ML@NOVA: [DDF]

The three-body problem

Team identification

Name 1: Diogo Almeida

Number 1:70140

Name 2: Filipe Colla David

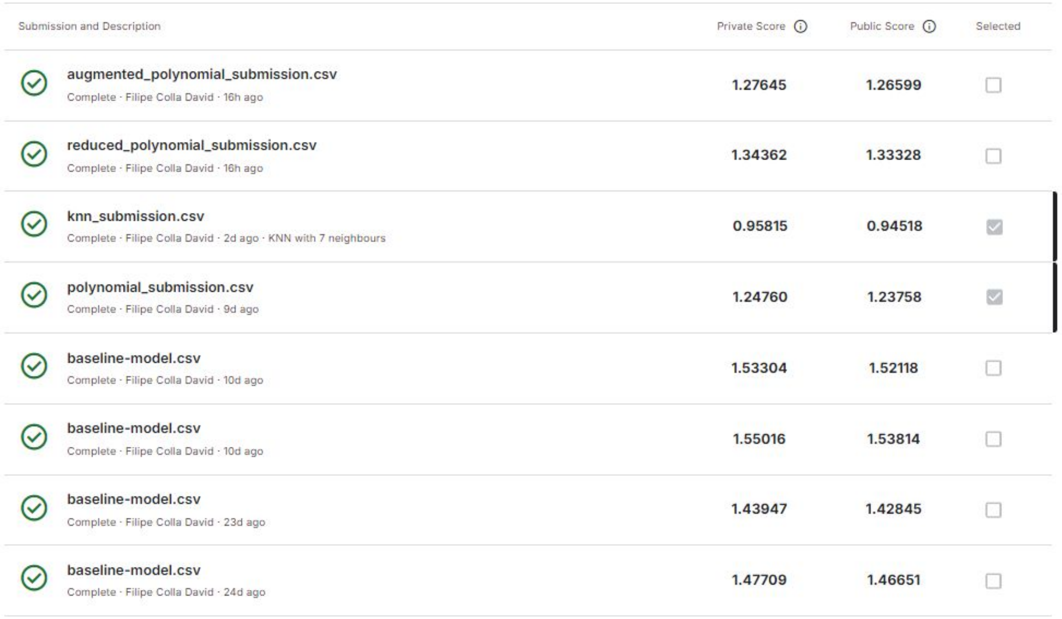
Number 2: 70666

Name 3: Duarte Rodrigues

Number 3:70150

Final score: 0.94518

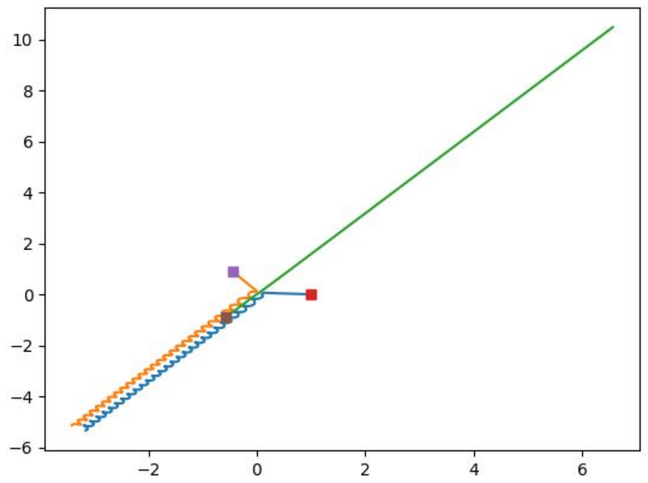
Leaderboard Private Ranking:



