In [ ]:
```
# Import needed libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
from scipy import stats
```

In [ ]:
```
from urllib.request import urlretrieve
urlretrieve("https://raw.githubusercontent.com/blakelobato/Predicting-Asteroid-Diar
```

Out[ ]:
```
('Pred_Ast_Diam_2.csv', <http.client.HTTPMessage at 0x20e52289520>)
```

In [ ]:
```
from tabulate import tabulate
plt.figure()
df = pd.read_csv("Pred_Ast_Diam_2.csv")
df = df.fillna(np.nan,axis=0)
print(df.shape)
df.head(25)
```

```
(126497, 23)
```

Out[ ]:

	orbit_id	e	a	i	om	w	ma	n	
0	JPL 35	0.242027	2.201791	2.536221	313.311389	18.989048	301.072249	0.301675	2.45879
1	JPL 25	0.256856	2.338209	22.326589	10.489602	105.115594	87.454449	0.275663	2.45828
2	JPL 28	0.160543	2.228812	1.747387	121.579382	252.465454	208.942016	0.296206	2.45911
3	JPL 35	0.167945	2.241299	2.428619	161.636895	172.846491	20.350289	0.293734	2.45853
4	JPL 34	0.253295	2.467536	6.757106	137.130656	259.158793	127.366908	0.254278	2.45810
5	JPL 67	0.073742	1.944104	22.508840	175.320955	124.031963	224.445860	0.363601	2.45897
6	JPL 34	0.103066	2.244712	5.995089	203.399440	264.392525	244.456912	0.293064	2.45899
7	JPL 29	0.110058	2.230630	5.389819	72.373045	354.339267	127.022318	0.295844	2.45817
8	JPL 29	0.239092	2.253449	5.680974	336.764958	76.779174	181.329773	0.291362	2.45921
9	JPL 33	0.353074	2.628043	32.583941	155.112383	76.773043	169.922720	0.231342	2.45786
10	JPL 36	0.127957	2.220819	1.710039	41.005619	18.012785	213.767825	0.297807	2.45909
11	JPL 27	0.213370	2.324423	6.902487	302.807248	38.110992	184.455090	0.278120	2.45923
12	JPL 27	0.194725	2.441702	7.331119	254.679340	175.697462	32.854176	0.258324	2.45847
13	JPL 34	0.278983	2.545744	12.715483	357.019751	349.524955	156.468717	0.242651	2.45795
14	JPL 31	0.137261	2.174837	2.458112	213.756930	174.868642	291.496345	0.307301	2.45882
15	JPL 34	0.107869	2.180337	4.273327	281.903596	90.788273	301.127993	0.306139	2.45879
16	JPL 33	0.195356	2.237341	6.089728	357.062379	300.173213	359.791080	0.294514	2.45860
17	JPL 30	0.144933	2.171862	5.638375	45.681873	257.356353	284.562935	0.307933	2.45884
18	JPL 25	0.250754	2.454325	11.712953	204.245269	205.874108	329.451449	0.256334	2.45872
19	JPL 37	0.088155	2.253581	4.257320	82.642185	55.099876	249.109885	0.291336	2.45898
20	JPL 25	0.166604	2.240171	4.082515	289.823551	89.008528	161.897890	0.293956	2.45805
21	JPL 27	0.284648	2.545009	5.533335	326.300184	71.906169	329.714683	0.242756	2.45872
22	JPL 31	0.078581	2.195946	5.202225	7.826440	115.615591	43.781770	0.302881	2.45845
23	JPL 73	0.191806	2.283007	7.144705	154.518262	196.370783	2.281690	0.285722	2.45859
24	JPL 37	0.180236	2.178945	2.611783	152.694879	195.861723	75.692806	0.306432	2.45835

25 rows × 23 columns

```
<Figure size 640x480 with 0 Axes>
```

In [ ]:

```
import pandas_profiling
df.profile_report()
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	23
Number of observations	126497
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	22.2 MiB
Average record size in memory	184.0 B

Variable types

Categorical	3
Numeric	20

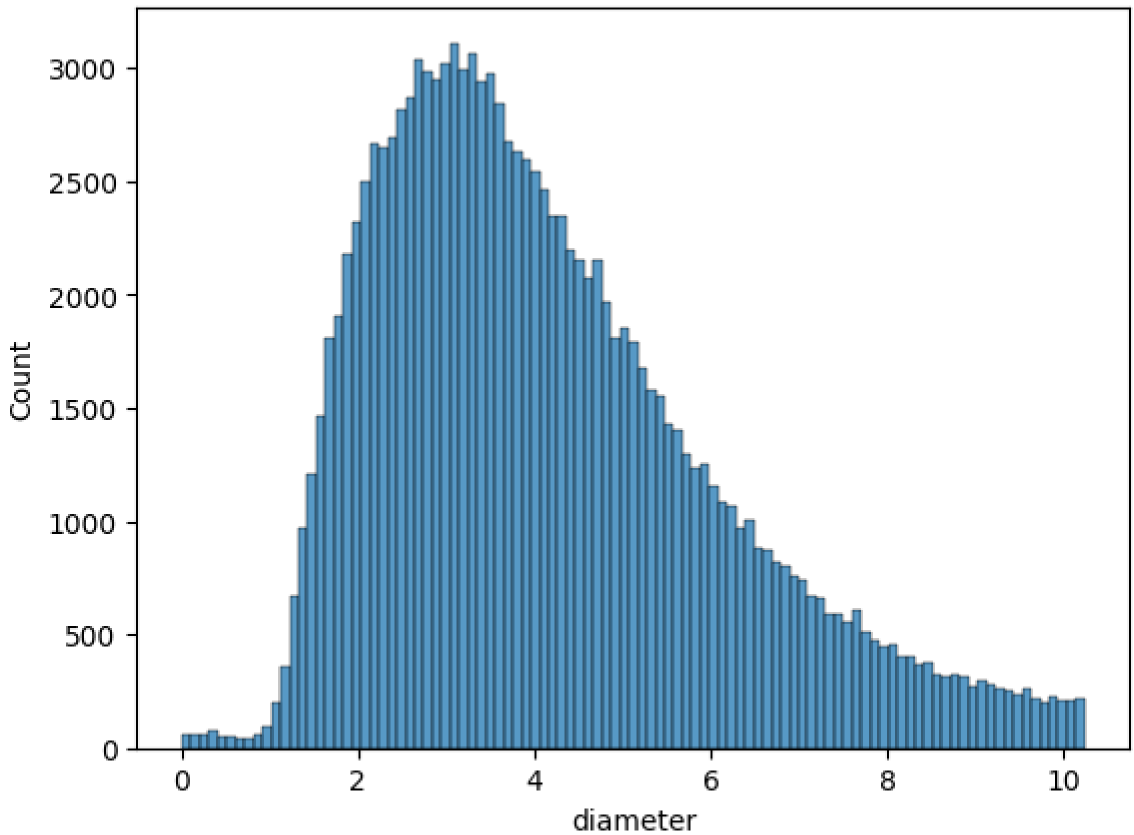
Alerts

`orbit_id` has a high cardinality: 222 distinct values	**High cardinality**
`e` is highly overall correlated with `moid`	**High correlation**
`a` is highly overall correlated with `n` and 4 other fields (n, moid, moid_jup, diameter, class)	**High correlation**

Out[ ]:

Histogram Plot of target variable

In [ ]:
```
#Look at the distribution of the target variable
plt.figure()
sns.histplot(df.diameter)
plt.savefig("Histogram.png")
```



Start of data pre-processing

In [ ]:
```
#Ensuring all the missing values have been taken care of previously
df.isna().sum()
```

Out[ ]:
```
orbit_id          0
e                 0
a                 0
i                 0
om                0
w                 0
ma                0
n                 0
tp                0
moid              0
moid_jup          0
class             0
producer          0
data_arc          0
n_obs_used        0
rms               0
diameter          0
albedo            0
diameter_sigma    0
first_year_obs    0
first_month_obs   0
last_obs_year     0
last_obs_month    0
dtype: int64
```

In [ ]:
```
# Get an idea of the different means, distributions, and values associated with the
df.describe()
```

Out[ ]:

	e	a	i	om	w	ma
count	126497.000000	126497.000000	126497.000000	126497.000000	126497.000000	126497.00000
mean	0.146644	2.756965	10.203665	169.819406	181.823887	182.532163
std	0.076841	0.453027	6.689924	102.749965	103.538522	103.41604
min	0.000488	0.626226	0.021855	0.000929	0.004466	0.000517
25%	0.091182	2.510297	5.051481	82.100534	91.822257	93.746347
50%	0.140047	2.729370	9.244113	160.539684	183.660501	185.542573
75%	0.192297	3.074005	13.538838	256.258893	271.540490	270.95750
max	0.968381	69.576833	158.535394	359.990858	359.995174	359.99922

In [ ]:

```
# look at possibly doing a time split for this data
df.first_year_obs.describe()
```

Out[ ]:

```
count    126497.000000
mean       1995.518985
std          11.947776
min        1892.000000
25%        1993.000000
50%        1998.000000
75%        2001.000000
max        2014.000000
Name: first_year_obs, dtype: float64
```

In [ ]:

```
#Start with splitting the data into a train, validation, and test case using an 80/

from sklearn.model_selection import train_test_split
# Split into Train and Test sets
train, test = train_test_split(df, train_size=.80, test_size=0.20, random_state=42

# Split train into train & val
train, val = train_test_split(train, train_size=0.80, test_size=0.20, random_state

train.shape, val.shape, test.shape
```

Out[ ]:

```
((80957, 23), (20240, 23), (25300, 23))
```

In [ ]:

```
#Get an idea of what the train dataframe now looks like (random selection of rows j
train.head()
```

Out[ ]:

	orbit_id	e	a	i	om	w	ma	n	
47132	JPL 16	0.120403	3.134257	2.631827	98.964437	147.362310	312.464742	0.177624	2.4
3303	JPL 30	0.054356	2.946768	2.633897	144.928248	146.121733	203.156078	0.194843	2.4
20238	JPL 20	0.127246	2.429705	5.758059	156.222484	40.709290	163.512166	0.260240	2.4
118643	JPL 5	0.328479	2.527473	17.669599	123.731397	241.174366	40.363736	0.245287	2.4
9231	JPL 32	0.208735	2.406466	2.319565	53.082561	225.109935	67.976584	0.264019	2.4

5 rows × 23 columns

In [ ]:

```
#columns are the features we are using
```

```
train.columns
```

Out[ ]:

```
Index(['orbit_id', 'e', 'a', 'i', 'om', 'w', 'ma', 'n', 'tp', 'moid',
       'moid_jup', 'class', 'producer', 'data_arc', 'n_obs_used', 'rms',
       'diameter', 'albedo', 'diameter_sigma', 'first_year_obs',
       'first_month_obs', 'last_obs_year', 'last_obs_month'],
      dtype='object')
```

In [ ]:

```
# Reminder to myself which columns are categorical and numeric
features = ['orbit_id', 'e', 'a', 'i', 'om', 'w', 'ma', 'n', 'tp', 'moid','moid_ju
numeric_cols = ['e', 'a', 'i', 'om', 'w', 'ma', 'n', 'tp', 'moid','moid_jup', 'dat
#categorical_cols = ['orbit_id, 'class', 'producer']
target = 'diameter'
```

In [ ]:

```
# Arrange data into X features matrix and y target vector
X_train = train[features]
y_train = train[target]
X_val = val[features]
y_val = val[target]
X_test = test[features]
```

In [ ]:

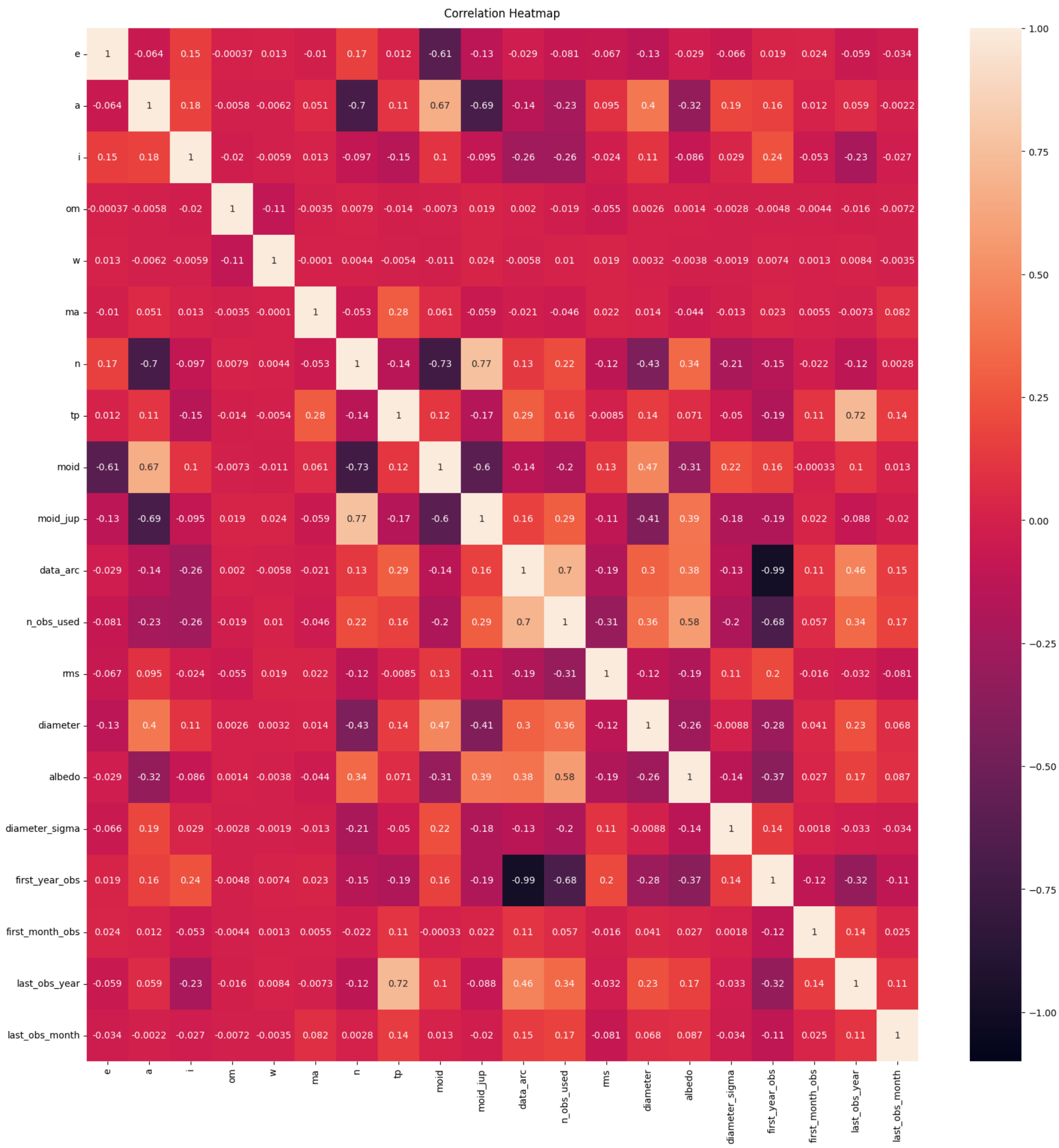
```
df.corr(numeric_only=True)
```

Out[ ]:

	e	a	i	om	w	ma	n	
e	1.000000	-0.064267	0.154808	-0.000375	0.012897	-0.010074	0.169033	0.011
a	-0.064267	1.000000	0.175739	-0.005790	-0.006180	0.050764	-0.701515	0.112
i	0.154808	0.175739	1.000000	-0.020434	-0.005927	0.012564	-0.097170	-0.149
om	-0.000375	-0.005790	-0.020434	1.000000	-0.105144	-0.003498	0.007932	-0.013
w	0.012897	-0.006180	-0.005927	-0.105144	1.000000	-0.000103	0.004396	-0.005
ma	-0.010074	0.050764	0.012564	-0.003498	-0.000103	1.000000	-0.053084	0.282
n	0.169033	-0.701515	-0.097170	0.007932	0.004396	-0.053084	1.000000	-0.140
tp	0.011687	0.112429	-0.149588	-0.013961	-0.005410	0.282615	-0.140458	1.000
moid	-0.608618	0.671739	0.100137	-0.007333	-0.011167	0.061354	-0.730948	0.117
moid_jup	-0.129100	-0.692615	-0.094843	0.018550	0.023610	-0.058957	0.767099	-0.172
data_arc	-0.028987	-0.143693	-0.258786	0.001979	-0.005783	-0.020675	0.126888	0.293
n_obs_used	-0.080970	-0.230815	-0.256723	-0.019454	0.010177	-0.046394	0.220639	0.164
rms	-0.067351	0.095368	-0.024041	-0.055052	0.019478	0.021927	-0.121095	-0.008
diameter	-0.128467	0.403511	0.105850	0.002646	0.003206	0.013657	-0.434750	0.141
albedo	-0.028831	-0.319065	-0.085915	0.001372	-0.003771	-0.044436	0.342458	0.071
diameter_sigma	-0.065661	0.190137	0.029262	-0.002800	-0.001873	-0.013239	-0.208430	-0.050
first_year_obs	0.019478	0.162742	0.238657	-0.004819	0.007429	0.022840	-0.154269	-0.190
first_month_obs	0.024310	0.012184	-0.052527	-0.004384	0.001289	0.005511	-0.021848	0.110
last_obs_year	-0.059243	0.059071	-0.225271	-0.015883	0.008446	-0.007270	-0.116838	0.721
last_obs_month	-0.033905	-0.002197	-0.026520	-0.007161	-0.003508	0.081715	0.002764	0.139

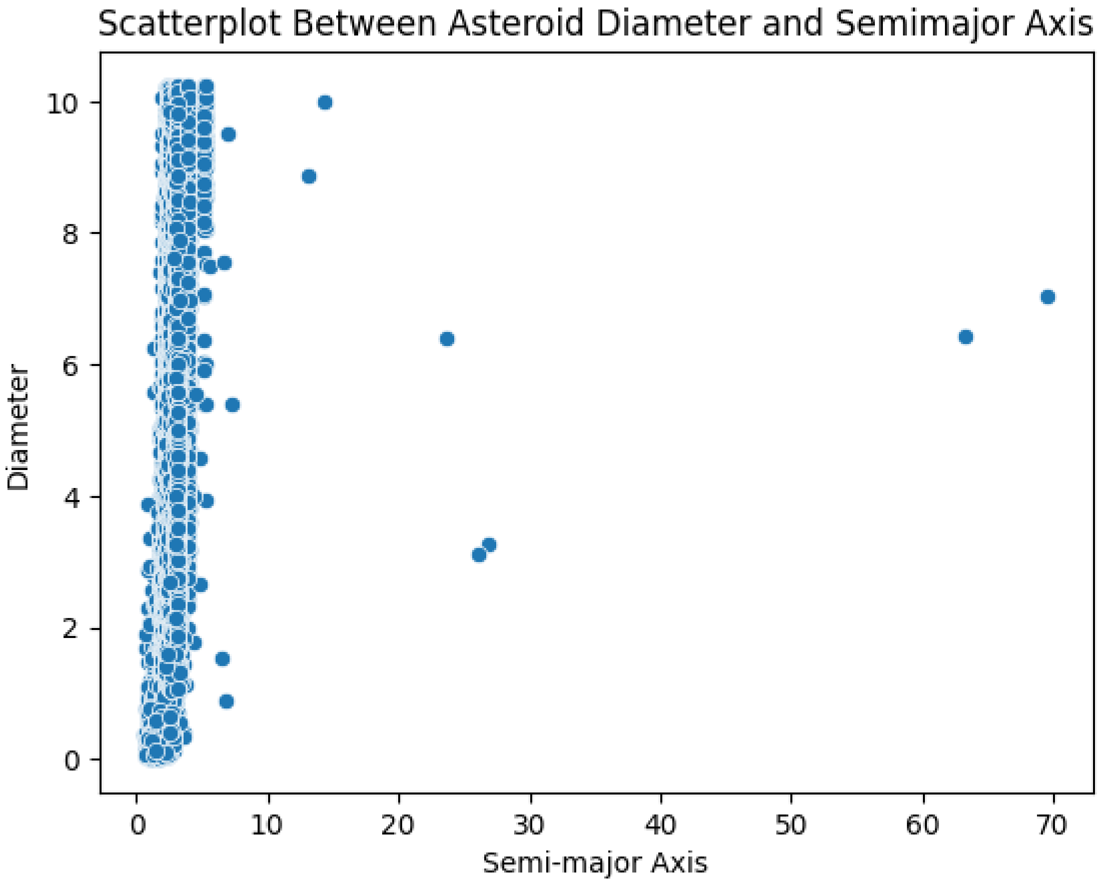
Correlation HeatMap (see attached png)

