# Random forest usimg grid search

## Methodology

- **Data Cleaning:** Checking for null values and based on their number either droping them or replacing with mean, median, mode based on the type and description of data. Droping decscrete and catagorical variables that have highly skewed histograms.
- **Data Visualization:** This step helps understand the understand the data in a visually. We can understand normality of the data as well. This helps us to decide whether to normalize the data. In case of catagorical variables it also helps in feature selection.
- **Feature Selection:** Based on the Pearson correlation between the labeled column and rest of the features. In general, a very great correlation should have an absolute value greater than 0.75. When the labeled column is depended on multiple columns, the correlation with one column may be less. But combined features may have higher effect.
- **Train Test Split:** We split the data into 80:20 ratio for tarining testing respectively.
- **Model Selection:** Based on the data visualization and data correlation, we need to select a model that would best suit. Here we need to use XGBOOST.
- **Evalution:** In this case we are using RMSE, R2 Score to determine the accuracy of the predicting model.

### importing libraries

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

### Reading data

```
df=pd.read_csv(r"drive/My Drive/Asteroid_Updated.csv")
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718: DtypeWarning: Columns (0,10,15,16,23,24) have mixed types.Specify dtype option on import or set low_memory=Fa
  interactivity=interactivity, compiler=compiler, result=result)
```

### Null value percentages

```
Null=[]
for i in df:
    Null.append((i,df[i].isna().mean()*100))
Null=pd.DataFrame(Null,columns=['class','per'])
Null
```

| | class | per |
|---|---|---|
| 0 | name | 97.383990 |
| 1 | a | 0.000238 |
| 2 | e | 0.000000 |
| 3 | i | 0.000000 |
| 4 | om | 0.000000 |
| 5 | w | 0.000000 |
| 6 | q | 0.000000 |
| 7 | ad | 0.000715 |
| 8 | per_y | 0.000119 |
| 9 | data_arc | 1.842770 |
| 10 | condition_code | 0.103249 |
| 11 | n_obs_used | 0.000000 |
| 12 | H | 0.320228 |
| 13 | neo | 0.000715 |
| 14 | pha | 1.958048 |
| 15 | diameter | 83.609181 |
| 16 | extent | 99.997856 |
| 17 | albedo | 83.755302 |
| 18 | rot_per | 97.761619 |
| 19 | GM | 99.998333 |
| 20 | BV | 99.878411 |
| 21 | UB | 99.883413 |
| 22 | IR | 99.999881 |
| 23 | spec_B | 99.801599 |
| 24 | spec_T | 99.883294 |
| 25 | G | 99.985829 |
| 26 | moid | 1.958048 |

## Columns with percentage of null values Greater than 85%

```
l=Null[Null['per']>85]['class']
Null[Null['per']>85]
```

| | class | per |
|---|---|---|
| **0** | name | 97.383990 |
| **16** | extent | 99.997856 |
| **18** | rot_per | 97.761619 |
| **19** | GM | 99.998333 |
| **20** | BV | 99.878411 |

These columns will have very less impact on the decision since most of their values are null.

