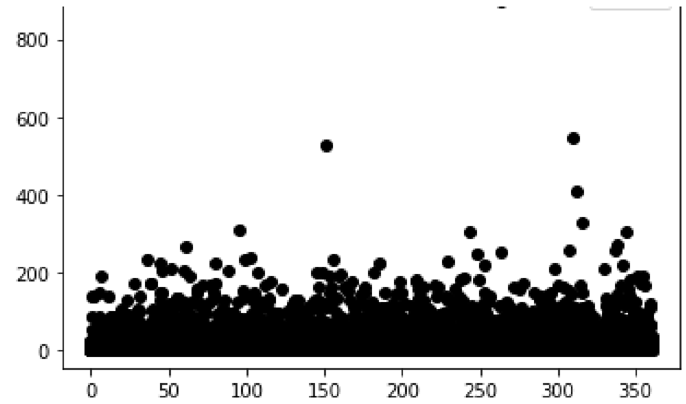
```
for col in DataSet.columns:
    plt.scatter(DataSet.loc[:,col], DataSet.loc[:,'diameter'], color = 'black')
    plt.legend([col],loc = 'upper right')
    plt.show()
```

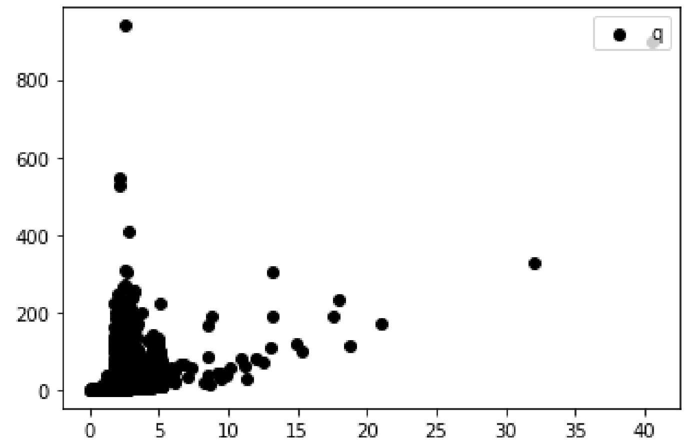


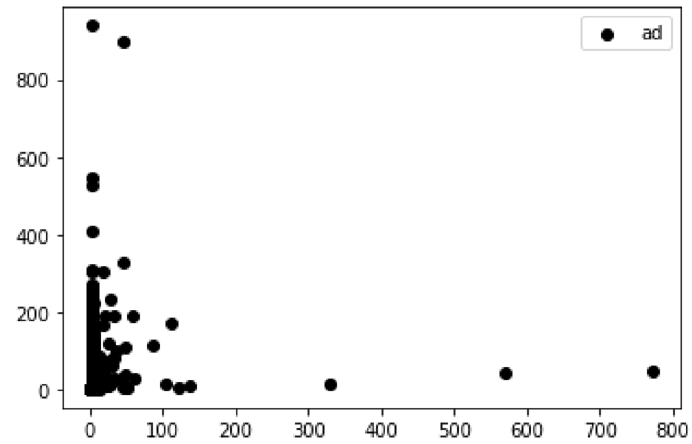


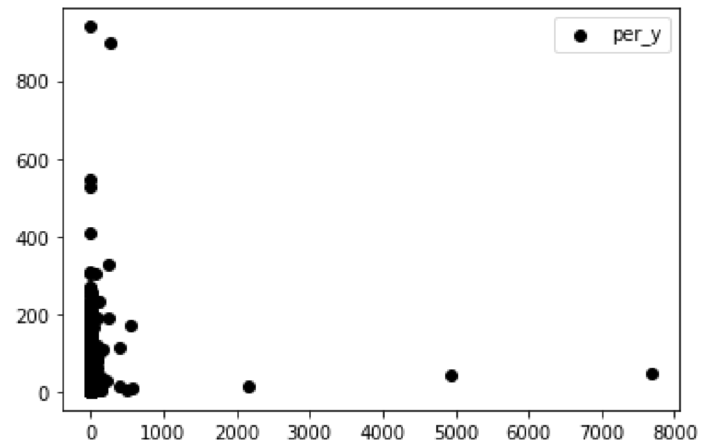


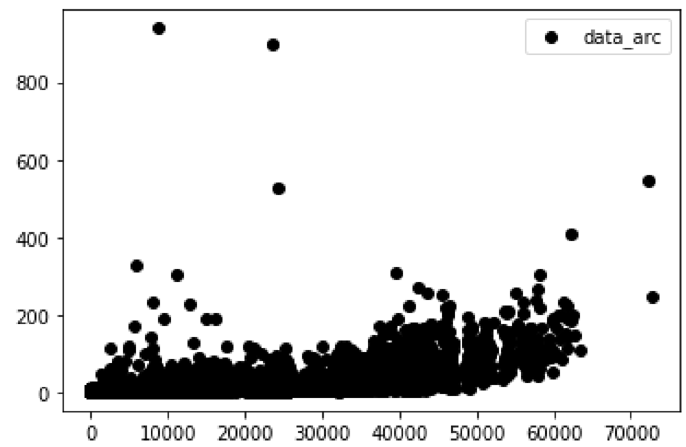


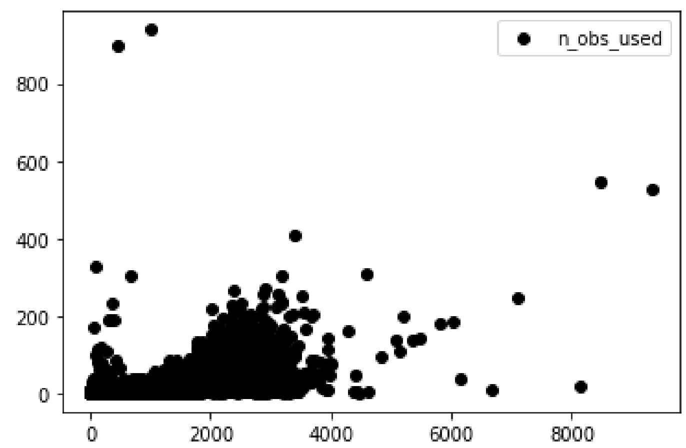


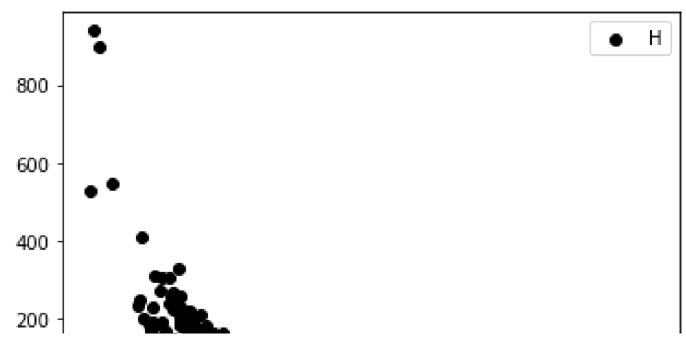

g

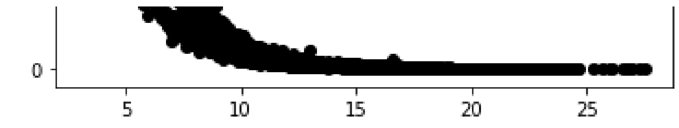

ad

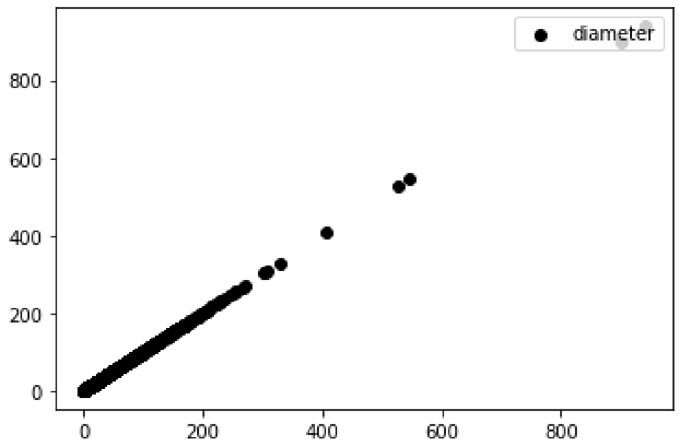

per_y

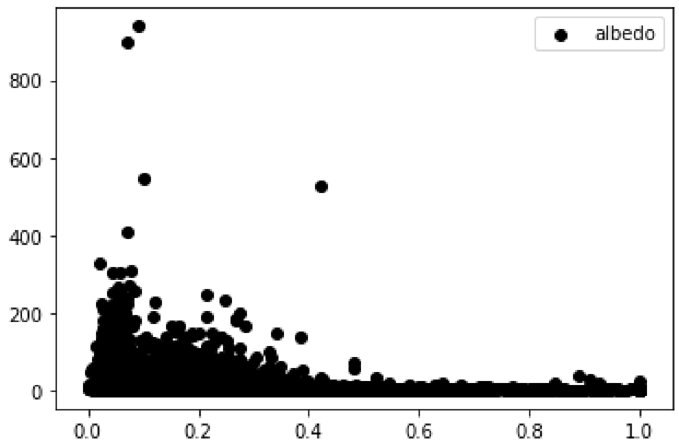

data_arc

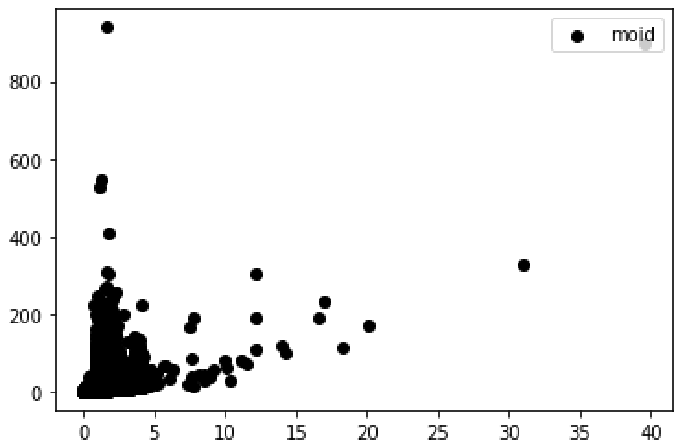

n_obs_used

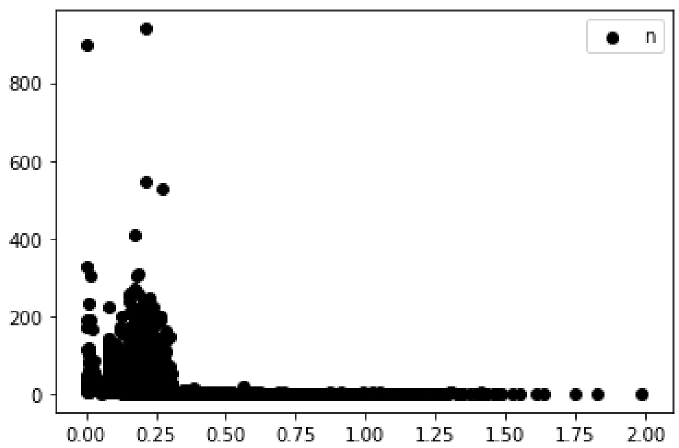

H

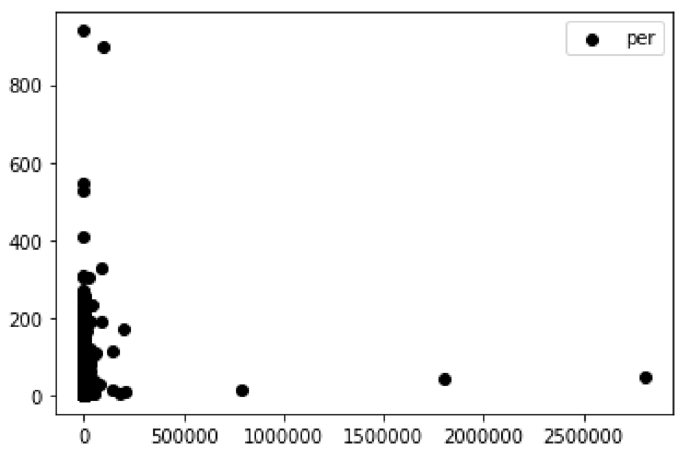