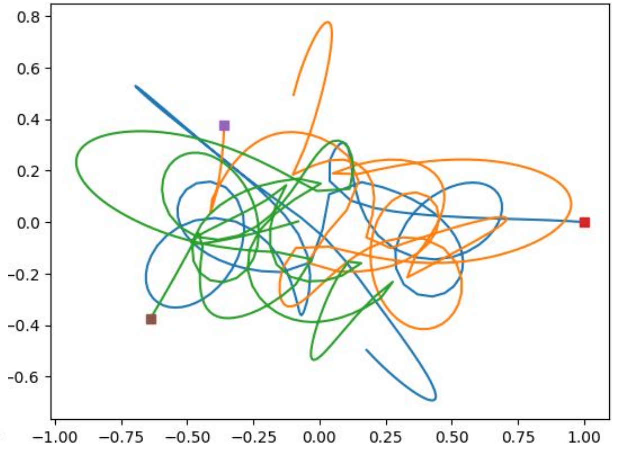
Submission and Description	Private Score	Public Score	Selected
✓ augmented_polynomial_submission.csv Complete · Filipe Colla David · 16h ago	1.27645	1.26599	☐
✓ reduced_polynomial_submission.csv Complete · Filipe Colla David · 16h ago	1.34362	1.33328	☐
✓ knn_submission.csv Complete · Filipe Colla David · 2d ago · KNN with 7 neighbours	0.95815	0.94518	✓
✓ polynomial_submission.csv Complete · Filipe Colla David · 9d ago	1.24760	1.23758	✓
✓ baseline-model.csv Complete · Filipe Colla David · 10d ago	1.53304	1.52118	☐
✓ baseline-model.csv Complete · Filipe Colla David · 10d ago	1.55016	1.53814	☐
✓ baseline-model.csv Complete · Filipe Colla David · 23d ago	1.43947	1.42845	☐
✓ baseline-model.csv Complete · Filipe Colla David · 24d ago	1.47709	1.46651	☐

Data Visualization

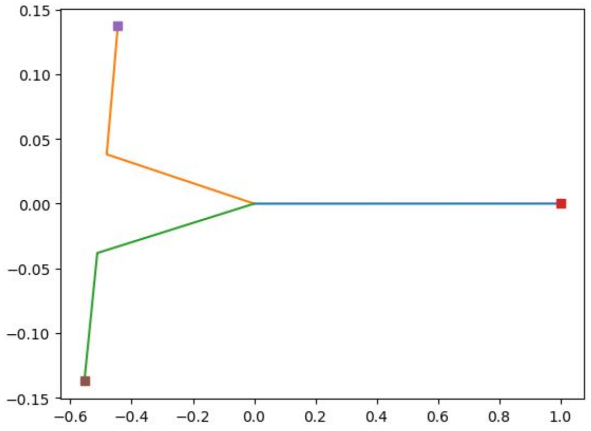
Types of Trajectories



Stable (one of the bodies left)



Chaotic



Collisions

Data Preparation

Preparing the data

- Created a new column with unique IDs for each trajectory. This is useful for later, when we want to split the dataset into train, test and validation sets.
- For each trajectory where a collision occurred, removed all the data points after the instant where they collided. These trajectories are represented in the dataset by having all the columns after the collision, except ‘Id’ and ‘trajectory_id’ equal to 0.

Creating the Feature and Target Datasets.

- Throughout the code, the *feature* and *target* datasets are represented by X_ and y_, respectively.
- The *features* dataset, for each individual trajectory, contains just the initial positions. Meaning that all the rows for the columns except ‘t’, ‘trajectory_id’ and ‘Id’ will be replaced by the initial position for that trajectory. The t=0 is also removed, since it will also be present in the target dataset.
- The *target* dataset will only have the features that we want to predict with our model, so it will have the positions (x,y,z) and for convenience, the the ‘Id’ and ‘trajectory_id’ column.
- These convenience columns are removed for the training of the models.

Data Split

- The data split function, *custom_train_test_split*, will split the dataset with the selected split size (20% for testing by default) using trajectories_id.
- This was achieved by splitting the trajectories_id into two sets (train and test), and then using the resulted columns to filter the X and y datasets with the corresponding trajectories_id
- This function has a flag to drop the trajectory_id and Id columns if needed while also allowing for additional columns to be dropped
- Before any data models were made, a Test, Train split was done. This created a test dataset of 10% that will only be used to ensure the performance of the model. The other 90% will be divided into train and validation for checking each individual model.

