**Agenda**:

* Merging all datasets together
* Iterative thresholding
* Model comparison

**Merging all datasets together:**

I merged WHO and World Bank datasets that measured indicators of women’s health, economic status, health system status, environmental status, illness prevalence, and women’s empowerment.

*Merging process:*

* For each dataset, I extracted each indicator and re-labelled the ‘estimate’ column with the indicator name.
* I then merged all columns by country, date, dimension, and subgroup.
  + Dimension could measure things like ‘Sex’ and ‘Economic status (wealth decile)’.
  + Subgroup is a subset of the dimension (e.g. ‘Female’).

*Potential issues to flag:*

* Some indicators are the same, but are named different things, causing duplication in the dataset:
  + E.g.:
    - ‘Infant mortality rate (deaths per 1000 live births)’
    - ‘Mortality rate, infant (per 1,000 live births)’
  + However, the duplicated indicators contained different data (potentially due to different collection mechanisms).
    - For example, in this case, the rightmost indicator is measured until a year of age while the leftmost is measured until 11 months.
  + Will including both indicators increase nuance of the training dataset?

A screenshot of a computer

AI-generated content may be incorrect.

* The different datasets have been joined on country, year, dimension and subgroup to ensure that indicators for men are joined to those about men (for example).
  + When an indicator is not recorded for that specific subgroup, its value is listed as NAN.
    - However, this could mean that subgroups recorded for only a handful of features could be lost via thresholding (e.g. the empowerment indicators).

**Iterative thresholding:**

I applied iterative thresholding to the full dataset with the following results.

* I first tried iteratively thresholding by columns then rows. However, when I split the data into train/test subsets, some columns had only missing values.
  + I fixed this problem by thresholding by rows before columns.
* After applying thresholding for the rows and columns of the dataset, I tested whether any of the new columns or rows breached the threshold. I only stopped iterative thresholding when no breach occurred.

When no iterative thresholding is applied (threshold = 100%), approximately 94% of the dataset is missing. As a point of comparison, when a 50% iterative threshold is applied, roughly 8.8% of the dataset consists of missing data. As expected, the proportion of missing data left in the dataset increases as the threshold increases.

A graph with blue dots

AI-generated content may be incorrect.

As the threshold increased, the number of rows and columns left in the dataset also increased. If no thresholding is applied (threshold = 1.0, or 100%), the dataset contains 667 columns/features and 50,115 rows. In contrast, when a threshold of 50%, or 0.5, is applied, the dataset contains only 382 columns and 852 rows. A threshold of 95% balances the need for missing value removal (~68% of the dataset contains missing values), with preserving as many columns and rows as possible (441 columns and 9245 rows). Therefore, I will be using a threshold of 95% when testing combinations of different imputation and machine learning models.

A graph with blue dots

AI-generated content may be incorrect.

A graph with blue dots

AI-generated content may be incorrect.

*Note on columns lost during the iterative process:*

* I created and added two CSV files to my Gituhub. On the first page (100% threshold) of each, I have a full list of all the features/columns in the dataset
* Each page of the csv files contains a list of columns lost due to that level of thresholding.

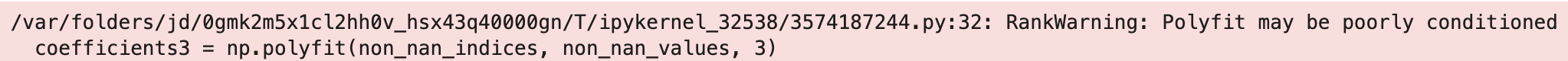
**Model training procedure**:

* Split the data into training and testing sets.
  + There was a training and testing set for each of the 100% and 95% thresholded data.

*Imputation methods:*

* KNN (neighbours=5)
* Miss Forest imputation
* 1st order polynomial imputation
* 2nd order polynomial imputation
* 3rd order polynomial imputation
* Iterative imputation
  + An imputation method from sci-kit learn (subject to change because it is experimental) that imputes missing features using the values of other features. It imputes one feature at a time in a round robin fashion.
  + Initially, when I used this function, I received errors about values in my imputed dataset having inifinite values. Changed the max\_value from np.inf to 500,000 to fix this.
  + Unfortunately, this function took a very long time to converge and its use was not practical (the kernel kept dying in my Jupyter notebook). Therefore, it was excluded from the analysis.