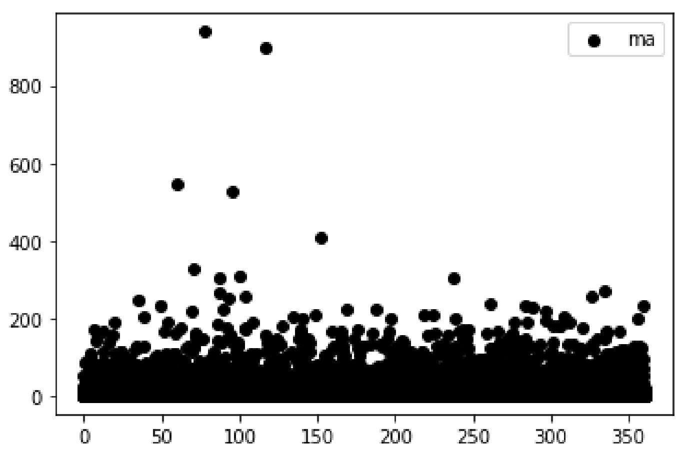








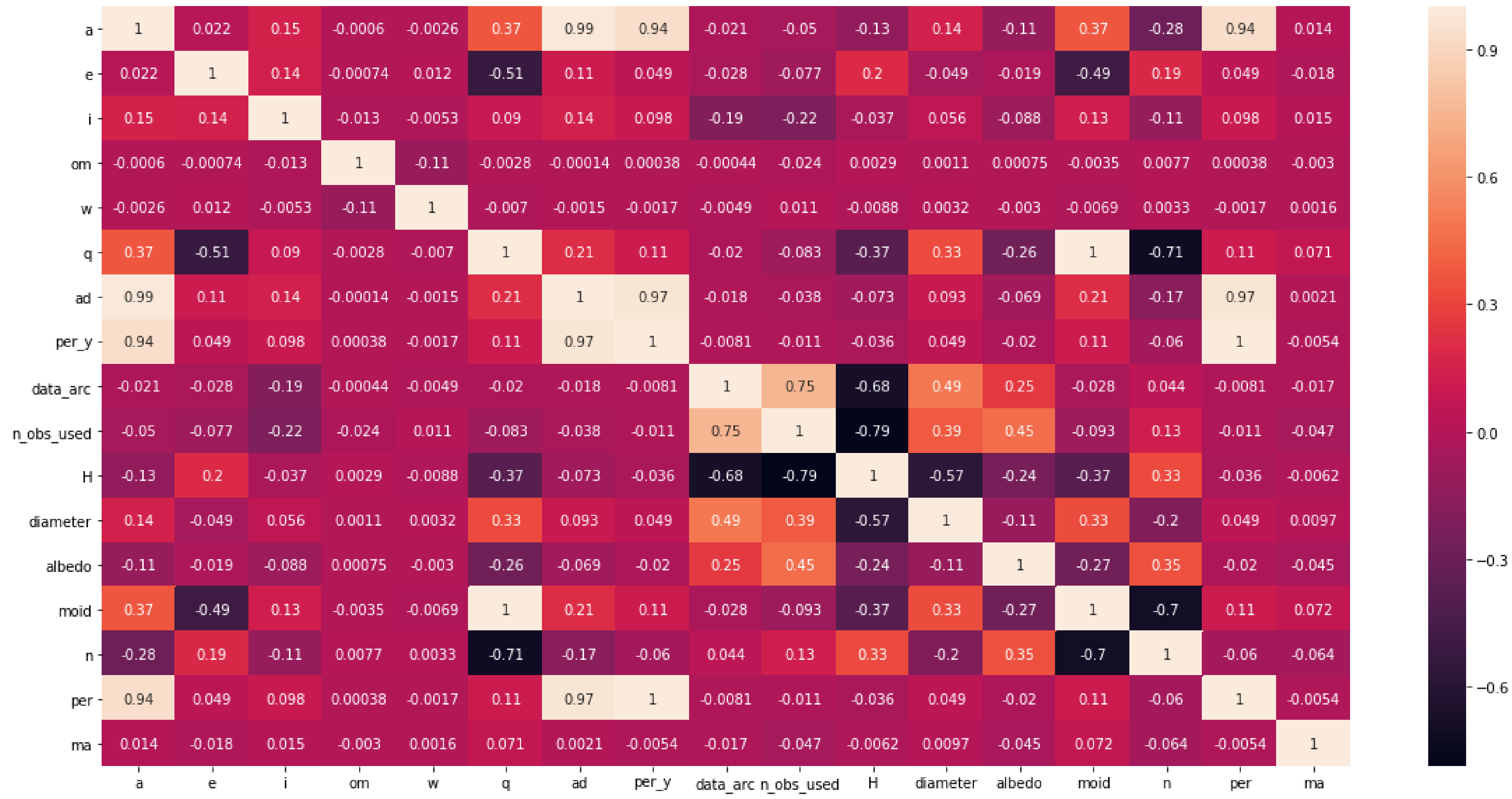




Plotting Heatmap to better visualize the features."

In [15]:

```
def Heat(Datax):
    corelate = Datax.corr()
    fig, ax = plt.subplots(figsize = (20,10))
    ax = sns.heatmap(corelate, annot = True)
    plt.show()
Heat(DataSet)
```



Since XGBoost used Decision Trees which checks for every features individually in stumps to train the model, we don't need to eliminate any features based on coreltion matrix.

In [16]:

```
DataSet.info()
DataSet.shape
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 136005 entries, 0 to 810375
Data columns (total 17 columns):
a             136005 non-null float64
e             136005 non-null float64
i             136005 non-null float64
om            136005 non-null float64