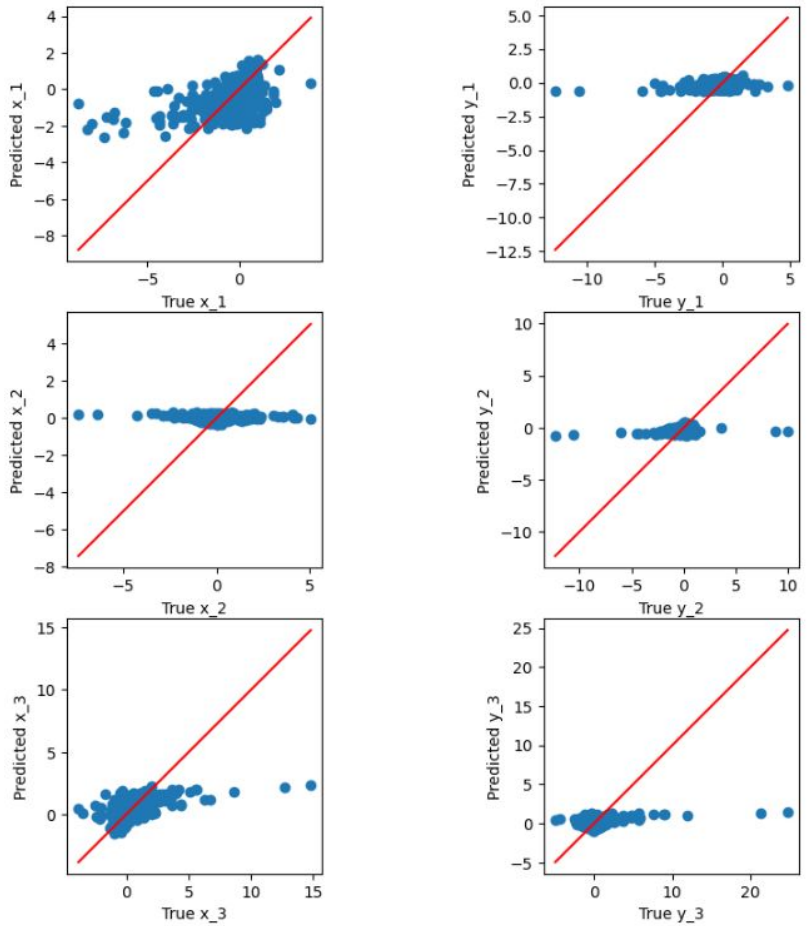
Linear Regression

What was done in Linear Regression

- The learning problem for this Linear Regression consists in predicting a position at a given t from the initial position.
- Taking the 90% of data remaining from the initial split, we created the train datasets (X_train, y_train) and the validation datasets (X_val, y_val)
- Removed the speeds, because we are not trying to predict the speeds in the target variable.
- Used a StandardScaler() to scale the features into unit variance.
- Trained the model with the X_train and y_train data.
- Validated the results by comparing the MSE of the values predicted and the actual y_val.
- Applied some regularization to the models.

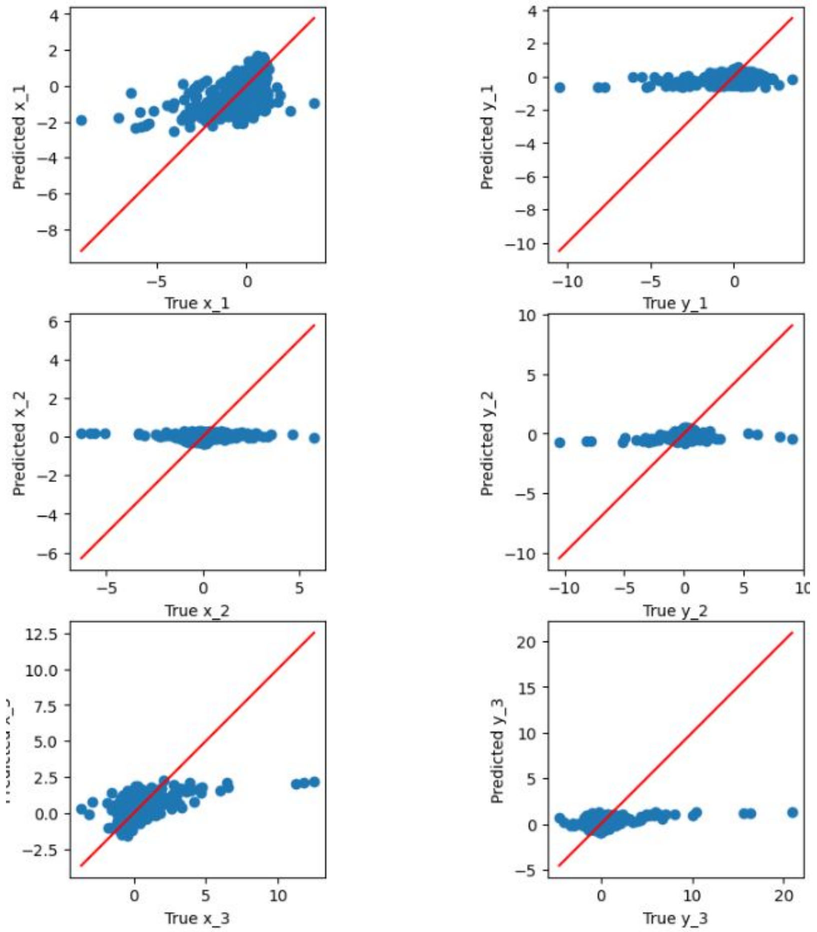
Results

- Normal linear regression
- Mean Squared Error:
  - Train: 1.395378151220457
  - Validation: 1.395378151220457



