Electronics and Computer Science
Faculty of Engineering and Physical Sciences
University of Southampton

Miroslav Miroslavov Milanov

03.05.2022

Comparison of machine learning algorithms in predicting missing data in exoplanet survey

Abstract

In the search of other habitable planets, we've recently started to collect exoplanetary data. In order to make educated decisions about which planets to dedicate more research and attention to, scientists need data that is of high quality. Most of the data available to us currently, however, contains mistakes and missing values. To alleviate this problem, we need to spend more effort researching efficient ways to improve the quality of our data. This paper will focus on comparing different machine learning (ML) approaches on their efficacy in predicting missing exoplanetary data. This will be done by predicting said data and evaluating how well the different models perform based on accuracy. The results will then be analysed and some insights into the strengths and weaknesses of the algorithms will be given. Finally, a conclusion will be given about which algorithms perform well and what improvements could be done to the predictive algorithm.

Contents

1 Introduction

People have been looking at the sky and identifying planets from ancient times, however, it wasn't until the late 2000's that the number of identified planets drastically increased. Technological advances have allowed us to discover planets that we cannot detect using regular imagery. This mostly includes transit photometry (77%) and radial velocity (21%). The issue with these methods is that while astronomers have devised accurate mathematical formulas to extract data from simple signals such as a slight disturbance in the amount of light coming from a star, they are prone to errors and corruption of data. Such corruptions could be caused by different particles, cosmic radiation, gravity manipulations, or other forces we do not yet understand. Because of this, there are entries in databases such as the NASA exoplanet archive or the PHL-EC exoplanet catalogue which contain empty data or otherwise improbable corrupt data.

The majority of the data in the PHL-EC dataset has been collected by the Kepler telescope that was launched into space on the 6th of March 2009. Moreover, Kepler looked only at around 105 square degrees of the night sky [1]. The TESS telescope launched on the 18th of April 2018. While its range is shorter than Kepler – about 200 light-years compared to Kepler's up to 3000 light-years, TESS is capable of observing an area 400 times larger than Kepler – which is around 85% of the entire sky [2]. This means that soon we will have a significantly larger quantity of data than we do now, which we need to be prepared to process.

Assuming we can devise algorithms that work with accuracy comparable to or better than methods done by human scientists, we can be looking at models that are capable of predicting metrics such as planet habitability. All of the data that we can predict is important, but perhaps one of the biggest goals of exoplanet discovery is the finding of planets similar to ours. What is just sheer fascination with potentially habitable planets right now, can turn into researching which planet to colonize next in a couple of hundreds or thousands of years. The potential benefits of studying planets can be quite beneficial not only in the area of planetary colonization. By gaining insights into other planets, we might collect more information on topics such as the origin of life and its likelihood for example.

In any case, for all of this to be possible, we need accurate algorithms that are capable of fulfilling niche tasks.

2 Project Scope

The project aims to explore different machine learning algorithms and their efficiency at predicting gaps in exoplanetary data on the PHL-EC database.

The dataset in question contains 4048 entries with 112 features. Most of them are numerical, with a couple being categorical. For example – there is a feature called P_TYPE with values such as: “Terran”, “Superterran”, “Jovian”, etc. These features can be vectorized. What this means is that the problem of predicting these values can be reduced to a regression problem. While other solutions are possible – such as clustering or an unsupervised neural network approach, the data in the dataset is labelled and organized. Moreover, we know what we’re looking for in data at every single point of algorithm execution – the prediction of particular features. Finally, the accuracy of unsupervised approaches can vary as it requires a lot more validation than a supervised approach. In specialized fields such as exoplanet discovery, this would require extensive knowledge about astrophysics.

For the reasons listed above, this study will explore and compare different regression algorithms and their efficacy in predicting the values from this dataset. The aim will be to collect as much data as possible for several algorithms, visualize it, compare it and make conclusions based on the result.

The code for this solution will be written in Python for use of libraries such as Pandas, Numpy, and Scikit-learn. The regression ML algorithms in Scikit-learn can only support datasets where all input data is fully labelled. Our dataset, however, contains data that is Missing at Random (MAR). It is not completely at random, because, as this paper will explain later on, there is some connection between the missing data.
This causes some difficulty in prediction using those algorithms since sometimes input data will be missing. Because of that, a proper imputation method needs to be discovered and employed. This needs to be done carefully as some algorithms are biased towards some particular type of data.
With a proper imputation method selected, an algorithm needs to be created that takes the input data and prepares it with imputation so that the values can be predicted. This study will propose an iterative algorithm that predicts data row by row in order of fullness of entries, to maximize the effectiveness of predictions so that it introduces as minimal bias as possible.

Finally, a method of collecting information about the accuracy of these algorithms needs to be devised. Due to the nature of the algorithm that this study will employ, the most natural accuracy metric that could be employed is a mean average error. As we make predictions during the algorithm, we will predict data that is already present in the original dataset. We can compare this data to our predicted data, calculate the mean error and store it. Finally, once we’ve predicted the whole dataset, we can calculate the average mean error in one of two ways – compute the error of the whole feature using a weighted scaling of the mean errors on each iteration or by comparing prediction results on previously expunged data. This will be further explained in the algorithm section of this study.

3 Background and report of literature search

3.1 Exoplanet related

Studies in the area of exoplanet discovery and analysis are relatively new and as such the topic isn't as rich in scholarly papers. However, it is exactly for that reason that this topic deserves a lot of attention.

Perhaps the primary interest in research papers on the topic of exoplanets is the classification of the habitability of planets [3]. Our best tools right now are different mathematical formulas such as ESI (Earth Similarity Index), MSI (Mars Similarity Index), and TIA (Tardigrade Index Approach). We are good at constructing complex mathematics to solve problems for us, but as we have observed, computers are capable of finding even more finely tuned dependencies between data.

According to Kashyap et al. [3], a good indexing approach is the Earth Similarity Index (ESI). The ESI is a mathematical formula that compares the numerical values of features of an exoplanet and the Earth and evaluates how similar they are. According to the same paper, Mars with an ESI of 0.73 falls within the list of potentially habitable planets (PHP). Some argue that the ESI of Mars is optimistic – suggesting that most planets with an ESI below Mars shouldn't be classified as habitable. In any case, it is accepted that generally, most planets in the PHP list have an ESI above 0.75.

Two other papers exist with similar mathematical approaches [4], [5]. The first of the two evaluates planets based on conditions for life of rock-dependent extremophiles and utilizes an index called the Rock Similarity Index (RSI). A relatively similar approach to the ESI papers is used. The main difference between the two is the metrics that take part in the formulas and the impact they have on the final result.

The third paper proposes two other indexes – the Active Tardigrade Index (ATI) and the Cryptobiotic Tardigrade Index (CTI). A Tardigrade, also known as a water bear, is an extremely resilient organism, known to survive in extreme conditions. The formula for both ATI and CTI can once again be calculated by referencing their exoplanetary features to those of the Earth.

There is an obvious connection between all of our available approaches for classifying exoplanets. Lacking measurements from the planet itself or near it, we are left with only one choice to measure its habitability – compare it with the only habitable planet we do have measurements of - Earth.

This means that we require high-quality, non-erroneous, full data. In reality, however, as we will see in the analysis of the dataset, our data does not exactly fulfil those requirements. As such we need to look into what are the optimal solutions to fix that.

3.2 Artificial Intelligence related

One paper bears similarities by attempting to predict another formula called the CEESA using machine learning [6] - CEESA being the Constant Earth Elasticity Similarity Approach and works in a relatively similar way to the ESI. Most of the papers look into the problem from the lens of physics and mathematics. As computer scientists, we can identify that this problem is a very likely candidate for a solution revolving around machine intelligence. There are plenty of papers that analyze different machine learning algorithms which we can use to derive an optimal algorithmic approach to the problem e.g., Kavitha et al. [7]. We can look through

papers that solve similar problems and identify their strengths and weakness at problem-solving. It is imperative when researching such papers to ensure similarly structured data is being used. Some papers written by Jozef et al. [8], Wen-Jing et al. [9], and Somayeh et al. [10] use similarly structured data to the one in this problem; There is an abundance of numerical data processed by ML algorithms and most of them use a regression approach. A paper by Ki-Young et al. [11], focuses on numerical data but contains a small portion of categorical data that could prove useful.

Two papers, one by Anil et al. [12] and another by Christos et al. [13], provide interesting insight into what imputation methods work well in different datasets. We will look into these two particular papers later on when we discuss the imputation method used.

4 Database

4.1 Introduction

The database is issued and indexed by the Planetary habitability laboratory UPR Arecibo. The database was publicly issued and supported until late November 2021 when the project was slightly changed. This paper will make use of the August 2021 edition of the database.

The database contains 112 features, 68 of which are distinct. The other 44 are 22 pairs of parameters that describe error ranges. 38 (including error margins) of those 112 are related to the host star. The rest are related to the planet.

While it is important to predict as much as possible from the data, some features offer us little in terms of information and others could confuse the regression algorithm. We will need to perform a dimensionality reduction on the dataset if we want to eliminate noise.

4.2 Analysis

We can break up the different features into groups based on importance:

- (K) - Key identifiable data
- (Y) - Yes - high importance
- (H) - Helpful information
- (M) - Maybe contains helpful information
- (N) - No - low to no importance

The purpose of this classification was to identify data that could be immediately expunged from the working database due to its nature (N). This included metadata such as:

- "Planet discovery year"
- "Entry Last Update"
- "Planet alternative names"

Some examples from the (Y) group include:

- "Planet Mass"
- "Planet Radius"
- "Planet Density"
- "Planet Gravity"

Some examples from the (H) group include:

- "Planet Escape Velocity"
- "Planet Mean Distance from Star"
- "Planet Periastron"
- "Planet Apastron"

Some examples from the (M) group include:

- "Planet Eccentricity"
- "Planet Orbital Inclination"
- "Planet Measured Equilibrium Temperature"

The (K) group is reserved for only a few fields such as:

- "Planet Name"

- “Star Name”

Which can identify the entry in question.

This classification helped reduce the database to 97 columns. Furthermore, 36 of those columns are error columns. This means we are left with 61 data columns. 2 of them are keys. We have 59. On those 59, a numerical analysis was performed to find out which of the data columns have the highest density of valued cells and which of the columns contained the same quantity of cells. Those would therefore be linked.

The result is as follows:

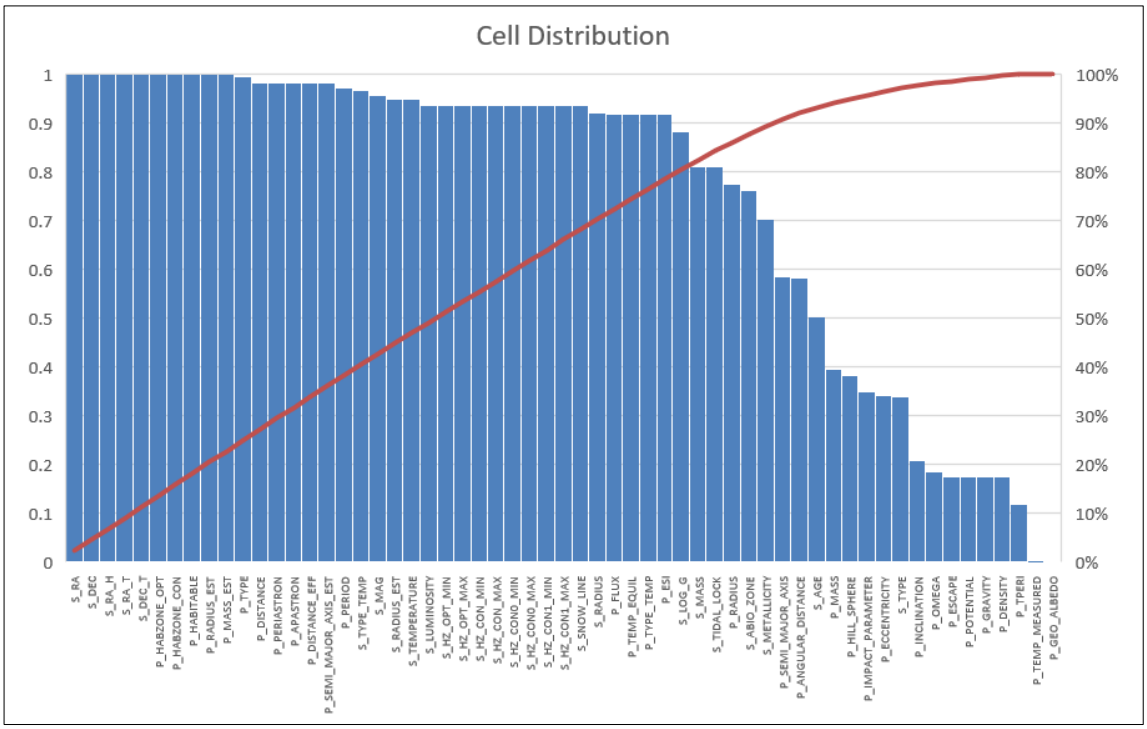

Figure 1:Cell Distribution of different features. X-axis shows features in the PLC-EC database. Left Y-axis shows how densely populated a feature is. Right Y-axis shows the cumulative amount of filled cells in all features.

Further reducing the dimensionality of the dataset by removing uninformative features leaves us with a dataset with 48 features. This is without key identifiable data, error fields, or metadata. Here is the list of features:
['P_MASS', 'P_RADIUS', 'P_GRAVITY', 'P_DENSITY', 'P_ESI', 'P_RADIUS_EST', 'P_MASS_EST', 'P_SEMI_MAJOR_AXIS_EST', 'P_ESCAPE', 'P_POTENTIAL', 'P_HILL_SPHERE', 'P_DISTANCE_EFF', 'P_PERIOD', 'P_TYPE', 'P_SEMI_MAJOR_AXIS', 'P_ECCENTRICITY', 'P_INCLINATION', 'P_OMEGA', 'P_TPERI', 'P_ANGULAR_DISTANCE', 'P_IMPACT_PARAMETER', 'P_DISTANCE', 'P_PERIASTRON', 'P_APASTRON', 'P_FLUX', 'P_TEMP_EQUIL', 'S_TEMPERATURE', 'S_MASS', 'S_RADIUS', 'S_AGE', 'S_TYPE_TEMP', 'S_LOG_G', 'S_RADIUS_EST', 'S_DISTANCE', 'S_MAG', 'S_METALLICITY', 'S_LUMINOSITY', 'S_HZ_OPT_MIN', 'S_HZ_OPT_MAX', 'S_HZ_CON_MIN', 'S_HZ_CON_MAX', 'S_HZ_CON0_MIN', 'S_HZ_CON0_MAX', 'S_HZ_CON1_MIN', 'S_HZ_CON1_MAX', 'S_SNOW_LINE', 'S_ABIO_ZONE', 'S_TIDAL_LOCK']

It is important to know that the database contains a lot of empty cells. The number of missing cells divided by the total cells in the original dataset is roughly 0.33. The number in the final selected dataset is roughly the same – 0.34. This is problematic because it is difficult to

perform accurate predictions on such a small sample size - 4048. We can further break down the numbers into columns and rows to get a better idea of the data distribution.

