SEMESTER 2

Ankita Pandey 23060641070
Girija Deshpande 23060641058

ASSIGNMENT -LINEAR MODEL

GROUP NO. 33

TOPIC: MODEL DEVELOPMENT OF ASTROPHYSICS

CODE:-

```
TO#Importing all necessary
libraries import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import missingno as msno
from prettytable import PrettyTable
import plotly.express as px import
scipy.stats
from plotly.subplots import make_subplots
import plotly.graph_objects as go from
pandas_profiling import ProfileReport
import plotly.offline as pyo
pyo.init_notebook_mode() # looking at the
data shape

print(df.shape)
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns")
(139708, 37)
There are 139708 rows and 37 columns
df.head()
                  full_name     name neo pha     H     G  diameter  \
0         1 Ceres (A801 AA)    Ceres   N   N  3.33  0.12    939.40
1        2 Pallas (A802 FA)   Pallas   N   N  4.11  0.11    513.00
2          3 Juno (A804 RA)     Juno   N   N  5.12  0.32    246.59
3         4 Vesta (A807 FA)    Vesta   N   N  3.20  0.32    525.40
4       5 Astraea (A845 XA)  Astraea   N   N  7.01   NaN    106.69
                  extent  albedo    rot_per  ...       n           tp
per  \
0  964.4 x 964.2 x 891.8  0.0900   9.074170  ...  0.2142  2459920.37
1680.0
1            568x532x448  0.1550   7.813221  ...  0.2139  2460010.50
1680.0
2                    NaN  0.2140   7.210000  ...  0.2259  2460036.83
1590.0
3  572.6 x 557.2 x 446.4  0.4228   5.342128  ...  0.2715  2459575.12
1330.0
4                    NaN  0.2740  16.806000  ...  0.2382  2460436.30
1510.0
   per_y  moid moid_jup  class  data_arc  condition_code      rms
0   4.60  1.59     2.09    MBA    9520.0               0  0.43153
1   4.61  1.23     1.85    MBA   79390.0               0  0.35570
2   4.36  1.04     2.19    MBA   79466.0               0  0.34530
```

```
3   3.63  1.14     2.47    MBA   25743.0               0  0.40095
4   4.14  1.10     1.96    MBA   64243.0               0  0.52133

[5 rows x 37 columns]
```

```
df.tail()
```

```
                 full_name name neo pha      H   G  diameter extent
albedo  \
139703          (2019 AR40)  NaN   N   N  18.30 NaN     1.870    NaN
0.073
139704          (2019 BY5)  NaN   N   N  17.10 NaN     2.182    NaN
0.054
139705          (2019 BX6)  NaN   N   N  17.45 NaN     1.688    NaN
NaN
139706          (2019 BB7)  NaN   N   N  16.80 NaN     2.887    NaN
0.111
139707          (2019 EJ2)  NaN   N   N  17.40 NaN     2.226    NaN
0.074

        rot_per  ...       n          tp     per  per_y   moid
moid_jup  \
139703      NaN  ...  0.2171  2460389.46  1660.0   4.54  1.120
1.82
139704      NaN  ...  0.2200  2459982.46  1640.0   4.48  0.985
2.01
139705      NaN  ...  0.2245  2459941.00  1600.0   4.39  1.060
2.11
139706      NaN  ...  0.2117  2460429.02  1700.0   4.66  1.360
1.77
139707      NaN  ...  0.2185  2460092.67  1650.0   4.51  1.120
1.93

        class  data_arc  condition_code      rms  139703
MBA    6557.0               1  0.52751
139704    MBA    3763.0               0  0.42858
139705    MBA    7084.0               0  0.46238
139706    MBA    4704.0               0  0.51827
139707    MBA    5246.0               0  0.52810

[5 rows x 37 columns]
```

```
# the variaous columns we have in our dataset
df.columns
```

```
Index(['full_name', 'name', 'neo', 'pha', 'H', 'G', 'diameter',
'extent',
       'albedo', 'rot_per', 'GM', 'BV', 'UB', 'IR', 'spec_B',
'spec_T',
```

```
       'H_sigma', 'diameter_sigma', 'epoch', 'e', 'a', 'q', 'i', 'om',
'w',
       'ma', 'ad', 'n', 'tp', 'per', 'per_y', 'moid', 'moid_jup',
'class',
       'data_arc', 'condition_code', 'rms'],
dtype='object')
```

```python
# summary of dtypes
df.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 139708 entries, 0 to 139707
Columns: 37 entries, full_name to rms
dtypes: float64(28), int64(1), object(8)
memory usage: 39.4+ MB
```

```python
# detailed infoabout each columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 139708 entries, 0 to 139707
Data columns (total 37 columns):
 #   Column          Non-Null Count   Dtype  -
--  ------          --------------   -----   0
full_name       139708 non-null  object  1
name            15984 non-null   object  2
neo             139708 non-null  object
 3   pha             139708 non-null  object
4   H               138594 non-null  float64
5   G               119 non-null     float64
6   diameter        139708 non-null  float64
7   extent          19 non-null      object  8
albedo          138569 non-null  float64  9
rot_per         19264 non-null   float64  10
GM              15 non-null      float64
11  BV              1007 non-null    float64
12  UB              966 non-null     float64
13  IR              1 non-null       float64  14  spec_B          1414
    non-null    object
 15  spec_T          966 non-null     object
16  H_sigma         2631 non-null    float64
17  diameter_sigma  139583 non-null  float64
18  epoch           139708 non-null  float64
19   e              139708 non-null  float64
20   a              139708 non-null  float64
21   q              139708 non-null  float64
22   i              139708 non-null  float64
23   om             139708 non-null  float64
24   w              139708 non-null  float64
25   ma             139708 non-null  float64
```

```
26   ad              139708 non-null  float64
27   n               139708 non-null  float64
28   tp              139708 non-null  float64
29   per             139708 non-null  float64
30   per_y           139708 non-null  float64
31   moid            139708 non-null  float64
32   moid_jup        139708 non-null  float64  33  class
139708 non-null  object  34  data_arc       139683 non-null  float64
35  condition_code  139708 non-null  int64   36  rms
139708 non-null  float64 dtypes: float64(28), int64(1), object(8)
memory usage: 39.4+ MB df.describe()
```

```
                  H          G       diameter         albedo
rot_per  \
count  138594.000000  119.000000  139708.000000  138569.000000
19264.000000
mean       15.352342    0.179076       5.457189       0.130070
31.403305
std         1.419058    0.133822       9.305528       0.110338
84.314215
min         3.200000   -0.250000       0.002500       0.001000
0.008500
25%        14.620000    0.100000       2.762000       0.053000
4.813625
50%        15.440000    0.190000       3.948000       0.078000
8.263065
75%        16.230000    0.250000       5.730000       0.188000
19.612975
max        29.900000    0.600000     939.400000       1.000000
1880.000000
```

```
                GM           BV          UB    IR      H_sigma  ...
\ count  1.500000e+01  1007.000000  966.000000  1.00  2631.000000
...  mean   7.221682e+00     0.768836    0.364093 -0.33     0.305580
...  std    1.626088e+01     0.088303    0.095659   NaN     0.103657
...  min    2.100000e-09     0.580000    0.120000 -0.33     0.000000
...  25%    2.230161e-04     0.700000    0.289000 -0.33     0.240000
...
50%    4.910000e-01     0.743000    0.360000 -0.33     0.300000  ...
75%    6.000000e+00     0.849500    0.438750 -0.33     0.360000  ...
max    6.262840e+01     1.077000    0.655000 -0.33     0.810000  ...
```

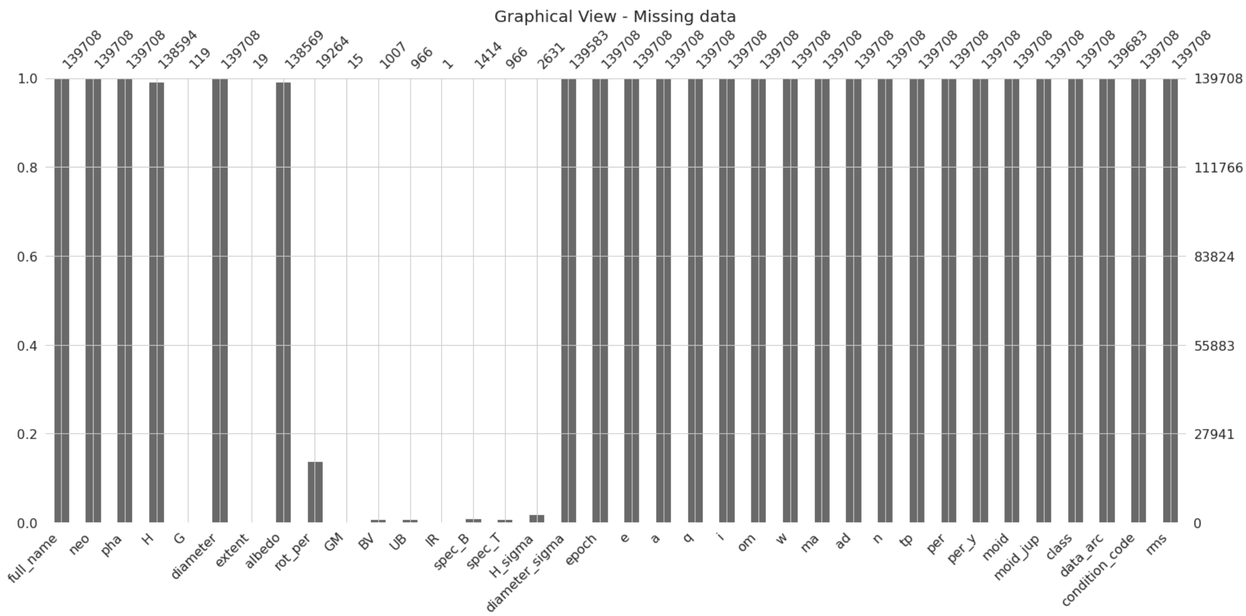
```
                 ad              n             tp            per  \
count  139708.000000  139708.000000  1.397080e+05  1.397080e+05
mean        3.246343       0.218882  2.459732e+06  1.791538e+03
std         2.888508       0.060228  6.642237e+02  9.304166e+03
min         1.000000       0.000126  2.426218e+06  1.810000e+02
25%         2.890000       0.180700  2.459349e+06  1.490000e+03
50%         3.190000       0.214900  2.459750e+06  1.680000e+03
75%         3.490000       0.242300  2.460178e+06  1.990000e+03
max       781.390000       1.989000  2.468617e+06  2.850000e+06
              per_y           moid       moid_jup       data_arc  \
count  139708.000000  139708.000000  139708.000000  139683.00000
mean        4.905024       1.419095       2.049259   10156.76786
std        25.478303       0.516136       0.478788    5945.92562
min         0.496000       0.000109       0.000416       1.00000
25%         4.070000       1.080000       1.800000    7297.00000
50%         4.590000       1.390000       2.070000    8550.00000
75%         5.450000       1.700000       2.350000   10729.00000
max      7810.000000      39.400000      35.600000   79466.00000
       condition_code            rms
count   139708.000000  139708.000000
mean         0.103122       0.519493
std          0.877247       0.061549
min          0.000000       0.055102
25%          0.000000       0.485240
50%          0.000000       0.519490
75%          0.000000       0.553640
max          9.000000       2.506000
[8 rows x 29 columns]
# comparing two features at random to take general overview on data variance
df[['a', 'e']].describe()
                   a              e
```

```
count  139708.000000  139708.000000
mean        2.823148       0.149425
std         1.518900       0.081682
min         0.626200       0.000600
25%         2.548000       0.091800
50%         2.760000       0.141800
75%         3.098000       0.194800
max       393.800000       0.984400
```

```
# here is the list of problem columns
cols_with_mixed_dtype=df.columns[[1,7,14,15]].values
cols_with_mixed_dtype
```

```
array(['name', 'extent', 'spec_B', 'spec_T'], dtype=object)
# looping through the problem columns and getting value counts for the
dtypes for i in cols_with_mixed_dtype:     print(f"Column name: {i}")
    print(df[i].apply(type).value_counts())
print()

Column name: name
<class 'float'>    123724
<class 'str'>       15984
Name: name, dtype: int64

Column name: extent
<class 'float'>    139689
<class 'str'>          19
Name: extent, dtype: int64

Column name: spec_B
<class 'float'>    138294
<class 'str'>        1414
Name: spec_B, dtype: int64

Column name: spec_T
<class 'float'>    138742
<class 'str'>         966
Name: spec_T, dtype: int64


#lets check if these column are any way affected by missing data
for i in cols_with_mixed_dtype:     print(f"Column name: {i}")
print(df[i].isnull().sum())     print()
Column    name:
name
123724
Column      name:
extent
139689
Column      name:
spec_B
138294
Column      name:
spec_T
138742
# check for duplicates
df[df.duplicated()].shape
```

```
(0, 37)
#creating a new dataframe called 'data' from the master dataframe 'df'
data = df.drop(['name'], axis=1)

# checking the shape def dataset_shape():     print("At this
stage the DataFrame 'data' looks like this")     x =
PrettyTable()
    x.field_names =['# rows', '# cols']
    x.add_row([data.shape[0], data.shape[1]])
print(x) dataset_shape()
At this stage the DataFrame 'data' looks like this
+--------+--------+
| # rows | # cols |
+--------+--------+
| 139708 |   36   |
+--------+--------+
data.columns

Index(['full_name', 'neo', 'pha', 'H', 'G', 'diameter', 'extent',
'albedo',
       'rot_per', 'GM', 'BV', 'UB', 'IR', 'spec_B', 'spec_T',
'H_sigma',
       'diameter_sigma', 'epoch', 'e', 'a', 'q', 'i', 'om', 'w', 'ma',
'ad',
       'n', 'tp', 'per', 'per_y', 'moid', 'moid_jup', 'class',
'data_arc',
       'condition_code', 'rms'],
dtype='object')

# we have this missingno library which we can use to graphically look
at the missing value status
# making use of python library missingno
msno.bar(data)
plt.title("Graphical View - Missing data", size=20)
plt.grid() plt.show()
```

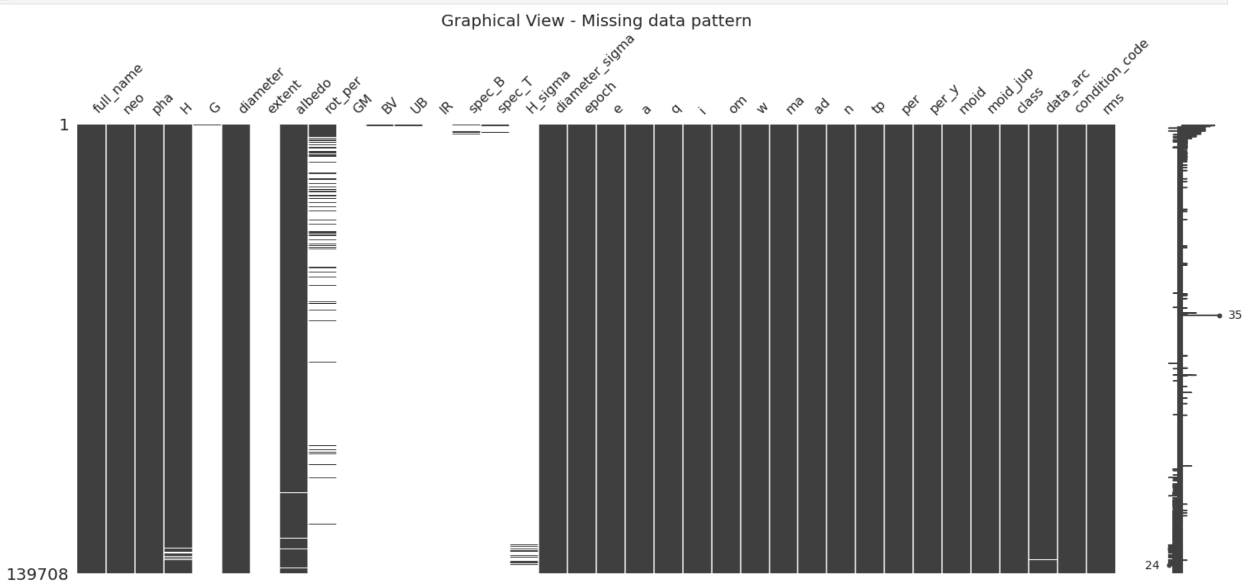


Observations:

- features IR, GM, extent, G, UB, spec_T, BV, spec_B, H_sigma, rot_per are missing in high percentage

Lets look we can find any missing patterns -

```
msno.matrix(data)
plt.title("Graphical View - Missing data pattern", size=20) plt.plot()

[]
```

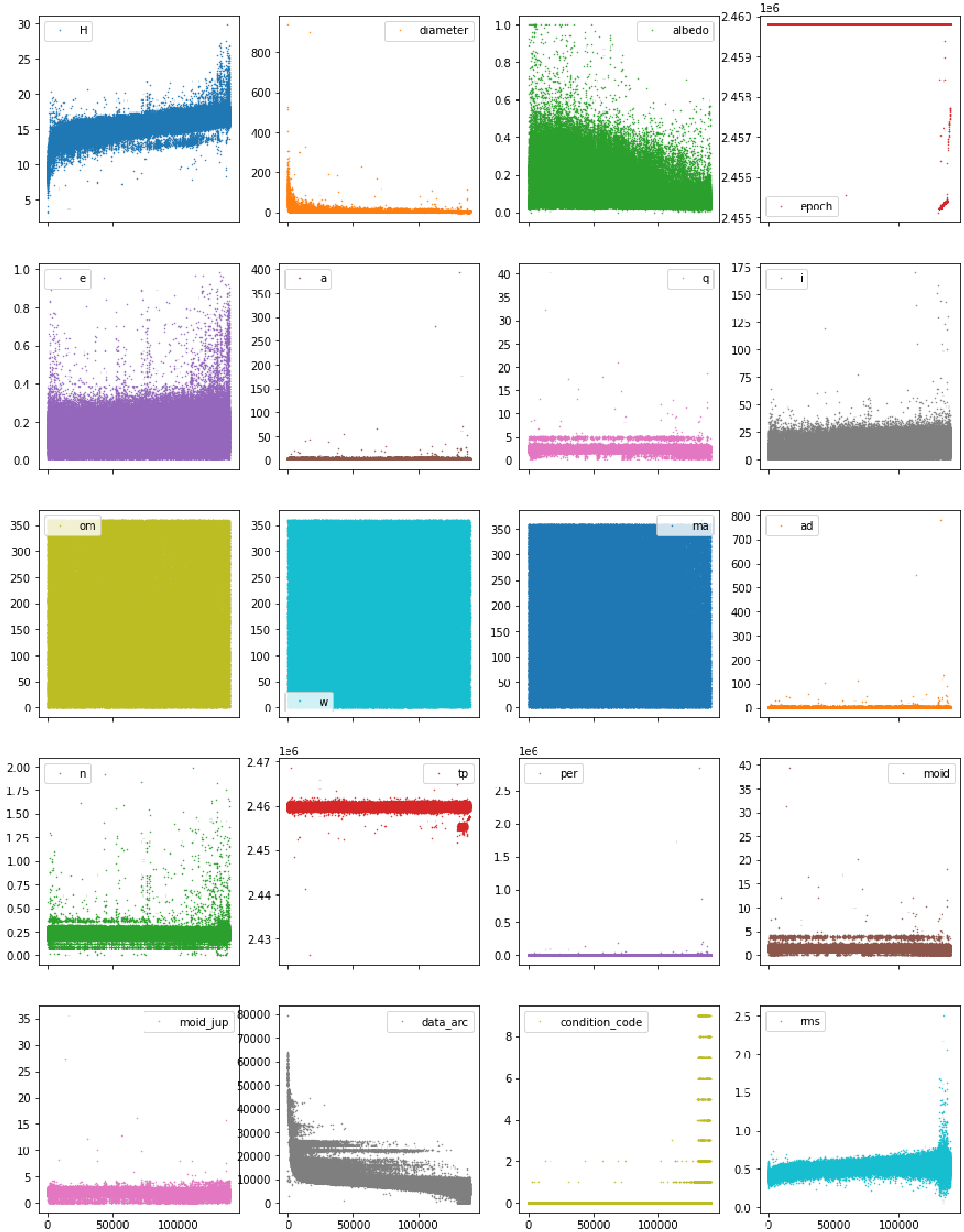


Univariate study -

Our focus here is to understand:

- How is the target variable distributed?
- How are the Independent Variables distributed?

```
data = df[['full_name', 'neo', 'pha', 'H', 'diameter', 'albedo',
'epoch', 'e', 'a', 'q', 'i', 'om', 'w', 'ma', 'ad', 'n', 'tp', 'per',
     'moid', 'moid_jup', 'class', 'data_arc', 'condition_code',
'rms']]

# plotting the data overview for every features
df.plot(lw=0, marker=".", subplots=True, layout=(-1, 4), figsize=(15,
20), markersize=1)
plt.show()
```

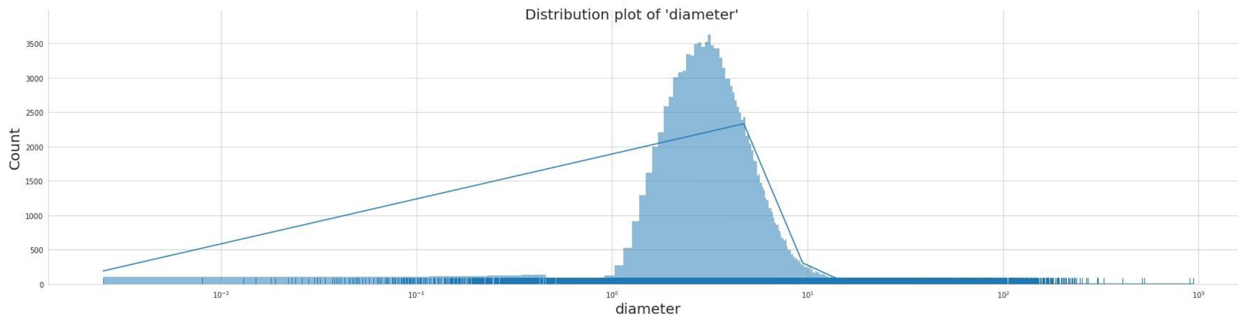



```
#defining a function to compute quantile stats
```

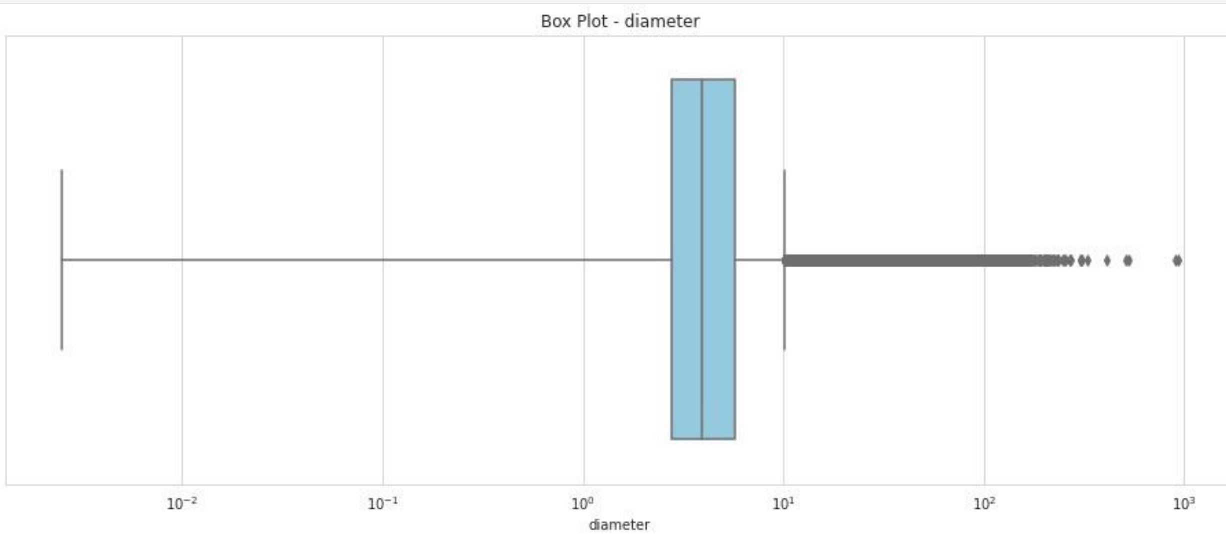
```
def quantile_stats(series):
    """ function to compute quantile statistics for series
return: list of two list, first list with statistic name, and
second list the value for the statistic parameter"""
min_value = series.min()
    quantile_25 = series.quantile(0.25)
quantile_50 = series.quantile(0.50)
quantile_75 = series.quantile(0.75)
max_value= series.max()
    range = round(max_value - min_value,3)
    IQR = round(scipy.stats.iqr(series, axis=0 , rng=(25, 75),
interpolation='lower'),3)

    return [['min_value', 'quantile_25', 'quantile_50', 'quantile_75',
'max_value', 'range', 'IQR'], [min_value, quantile_25, quantile_50,
quantile_75, max_value, range, IQR]]

#defining a function to compute descriptive  stats

def descriptive_stats(series):
    """ function to compute descriptive statistics for series
return: list of two list, first list with statistic name, and
second list the value for the statistic parameter"""
mean_value = round(np.mean(series),3)     median_value =
np.median(series)
    mod = scipy.stats.mstats.mode(series, axis=0)
mode_value= mod[0]     mode_count = mod[1]
    std_dev = round(np.std(series),3)
variance = round(np.var(series, axis=0),3)
    kurtosis = round(scipy.stats.kurtosis(series, axis=0,
fisher=True),3)
    skewness = round(scipy.stats.skew(series, axis=0),3)

    return [['mean', 'median', 'mode', 'std_dev', 'variance',
'kurtosis', 'skewness'], [mean_value, median_value, mode_value,
std_dev, variance, kurtosis, skewness]]

# defining a function to create a table display the quantile and
descriptive stats def print_stats_table(series):     """
creating statistics table """
    df1 = pd.DataFrame({'Quantile Stats':quantile_stats(data.diameter)
[0], 'Values':quantile_stats(data.diameter)[1]})
df2 = pd.DataFrame({'Descriptive
Stats':descriptive_stats(data.diameter)[0],
'Values':descriptive_stats(data.diameter)[1]})
return pd.concat([df1, df2], axis=1)
```

```
## plotting diameter distribution plot
```