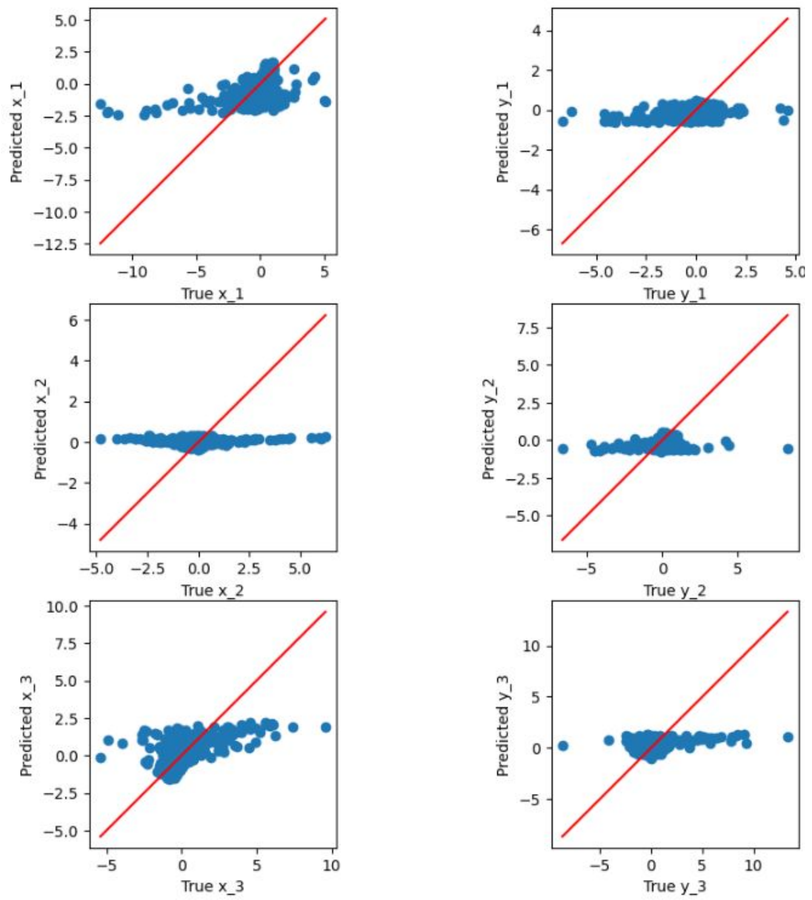
Results

- Ridge regularization.
- Mean Squared Error:
  - Multiple Alpha:
    - alpha: 0.1
      - train: 1.3954014139799398
      - validation: 1.2824569283352207
    - alpha: 1.0
      - train: 1.3954014139799977
      - validation: 1.2824569341456202
    - alpha: 10.0
      - train: 1.3954014139863318
      - validation: 1.2824569922559488
    - alpha: 100.0
      - train: 1.3954014146192175
      - validation: 1.2824575739943913
  - Best alpha: 0.1
  - 1.2824569283352207



Results

- Lasso regularization.
- Mean Squared Error:
    - Multiple Alpha:
        - alpha: 0.1
            - train: 1.4038605082740254
            - validation: 1.2987328831660443
        - alpha: 1.0
            - train: 1.4781031653387606
            - validation: 1.3933986969545684
        - alpha: 10.0
            - train: 1.4781031653387606
            - validation: 1.3933986969545684
        - alpha: 100.0
            - train: 1.4781031653387606
            - validation: 1.3933986969545684
    - Best alpha: 0.1
    - 1.2987328831660443



Conclusion - Linear Regression

- With this particular case, using regularizers like Lasso or Ridge didn’t improve the model performance
- Overall best model was the linear regression without any regularizers.
- Our submission for Kaggle with this model was a very close value to our experiment, validating our results:
- Kaggle Score: 1.42845