Each column or row has a fille value between 0 and 1. 1 meaning that the row/column is fully filled and 0 – that all values are missing.

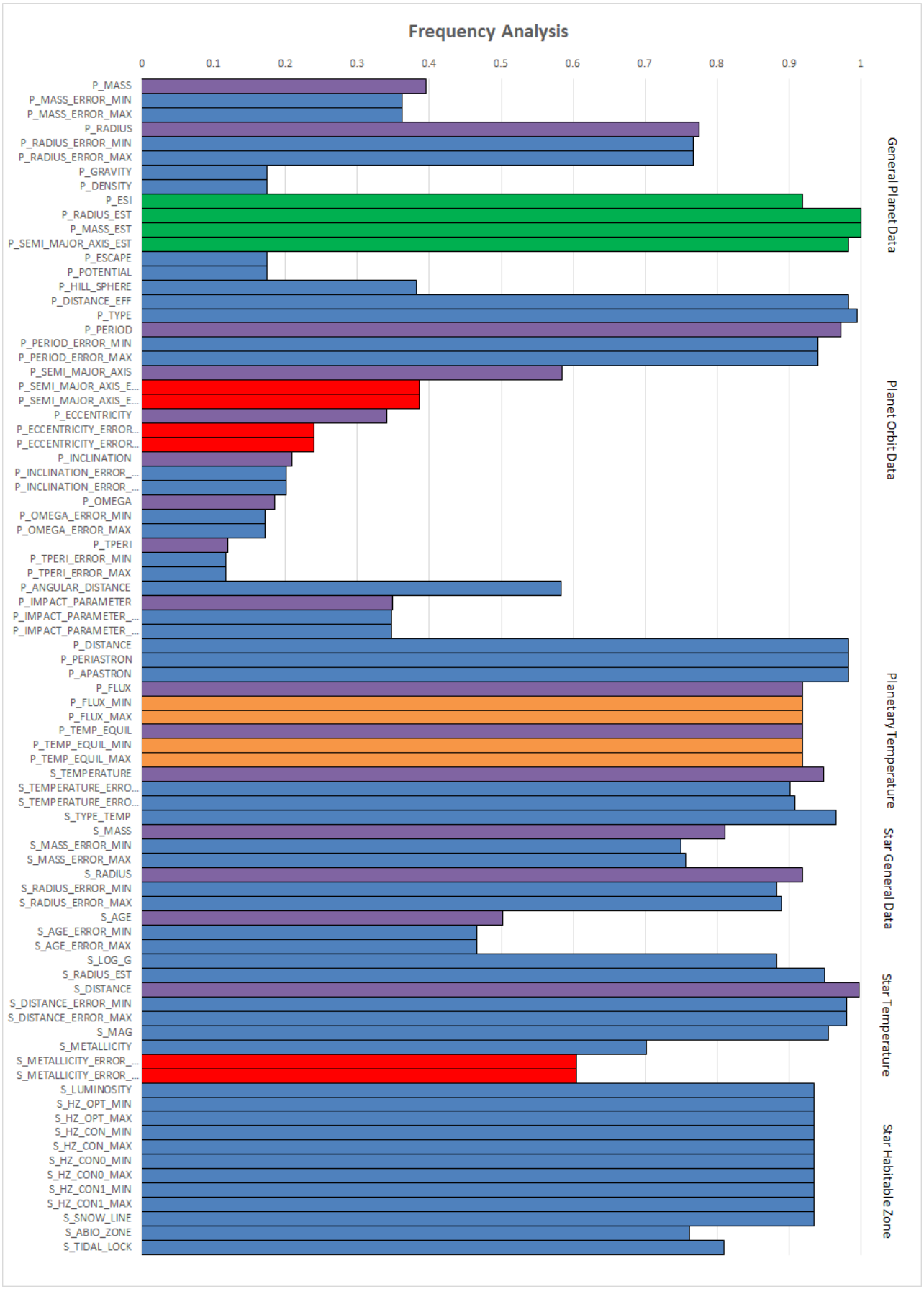

Figure 2: Feature Fill Values

Legend:

Purple-colored – This feature has error features – the two following bars.

Green-colored – These features are not observed, but manually calculated and inserted by the people at the university of Paraceibo. Therefore, they are less prone to mechanical error or physical abnormality but have the possibility of human error.

Red-colored – These error features are significantly less filled compared to their parent features. While we will not use error features to make predictions or calculate accuracy, these features might have a hidden implication for the accuracy of the original data. In any case, it is impossible to say without an expert's opinion.

Orange-colored – These error features have a 1:1 fill ratio with their parent feature. While this doesn't initially raise any alarms, in the "database issues" section later on we will discuss a serious problem with some entries – values being copy-pasted from one entry to another. This is a feature that is more susceptible to this mistake.

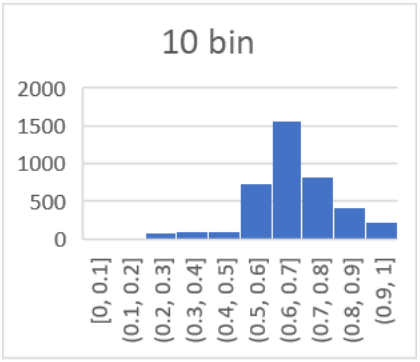

Figure 3: 10 Bin Row Fill

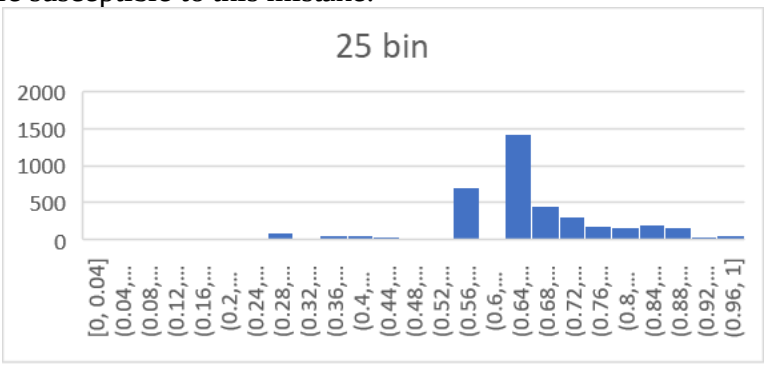

Figure 4: 25 Bin Row Fill

Figure 2 shows us the fill values of the features. Similarly, some features have quite a low value – TPERI, INCLINATION, OMEGA, etc.
As such we can expect a slightly lower accuracy from some features compared to others.

From Figures 3 and 4 we can see that the majority of rows have a fill value between 0.5 and 0.8, with a small distribution of rows having around 0.2. This will play a major role in deciding which imputation method we will go for later on.

4.3 Issues with database

There are a few obvious problems with the database. The three figures above reveal the varying amount of fill values in features and rows. It is hard to pinpoint an exact reason for the variety of fill values. If the original dataset without any dimensionality reduction had the same visual representation as the one in Figure 2 and they were sorted by their fill value, the resulting graph would look like a sigmoid. Figures 3 and 4 have a very close resemblance to a Gaussian distribution. If there was an order that dictated which values are missing it would be easier to identify the proper algorithms to ensure high accuracy in prediction.

Some features have an exceptionally low fill value. The less populated a feature is, the harder it becomes for a machine-learning algorithm to predict it. Some machine learning algorithms perform better on lowly populated features compared to others. This just raises the questions – what algorithms and imputation methods do we use for this database? What combination can we use to ensure high accuracy? How do we ensure that we do not introduce bias if we use different algorithms for different features? We will look into these questions later on.

The more you observe this database, the more errors you begin to discover. Some of these are mechanical, some are logical and some may be human error.
In any case, the database contains a lot of information that can mislead a machine learning algorithm into predicting bad data.

The least severe issue is that some features have a mathematical dependency on others. For example – mass depends on density and volume. There are some features where such a calculation can be made, but it is missing. It is perhaps then a better idea to calculate it using a mathematical formula and use it to predict other data with higher accuracy.

On the other side of this spectrum – sometimes there are values entered, which should not have been. For example – if you sort P_RADIUS_EST by ascending order, the first 15 values are 0s. However, it is a logical fallacy that a radius can be a zero. This could be a human error – wrongfully inputted data, or a mechanical one – bad parsing from NaN to 0s. Regardless, this data is not correct.

There are purely mechanical errors – S_AGE has a couple of entries that have a value around the negative long integer overflow: -2147483600. Unfortunately, this data is not recoverable.

It was mentioned earlier that some values had been copy-pasted from other entries. This could make sense for data regarding the star of a planet since they could be orbiting the same celestial body, however, some of this was found in planetary data. Several features had identical 1:1 values with several digits after 0. Having two planets with the same measurement and that particular accuracy is incredibly unlikely in such a small dataset, let alone having several features be the same. We can only conclude that this is erroneous.

There have been plenty of logical violations as well. Most of the features in the dataset are in the set of positive real numbers. This means any entry in those features that is negative or 0 is erroneous and there have been several violations. Some features follow a strict range – somewhere around 0 and 1. S_AGE is bound by the age of the universe. Other features are radial - therefore their range is anywhere between -360 and 360 – e.g., P_INCLINATION, P_OMEGA.

There have also been features with other logical implications. For example – P_ECCENTRICITY is a measurement between 0 and 1 inclusively about how irregular a circle is. In this case, the feature measures the orbit of the planet. A value of 0 means that it is a perfect circle. A value of 1 means that it is a straight line. A value of 1 would be theoretically impossible as the planet wouldn't have an orbit. A value of 0, while plausible, begs the question of how likely it is to happen? Common intuition suggests that it occurring even once in the dataset is a probabilistic miracle, let alone having a fifth of the eccentricity feature be 0s. For the sake of simplicity, this study assumes that this is erroneous information.
Moreover, on this topic, S_AGE is measured in billions of years. The age of our universe is ~13.787 billion years. S_AGE contains a few entries that have a value higher than that.
One planet had a planetary(orbital) period of about 7 million years. Its other values such as orbit metrics had nothing abnormal about them (large or irregular orbit)– which suggested an erroneous measure.

Finally, some features in the database exhibit a property where the first few values in descending order were many times larger than the mean average of the feature.

Table 1: Descending Values Analysis of the 20 highest values post-manual processing of selected features.

P_GRAVITY	P_SEMI_MAJOR_AXIS	P_HILL_SPHERE	P_ANGULAR_DISTANCE	S_ABIO_ZONE	S_TIDAL_LOCK
1254.45	2000	443.3371	42600	488.9324	1.322542
978.9127	800	373.5275	41700	383.8253	1.024134
923.0125	773	281.5678	12400	246.1885	1.02069
699.5122	670	143.2288	8030	99.15991	1.02069
629.2409	650	86.20447	6290	60.59814	0.961746
539.8619	487.1	72.49746	5710	52.473	0.951012
497.0646	440	57.01097	5190	51.38448	0.871009
269.9309	345	53.43772	4640	45.9061	0.747861
202.6267	330	47.56128	3960	42.59173	0.71354
182.7667	330	47.29474	3110	39.67354	0.700445
180.7852	320	43.20129	2670	38.49801	0.680539
142.0141	260	41.48673	2480	25.38719	0.680539
114.3475	230	37.83864	2360	25.37044	0.671951
110.6	210	37.7998	2360	23.05033	0.668306
88.36359	210	22.29272	2280	21.86212	0.668306
85.18138	157	20.8686	2210	21.79384	0.648683
84.0727	156	18.36329	1750	20.79799	0.648683
81.74756	102	13.99226	1440	19.61794	0.647905
80.02906	100	10.43837	1160	19.53186	0.6424
77.33877	92	7.383874	1100	19.32438	0.6424

This is a small slice of some of the features with more significant differences. In reality about half of the features exhibit this property. Each column represents the twenty highest entries in a given feature (After manual pre-processing). As you can see, the difference in values between the highest and 20^{th} highest is significantly higher than the mean. While such distribution is completely plausible in a Gaussian distribution, this introduces some inconsistencies in our dataset.

5 Implementation

5.1 Pre-processing

The bulk of the pre-processing has been done manually. By analysing the dataset, we can identify the obvious errors and remove them. This was done using qualitative analysis of the different features – their nature, their ranges, and any possible connections with other features. The result was the errors outlined above.

Two techniques were used in the manual pre-processing

5.1.1 Soft data expunging

When erroneous data that warranted removal was found but was not significant enough to cause the whole entry from being expunged – only that given value was removed from the database. This ensured as much data as possible was maintained while also improving the quality of the dataset. This was the case for some entries with feature violations such as minor range violations, presumed misinterpretation of NaNs as 0's, integer overflows, etc.

5.1.2 Hard data expunging

When erroneous data that warranted removal was found and it was significant enough that the whole entry was deemed irregular, the planet was removed from the selection. This was the case for entries with systemic range violations, major human/mechanical errors such as copy-pasting of values, and irreparable logical violations.

The total amount is of planets removed this way is 40 and the list is:

["K2-296 b", "GJ 1061 d", "K2-296 c", "GJ 1061 c", "GJ 1061 b", "GJ 687 b", "HD 217850 b", "HD 181234 b", "bet Pic b", "HIP 67851 b", "HIP 14810 c", "HD 102117 b", "Kepler-20 d", "WTS-1 b", "Kepler-238 b", "HAT-P-70 b", "Oph 11 b", "Fomalhaut b", "HR 8799 b", "WD 0806-661 b", "Ross 458 c", "Kepler-411 e", "Kepler-447 b", "Kepler-51 c", "K2-266 b", "KOI-55 b", "KOI-55 c", "WASP-183 b", "NN Ser c", "NN Ser d", "NSVS 14256825 b", "V0391 Peg b", "kap And b", "HIP 78530 b", "HD 100546 b", "KELT-9 b", "tau Cet e", "tau Cet f", "tau Cet g", "tau Cet h"]

As you can see, it was often the case that planets orbiting the same star were more likely to exhibit erroneous properties.

5.1.3 Programatic expunging

One of the bigger challenges of selecting the 48 features that will take part in the predictive algorithm was to find dependencies between features and make the selection such that no feature was left without any dependent features by which predictions to be made. Figure 2 shows some of the high-level dependencies between different features. The problem is that in reality, most of these dependencies are subtle and hard to detect by human analysis. Other times features are ignored or given less weight under the impression that they do not have a significant impact on other features. However, it is difficult to know for certain without meticulous analysis from an astrophysical perspective.

This highlights the issue brought up in Table 1. We cannot know for certain what the exact dependencies are between features. They certainly aren't linear; they might not even be polynomial. Outliers such as the ones in Table 1 have a significantly higher weight in the prediction algorithm. The following will be explained fully in the algorithm section of this study, but briefly – due to how the dataset is filled, we have to build the predictive algorithm iteratively from full entries to less full ones. This means that a small error in prediction in early iterations is propagated in later stages. This was quite evident during the building of the algorithm. Some test cases had mean error values that scaled up from near the mean of a given feature (in the first few iterations) up to 10^{10} – 10^{30} (in the last few iterations).

The brute-like solution to this problem was to remove these outliers. The ten highest valued entries of each feature were expunged from the dataset. This means that for the 48 features selected, from the 4048 features 480 plus the 40 from the manual erroneous pre-processing were removed. This is about one-eight of the dataset and represents a significant portion of it. We will get into more details about this issue in the Critical Evaluation section of this study.

