Predicting Asteroid Diameter Using Statistical Machine Learning

SEPTEMBER 1

HEER BIPIN PATEL

Introduction

Statistical Machine Learning is helping solve some of the biggest challenges faced by humans. In the following report, we will discuss and analyze the capabilities of statistical machine learning models in saving Earth from asteroid collision (Blakelobato, 2020a). More specifically, we will look at the possibilities of machine learning being used to accurately predict asteroid diameter using the exhaustive data provided by the NASA Jet Propulsion Laboratory (JPL) Small-Body Database Search Engine (NASA, 2020).

The Dataset

The dataset was originally obtained from the NASA Jet Propulsion Laboratory (JPL) Small-Body Database Search Engine. It was then cleaned and preprocessed to remove redundant and missing values (Blakelobato, 2020b). The modified dataset was used as the entry point for our project.

A comprehensive description of the data set can be found below.

Name	Description	Type
orbit_id	Orbital Solution ID	String
e	Eccentricity	Number
a	Semi-major axis	au
i	Inclination	deg
om	Longitude of ascending node	deg
w	Argument of perihelion	deg
ma	Mean anomaly	deg
tp	Time of perihelion passage	Julian day form
n	Mean motion	deg/day
moid	Minimum distance between earth and small body	au
moid_jup	Minimum distance between orbits of Jupiter and small body	au

data_arc	Number of days spanned by observations used in orbit determination	days
n_obs_used	Total number of observations used	Number
albedo	Geometric albedo	Number
diameter	Effective body diameter	km
class	Orbit classification	String
producer	Name of the person who computed the orbit	String
rms	Root Mean Squared of the orbit fit	arcsec
first_year_obs	The year of first observation	Number
first_month_obs	The month of first observation	Number
last_year_obs	The year of last observation	Number
last_month_obs	The month of last observation	Number

Table 1 – *Dataset description* (NASA, 2020)

We began by looking at the distributions of different numerical values and noticed that they were all on different scales of magnitude. Moreover, many of the numerical variables were extremely skewed. To fix that, we used Inter-Quartile Ranges (IQR) that form an optimal range around the data, hence getting rid of outliers with it.

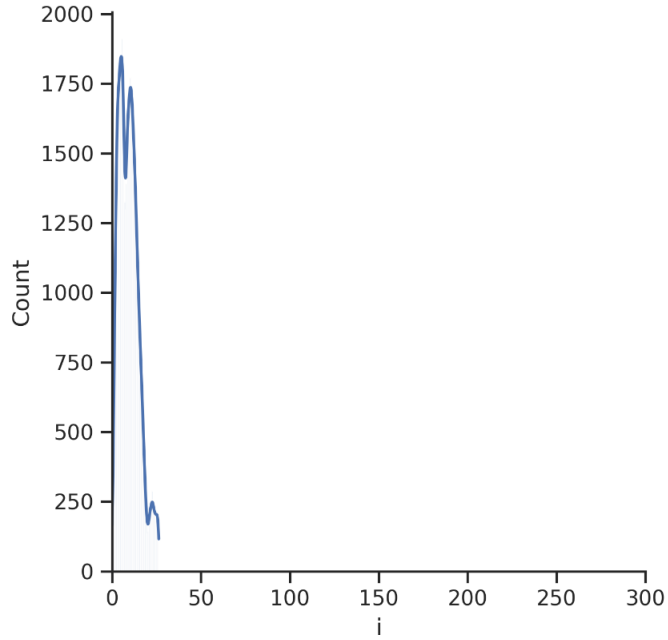


Figure 2 – *Distribution of feature "i" including outliers* (Made in Python)

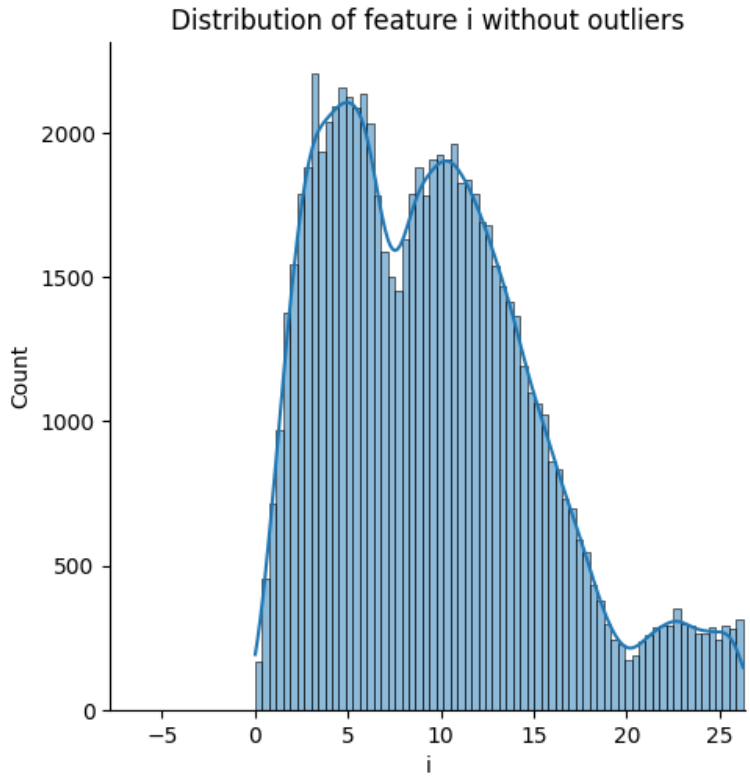


Figure 3 – *Distribution of feature "i" without outliers* (Made in Python)

We then computed the correlation matrix to find out the correlation among our variables. The plots are shown below:

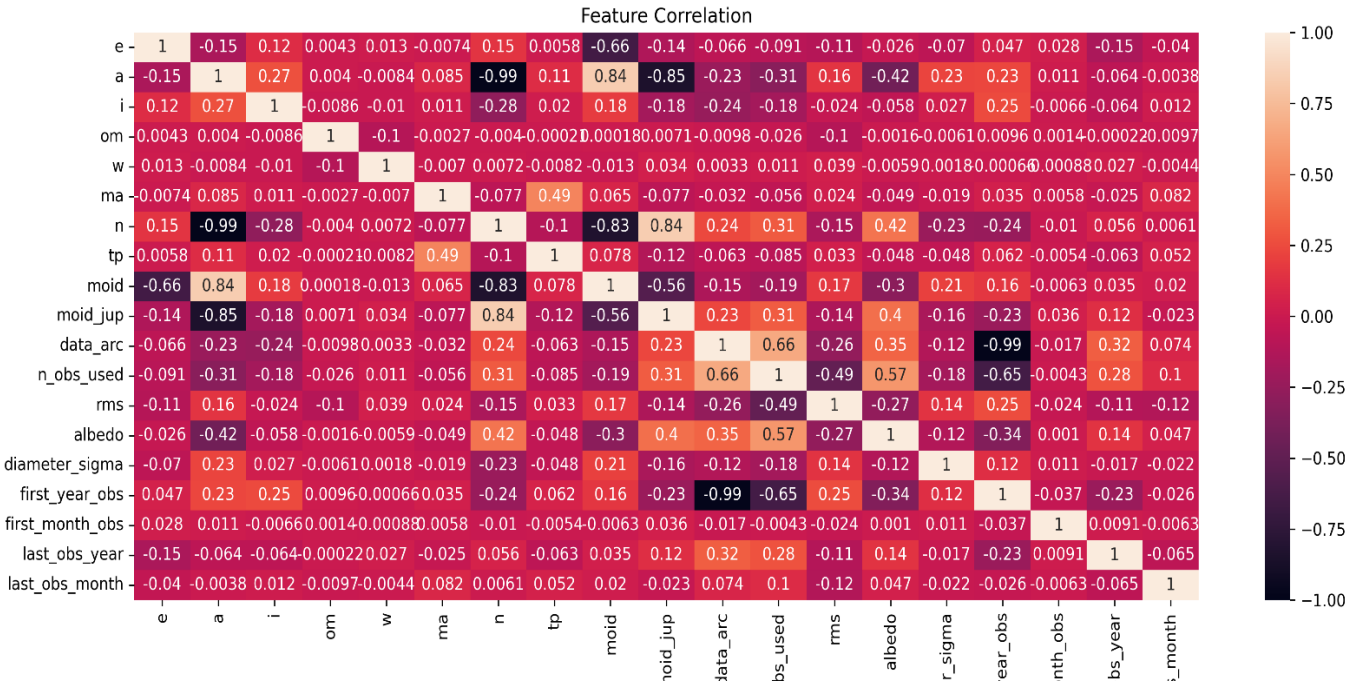


Figure 4 – *Correlation Matrix* (Made in Python)

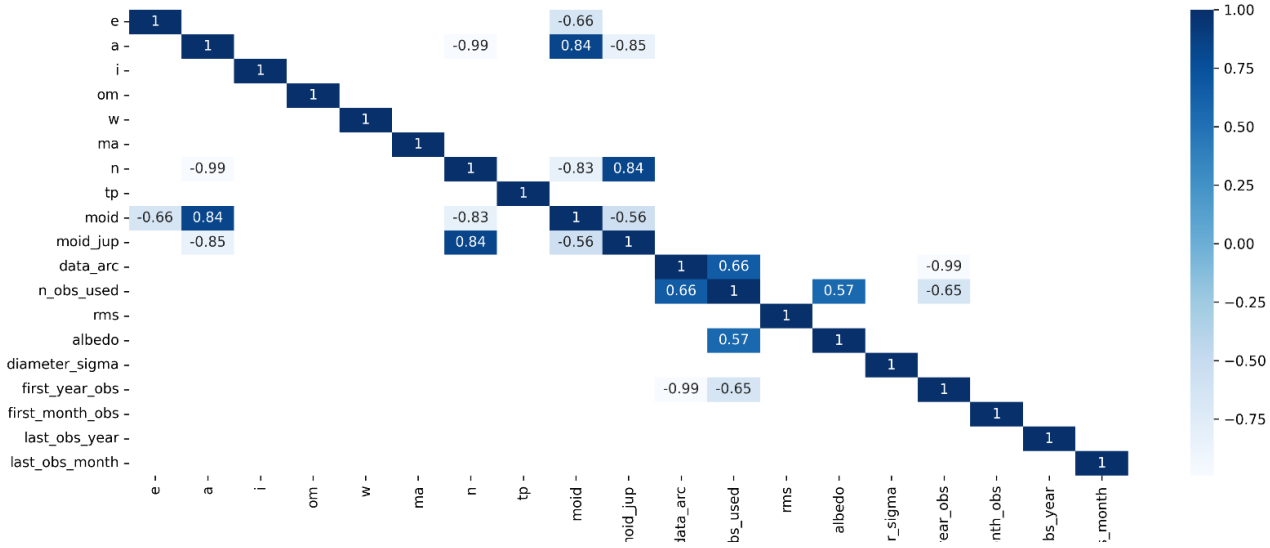


Figure 5 – *Correlation Matrix highlighting the highest correlations* (Made in Python)

Many input variables were highly correlated to other input variables, such as the 99% negative correlation between “n” and “a”, as seen in the figure below.