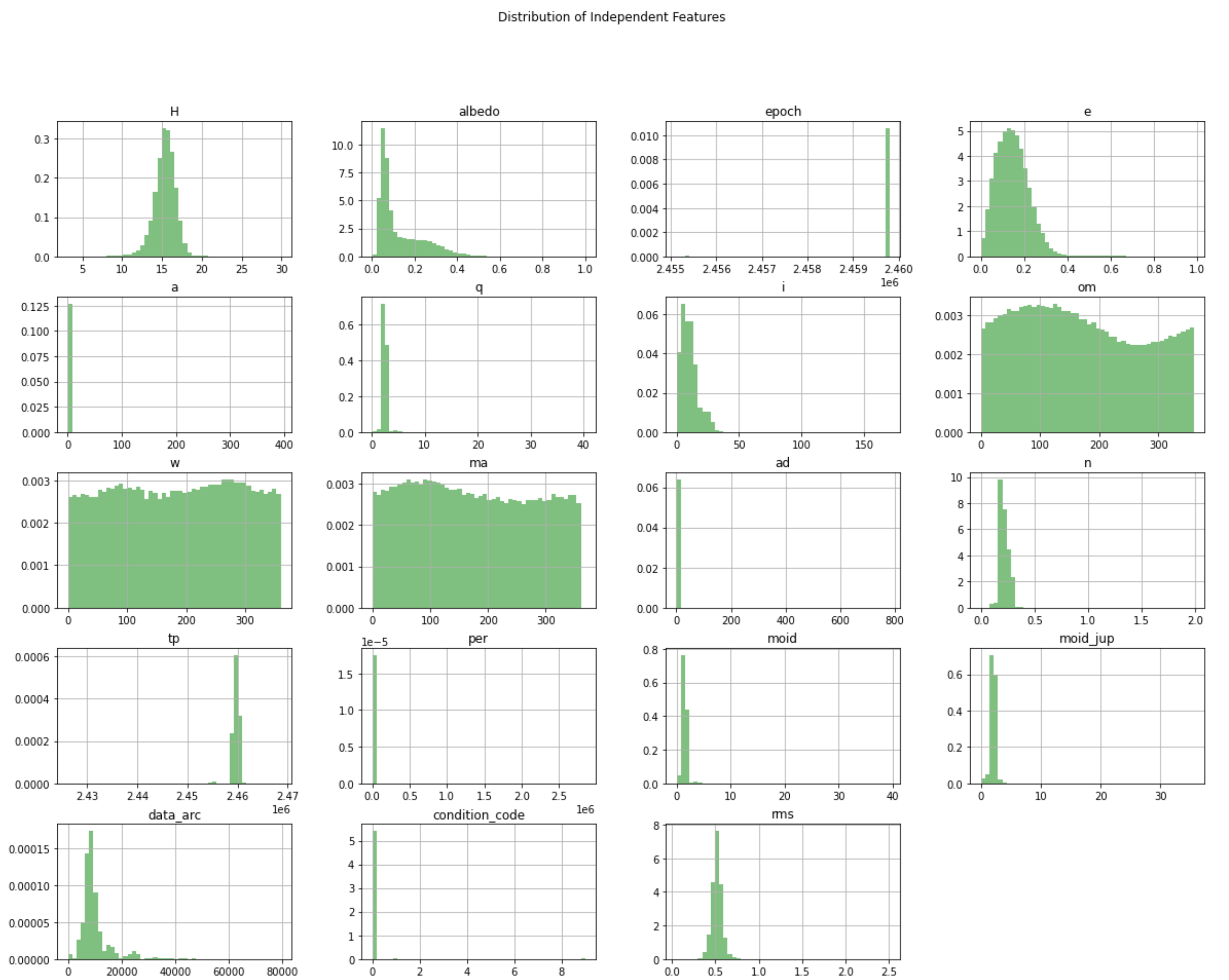
```
sns.set_style("whitegrid")
g = sns.displot(data=data, x='diameter', kde=True, rug=True, height=6,
aspect=4)
g.set(xscale='log')
g.set_xlabels(fontsize=20)
g.set_ylabels(fontsize=20)
g.fig.suptitle("Distribution plot of 'diameter'", fontsize=20)
plt.show()
```




```
##plotting box plot

fig, ax = plt.subplots(figsize=(16,6))
sns.boxplot(x="diameter", data=data, color='skyblue')
ax.set(xscale='log')
plt.title('Box Plot - diameter')
plt.show()
```




```
print_stats_table(data.diameter)
plt.suptitle("Distribution of Independent Features")
plt.plot() []
```

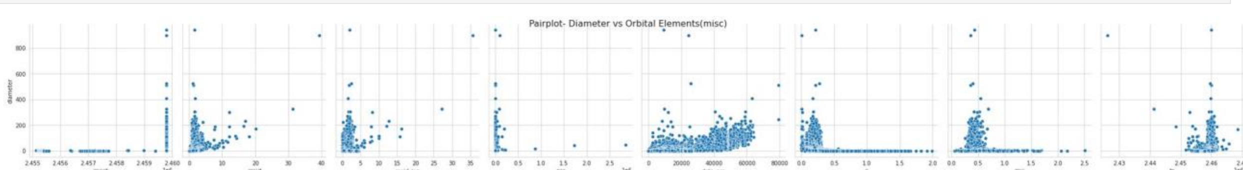



```
# plotting pie chart for 'pha' value=
data.pha.value_counts().values names =
['Not Hazardous', 'Hazardous']
fig = px.pie(data, values=value, names=names, title='Potentially
Hazardous Asteroids - Distribution')
fig.show()
```

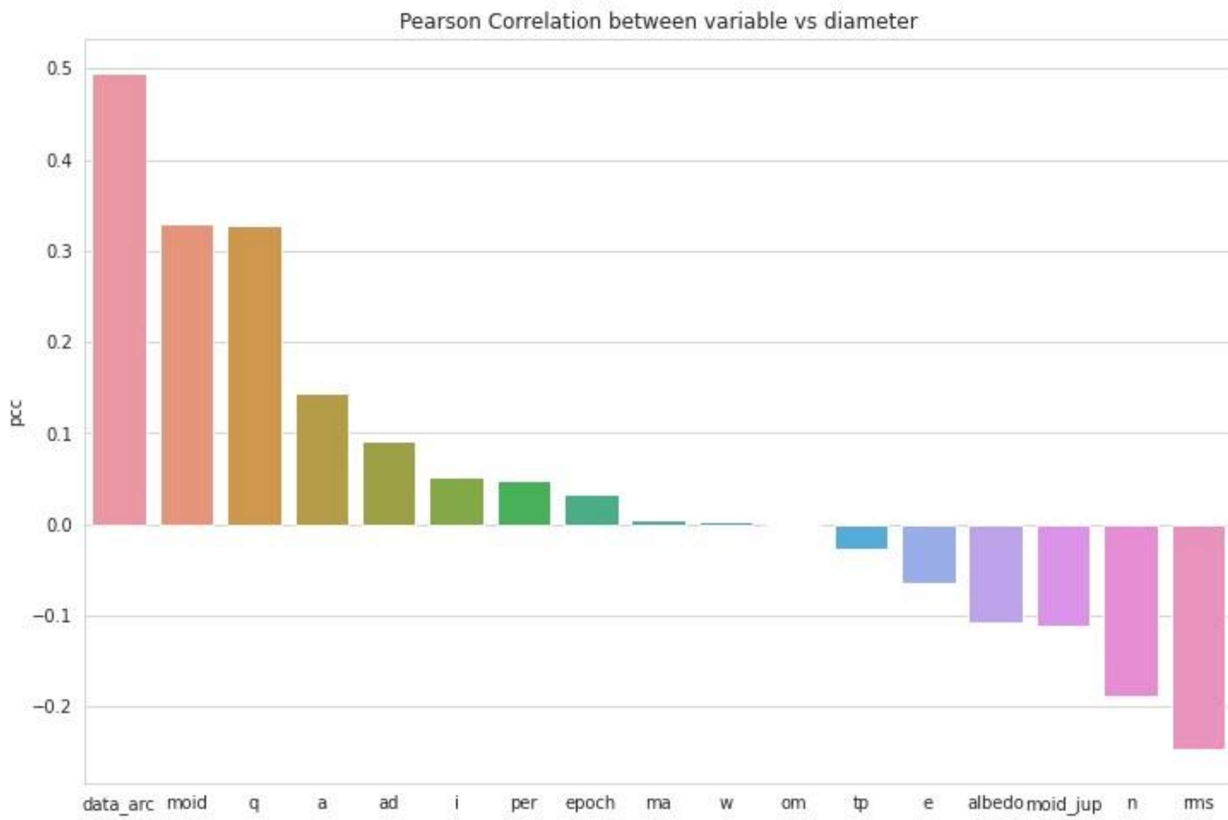
Observations:

- approx. 99% of asteroid are labeled and considered as not hazardous
- very small percent is considered as potentially hazardous

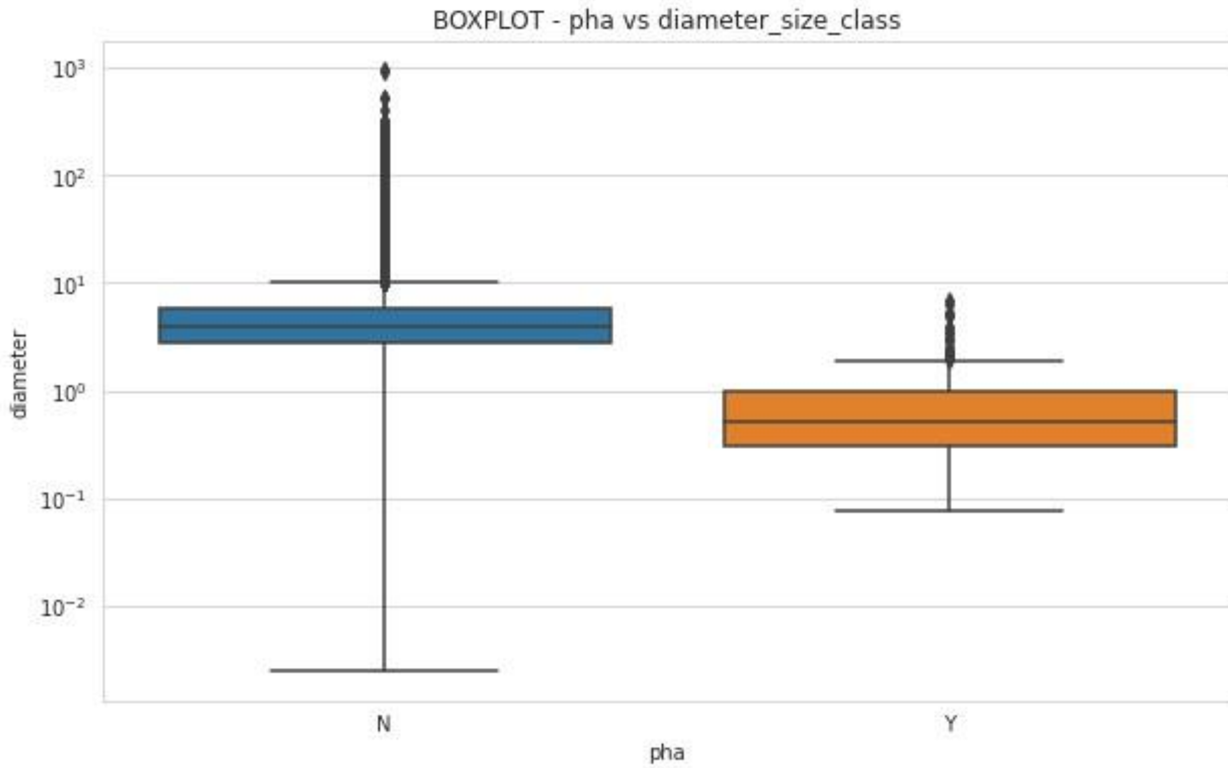
```
# plotting pie chart for 'neo'class
```



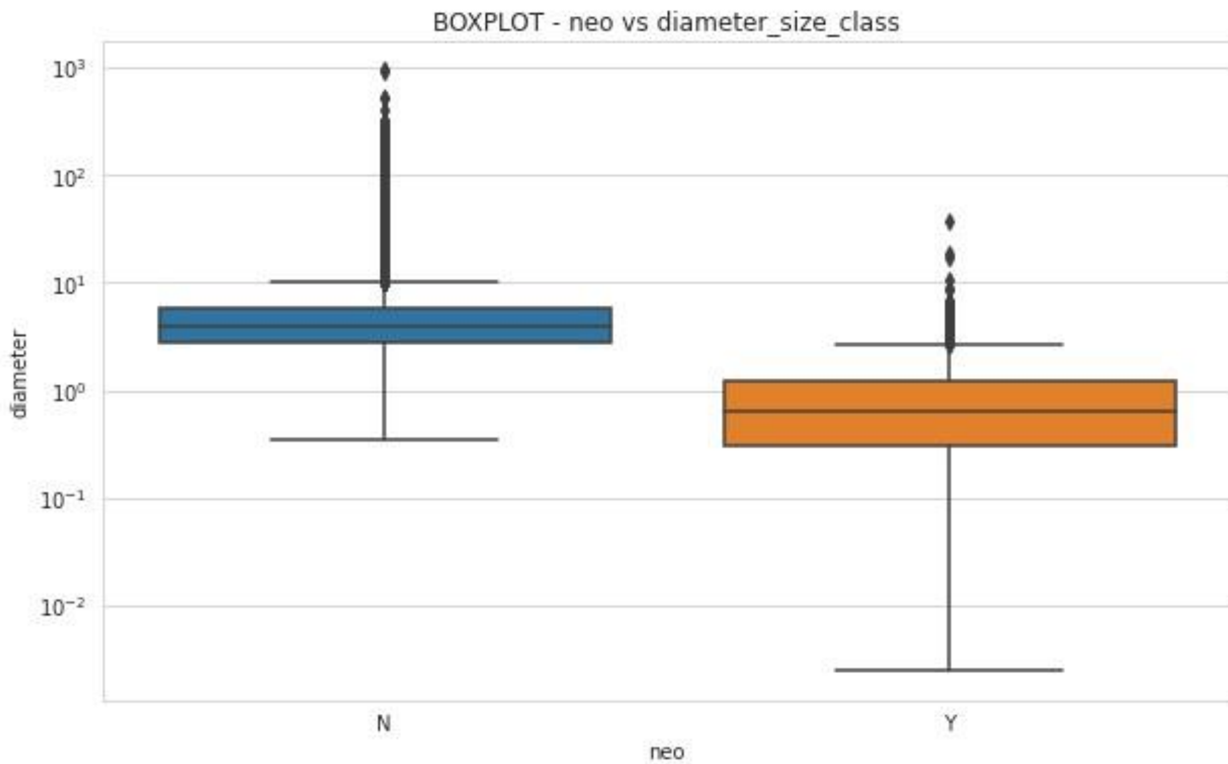
```
# pairwise correlation between continous variable and diameter

variables = data[['albedo', 'epoch', 'e', 'a', 'q', 'i', 'om', 'w',
'ma', 'ad', 'n', 'tp', 'per', 'moid', 'moid_jup', 'class', 'data_arc',
'rms']].copy()
corr_mat = variables.corrwith(data['diameter'])
corr_mat = pd.DataFrame(corr_mat, columns=['pcc']).sort_values('pcc',
ascending=False) corr_mat

## plotting the correlation plt.figure(figsize=(12,8))
sns.barplot(data=corr_mat, x=corr_mat.index, y='pcc')
plt.title('Pearson Correlation between variable vs diameter')
plt.show()
```




```
# plotting box plot for pha vs diameter
fig, ax = plt.subplots(figsize=(10,6))
sns.boxplot(data=data, x="pha", y="diameter", ax=ax)
ax.set(yscale='log')
ax.set_title("BOXPLOT - pha vs diameter")
plt.show()
```



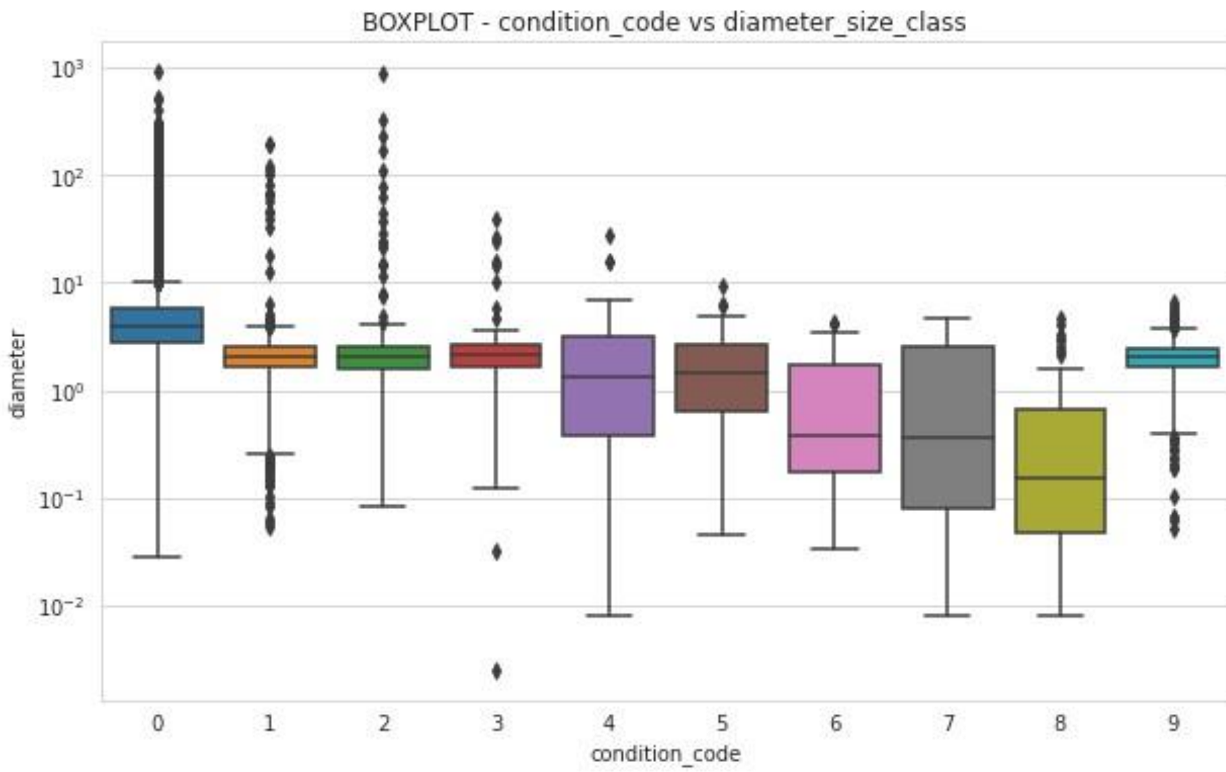

```
# plotting a count plot pha vs diameter size_class
fig, ax = plt.subplots(figsize=(10,6))
sns.countplot(data=data, x='pha', hue='size_class', ax=ax)
ax.set(yscale='log')
ax.set_title("pha vs diameter_size_class")
plt.show()
```



Observations:

- Near Earth asteroids mostly appears to be <= 1 km diameter
- Non NEO asteroids also are mostly approx. between 5-8 km diameter but the category also include bigger sized ones Insights:
- this feature could give some direction to diameter prediction

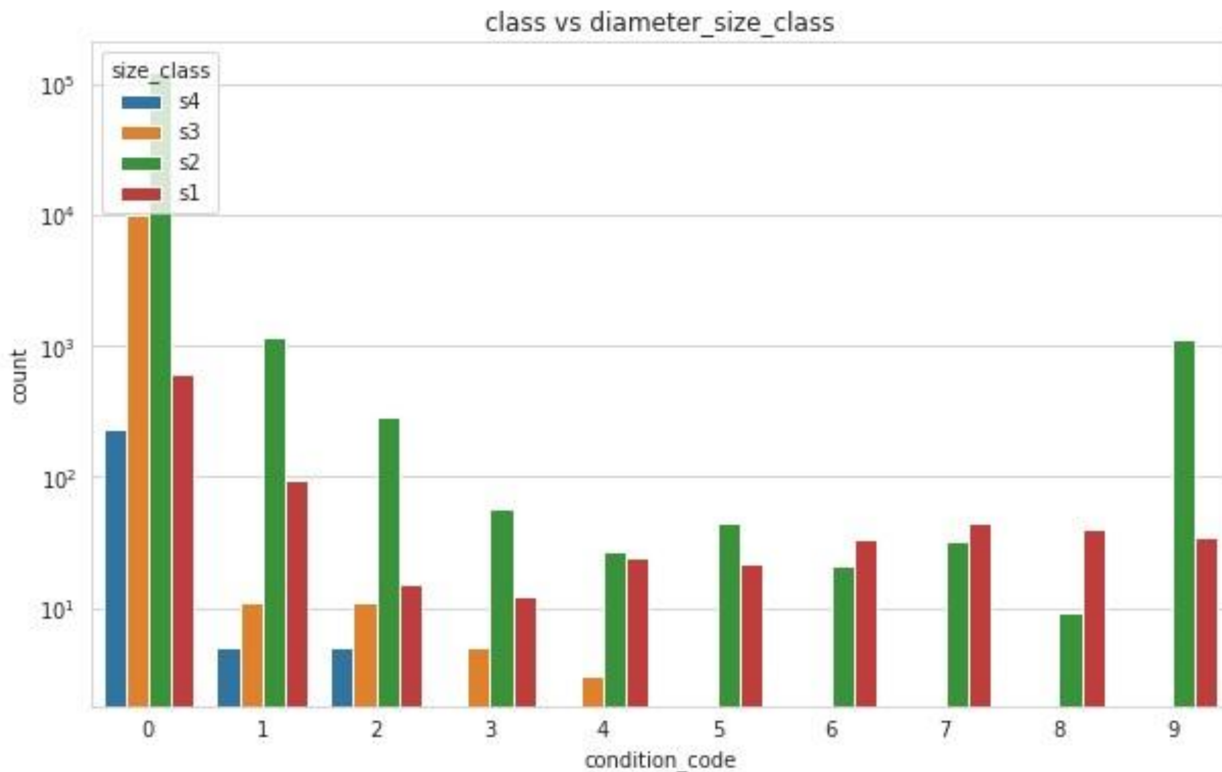
```
# plotting a count plot neo vs diameter size_class
fig, ax = plt.subplots(figsize=(10,6))
sns.countplot(data=data, x='neo', hue='size_class', ax=ax)
ax.set(yscale='log')
ax.set_title("neo vs diameter_size_class")
plt.show()
```



Observation:

- for asteroid with larger diameter, we observe the orbit uncertainity estimate is low and is good

```
# plotting a count plot condition_code vs diameter size_class
fig, ax = plt.subplots(figsize=(10,6))
sns.countplot(data=data, x='condition_code', hue='size_class', ax=ax)
ax.set(yscale='log')
ax.set_title("class vs diameter_size_class")
plt.show()
```



Observation:

- conditon_code 5-9 have s1 and s2 sized asteroids
- condition_code 3, 4 have s1, s2, s3
- condition_code 0, 1, 2 have s1, s2, s3, s4

```
# plotting a count plot condition_code vs diameter size_class
fig, ax = plt.subplots(figsize=(10,6))
sns.countplot(data=data, x='condition_code', hue='size_kbin', ax=ax)
ax.set(yscale='log')
ax.set_title("class vs diameter_size_class")
plt.show()
```

```
        print(f"degree of freedom: {degree}")

        #computing Cramer's V
        N= np.sum([table.iloc[i].values for i in range(len(table))])
score = np.sqrt(chi2/(N * min(table.shape[1]-1, table.shape[0]-
1)))
        print(f"Cramer's V is: {score}")

# compute chi-square, and Cramer's V score
chi_cramer(table_pha)
chi2: 30689.42721092655
p-value: 0.0
degree of freedom: 3
Cramer's V is: 0.4686879130081287
table_pha_kbin = pd.crosstab(columns=data.size_kbin, index=data.pha)
table_pha_kbin
size_kbin      0.0    1.0   2.0   3.0
ph
N           137055   2011   299     2
Y              341      0     0     0
# compute chi-square, and Cramer's V score
chi_cramer(table_pha_kbin)
chi2: 5.752139942803233
p-value: 0.1243119574434259
degree of freedom: 3
Cramer's V is: 0.006416586953545752
```

Observation:

- with p-value 0.12, we accept Null Hypothesis that feature pha and size_kbin are not correlated

1. NEO

```
data.neo.value_counts()
N    138460
Y      1248
Name: neo, dtype: int64
##creatinga contingency table using pandas crosstab
table_neo = pd.crosstab(columns=data.size_class, index=data.neo)
table_neo
size_class   s1      s2     s3    s4
ne
```

```
MCA            18      576      8     0
OMB             1     6417   1367    24
TJN             0      107   1756    17
TNO             0        2      5     5
# compute chi-square, and Cramer's V score
chi_cramer(table_class)

chi2: 115191.8488833281 p-
value: 0.0
degree of freedom: 30
Cramer's V is: 0.5242514126137287

##creatinga contingency table using pandas crosstab
table_class_kbins = pd.crosstab(columns=data.size_kbin,
index=data['class']) table_class_kbins
size_kbin      0.0    1.0   2.0   3.0
clas
AMO            345      1     0     0
APO            773      0     0     0
AST              9      1     0     0
ATE            129      0     0     0
CEN             20     22     8     0
IMB            497      0     0     0
MBA         126099   1273   227     1
MCA            599      3     0     0
OMB           7507    266    36     0
TJN           1414    442    24     0
TNO              4      3     4     1
# compute chi-square, and Cramer's V score
chi_cramer(table_class_kbins)
chi2: 14736.277052583173
p-value: 0.0
degree of freedom: 30
Cramer's V is: 0.18750921854130925
```

1. Condition code

```
##creatinga contingency table using pandas crosstab
table_code = pd.crosstab(columns=data.size_class,
index=data.condition_code) table_code
size_class        s1       s2     s3    s4
condition_cod
0                605   125856   9902   230
1                 93     1151     11     5
```

```
2                15      284     11     5
3                12       57      5     0
4                24       27      3     0
5                22       44      0     0
6                33       21      0     0
7                45       32      0     0
8                40        9      0     0
9                34     1132      0     0
# compute chi-square, and Cramer's V score
chi_cramer(table_code)
chi2: 16083.847547334812
p-value: 0.0
degree of freedom: 27
Cramer's V is: 0.19589516129680848
```

Observations:

- chi2 value is high indicating correlation
- p-value is 0, which is less than 0.05, hence we reject the Null hypothesis here, and accept the alternate hypothesis stating that there is a coorelation between 'condition_code' and diameter
- Cramer's V as 0.19589516129680848 depicts that the variable 'pha' and diameter are weakly associated

Inferences from the Hypothesis Tests:

- **NEO, PHA, and Orbit class Categorical variables are strongly associated with diameter**