We can drop them

```
df=df.drop(l,axis=1)
df
```

| | a | e | i | om | w | q | ad | per_y | data_arc | condition_code | n_obs_used | H | neo | pha | diameter | albedo | moid | class | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2.769165 | 0.076009 | 10.594067 | 80.305532 | 73.597694 | 2.558684 | 2.979647 | 4.608202 | 8822.0 | 0 | 1002 | 3.340 | N | N | 939.4 | 0.0900 | 1.594780 | MBA | 0.213885 |
| **1** | 2.772466 | 0.230337 | 34.836234 | 173.080063 | 310.048857 | 2.133865 | 3.411067 | 4.616444 | 72318.0 | 0 | 8490 | 4.130 | N | N | 545 | 0.1010 | 1.233240 | MBA | 0.213503 |
| **2** | 2.669150 | 0.256942 | 12.988919 | 169.852760 | 248.138626 | 1.983332 | 3.354967 | 4.360814 | 72684.0 | 0 | 7104 | 5.330 | N | N | 246.596 | 0.2140 | 1.034540 | MBA | 0.226019 |
| **3** | 2.361418 | 0.088721 | 7.141771 | 103.810804 | 150.728541 | 2.151909 | 2.570926 | 3.628837 | 24288.0 | 0 | 9325 | 3.200 | N | N | 525.4 | 0.4228 | 1.139480 | MBA | 0.271609 |
| **4** | 2.574249 | 0.191095 | 5.366988 | 141.576605 | 358.687607 | 2.082324 | 3.066174 | 4.130323 | 63507.0 | 0 | 2916 | 6.850 | N | N | 106.699 | 0.2740 | 1.095890 | MBA | 0.238632 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **839709** | 2.812945 | 0.664688 | 4.695700 | 183.310012 | 234.618352 | 0.943214 | 4.682676 | 4.717914 | 17298.0 | 0 | 118 | 20.400 | Y | Y | NaN | NaN | 0.032397 | APO | 0.208911 |
| **839710** | 2.645238 | 0.259376 | 12.574937 | 1.620020 | 339.568072 | 1.959126 | 3.331350 | 4.302346 | 16.0 | 9 | 15 | 17.507 | N | N | NaN | NaN | 0.956145 | MBA | 0.229090 |
| **839711** | 2.373137 | 0.202053 | 0.732484 | 176.499082 | 198.026527 | 1.893638 | 2.852636 | 3.655884 | 5.0 | 9 | 6 | 18.071 | N | N | NaN | NaN | 0.893896 | MBA | 0.269600 |
| **839712** | 2.260404 | 0.258348 | 9.661947 | 204.512448 | 148.496988 | 1.676433 | 2.844376 | 3.398501 | 10.0 | 9 | 13 | 18.060 | N | N | NaN | NaN | 0.680220 | MBA | 0.290018 |
| **839713** | 2.546442 | 0.287672 | 5.356238 | 70.709555 | 273.483265 | 1.813901 | 3.278983 | 4.063580 | 11.0 | 9 | 11 | 17.406 | N | N | NaN | NaN | 0.815280 | MBA | 0.242551 |

839714 rows × 21 columns

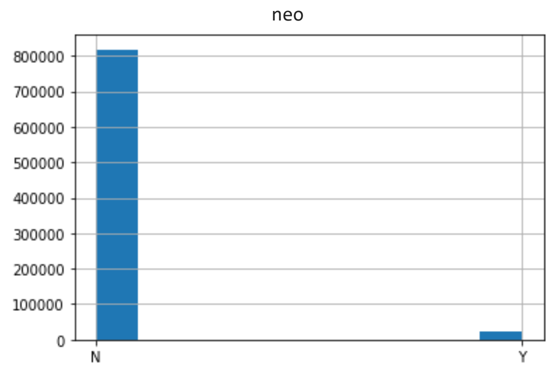
```
df.dtypes
```

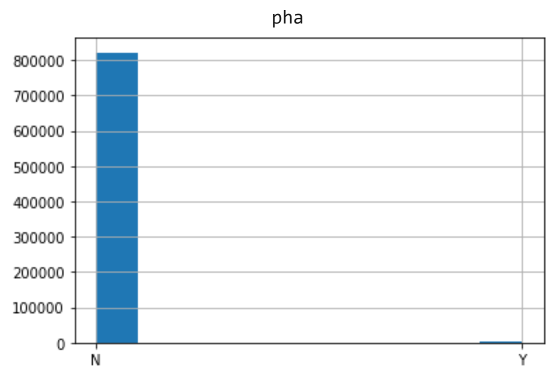
```
a                  float64
e                  float64
i                  float64
```

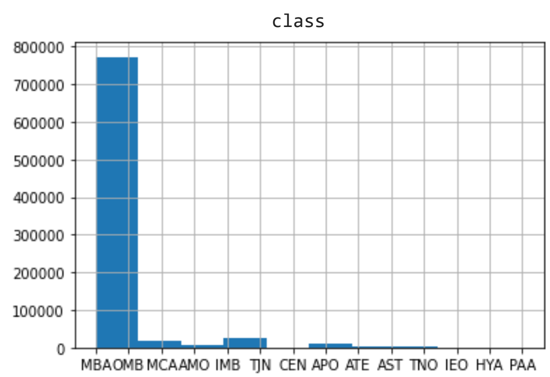
## Catagorical columns with skewed *distribution*

```
q                  float64
```

```
k=['neo','pha','class']
for i in k:
  print("\t\t\t",i)
  df[i].hist()
  plt.show()
```