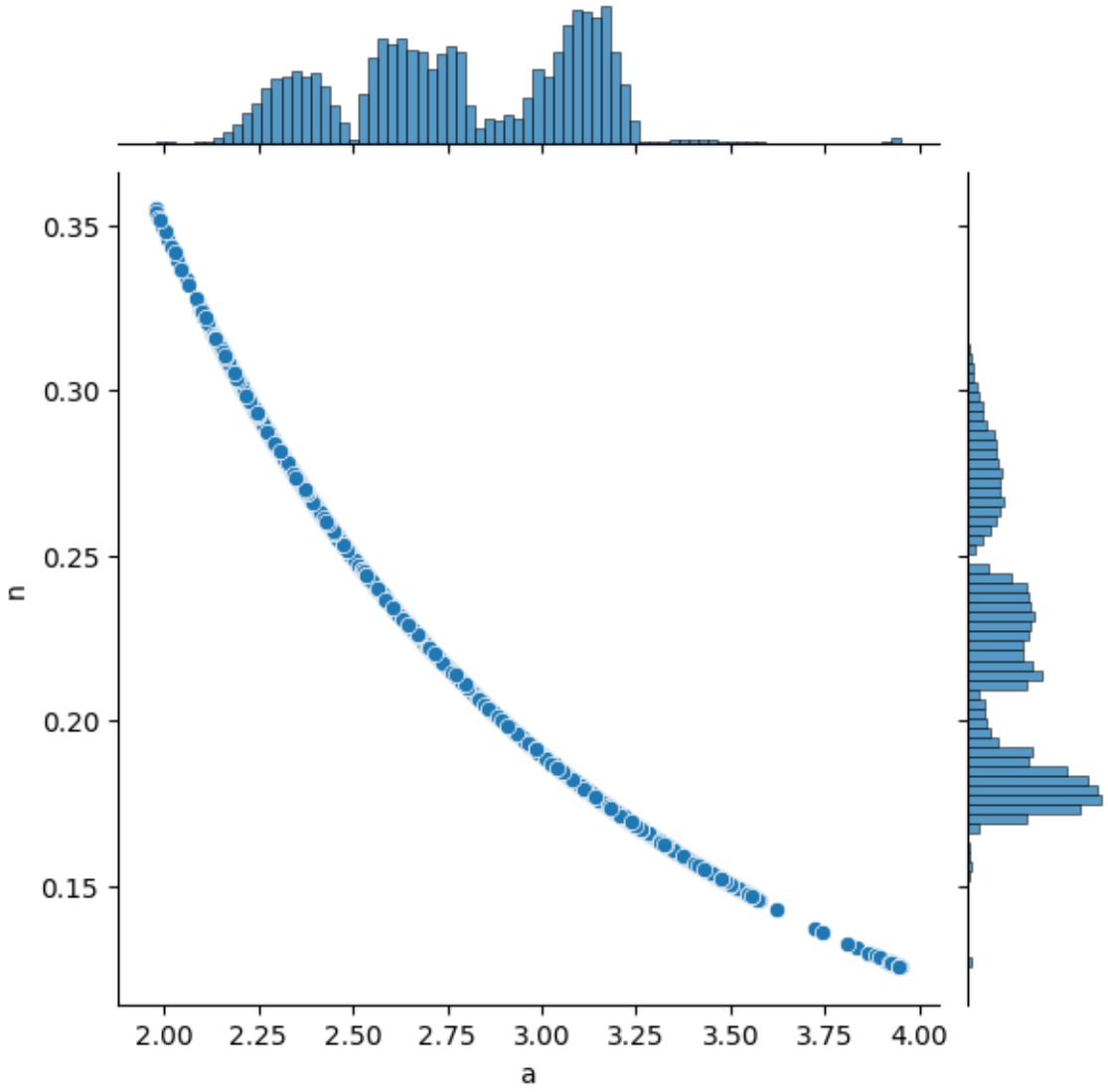
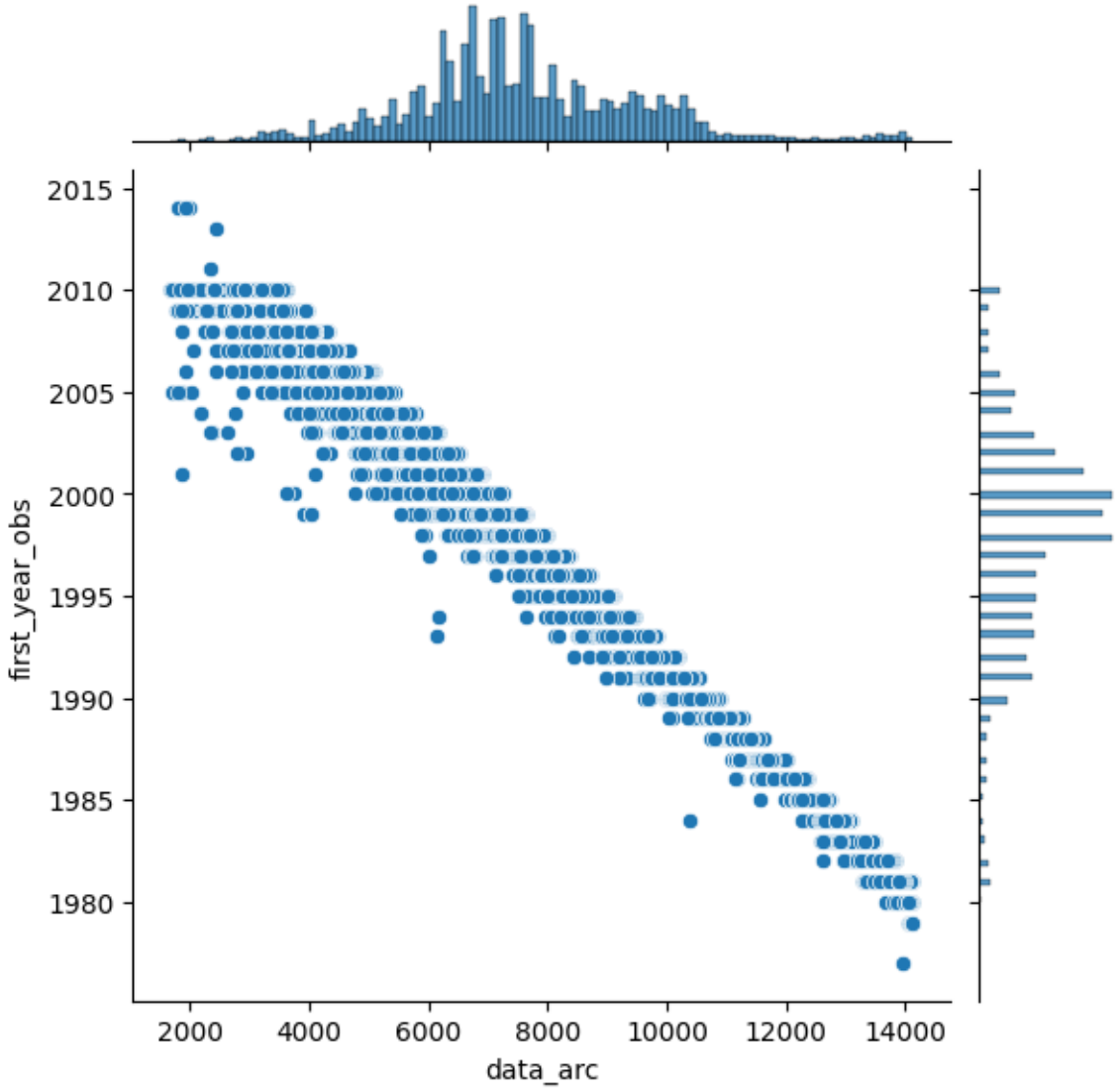