Multivariate Analysis

1. Correlation between Continuous independent variables using Pearson's correlation coefficient

```
data.columns

Index(['full_name', 'neo', 'pha', 'H', 'diameter', 'albedo', 'epoch',
'e', 'a',
       'q', 'i', 'om', 'w', 'ma', 'ad', 'n', 'tp', 'per', 'moid',
'moid_jup',
       'class', 'data_arc', 'condition_code', 'rms', 'size_class'],
dtype='object')

#plotting heatmap for data correlation corr_matrix
=  data.drop('diameter', axis=1).corr()
plt.figure(figsize=(20, 8))
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
sns.heatmap(corr_matrix, vmax=0.8, annot=True, mask=mask, fmt='.2f',
cmap="PiYG")
plt.title("Asteroid data Heatmap")
plt.show()
```


```
data_experiment = data.copy()
data_experiment.dropna(inplace=True)
y1 = data_experiment.diameter     y2
= data_experiment.size_class
data_experiment.drop(['full_name', 'diameter', 'size_class'], axis=1,
inplace=True)

# need to encode catageorical variables
data_experiment['pha'] = data_experiment.pha.apply(lambda x: 0 if
x=='N' else 1)
data_experiment['neo'] = data_experiment.neo.apply(lambda x: 0 if
x=='N' else 1)

# encode catgeorical variable 'class' - this has got 11 categories
dummy = pd.get_dummies(data_experiment['class'], drop_first=True)

# concatenating the encoding dummy with original dataframe
data_experiment = pd.concat([data_experiment, dummy], axis=1)
data_experiment.drop('class', axis=1, inplace=True)
data_experiment.columns

Index(['neo', 'pha', 'H', 'albedo', 'epoch', 'e', 'a', 'q', 'i', 'om',
'w',
       'ma', 'ad', 'n', 'tp', 'per', 'moid', 'moid_jup', 'data_arc',
       'condition_code', 'rms', 'APO', 'AST', 'ATE', 'CEN', 'IMB',
'MBA',
       'MCA', 'OMB', 'TJN', 'TNO'],
dtype='object')

# computing variance_inflation_factor value for the independent
variables
```

```
from statsmodels.stats.outliers_influence import
variance_inflation_factor
vif = [variance_inflation_factor(data_experiment.values, i) for i in
range(data_experiment.shape[1])] vif_df = pd.DataFrame({'vif_value':
vif},
index=data_experiment.columns).sort_values('vif_value',
ascending=False) vif_df
                    vif_value
epoch            9.707444e+08
MBA              7.403759e+07
OMB              4.954272e+07
TJN              1.265855e+07
neo              8.127347e+06
MCA              3.976538e+06
IMB              3.048337e+06
a                1.069014e+06
ad               9.673691e+05
CEN              3.344619e+05
TNO              8.194071e+04
AST              5.461140e+04
q                3.164634e+04
moid             3.071890e+02
per              6.784970e+01
n                1.319430e+01
moid_jup         5.259566e+00
e                4.546555e+00
H                4.320281e+00
APO              4.311419e+00
ATE              3.652087e+00
data_arc         2.460106e+00
rms              1.916759e+00
condition_code   1.878699e+00
i                1.590785e+00
pha              1.581403e+00
albedo           1.563250e+00
tp               1.448299e+00
ma               1.324803e+00
om               1.021303e+00
w                1.018119e+00
## plotting the vif scores plt.figure(figsize=(12,8))
sns.barplot(data=vif_df, x=vif_df.index, y='vif_value')
plt.xscale('log')
plt.title('VIF - Independent Continuous variable')
```

```
plt.xticks(rotation=70)
plt.show()

# number of features having VIF score more than 10
print(f"Number of features with VIF score >10:
{vif_df[vif_df.vif_value>10].shape[0]}")
print(f"Number of features with VIF score >5:
{vif_df[vif_df.vif_value>5].shape[0]}") print(f"%
of features with VIF score >5:
{round(vif_df[vif_df.vif_value>5].shape[0]/vif_df.shape[0] * 100,2)}
%")

Number of features with VIF score >10: 16
Number of features with VIF score >5: 17
% of features with VIF score >5: 54.84%
```

Experimenting with removing 'epoch', feature with highest VIF value. Note epoch has negligible correlation with target diameter

```
# removing epoch from data and evaluating VIF again

data_experiment.drop('epoch', axis=1, inplace=True)
vif1 = [variance_inflation_factor(data_experiment.values, i) for i in
range(data_experiment.shape[1])]
vif_df1 = pd.DataFrame({'vif_value': vif1},
index=data_experiment.columns).sort_values('vif_value',
ascending=False)

## plotting the correlation
# fig = px.bar(vif_df1, x='vif_value', title='VIF - Independent
Continuous variable')
# fig.update_xaxes(type='log')
# fig.show()

# number of features having VIF score more than 10
print(f"Number of features with VIF score >10:
{vif_df1[vif_df1.vif_value>10].shape[0]}")
print(f"Number of features with VIF score >5:
{vif_df1[vif_df1.vif_value>5].shape[0]}")
print(f"% of features with VIF score >5:
{round(vif_df1[vif_df1.vif_value>5].shape[0]/vif_df1.shape[0] *
100,2)}%")

Number of features with VIF score >10: 19
Number of features with VIF score >5: 22
% of features with VIF score >5: 73.33%
```

Information Gain based evaluation of feature importance for predicting diameter

we will experiment with sklearn methods:

- sklearn.feature_selection.mutual_info_regression
- also we are making using of Binned Diameter varaiable as target and experimenting with sklearn.feature_selection.mutual_info_classif
- the above two methods returns estimated mutual information between each feature and the target

```
data_experiment = data.copy()
data_experiment.dropna(inplace=True)
y1 = data_experiment.diameter     y2
= data_experiment.size_class y3 =
data_experiment.size_kbin
data_experiment.drop(['full_name', 'diameter', 'size_class',
'size_kbin'], axis=1, inplace=True)

# need to encode catageorical variables
data_experiment['pha'] = data_experiment.pha.apply(lambda x: 0 if
x=='N' else 1)
data_experiment['neo'] = data_experiment.neo.apply(lambda x: 0 if
x=='N' else 1)

# encode catgeorical variable 'class' - this has got 11 categories
dummy = pd.get_dummies(data_experiment['class'], drop_first=True)

data_experiment = pd.concat([data_experiment, dummy], axis=1)

data_experiment.drop('class', axis=1, inplace=True) from

sklearn.feature_selection import mutual_info_regression

mutual_information = mutual_info_regression(data_experiment, y1)
mi_df = pd.DataFrame({'mi_value':mutual_information},
index=data_experiment.columns).sort_values('mi_value',
ascending=False)

plt.figure(figsize=(12,8))
sns.barplot(data=mi_df, x=mi_df.index, y='mi_value')
plt.title('Feature Importance- predicting diameter')
plt.xticks(rotation=70) plt.show()
```

Part II:

Handling the missing values by cleaning the dataset, and using imputation

```
df_null =
pd.DataFrame(np.round(100*df_asteroid.isnull().sum()/df_asteroid.shape
[0],4))
df_null
                         0
name               97.3840
a                   0.0002
e                   0.0000
i                   0.0000
om                  0.0000
w                   0.0000
q                   0.0000
ad                  0.0007
per_y               0.0001
data_arc            1.8428
condition_code      0.1032
n_obs_used          0.0000
H                   0.3202
neo                 0.0007
pha                 1.9580
diameter           83.6092
extent             99.9979
albedo             83.7553
rot_per            97.7616
GM                 99.9983
BV                 99.8784
UB                 99.8834
IR                 99.9999
spec_B             99.8016
spec_T             99.8833
G                  99.9858
moid                1.9580
class               0.0000
n                   0.0002
per                 0.0007
ma                  0.0010
df_asteroid['diameter']=pd.to_numeric(df_asteroid['diameter'],errors='
coerce')
dropindexes = df_asteroid['diameter']
[df_asteroid['diameter'].isnull()].index dropped_df
= df_asteroid.loc[dropindexes] df_asteroid =
df_asteroid.drop(dropindexes, axis=0)
```

```
More_Na =
df_asteroid.columns[df_asteroid.isna().sum()/df_asteroid.shape[0] >
0.5]
df_asteroid = df_asteroid.drop(More_Na, axis=1)
df_asteroid = df_asteroid.drop(['condition_code', 'neo', 'pha',
'albedo', 'H', 'class'],axis=1)
```

```
df_asteroid = df_asteroid.fillna(df_asteroid.mean())
```

```
df_asteroid.head()
```

```
          a         e          i          om           w         q
ad      per_y  data_arc  n_obs_used  diameter     moid         n
per           ma
0  2.769165  0.076009  10.594067   80.305532   73.597694  2.558684
2.979647  4.608202    8822.0        1002   939.400  1.59478  0.213885
1683.145708   77.372096
1  2.772466  0.230337  34.836234  173.080063  310.048857  2.133865
3.411067  4.616444   72318.0        8490   545.000  1.23324  0.213503
1686.155999   59.699133
2  2.669150  0.256942  12.988919  169.852760  248.138626  1.983332
3.354967  4.360814   72684.0        7104   246.596  1.03454  0.226019
1592.787285   34.925016
3  2.361418  0.088721   7.141771  103.810804  150.728541  2.151909
2.570926  3.628837   24288.0        9325   525.400  1.13948  0.271609
1325.432765   95.861936
4  2.574249  0.191095   5.366988  141.576605  358.687607  2.082324
3.066174  4.130323   63507.0        2916   106.699  1.09589  0.238632
1508.600458  282.366289
```

```
df_asteroid = df_asteroid.fillna(df_asteroid.mean())
df_asteroid.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
diameter      0
moid          0
n             0
per           0
ma            0
dtype: int64
```

```
df_asteroid['diameter'] = df_asteroid['diameter'].apply(np.log)

for column in df_asteroid.columns.drop(['diameter']):
df_asteroid['log('+column+')'] = df_asteroid[column].apply(np.log)
df_asteroid.corr()['diameter'].abs().sort_values(ascending=False)
diameter          1.000000
log(a)            0.563616
log(per_y)        0.563616
log(n)            0.563616
log(per)          0.563616
log(q)            0.543737
log(moid)         0.528689
n                 0.525392
q                 0.522404
moid              0.521095
data_arc          0.519390
n_obs_used        0.511250
log(ad)           0.477252
log(n_obs_used)   0.433656
log(data_arc)     0.298793
a                 0.195634
e                 0.185047
log(e)            0.157921
ad                0.112606
i                 0.096037
log(i)            0.088749
per_y             0.046649
per               0.046649
ma                0.030946
log(ma)           0.023154
log(w)            0.006008
w                 0.005310
om                0.001478
log(om)           0.000169

Name: diameter, dtype: float64
```

Part III:

Splitting the dataframe into train and test dataframes and normalizing them for our regressions.

```
from sklearn.model_selection import train_test_split

predictors = df_asteroid.drop('diameter',axis=1) target =
df_asteroid['diameter'] X_train,X_test,Y_train,Y_test =
```

```
train_test_split(predictors,target,test_size=0.20,random_state=0)

X_train.head()

               a         e          i          om           w
q        ad      per_y  data_arc  n_obs_used     moid          n
per           ma     log(a)     log(e)    log(i)   log(om)    log(w)
log(q)   log(ad)  log(per_y)  log(data_arc)  log(n_obs_used)
log(moid)    log(n)  log(per)   log(ma)
474961  3.148871  0.130545  14.123745   20.870476  335.017941
2.737801  3.559942  5.587796    5812.0          72  1.74091  0.176389
2040.942568  171.197625  1.147044 -2.036035  2.647857  3.038336
5.814184  1.007155  1.269744    1.720585       8.667680
4.276666   0.554408 -1.735063  7.621167  5.142819
283914  3.104229  0.169382  19.711359  107.671134  182.187629
2.578429  3.630030  5.469390    6824.0         186  1.56154  0.180208
1997.694859  177.806839  1.132766 -1.775599  2.981195  4.679082
5.205037  0.947180  1.289241    1.699167       8.828201
5.225747   0.445673 -1.713645  7.599749  5.180698
241049  3.170379  0.104378   1.607302  151.293279  138.910748
2.839460  3.501298  5.645143    6684.0         272  1.82279  0.174597
2061.888649  153.747173  1.153851 -2.259734  0.474557  5.019220
4.933832  1.043614  1.253134    1.730796       8.807472
5.605802   0.600368 -1.745274  7.631378  5.035310
359366  3.123361  0.232180  13.565848   54.472085  258.745939
2.398180  3.848542  5.520031    7120.0         185  1.42886  0.178554
2016.191342  323.992520  1.138910 -1.460244  2.607555  3.997688
5.555847  0.874710  1.347694    1.708383       8.870663
5.220356   0.356877 -1.722862  7.608966  5.780720
110551  2.646488  0.191386  13.100536   39.183682  344.363064
2.139987  3.152990  4.305397    6660.0         711  1.14667  0.228928
1572.546205   77.728902  0.973234 -1.653462  2.572653  3.668260
5.841697  0.760800  1.148351    1.459869       8.803875
6.566672   0.136862 -1.474347  7.360451  4.353227 from
sklearn import preprocessing

#Input standard normalization: std_scaler =
preprocessing.StandardScaler().fit(X_train)

def scaler(X):     x_norm_arr=
std_scaler.fit_transform(X)
    return pd.DataFrame(x_norm_arr, columns=X.columns, index =
X.index)

X_train_norm = scaler(X_train)
X_test_norm = scaler(X_test)
```

```
def inverse_scaler(X):     x_norm_arr=
std_scaler.inverse_transform(X)
    return pd.DataFrame(x_norm_arr, columns=X.columns, index =
X.index)
```

##Part IV:

Trying different regressions and ranking them according to their R^2.

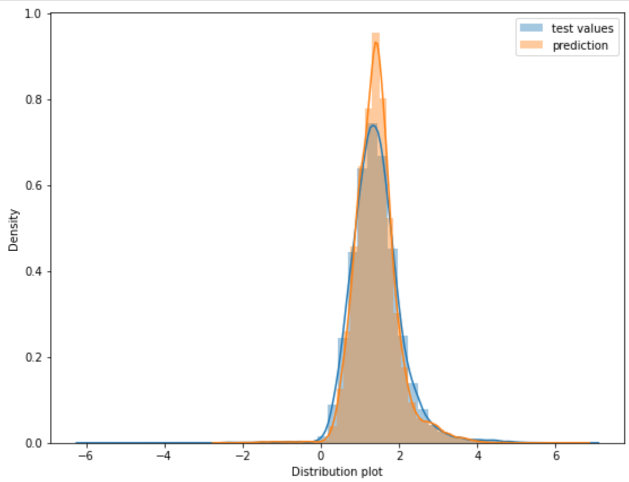
Algorithms used:

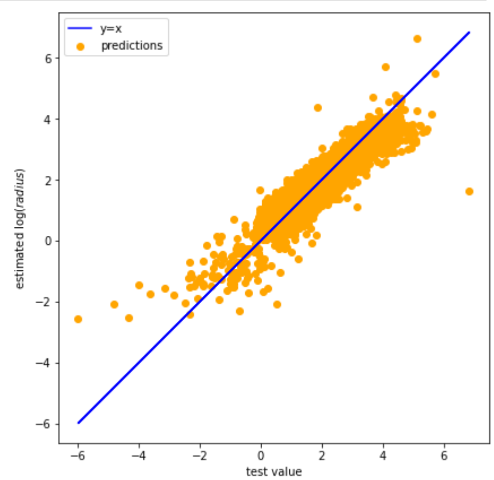
- Linear Regression
- Elastic Net
- k-Nearest Neighbours
- Decision Tree
- Random Forest
- SVM
- Neural Network • XGBoost.

```
from sklearn.metrics import r2_score
import seaborn as sns

def plot(prediction):     fig, (ax1, ax2) = plt.subplots(1,
2,figsize=(20,7))
sns.distplot(Y_test.values,label='test values', ax=ax1)
sns.distplot(prediction ,label='prediction', ax=ax1)
ax1.set_xlabel('Distribution plot')
    ax2.scatter(Y_test,prediction, c='orange',label='predictions')
ax2.plot(Y_test,Y_test,c='blue',label='y=x')
ax2.set_xlabel('test value')
    ax2.set_ylabel('estimated $\log(radius)$')
ax1.legend()     ax2.legend()
    ax2.axis('scaled') #same x y scale def score(prediction):
score = r2_score(prediction,Y_test)     return score def
announce(score):     print('The R^2 score achieved using this
regression is:', round(score,3)) algorithms = [] scores = []
```

Linear Regression

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train,Y_train)
Y_pred_lr = lr.predict(X_test)
score_lr = score(Y_pred_lr)
announce(score_lr)
algorithms.append('LR')
scores.append(score_lr)
The R^2 score achieved using this regression is: 0.67
plot(Y_pred_lr)
```





Feature Engineering

Feature Selection based on EDA studies

```
# original dataset shape
df.shape
(139708, 37)
# creating a new dataframe from master
dfe = df.copy()
```

- drop features - as inferred from the Missing Value review(>80% data insuffieciency), and other reasons like data irrelevance
- drop 'w', 'ma', 'om' as understood from EDA feature correlation and feature importance studies
- Top most important features - albedo, H, data_arc, 'ad', 'a', 'q', 'e', moid. Orbit class, neo, pha too have strong corrlation with diameter

```
# dataframe dfe is a fresh copy from master df,
# removing features owing to feature irrelevance and missing value
analysis (missing>80% data)
dfe.drop(['full_name','name', 'extent', 'rot_per', 'G', 'GM', 'BV',
'UB', 'IR', 'spec_B', 'spec_T',
       'H_sigma', 'diameter_sigma', 'per_y'], axis=1, inplace=True)

# dropping features as inferred as irrelevant from EDA
dfe.drop([ 'w', 'ma', 'om'], axis=1, inplace=True)
```

Below is our to_drop list for our dataset

```
to_drop_columns = ['full_name','name', 'extent', 'rot_per', 'G', 'GM',
'BV', 'UB', 'IR', 'spec_B', 'spec_T',
       'H_sigma', 'diameter_sigma', 'per_y', 'w', 'ma', 'om']

# columns in the dataset
dfe.columns

Index(['neo', 'pha', 'H', 'diameter', 'albedo', 'epoch', 'e', 'a',
'q', 'i',
       'ad', 'n', 'tp', 'per', 'moid', 'moid_jup', 'class',
'data_arc',
       'condition_code', 'rms'],
dtype='object')

# checking shape of dataset dfe
dfe.shape
```

```
    Found existing installation: lightgbm 2.2.3
    Uninstalling lightgbm-2.2.3:
      Successfully uninstalled lightgbm-2.2.3
  Attempting uninstall: holidays
    Found existing installation: holidays 0.10.5.2
    Uninstalling holidays-0.10.5.2:
      Successfully uninstalled holidays-0.10.5.2
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have
imgaug 0.2.9 which is incompatible.
```

```
Successfully installed Mako-1.2.1 alembic-1.8.1 autopage-0.5.1
category-encoders-2.4.0 cliff-3.10.1 cmaes-0.8.2 cmd2-2.4.2
colorlog6.6.0 gast-0.4.0 holidays-0.11.3.1 keras-2.7.0 lightgbm-3.3.0
optuna2.10.0 pbr-5.9.0 plotly-5.3.1 pyperclip-1.8.2 python-dateutil-
2.8.1 scikit-learn-1.0.1 stevedore-3.5.0 tensorflow-
2.7.0+zzzcolab20220506150900 tensorflow-estimator-2.7.0 verstack-3.2.3
```

{"pip_warning":{"packages":
["_plotly_utils","dateutil","plotly","sklearn"]}}

```
#!pip install protobuf==3.20.*
```

```
import verstack
```

```
verstack.__version__
```

{"type":"string"}

```
from verstack.stratified_continuous_split import scsplit
X_train, X_test, y_train, y_test = scsplit(X, y, stratify=y,
random_state=0) #test_size default=0.3
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((97795, 19), (41913, 19), (97795,), (41913,))
```

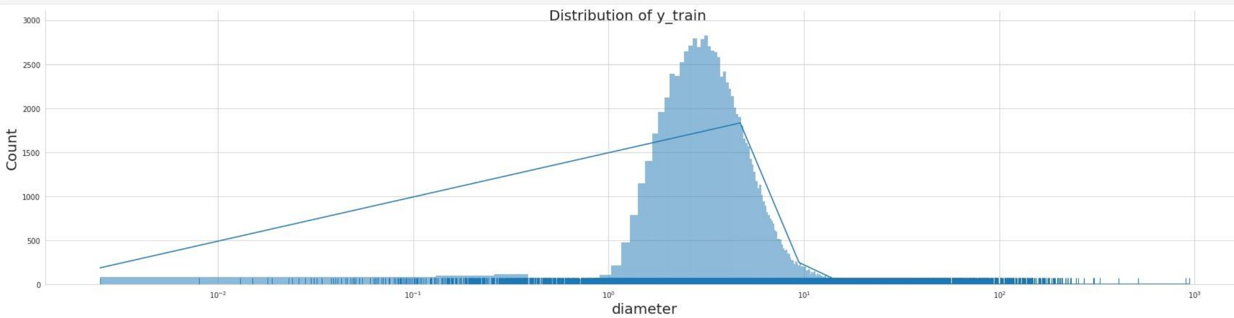
```
type(X_train), type(X_test), type(y_train), type(y_test)
```

```
(pandas.core.frame.DataFrame,
pandas.core.frame.DataFrame,
pandas.core.series.Series,
pandas.core.series.Series)
```

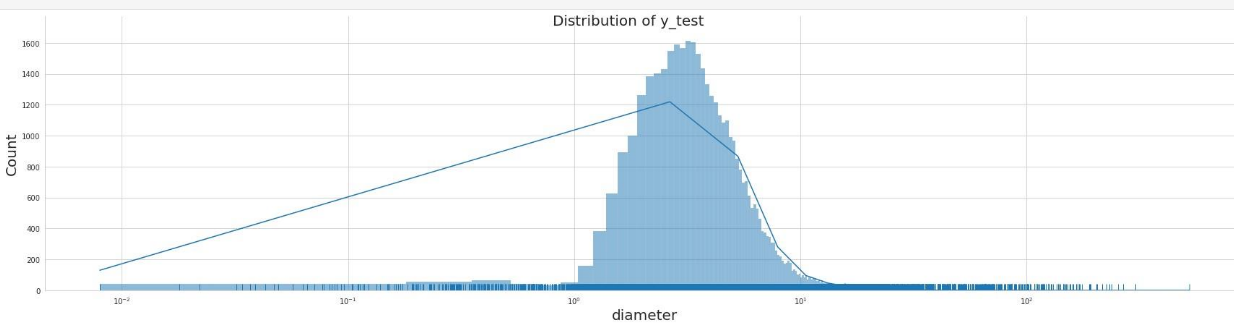
```
#save files
save_to_csv(X_train, 'X_train')
save_to_csv(X_test, 'X_test')
save_to_csv(y_train, 'y_train')
save_to_csv(y_test, 'y_test')
```

```
y_train =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/y_train',
```

```
squeeze=True)
y_test =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/y_test',
squeeze=True)

y_train.shape, y_test.shape
((97795,), (41913,))
#plotting y_train distribution
sns.set_style("whitegrid")
g = sns.displot(x=y_train, kde=True, rug=True, height=6, aspect=4)
g.set(xscale='log')
g.set_xlabels(fontsize=20)
g.set_ylabels(fontsize=20)
g.fig.suptitle("Distribution of y_train", fontsize=20)
plt.show()
```




```
# plotting y_test distribution
sns.set_style("whitegrid")
g = sns.displot(x=y_test, kde=True, rug=True, height=6, aspect=4)
g.set(xscale='log')
g.set_xlabels(fontsize=20)
g.set_ylabels(fontsize=20)
g.fig.suptitle("Distribution of y_test", fontsize=20)
plt.show()
```



Observations:

- distribution of y_train and y_test looks similar.

Categorical Variables Encoding - one hot encoding

We have three categorical variables:

- pha: 2 categories (Y/N)
- neo: 2 categories (Y/N)
- class: 11 categories of orbit classes

All three categories are nominal, so we will go with one hot encoding

```
# improting library
from sklearn.preprocessing import OneHotEncoder

X_train.dtypes
neo                   object
pha                   object
H                    float64
albedo               float64
epoch                float64
e                    float64
a                    float64
q                    float64
i                    float64
ad                   float64
n                    float64
tp                   float64
per                  float64
moid                 float64
moid_jup             float64
class                 object
data_arc             float64
condition_code         int64
rms                  float64

dtype: object
# getting hold of the categorical columns
cat_cols  =  X_train.dtypes[X_train.dtypes=='object'].index  num_cols  =
X_train.dtypes[(X_train.dtypes=='float64')  |  (X_train.dtypes=='int64')
].index

X_train[cat_cols].head()
       neo pha class
51219    N   N   MBA
108225   N   N   MBA
99696    N   N   MBA
27539    N   N   MBA
47816    N   N   MBA
```

```
# applying one hot encoding ohe =
OneHotEncoder(drop='first')
ohe.fit(X_train[cat_cols])
X_train_cat_cols = ohe.transform(X_train[cat_cols]).toarray()
X_test_cat_cols = ohe.transform(X_test[cat_cols]).toarray()
ohe.get_feature_names_out()
array(['neo_Y', 'pha_Y', 'class_APO', 'class_AST', 'class_ATE',
       'class_CEN', 'class_IMB', 'class_MBA', 'class_MCA'
'class_OMB',
       'class_TJN', 'class_TNO'], dtype=object)
# with open("encoder", "wb") as f:
#     pickle.dump(one_hot, f)
```

We need to concatenate encoded Categorical columns with the numerical set

```
X_train_encoded = np.hstack((X_train[num_cols].values,
X_train_cat_cols))
X_test_encoded = np.hstack((X_test[num_cols].values, X_test_cat_cols))

X_train_encoded.shape, X_test_encoded.shape
((97795, 28), (41913, 28))
# creating dataframes of these outputs
ncols = list(X_train[num_cols].columns)
ccols = list(ohe.get_feature_names_out())
ncols.extend(ccols) print(ncols)

['H', 'albedo', 'epoch', 'e', 'a', 'q', 'i', 'ad', 'n', 'tp', 'per',
'moid', 'moid_jup', 'data_arc', 'condition_code', 'rms', 'neo_Y',
'pha_Y', 'class_APO', 'class_AST', 'class_ATE', 'class_CEN',
'class_IMB', 'class_MBA', 'class_MCA', 'class_OMB', 'class_TJN',
'class_TNO']

X_train_encoded_df = pd.DataFrame(X_train_encoded, columns=ncols)
X_test_encoded_df = pd.DataFrame(X_test_encoded, columns=ncols)