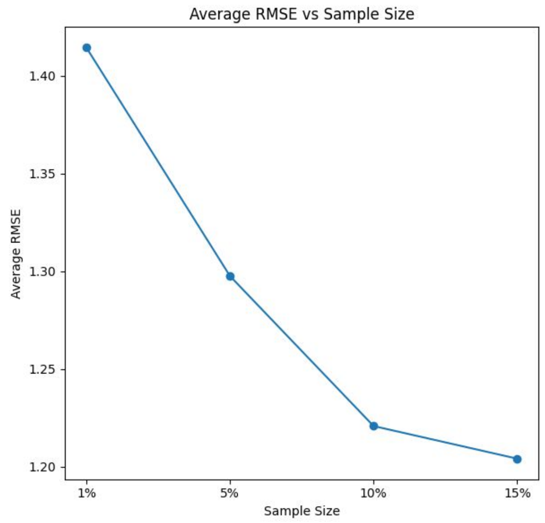
Polynomial Regression

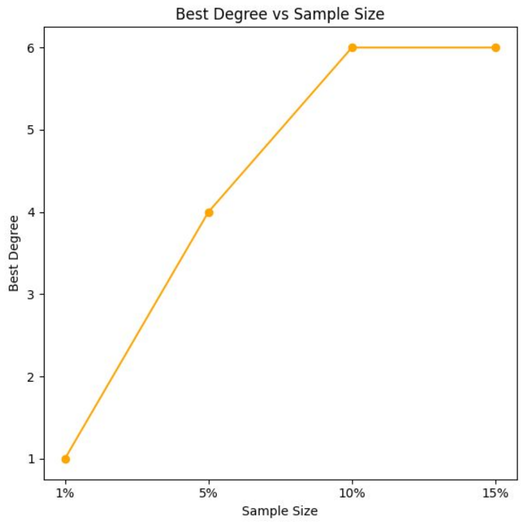
Polynomial Regression

- The polynomial regression was tested with a small percentage of the dataset in order to speed up the selection of the best degree.
- The function evaluate_polynomial, randomly samples a percentage of the data multiple times, compares all the degrees from 1 to 6 and chooses the best polynomial over 20 times, based on the frequency of the best choice.
- Due to computational resources limitations, testing over degree 7 was not feasible.
- Multiple regularizations (Lasso and Ridge) were tried.

Polynomial Regression - No Regressor

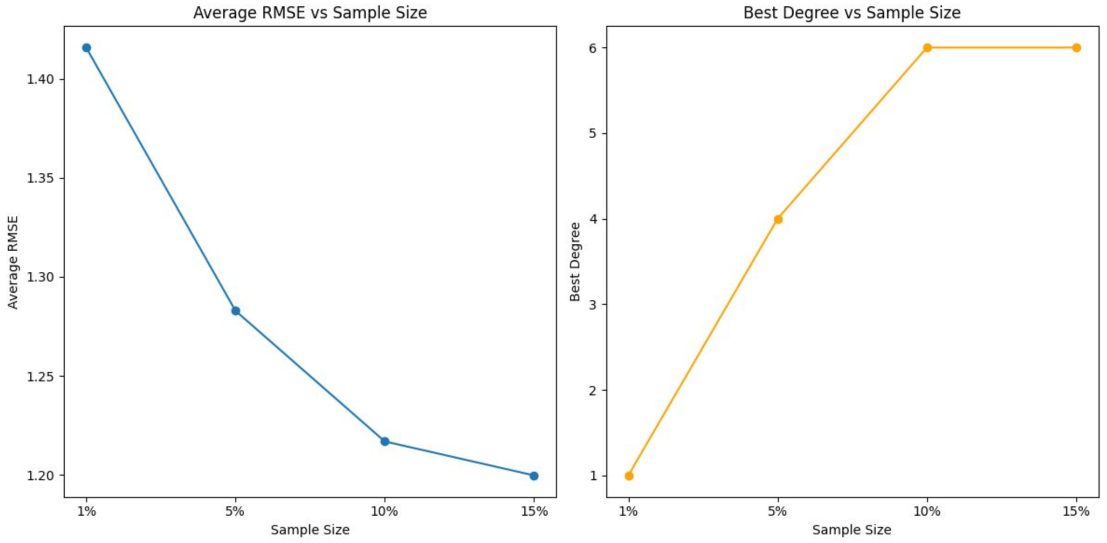
- There is a steep reduction in the average RMSE as we increase the Sample size.
- With larger sample data, the higher degrees also perform better.
- Best Degree 6
- Best Average RMSE: 1.20