Since the most values have only one catagory so, the impact of rest of the catagories is not significant, and droping them will not effect the decision variable.

```
df=df.drop(k,axis=1)
```

```
df.dtypes
```

```
a                 float64
e                 float64
i                 float64
om                float64
w                 float64
q                 float64
ad                float64
per_y             float64
data_arc          float64
condition_code     object
n_obs_used          int64
H                 float64
diameter           object
albedo            float64
moid              float64
n                 float64
per               float64
ma                float64
dtype: object
```

## Handling diameter columns data

```
df.diameter.unique()
```

```
array(['939.4', '545', '246.596', ..., 0.122, 0.6509999999999999, 1.077],
      dtype=object)
```

The data is in both numeric and catagorical values, we need to convert it to int data type

```
df['diameter']= df.diameter.astype(float,errors='ignore')
```

```
df.diameter.unique()
```

```
array([9.39400e+02, 5.45000e+02, 2.46596e+02, ..., 1.22000e-01,
       6.51000e-01, 1.07700e+00])
```

## Droping null values

```
df.isna().sum()/len(df)*100
```

```
a                  0.000238
```

there are null values in predicting column and it is a high number filling them increases the error in the data so, dropping them is only the option.

```
w                  0.000000
```

```
df = df[df['diameter'].notna()]
```

```
per_y              0.000119
```

fill the remaining with median value

```
n_obs_used         0.000000
```

```
df=df.fillna(df.mean())
df.isna().sum()
```

```
a                 0
e                 0
i                 0
om                0
w                 0
q                 0
ad                0
per_y             0
data_arc          0
condition_code    0
n_obs_used        0
H                 0
diameter          0
albedo            0
moid              0
n                 0
per               0
ma                0
dtype: int64
```

## Handling condition_code columns data

```
df['condition_code'].unique()
```

```
array([0, 1, 3, 2, '0', '1', '2', '3', '4', '5', '9', '7', 5.0, 6.0, 4.0,
       7.0, 9.0, 8.0, '8', '6'], dtype=object)
```

```
df['condition_code']= df.condition_code.astype(int)
```