Figure 6 – *Feature a vs. n plot* (Made in Python)

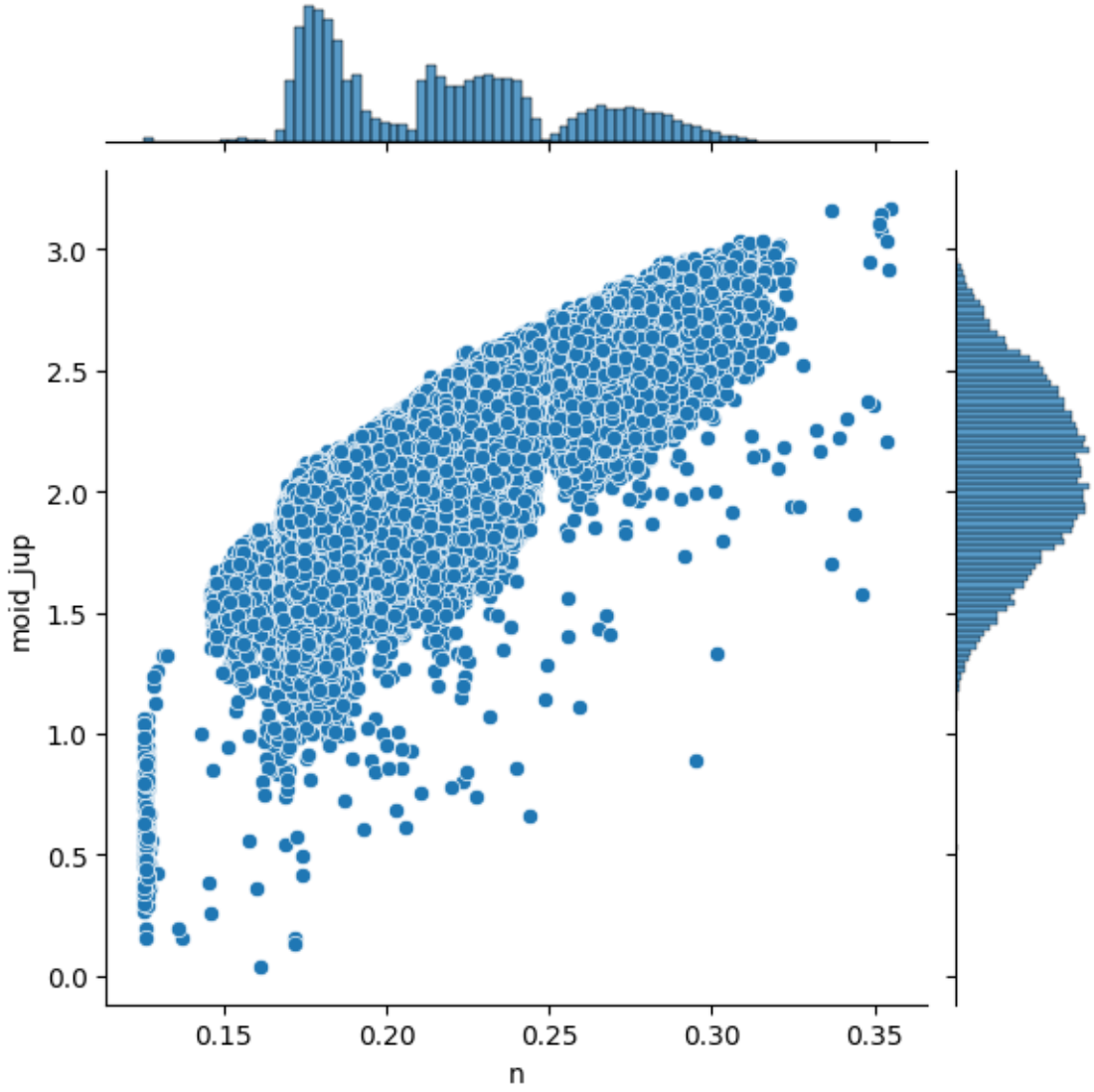
Moreover, data_arc was also found to be highly negatively correlated with the first_obs_year.