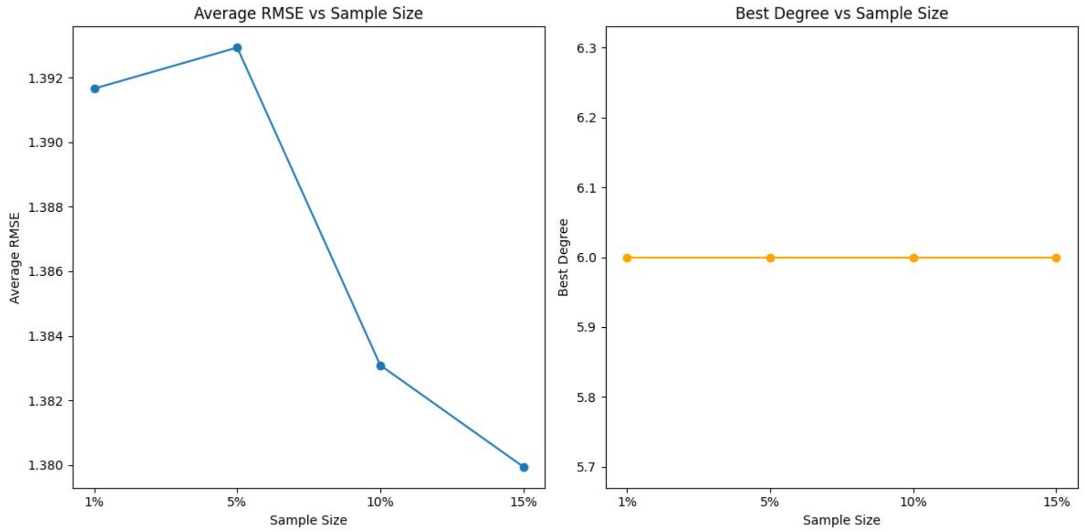
Polynomial Regression - Ridge

- Average RMSE and Best Degree evolution over Sample Size similar to no regularization.
- Faster runtime
  - No regularization: ~15 min
  - Ridge: ~5 min
- Similar Performance
  - No regularization: 1.20
  - Ridge: 1.2



Polynomial Regression - Lasso

- This approach proved to be different, where the average RMSE went up with 5% of sample size (probably some data irregularities).
- The best degree was 6 for all sample sizes.
- Worse performance:
    - No regularization: 1.20
    - Ridge: 1.2
    - Lasso: 1.380



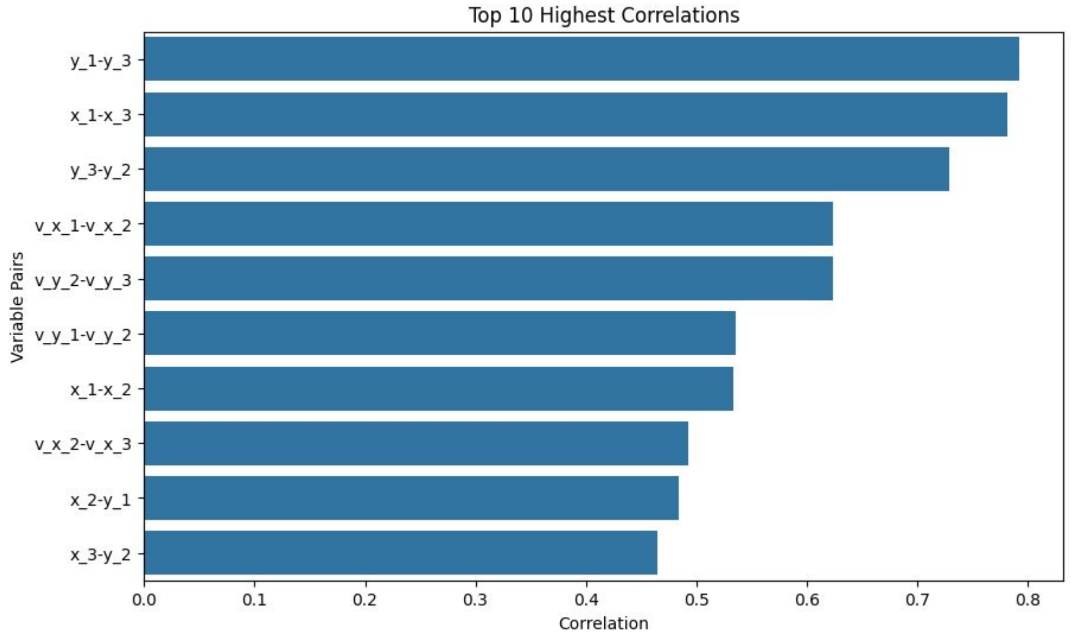
Polynomial Regression - Conclusions

- Overall, the best result was achieved with the polynomial degree 6 and the Ridge regularizer.
- Our submission to Kaggle with a polynomial degree had a MSE a little bit higher to our expectations, but close enough to validate our experiment.
- Kaggle Score for Polynomial Regression: 1.23758