As a note on the imputation methods, when I impute the 100% thresholded training set with polynomials of order 3, I receive the following error for one column (for split within year):



I receive this error for four columns on 100% thresholded testing data for imputation with all orders of polynomial on the split within year data.

Thus, wherever possible, the testing set was not imputed. When this was impossible (i.e. the model could not work with missing data), imputation was used. This could be a source of prediction error to keep in mind.

The split by year dataset produced more of this error when imputed with polynomials. Multiple instances of the error occurred for both the training and testing datasets.

*Machine learning models tested*:

* No imputation needed: (no imputation was done on the test set)
  + Random forest
  + XGBoost
  + LightGBM
* The following machine learning models cannot work with missing data. Therefore, they were only tested with the 95% threshold (to prevent whole rows or columns from being missing, for which imputation would not work). Additionally, the testing set used for model prediction was imputed where necessary with the same method used for the training set.
  + AdaBoost
  + KNN Regressor
  + Support vector machines
  + Multi-layer perceptron regressor

To compare the different models and imputation methods, I plotted the mean average percentage error for each model-imputation combination on 100% and 95% thresholded data separately.

\*can I compare models that had imputed testing sets, as these sets would differ?

Split within year:

*100% thresholded data:*

Only the random forest, XGBoost, and LightGBM models were included in the following plot, as the other models could not work with a full column or row having missing data.

A screen shot of a computer

AI-generated content may be incorrect.

Observations:

* When no thresholding occurs (100% threshold) and no imputation is done to the testing sets, the random forest model performs the best by an order of magnitude regardless of imputation method.
* While the gradient boosting models perform more similarly, XGBoost appears to work better for the majority of imputation methods. Nevertheless, LightGBM with Miss Forest imputation produces the lowest error after the worst random forest model.

To better visualise the Random Forest result:

A graph of blue rectangular bars

AI-generated content may be incorrect.

* The polynomial imputation methods perform approximately 3% better than the KNN and Miss Forest imputation methods.
* While 3rd order polynomial imputation performs the best, all polynomial methods performed within a percent of each other (4 to 5% MAPE).

*95% thresholded data:*

A screen shot of a graph

AI-generated content may be incorrect.

Observations:

* The SVM and Multi-Layer Perceptron models, as well as XGBoost with KNN and Miss Forest imputation (overlapping) performed the worst. They were excluded and the data was re-plotted to better visualise the best performing models.

A screen shot of a graph

AI-generated content may be incorrect.