What is important is that without the erroneous information and with the problematic outliers expunged, an algorithm could be coded that makes accurate predictions. Without the outliers, in the testing phase, non-linear ML algorithms no longer go into propagation errors, while the linear regression one went from having more than half of the features scale up to several magnitudes higher than the mean, to having just a few of the features propagate. While still problematic, there is a solution that enables even the linear algorithm to achieve non-erroneous predictions, which will be discussed later on.

The next step in the pre-processing part of the algorithm is the vectorization of the two categorical features in the selection – P_TYPE and S_TYPE_TEMP. This is because the regression algorithms work only on numerical data.

Finally, and this touches slightly upon the issues highlighted by the outliers, the dataset was normalized. The dataset has a wide variety of data that has different ranges and means. There are three general groups of data by ranges – those ranging between 0 and 1, those in the range of positive real numbers, and the irregular group (Radial, -infinity to +infinity, irregularly distributed data). In a dataset without normalization, a feature that ranges in the hundreds of millions will be much more sensitive to weight adjustments during the training of the algorithm. For example, the feature P_TPERI ranges between roughly 2,400,000 and 2,500,000 and its weight will behave differently than a feature ranging between 0 and 1. Normalization enables the ML algorithms to make accurate predictions due to having a strict restriction on the range of all data – between 0 and 1. This way all weights in the underlying algorithms are equally as viable. The above is especially true in this example where we will use default ML algorithm setups without any hyperparameter tuning.

5.2 INK Algorithm

INK stands for Iterative N-K.

5.2.1 N Iterations

The basic idea of this algorithm is that we will iterate over the dataset based on how filled entries are in descending order.

Table 2: Cell Fillage Example. Each row is calculated using the CF formula.

S_MAG	S_TIDAL_LOCK	Cell Fillage
7.87	0.402374	1
9.376	0.526231	1
10.612	0.530603	1
8.25	0.482546	1
	0.468248	0.987805
	0.520143	0.987805
13.944	0.450583	0.963415
15.63	0.519296	0.963415
13.69	0.475708	0.963415
10.812	0.429058	0.963415

The exact formula for the Cell Fillage(CF) is :

$$\frac{\#Cells\ that\ are\ filled}{\#Cells\ in\ total}$$

The dataset can have at most N-1 distinct values of Cell Fillage, where N is equal to the number of features – in this case, 48 (an entry cannot have a fill value of 0, as it would not be an entry). For each iteration, we will predict a selection of the entries that have a given Cell Fillage value, starting from 1, until we get to the end of the dataset. And since an entry with a Cell Fillage value of 1 is already fully intact, we can skip predicting that one.

This means that the N iterations loop can have at most N-2 iterations. The loop will begin with a Cell Fillage value of:

$$\frac{N-1}{N}$$

To predict this selection of entries, we will build a model using the selection with:

$$\frac{N}{N} = 1\ \ Cell\ Fillage$$

When we successfully predict the values, we update the dataset with the new entries and we go to the next iteration where we will predict selections with Cell Fillage of:

$$\frac{N-2}{N}$$

We build the model on the now bigger selection of entries with a Cell Fillage value of 1. We repeat this process until we reach the end of the dataset. Worst case we go through N-2 iterations, though in this dataset there are fill values that aren't present, so the iterations are less than that (At most around 30-35).

5.2.2 K Iterations

One issue that arises during this N-iteration process is that sometimes different combinations of features are missing from entries with the same CF value:

Table 3: Example dummy dataset to visualize the K-step process. Non-representative of actual data.

S_AGE	S_MAG	Cell Fillage
3.7	7.87	1
	15.22	0.987805
0.32		0.987805

In this example, we have a small sample dataset with only two features – S_AGE and S_MAG. Two distinct features have missing values. We cannot use a regression model to predict either of them because our input would have missing values. This is where we use imputation. We are going to iterate through each feature that is missing, impute all other data and predict the given feature. We repeat this K times for each column that is missing, where K is equal to:
$N - \#Cells\ that\ are\ filled.$

Here is a step-by-step process of this in action:

1. First, we select **S_AGE**.

S_AGE	S_MAG	Cell Fillage
3.7	7.87	1
	15.22	0.987805
0.32	**1.23**	0.987805

We need to impute everything else. Note that this is crafted data and is not representative of the actual dataset or the algorithm's predictive qualities.

2. We build a model on the entries with a fill value of 1 and we predict S_AGE.

S_AGE	S_MAG	Cell Fillage
3.7	7.87	1
5.8	15.22	0.987805
0.32	**1.23**	0.987805

We store any predicted values and we discard all changes.

S_AGE	S_MAG	Cell Fillage
3.7	7.87	1
	15.22	0.987805
0.32		0.987805

3. We iterate to the next step – **S_MAG**.

S_AGE	S_MAG	Cell Fillage
3.7	7.87	1
7.202	15.22	0.987805
0.32		0.987805

We impute all other features.

4. We build another model that predicts S_MAG and use it to calculate the value.

S_AGE	S_MAG	Cell Fillage
3.7	7.87	1
7.202	15.22	0.987805
0.32	**1.12**	0.987805

We then store the predicted values and discard all changes.

It is important to note that when predicting a given feature we will get prediction values regardless of whether or not a cell is filled. What that means is that for this example in the S_AGE iteration we would get this array:

S_AGE
3.7
0.32

Original

S_AGE
3.2
5.8
0.62

Predicted

What we can then do is put the predicted values in the dataset of those cells that have been missing and compare the predicted values of those that we already had to get an idea of how accurate our algorithm is.
The new dataset would look like this:

S_AGE	S_MAG	Cell Fillage
3.7	7.87	1
5.8	15.22	1
0.32	**1.12**	1

We can also calculate that the mean error of S_AGE is (in this example):

$$\frac{0.5 + 0.3}{2} = 0.4$$

Ultimately this boils down to a doubly-nested for loop with a different prediction model for each K[th] iteration.

To fully predict a dataset at most:

$$(N-2) * K = (N-2)(N-1) \text{ iterations are required}$$

The reason it was decided to make and use this algorithm instead of making predictions using a simpler algorithm – for example just doing the K steps over all features, imputing the whole dataset, and predicting each feature one by one – is that this gives the most insight into how accurate a regression models is at predicting the data.

The algorithm such as the above example would not focus hard enough on the actual prediction as opposed to imputation. The INK algorithm maximizes the number of predictions we make and minimizes imputation by slowly filling up the entire dataset. That way imputations are limited to the least amount of information – just enough to enable an ML algorithm to make predictions.

That being said, this algorithm does have a weakness. Any impurities in the original dataset have a significant impact on the efficacy of any ML model using the INK algorithm. A normal model would make a single prediction and erroneous information would have a small impact on the output. In INK, making predictions using erroneous information predicts slightly erroneous information. For reference, the number of entries that are completely full in the original 4048 entries is 4. That means that the whole dataset is built upon these 4 initial entries. Should they or any of the first few iterations have any significant errors in the values of features, these errors will propagate and become significantly more erroneous towards the later iterations. We've already talked about this in the "Issues with Database" section, but more examples will be shown in the results.

5.2.3 Improvements

Before moving on to the next section it would be beneficial to talk about possible improvements to the algorithm. Initially, when designing the algorithm, one of the ideas was to include a "mid-processing" step between each N^{th} iteration. The purpose of this mid-processing would be similar to the manual pre-processing at the beginning – except this time done by the algorithm. The predicted values would be checked for any range violations. For example, it was mentioned that most features are positive real numbers. Any negative numbers would be in violation. A "mid-processing" step could change these erroneous predictions into values that are within the bounds. E.g., a -7 could become a 0.01. A -200 could become a 0.0001. These resulting values would scale proportionally to the absolute difference from the predicted value and 0. A similar method could be employed for features bound on the upper scale such as S_AGE which goes up to ~13.787, or P_ECCENTRICITY, which is between 0 and 1.

The issue with this approach is that it introduces a similar problem to the one in the "Issues with Database" section. These values would skewer the distribution of values in a feature, causing some values to become significantly larger than the mean than they previously were. During the following N iterations, the mean error kept increasing and the predicted values increased exponentially causing a similar propagation.

A similar but improved method could utilize natural selection. Predictions that aren't within range or violate some other logical rules are discarded. This means no obvious erroneous information will be included in the selection upon which models are built. On the other hand, though, this would mean more iterations will be required. That is more time spent and higher computational costs. Another problem might arise, where the algorithm gets stuck predicting the same value and always giving a prediction that is out of range. However, the price of time

and computation might be worth it, especially for algorithms where there is a higher mean error such as Linear Regression, due to there being more range violations.
One last concern with any type of "mid-processing" is the possibility of losing out variance.

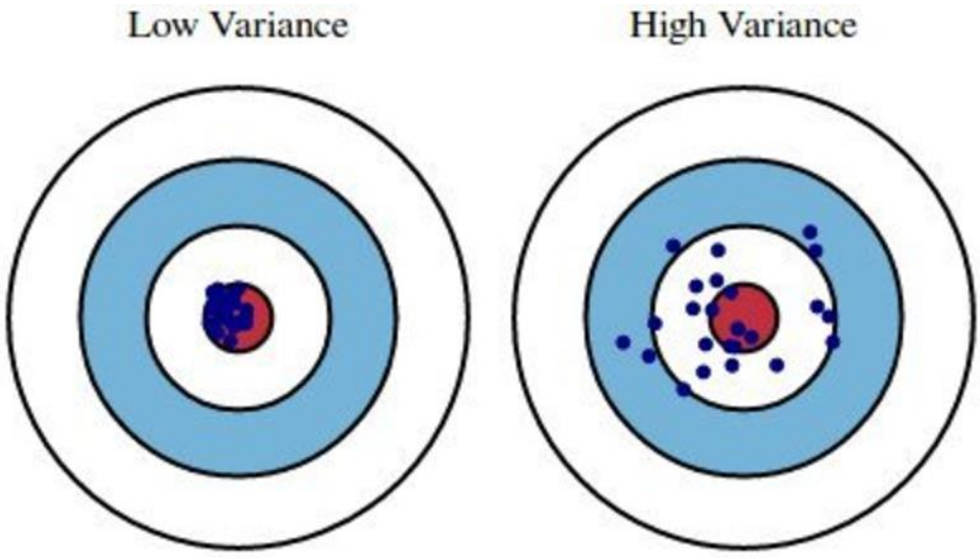

Figure 5: Difference between low and high variance. Source - shorturl.at/afgzQ

A feature will have a given variance by default. An ML algorithm will try to keep that variance and by removing some values from the result, we might change that variance. Worse, if a feature is limited on one end – for example positive real, and we remove negative values, the positive ones might grow, while the ones near 0 remain the same. This has the possibility of introducing bias as well.

A third approach could have the predictions inside the INK algorithm remain the same, however, once the whole dataset is predicted, the same natural selection could be done to remove any erroneous information. The new dataset with fewer missing values could be run through the same INK algorithm. Each time it is run through the algorithm, there will be fewer and fewer missing values.
This is by far the costliest implementation of the error removal, but in theory, it should be the safest when it comes to not introducing variance and bias.

5.3 Data Collection

In the example of INK, we went over how we would gather information on the efficacy of predictive algorithms at filling in missing data.

Each feature will have 10% of its data expunged (independently of one another in different cases). The INK algorithm will be run, utilizing different ML regression models. During the actual running of INK, data will be collected on each K^{th} iteration. The actual values will be compared to the predicted ones, mean error will be calculated and averaged over each iteration and then appended to an error Dataframe

Here is the structure:

Table 4: Algorithm Error Collection Dataframe

Iteration	SampleCount	P_MASS	P_MASS_Weight	P_OMEGA	P_OMEGA_Weight	S_AGE	S_AGE_Weight	…
56	51	437.8077577	50	108.0128	50	3.260638	47	
55	150	39.92160307	144	128.1982	19	1.220903	144	
54	70	8.123116475	56	208.7789	9	0.80973	37	

For each feature, we collect the mean average error as well as the number of cells that that iteration has predicted. The mean error can then be used to gather insights into the efficacy of different models at different iterations of the INK algorithm.

Additionally, we can use the 10% of data that has been expunged by comparing it to the predicted values. This will give us an estimate of the mean average error of a given ML model over the whole feature of the dataset. An alternative solution to finding the MAE would be:

$$\frac{\sum feature\ mean\ error * feature\ weight}{\sum feature\ weight} \mid The\ sums\ being\ over\ the\ iterations$$

Either should be capable of providing an accurate efficacy metric of the different ML algorithms. This study will make use of the 10% data expunged method.

We can build the collection of all errors in the following way:

Column	Mean_Error
P_MASS	26.99427554
P_RADIUS	0.021857625
P_GRAVITY	0.367945132

Having both of these DataFrames is important because it will let us get insights into how the algorithm performs overall and how well they perform on a given feature over the iterations.

It should be noted, however, that using the 10% data expunge method introduces slight variability in the results. This is especially evident in the Linear Regression model which was more likely to have erroneously high prediction values. As such, in order to receive non-erroneous results, in particular in the LR model, the algorithm will run several times and the highest result will be taken into account. Ideally, a mean of the results would be taken, however, in erroneous results, some of the features will have mean error values of several magnitudes higher than the mean, which would skewer the results. It is obvious which ones are erroneous when the results are looked at, however it is difficult to give an exact boundary value for when a result becomes erroneous. E.g. for a mean feature value of 100, is a mean error of 10 erroneous? Is 100 erroneous? Is 1,000 erroneous? Etc.

Additionally, the 10% data expunge method requires the algorithm to run over all 48 features one by one, expunging data from them individually. While more costly in terms of time and computation power, this method offers a higher quantity of data.

6 Design choices

The choice of algorithm space, as discussed earlier, will be in regression models. We have a simple dataset with labelled data, some of which is missing. The question becomes – which algorithms to compare?
A good selection of algorithms is required to not only find the most efficient algorithms to use but to also gather insight into what types of algorithms perform well on this type of task.

An important decision has to be made regarding what imputation method to use. Imputation methods have varying efficacy based on the dataset nature in question, the quality of the data, and how full the data is.

According to Anil et al. [12], the kNN method outperforms the other imputation methods. In a study performed by Christos and Georgios [13], along with kNN, other more complex imputation methods have been used such as mice, missForest, Iterativeimputer, etc. The above study analyzed several datasets of which two have some similarities in nature and size to the PHL-EC dataset. Those are the Tadpole and Gestation datasets. The two are roughly twice as big as the exoplanet dataset, having most features be positive real data with a couple of features in each dataset that contained very large numbers. This resembles what is seen in our exoplanetary data.

While the study doesn't find a clear superior algorithm that ranks consistently above others, two algorithms are generally observed to perform well – kNN and missForest. kNN seemingly scores better on data that has less than half of it missing and missForest is generally better when there is less than half of the data.

According to figure 2, and if we ignore the MIN/MAX error features, the features that have less than a 0.5 Cell Fill value constitute about a third of all fields.