Feature Engineering

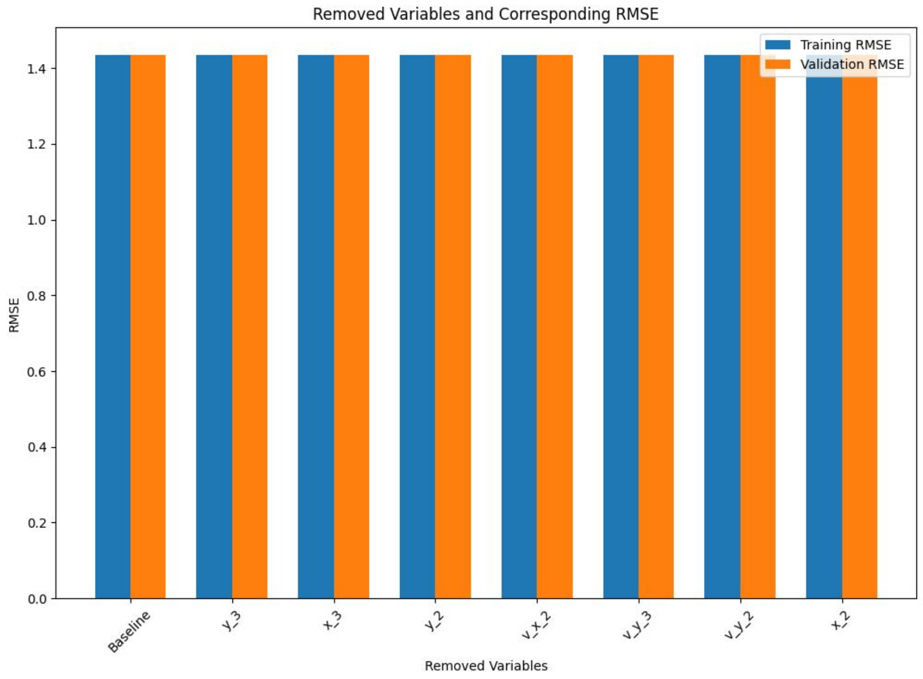
Feature Engineering - Variable correlations.

- For feature engineering, a study for the correlation between two variables was made.
- Tested the hypothesis of dropping some of the highly correlated variables (one of the pair)



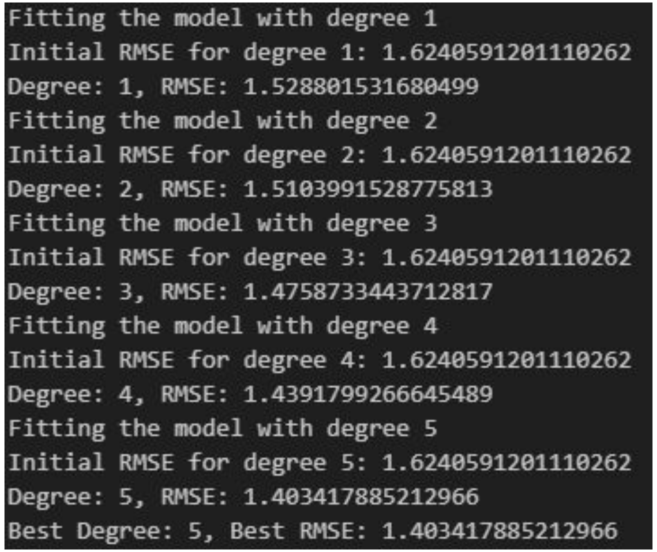
Removing Variables

- After removing the variables with most correlation, we’ve concluded that it doesn’t change significantly the RMSE



Adding Variables

- After adding variables with the highest correlation, we found that the RMSE didn't change significantly. The added variables were the normalized Euclidean distances between x3 and x1, and y3 and y1.



```
Fitting the model with degree 1
Initial RMSE for degree 1: 1.6240591201110262
Degree: 1, RMSE: 1.528801531680499
Fitting the model with degree 2
Initial RMSE for degree 2: 1.6240591201110262
Degree: 2, RMSE: 1.5103991528775813
Fitting the model with degree 3
Initial RMSE for degree 3: 1.6240591201110262
Degree: 3, RMSE: 1.4758733443712817
Fitting the model with degree 4
Initial RMSE for degree 4: 1.6240591201110262
Degree: 4, RMSE: 1.4391799266645489
Fitting the model with degree 5
Initial RMSE for degree 5: 1.6240591201110262
Degree: 5, RMSE: 1.403417885212966
Best Degree: 5, Best RMSE: 1.403417885212966
```