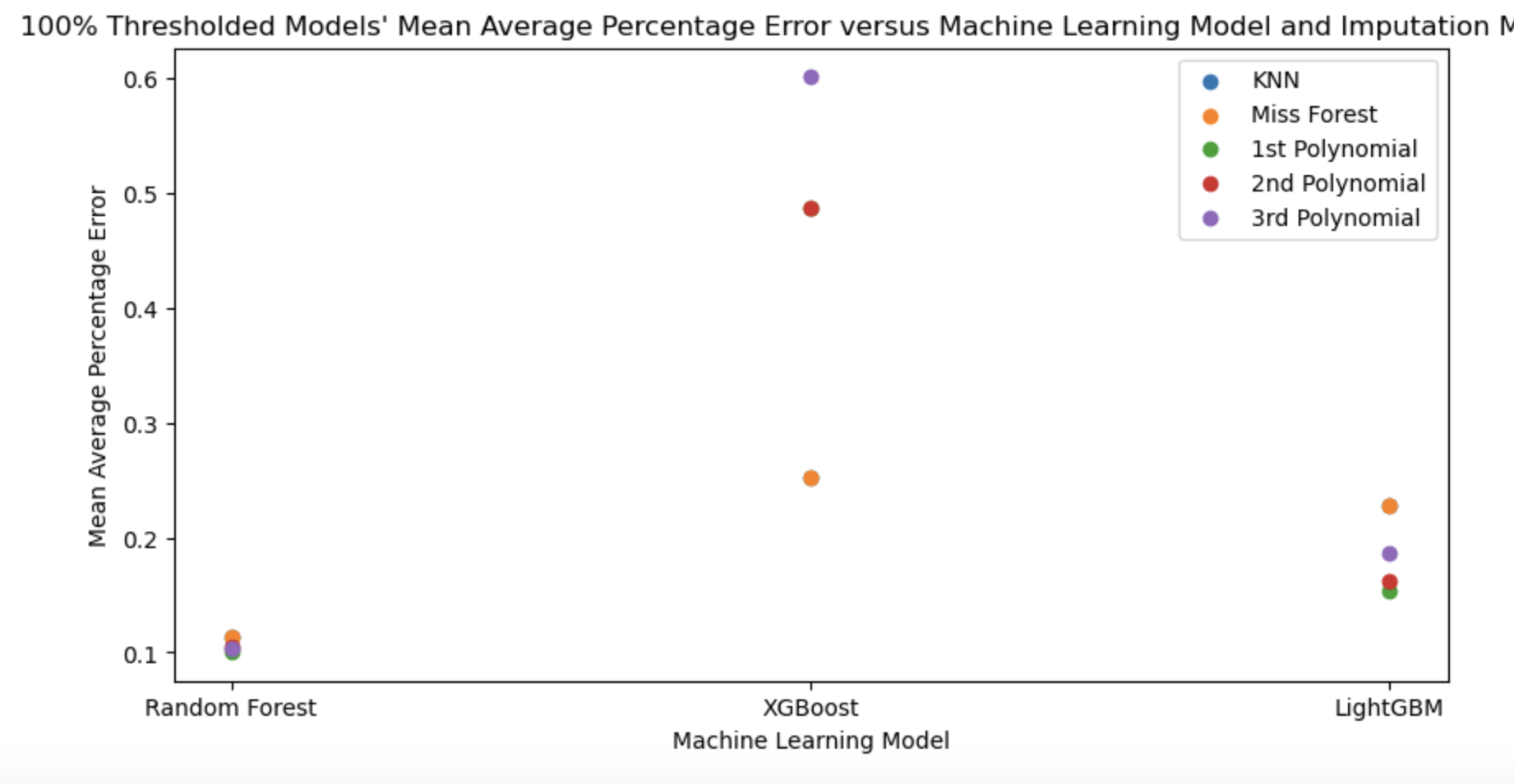
Observations:

* AdaBoost performs the worst, with a MAPE of approximately 0.35 across imputation methods.
* Random Forest still performs the best, with the 1st order polynomial imputation producing the most accurate results of the possible imputation methods.
* KNN regressor performed the second-best of the models tested.

Split by year:

At first, I split the dataset by classifying 2000-2014 as training and 2015 to 2019 as testing. However, this meant that 95% thresholded testing data could contain columns with 100% missing data, preventing some of the models from working. Therefore, I instead split the data by classifying 2000-2012 as training and 2013-2019 as testing.