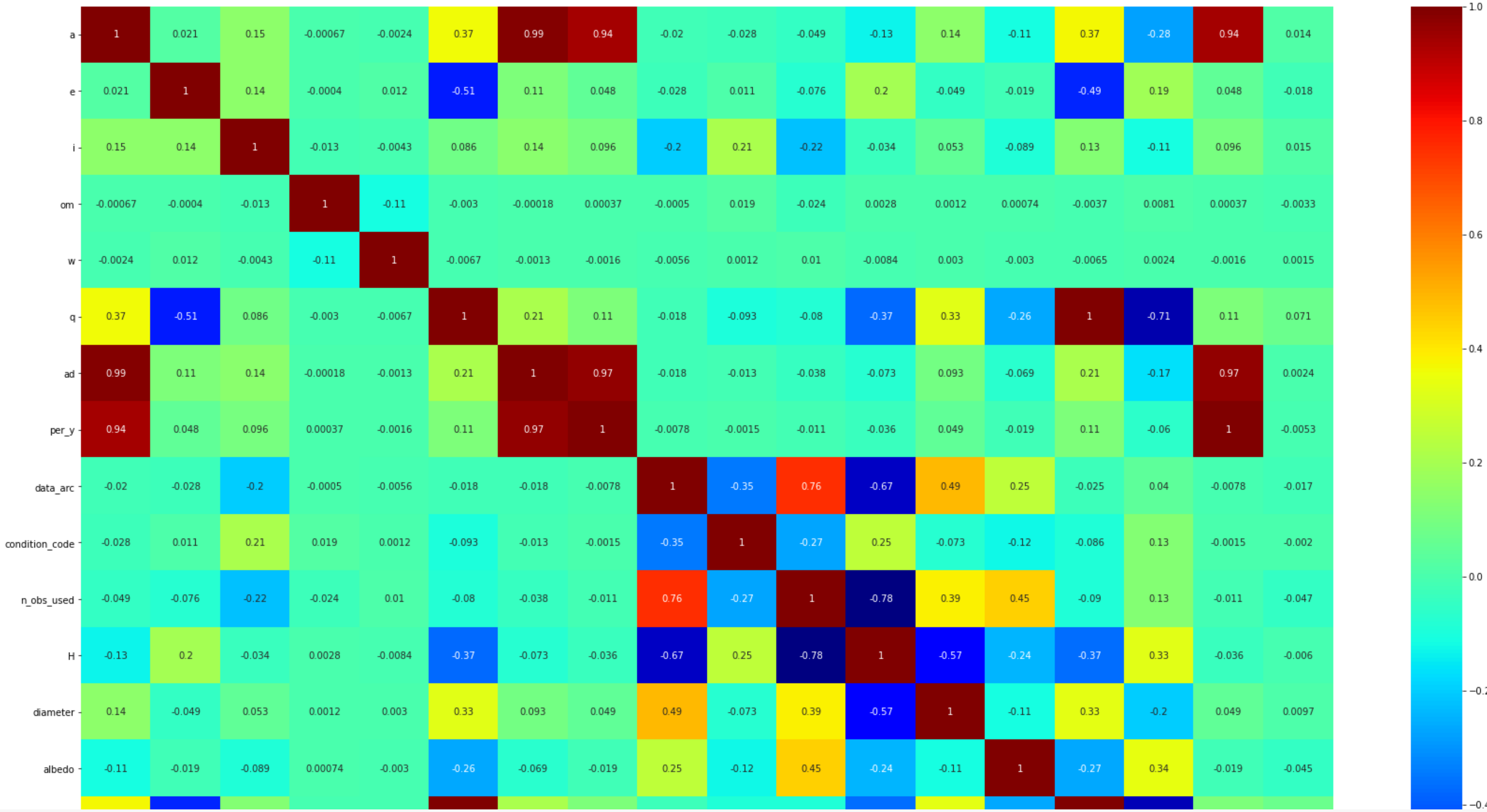
```
df['condition_code'].unique()
```

```
array([0, 1, 3, 2, 4, 5, 9, 7, 6, 8])
```

## Feature

```
import seaborn as sns
plt.figure(figsize=(30,20))
sns.heatmap(df.corr(),annot = True,cmap="jet")
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
<matplotlib.axes._subplots.AxesSubplot at 0x7f3d67f18ef0>
```




```
abs(df.corr()['diameter']*100).sort_values(ascending=False)
```

```
diameter          100.000000
H                  56.849287
data_arc           49.157968
```

15 highly correlated features

```
n                  20.102275
```

```
l=dict(abs(df.corr()['diameter']*100).sort_values(ascending=False)[:-3]).keys()
l
```

```
dict_keys(['diameter', 'H', 'data_arc', 'n_obs_used', 'moid', 'q', 'n', 'a', 'albedo', 'ad', 'condition_code', 'i', 'e', 'per', 'per_y'])
e                   4.913337
```

```
df=df[l]
```

```
ma                  0.965894
```

## Train Test Split

```
cols = [col for col in df.columns if col not in ["diameter"]]
X = df[cols]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df, df['diameter'], test_size=0.2)
```

## XGBOOST Model

```
from xgboost import XGBRegressor
```

```
model = XGBRegressor(max_depth=8,n_jobs=6,booster='dart')
model.fit(X_train, y_train)
```

```
[19:25:16] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGBRegressor(base_score=0.5, booster='dart', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=8, min_child_weight=1, missing=None, n_estimators=100,
             n_jobs=6, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

## R2 Score on Test and Train set

```
model.score(X_train, y_train)
```

```
0.9999974603797344
```

```
model.score(X_test, y_test)
```

```
0.9205378495840809
```