Example of results with polynomial regression using 5 degrees

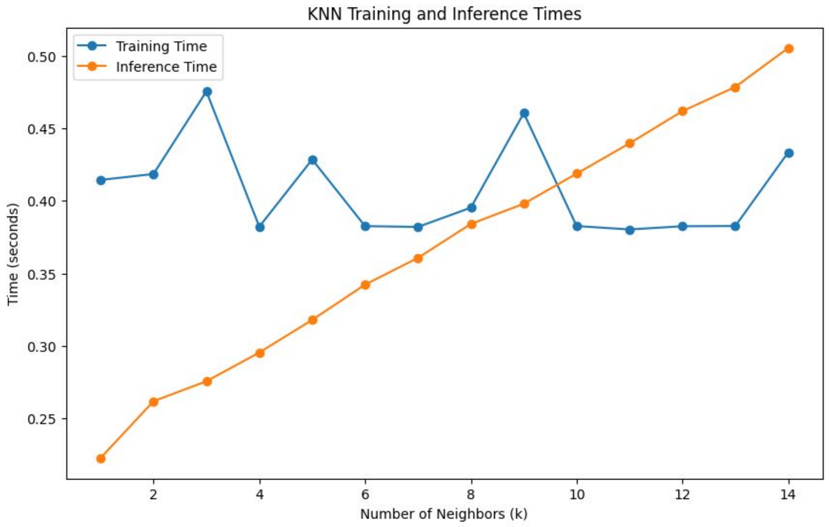
K-Nearest Neighbors

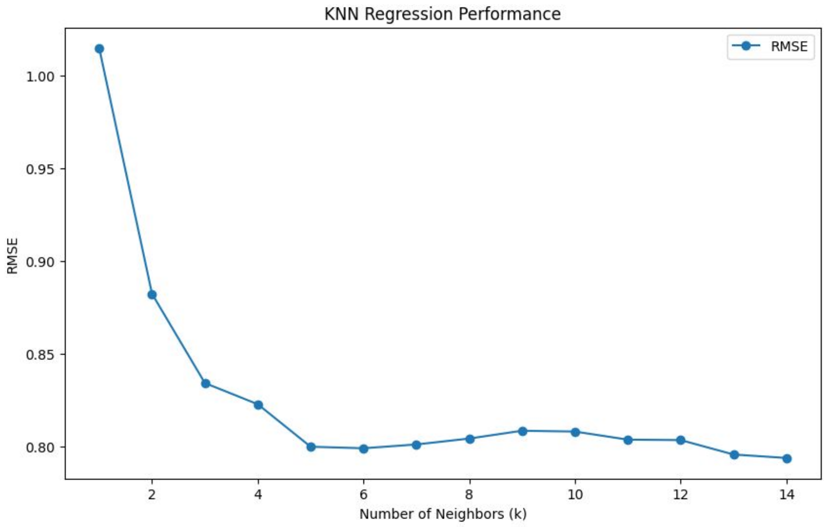
K-Nearest Neighbors

- For the development of the non-parametric model, a study for the best number of k (the number of neighbors) was made
- This was done by testing multiple values for k between 1 and 15, and analyzing the RMSE and the training and inference times

K-Nearest Neighbors - Conclusions

- With this experiment, by comparing the RMSE and the inference time, the best number of neighbors (k) is around 6 or 7.
- For our Kaggle submission, we chose the level 7 and obtained a score of 0.94518





Analysis

- We developed several machine learning model to predict the trajectories of the three body problem:
  - Linear Regression: We regularized the model with the Lasso and the Ridge regressions, with no significant improvement.
  - Polynomial Regression: This model had improved accuracy in higher degrees with the Ridge regularization with degree 6 yielding the best results. The model performed better than the linear regression.
  - K_Nearest Neighbors: We achieved the best performance using K=7, showing the model’s ability to handle non-linearities

Analysis - Key Findings

- With the conclusion of the project we arrived at a few conclusions:
  - Correctly preprocessing the data with its splitting and collision handling were crucial for the performance of the models
  - Our non-linear models outperformed the linear models, highlighting the relevance of complex relationships

Overall Assessment

What went wrong

- Not having a good machine learning workflow can delay and can yield imprecise results. In this particular case, our ML workflow was not the best for incrementing and changing the model easily.
- Using Python Notebooks can be useful for visualizing the plots, however it can get confusing when using a single file. One must be careful when using variables to not introduce any data leakage into the models.

What went great

- Understanding the problem requirements and doing every task.
- Understand how the principles of automatic learning can be applied in the real world