One approach is to use Knn for features that have a fill value of more than 0.5, while missForest is used on features with a fill value of less than 0.5.
In theory, however, only one imputation method should be used. Using several imputation methods for different features is bound to introduce bias in the predictions. To preserve the integrity of the results, only one imputation method will be used for the whole dataset.

While either imputation method should provide us with accurate and proportional results, due to the findings of the studies listed above and the fact that the majority of features in the dataset have a Cell Fill value of more than 0.5, Knn will be the imputation method of choice.

6.1 Algorithms

6.1.1 Linear Regression

Linear Regression is a classic choice when solving a regression problem. It is simple to use and powerful enough to solve most problems. As the name suggests, it is most accurate when used on linear problems, which as we discussed is probably not the case in this dataset. Its use is to primarily set an example of what to expect as output.

6.1.2 Ridge Regression

The following two algorithms will be Ridge and Lasso Regression.

Ridge Regression is another linear model. When building the model, a formula is used called residual sum of squares (RSS) that measures the discrepancy between the data and the estimation model [14].

$$RSS = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

Where $\hat{y}$ is the predicted value and y is the actual value of an entry.

The Ridge Regression model adds another term to this formula:

$$RSS = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 + \lambda \sum_{i=1}^{n} {\beta_j}^2$$

Where β_j are the weights of the given variables in the prediction model.

This second term is called a shrinkage penalty.
The idea of it is to introduce some bias so that the variance can be reduced. As you can recall, the INK algorithm was prone to a cascading mean error increase observed in later iterations. A likely cause of it could be substantial variance in the dataset. The idea behind Ridge and Lasso Regressions is to give a trade-off between bias and variance.

Ridge regression's efficiency increases when many predictor variables are significant and their coefficients are roughly equal [14].

6.1.3 Lasso Regression

Lasso Regression works very similarly to Ridge regression. The main difference lies in the second term:

$$RSS = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 + \lambda \sum |\beta_j|$$

Instead of it being the sum of squares it's the absolute values of the weights. The idea behind it is identical to Ridge regression.

Lasso regression's efficiency increases when only a small number of predictor variables are significant [14].

6.1.4 Support Vector Machine Regressor with polynomial kernel

Support Vector Machine (SVM) regressor is a regression model that provides higher potential accuracy without a significant increase in computation.

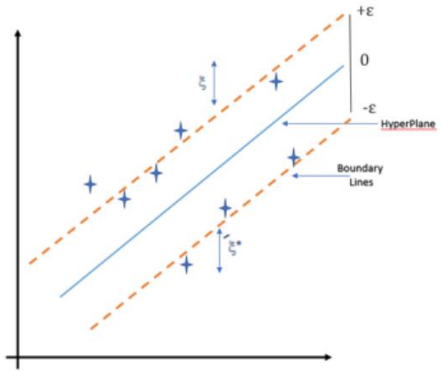

Figure 6: Example visualization of an SVM. Source - shorturl.at/cemMX

There is a single hyperplane that defines the model with two boundary lines for the variables. The model is built using the input. If predictions are not within the boundary space, the model is adjusted accordingly.

For our dataset, we will want to use a kernel to map the data into higher dimensions.

The first kernel we will use is the polynomial kernel.

6.1.5 Support Vector Machine Regressor with Radial Basis Function kernel

The RBF kernel computes similarities (distances) between two points and makes predictions based on how close they are.

In terms of complexity, the RBF kernel is slightly less costly in terms of time and computations and is expected to have slightly higher accuracy on most problems.

6.1.6 Multi-layer Perceptron Regressor with lbfgs solver

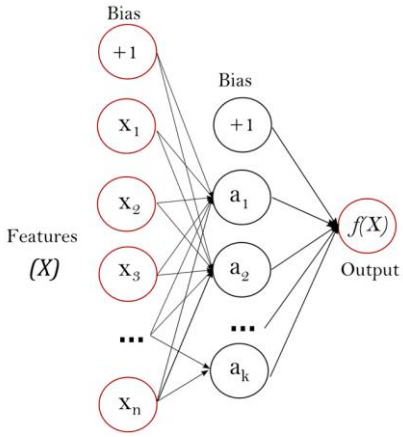

Figure 7: Example visualization of an MLPReg. Source - shorturl.at/bdyEY

An MLP regressor is a supervised learning algorithm that works similarly to an artificial neural network. It has an input layer, a given number of hidden layers, and an output. During building, biases and weights are assigned and re-adjusted to the perceptrons.

The difference between the two MLP algorithms chosen in this study is in the solver – that is the part of the model that is responsible for the optimization of the weights.

The lbfgs solver is an optimizer in the quasi-Newton method family.

6.1.7 Multi-layer Perceptron Regressor with adam solver

The adam solver is a stochastic gradient-based optimizer.

It generally works well on larger datasets, whereas the lbfgs solver is better suited for smaller ones.

6.1.8 Decision Tree Regressor

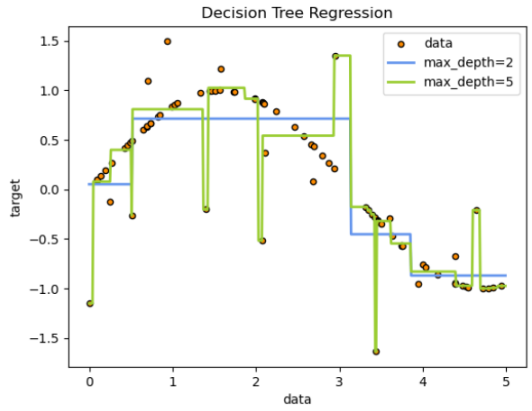

Figure 8: Example visualization of a Decision Tree. Source - shorturl.at/hiBF1

A decision tree is a model that predicts values by constructing a set of rules, which enable data predictions to be inferred from the input.

This can be applied to this regression problem since we have labeled input data.

Decision trees are quite versatile because they require little data preparation (normalization, vectorization) and are easy to visualize. In our case, this means that the model is likely to perform well even if there are issues with the database such as high variance, bias, etc.

They are also time and computation efficient.

6.2 Design Conclusion

These are the selection of algorithms that are going to be compared in this study.
It is important to note that all of them are present in the same library – scikit-learn and that all (except the ones that require a different kernel or solver) are being used with default parameters, without any hyperparameter tuning.

The models will probably show more accurate results if a single algorithm is made to predict the exoplanetary data, with more attention given to model input parameters. However, the hyperparameters set up by default in the library have been selected to work well in the general case. Moreover, different algorithms might have different ranges of improvements depending on the hyperparameter tuning. E.g. if you change the learning rate of the Linear Regression model the mean error might significantly improve compared to one of the SVM models for this particular problem. The issue is this might not be consistent throughout different, but similar datasets, or even a different selection of the same dataset. Finally, should anyone want to recreate the results in this study for themselves, the only requirement is to use the appropriate models with the right solvers and kernels.

For the reasons outlined above, all models use default hyperparameters, except for the SVMs and MLPs, which require a different selection of the “solver” and “kernel” parameters.

7 Results

The version of the algorithm, at the time of writing the study, does not support concurrent processing. Two improvements could be done – one to utilize a GPU in the processing of the predictions and the other is to use concurrent processing to speed up computation.

The algorithms were computed on the Southampton University's Lyceum 5 cluster. Two 2GHz Intel Skylake processors were used with a total of 80 cores of which around 50 were used. Each computation of each machine learning model took between an hour and 4-6 hours to complete depending on the model chosen. This takes into account all of the data computed in this study.

It is possible to predict only a single computation of an algorithm in which case the fastest completion time was about 50 minutes achieved by Linear Regression. Additionally, the fastest time a single feature can be predicted and its accuracy evaluated is in 1-2 minutes depending on the feature. This is once again with the Linear Regression algorithm.

First, we will look at the general results of the algorithms, and then we will examine some of the more interesting features of particular algorithms.

For the sake of simplicity in the visualization graphs, not all features will be included. A total of 20 distinct features will be shown in the group graphs. The features have been put into selections that have a similar value range. For example – positive real features will be shown with other positive real features. Features ranging between 0 and 1 will be shown with other features of the same type.

7.1 Grouped

Here is a graph of some positive real values on a logarithmic scale:

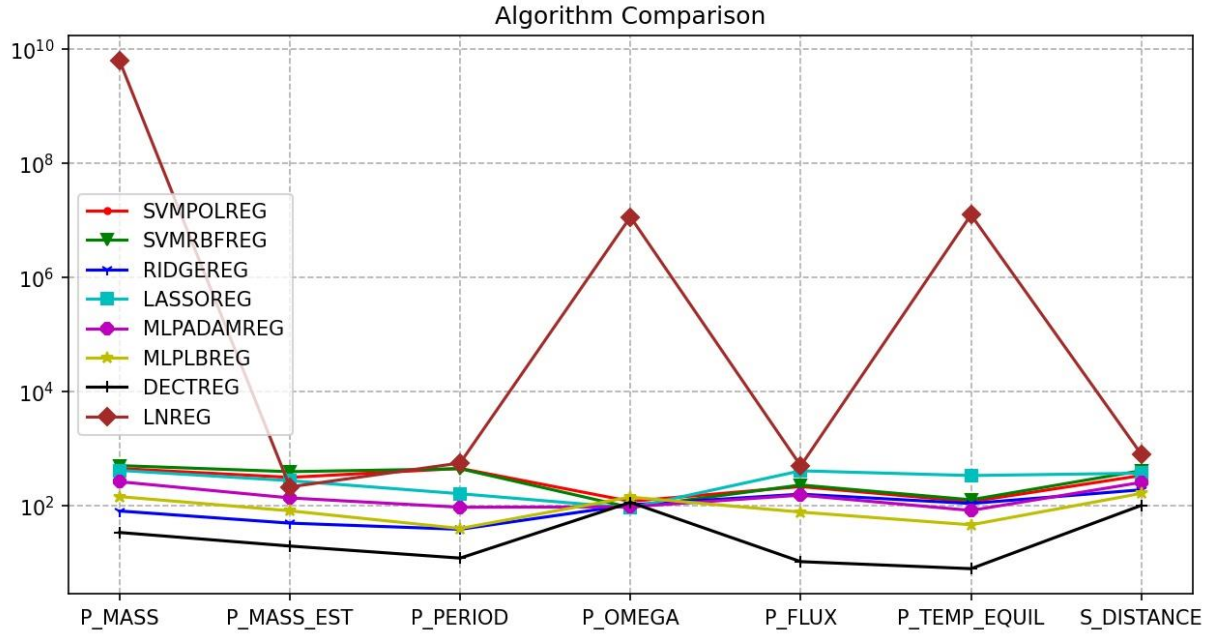

Figure 9: Mean Error of Positive Real Features; Logarithmic

As you can see, there is a significant issue with some of the features in the Linear Regression model (LNREG). This graph was created using the results of a single computation of the INK algorithm. The LNREG values that have not propagated are relatively similar to those of the low accuracy algorithms. Should enough computations of the INK algorithm be computed and only the minimum values of the mean errors taken, LNREG would have a somewhat similar result to the other algorithms, albeit slightly worse.

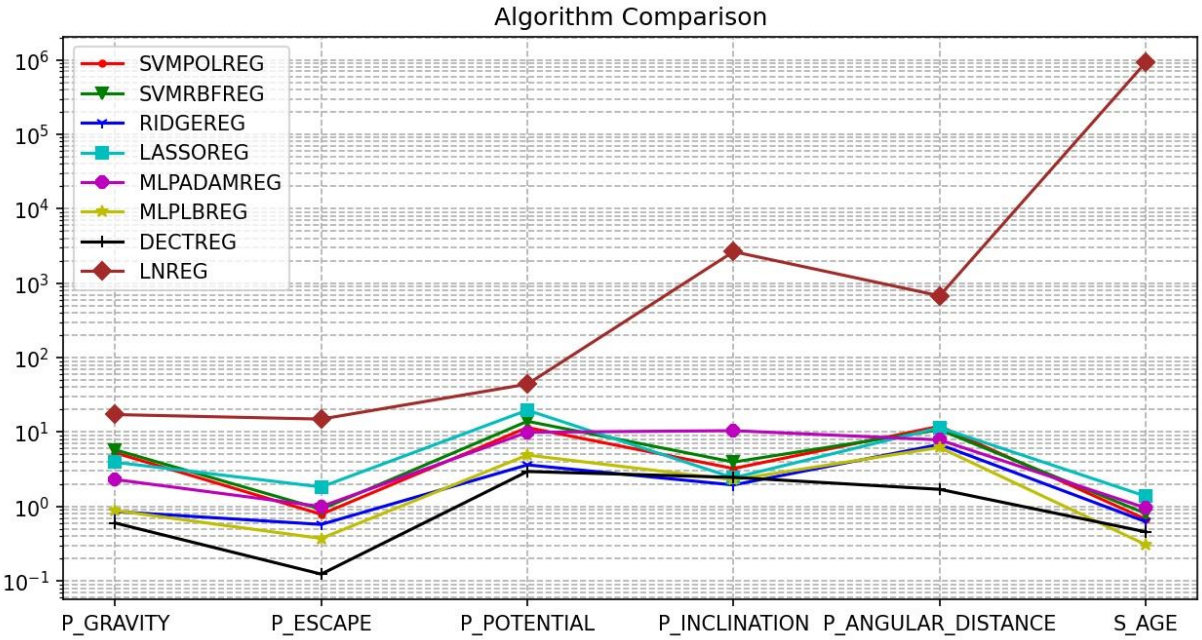

Figure 10: Positive Reals with Low Values; Logarithmic

Generally speaking, for the LNREG model, when the predictions are not erroneous it performs on average the worst out of all the algorithms. This is consistent throughout most features.

We will have a look at the LNREG results in a little bit more in detail later, so in order to get a little bit more visibility on the other algorithms, the LNREG algorithm will be removed from the next few graphs:

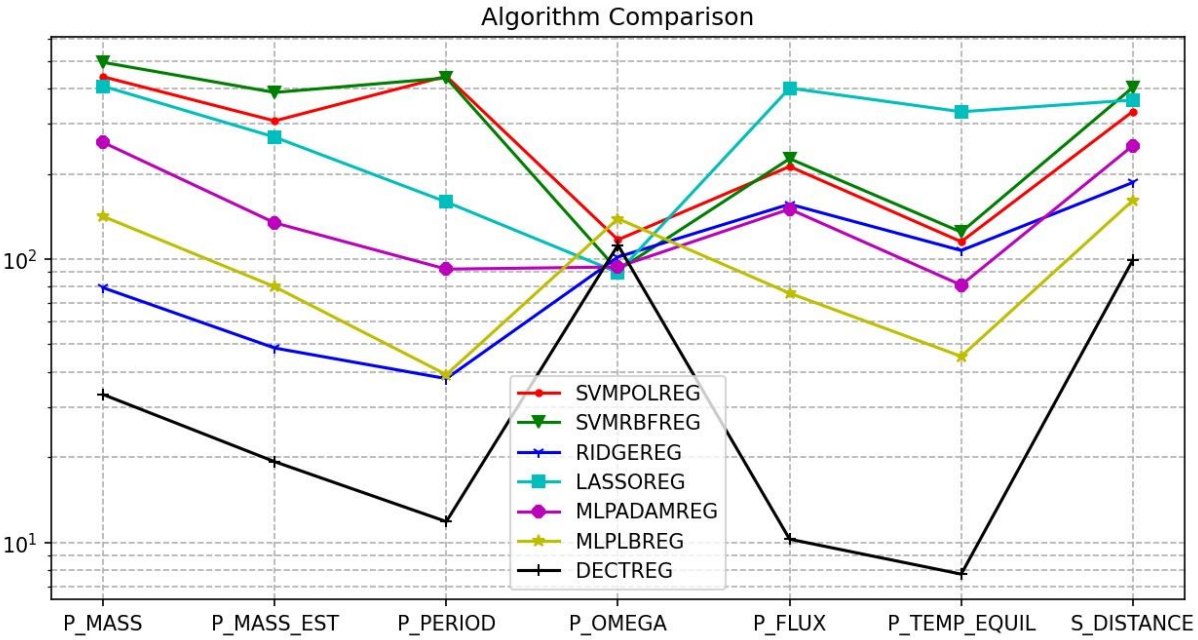

Figure 11: Positive Reals; Logarithmic

This is the same graph as Figure 9, except without the LNREG algorithm.