```
In [ ]: plt.figure(figsize=(20, 20))
        heatmap = sns.heatmap(df.corr(numeric_only=True), vmin=1, vmax=-1, annot=True)
        heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':12}, pad=12)
        plt.savefig('myplot.png')
```

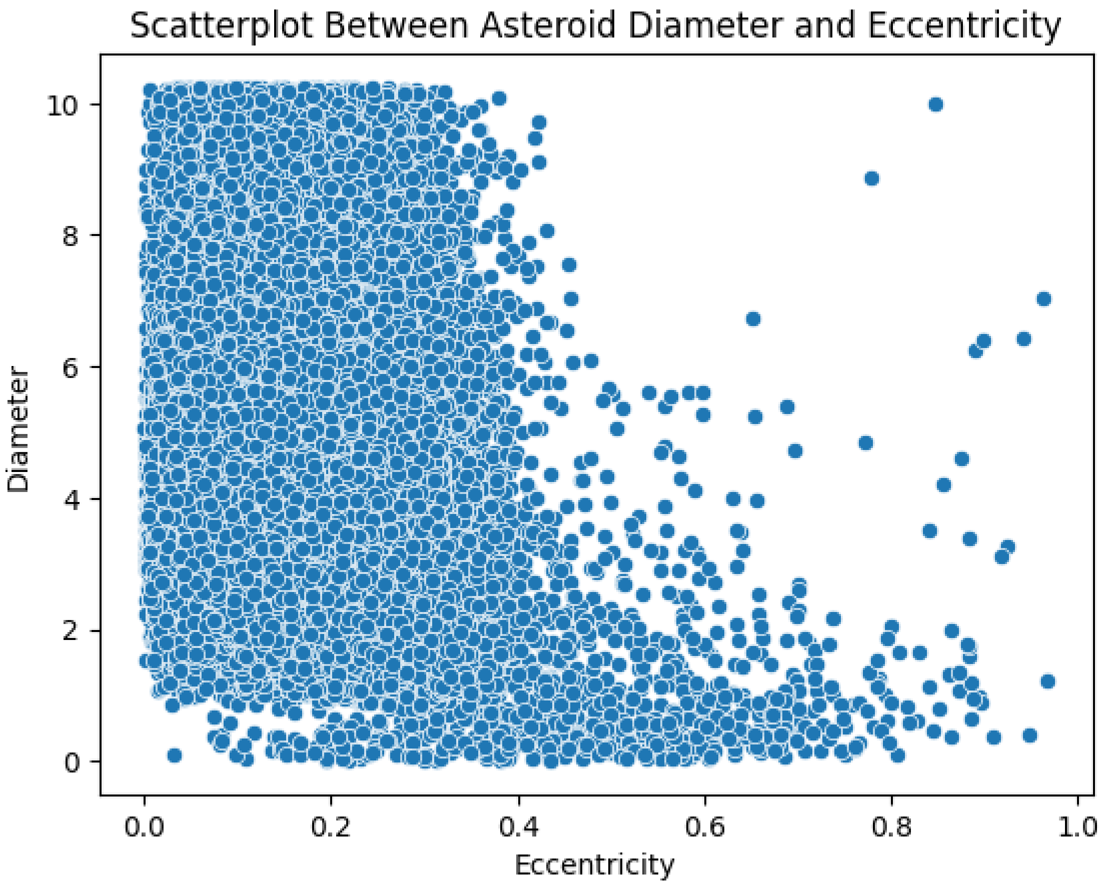


Scatterplots (see attached png)

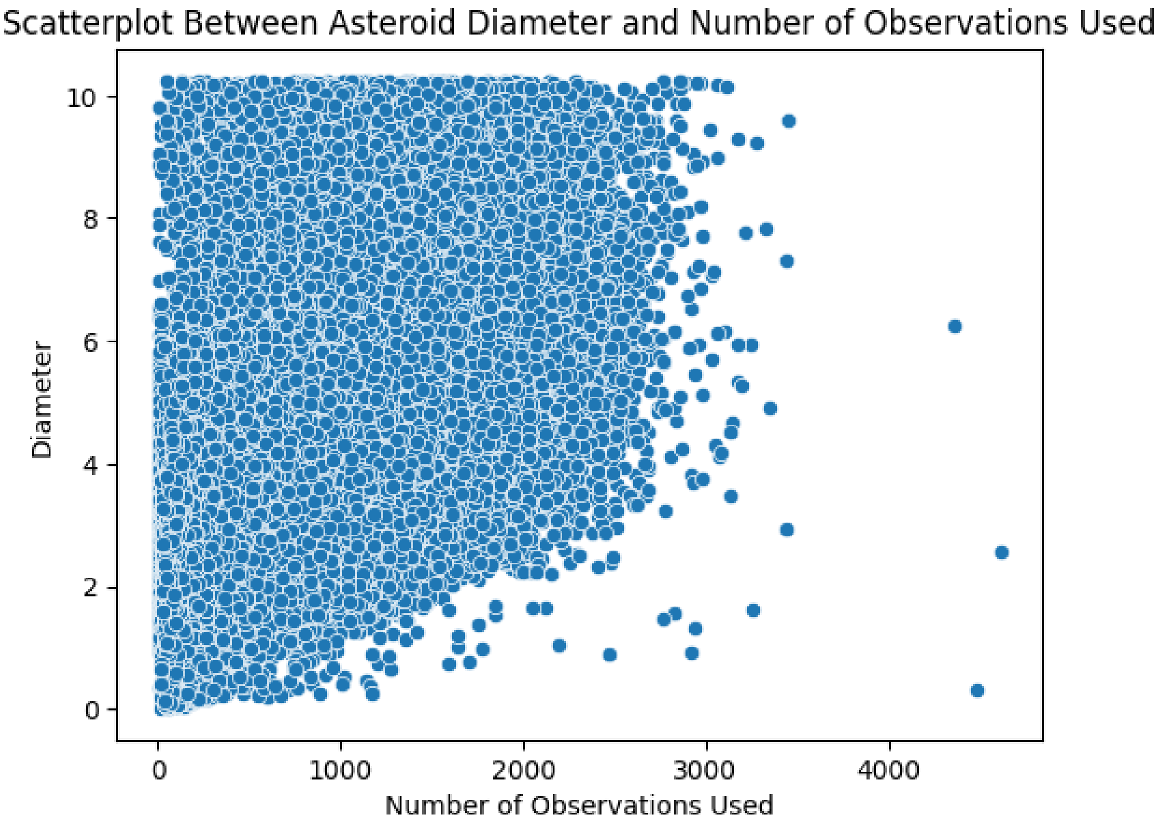
```
In [ ]: plt.figure()
        sns.scatterplot(
            data=df,
            x='a',
            y='diameter')
        plt.title('Scatterplot Between Asteroid Diameter and Semimajor Axis')
        plt.xlabel('Semi-major Axis')
        plt.ylabel('Diameter')
        plt.savefig('Scatterplot Between Asteroid Diameter and Semimajor Axis.png')
```



In [ ]:
```
plt.figure()
sns.scatterplot(
    data=df,
    x='e',
    y='diameter')
plt.title('Scatterplot Between Asteroid Diameter and Eccentricity')
plt.xlabel('Eccentricity')
plt.ylabel('Diameter')
plt.savefig('Scatterplot Between Asteroid Diameter and Eccentricity.png')
```



In [ ]:
```
plt.figure()
sns.scatterplot(
    data=df,
    x='n_obs_used',
    y='diameter')
plt.title('Scatterplot Between Asteroid Diameter and Number of Observations Used')
plt.xlabel('Number of Observations Used')
plt.ylabel('Diameter')
plt.savefig('Scatterplot Between Asteroid Diameter and Number of Observations Used
```



Baseline Model for Dataset

In [ ]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

# Arrange y target vectors
target = 'diameter'
y_train = train[target]
y_val = val[target]
y_test = test[target]

# Get mean baseline
print('Mean Baseline (using 0 features)')
guess = y_train.mean()

# Train Error
y_pred = [guess] * len(y_train)
mae_t = mean_absolute_error(y_train, y_pred)
mse_t = mean_squared_error(y_train, y_pred)
rmse_t = math.sqrt(mse_t)
r2_t = r2_score(y_train, y_pred)
print(f'Training MAE Error: {mae_t:.2f} km standarized')
print(f'Training MSE Error: {mse_t:.2f} km standarized')
print(f'Validation RMSE Error: {rmse_t:.2f} km standarized')
print(f'Training R^2 Error: {r2_t:.2f}%')

# Validation Error
y_pred = [guess] * len(y_val)
mae_v = mean_absolute_error(y_val, y_pred)
mse_v = mean_squared_error(y_val, y_pred)
rmse_v = math.sqrt(mse_v)
r2_val = r2_score(y_val, y_pred)
print(f'Validation MAE Error: {mae_v:.2f} km standarized ')
print(f'Validation MSE Error: {mse_v:.2f} km standarized')
```

```
print(f'Validation RMSE Error: {rmse_v:.2f} km standarized')
print(f'Validation R^2 Error: {r2_val:.2f}%')
```

```
Mean Baseline (using 0 features)
Training MAE Error: 1.54 km standarized
Training MSE Error: 3.74 km standarized
Validation RMSE Error: 1.93 km standarized
Training R^2 Error: 0.00%
Validation MAE Error: 1.55 km standarized
Validation MSE Error: 3.73 km standarized
Validation RMSE Error: 1.93 km standarized
Validation R^2 Error: -0.00%
```