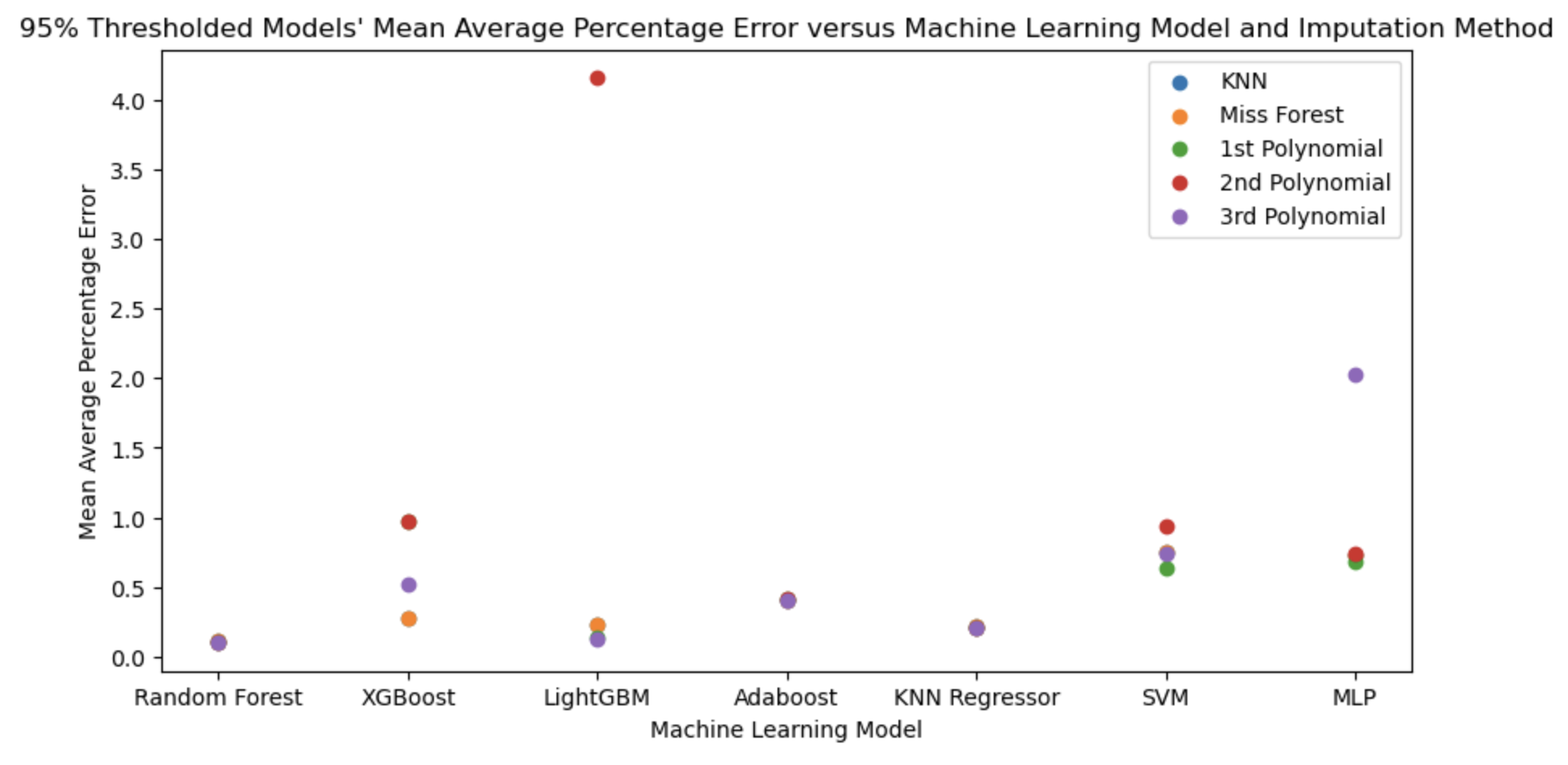
*100% thresholded data:*



Observations:

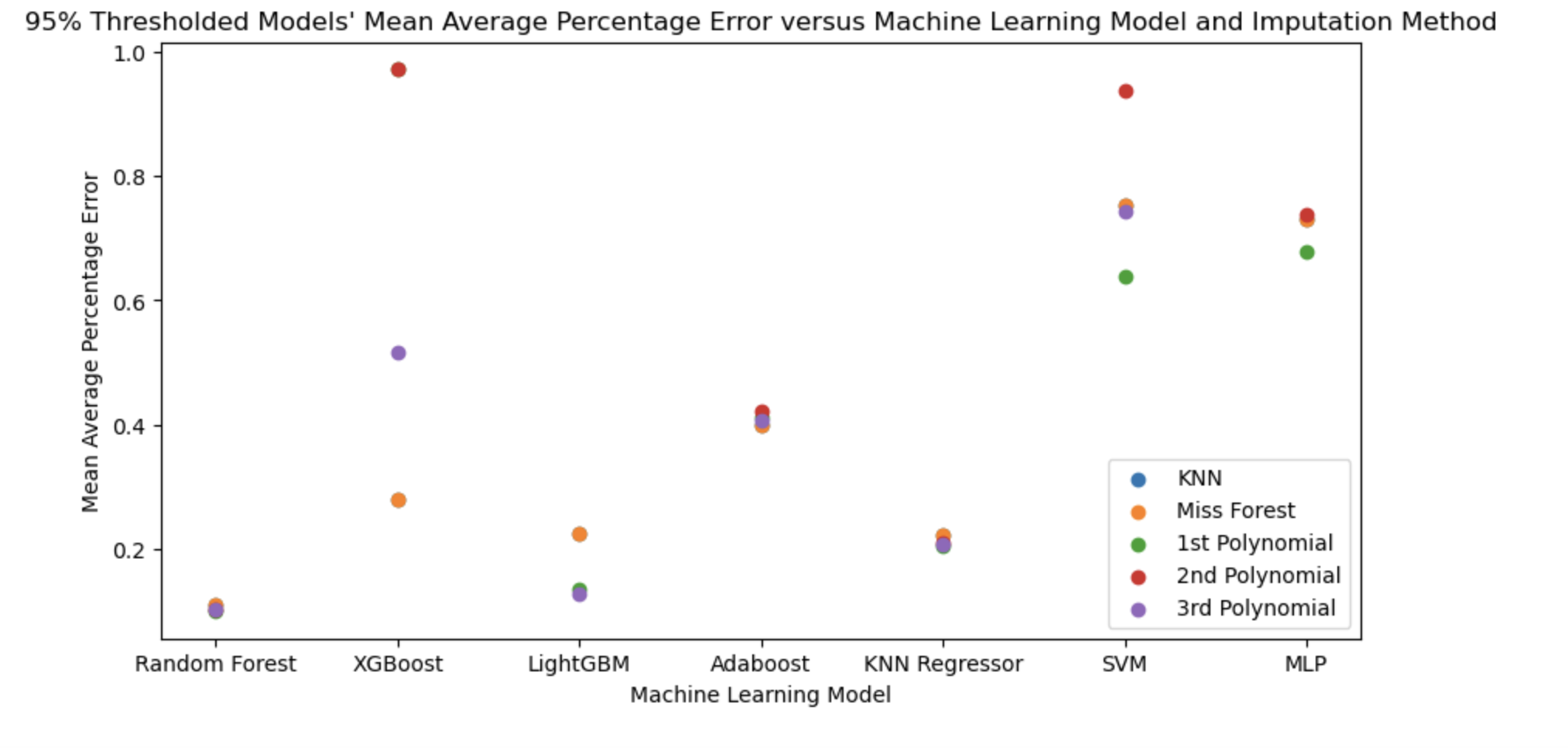
* Random Forest again is the most accurate model, with 1st order polynomial imputation producing the smallest MAPE.
* LightGBM is the second-best model while XGBoost is the worst.
* The difference between LightGBM’s best MAPE and Random Forest’s worst MAPE is only a few percentage points.
* The difference in MAPE between imputation methods from a single model type is smallest for Random Forest and largest for XGBoost.