X_train_encoded_df.head()
```

	H	albedo	epoch	e	a	q	i	ad	n
\ 0	14.22	0.066	2459800.5	0.0710	3.174	2.948	8.62	3.40	0.1743
1	16.40	0.115	2459800.5	0.1600	2.939	2.468	15.92	3.41	0.1957
2	16.56	0.094	2459800.5	0.0716	2.451	2.275	11.20	2.63	0.2569

```
3  15.21   0.243  2459800.5  0.0630  2.703  2.533   6.69  2.87  0.2218
4  14.35   0.063  2459800.5  0.1566  2.789  2.352   8.83  3.23  0.2117
           tp  ...  class_APO  class_AST  class_ATE  class_CEN
class_IMB  \
0  2460678.06  ...        0.0        0.0        0.0        0.0
0.0
1  2459055.14  ...        0.0        0.0        0.0        0.0
0.0
2  2459345.55  ...        0.0        0.0        0.0        0.0
0.0
3  2459025.12  ...        0.0        0.0        0.0        0.0
0.0
4  2459793.17  ...        0.0        0.0        0.0        0.0
0.0
   class_MBA  class_MCA  class_OMB  class_TJN  class_TNO
0        1.0        0.0        0.0        0.0        0.0
1        1.0        0.0        0.0        0.0        0.0
2        1.0        0.0        0.0        0.0        0.0
3        1.0        0.0        0.0        0.0        0.0
4        1.0        0.0        0.0        0.0        0.0
[5 rows x 28 columns]
```

Correlation study with encoded independent features

Now that we have have X_train scaled and categoricaly encoded, lets have a look at the correlation between these transformed independent features

```
#plotting heatmap for data correlation
corr_matrix =  X_train_encoded_df.corr()
plt.figure(figsize=(20, 8))
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
sns.heatmap(corr_matrix, vmax=0.8, annot=True, mask=mask, fmt='.2f',
cmap="PiYG")
plt.title("X_train heatmap")
plt.show()
```

Missing value Imputation

```
X_train_encoded_df.isnull().sum()
[X_train_encoded_df.isnull().sum().values>0]
H             771
albedo        754
data_arc       16
dtype: int64
X_train_encoded_df.isnull().sum()
[X_train_encoded_df.isnull().sum().values>0]
H             771
albedo        754
data_arc       16
dtype: int64
```

Discussion on coming up with a strategy to impute missing values in albedo, H, data-arc

1. '**albedo**' - from our EDA, we know overall we have 0.8153% of missing data for the feature

```
#let's look at X_train albedo statistics
print(f"'albedo' mean: {round(X_train_encoded_df.albedo.mean(), 2)}")
print(f"'albedo' median: {X_train_encoded_df.albedo.median()}")
albedo_stats_before_imputation =
X_train_encoded_df[['albedo']].describe()
'albedo' mean: 0.13
'albedo' median: 0.078
```

- Domain-specific information:
  - Astronomical albedos are usually given by the IRAS minor planet survey[1] or the MSX minor planet survey[2] (available at the PDS). These are geometric albedos. If there is no IRAS/MSX data a **rough average of 0.1** can be used. (Source: Source: https://en.wikipedia.org/wiki/Standard_asteroid_physical_characteristics#Albedo)
  - Only a coarse estimation of size can be found from the object's magnitude because an assumption must be made for its albedo which is also not usually known for certain. The NASA near-Earth object program uses an **assumed albedo of 0.14** for this purpose.(Source: https://en.wikipedia.org/wiki/Potentially_hazardous_object#Size)
  - commonly assumed albedo range for asteroid is between 0.05 - 0.25
    can be estimated from its absolute magnitude H and an assumed geometric albedo

- CNEOS, NASA- Asteroid Size Estimator - https://cneos.jpl.nasa.gov/tools/ast_size_est.html

- https://www.boulder.swri.edu/clark/jssrpb04.pdf
- Data Statistics based info- 'H' feature mean and median value `'H' mean: 15.35 'H' median: 15.44` ---

Imputation strategy:

- Since 'diameter' is the target variable, we cannot apply the domain info of computing 'H' given diameter and albedo
- We had the observed this in CDF plots:
  - 80% of s1 is approx. <=22 absolute magnitude – 80% of s2 is approx. <=17 absolute magnitude
  - 80% of s3 is approx. <=14 absolute magnitude
  - 80% of s4 is approx. <=8 absolute magnitude
- mean and median values are closer
- We can try imputing with mean value and see how it works out

3. **data_arc** from our EDA, we know overall we have 0.0179% of missing data for the feature

data-arc span is number of days spanned by the data-arc

```
print(f"data-arc span mean: {round(X_train.data_arc.mean(),2)} days")
print(f"data-arc span median: {round(X_train.data_arc.median(),2)}
days")
print(f"data-arc span mean(yr): {round(X_train.data_arc.mean()/365,2)}
years") print(f"data-arc span median(yr):
{round(X_train.data_arc.median()/365,2)} years")

data-arc span mean: 10152.49 days data-
arc span median: 8549.0 days data-arc
span mean(yr): 27.82 years data-arc span
median(yr): 23.42 years

print(f"data_arc mean for near earth objects(neo):
{X_train[X_train.neo=='Y'].data_arc.mean()}")
print(f"data_arc median for near earth objects(neo):
{X_train[X_train.neo=='Y'].data_arc.median()}")
print(f"data_arc min for near earth objects(neo):
{X_train[X_train.neo=='Y'].data_arc.min()}")
print(f"data_arc max for near earth objects(neo):
{X_train[X_train.neo=='Y'].data_arc.max()}")
```

```
data_arc mean for near earth objects(neo): 6566.335585585585
data_arc median for near earth objects(neo): 5164.5
data_arc min for near earth objects(neo): 1.0
data_arc max for near earth objects(neo): 46582.0
print(f"data_arc mean for not-neo:
{X_train[X_train.neo=='N'].data_arc.mean()}")
print(f"data_arc median for not-neo:
{X_train[X_train.neo=='N'].data_arc.median()}")
print(f"data_arc min for not-neo:
{X_train[X_train.neo=='N'].data_arc.min()}")
print(f"data_arc max for not-neo:
{X_train[X_train.neo=='N'].data_arc.max()}")
data_arc mean for not-neo: 10185.351993477207
data_arc median for not-neo: 8556.0
data_arc min for not-neo: 1.0
data_arc max for not-neo: 79466.0
```

Exploring the data-arc domain information, could not get enough evidence if we can consider the mean/median as nominal value to substitute missing values. **Imputation Strategy:**

- Mean and median is not close when looking at whole train data
- Mean and median are close for only neo data
- While these value are again apart for non-neo data, which is expected as the variance is larger in this set
- Median seems more reasonable here, lets try with median Imputation

  Implementation :

- **Mean Imputation** for albedo and H • **Median Imputation** for data-arc

note - try linear regression instead of KNN

Try Simple Imputer on all three and check accuracy try linear regression base imputing and check accuracy

```
## checking albedo stats
albedo_before_imp = X_train_encoded_df[['albedo']].describe()
H_before_imp = X_train_encoded_df[['H']].describe()

X_train_imp = X_train_encoded_df.copy()
X_test_imp  = X_test_encoded_df.copy()

# appling mean imputation
# Importing library
from sklearn.impute import SimpleImputer
mean_imp = SimpleImputer(strategy ='mean')
```

```
mean_imp.fit(X_train_imp.loc[:, ['albedo', 'H']])
X_train_imp.loc[:, ['albedo', 'H']] =
mean_imp.transform(X_train_imp.loc[:, ['albedo', 'H']])
X_test_imp.loc[:, ['albedo', 'H']] =
mean_imp.transform(X_test_imp.loc[:, ['albedo', 'H']])

## looking at the descriptive stats of albedo and H
albedo_after_imp = X_train_imp[['albedo']].describe()
H_after_imp = X_train_imp[['H']].describe()
pd.DataFrame({'albedo_before': albedo_before_imp.albedo,
'albedo_after': albedo_after_imp.albedo, 'H_before': H_before_imp.H,
'H_after': H_after_imp.H}, index=albedo_after_imp.index)
```

	albedo_before	albedo_after	H_before	H_after
count	97041.000000	97795.000000	97024.000000	97795.000000
mean	0.129928	0.129928	15.352153	15.352153
std	0.110231	0.109805	1.419130	1.413525
min	0.001000	0.001000	3.330000	3.330000
25%	0.053000	0.053000	14.620000	14.630000
50%	0.078000	0.079000	15.440000	15.430000
75%	0.188000	0.187000	16.230000	16.220000
max	1.000000	1.000000	29.900000	29.900000

Obervation:

- the descriptive stats before and after imputation looks similar.

```
## checking the null values in the imputed train df
X_train_imp.isnull().sum()[X_train_imp.isnull().sum() > 0]
data_arc    16
dtype: int64
```

We see that we have now only data_arc feature with missing values, we will impute it with median startegy

```
dataarc_before_imp = X_train_imp[['data_arc']].describe()

#creating median imputer object and fitting and transforming data

median_imp = SimpleImputer(strategy ='median')
median_imp.fit(X_train_imp)
X_train_imp = median_imp.transform(X_train_imp)
X_test_imp = median_imp.transform(X_test_imp)

# Converting th eimputed numpy arrays to dataframe
X_train_imp_df= pd.DataFrame(X_train_imp,
columns=X_train_encoded_df.columns)
X_test_imp_df = pd.DataFrame(X_test_imp,
columns=X_train_encoded_df.columns)
```

```
# making a copy of the dataframe and proceed with imputation on the
copied version
```

```python
X_train_iter_imp = X_train_encoded_df.copy()
X_test_iter_imp = X_test_encoded_df.copy()

# Note This estimator is still experimental for now: the predictions
and the API might change without any deprecation cycle.
# To use it, you need to explicitly import enable_iterative_imputer
from sklearn.experimental import enable_iterative_imputer from
sklearn.impute import IterativeImputer

iter_imp = IterativeImputer(random_state=0)
iter_imp.fit(X_train_iter_imp)
X_train_iter_imp = iter_imp.transform(X_train_iter_imp)
X_test_iterr_imp = iter_imp.transform(X_test_iter_imp)

# Converting th eimputed numpy arrays to dataframe
X_train_iter_imp= pd.DataFrame(X_train_iter_imp,
columns=X_train_encoded_df.columns)
X_test_iter_imp = pd.DataFrame(X_test_iter_imp,
columns=X_train_encoded_df.columns)

pd.DataFrame({'albedo_before_imputation': albedo_before_imp.albedo,
'albedo_after_mean_imputation': albedo_after_imp.albedo,
'albedo_iter_imp': X_train_iter_imp.albedo.describe()},
index=albedo_after_imp.index)
```

	albedo_before_imputation	albedo_after_mean_imputation	albedo_iter_imp
count	97041.000000	97795.000000	97795.000000
mean	0.129928	0.129928	0.129439
std	0.110231	0.109805	0.110067
min	0.001000	0.001000	-0.310484
25%	0.053000	0.053000	0.053000
50%	0.078000	0.079000	0.078000
75%	0.188000	0.187000	0.187000
max	1.000000	1.000000	1.000000

```python
pd.DataFrame({'H_before_imputation': H_before_imp.H,
'H_after_mean_imputaton': H_after_imp.H,
'H_after_iter_imputation':X_train_iter_imp.H.describe()},
index=albedo_after_imp.index)
```

```
       H_before_imputation  H_after_mean_imputaton
H_after_iter_imputation
count         97024.000000            97795.000000
97795.000000
mean             15.352153               15.352153
15.381626
std               1.419130                1.413525
1.460900
min               3.330000                3.330000
3.330000
25%              14.620000               14.630000
14.630000
50%              15.440000               15.430000
15.460000
75%              16.230000               16.220000
16.250000
max              29.900000               29.900000
29.900000

pd.DataFrame({'dataarc_before_imputation':
dataarc_before_imp.data_arc, 'dataarc_after_mean_imputation':
dataarc_after_imp.data_arc,
'dataarc_after_iter_imputation':X_train_iter_imp.data_arc.describe()},
index=dataarc_after_imp.index)
       dataarc_before_imputation  dataarc_after_mean_imputation  \
count               97779.000000                   97795.000000
mean                10152.485155                   10152.222813
std                  5940.434827                    5939.984257
min                     1.000000                       1.000000
25%                  7298.000000                    7298.000000
50%                  8549.000000                    8549.000000
75%                 10723.000000                   10722.500000
max                 79466.000000                   79466.000000

       dataarc_after_iter_imputation
count                   97795.000000
mean                    10150.455118
std                      5942.673781
min                    -15402.191152
25%                      7297.000000
50%                      8549.000000
75%                     10723.000000
max                     79466.000000
```

Observation:

- We observe that using Iterative Imputer, we ended up with some negative values for albedo and data_arc, which is not right, as we know from domain info that albedo lies `elif x in ['OMB', 'CEN', 'TJN', 'MBA', 'TNO']:                return 'c3'`

```
# making a copy of df
X_train_class_feat1 = X_train[['class']].copy() X_test_class_feat1 =
X_test[['class']].copy()

# applying featurization function based on size_kbin
X_train_class_feat1['orbit_class'] =
X_train_class_feat1['class'].apply(orbit_reclassify)
X_test_class_feat1['orbit_class'] =
X_test_class_feat1['class'].apply(orbit_reclassify)
```

Encoding orbit_class feature

```
ohe_class_1 = OneHotEncoder(drop='first')
ohe_class_1.fit(X_train_class_feat1[['orbit_class']])
OneHotEncoder(drop='first')
train_class_feat1 =
ohe_class_1.transform(X_train_class_feat1[['orbit_class']]).toarray()
test_class_feat1 =
ohe_class_1.transform(X_test_class_feat1[['orbit_class']]).toarray()

# replacing original encoded class variables with featurized class
variables in the imputed train dataframe X_train_class_featured1 =
X_train_imp_df.copy()
X_train_class_featured1.drop(['class_APO', 'class_AST', 'class_ATE',
'class_CEN',
       'class_IMB', 'class_MBA', 'class_MCA', 'class_OMB',
'class_TJN',
       'class_TNO'], axis=1, inplace=True)

X_test_class_featured1 = X_test_imp_df.copy()
X_test_class_featured1.drop(['class_APO', 'class_AST', 'class_ATE',
'class_CEN',
       'class_IMB', 'class_MBA', 'class_MCA', 'class_OMB',
'class_TJN',
       'class_TNO'], axis=1, inplace=True)

# getting the column names for dataframe creation
encoded_col_names= list(ohe_class_1.get_feature_names_out()) # from
onehot encoding
cols= list(X_train_class_featured1.columns)
cols.extend(encoded_col_names)

X_train_class_featured1 =
pd.DataFrame(np.hstack((X_train_class_featured1.to_numpy(),
train_class_feat1)), columns=cols)
X_test_class_featured1 =
```

```
pd.DataFrame(np.hstack((X_test_class_featured1.to_numpy(),
test_class_feat1)), columns=cols)
X_train_class_featured1.columns
Index(['H', 'albedo', 'epoch', 'e', 'a', 'q', 'i', 'ad', 'n', 'tp',
'per',
       'moid', 'moid_jup', 'data_arc', 'condition_code', 'rms',
'neo_Y',
       'pha_Y', 'orbit_class_c2', 'orbit_class_c3'],
      dtype='object')
```

Dataset_II

- X_train_class_featured1
- X_test_class_featured1

Evaluate if this reclassification compared to existing the classes improves mae score

```
## class featurized based on size_kbin
model_cls_1 = LinearRegression()
metric = mean_absolute_error
mae_cls_1, score_train_cls1, score_test_cls1 =
evaluate(X_train_class_featured1, y_train, X_test_class_featured1,
y_test, model_cls_1, metric)

print(f"MAE score for dataset with orbit_featurized_1 is:
{mae_cls_1}")
print(f"Train score: {score_train_cls1}")
print(f"Test score: {score_test_cls1}")

MAE score for dataset with orbit_featurized_1 is: 2.5820814399019762
Train score: 0.5118926533133668
Test score: 0.4631634685985727

datasets.append(('X_train_class_featured1', 'X_test_class_featured1'))
mae_datasets.append(mae_cls_1) train_scores.append(score_train_cls1)
test_scores.append(score_test_cls1)
```

Featurize by Orbit classification based on diameter size class

- Based on size_class, we can club these orbit_classes as follows:: –
  class_1: APO and ATE has only s1 and s2
  - class_2: MCA, AMO, IMB have s1, s2, s3 sized asteroids
  - class_3: AST have one s2 and s3
  - class_4: MBA, OMB and CEN has all four sized asteroids - s1, s2, s3, s4
  - class_5: TJN, TNO has only s2, s3, s4

```
def orbit_reclassify_size_class(x):
if x =='APO' or x=='ATE':
return 'c1'       elif x in ['MCA',
'AMO', 'IMB']:             return 'c2'
elif x == 'AST':              return
'c3'       elif x in ['MBA', 'OMB',
'CEN']:             return 'c4'
elif x in ['TJN', 'TNO']:
return 'c5'

# making a copy of df
X_train_class_feat2 = X_train[['class']].copy()
X_test_class_feat2 = X_test[['class']].copy()

# applying featurization function based on size_kbin
X_train_class_feat2['orbit_class'] =
X_train_class_feat2['class'].apply(orbit_reclassify_size_class)
X_test_class_feat2['orbit_class'] =
X_test_class_feat2['class'].apply(orbit_reclassify_size_class)

# looking at the value counts
X_train_class_feat2['orbit_class'].value_counts().sum()
97795
ohe_class_2 = OneHotEncoder(drop='first')
ohe_class_2.fit(X_train_class_feat2[['orbit_class']])
OneHotEncoder(drop='first')
train_class_feat2 =
ohe_class_2.transform(X_train_class_feat2[['orbit_class']]).toarray()
test_class_feat2 =
ohe_class_2.transform(X_test_class_feat2[['orbit_class']]).toarray()

# replacing original encoded class variables with featurized class
variables in the imputed train dataframe X_train_class_featured2 =
X_train_imp_df.copy()
X_train_class_featured2.drop(['class_APO', 'class_AST', 'class_ATE',
'class_CEN',
       'class_IMB', 'class_MBA', 'class_MCA', 'class_OMB',
'class_TJN',
       'class_TNO'], axis=1, inplace=True)

X_test_class_featured2 = X_test_imp_df.copy()
X_test_class_featured2.drop(['class_APO', 'class_AST', 'class_ATE',
'class_CEN',
       'class_IMB', 'class_MBA', 'class_MCA', 'class_OMB',
```

```
'class_TJN',
       'class_TNO'], axis=1, inplace=True)

# getting the column names for dataframe creation
encoded_col_names= list(ohe_class_2.get_feature_names_out()) # from
onehot encoding
cols= list(X_train_class_featured2.columns)
cols.extend(encoded_col_names)

X_train_class_featured2 =
pd.DataFrame(np.hstack((X_train_class_featured2.to_numpy(),
train_class_feat2)), columns=cols) X_test_class_featured2 =
pd.DataFrame(np.hstack((X_test_class_featured2.to_numpy(),
test_class_feat2)), columns=cols)
```

Dataset_III

- X_train_class_featured2
- X_test_class_featured2

Evaluate if this recalssification compared to existing the classes improves mae score

```
## class featurized based on size_kbin
model_cls_2 = LinearRegression()
metric = mean_absolute_error
mae_cls_2, score_train_cls2, score_test_cls2 =
evaluate(X_train_class_featured2, y_train, X_test_class_featured2,
y_test, model_cls_2, metric)

print(f"MAE score for dataset with orbit_featurized_by size_class is:
{mae_cls_2}")
print(f"Train score: {score_train_cls2}")
print(f"Test score: {score_test_cls2}")

MAE score for dataset with orbit_featurized_by size_class is:
2.5822784197907858
Train score: 0.5130757724895665
Test score: 0.4596074187682029

datasets.append(('X_train_class_featured2', 'X_test_class_featured2'))
mae_datasets.append(mae_cls_2) train_scores.append(score_train_cls2)
test_scores.append(score_test_cls2)
```

Observattions:

- MAE score with dataset with original orbit class features is the best, although difference is very small
- MAE score with orbit class featurized by size_class is slightly better than featurized by k_bins

We have got two Datasets from Orbit featurization here:

#####- Dataset_II: X_train_class_featured1, X_test_class_featured1 (based on kbinned diameter)

#####- Dataset_III: X_train_class_featured2, X_test_class_featured2 (based size binned diameter)

we will explore these during Modelling

```
#save files
save_to_csv(X_train_class_featured1, 'X_train_class_featured1')
save_to_csv(X_test_class_featured1, 'X_test_class_featured1')
save_to_csv(X_train_class_featured2, 'X_train_class_featured2')
save_to_csv(X_test_class_featured2, 'X_test_class_featured2')

#loading the dataframes from saved csv
X_train_class_featured1 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_train_clas
s_featured1')
X_test_class_featured1 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_test_class
_featured1')
X_train_class_featured2 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_train_clas
s_featured2')
X_test_class_featured2 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_test_class
_featured2')
```

Absolute magnitude featurization

- https://towardsdatascience.com/discretisation-using-decision-trees21910483fa4b
- we learnt from reference that the size (diameter of an equivalent sphere) of an asteroid can be estimated from its absolute magnitude H and an assumed geometric albedo
- H ranges between 0-40

```
#plotting H distribution
sns.set_style("whitegrid")
g = sns.displot(x=X_train.H, kde=True, rug=True, height=6, aspect=4)
g.set_xlabels(fontsize=20)
g.set_ylabels(fontsize=20)
g.fig.suptitle("Distribution of Absolute Magnitude", fontsize=20)
plt.show()
```

```
print(f"Train score for Dataset_I is: {score_train_h1_kbin}")
print(f"Test score for Dataset_I is: {score_test_h1_kbin}")
```

```
MAE score with featurized 'H' dataset based on diameter kbins:
1.7933822337101764
Train score for Dataset_I is: 0.6697213372907871
Test score for Dataset_I is: 0.6351207135936657
```

```
datasets.append(('X_train_H_featurized_1', 'X_test_H_featurized_1'))
mae_datasets.append(mae_h1_kbin)
train_scores.append(score_train_h1_kbin)
test_scores.append(score_test_h1_kbin)

#save files
save_to_csv(X_train_H_featurized_1, 'X_train_H_featurized_1')
save_to_csv(X_test_H_featurized_1, 'X_test_H_featurized_1')

#loading the dataframes from saved csv
X_train_H_featurized_1 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_train_H_fe
aturized_1')
X_test_H_featurized_1 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_test_H_fea
turized_1')
```

Featurizing 'H' set-3: Discreatization based on target - binned diameter by size

```
#making a copy of df
X_train_H_featurized_2 = X_train_imp_df.copy()
X_test_H_featurized_2 = X_test_imp_df.copy()

def size_bin(x):     if
x<=1:
sizebin='s1'     elif x>1
and x<=10:
sizebin='s2'     elif
x>10 and x<=100:
        sizebin='s3'
elif x>100:
        sizebin='s4'
return sizebin

# binning train and test target variables by size_class

y_tr_binned_size = y_tr_continous.apply(size_bin) from

sklearn.tree import DecisionTreeClassifier, plot_tree

# applying decision tree algorithm
dt2 = DecisionTreeClassifier(max_depth=2, random_state=0)
dt2.fit(X_train_H_featurized_2[['H']], y_tr_binned_size)
```

```
|   |   |   |   |--- H >  13.62
|   |   |   |   |   |--- class: s2
|--- H >  13.66
|   |--- H <= 18.77
|   |   |--- H <= 14.07
|   |   |   |--- H <= 13.91
|   |   |   |   |--- H <= 13.80
|   |   |   |   |   |--- class: s2
|   |   |   |   |--- H >  13.80
|   |   |   |   |   |--- class: s2
|   |   |   |--- H >  13.91
|   |   |   |   |--- H <= 13.93
|   |   |   |   |   |--- class: s2
|   |   |   |   |--- H >  13.93
|   |   |   |   |   |--- class: s2
|   |   |--- H >  14.07
|   |   |   |--- H <= 18.18
|   |   |   |   |--- H <= 14.24
|   |   |   |   |   |--- class: s2
|   |   |   |   |--- H >  14.24
|   |   |   |   |   |--- class: s2
|   |   |   |--- H >  18.18
|   |   |   |   |--- H <= 18.45
|   |   |   |   |   |--- class: s2
|   |   |   |   |--- H >  18.45
|   |   |   |   |   |--- class: s2
|   |--- H >  18.77
|   |   |--- H <= 19.53
|   |   |   |--- H <= 19.02
|   |   |   |   |--- H <= 18.89
|   |   |   |   |   |--- class: s1
|   |   |   |   |--- H >  18.89
|   |   |   |   |   |--- class: s2
|   |   |   |--- H >  19.02
|   |   |   |   |--- H <= 19.49
|   |   |   |   |   |--- class: s1
|   |   |   |   |--- H >  19.49
|   |   |   |   |   |--- class: s1
|   |   |--- H >  19.53
|   |   |   |--- H <= 20.09
|   |   |   |   |--- H <= 20.07
|   |   |   |   |   |--- class: s1
|   |   |   |   |--- H >  20.07
|   |   |   |   |   |--- class: s1
|   |   |   |--- H >  20.09 |
|   |   |   |--- class: s1
```

```
def get_h_binned_by_size(data):
h_kbinned=[]
```

```
#drop original 'H' and add featurized 'H'
X_train_H_featurized_2.drop('H', axis=1, inplace=True)
X_train_H_featurized_2['H'] = train_h_binned_size

#drop original 'H' and add featurized 'H'
X_test_H_featurized_2.drop('H', axis=1, inplace=True)
X_test_H_featurized_2['H'] = test_h_binned_size
```

Dataset_VI

- X_train_H_featurized_2
- X_test_H_featurized_2

Evaluating the featurized data

```
## evaluating for thefetaurized dataset
model_h_bin_size = LinearRegression()
metric = mean_absolute_error
mae_h_bin_size, score_train_h_bin_size, score_test_h_bin_size =
evaluate(X_train_H_featurized_2, y_train, X_test_H_featurized_2,
y_test, model_h_bin_size, metric)

print(f"MAE score with featurized 'H' dataset based on diameter kbins:
{mae_h_bin_size}")
print(f"Train score for Dataset_I is: {score_train_h_bin_size}")
print(f"Test score for Dataset_I is: {score_test_h_bin_size}")

MAE score with featurized 'H' dataset based on diameter kbins:
2.2754171095167006
Train score for Dataset_I is: 0.519860805537821
Test score for Dataset_I is: 0.46310235981569137

#updating dataset and score list
datasets.append(('X_train_H_featurized_2', 'X_test_H_featurized_2'))
mae_datasets.append(mae_h_bin_size)
train_scores.append(score_train_h_bin_size)
test_scores.append(score_test_h_bin_size)