w             136005 non-null float64
q             136005 non-null float64
ad            136005 non-null float64
per_y         136005 non-null float64
data_arc      136005 non-null float64
n_obs_used    136005 non-null float64
H             136005 non-null float64
diameter      136005 non-null float64
albedo        136005 non-null float64
moid          136005 non-null float64
n             136005 non-null float64
per           136005 non-null float64
ma            136005 non-null float64
dtypes: float64(17)
memory usage: 18.7 MB
```

Out[16]:

```
(136005, 17)
```

Splitting Data Set into Features and Target Variables.

In [17]:

```
Y = DataSet.loc[:,'diameter']
X = DataSet.drop('diameter', axis = 1)
```

Splitting Data Sets into Training and Testing Data

In [18]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=5)
```

Applying XGBoost to train a model 'XG'.

In [19]:

```python
XG = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.5, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 20)
```

Fitting our training data to the model

In [20]:

```python
XG.fit(X_train,y_train)
```

```
[12:51:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.2.0/src/objective/regression_obj.cu:174: reg:linear i
s now deprecated in favor of reg:squarederror.
[12:51:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.2.0/src/objective/regression_obj.cu:174: reg:linear i
s now deprecated in favor of reg:squarederror.
```

Out[20]:

```
XGBRegressor(alpha=10, base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.5, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.1, max_delta_step=0, max_depth=5,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=20, n_jobs=0, num_parallel_tree=1,
             objective='reg:linear', random_state=0, reg_alpha=10, reg_lambda=1,
             scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

Predicting values of testing Data using the model.

In [21]:

```python
Y_Predict = XG.predict(X_test)
```

Calculating RMSE and r2 Score based on predictions done on Testing Data

In [22]:

```python
rmse = np.sqrt(mean_squared_error(y_test, Y_Predict))
print("RMSE: %f" % (rmse))
r2 = r2_score(y_test, Y_Predict)
print(r2)
```

```
RMSE: 2.213173
0.9274388301899217
```

RMSE for predicted values on Testing Data: 2.21.

R2 Score : .92

In [ ]:

```python

```