*95% thresholded data:*



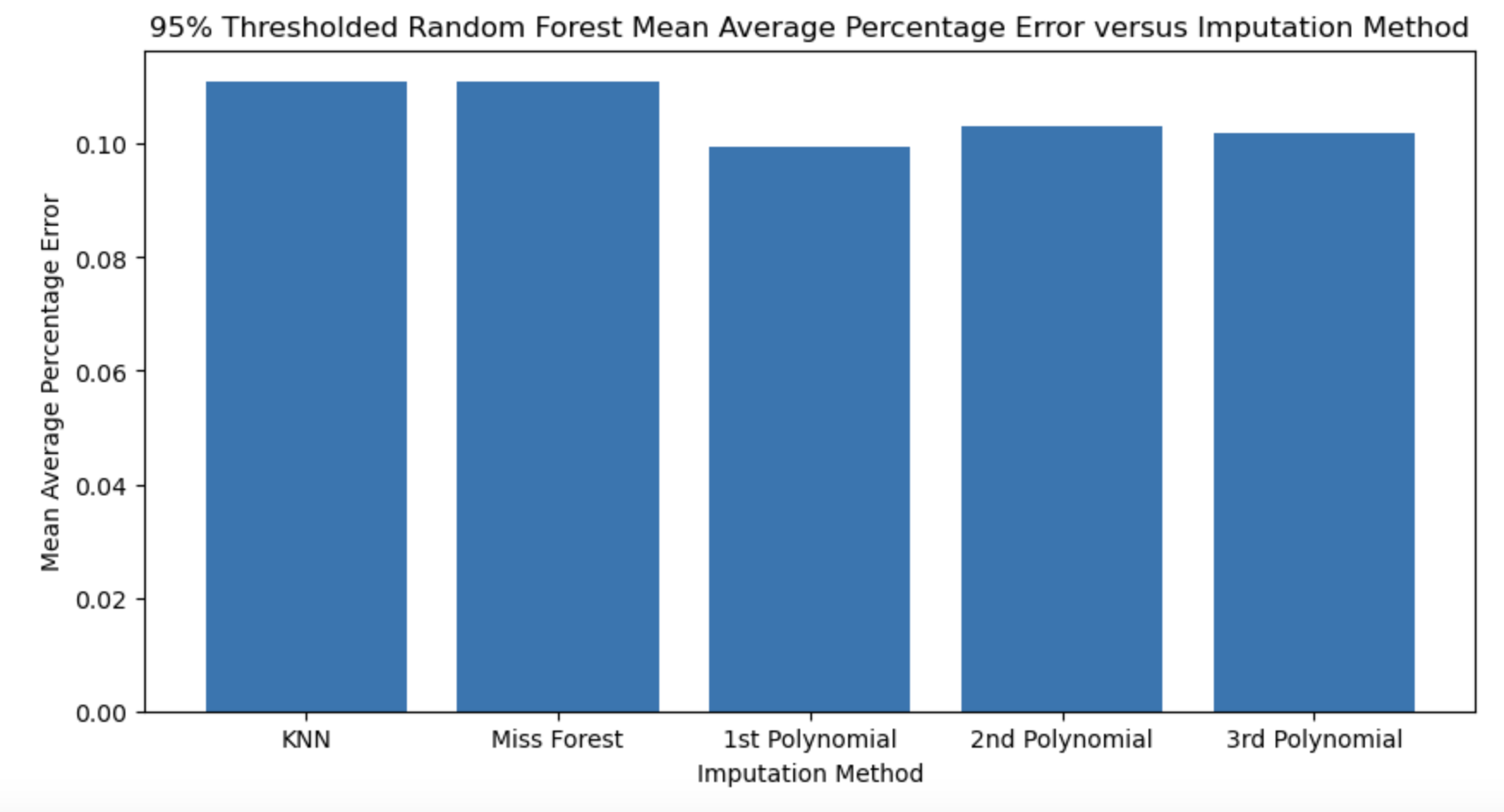
* 2nd order polynomial imputation + LightGBM and 3rd order polynomial imputation with MLP had the highest errors of any tested imputation/model combination.

These combinations were removed and the data was re-plotted to better visualise the results.



Observations:

* Random Forest again is the best model, with the MAPE of the other models fluctuating between 0 and 1.0.
* The SVM and MLP models performed the worst (MAPE scores consistently above 0.6) while LightGBM generally performed second-best.
* Random Forest, AdaBoost and KNN Regressor had the smallest imputation method-dependent difference in MAPE scores.



* The Random Forest method performed the best when imputation was done with KNN or Miss Forest. However, the difference between imputation methods was within 1-2 percent.

**Overall Observations:**

* Regardless of data split method or thresholding, Random Forest had the lowest MAPE, with different imputation methods producing similar MAPE scores.
* Across both data split methods, KNN Regressor performed the second-best for a threshold of 95% and LightGBM /XGBoost performed similarly for a 100% threshold (with the best of LightGBM beating the best of XGBoost).

**To-Dos:**

* Hyperparameter testing
* Introduction drafting to better understand the context of this research.
* Validation set?

Comments:

* Label random forest as ‘Sklearn Random Forest’ to be more specific
* At some point, you have a large enough dataset where you bootstrapping has no effect
  + Can compare bootstrapping vs no bootstrapping for different oversampling sizes