Feature data_arc vs first_year_obs plot : Correlation -0.9917498282446946

Figure 7 – *Feature data_arc vs. first_year_obs plot* (Made in Python)

The features n and moid_jup were found to be positively correlated, as seen below:



Feature n vs moid_jup plot : Correlation 0.8402471965876056

Figure 8 – *Feature n vs. moid_jup plot* (Made in Python)

For the year columns, we tried a binning approach by taking three bins each, but the result was still skewed due to the inadequacy of samples. The bin plots can be seen below.

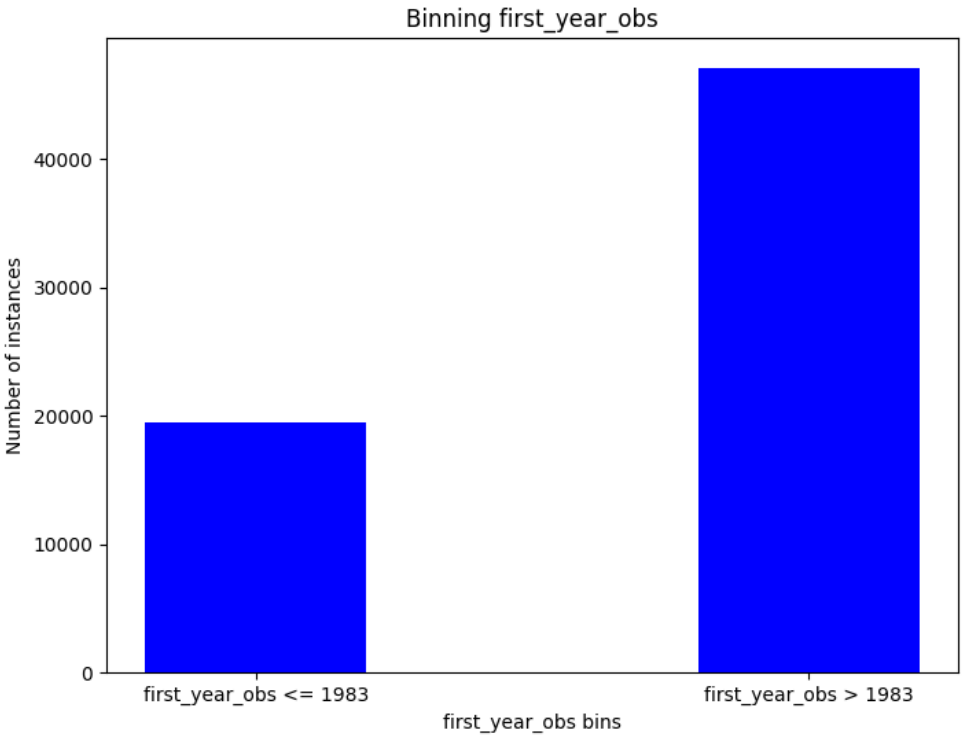


Figure 9 – *first_year_obs bins* (Made in Python)

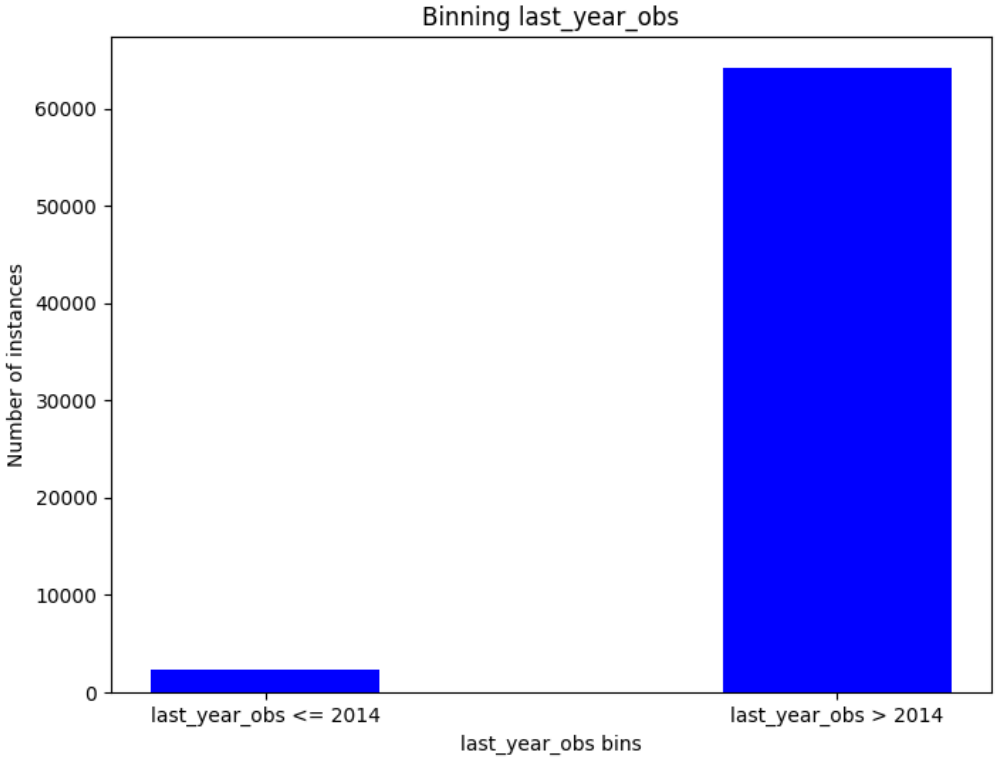


Figure 10 – *last_year_obs bins* (Made in Python)

To finally preprocess the data, we created three pipelines that follow different approaches as it isn't obvious which would perform better against the other. The first pipeline scales all the numerical features and fits an Ordinal encoder on the rest of them.