Logistic Regression

In [ ]:

```
from sklearn.linear_model import LogisticRegressionCV
from sklearn.preprocessing import StandardScaler
import category_encoders as ce
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.pipeline import make_pipeline

#1. Import the appropriate estimator class from Scikit-Learn
from sklearn.linear_model import LinearRegression

# 2. Instantiate this class
model = LinearRegression()

# 3. Arrange X features matrices (already did y target vectors)
featurelr = ['n_obs_used']
X_train = train[featurelr]
X_val = val[featurelr]
X_test = test[featurelr]
print(f'Linear Regression, dependent on {featurelr}:')

# 4. Fit the model
model.fit(X_train, y_train)
y_pred_t = model.predict(X_train)
mae_t = mean_absolute_error(y_train, y_pred_t)
mse_t = mean_squared_error(y_train, y_pred_t)
rmse_t = math.sqrt(mse_t)
r2_t = r2_score(y_train, y_pred_t)
print(f'Training MAE Error: {mae_t:.2f} km standarized')
print(f'Training MSE Error: {mse_t:.2f} km standarized')
print(f'Training RMSE Error: {mse_t:.2f} km standarized')
print(f'Training R^2 Error: {r2_t:.4f}')

# 5. Apply the model to new data
y_pred_v = model.predict(X_val)
mae_v = mean_absolute_error(y_val, y_pred_v)
mse_v = mean_squared_error(y_val, y_pred_v)
rmse_v = math.sqrt(mse_v)
r2_val = r2_score(y_val, y_pred_v)
print(f'Validation MAE Error: {mae_v:.2f} km standarized ')
print(f'Validation MSE Error: {mse_v:.2f} km standarized')
print(f'Training RMSE Error: {mse_v:.2f} km standarized')
print(f'Validation R^2 Error: {r2_val:.4f}')
```

```
Linear Regression, dependent on ['n_obs_used']:
Training MAE Error: 1.45 km standarized
Training MSE Error: 3.26 km standarized
Training RMSE Error: 3.26 km standarized
Training R^2 Error: 0.1281
Validation MAE Error: 1.45 km standarized
Validation MSE Error: 3.22 km standarized
Training RMSE Error: 3.22 km standarized
Validation R^2 Error: 0.1363
```