#save files
save_to_csv(X_train_H_featurized_2, 'X_train_H_featurized_2')
save_to_csv(X_test_H_featurized_2, 'X_test_H_featurized_2')

#loading the dataframes from saved csv
X_train_H_featurized_2 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_train_H_fe
aturized_2')
X_test_H_featurized_2 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_test_H_fea
turized_2')
```

```
## evaluating for thefetaurized dataset
model_featuredset_2 = LinearRegression()
metric = mean_absolute_error
mae_featuredset_2, score_train_featured2, score_test_featured2 =
evaluate(X_train_featurized_2, y_train, X_test_featurized_2, y_test,
model_featuredset_2, metric)

print(f"MAE for featurized set 2: {mae_featuredset_2}")
print(f"Train score for featurized set 2: {score_train_featured2}")
print(f"Test score featurized set 2: {score_test_featured2}")

MAE for featurized set 2: 2.3212009363548707
Train score for featurized set 2: 0.5179471926421977
Test score featurized set 2: 0.46514754021022764

#updating dataset and score list
datasets.append(('X_train_featurized_2', 'X_test_featurized_2'))
mae_datasets.append(mae_featuredset_2)
train_scores.append(score_train_featured2)
test_scores.append(score_test_featured2)

#save files
save_to_csv(X_train_featurized_2, 'X_train_featurized_2')
save_to_csv(X_test_featurized_2, 'X_test_featurized_2')

#loading the dataframes from saved csv X_train_featurized_2
=
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_train_feat
urized_2')
X_test_featurized_2 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_test_featu
rized_2')
```

Dataset_IX: Concatenated featurized_set_3

To the Orbit class featurized set 1(X_train_class_featured1) dataset, we will add featurized 'H' from X_train_H_featurized_opt which was generated using optimal binning method based on continous target diameter values

```
datasets

[('X_train_imp_df', 'X_test_imp_df'),
 ('X_train_class_featured1', 'X_test_class_featured1'),
 ('X_train_class_featured2', 'X_test_class_featured2'),
 ('X_train_H_featurized_opt', 'X_test_H_featurized_opt'),
 ('X_train_H_featurized_1', 'X_test_H_featurized_1'),
 ('X_train_H_featurized_2', 'X_test_H_featurized_2'),
 ('X_train_featurized_1', 'X_test_featurized_1'),
 ('X_train_featurized_2', 'X_test_featurized_2')]
X_train_class_featured1.columns
```

```
Index(['H', 'albedo', 'epoch', 'e', 'a', 'q', 'i', 'ad', 'n', 'tp',
'per',
       'moid', 'moid_jup', 'data_arc', 'condition_code', 'rms',
'neo_Y',
       'pha_Y', 'orbit_class_c2', 'orbit_class_c3'],
dtype='object')
```

```
# featurized column H_optbin1
X_train_H_featurized_opt.columns
```

```
Index(['albedo', 'epoch', 'e', 'a', 'q', 'i', 'ad', 'n', 'tp', 'per',
'moid',
       'moid_jup', 'data_arc', 'condition_code', 'rms', 'neo_Y',
'pha_Y',
       'class_APO', 'class_AST', 'class_ATE', 'class_CEN',
'class_IMB',
       'class_MBA', 'class_MCA', 'class_OMB', 'class_TJN',
'class_TNO', 'H'],
dtype='object')
```

```
# creating final featurized train set
X_train_featurized_3 = X_train_class_featured1.copy().drop('H',
axis=1)
X_train_featurized_3['H'] = X_train_H_featurized_opt.H

# same for test set
X_test_featurized_3 = X_test_class_featured1.copy().drop('H', axis=1)
X_test_featurized_3['H'] = X_test_H_featurized_opt.H

X_test_featurized_3.columns
```

```
Index(['albedo', 'epoch', 'e', 'a', 'q', 'i', 'ad', 'n', 'tp', 'per',
'moid',
       'moid_jup', 'data_arc', 'condition_code', 'rms', 'neo_Y',
'pha_Y',
       'orbit_class_c2', 'orbit_class_c3', 'H'],
dtype='object')
```

```
#check shapes
X_train_featurized_3.shape, X_test_featurized_3.shape
```

```
((97795, 20), (41913, 20))
```

Evaluate featurized set 3

```
## evaluating for thefetaurized dataset
model_featuredset_3 = LinearRegression()
metric = mean_absolute_error
mae_featuredset_3, score_train_featured3, score_test_featured3 =
```

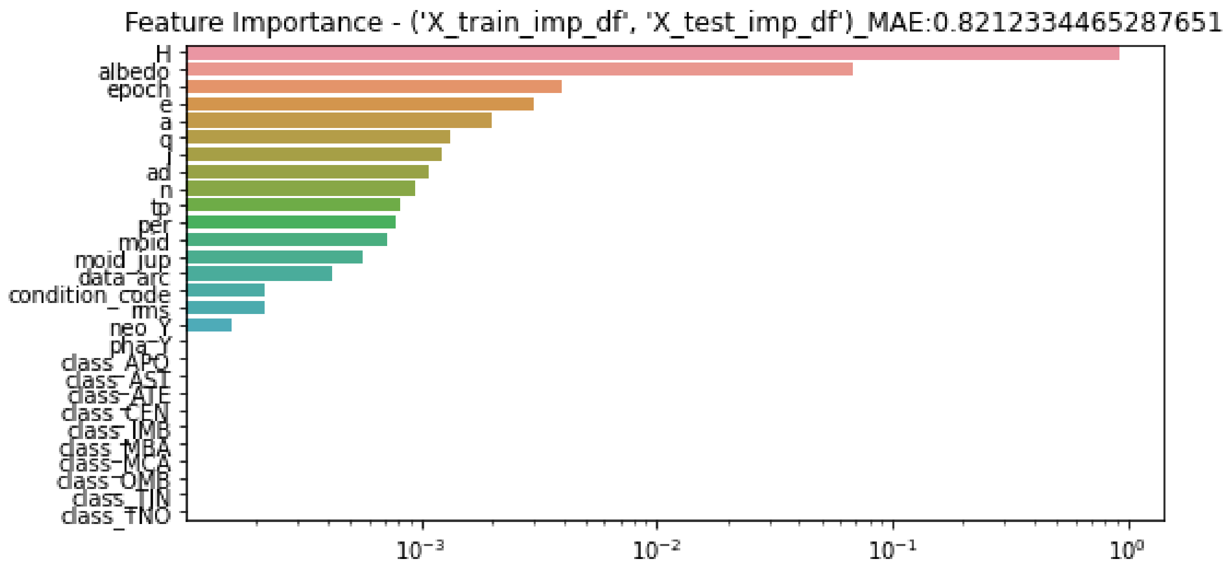
```
2       0.795377     0.828564
3       -0.197422
        0.560686
4       -1.907079
        0.686076
5       -2.057298
        0.632194
6       -1.298997
        0.702049
7       -9.321868
        0.648223
8       -0.197422
        0.560686
```

Observation:

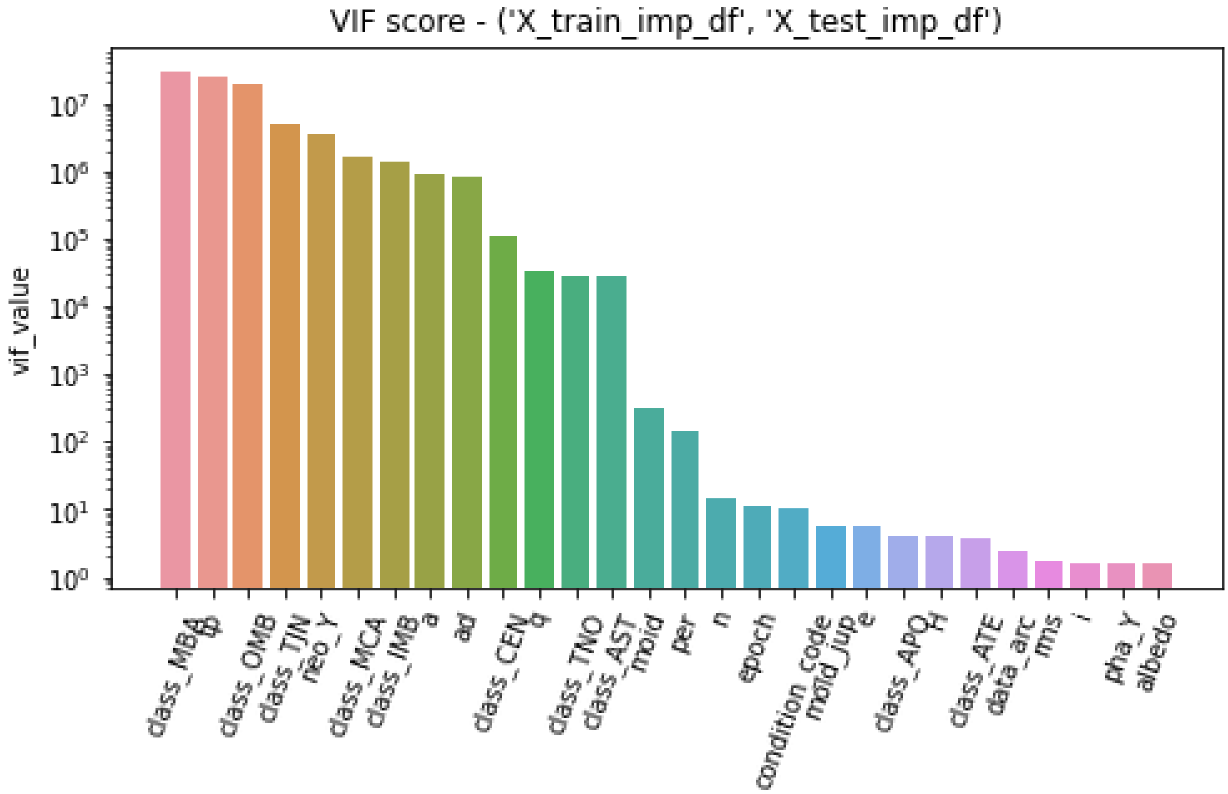
- MAE score seems to improved
- R2 score for train is terrible
- Choose not to go with these transformations

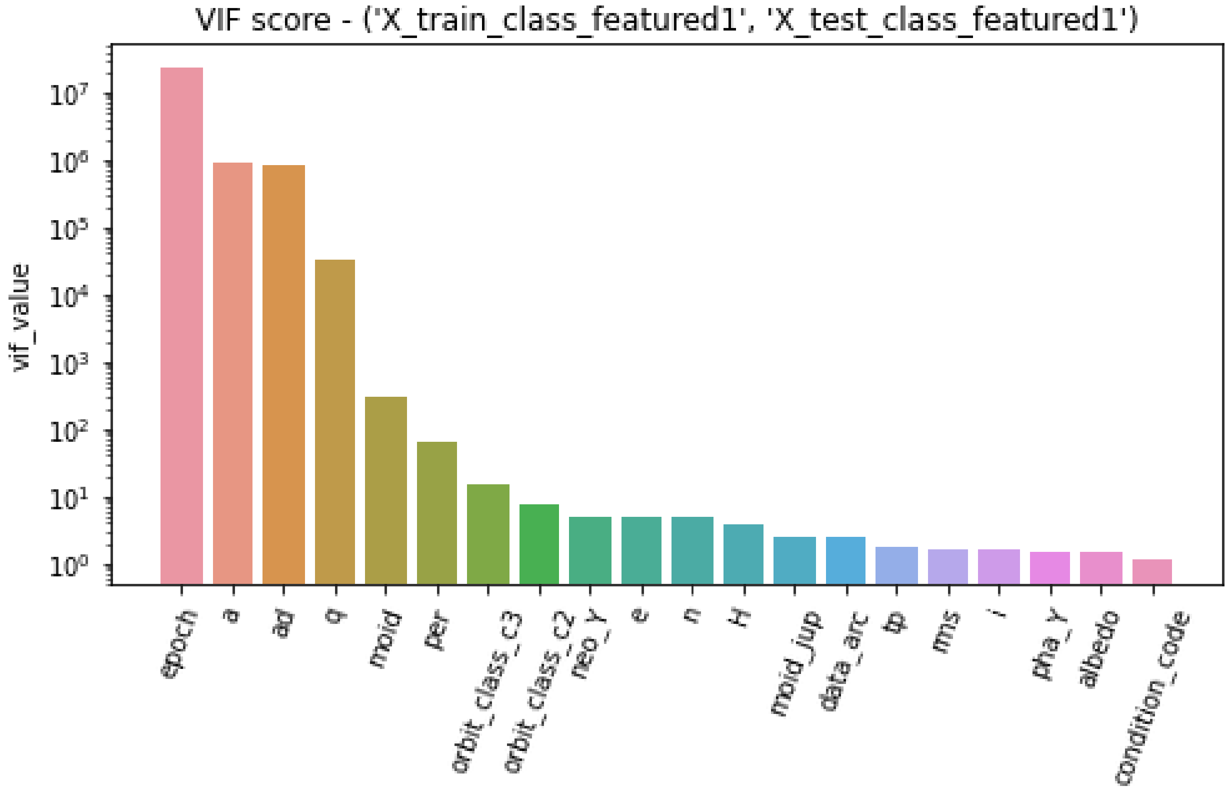
Feature Importance in the generated datasets

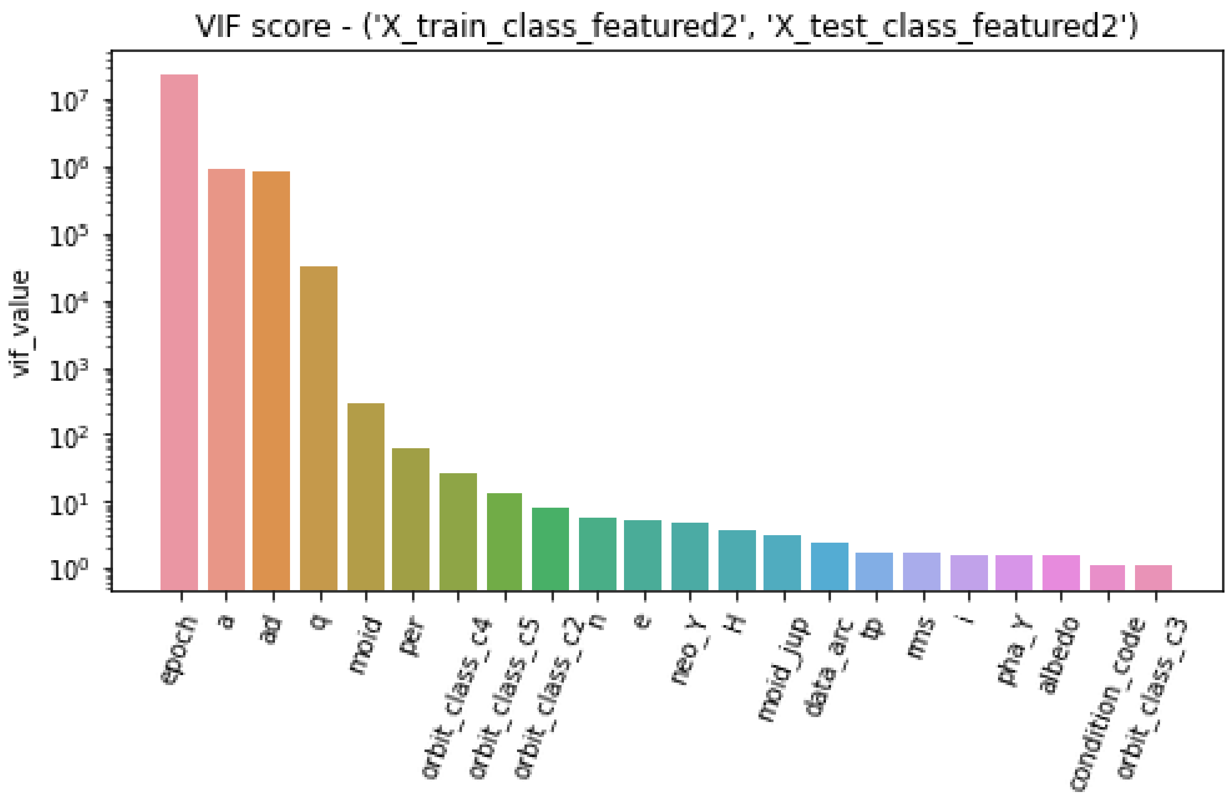
```
#lets look at the Feature importance
from sklearn.ensemble import RandomForestRegressor

mae_rf_scores_dataset=[] for
dataset in datasets:
train_data = eval(dataset[0])
test_data = eval(dataset[1])
lr = LinearRegression()
      rf1 = RandomForestRegressor(n_estimators=100, n_jobs=-1,
max_depth=5, random_state=42)
      mae_rf1, train_score_rf1, test_score_rf1= evaluate(train_data,
y_train, test_data, y_test, rf1, metric)
mae_rf_scores_dataset.append(mae_rf1)

      plot_feature_imp(rf1, train_data, f'{dataset}_MAE:{mae_rf1}')
```




```
def compute_vif(train_data, feature_set):
      """ function to compute vif and plot graph, return list of
collinear features and new data with removed features"""
      vif = [variance_inflation_factor(train_data.values, i) for i in
range(train_data.shape[1])]
      #dataframe - feature VIF values
      vif_df = pd.DataFrame({'vif_value': vif},
index=train_data.columns).sort_values('vif_value', ascending=False)

      ##plot graph
      xlabels =[i for i in vif_df.index]
fig, ax = plt.subplots(figsize=(8,4))
      ax = sns.barplot(x=vif_df.index,y="vif_value", data=vif_df)
ax.set(yscale='log')
      ax.set_xticklabels(labels=xlabels, rotation=70)
ax.set_title(f'VIF score - {feature_set}')
plt.plot()

## computing VIF for the feature sets for i
in range(len(datasets)):     train_data =
eval(datasets[i][0]).copy()
compute_vif(train_data, datasets[i])
```


VIF score - ('X_train_imp_df', 'X_test_imp_df')
vif_value
class_MBA
q
class_OMB
class_TJN
neo_Y
class_MCA
class_IMB
a
ad
class_CEN
q
class_TNO
class_AST
moid
per
n
epoch
condition_code
moid_jup
e
class_APO
H
class_ATE
data_arc
rms
i
pha_Y
albedo


VIF score - ('X_train_class_featured1', 'X_test_class_featured1')
vif_value
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
epoch
a
ad
q
moid
per
orbit_class_c3
orbit_class_c2
neo_Y
e
n
H
moid_jup
data_arc
tp
ms
i
pha_Y
albedo
condition_code


VIF score - ('X_train_class_featured2', 'X_test_class_featured2')
vif_value
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
epoch
a
ad
q
moid
per
orbit_class_c4
orbit_class_c5
orbit_class_c2
n
e
neo_Y
H
moid_jup
data_arc
tp
ms
i
pha_Y
albedo
condition_code
orbit_class_c3

Remove highly collinear features

```
def remove_high_vifs(train_data):
      """ function to compute vif and plot graph, return list of
collinear features and new data with removed features"""

      drop = True
      collinear_features = []       while drop:           vif =
[variance_inflation_factor(train_data.values, i) for i in
range(train_data.shape[1])]

          #dataframe - feature VIF values
          vif_df = pd.DataFrame({'vif_value': vif,
'feature':train_data.columns}).sort_values('vif_value',
ascending=False)
          vif_max_value = max(vif_df.vif_value.values)

          if     vif_max_value     >=     5:
vif_max_feat =
vif_df[vif_df.vif_value==vif_max_value].feature.values.item()
collinear_features.append(vif_max_feat)
              train_data = train_data.drop(vif_max_feat, axis=1)
else:              drop=False       return collinear_features
## computing high collinearity features

high_collinear_features = [] for i in
range(len(datasets)):     train_data =
eval(datasets[i][0]).copy()     cf =
remove_high_vifs(train_data)
high_collinear_features.append(cf)

pd.set_option('display.max_colwidth', 0)
high_collinear_features_df = pd.DataFrame({'Dataset':datasets,
'High_collinear_features':high_collinear_features})

high_collinear_features_df

                                               Dataset  \
0  (X_train_imp_df, X_test_imp_df)                        1
(X_train_class_featured1, X_test_class_featured1)
2  (X_train_class_featured2, X_test_class_featured2)     3
(X_train_H_featurized_opt, X_test_H_featurized_opt)   4
(X_train_H_featurized_1, X_test_H_featurized_1)
5  (X_train_H_featurized_2, X_test_H_featurized_2)       6
(X_train_featurized_1, X_test_featurized_1)
```

```
7   (X_train_featurized_2, X_test_featurized_2)
8   (X_train_featurized_3, X_test_featurized_3)
                                              High_collinear_features

0   [class_MBA, epoch, a, q, tp, H, ad, n, rms, moid_jup, moid]

1   [epoch, a, q, tp, H, orbit_class_c3, ad, rms, n, moid_jup]

2   [epoch, a, q, tp, H, orbit_class_c4, ad, n, rms, moid_jup, moid]

3   [class_MBA, epoch, a, q, tp, ad, n, rms, moid_jup, data_arc, moid]

4   [class_MBA, epoch, a, q, tp, ad, n, rms, moid_jup, moid]

5   [class_MBA, epoch, a, q, tp, ad, n, H, rms, moid_jup, moid]

6   [epoch, a, q, tp, orbit_class_c3, ad, rms, n, moid_jup, data_arc]

7   [epoch, a, q, tp, orbit_class_c4, ad, H, n, rms, moid_jup, moid]
8   [epoch, a, q, tp, orbit_class_c3, ad, rms, moid_jup, n, data_arc]
```

Observations:

- Note, we have very important features like 'H', 'data_arc' got removed
- features like 'a', 'q', 'epoch' also were relevant ones as understood from EDA

Removing highly collinear features

```
#defined vif addressed dataset

X_train_set1 = X_train_imp_df.drop(high_collinear_features[0], axis=1)
X_test_set1 = X_test_imp_df.drop(high_collinear_features[0], axis=1)

X_train_set2 =
X_train_class_featured1.drop(high_collinear_features[1], axis=1)
X_test_set2 = X_test_class_featured1.drop(high_collinear_features[1],
axis=1)

X_train_set3 =
X_train_class_featured2.drop(high_collinear_features[2], axis=1)
X_test_set3 = X_test_class_featured2.drop(high_collinear_features[2],
axis=1)

X_train_set4 =
X_train_H_featurized_opt.drop(high_collinear_features[3], axis=1)
X_test_set4 = X_test_H_featurized_opt.drop(high_collinear_features[3],
axis=1)
```

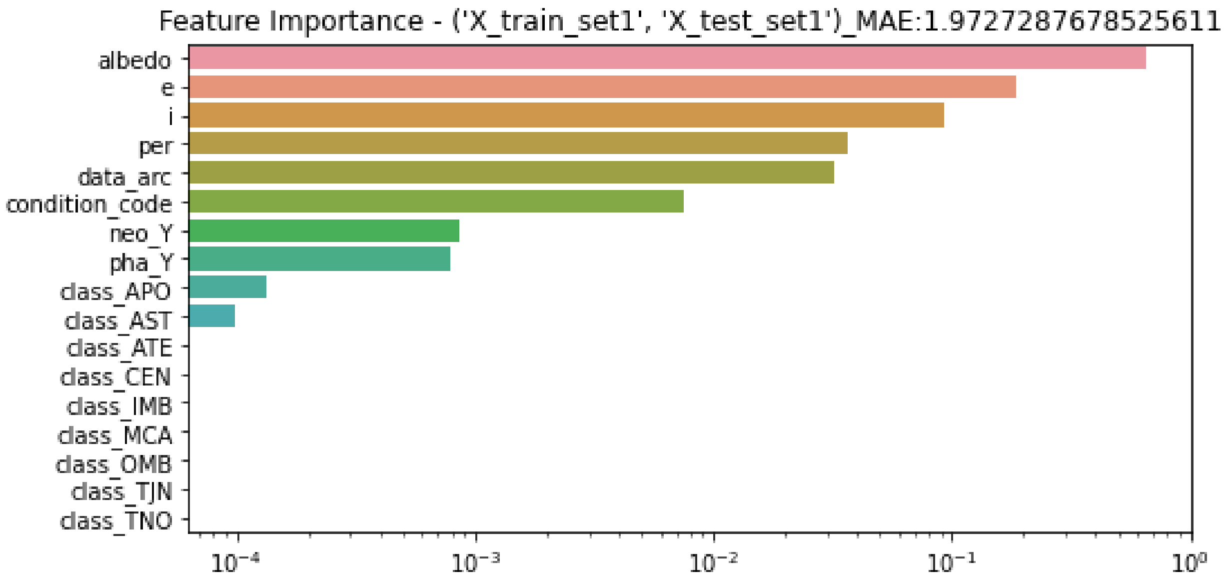
```
X_train_set5 = X_train_H_featurized_1.drop(high_collinear_features[4],
save_to_csv(X_test_set7,    'X_test_set7')    save_to_csv(X_test_set8,
'X_test_set8') save_to_csv(X_test_set9, 'X_test_set9')
```