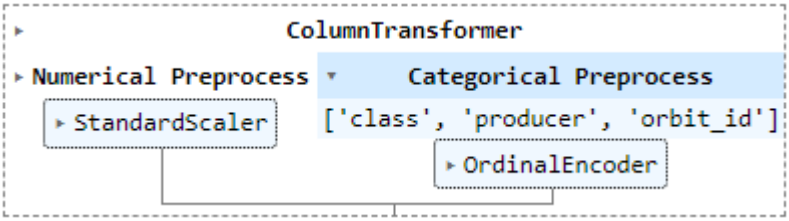


Figure 11 – *First pipeline* (Made in Python)

The second pipeline scales all the numerical features, ordinally encodes the "orbit_id" column, and one-hot encodes the rest of them.

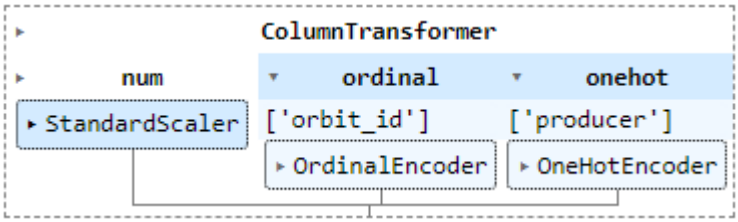


Figure 12 – *Second pipeline* (Made in Python)

The third pipeline scales all the numerical features, and one-hot encodes the rest of them.

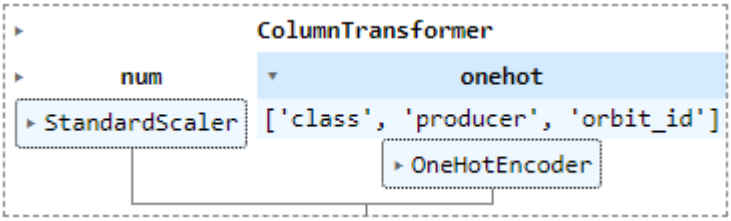


Figure 13 – *Third pipeline* (Made in Python)

Model Implementation

We chose Linear Regression, Elastic Net, Decision Tree, Random Forest, and XGBoost to train using our data and then test on the test dataset. The metrics of choice were

Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 Score. The R^2 Score tells us how much variance in the target variable can be explained by the input features.

We will only consider the scores obtained for the preprocessing pipeline 1 as it gave the highest score on our final model. The calculated metrics are tabulated below.

Model	MSE	RMSE	R^2	Comments
Linear Regression	3.41	1.85	8.55%	Worst performance
Elastic Net	1.81	1.34	51.58%	Average performance considering the model simplicity. The training time was minimal too.
Decision Tree	0.62	0.79	83.26%	Good performance
Random Forest	0.43	0.65	88.53%	Great performance. It takes much more time to train than decision trees.
XGBoost	0.37	0.61	90.06%	Best performance. The parameters were found using Randomized Search.

Table 2 – *Model performance table*

Based on the metrics, the XGBoost model (XGBRegressor) gave the best performance out of all other models, although the training time was higher as well.

Model Insights

As we used a tree-based model as our final model, we can visualize the feature importances internally calculated by the model.

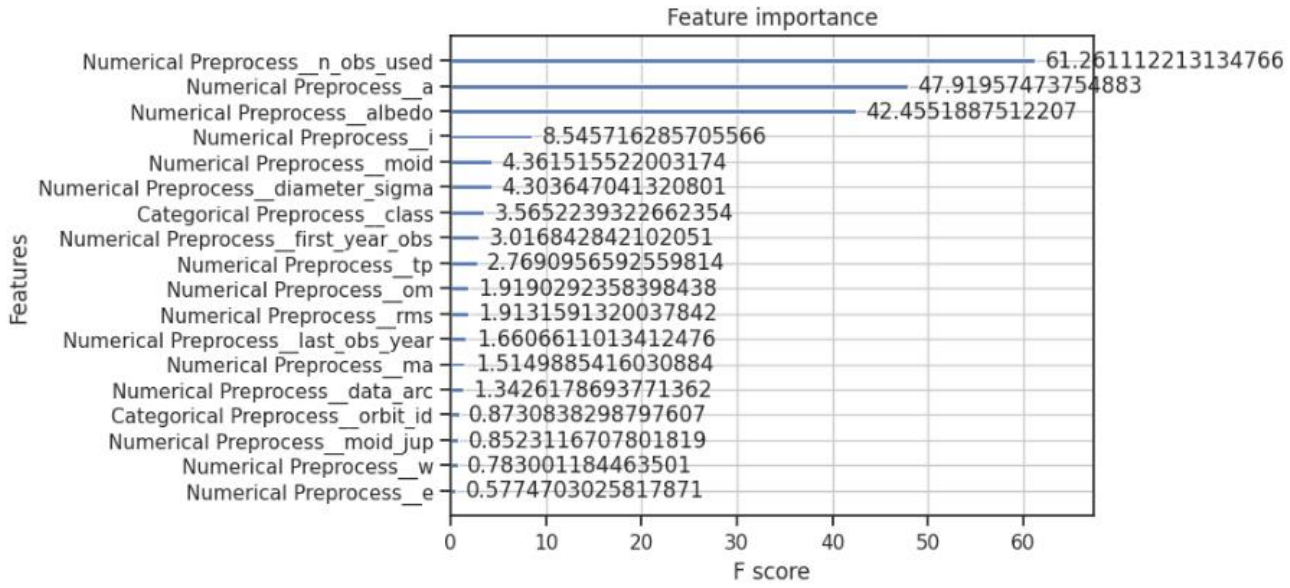


Figure 14 – *Feature importances calculated by XGBoost* (Made in Python)