There are a few trends in the results. The first one is that in most features, the following three algorithms perform the worst – SVM with RBF kernel (SVMRBFREG), SVM with POLY kernel (SVMPOLREG), Lasso Regression (LASSOREG). There doesn't seem to be a particular order in which one performs better than the others, however, the algorithms are almost consistently at the bottom three in accuracy.

Here is the same graph on a linear scale:

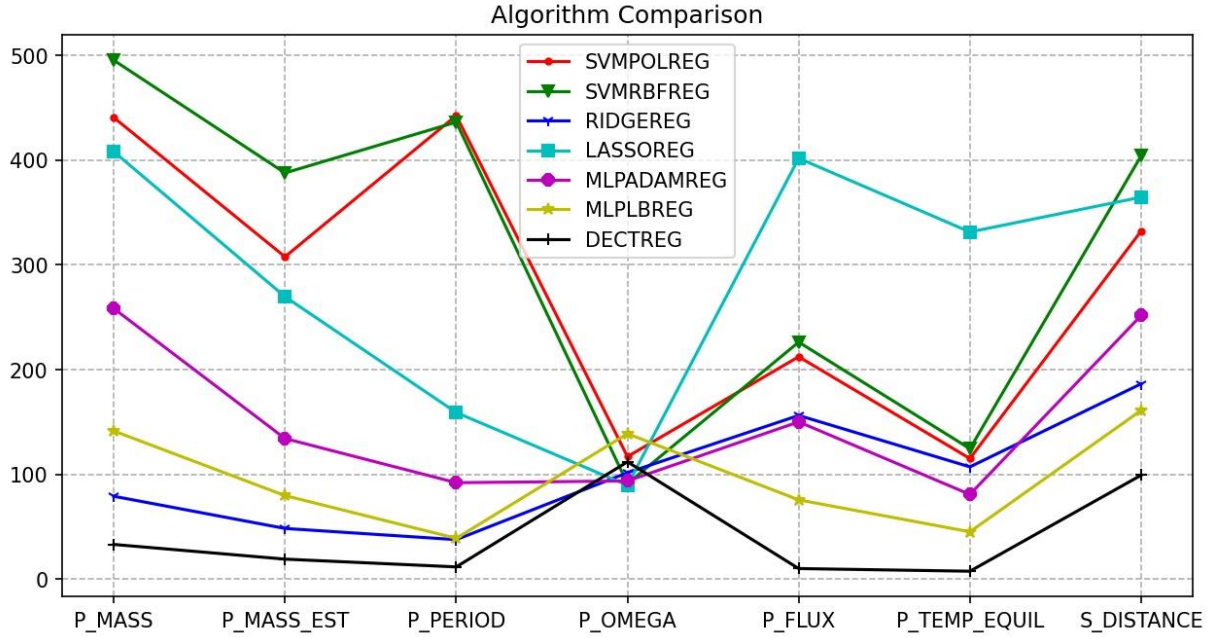

Figure 12: Positive Reals; Linear

The second trend is for the Multi-layer Perceptron algorithm with adam solver (MLPADAMREG) to rank 4th. There are of course exceptions, but generally speaking, that is the case.

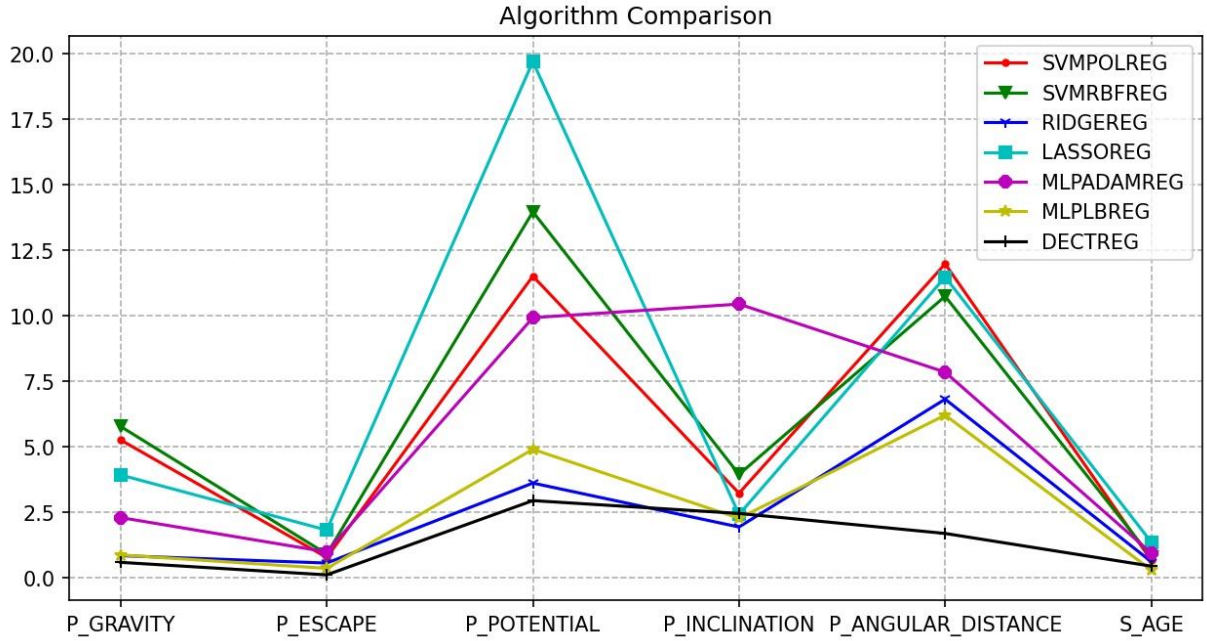

Figure 13: Positive Reals Small; Linear

Finally, Multi-layer perceptron with lbfgs solver (MLPLBREG), Ridge Regression (RIDGEREG), and Decision Tree Regressor (DECTREG) seem to consistently perform in the top three positions. It is difficult to determine which one between MLPLBREG and RIDGEREG performs better as they tend to outperform each other in different features.

The Decision Tree algorithm, however, seems to perform the best on most features.

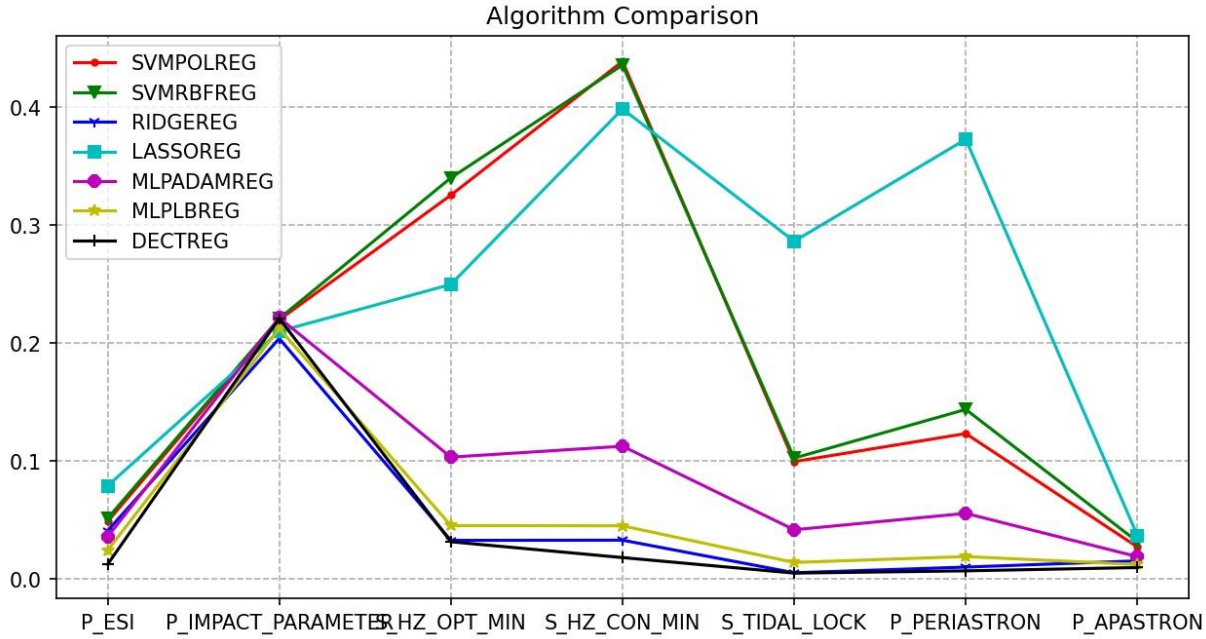

Figure 14:Between 0 and 1; Linear

7.2 Individual

P_MASS is a feature that is quite intuitive to understand and the results are quite easy to visualize the efficacy of the predictive algorithms. For that reason, when comparing each algorithm, a graph will be presented of how well it performs on the P_MASS metric, as well as possibly showcase features that are interesting or provide some different insight.

7.2.1 Linear Regression (LNREG)

With the addition of the modification that removed the 480 highest values in the dataset, the Linear Regression model was the only one that still had some erroneously propagating features. In Figure 9 you can see that the P_MASS feature of LNREG reaches almost 10^{10} mean error. Despite that, in another computation, LNREG manages to converge properly:

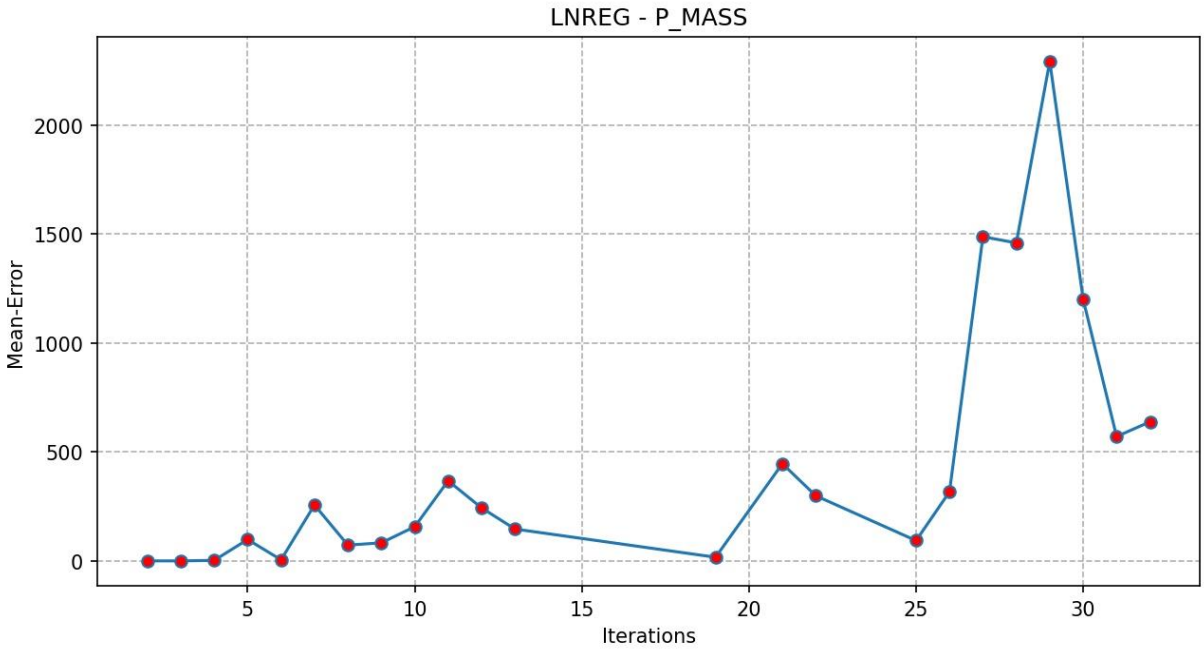

Figure 15: P_MASS, LNREG Mean Error

P_MASS is a measurement that describes the mass of planets in earth masses. The minimum value in the dataset is 0.01907 and the maximum is 17668.059, with a mean value of 785.872168.

The mean error of the graph above is roughly 123, which is one of the features of LNREG that has the highest accuracy compared to other algorithms. In general, though, LNREG lives up to the expectation of a baseline algorithm, given the fact that, as expected, most of the features are not related linearly. As such it is difficult to recommend LNREG as a model that performs well on this particular dataset.

Algorithm	Mean Error
LNREG	123.591342

It is fascinating though to see LNREG perform so well on the first few iterations of P_MASS.

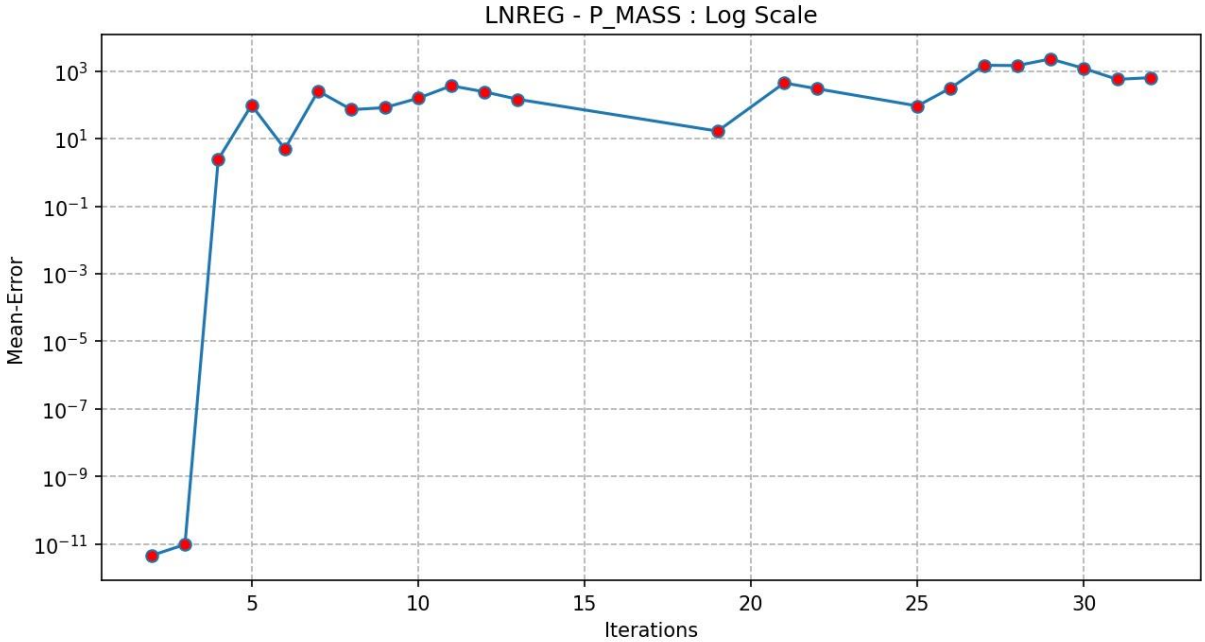

Figure 16: P_MASS LNREG; Logarithmic Mean Error

For reference – P_MASS has a sample size of 143 and 56 on N iterations 2 and 3 respectively (with no occurrences in iteration 1). This means that for those roughly 200 entries, the LNREG model predicts with an accuracy of $1 \cdot 10^{-11}$. It is remarkable to know that such a prediction is possible and it only begs the question – how accurate could our exoplanet predictions become once ML algorithms become more powerful and our observation methods allow us to gather less erroneous data?