```
collinearity_removed_dataset = [('X_train_set1','X_test_set1'),
('X_train_set2','X_test_set2'), ('X_train_set3','X_test_set3'),
('X_train_set4','X_test_set4'),('X_train_set5','X_test_set5'),
('X_train_set6','X_test_set6'), ('X_train_set7','X_test_set7'),
('X_train_set8','X_test_set8'), ('X_train_set9','X_test_set9')]
```

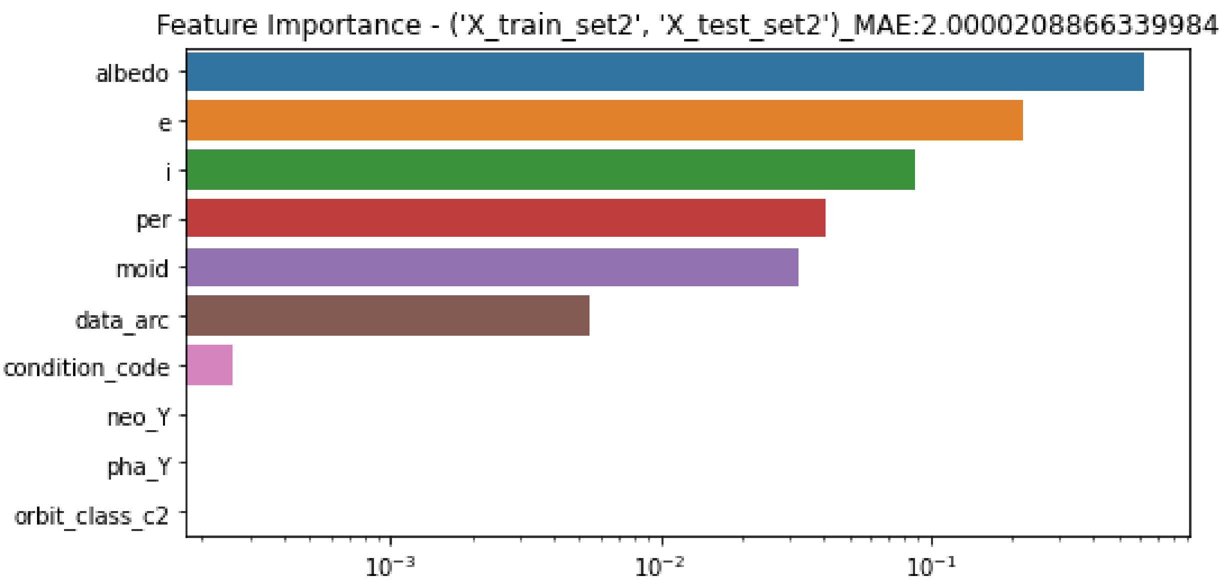
Feature Importance for the multicoolinearity addressed datasets

evaluating feature importance and MAE score using target y_train, y_test

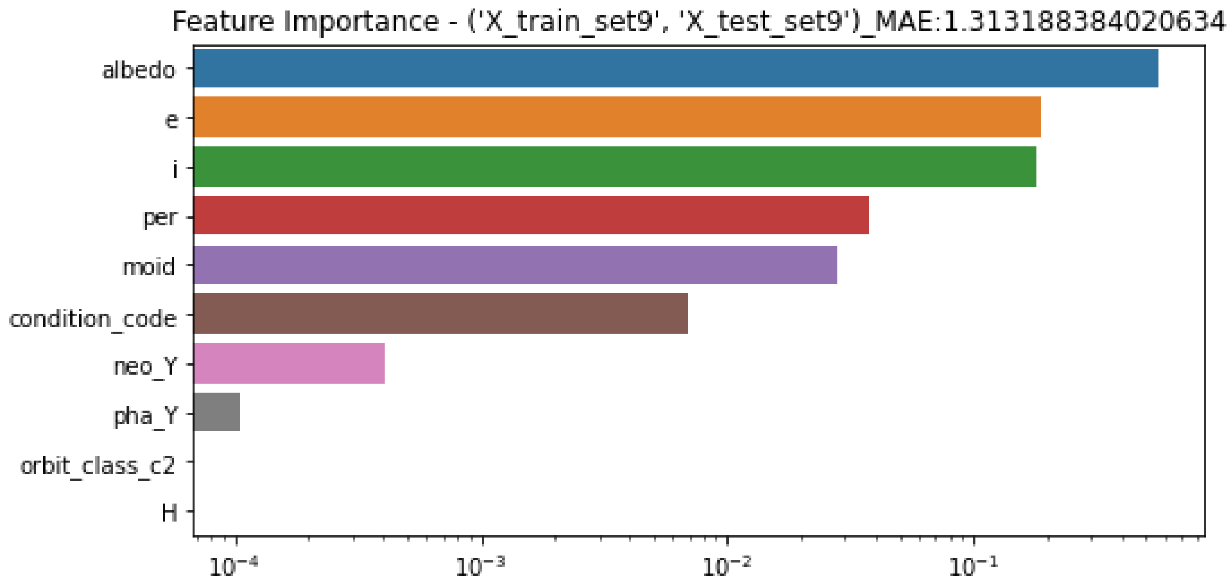
```
from sklearn.ensemble import RandomForestRegressor

mae_scores_new_dataset =[]
mae_rf_scores_new_dataset=[] for dataset in
collinearity_removed_dataset:
train_data = eval(dataset[0])
test_data = eval(dataset[1])       lr =
LinearRegression()
      rf1 = RandomForestRegressor(n_estimators=100, n_jobs=-1,
max_depth=5, random_state=42)
      mae_lr, train_score, test_score= evaluate(train_data, y_train,
test_data, y_test, lr, metric)
      mae_rf1, train_score_rf1, test_score_rf1= evaluate(train_data,
y_train, test_data, y_test, rf1, metric)
mae_scores_new_dataset.append(mae_lr)
mae_rf_scores_new_dataset.append(mae_rf1)

      plot_feature_imp(rf1, train_data, f'{dataset}_MAE:{mae_rf1}')
```



Feature Importance - ('X_train_set1', 'X_test_set1')_MAE:1.9727287678525611
albedo
e
i
per
data_arc
condition_code
neo_Y
pha_Y
class_APO
class_AST
class_ATE
class_CEN
class_IMB
class_MCA
class_OMB
class_TJN
class_TNO
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}



Feature Importance - ('X_train_set2', 'X_test_set2')_MAE:2.0000208866339984
albedo
e
i
per
moid
data_arc
condition_code
neo_Y
pha_Y
orbit_class_c2
10^{-3}
10^{-2}
10^{-1}



Top Important features are albedo, 'e', 'i', per, data_arc, neo_y, pha_y All datasets and Scores

```
Datasets_scores['Collinearity_fixed_set'] =
collinearity_removed_dataset
Datasets_scores['LR_Mae_cf_set'] = mae_scores_new_dataset
Datasets_scores['RF_Mae_cf_set'] = mae_rf_scores_new_dataset

pd.set_option('display.max_colwidth', 30)
Datasets_scores
           #              Dataset  LR_MAE_Scores  Train Score  Test
Score  \
0  Dataset_1  (X_train_imp_df,...       2.519004     0.51702
0.451986
1  Dataset_2  (X_train_class_f...       2.582081     0.51189
0.463163
2  Dataset_3  (X_train_class_f...       2.582278     0.51307
0.459607
3  Dataset_4  (X_train_H_featu...       2.243917     0.48425
0.416205
4  Dataset_5  (X_train_H_featu...       1.793382     0.66972
0.635121
5  Dataset_6  (X_train_H_featu...       2.275417     0.51986
0.463102
6  Dataset_7  (X_train_featuri...       1.848656     0.66434
0.637728
7  Dataset_8  (X_train_featuri...       2.321201     0.51794
0.465148
8  Dataset_9  (X_train_featuri...       2.339326     0.47714
0.425648
```

```
   RF_MAE_Scores Collinearity_fixed_set  LR_Mae_cf_set  RF_Mae_cf_set 0
0.821233  (X_train_set1, X...         2.655051       1.972729
1        0.820748  (X_train_set2, X...         2.713920       2.000021
2        0.819560  (X_train_set3, X...         2.662630       1.972478
3        1.212846  (X_train_set4, X...         1.650721       1.311443
4        1.650528  (X_train_set5, X...         1.962012       1.713343
5        1.627899  (X_train_set6, X...         2.655051       1.972729
6        1.651855  (X_train_set7, X...         2.160515       1.986037
7        1.625726  (X_train_set8, X...         2.662630       1.972478
8        1.213021  (X_train_set9, X...         1.832891       1.313188
save_to_csv(Datasets_scores, 'Datasets_scores')
```

Observations:

- Dataset based on OptimalBinning of Abosulte magnitude gave the lowest mean absolute error:
  - (X_train_H_featurized_opt, X_test_H_featurized_opt)

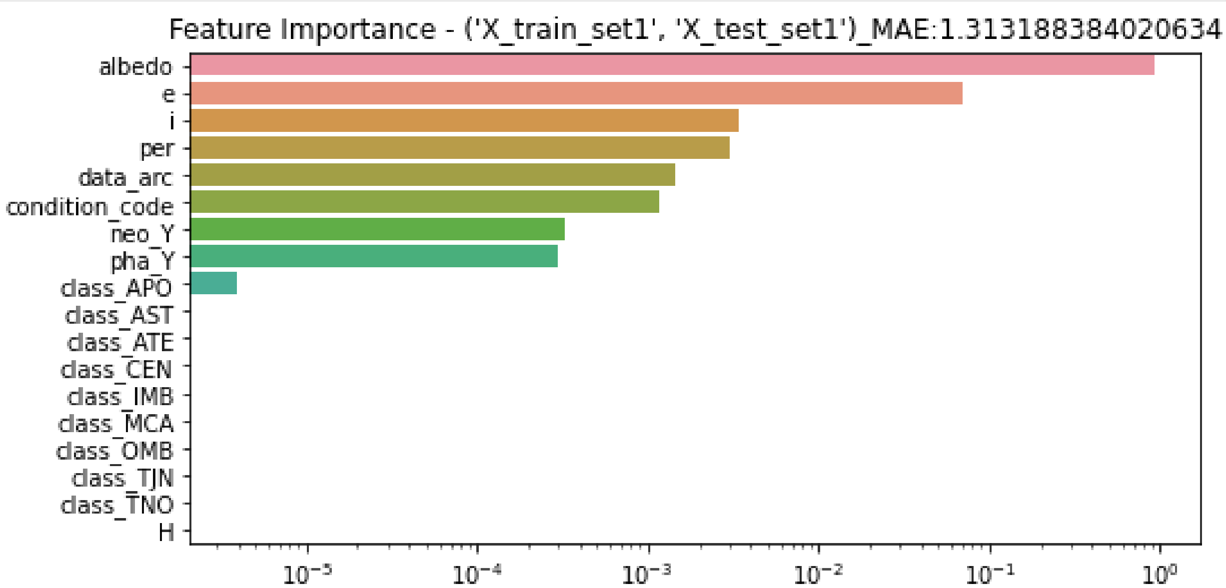
- (X_train_featurized_3, X_test_featurized_3)

- Next best Dataset based on MAE scores is (X_train_H_featurized_1, X_test_H_featurized_1), which has absolute magnitude feature discreatized based on target Kmeans binned diameter

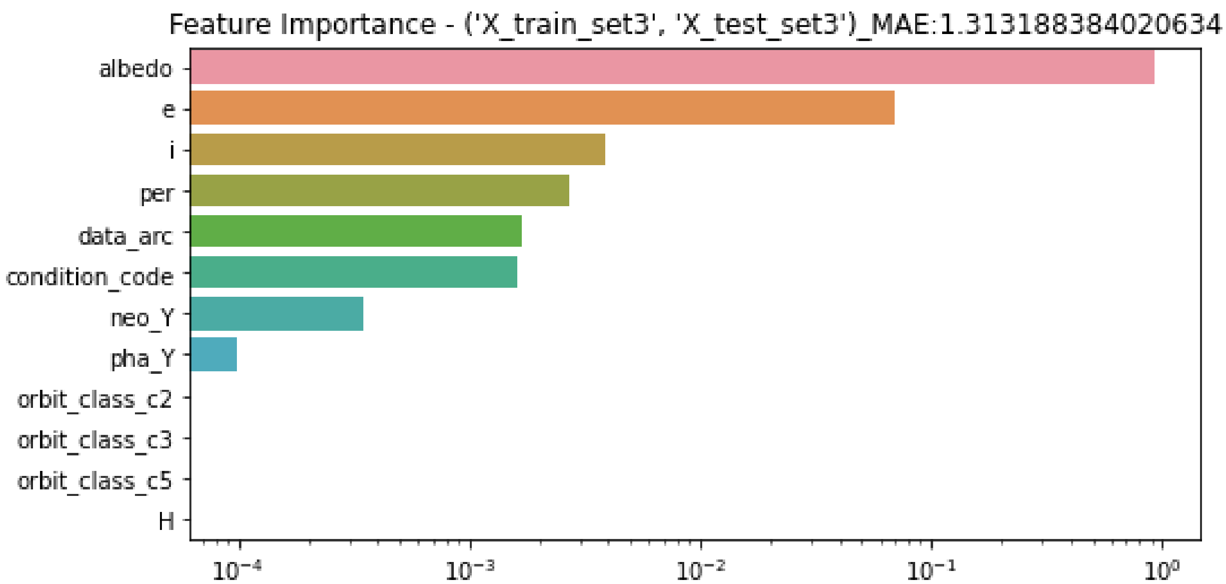
- **Note** the sets where the MAE went up after collinearity fix, are the ones from which abosulte magnitude feature got removed, so lets add it back and recalculate

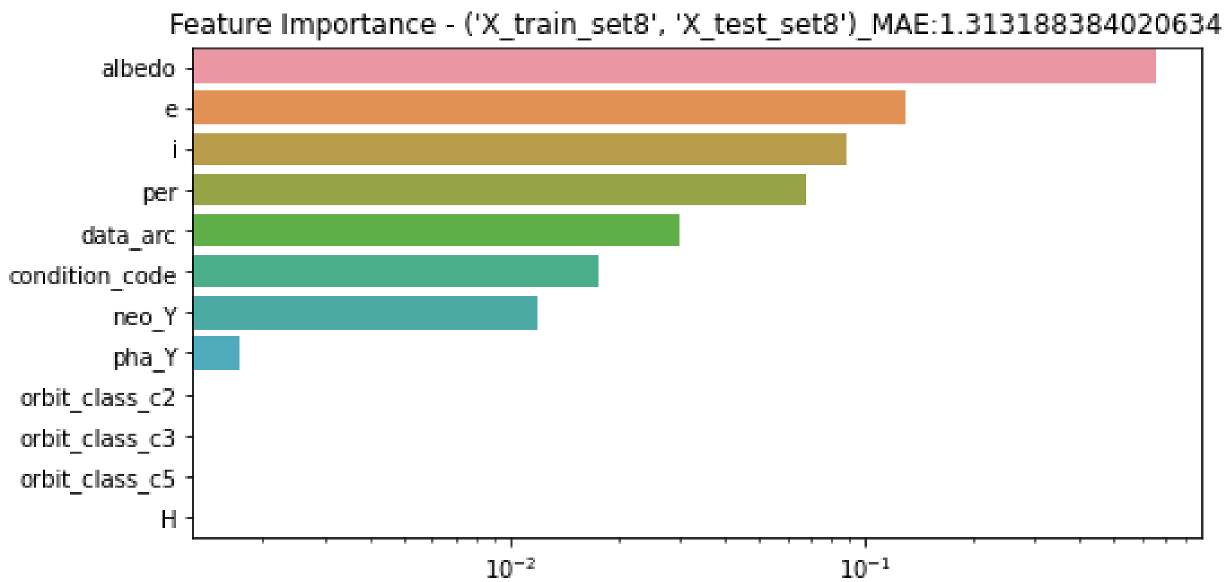
```
#loading the dataframes from saved csv
X_train_set1 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_train_set1
')
X_test_set1 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_test_set1'
)
X_train_set2 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_train_set2
')
X_test_set2 =
pd.read_csv('/content/drive/MyDrive/project_asteroid/data/X_test_set2'
)
```

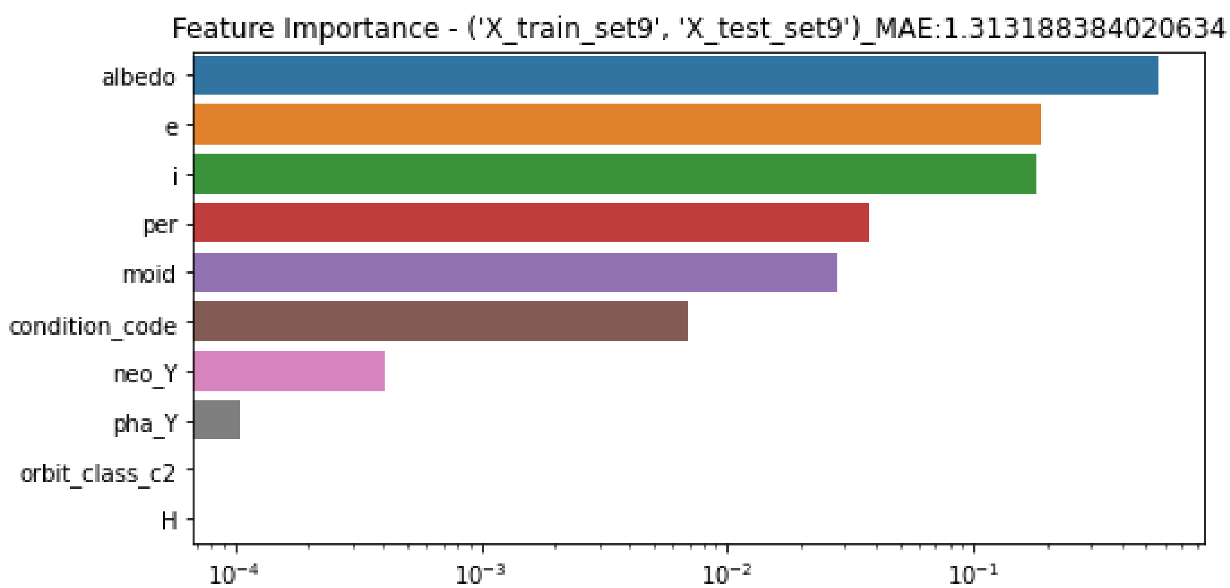
```
X_train_set3['H'] = X_train_imp_df.H
X_test_set3['H'] = X_test_imp_df.H

X_train_set6['H'] = X_train_H_featurized_2.H
X_test_set6['H'] = X_test_H_featurized_2.H

X_train_set8['H'] = X_train_H_featurized_2.H
X_test_set8['H'] = X_test_H_featurized_2.H

mae_scores_new_dataset_h =[]
mae_rf_scores_new_dataset_h=[] lr_scores_new
= [] rf_scores_new = [] for dataset in
collinearity_removed_dataset:
train_data = eval(dataset[0])
test_data = eval(dataset[1])       lr =
LinearRegression()
      rf1 = RandomForestRegressor(n_estimators=100, n_jobs=-1,
max_depth=5, random_state=42)
      mae_lr_h, train_score, test_score= evaluate(train_data, y_train,
test_data, y_test, lr, metric)
      mae_rf1_h, train_score_rf1, test_score_rf1= evaluate(train_data,
y_train, test_data, y_test, rf1, metric)
mae_scores_new_dataset_h.append(mae_lr_h)
mae_rf_scores_new_dataset_h.append(mae_rf1_h)
lr_scores_new.append((train_score, test_score))
rf_scores_new.append((train_score_rf1, test_score_rf1))
plot_feature_imp(rf1, train_data, f'{dataset}_MAE:{mae_rf1}')
```




Feature Importance - ('X_train_set2', 'X_test_set2')_MAE:1.313188384020634
albedo
e
i
per
moid
data_arc
condition_code
neo_Y
pha_Y
orbit_class_c2
H
10^{-3}
10^{-2}
10^{-1}
10^{0}


Feature Importance - ('X_train_set3', 'X_test_set3')_MAE:1.313188384020634
albedo
e
i
per
data_arc
condition_code
neo_Y
pha_Y
orbit_class_c2
orbit_class_c3
orbit_class_c5
H
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}





Observations:

- Although adding 'H' to set 1,2,3,6,9 did improve MAEs, but if we notice the feature 'H' is not being considered as an important feature

```
## Scores of the sets after adding absolute magnitude feature
pd.DataFrame({"Datasets":collinearity_removed_dataset,
"LR_MAE":mae_scores_new_dataset_h,
"RF_MAE":mae_rf_scores_new_dataset_h, "LR_R2_Scores":lr_scores_new,
"RF_R2_Scores":rf_scores_new})
               Datasets    LR_MAE    RF_MAE         LR_R2_Scores  \
0  (X_train_set1, X...  2.424977  0.820436  (0.4684000150523...
1  (X_train_set2, X...  2.563395  0.817732  (0.4601032015051...
```

Evaluate - considering only top important features

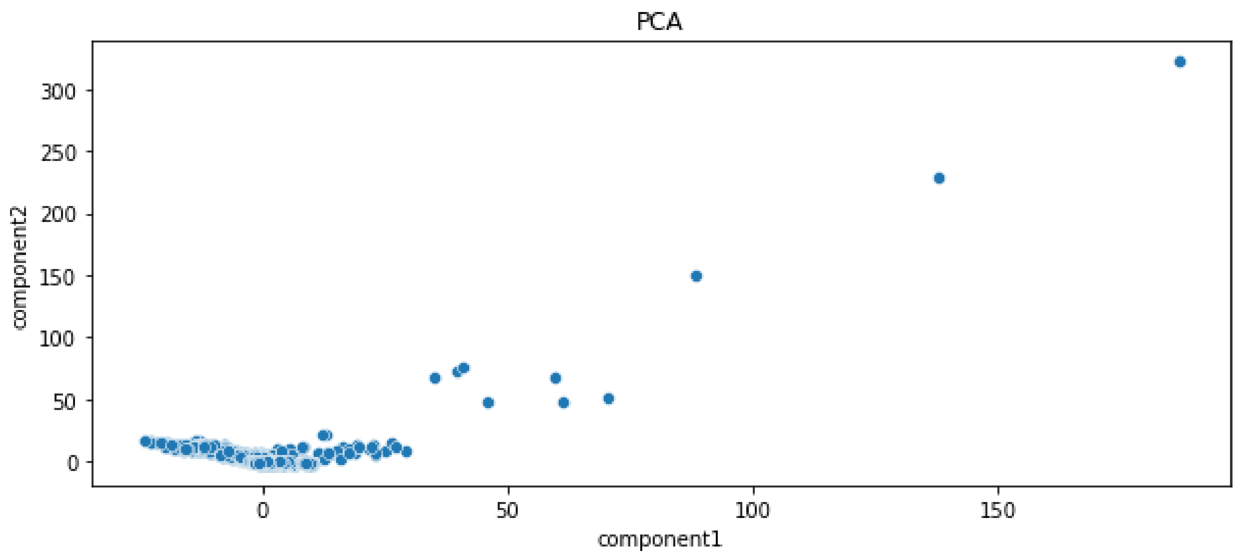
Based on feature Importance graphical view, below is what we call most important features for each set:

```
set1: ['albedo', 'e', 'i', 'per', 'data_arc', 'condition_code',
'neo_Y', 'pha_Y']
set2: ['albedo', 'e', 'i', 'per', 'moid', 'data_arc',
'condition_code', 'neo_Y']
set3: ['albedo', 'e', 'i', 'per', 'data_arc', 'condition_code',
'neo_Y', 'pha_Y']
set4: ['albedo', 'e', 'i', 'per', 'condition_code', 'neo_Y', 'pha_Y',
'class_APO', 'class_AST', 'class_ATE', 'class_CEN', 'class_IMB']
set5: ['albedo', 'e', 'i', 'per', 'data_arc', 'condition_code',
'neo_Y', 'pha_Y', 'class_APO', 'class_AST','class_ATE'] set6:
['albedo', 'e', 'i', 'per', 'data_arc', 'condition_code',
'neo_Y', 'pha_Y', 'class_APO', 'class_AST']
set7: ['albedo', 'e', 'i', 'per', 'moid', 'condition_code', 'neo_Y',
'pha_Y', 'orbit_class_c2]
set8: ['albedo', 'e', 'i', 'per', 'data_arc', 'condition_code',
'neo_Y', 'pha_Y']
set9: ['albedo', 'e', 'i', 'per', 'condition_code', 'neo_Y',
'pha_Y','class_APO', 'class_AST', 'class_ATE', 'class_CEN',
'class_IMB']
```

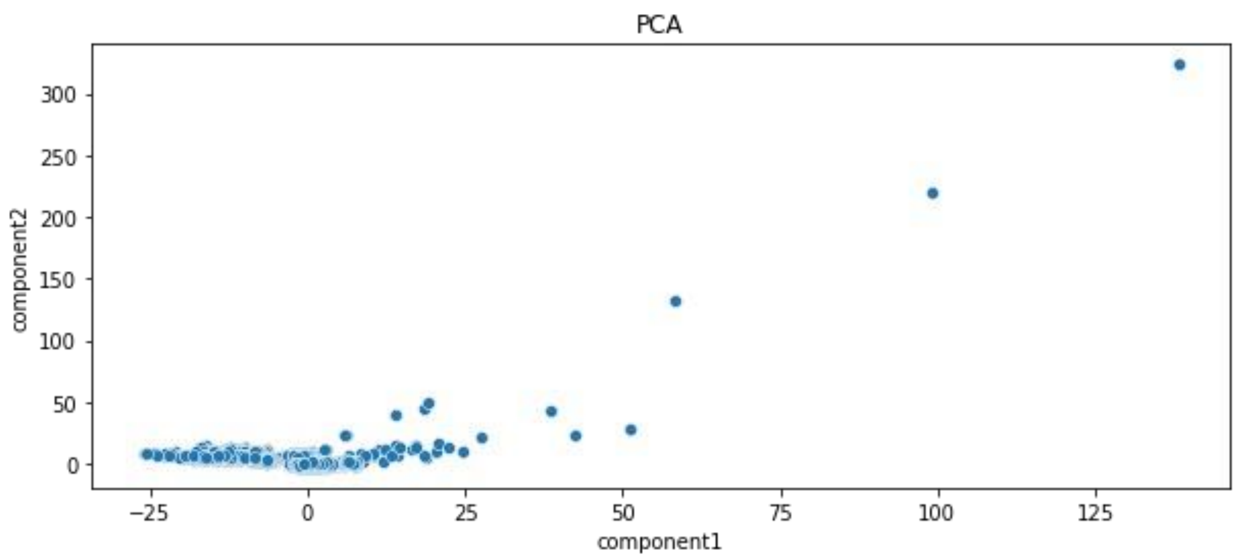
if we note we have 6 unique sets, so lets create these feature sets and evaluate