From the importances, we can see that the most important features are “n_obs_used”, “a”, and “albedo” among all other features.

To obtain model explanations, we use Shapley values which indicate the contribution of a particular feature for a given data point in predicting the final output. The model used is XGBoost (XGBRegressor).

The Shapley waterfall plot for the 5th input data point is shown below.

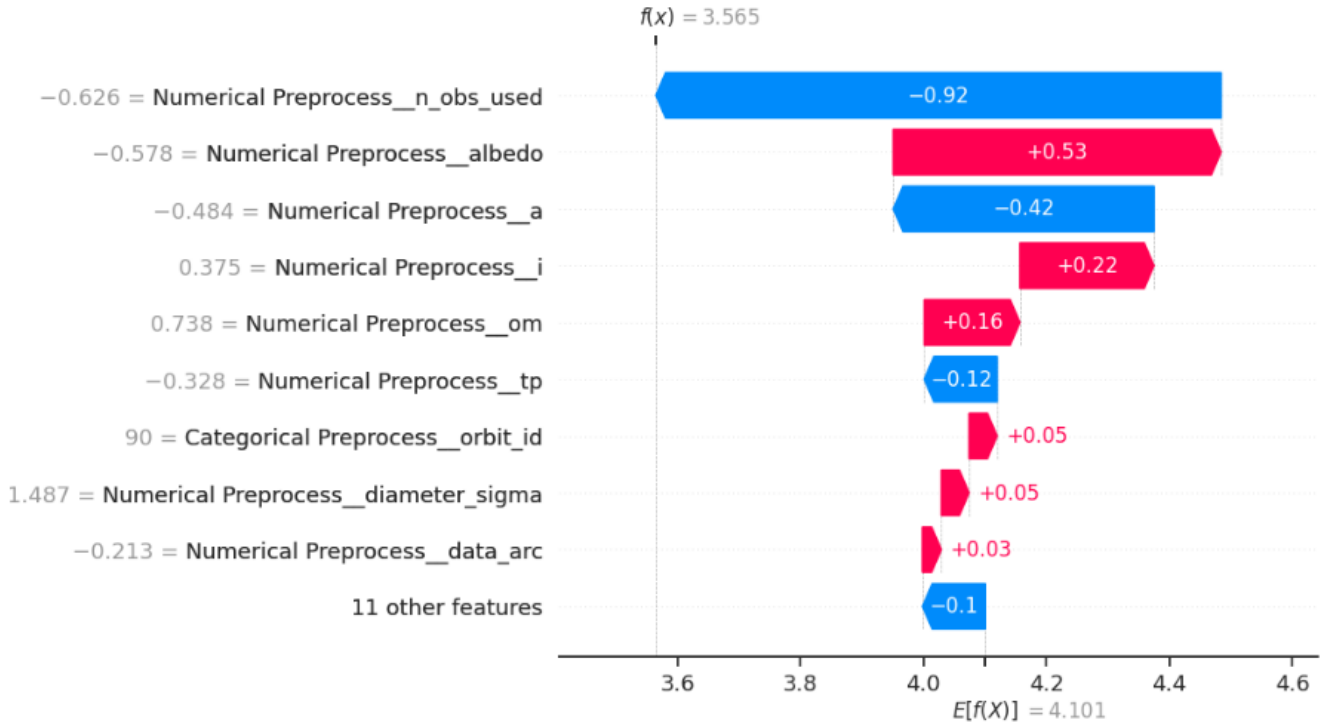


Figure 15 – *Shapley explanation for input point 5* (Made in Python)

From the plot, we can see that the expected (mean) value of the diameter is 4.101. This value is shifted towards the left by -0.1 using the lower-performing 11 features. The highest contributing feature was “n_obs_used” which contributed -0.92km to the final prediction, followed by “albedo” contributing 0.53km, and “a” contributing -0.42km.

These values indicate the deviation caused by the corresponding features to the mean of the distribution. Features that contributed the most are shown below:

```
[!] Top three features that contributed positively:
om contributed 0.1557458757923564 km to the output.
i contributed 0.21790064854819322 km to the output.
albedo contributed 0.5330864443833707 km to the output.

[!] Top three features that contributed negatively:
n_obs_used contributed -0.919058038938947 km to the output.
a contributed -0.42393750193036794 km to the output.
tp contributed -0.1186790649544173 km to the output.
```

Figure 17 – *Highest negative and positive contributing features* (Made in Python)

To obtain a summary of the effects of all the features, we plot a summary beeswarm plot as shown below.