The exact values for iterations 2 and 3 are 4.556759010525142e-12 and 9.654324958522205e-12 respectively.

7.2.2 Ridge Regression (RIDGEREG)

It was surprising to see the RIDGEREG model rank this highly, considering its counterpart – LASSOREG ranked in the bottom half.

Recall that ridge regression's efficiency increases when many predictor variables are significant and their coefficients are roughly equal [14].

What we could interpret from this is that the 48 features that have been chosen have a significant influence on one another. We can use human intuition to verify this – e.g., we know that there is a mathematical connection between some features such as mass, density, gravity, etc. Of course, in reality, most if not all the features are connected – we just do not have the means to measure the correlation between some of them.

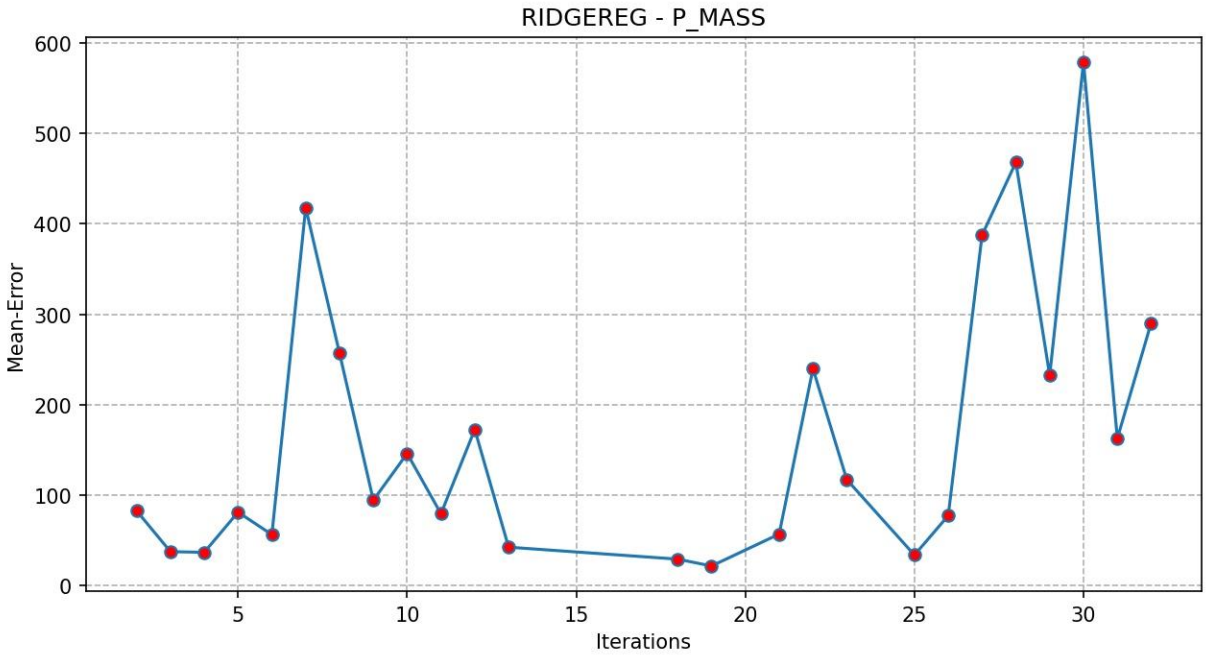

Figure 17:P_MASS RIDGEREG; Linear

By comparing the two graphs, one could notice something peculiar. There is a clear rise in mean error around the 25th to 30th iterations in both graphs. Later on, it will become evident that most graphs have a spike in mean error in that iteration range. Another peculiar spike that is present in several graphs is the one between iterations 5 and 10.

It is difficult to give a conclusive answer as to what causes that. Common intuition could find an answer for the spike at the 25-30 iterations – errors made early in the algorithm lifespan propagate and become larger in later iterations. Expectations should then be for the mean error

to rise linearly or at least polynomially. In reality, the spike looks more like an exponential function that for some reason ceases and returns to regular values. This hypothesis is further voided due to the existence of the spike in earlier iterations – 5 to 10.

Another potential explanation for the two spikes is erroneous information or dominance of outlier entries in those particular iterations.

The mean value of the P_MASS feature of RIDGEREG is around 74.

Algorithm	Mean Error
LNREG	123.591342
RIDGEREG	74.0296781

7.2.3 Lasso Regression (LASSOREG)

The difference between Ridge Regression and Lasso regression is that Ridge has a regularization term that is the sum of the squares of weights, whereas Lasso it's their absolute value. Despite that, the mean error of LASSOREG is on average around 2-3 times larger than that of RIDGEREG.

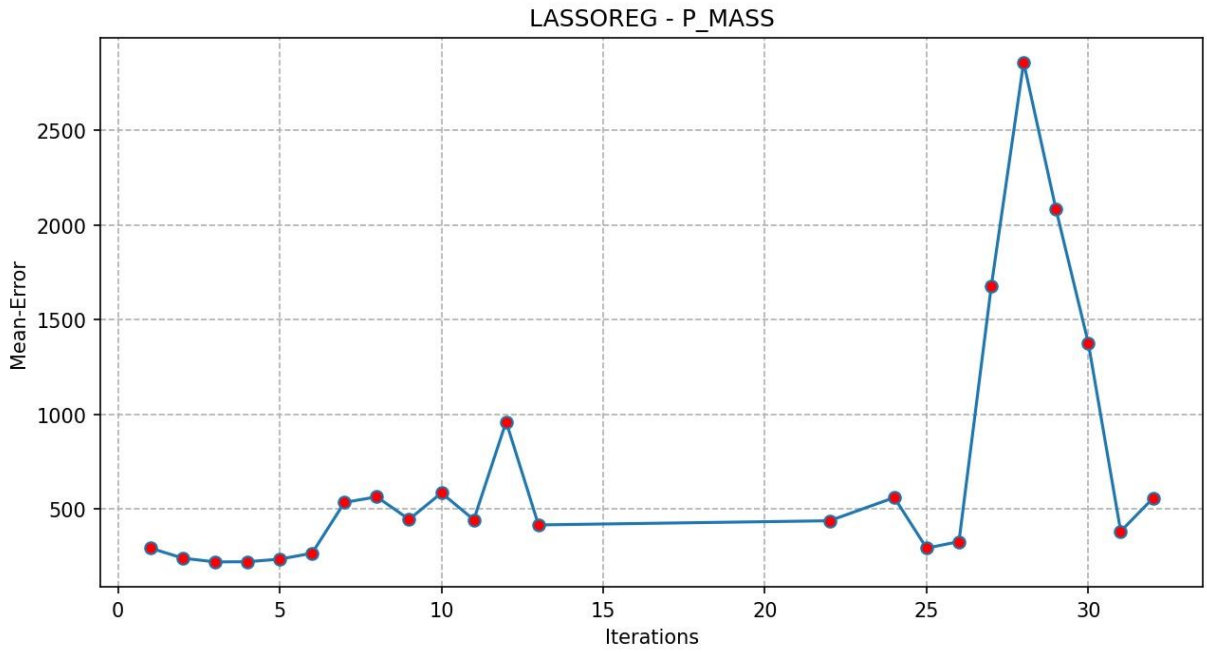

Figure 18: P_MASS LASSOREG; Linear

The mean error of the P_MASS feature for LASSOREG is around 362.

Algorithm	Mean Error
LNREG	123.591342
RIDGEREG	74.0296781
LASSOREG	362.679332

Interestingly enough, LASSOREG is the algorithm that performs the best on the P_OMEGA feature and is among the top 3 in P_INCLINATION.

P_OMEGA is the planet argument of periastron – which in simpler terms is the degree between the middle point of a planet's orbit and its periapsis (The point in the orbit closest to the star). What's important here is the fact that it is a radial measure.
P_INCLINATION is the planet's orbital inclination or – the degree of tilt of the orbit compared to the plane of reference. It is also a radial measurement.

LASSOREG consistently scored well on radial measurements. Moreover, this is one of the few graphs that doesn't follow the trend of a large spike around the 25-30 iterations. Part of this is because radial measurements are limited somewhere between -360 and 360. Still, the mean error in this measurement even has a dip in iterations 25-30.

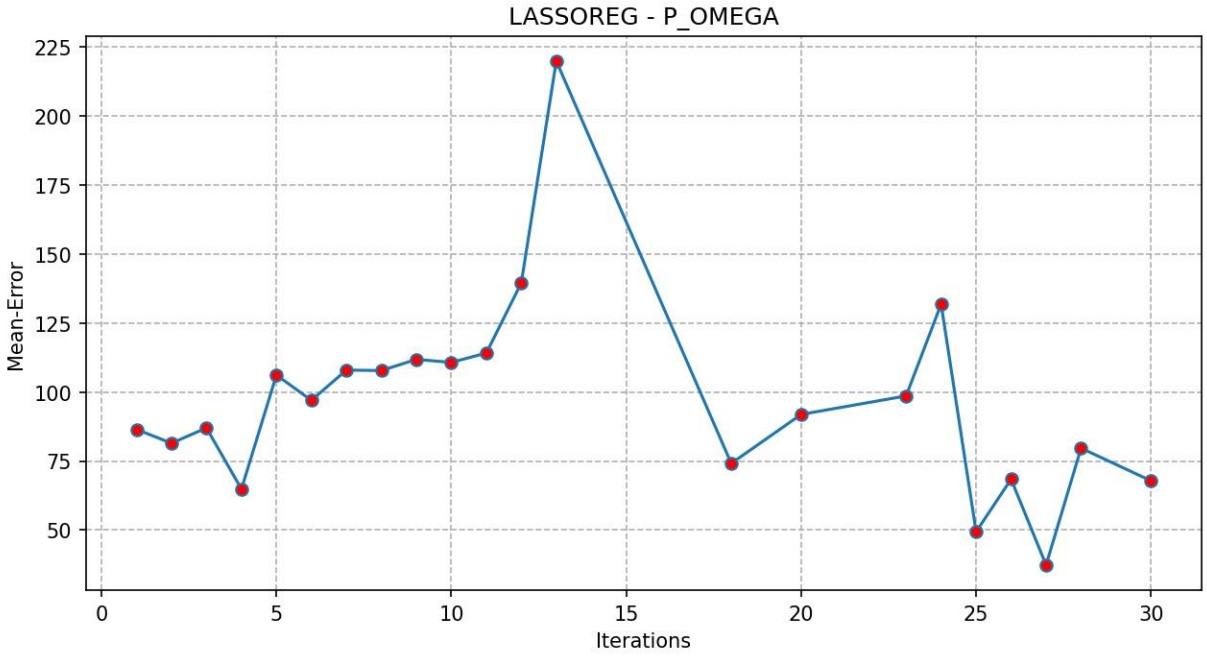

Figure 19: P_OMEGA LASSOREG; Linear

7.2.4 Support Vector Machine Regressor with polynomial kernel (SVMPOLREG)

Surprisingly, the two SVM algorithms came on average with two out of the three worst accuracies (Excluding LNREG).
Given the nature of SVM algorithms and how they work, this could mean that it is difficult to represent the dataset visually in space.

This could have implications for the use of unsupervised methods e.g., clustering, in other areas of exoplanetary or astrophysics-related research. In any case, this is just an observation and not conclusive evidence.

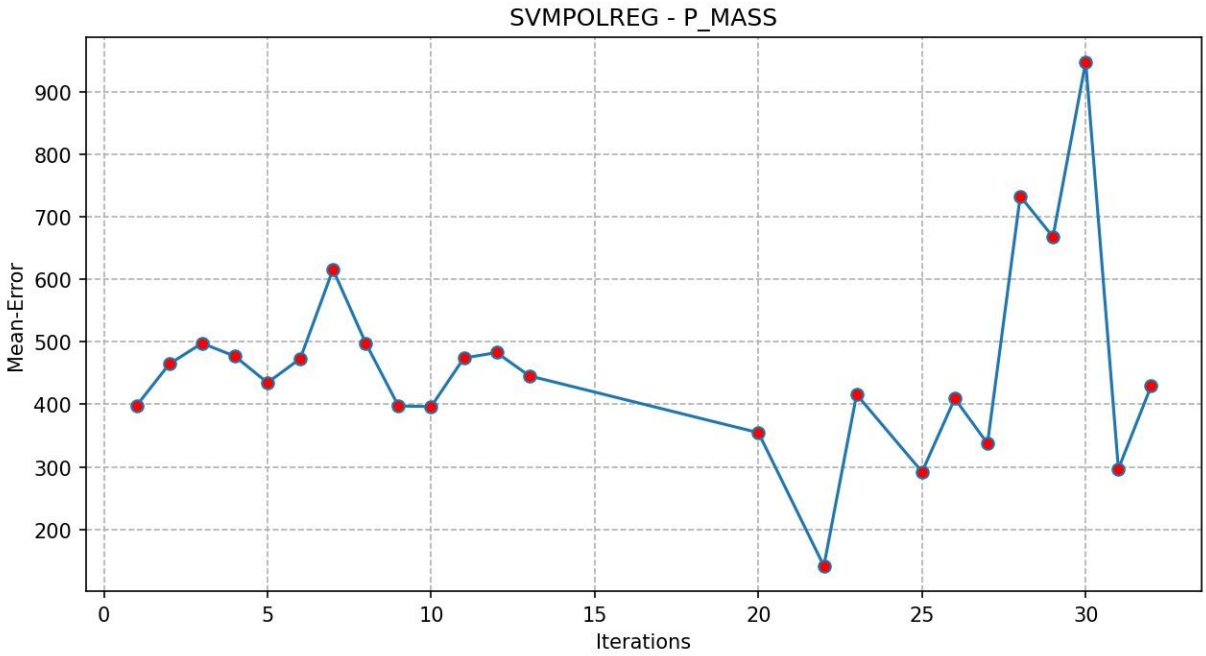

Figure 20: P_MASS SVMPOLREG; Linear

The mean error of the P_MASS feature for SVMPOLREG is roughly 426.