Decision Tree

In [ ]:

```
import category_encoders as ce
#from sklearn.feature_selection import f_regression, SelectKBest
from sklearn.impute import SimpleImputer
#from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

features = ['orbit_id', 'e', 'a', 'i', 'om', 'w', 'ma', 'n', 'tp', 'moid','moid_ju
target = 'diameter'

# Arrange data into X features matrix and y target vector
X_train = train[features]
y_train = train[target]
X_val = val[features]
y_val = val[target]
X_test = test[features]
X_train = train.drop(columns=target)
X_val = val.drop(columns=target)

# Create a pipeline, one hot encode the low cardinality values, ordinal encode the
pipeline = make_pipeline(
    ce.OneHotEncoder(use_cat_names=True, cols=['class','producer']),
    ce.OrdinalEncoder(cols = ['orbit_id']),
    StandardScaler(),
    DecisionTreeRegressor(criterion='friedman_mse', max_depth=15, min_samples_leaf
)

# fit the pipeline on training data
pipeline.fit(X_train,y_train)


y_pred_train = pipeline.predict(X_train)
y_pred_val = pipeline.predict(X_val)


print('- Training R^2 value', pipeline.score(X_train, y_train))
print('- Validation R^2 value', pipeline.score(X_val, y_val))
print(f'- Training MAE: {mean_absolute_error(y_train,y_pred_train)} km (standarize
print(f'- Validation MAE: {mean_absolute_error(y_val,y_pred_val)} km (standarized)
print(f'- Training MSE: {mean_squared_error(y_train,y_pred_train)} km (standarized
print(f'- Validation MSE: {mean_squared_error(y_val,y_pred_val)} km (standarized)'
print(f'- Training RMSE: {math.sqrt(mean_squared_error(y_train,y_pred_train))} km
print(f'- Validation RMSE: {math.sqrt(mean_squared_error(y_val,y_pred_val))} km (s
```

```
- Training R^2 value 0.917263649870226
- Validation R^2 value 0.8550048659257631
- Training MAE: 0.3985889005994833 km (standarized)
- Validation MAE: 0.5261369221915055 km (standarized)
- Training MSE: 0.3095382154788729 km (standarized)
- Validation MSE: 0.5411865502211649 km (standarized)
- Training RMSE: 0.5563615869907563 km (standarized)
- Validation RMSE: 0.7356538249891487 km (standarized)
```

Hyperoptimisation for Decision Tree

In [ ]:
```python
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

features = ['orbit_id', 'e', 'a', 'i', 'om', 'w', 'ma', 'n', 'tp', 'moid','moid_ju
target = 'diameter'
X_train = train[features]
X_train = train.drop(columns=target)
y_train = train[target]

pipeline = make_pipeline(
    ce.OneHotEncoder(use_cat_names=True, cols=['class','producer']),
    ce.OrdinalEncoder(cols = ['orbit_id']),
    StandardScaler(),
    DecisionTreeRegressor(random_state=42)
)

#different parameter distributions to test for the best possible combination
param_distributions = {
   'decisiontreeregressor__min_samples_leaf': [1, 3, 5, 7, 9, 10, 15],
    'decisiontreeregressor__max_depth': [5, 7, 9, 10, 13, 15, 17, 20, 21,  25, 30]
    'decisiontreeregressor__min_samples_split': [2, 3, 4, 5, 7],
}




# If you're on Colab, decrease n_iter & cv parameters
search = RandomizedSearchCV(
    pipeline,
    param_distributions=param_distributions,
    n_iter=100,
    cv=5,
    scoring='neg_mean_absolute_error',
    verbose=10,
    return_train_score=True,
    n_jobs=-1
)

search.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

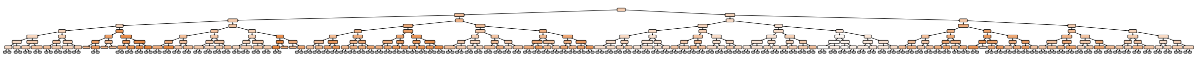
In [ ]:
```python
print('Best hyperparameters', search.best_params_)
print('Cross-validation MAE', -search.best_score_)
```

```
Best hyperparameters {'decisiontreeregressor__min_samples_split': 3, 'decisiontree
regressor__min_samples_leaf': 15, 'decisiontreeregressor__max_depth': 15}
Cross-validation MAE 0.5310870583086229
```

Graphing Decision Tree (hyperoptimisation was trialed and error based on hyperoptimisation)

In [ ]:
```python
import graphviz
```

```
import os
from sklearn.tree import export_graphviz

os.environ["PATH"] += os.pathsep + r"D:\Graphviz\bin"
ord_encoder = ce.OrdinalEncoder(cols = ['orbit_id'])
X_train_ordencoded = ord_encoder.fit_transform(X_train)
X_val_ordencoded = ord_encoder.transform(X_val)

oh_encoder = ce.OneHotEncoder(use_cat_names=True, cols=['class','producer'])
X_train_encoded = oh_encoder.fit_transform(X_train_ordencoded)
X_val_encoded = oh_encoder.transform(X_val_ordencoded)

dt = pipeline.named_steps['decisiontreeregressor']

dt.fit(X_train_encoded,y_train)

encoded_columns = X_train_encoded.columns

dot_data = export_graphviz(dt,
                           out_file=None,
                           max_depth=7,
                           feature_names=encoded_columns,
                           impurity=False,
                           filled=True,
                           proportion=True,
                           rounded=True)
display(graphviz.Source(dot_data))
```



Random Forest Regressor

In [ ]:

```
import category_encoders as ce
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

features = ['orbit_id', 'e', 'a', 'i', 'om', 'w', 'ma', 'n', 'tp', 'moid','moid_ju
target = 'diameter'

# Arrange data into X features matrix and y target vector
X_train = train[features]
y_train = train[target]
X_val = val[features]
y_val = val[target]
X_test = test[features]
X_train = train.drop(columns=target)
X_val = val.drop(columns=target)

pipeline = make_pipeline(
    ce.OneHotEncoder(use_cat_names=True, cols=['class','producer']),
    ce.OrdinalEncoder(cols = ['orbit_id']),
    StandardScaler(),
    RandomForestRegressor(n_estimators=450, max_depth=None, max_features=.77, min_
)

pipeline.fit(X_train,y_train)

y_pred_train = pipeline.predict(X_train)
```

```
y_pred_val = pipeline.predict(X_val)



print('Training R^2 value', pipeline.score(X_train, y_train))
print('Validation R^2 value', pipeline.score(X_val, y_val))
print(f'Training MAE: {mean_absolute_error(y_train,y_pred_train)} km (standarized)
print(f'Validation MAE: {mean_absolute_error(y_val,y_pred_val)} km (standarized)')
print(f'Training MSE: {mean_squared_error(y_train,y_pred_train)} km (standarized)'
print(f'Validation MSE: {mean_squared_error(y_val,y_pred_val)} km (standarized)')
print(f'Training RMSE: {math.sqrt(mean_squared_error(y_train,y_pred_train))} km (st
print(f'Validation RMSE: {math.sqrt(mean_squared_error(y_val,y_pred_val))} km (stan
```

```
Training R^2 value 0.976558034056286
Validation R^2 value 0.9092269240568146
Training MAE: 0.2006249038237154 km (standarized)
Validation MAE: 0.4129713610682327 km (standarized)
Training MSE: 0.08770249466107992 km (standarized)
Validation MSE: 0.33880563052208673 km (standarized)
Training RMSE: 0.29614606980522284 km (standarized)
Validation RMSE: 0.5820701250898269 km (standarized)
```

In [ ]:

```
target = 'diameter'
features = train.columns.drop('diameter')

X_train = train[features]
y_train = train[target]

X_val = val[features]
y_val = val[target]

X_test = test[features]
y_test = test[target]
```

Cross-validation for Random Forest

In [ ]:

```
import category_encoders as ce
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV

ord_encoder = ce.OrdinalEncoder(cols = ['orbit_id'])
X_train_ordencoded = ord_encoder.fit_transform(X_train)
X_val_ordencoded = ord_encoder.transform(X_val)
X_test_ordencoded = ord_encoder.transform(X_test)

oh_encoder = ce.OneHotEncoder(use_cat_names=True, cols=['class','producer'])
X_train_encoded = oh_encoder.fit_transform(X_train_ordencoded)
X_val_encoded = oh_encoder.transform(X_val_ordencoded)
X_test_encoded = oh_encoder.transform(X_test_ordencoded)

scaler = StandardScaler()
scaler.fit(X_train_encoded)
scaler.fit(X_val_encoded)
scaler.fit(X_test_encoded)

model_dt_shap = DecisionTreeRegressor(criterion='friedman_mse', max_depth=15, min_
```

```
model_dt_shap.fit(X_train_encoded,y_train)
```

Out[ ]:

DecisionTreeRegressor

```
DecisionTreeRegressor(criterion='friedman_mse', max_depth=15,
                      min_samples_leaf=15, min_samples_split=4,
                      random_state=42)
```

In [ ]:

```
import shap
explainer = shap.TreeExplainer(model_dt_shap)
shap_values = explainer.shap_values(X_test_encoded)
```

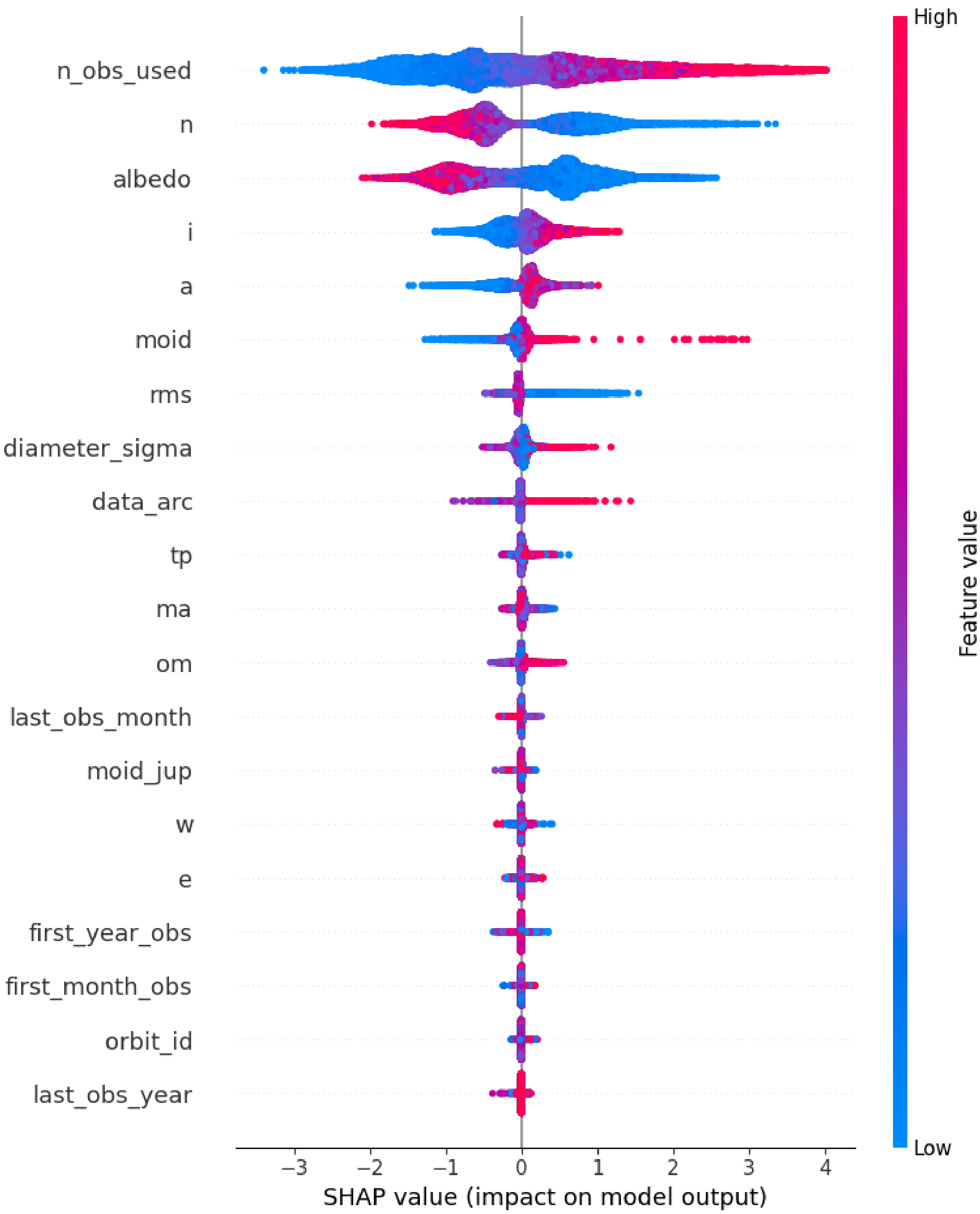
```
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\shap\utils\_clustering.py:35: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _pt_shuffle_rec(i, indexes, index_mask, partition_tree, M, pos):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\shap\utils\_clustering.py:54: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def delta_minimization_order(all_masks, max_swap_size=100, num_passes=2):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\shap\utils\_clustering.py:63: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _reverse_window(order, start, length):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\shap\utils\_clustering.py:69: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _reverse_window_score_gain(masks, order, start, length):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\shap\utils\_clustering.py:77: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _mask_delta_score(m1, m2):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\shap\links.py:5: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def identity(x):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\shap\links.py:10: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _identity_inverse(x):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\shap\links.py:15: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def logit(x):
```