```
feature_important_train_set1 = X_train_set1[['albedo', 'e', 'i',
'per', 'data_arc', 'condition_code', 'neo_Y', 'pha_Y']]
feature_important_test_set1 = X_test_set1[['albedo', 'e', 'i', 'per',
'data_arc', 'condition_code', 'neo_Y', 'pha_Y']]

lr= LinearRegression()
rf1 = RandomForestRegressor(n_estimators=100, n_jobs=-1, max_depth=5,
random_state=42)
print(evaluate(feature_important_train_set1, y_train,
feature_important_test_set1, y_test, lr, metric))
print(evaluate(feature_important_train_set1, y_train,
feature_important_test_set1, y_test, rf1, metric))
(2.767421829826543, 0.319479362692358, 0.3447801947642173)
(1.9738053527602248, 0.7118078007045394, 0.6313314833617341)
feature_important_train_set2 = X_train_set2[['albedo', 'e', 'i',
'per', 'moid', 'data_arc', 'condition_code', 'neo_Y']]
feature_important_test_set2 = X_test_set2[['albedo', 'e', 'i', 'per',
'moid', 'data_arc', 'condition_code', 'neo_Y']]

lr= LinearRegression()
rf1 = RandomForestRegressor(n_estimators=100, n_jobs=-1, max_depth=5,
```

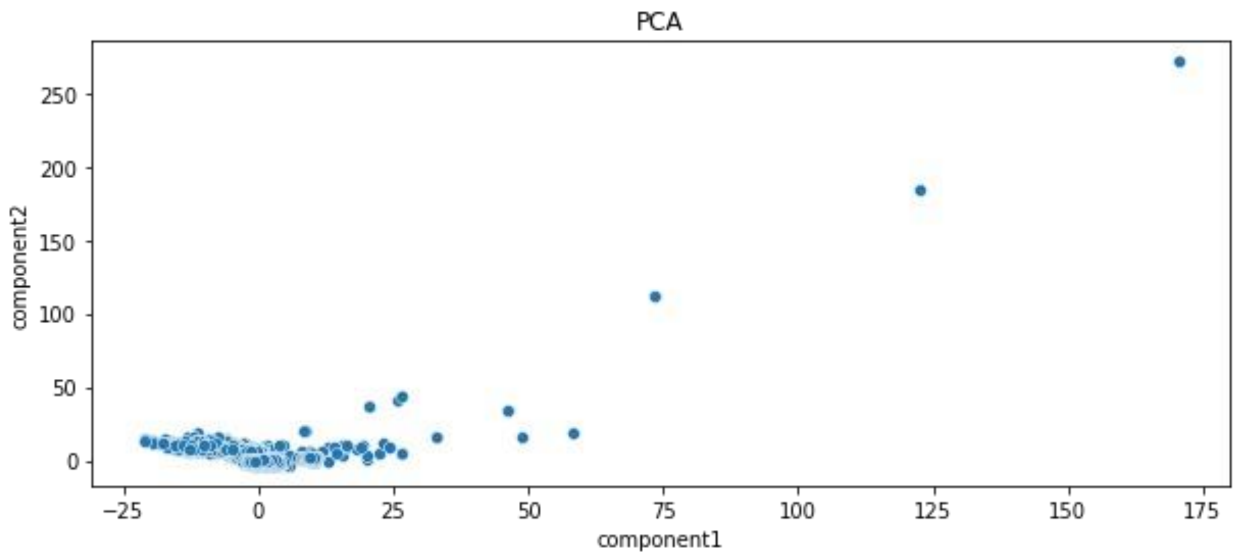
```
random_state=42)
print(evaluate(feature_important_train_set2, y_train,
feature_important_test_set2, y_test, lr, metric))
print(evaluate(feature_important_train_set2, y_train,
feature_important_test_set2, y_test, rf1, metric))

(2.71316619122659, 0.42612066182376185, 0.40495629721809323)
(1.999836874179555, 0.7081655676707488, 0.6446306947052883)

feature_important_train_set4 = X_train_set4[['albedo', 'e', 'i',
'per', 'condition_code', 'neo_Y', 'pha_Y', 'class_APO', 'class_AST',
'class_ATE', 'class_CEN', 'class_IMB']]
feature_important_test_set4 = X_test_set4[['albedo', 'e', 'i', 'per',
'condition_code', 'neo_Y', 'pha_Y', 'class_APO', 'class_AST',
'class_ATE', 'class_CEN', 'class_IMB']]

lr= LinearRegression()
rf1 = RandomForestRegressor(n_estimators=100, n_jobs=-1, max_depth=5,
random_state=42)
print(evaluate(feature_important_train_set4, y_train,
feature_important_test_set4, y_test, lr, metric))
print(evaluate(feature_important_train_set4, y_train,
feature_important_test_set4, y_test, rf1, metric))

(2.9738037005873768, 0.03244582615976399, 0.03486274565715086)
(2.6513352984856966, 0.2088216513443797, 0.09700221365648187)

feature_important_train_set5 = X_train_set5[['albedo', 'e', 'i',
'per', 'data_arc', 'condition_code', 'neo_Y', 'pha_Y', 'class_APO',
'class_AST','class_ATE']]
feature_important_test_set5 = X_test_set5[['albedo', 'e', 'i', 'per',
'data_arc', 'condition_code', 'neo_Y', 'pha_Y', 'class_APO',
'class_AST','class_ATE']]

lr= LinearRegression()
rf1 = RandomForestRegressor(n_estimators=100, n_jobs=-1, max_depth=5,
random_state=42)
print(evaluate(feature_important_train_set5, y_train,
feature_important_test_set5, y_test, lr, metric))
print(evaluate(feature_important_train_set5, y_train,
feature_important_test_set5, y_test, rf1, metric))
(2.768209995753874, 0.31967309045571324, 0.3449717045688069)
(1.9728482989438603, 0.7117941653907474, 0.6329271866609465)
feature_important_train_set6 = X_train_set6[['albedo', 'e', 'i',
'per', 'data_arc', 'condition_code', 'neo_Y', 'pha_Y', 'class_APO',
'class_AST']]
feature_important_test_set6 = X_test_set6[['albedo', 'e', 'i', 'per',
'data_arc', 'condition_code', 'neo_Y', 'pha_Y', 'class_APO',
'class_AST']]
```



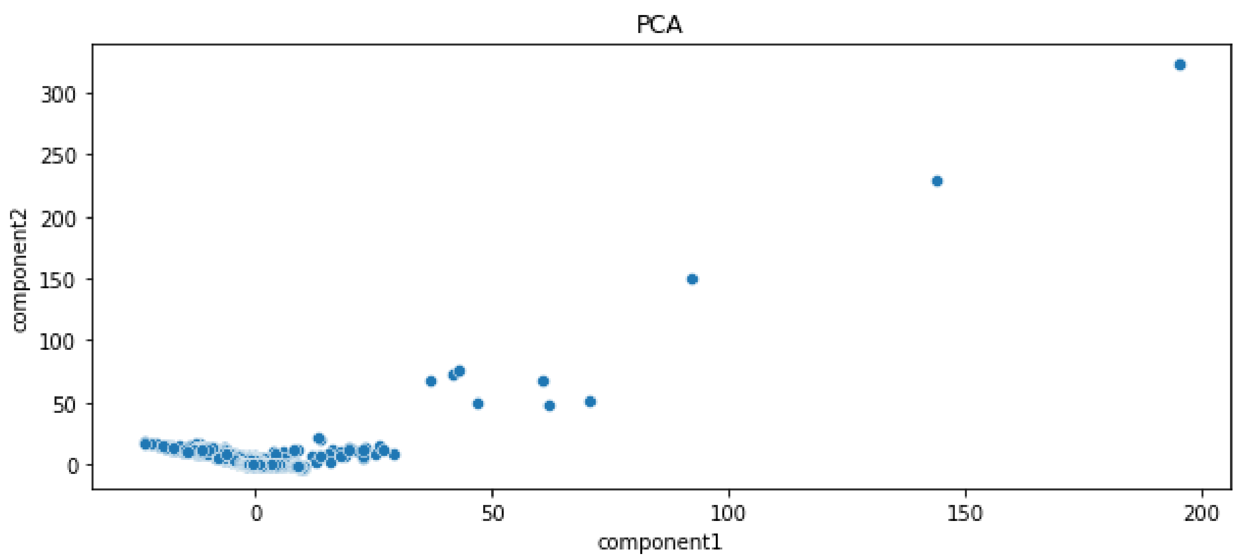

```
pca_cluster(X_train_class_featured1) % of
variance retained: 37.23243177441206
```



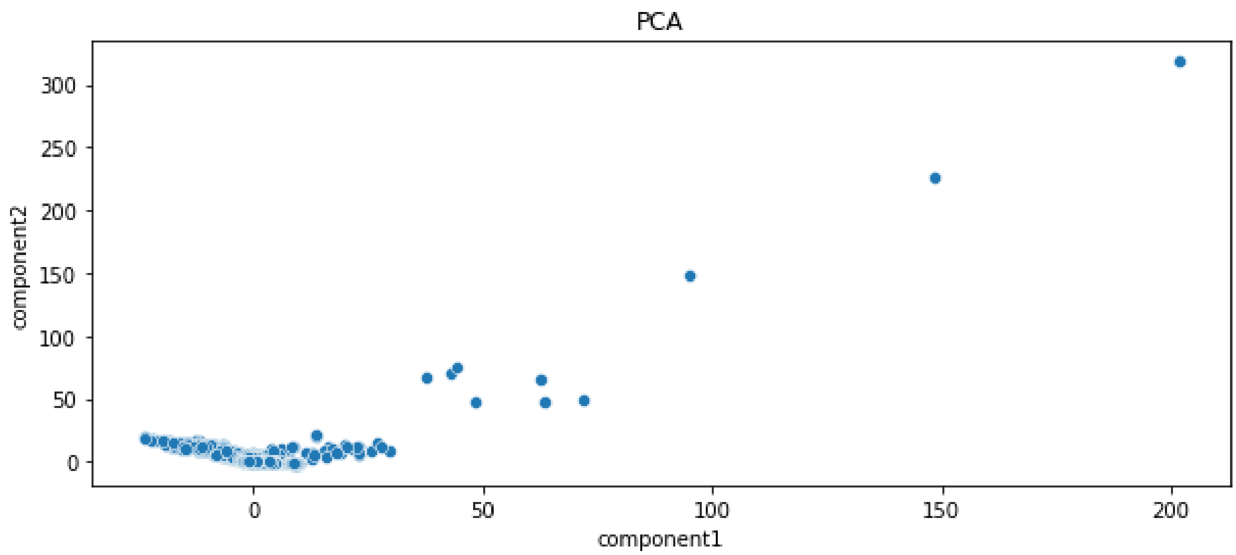

```
pca_cluster(X_train_class_featured2)
% of variance retained: 33.634210623025616
```



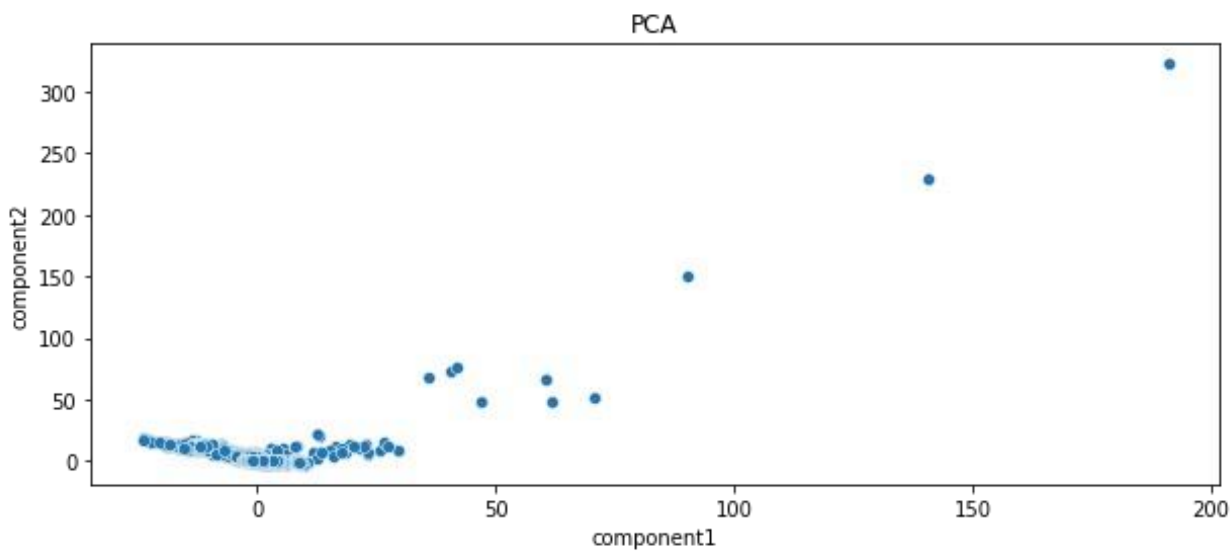

```
pca_cluster(X_train_H_featurized_opt) % of
variance retained: 28.857424384604112
```



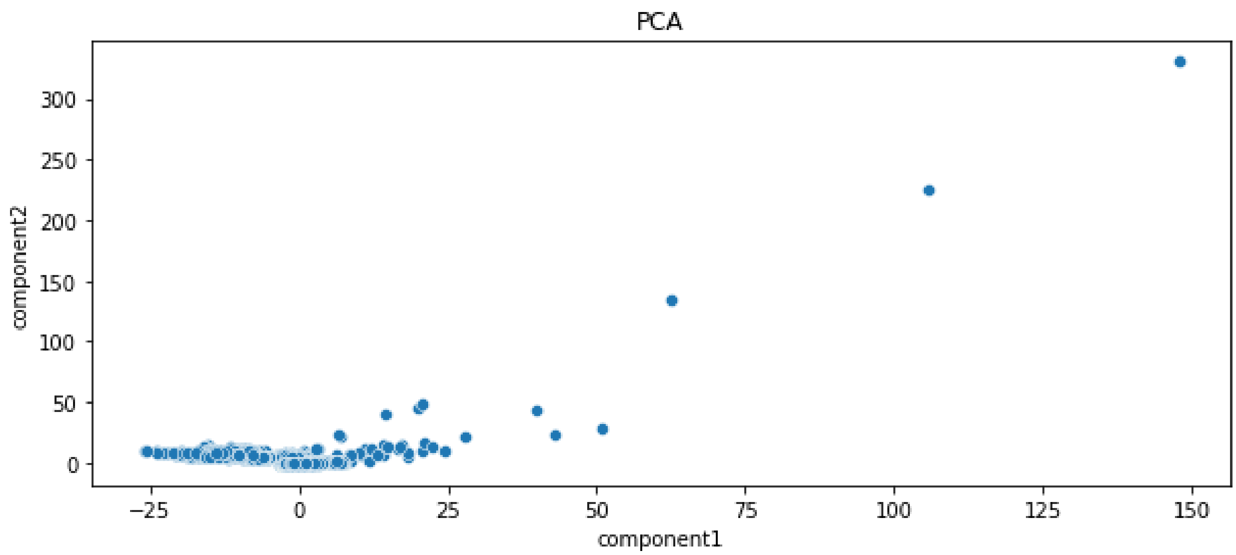

```
pca_cluster(X_train_H_featurized_1)
% of variance retained: 28.633234709651518
```



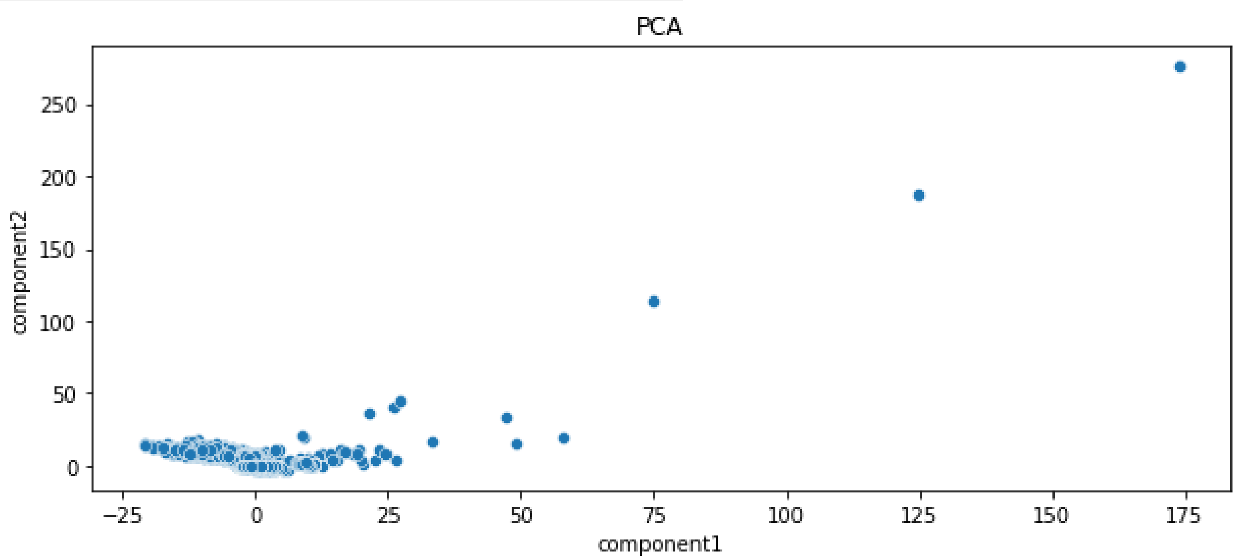

```
pca_cluster(X_train_H_featurized_2) % of

variance retained: 29.01003905550523
```



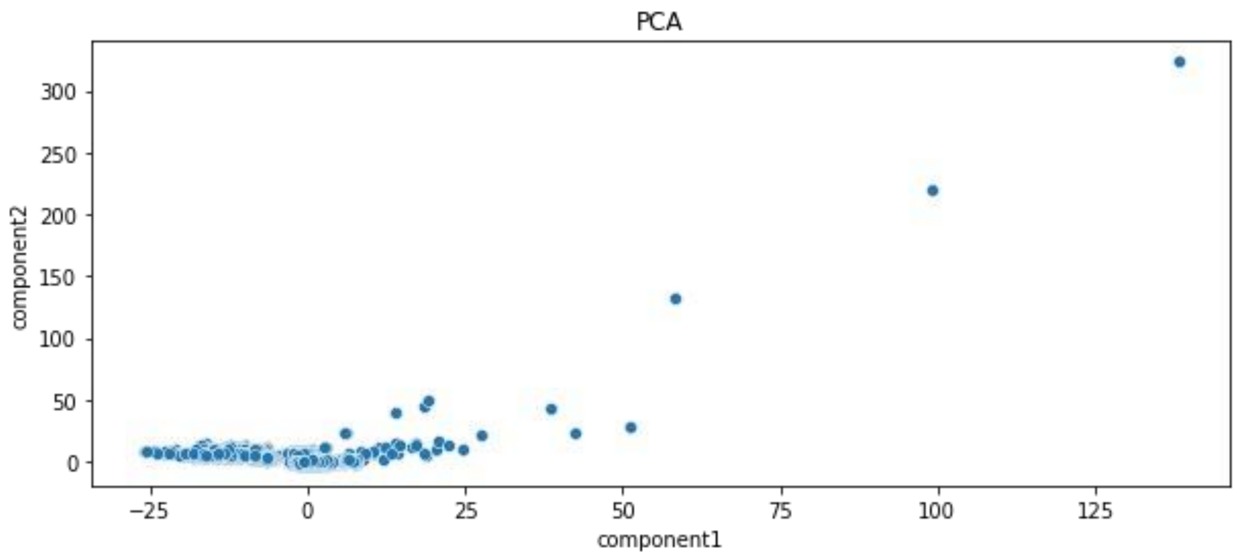

```
pca_cluster(X_train_featurized_1)

% of variance retained: 36.52261706538867
```




```
pca_cluster(X_train_featurized_2) % of

variance retained: 33.4043284316324
```




```
pca_cluster(X_train_class_featured1)

% of variance retained: 37.23243180993862
```

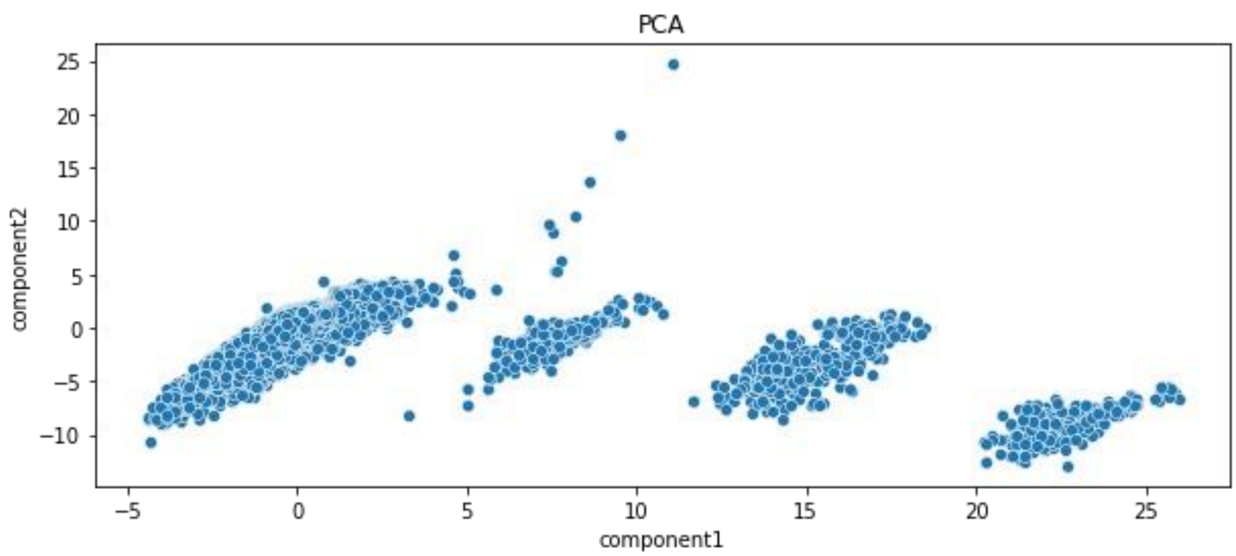


Observations:

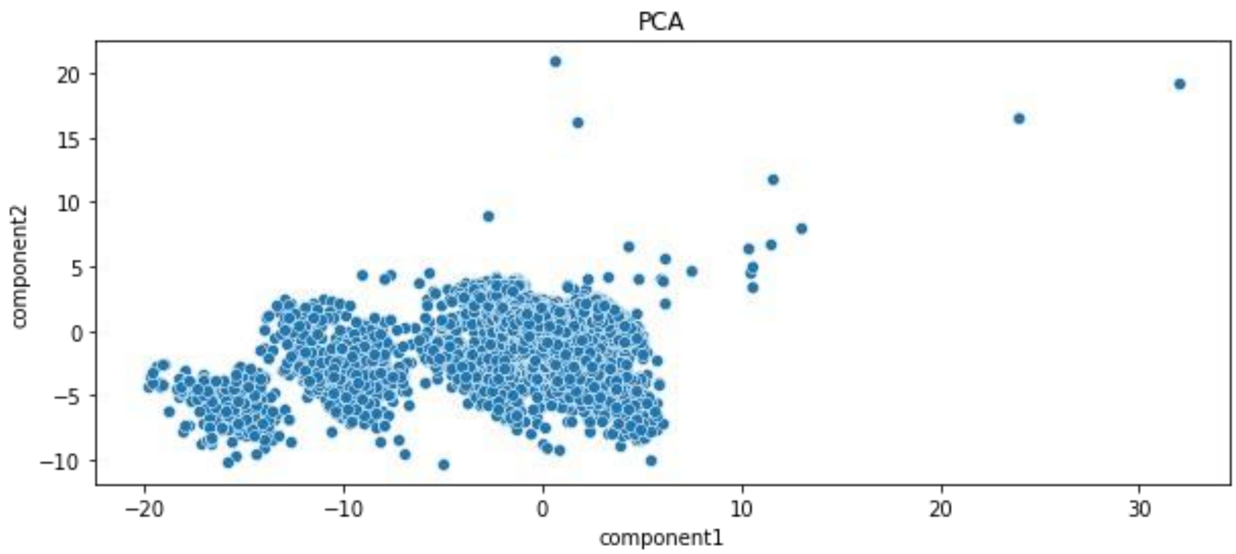
- we did not observe different distinct clusters in any of the sets PCA:

collinearity fixed datasets

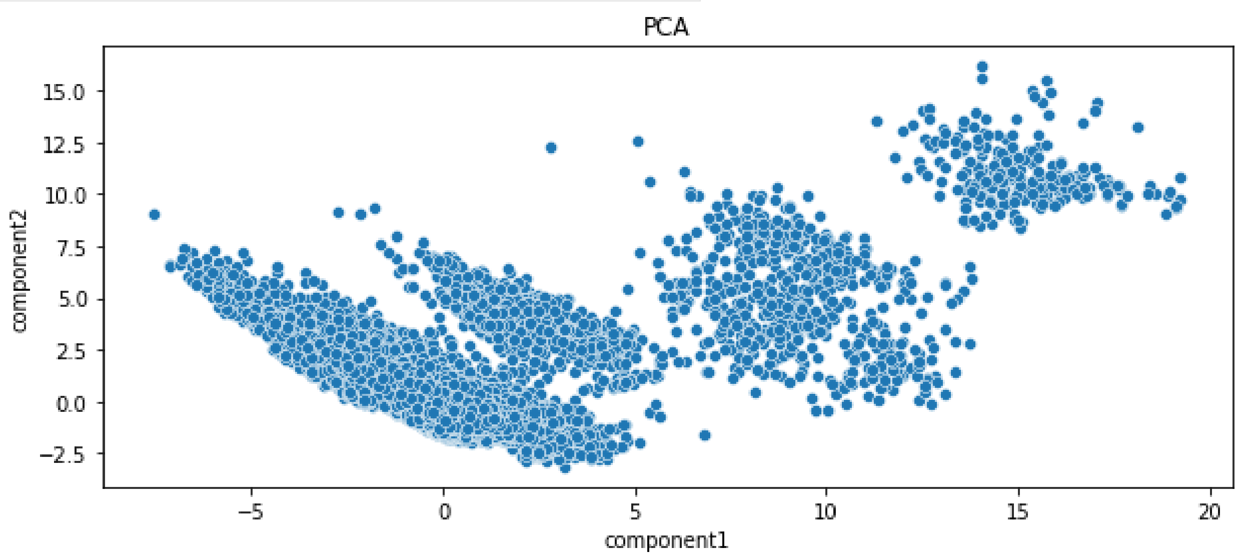
```
pca_cluster(X_train_set1)

% of variance retained: 25.144992868756866
```



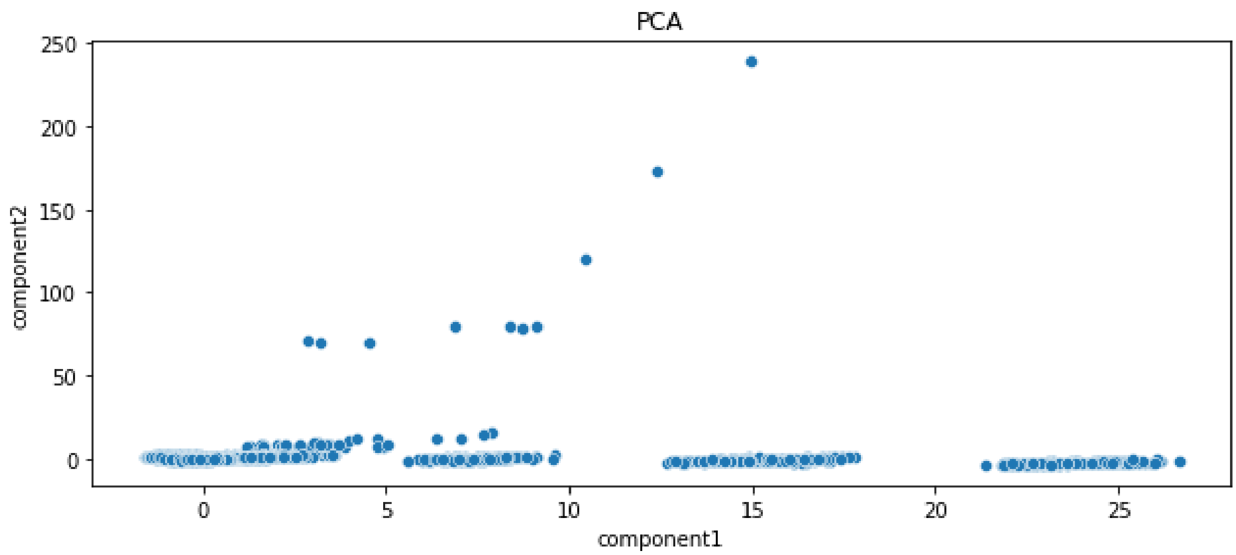