In all features, together with the other SVM model, they are a pair of unremarkable algorithms.

Algorithm	Mean Error
LNREG	123.591342
RIDGEREG	74.0296781
LASSOREG	362.679332
SVMPOLREG	426.733302

The only worthwhile thing to mention is that throughout different computations, unlike most other algorithms, SVMPOLREG and SVMRBFREG maintain relatively consistent accuracy.

LASSOREG	SVMPOLREG	SVMRBFREG
362.6793	443.3907	489.8803
449.4191	463.6604	462.6408
451.4099	454.5648	500.6846
504.8795	426.7333	495.4243
408.1116	440.6645	495.2996

This is a table of the P_MASS accuracies of five different computations of the three algorithms. LASSOREG has been chosen as a comparison due to being the algorithm with the most similar mean error.

As you can see, the two algorithms have a tighter range of predicted mean errors.

Perhaps if one of the requirements of a solution for a problem is for the predictions to be highly consistent, then an SVM-based solution might make a better fit than one of the others. The downside of course is a lower accuracy.

7.2.5 Support Vector Machine Regressor with Radial Basis Function (SVMRBFREG)

Similar to the polynomial kernel SVM, this one is not particularly interesting apart from the point made above.

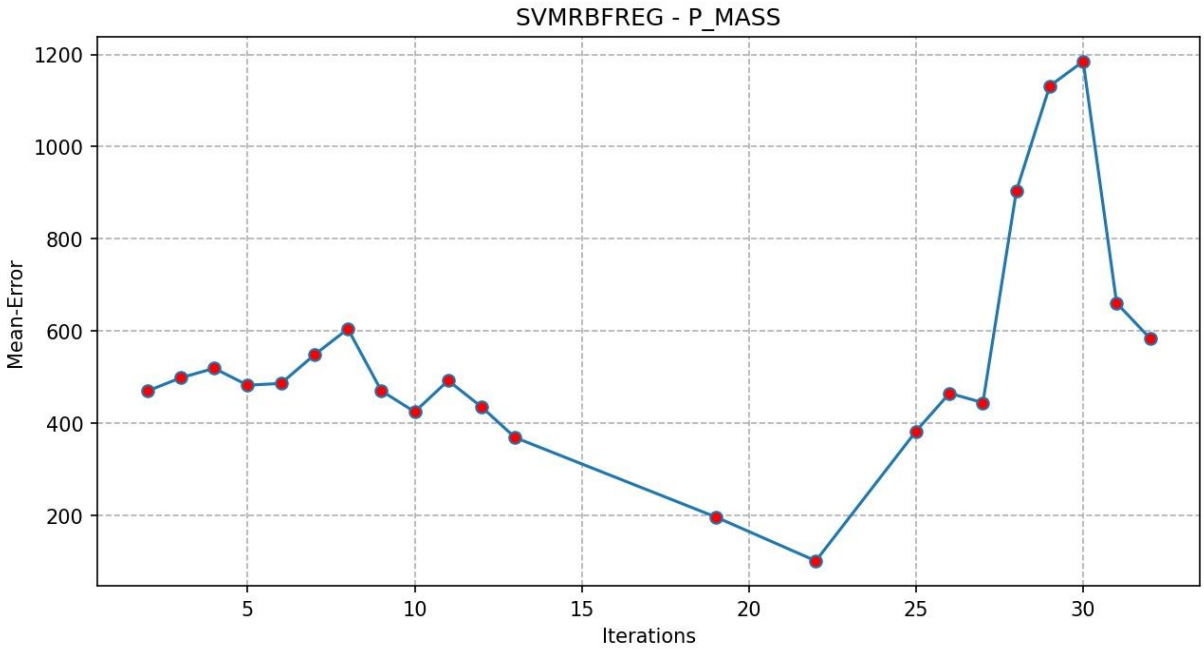

Figure 21: P_MASS SVMRBFREG; Linear

The mean error of the P_MASS feature for SVMRBFREG is roughly 462. This is the highest mean error for this particular feature.

Interestingly, the above figure is one of the few where the algorithm makes significantly better predictions for a while after half of the iterations (Iter. 19, 22, and 25).

Algorithm	Mean Error
LNREG	123.591342
RIDGEREG	74.0296781
LASSOREG	362.679332
SVMPOLREG	426.733302
SVMRBFREG	462.640793

7.2.6 Multi-layer Perceptron with lbfgs solver (MLPLBREG)

It doesn't come off as a surprise that the two regression algorithms based on a neural network approach are ranked in the top half of the models. Neural networks are considered to be one of the most potent types of machine learning algorithms currently. A big reason for that is their capability to find relations between inputs that are too difficult for the human eye to notice.

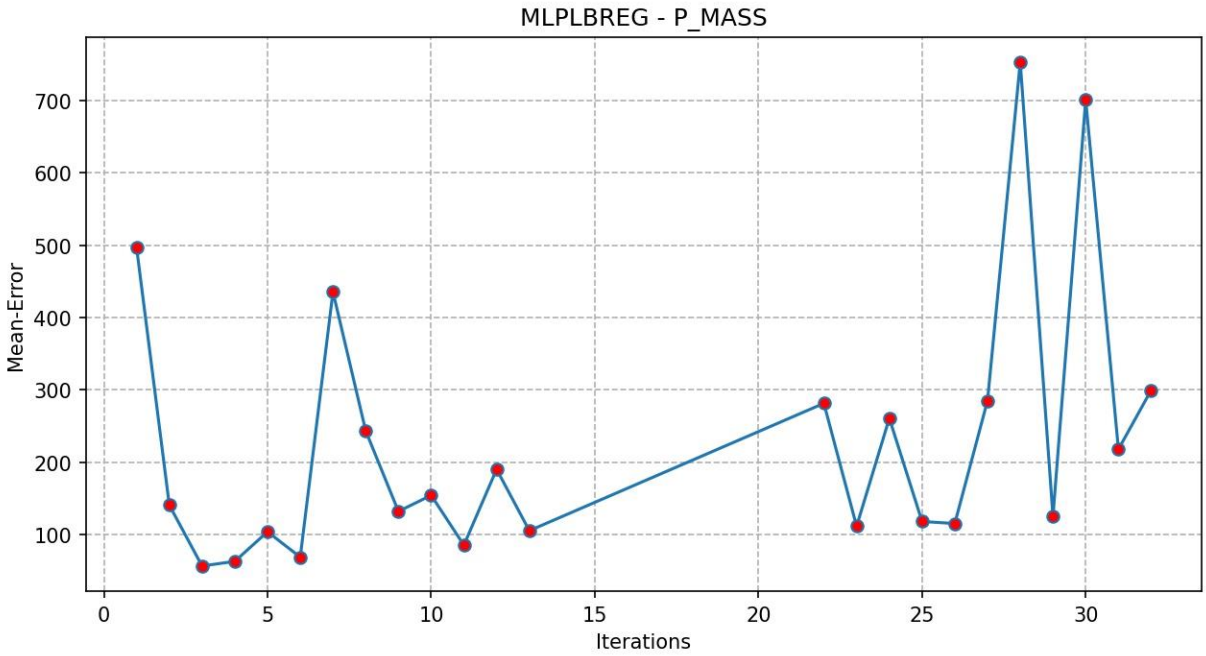

Figure 22: P_MASS MLPLBREG; Linear

At first glance, it is difficult to see how this algorithm on this particular computation could be within the ones with the highest accuracies. The mean errors have a peculiar jagged trend throughout the iterations. Finding a plausible explanation for this jaggedness proves difficult, especially for the large dip in the 29th iteration, which common interpolation would place somewhere around 700.

It is important to remember that the mean error is not calculated simply by summing the values and dividing over the iterations. Each iteration has a different number of entries that require to be predicted and this algorithm performs well on those iterations where a lot of predictions need to be made.

The mean error of the P_MASS feature for MLPLBREG is roughly 109.

Algorithm	Mean Error
LNREG	123.591342
RIDGEREG	74.0296781
LASSOREG	362.679332
SVMPOLREG	426.733302
SVMRBFREG	462.640793
MLPLBREG	109.906773

The Multi-layer perceptron algorithms perform exceptionally well in features that range between 0 and 1.

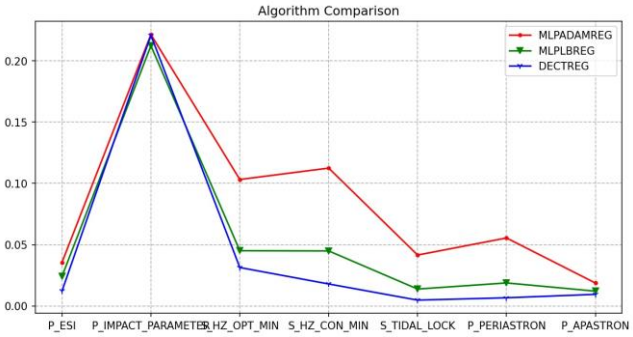

Figure 23:0 to 1 valued comparison of the three highest-ranked algorithms

7.2.7 Multi-layer Perceptron with adam solver (MLPADAMREG)

The two MLP algorithms are pretty similar to one another in results. They follow the same trends in graphs and perform relatively similarly in different types of features. Except for a few features, the lbfgs solver performs slightly better than adam.

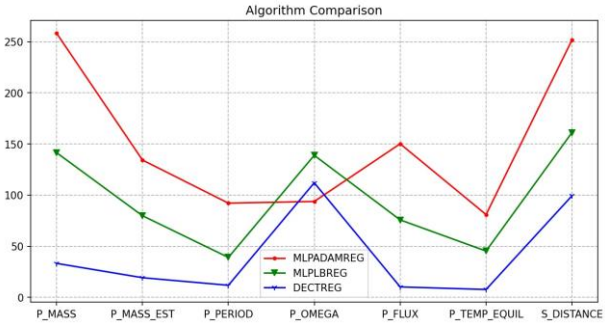

Figure 24: Top 3 comparison for large positive reals

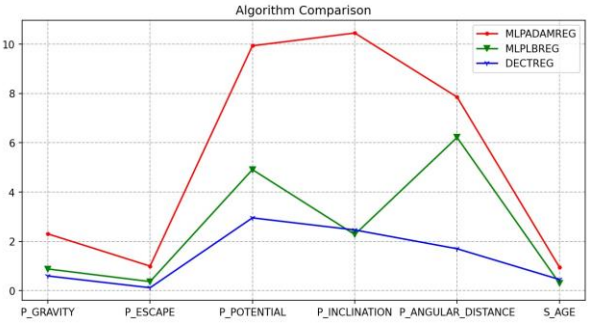

Figure 25: Top 3 comparison for small positive reals

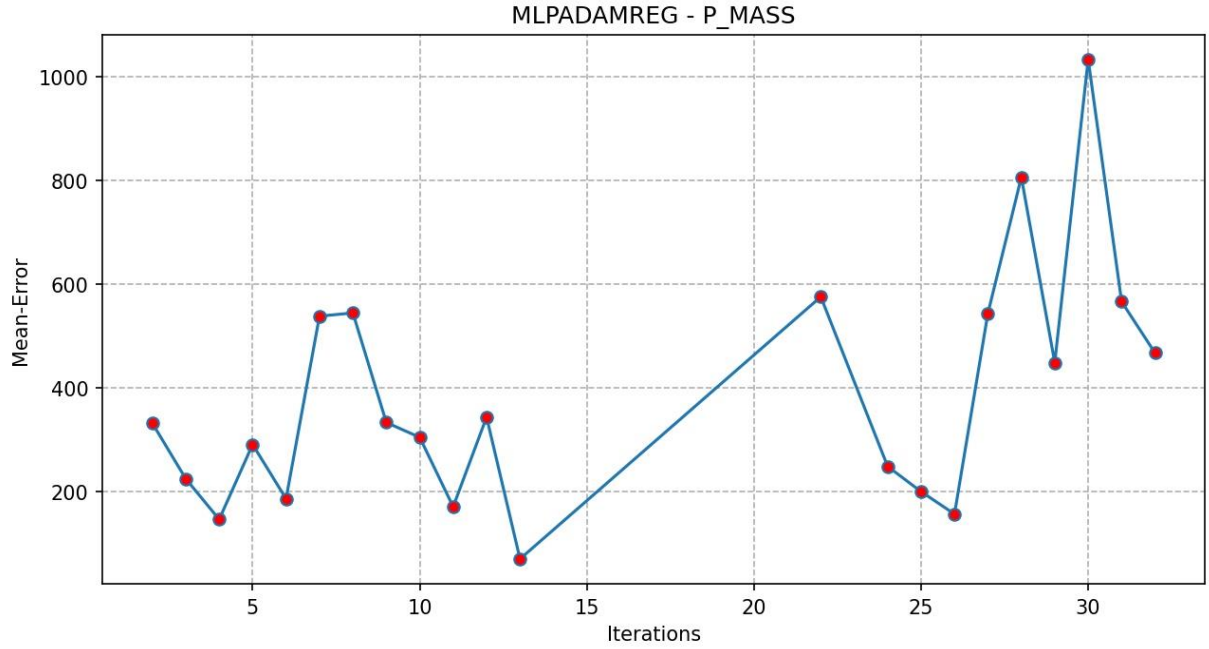

Figure 26: P_MASS MLPADAMREG; Linear

The mean error of the P_MASS feature for MLPADAMREG is roughly 235.

Algorithm	Mean Error
LNREG	123.591342
RIDGEREG	74.0296781
LASSOREG	362.679332
SVMPOLREG	426.733302
SVMRBFREG	462.640793
MLPLBREG	109.906773
MLPADAMREG	235.839522

7.2.8 Decision Tree Regressor (DECTREG)

Finally, the algorithm with, without a doubt, the best results in this particular database is the Decision Tree Regressor. The initial expectations for this algorithm were that it would

compute quickly, but would not be as accurate as others. The DECTREG model is capable of making accurate predictions in datasets with high bias and variance.

As we noticed in the analysis of the dataset, a high bias and variance is something that very well could be the reason why this particular algorithm performed so well, or rather, why the other algorithms performed worse than this one. Either way, as we will see, this algorithm makes predictions with impressive accuracy.

Despite that, it still struggles from the same issue observed in some of the other graphs – two spikes around the 5 to 10 iterations and 25 to 30 iterations.

Should the issue that causes that be discovered – whether in the INK algorithm or in the dataset itself, the DECTREG algorithm would be fully capable of making predictions in the single digits on features with a mean in the hundreds and a range in the thousands.