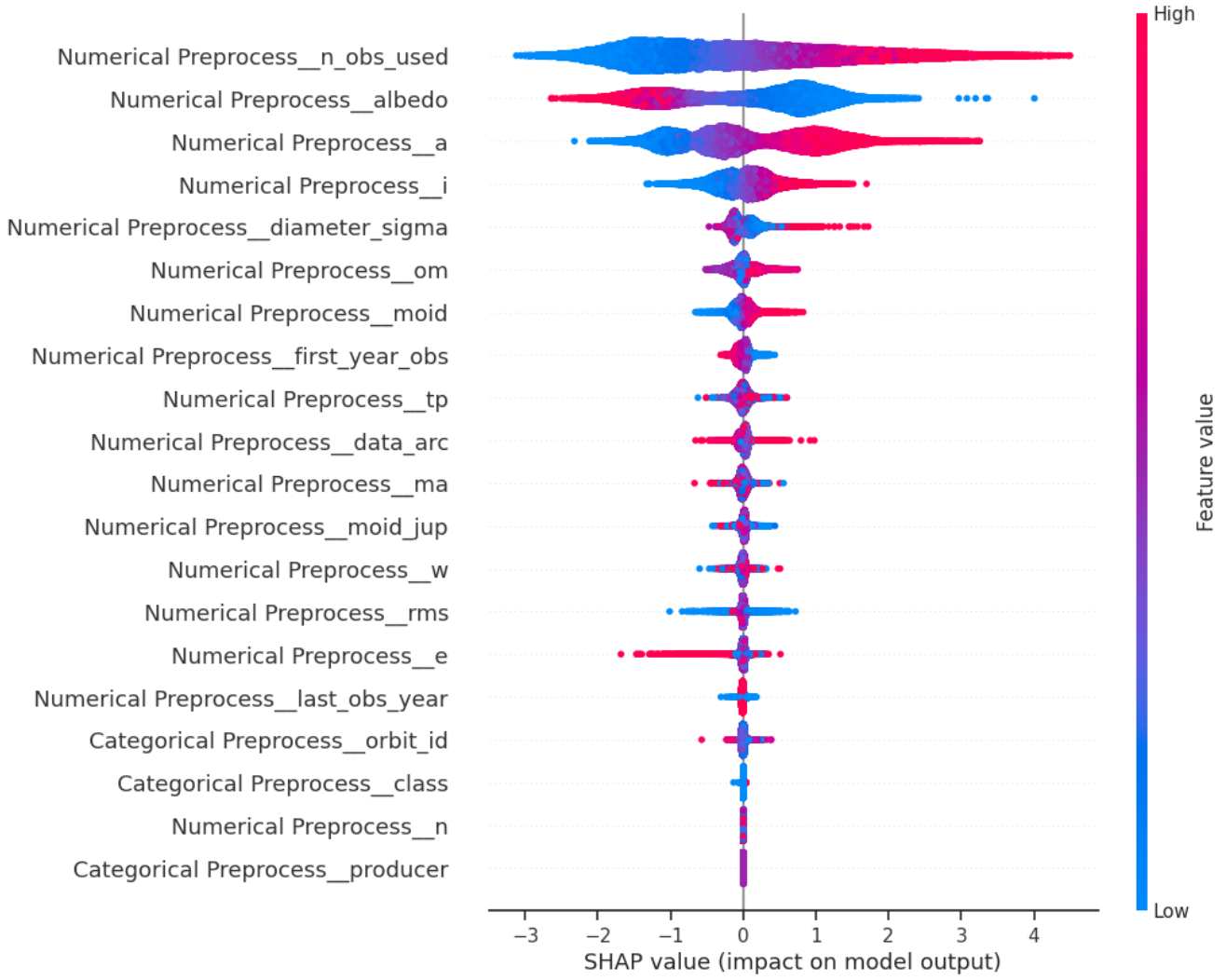


Figure 16 – *Shapley Summary Plot* (Made in Python)

There are many interesting insights that we can obtain, some of them are:

1. “n_obs_used” shows a consistent increase in the SHAP value with increase in its value.
2. “albedo” consists of a few outliers but otherwise, no other outliers throughout the data can be seen.
3. "e" has a strong negative impact when it has a high value.
4. "n" and "producer" have almost no effect on the output variable.
5. "diameter_sigma" has a strong positive impact when it has a high value.

Recommendations

Some issues with the implementation followed by Blakelobato (2020a) are:

1. They first perform one-hot and ordinal encoding on the dataset and then scale these features, this is not optimal and can affect the model's learning capabilities.
2. Some cells in the main notebook do not work because of errors

The implementation method followed a reasonable and efficient approach that wasn't too complex or time-consuming. There is still a lot that can be improved, such as:

1. Domain knowledge could be used to gather more information about these features, if they are useful for predicting diameter, etc.
2. Resolving the multicollinearity issues seen in the input data
3. Gathering enough data to have adequate samples for each category
4. Figuring out a better way to encode the dates
5. Performing Grid Search instead of Randomized Search to obtain better performing models

REFERENCES

Blakelobato (2020a, 9 January). Predicting asteroid diameter [Electronic resource: Python source code]. GitHub. Retrieved from https://github.com/blakelobato/Predicting-Asteroid-Diameter/blob/master/Main_Analysis_Notebook.ipynb

Blakelobato (2020b, 22 January). Predicting asteroid diameter [Electronic resource: Data]. GitHub. Retrieved from https://github.com/blakelobato/Predicting-Asteroid-DiameterDash/blob/master/model/Pred_Ast_Diam_2.csv

Blakelobato (2020c, 9 January). Shapley plot interactions [Electronic resource: data]. GitHub. Retrieved from https://github.com/blakelobato/Predicting-Asteroid-Diameter/blob/master/SHAPELY_PLOTS.ipynb

National Aeronautics and Space Administration (NASA). (2020, 6 October). *JPL small-body database search engine*. Retrieved from https://ssd.jpl.nasa.gov/sbdb_query.cgi