```
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfr
a8p0\LocalCache\local-packages\Python39\site-packages\shap\links.py:20: NumbaDepre
cationWarning: The 'nopython' keyword argument was not supplied to the 'numba.jit'
decorator. The implicit default value for this argument is currently False, but it
will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stabl
e/reference/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-u
sing-jit for details.
  def _logit_inverse(x):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfr
a8p0\LocalCache\local-packages\Python39\site-packages\shap\utils\_masked_model.py:
363: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to
 the 'numba.jit' decorator. The implicit default value for this argument is curren
tly False, but it will be changed to True in Numba 0.59.0. See https://numba.readt
hedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-bac
k-behaviour-when-using-jit for details.
  def _build_fixed_single_output(averaged_outs, last_outs, outputs, batch_position
s, varying_rows, num_varying_rows, link, linearizing_weights):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfr
a8p0\LocalCache\local-packages\Python39\site-packages\shap\utils\_masked_model.py:
385: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to
 the 'numba.jit' decorator. The implicit default value for this argument is curren
tly False, but it will be changed to True in Numba 0.59.0. See https://numba.readt
hedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-bac
k-behaviour-when-using-jit for details.
  def _build_fixed_multi_output(averaged_outs, last_outs, outputs, batch_position
s, varying_rows, num_varying_rows, link, linearizing_weights):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfr
a8p0\LocalCache\local-packages\Python39\site-packages\shap\utils\_masked_model.py:
428: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to
 the 'numba.jit' decorator. The implicit default value for this argument is curren
tly False, but it will be changed to True in Numba 0.59.0. See https://numba.readt
hedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-bac
k-behaviour-when-using-jit for details.
  def _init_masks(cluster_matrix, M, indices_row_pos, indptr):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfr
a8p0\LocalCache\local-packages\Python39\site-packages\shap\utils\_masked_model.py:
439: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to
 the 'numba.jit' decorator. The implicit default value for this argument is curren
tly False, but it will be changed to True in Numba 0.59.0. See https://numba.readt
hedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-bac
k-behaviour-when-using-jit for details.
  def _rec_fill_masks(cluster_matrix, indices_row_pos, indptr, indices, M, ind):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfr
a8p0\LocalCache\local-packages\Python39\site-packages\shap\maskers\_tabular.py:18
6: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to th
e 'numba.jit' decorator. The implicit default value for this argument is currently
False, but it will be changed to True in Numba 0.59.0. See https://numba.readthedo
cs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-be
haviour-when-using-jit for details.
  def _single_delta_mask(dind, masked_inputs, last_mask, data, x, noop_code):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfr
a8p0\LocalCache\local-packages\Python39\site-packages\shap\maskers\_tabular.py:19
7: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to th
e 'numba.jit' decorator. The implicit default value for this argument is currently
False, but it will be changed to True in Numba 0.59.0. See https://numba.readthedo
cs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-be
haviour-when-using-jit for details.
  def _delta_masking(masks, x, curr_delta_inds, varying_rows_out,
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfr
a8p0\LocalCache\local-packages\Python39\site-packages\shap\maskers\_image.py:175:
 NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the
 'numba.jit' decorator. The implicit default value for this argument is currently
 False, but it will be changed to True in Numba 0.59.0. See https://numba.readthed
ocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-back-b
```

```
ehaviour-when-using-jit for details.
  def _jit_build_partition_tree(xmin, xmax, ymin, ymax, zmin, zmax, total_ywidth,
 total_zwidth, M, clustering, q):
C:\Users\Anne\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfr
a8p0\LocalCache\local-packages\Python39\site-packages\shap\explainers\_partition.p
y:676: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied t
o the 'numba.jit' decorator. The implicit default value for this argument is curre
ntly False, but it will be changed to True in Numba 0.59.0. See https://numba.read
thedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-ba
ck-behaviour-when-using-jit for details.
  def lower_credit(i, value, M, values, clustering):
The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be change
d to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/de
precation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for d
etails.
The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be change
d to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/de
precation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for d
etails.
```

Shapley Plot for Astreroid Prediction (see attached png)

In [ ]:
```
# summarize the effects of all the features
plt.figure()
shap.summary_plot(shap_values, X_test_encoded)
plt.savefig("Shapley.png")
```



`<Figure size 640x480 with 0 Axes>`

Originally matplotlib was not supporting the graphs so I could not display them. I decided to save it as a png that is attached to my assessment instead. However it seems to be working now but I will leave the plots as they are, saved to the document just incase there is anything wrong with the code.