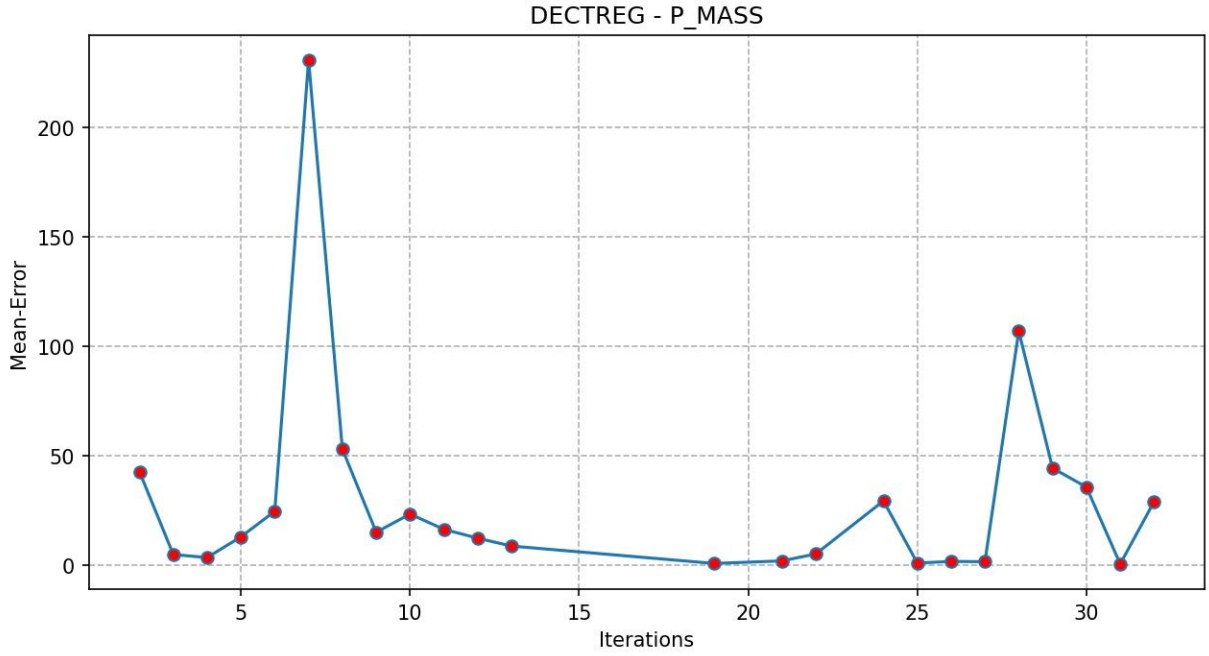

Figure 27: P_MASS DECTREG; Linear

The mean error of the P_MASS feature for DECTREG is roughly 26.

Here are the numerical values for the later iterations:

Iteration	Mean Error
24	29.52643
25	1.150184
26	1.88961
27	1.758388
28	107.0349
29	44.49589
30	35.85098
31	0.709551
32	29.14377

Table 6: P_MASS Mean Error of DECTREG

While this doesn't even come close to the mean error in the first few iterations of the LNREG, those were made at the beginning of the algorithm, whereas these predictions are at the latest iterations.

Algorithm	Mean Error
LNREG	123.591342
RIDGEREG	74.0296781
LASSOREG	362.679332
SVMPOLREG	426.733302
SVMRBFREG	462.640793
MLPLBREG	109.906773
MLPADAMREG	235.839522
DECTREG	26.9942755

Table 5: P_MASS Mean Errors of each algorithm

Finally, there is one feature, which, while it is not very informative for making a good comparison between different algorithms is perhaps the primary reason why this whole dataset exists and why research in this field is significant.

That feature is P_ESI – or Earth Similarity Index. The main reason behind mapping the planets, measuring their properties, and in this case predicting them is that we want to find planets similar to Earth. It is difficult to say whether or not a planet is similar to ours. For starters, many of the planets we presumed to have been similar ended up having different atmospheres, different elements in the crust, and different planet types, which in general made life as we know it impossible on those planets.

Several papers have suggested metrics by which we could direct our attention and resources towards planets that show similar properties as Earth. The result of one of these papers, as we mentioned earlier, has been the Earth Similarity Index. Being able to make accurate predictions on this particular metric could guide physicists on which planets to spend more resources analyzing.
The Decision Tree Regressor has the highest performance on the P_ESI metric.

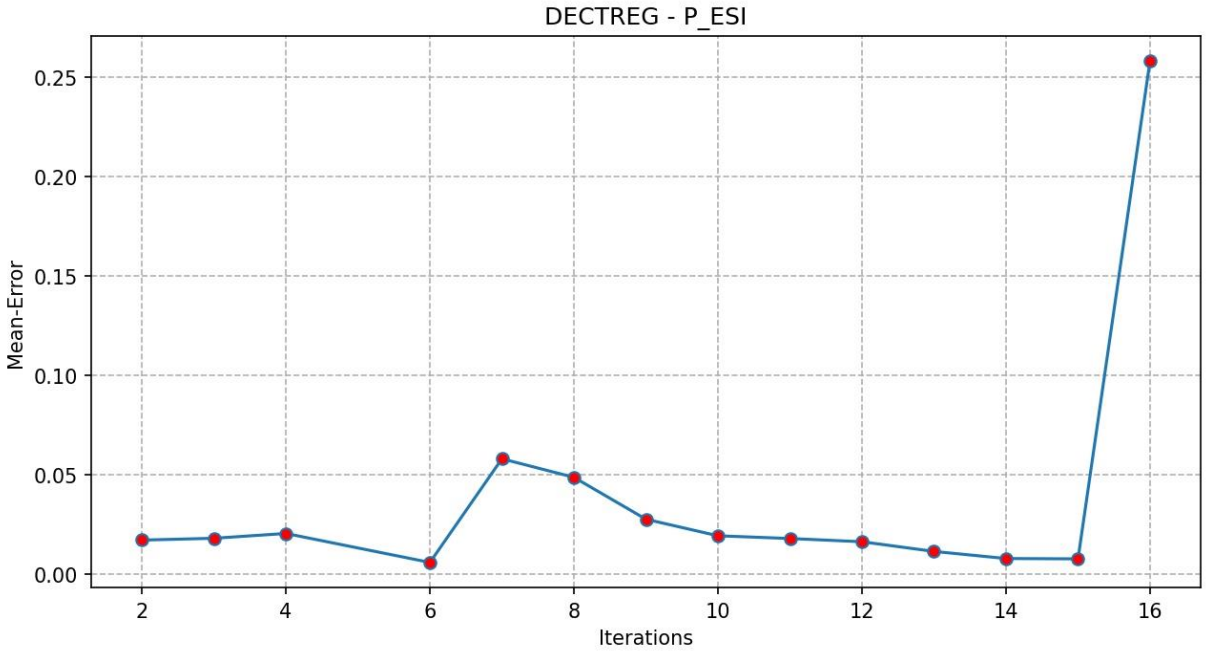

Figure 28:P_ESI DECTREG; Linear

Table 7: 15 highest values in the P_ESI feature

P_ESI
0.93120767
0.90286374
0.89839814
0.88802761
0.87158977
0.86977617
0.86745445
0.84642767
0.84342178
0.82767746
0.82348299
0.82325475
0.80561043
0.8046835
0.79777954

The P_ESI is a value between 0 and 1. The mean value of the feature in the original dataset is 0.260947 with a minimum value of 0.006768 and a maximum value of 0.931208. 75% of this data is between the minimum value and 0.303182.

These are the highest 15 values of the P_ESI feature.
After the highest 100 values, the P_ESI drops to below 0.6.

In other words, as we can guess, planets that are similar to Earth are quite rare. Even being able to predict an ESI with an error of around 0.05 is enough to give us a reasonable understanding of which planets might be potentially habitable, so we can focus on researching them.

Suppose that the ESI threshold of a planet that is potentially habitable is around 0.75. That means we've reduced the possibilities of habitable planets in this dataset to roughly 50 out of 4048.

This study can be used not only to gather insights on the efficacy of different predictive algorithms but also as a proof of concept that some ML algorithms could assist us in making educated guesses about which planets are worthwhile spending time on.

8 Critical Evaluation

8.1 Design Choices

The main design issues in this research are related to the dataset and how it is used. There is no doubt that the data in question is not perfect – there are many errors of different origins. A problem of significant importance is coming up with a solution that identifies and fixes erroneous information that is preferably automatic. While the dataset supplied is particularly small – 4048 entries, the information that we will have at our hands will increase exponentially in the following few years.

As mentioned in the introduction of this paper, more and more powerful telescopes will be put into space. This means that we need to be prepared for a larger quantity of data to be processed. While manual processing such as that done in this study is highly effective in finding and removing erroneous data, more automated techniques of data cleanup need to be discovered in future exoplanetary prediction algorithms.

The second significant issue is with the expunging of the 480 data entries. While extreme outliers should be considered erroneous, having planets with some eccentric features is plausible. Serious errors might occur when there are mechanical malfunctions or some type of outside influence – e.g., a black hole near a planet that cannot be detected by the transit method. In any case, more powerful predictive algorithms should be able to account for this bias and variance and if they can't more thoughtful solutions need to be presented for the issue above.

8.2 Implementation

Ignoring the anomalous spikes in the result graphs, there was a general trend for the mean errors to increase with iterations. This is only natural, due to small errors in previous predictions being used to make ever so slightly larger errors in latter ones.

An ideal solution for this particular dataset would only require one iteration to complete to reduce or remove this propagation effect. Performance could improve when more powerful models are used for the predictions. Approaches using unsupervised learning could also be of use due to them being good at finding correlations where humans can't find any. Of course, such models would come at a cost of much higher complexity to implement, as well as higher computational power requirements and time to compute. Not to mention the need to validate the data as well.

Due to poor scalability and the need for refactoring of the INK algorithm, it is not possible at the time of writing this report to automatically collect error data on the categorical data such as P_TYPE and S_TYPE_TEMP. While the algorithm predicts those features and outputs them to the full predicted dataset, the only way to measure their accuracy is to manually expunge data from the original algorithm and then manually compare the expunged data with the predictions.

8.3 Potential Improvements

The current implementation that solves the outliers problem by expunging the ten highest entries is seriously flawed.

A significant improvement to this brute-force-like approach would be instead of removing the entries completely to replace the outlier information with NaNs.

An even more sensible approach would be to have some sort of automatic outlier detection algorithm. Earlier in the study, it was mentioned how the data visualized had a lot of characteristics of a normal distribution. Hence it was concluded that some of these outliers were plausible. That being said, an algorithm could be devised that evaluates the data of a feature, and based on distribution makes educated guesses on whether or not data is erroneous or an outlier. E.g., data that is beyond some number of standard deviations could be deemed erroneous. The difficulty in this approach comes from its complexity and the fact that different features would have different requirements for the number of deviations. As such, an approach like that one would require extensive research, but would certainly yield better results.

Secondly, a type of genetic algorithm approach was proposed as an alternative to the failed idea of "mid-processing". A complex model using such an approach might find increased accuracy throughout the iterations. Moreover, in the final results, a small number of entries were observed to have erroneous predictions. This issue was more significant in earlier versions of the INK model, where predictions weren't accurate, but is still present even in the latest version.
Using that same genetic approach, such erroneous predictions can be filtered out and the resulting dataset can be run through the algorithm again, predicting the removed data. This would be a recommended approach for cases in which high accuracy of the predicted data is required as it would come with a time and computational complexity tradeoff.

9 Conclusion

As we can see, the data that we have available to us could use improvement. This could be done through algorithms that specialized algorithms that detect erroneous information and either remove it or find a way to restore it. Regardless, even without high-quality pre-processing, we can make programs that are capable of making accurate predictions on the exoplanetary datasets.

From the algorithms analyzed in the study, there is one that consistently performs well – the Decision Tree model. Whether its success owes to the fact it is less influenced by erroneous information and data with high bias and variance is difficult to say. It does however perform quite well on the PHL-EC dataset. Some algorithms have better performance on particular features than they have on others. For example, Lasso Regression seems to have higher accuracy on radial features, while the Multi-layer Perceptron Regressor has consistently high results on features valued between 0 and 1. This is not conclusive evidence, though, and more research should be conducted.

Finally, we have seen that the algorithms listed above are capable of making highly-accurate predictions on the habitability of planets without using complex mathematical formulas. According to some physicists, ESI values of less than 0.75 for most planets means that they are not habitable. Our current ML tools are capable enough to give us an educated guess into which planets are worth spending more time analyzing as potential habitable planet candidates.

10 Bibliography

[1] W. J. Borucki, D. Koch, G. Basri and e. al., "Kepler Planet-Detection Mission: Introduction and First Results," *Science Express,* p. 4, 2010.

[2] J. N. W. R. V. D. W. L. G. Á. B. J. L. B. Z. K. B.-T. T. M. B. L. B. N. R. B. R. P. B. W. J. C. D. B. C. J. C. George R. Ricker, "Transiting Exoplanet Survey Satellite," *Journal of Astronomical Telescopes, Instruments, and Systems,* 2014.

[3] S. B. G. U. D. M. S. Madhu Kashyap Jagadeesh, "Indexing of exoplanets in search for potential habitability: Application to Mars-like worlds," *Springer Science Business Media,* vol. 1, no. 1, p. 13, 2017.

[4] S. R. V. V. K. K. K. a. Ł. K. Madhu Kashyap Jagadeesh, "Indexing Exoplanets with Physical Conditions Potentially Suitable for Rock-Dependent Extremophiles," *life,* p. 8, 2019.

[5] M. R. ,. K. Madhu Kashyap Jagadeesh, "Tardigrade Indexing approach on exoplanets," *Life Sciences in Space Research,* vol. 1, no. 1, p. 16, 2018.

[6] S. S. A. M. K. B. S. M. M. S. S. A. Suryoday Basak, "CEESA Meets Machine Learning: A Constant Elasticity Earth Similarity Approach to Habitability and Classification of Exoplanets," Indian Institute of Astrophysics, Bangalore, 2019.

[7] V. S. R. R. Kavitha S, "A Comparative Analysis on Linear Regression and Support Vector Regression," International, 2016.

[8] A. S. L. J. G. Jozef Zurada, "A Comparison of Regression and Artificial Intelligence Methods in a Mass Appraisal Context," *J R E R,* vol. 33, no. 3, p. 40, 2011.

[9] Z.-K. F. B.-F. F. Y.-W. M. C.-T. C.-Z. Z. Wen-Jing Niu, "Comparison of Multiple Linear Regression, Artificial Neural Network, Extreme Learning Machine, and Support Vector Machine in Deriving Operation Rule of Hydropower Reservoir," *Water,* vol. 1, no. 11, p. 5, 2018.

[10] R. N. H. S. S. Somayeh Golbaz, "Comparative study of predicting hospital solid waste generation using multiple linear regression and artificial intelligence," *Journal of Environmental Health Science and Engineering,* vol. 1, no. 1, p. 11, 2019.

[11] K.-H. K. J.-J. K. S.-J. C. Y.-S. I. Y.-D. L. Y.-S. L. Ki-Young Lee, "Comparison and Analysis of Linear Regression & Artificial Neural Network," *International Journal of Applied Engineering Research,* vol. 12, no. 20, p. 6, 2017.

[12] D. P. a. K. R. Anil Jadhav, "Comparison of Performance of Data Imputation Methods for Numeric Dataset," Symbiosis Centre for Information Technology, Pune, 2019.

[13] G. P. Christos Platias, "A Comparison of Machine Learning Methods for Data Imputation," Institute of Informatics & Telecommunications National Centre for Scientific Research (N.C.S.R.) "Demokritos", Attiki.

[14] S. S. L.E. Melkumovaa, "Comparing Ridge and LASSO estimators for data analysis," in *3rd International Conference "Information Technology and Nanotechnology, ITNT-2017,* Samara, 2017.