```
 pca_cluster(X_train_set2)

% of variance retained: 37.74952551504938
```



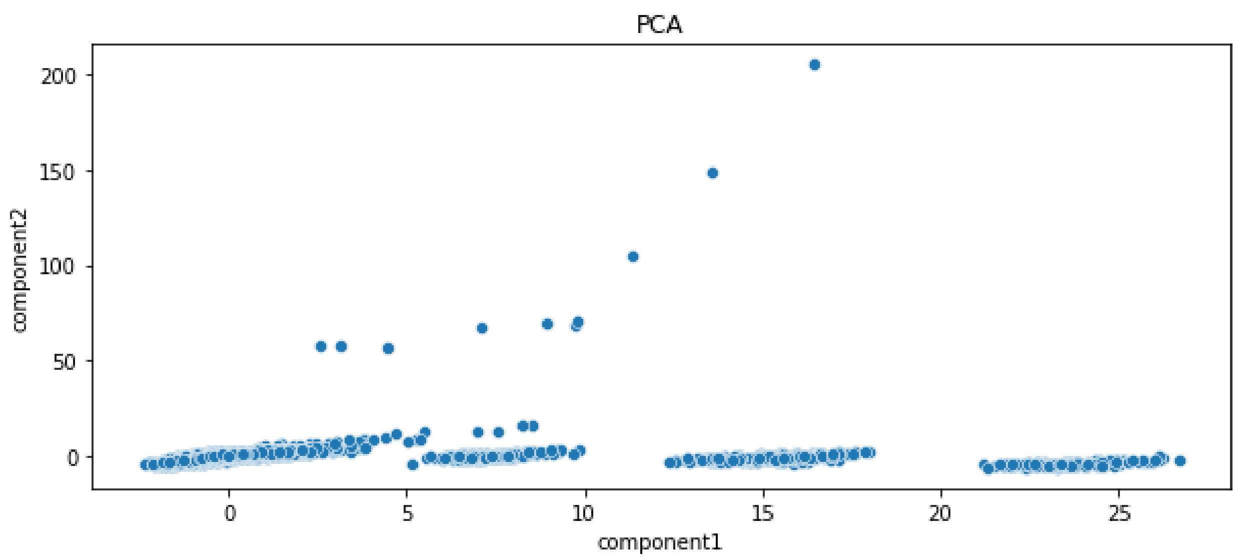

```
pca_cluster(X_train_set3) % of variance

retained: 32.21453398023109
```



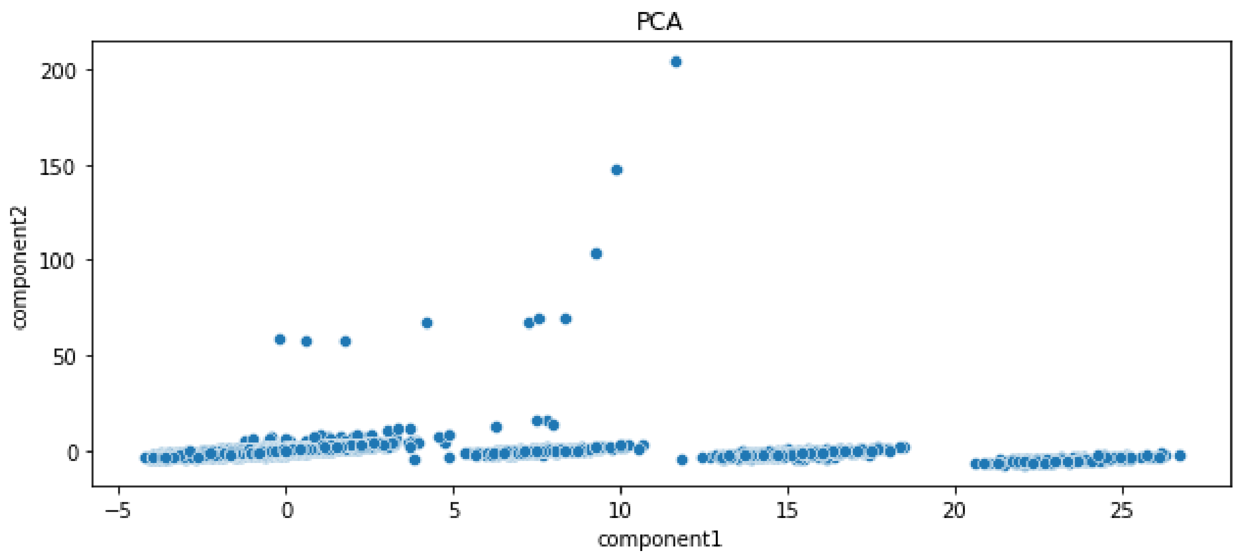

```
pca_cluster(X_train_set4)

% of variance retained: 25.0594184673352
```



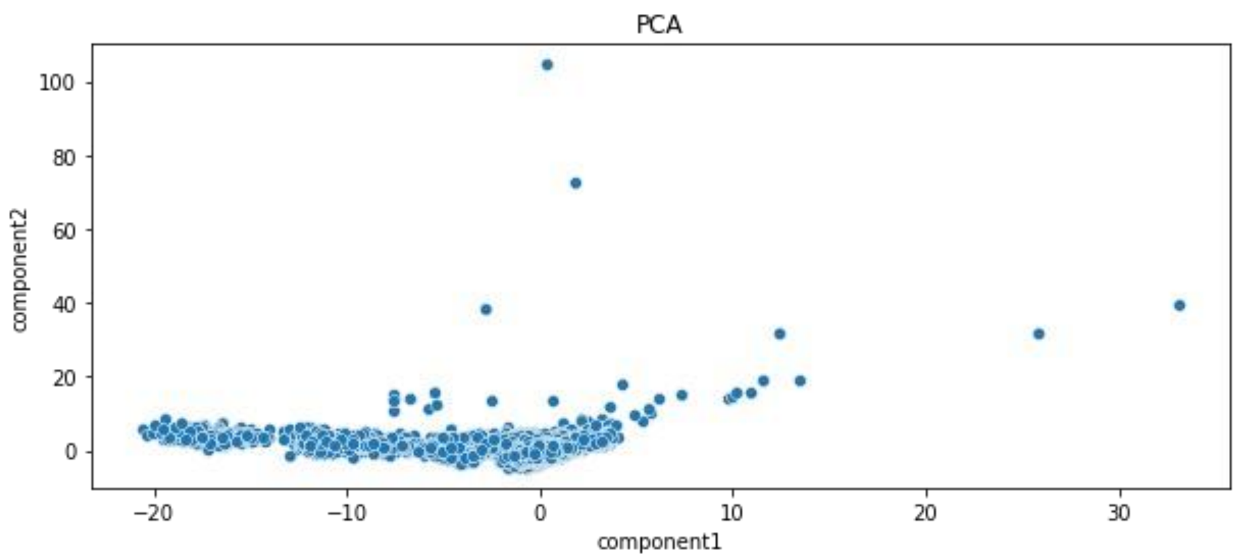

```
pca_cluster(X_train_set5) % of variance

retained: 23.766635123614062
```



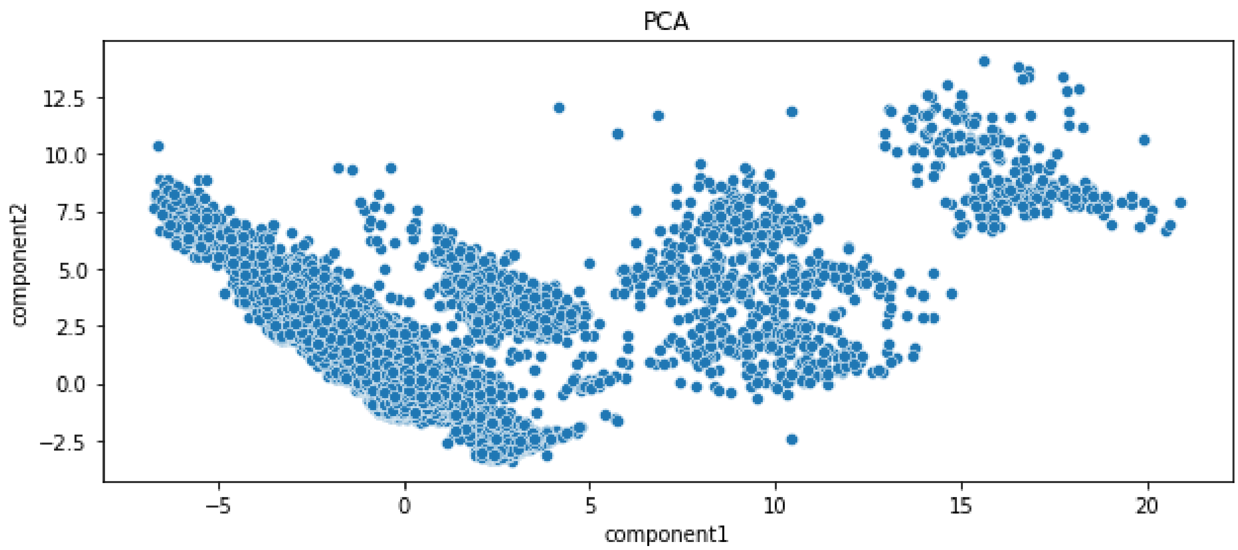

```
pca_cluster(X_train_set6)

% of variance retained: 24.38118155818627
```



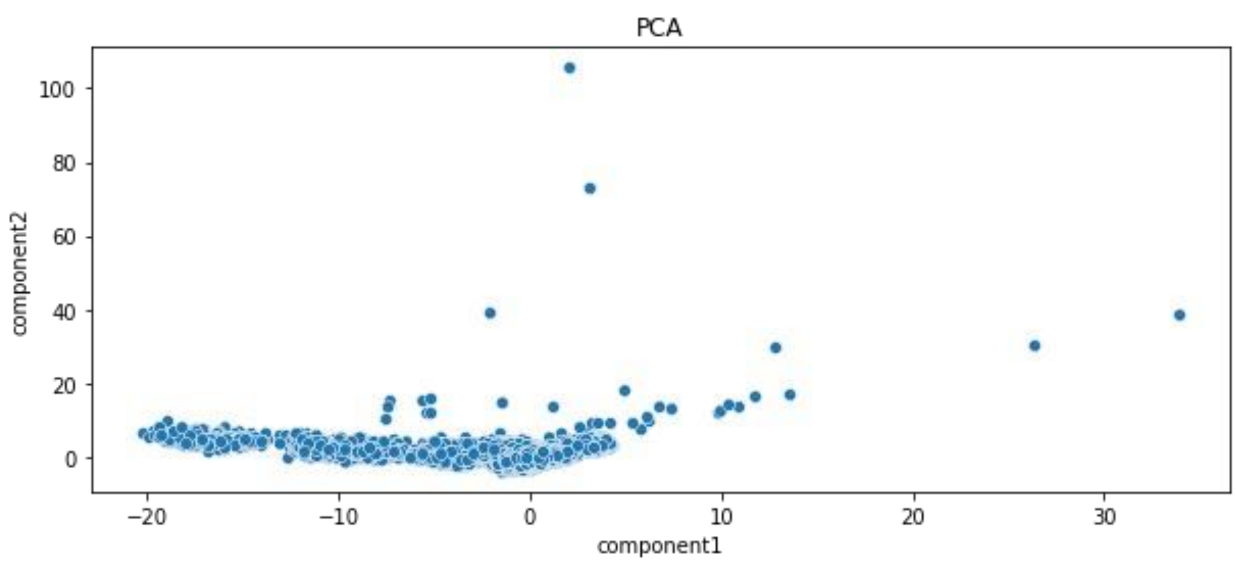

```
pca_cluster(X_train_set7) % of variance

retained: 35.58472360655176
```




```
pca_cluster(X_train_set8)

% of variance retained: 30.896865458693767
```




```
pca_cluster(X_train_set9)

% of variance retained: 35.91239741819779
```



Observation: Very interesting results

- Except for set7, all the set display distinct 4 clusters
- X_train_set1 clusters look great

```
#Importing all necessary libraries
import numpy as np import pandas
as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import missingno as msno
from prettytable import PrettyTable
from statsmodels.stats.outliers_influence import
variance_inflation_factor import pickle import
os import joblib
from sklearn.impute import KNNImputer
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
QuantileTransformer, PowerTransformer, MaxAbsScaler
from sklearn.metrics import mean_absolute_error,
median_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,
GradientBoostingRegressor, VotingRegressor from sklearn.neural_network
import MLPRegressor from sklearn.svm import SVR from xgboost import
XGBRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

#save dataframe to csv def
save_to_csv(dataframe, file_name):
path =
'/content/drive/MyDrive/project_asteroid/train_test_split/'
dataframe.to_csv(os.path.join(path, file_name), index=False)
## plotting feature importance def
plot_feature_imp(model, train_data, feature_set):
""" function to plot feature importance"""     fig,
ax = plt.subplots(figsize=(8,4))
    ax = sns.barplot(x=np.sort(model.feature_importances_)[::-1],
y=train_data.columns)     ax.set(xscale='log')
    ax.set_title(f'Feature Importance - {feature_set}')
plt.show()

test = pd.read_csv('/content/test1.csv', header=None)
test.head()
                0      1  2  3     4     5      6
7   \
0  Ceres (A801 AA)  Ceres  N  N  3.33  0.12  939.4  964.4 x 964.2 x
```

```
891.8
     8         9   ...       27           28       29    30     31     32
33   \
0   0.09   9.07417   ...   0.2142   2459920.37   1680.0   4.6   1.59   2.09
MB
       34   35        36
0   9520.0    0   0.43153
[1 rows x 37 columns]
```

Loading data

```
df =
pd.read_csv('/content/drive/MyDrive/project_asteroid/asteroids_final.c
sv')
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/
interactiveshell.py:3326: DtypeWarning: Columns (1,7,14,15) have mixed
types.Specify dtype option on import or set low_memory=False.
exec(code_obj, self.user_global_ns, self.user_ns) df.columns

Index(['full_name', 'name', 'neo', 'pha', 'H', 'G', 'diameter',
'extent',
       'albedo', 'rot_per', 'GM', 'BV', 'UB', 'IR', 'spec_B',
'spec_T',
       'H_sigma', 'diameter_sigma', 'epoch', 'e', 'a', 'q', 'i', 'om',
'w',
       'ma', 'ad', 'n', 'tp', 'per', 'per_y', 'moid', 'moid_jup',
'class',
       'data_arc', 'condition_code', 'rms'],
dtype='object')

df.iloc[:10, :].to_csv('/content/drive/MyDrive/test_cases/testcase')
```

Feature Selection based on EDA studies

```
# original dataset shape
df.shape
(139708, 37)
# creating a new dataframe from master
dfe = df.copy()
```

- drop features - as inferred from the Missing Value review(>80% data insuffieciency), and other reasons like data irrelevance
- drop 'w', 'ma', 'om' as understood from EDA feature correlation and feature importance studies
- Top most important features - albedo, H, data_arc, 'ad', 'a', 'q', 'e', moid. Orbit class, neo, pha too have strong corrlation with diameter

```
# dataframe dfe is a fresh copy from master df,
# removing features owing to feature irrelevance and missing value
analysis (missing>80% data)
dfe.drop(['full_name','name', 'extent', 'rot_per', 'G', 'GM', 'BV',
'UB', 'IR', 'spec_B', 'spec_T',
       'H_sigma', 'diameter_sigma', 'per_y'], axis=1, inplace=True)

# dropping features as inferred as irrelevant from EDA
dfe.drop([ 'w', 'ma', 'om'], axis=1, inplace=True)
```

Below is our to_drop list for our dataset

```
to_drop_columns = ['full_name','name', 'extent', 'rot_per', 'G', 'GM',
'BV', 'UB', 'IR', 'spec_B', 'spec_T',
       'H_sigma', 'diameter_sigma', 'per_y', 'w', 'ma', 'om']

# columns in the dataset
dfe.columns

Index(['neo', 'pha', 'H', 'diameter', 'albedo', 'epoch', 'e', 'a',
'q', 'i',
       'ad', 'n', 'tp', 'per', 'moid', 'moid_jup', 'class',
'data_arc',
       'condition_code', 'rms'],
dtype='object')

# checking shape of dataset dfe
dfe.shape

(139708, 20)
```

Train Test Split

```
# defining X, y
y = dfe.diameter
X = dfe.drop('diameter', axis=1)
```

To do a stratified splitting for the continuous target 'diameter', i am making use of verstack package tool:

- **verstack.stratified_continuous_split.scsplit**
- Reference - https://pypi.org/project/verstack/

```
! pip install verstack
{"type":"string"}
```

Splitting data into train and test, with test_size of 0.3

```
X.shape
(139708, 19)
from verstack.stratified_continuous_split import scsplit
X_train, X_test, y_train, y_test = scsplit(X, y, stratify=y,
test_size=0.3, random_state=0) #test_size default=0.3
X_train.shape, X_test.shape, y_train.shape, y_test.shape
((97795, 19), (41913, 19), (97795,), (41913,))
type(X_train), type(X_test), type(y_train), type(y_test)

(pandas.core.frame.DataFrame,
pandas.core.frame.DataFrame,
pandas.core.series.Series,
pandas.core.series.Series)

#save files
save_to_csv(X_train, 'X_train')
save_to_csv(X_test, 'X_test')
save_to_csv(y_train, 'y_train')
save_to_csv(y_test, 'y_test')

X_train =
pd.read_csv('/content/drive/MyDrive/project_asteroid/train_test_split/
X_train') X_test =
pd.read_csv('/content/drive/MyDrive/project_asteroid/train_test_split/
X_test')

X_train.shape, X_test.shape
((97795, 19), (41913, 19))
y_train =
pd.read_csv('/content/drive/MyDrive/project_asteroid/train_test_split/
y_train', squeeze=True) y_test =
pd.read_csv('/content/drive/MyDrive/project_asteroid/train_test_split/
y_test', squeeze=True) y_train.shape, y_test.shape

((97795,), (41913,))
```

```
Index(['neo', 'pha', 'class'], dtype='object')
num_cols
Index(['H', 'albedo', 'epoch', 'e', 'a', 'q', 'i', 'ad', 'n', 'tp',
'per',
       'moid', 'moid_jup', 'data_arc', 'condition_code', 'rms'],
      dtype='object')
X_train[cat_cols].head()
  neo pha class
0   N   N   MBA
1   N   N   MBA
2   N   N   MBA
3   N   N   MBA
4   N   N   MBA
# applying one hot encoding ohe =
OneHotEncoder(drop='first')
ohe.fit(X_train[cat_cols])
X_train_cat_cols = ohe.transform(X_train[cat_cols]).toarray()
X_test_cat_cols = ohe.transform(X_test[cat_cols]).toarray()

import pickle
pickle.dump(ohe,
open('/content/drive/MyDrive/project_asteroid/train_test_split/ohe',
'wb'))

ohe.get_feature_names_out()
array(['neo_Y', 'pha_Y', 'class_APO', 'class_AST', 'class_ATE',
       'class_CEN', 'class_IMB', 'class_MBA', 'class_MCA'
'class_OMB',
       'class_TJN', 'class_TNO'], dtype=object)
```

We need to concatenate encoded Categorical columns with the numerical set

```
X_train_encoded = np.hstack((X_train[num_cols].values,
X_train_cat_cols))
X_test_encoded = np.hstack((X_test[num_cols].values, X_test_cat_cols))

X_train_encoded.shape,  X_test_encoded.shape
((97795, 28), (41913, 28))
# creating dataframes of these outputs
ncols = list(X_train[num_cols].columns)
ccols = list(ohe.get_feature_names_out())
ncols.extend(ccols) print(ncols)
```

```
['H', 'albedo', 'epoch', 'e', 'a', 'q', 'i', 'ad', 'n', 'tp', 'per',
'moid', 'moid_jup', 'data_arc', 'condition_code', 'rms', 'neo_Y',
'pha_Y', 'class_APO', 'class_AST', 'class_ATE', 'class_CEN',
'class_IMB', 'class_MBA', 'class_MCA', 'class_OMB', 'class_TJN',
'class_TNO']

#converting into dataframes
X_train_encoded_df = pd.DataFrame(X_train_encoded, columns=ncols)
X_test_encoded_df = pd.DataFrame(X_test_encoded, columns=ncols)
X       H  albedo      epoch       e      a      q      i    ad       n

   14.73   0.100  2459800.5  0.0437  3.103  2.967   8.63  3.24  0.1803



\
0

1  17.33   0.071  2459800.5  0.1538  2.988  2.529  12.74  3.45  0.1908

2  14.40   0.207  2459800.5  0.0393  3.098  2.976  11.80  3.22  0.1808

3  13.67   0.178  2459800.5  0.1505  2.628  2.233  13.52  3.02  0.2313
4  13.74   0.077  2459800.5  0.1450  3.133  2.678   6.30  3.59  0.1778
           tp  ...  class_APO  class_AST  class_ATE  class_CEN
class_IMB  \
0  2459704.30  ...        0.0        0.0        0.0        0.0
0.0
1  2459459.77  ...        0.0        0.0        0.0        0.0
0.0
2  2459605.94  ...        0.0        0.0        0.0        0.0
0.0
3  2459501.23  ...        0.0        0.0        0.0        0.0
0.0
4  2459220.69  ...        0.0        0.0        0.0        0.0
0.0
   class_MBA  class_MCA  class_OMB  class_TJN  class_TNO
0        1.0        0.0        0.0        0.0        0.0
1        1.0        0.0        0.0        0.0        0.0
2        1.0        0.0        0.0        0.0        0.0
3        1.0        0.0        0.0        0.0        0.0
4        1.0        0.0        0.0        0.0        0.0
[5 rows x 28 columns]
save_to_csv(X_train_encoded_df, 'X_train_encoded_df')
save_to_csv(X_test_encoded_df, 'X_test_encoded_df')
```

We see that we have now only data_arc feature with missing values, we will impute it with median startegy

```
dataarc_before_imp = X_train_sim_imp[['data_arc']].describe()

#creating median imputer object and fitting and transforming data

median_imp = SimpleImputer(strategy ='median')
median_imp.fit(X_train_sim_imp)
X_train_sim_imp = median_imp.transform(X_train_sim_imp)
X_test_sim_imp = median_imp.transform(X_test_sim_imp)

pickle.dump(median_imp,
open('/content/drive/MyDrive/project_asteroid/train_test_split/deploy/
median_imp', 'wb'))

# Converting th eimputed numpy arrays to dataframe
X_train_sim_imp_df= pd.DataFrame(X_train_sim_imp,
columns=X_train_encoded_df.columns)
X_test_sim_imp_df = pd.DataFrame(X_test_sim_imp,
columns=X_train_encoded_df.columns)

# checking "data_arc" statistics before and after imputation

dataarc_after_imp     =     X_train_sim_imp_df[['data_arc']].describe()
pd.DataFrame({'data_arc_before': dataarc_before_imp.data_arc,
'data_arc_after': dataarc_after_imp.data_arc},
index=dataarc_after_imp.index)
```

	data_arc_before	data_arc_after
count	97776.000000	97795.000000
mean	10139.518154	10139.208170
std	5934.493159	5933.958303
min	1.000000	1.000000
25%	7293.000000	7293.000000
50%	8544.000000	8544.000000
75%	10716.000000	10716.000000
max	63747.000000	63747.000000

Observations:

- data_arc stats before and after imputation looks closer

```
X_test_sim_imp_df.head()
          H  albedo      epoch       e      a      q      i    ad
n  \
0  21.87000   0.110  2459800.5  0.1447  1.015  0.868  11.85  1.16
0.964
0
1  15.35697   0.127  2455364.5  0.2174  1.977  1.548  27.12  2.41
0.354
4
2  14.80000   0.063  2459800.5  0.1092  3.096  2.758  15.13  3.43
mean_squared_error for feature 'data_arc' imputed: 33597085.012547284
======================================================================
```

```
================================
Validation results for knn_model with n_neighbours:25
MEAN SQUARED ERROR for scaled y_pred and y_true
mean_squared_error for feature 'H' imputed: 1.0596882085732418
mean_squared_error for feature 'albedo' imputed: 1.013696218542176
mean_squared_error for feature 'data_arc' imputed: 0.9539789083979574

MEAN SQUARED ERROR for unscaled y_pred and y_true mean_squared_error
for feature 'H' imputed: 2.1459415898678933 mean_squared_error for
feature 'albedo' imputed: 0.012282966041193057 mean_squared_error for
feature 'data_arc' imputed: 33597085.012547284
======================================================================
================================
Validation results for knn_model with n_neighbours:50
MEAN SQUARED ERROR for scaled y_pred and y_true
mean_squared_error for feature 'H' imputed: 1.0436228773390182
mean_squared_error for feature 'albedo' imputed: 1.0136353138121952
mean_squared_error for feature 'data_arc' imputed: 0.9539789083979574

MEAN SQUARED ERROR for unscaled y_pred and y_true mean_squared_error
for feature 'H' imputed: 2.1134081878996462 mean_squared_error for
feature 'albedo' imputed: 0.012282228058041479 mean_squared_error for
feature 'data_arc' imputed: 33597085.012547284
======================================================================
================================
```

Data_arc feature mse looks wierd, trying below to experiment with KNNeighborsRegressor and plotting the errors. Considering here k=5 as from KNNImputer, we don't see much improvement with different K-values tried.

```
from sklearn.neighbors import KNeighborsRegressor

def knn_imputation(feature):
      X_train_enc = X_train_encoded_df.copy()
      train = X_train_enc[X_train_enc[feature].notnull()]
test = X_train_enc[X_train_enc[feature].isnull()]
test_indices =
X_train_enc[X_train_enc['data_arc'].isnull()].index.tolist()

      #defining y_train
      y_train_feature = train[feature]
      #print(y_train_feature)
      #dropping feature from train and test
      train.drop(['H', 'albedo', 'data_arc'], axis=1, inplace=True)
test.drop(['H', 'albedo', 'data_arc'], axis=1, inplace=True)
      #scaling data
```