University of St. Gallen

School of Economics, Law, Social Sciences, International Relations and Computer Science

INDIVIDUAL TASK SUBMISSION: Predicting BMI

By

Mathias Steilen

Matriculation N°: 19-608-512

Course: Data Analytics I: Predictive Econometrics (7,310,1.00)

Lecturer: Prof. Dr. Jana Mareckova

03.02.2023

Contents

List of Figures

List of Tables

1 Overview over folder structure

This page is only meant to give you an overview over the folder structure that I submitted, in case you want to dig into the code I wrote. You can ignore the files on the first level that were used for knitting this PDF. **The modelling code for each approach is in the Models folder**.

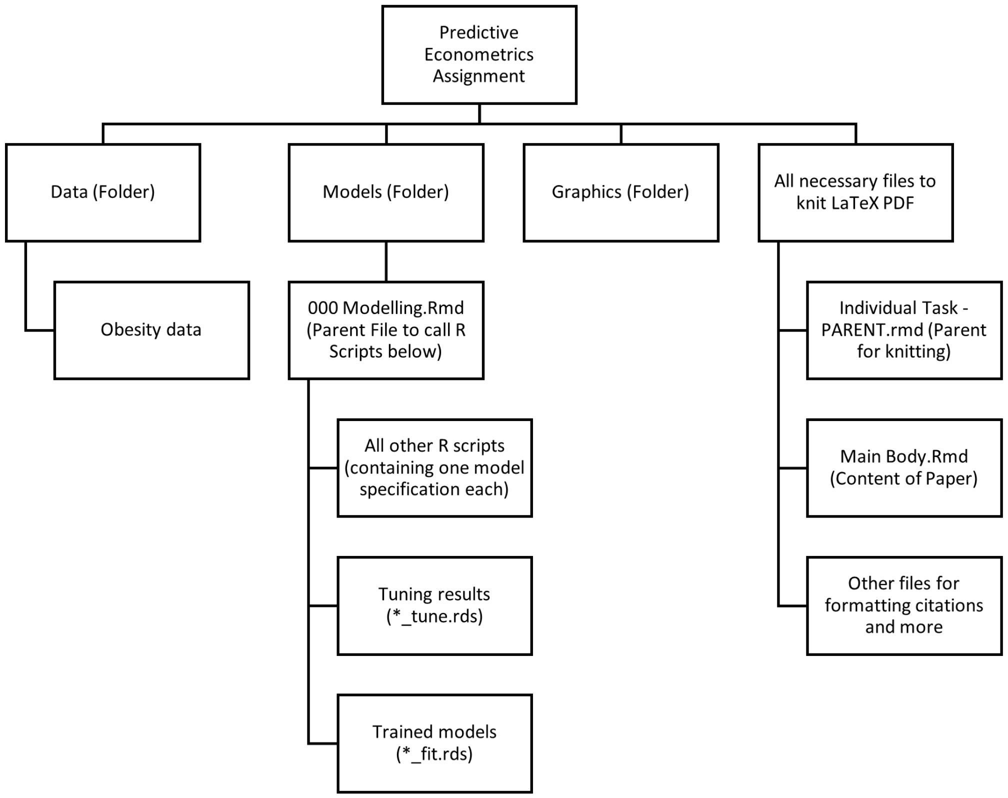
Predictive Econometrics Assignment
Data (Folder)
Models (Folder)
Graphics (Folder)
All necessary files to knit LaTeX PDF
Obesity data
000 Modelling.Rmd (Parent File to call R Scripts below)
All other R scripts (containing one model specification each)
Tuning results (*_tune.rds)
Trained models (*_fit.rds)
Individual Task - PARENT.rmd (Parent for knitting)
Main Body.Rmd (Content of Paper)
Other files for formatting citations and more

Figure 1: Overview over the project directory

2 Question 1

Are there any issues with the data (missing values, problematic labels, small groups in some categories, useless predictors, etc.)? Describe your data cleaning procedure.

Upon reading in the data, I converted all character columns to factors, in order to be able to use the `tidymodels` library for modelling, which requires to do that.

2.1 Data types

After reading in the data, it looks like most columns have the correct data type, i.e. there are no numeric columns that should be factors and vice versa. Two thing I noticed though, were the ordinal columns (FCVC, NCP, CH20, FAF and TUE) and the factor columns (CAEC and CALC). The former are currently stored as numeric, and it might be a consideration to convert them to categorical (factor). The latter are currently stored as factors, as they are free text, but might be encoded as ordinal variables (numeric), as they have a natural order, like the other ones.

I tried the default configuration of XGBoost, SVM and Elastic Net to test for what I should do with both of these cases. For FCVC, NCP, CH20, FAF and TUE stored as factors, it showed a slight negative impact for XGBoost, but improved the OOS R^2 for Elastic Net and SVM. Naturally, the other case (CAEC and CALC) painted the inverse picture. Therefore, I chose to convert the ordinal columns to factors for the Elastic Net and SVM, and do ordinal encoding on CAEC and CALC factor columns for the Gradient Boosting Approach, in order to improve all methods.

2.2 Missing values

There are no missing values in the data.

```r
colSums(is.na(data)) %>% enframe() %>% filter(value > 0)
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: name <chr>, value <dbl>
```

2.3 Problematic labels or small groups in categories

In order to check for problematic labels or small groups, I visualise the value counts of all nominal variables in Figure 2. As can be seen, there is a high class imbalance in virtually all predictors except for gender. However, there are a few extremely small groups within the nominal predictors:

- CALC: "always": n = 1
- MTRANS: "bike" and "motorbike": n = 5, n = 9
- SMOKE: highly unbalanced: only 2.13% smokers

My approach for these small groups will be using `step_other` from the `recipes` package within the `tidymodels` framework. This function takes a tunable threshold parameter, which sets the minimum frequency for levels within nominal predictors, below which all levels will be lumped into a generic "other" category. During tuning, the threshold parameter can be treated just like the other hyperparameters. It turned out that the threshold was always above 0, indicating that the model benefits from lumping infrequent levels together.

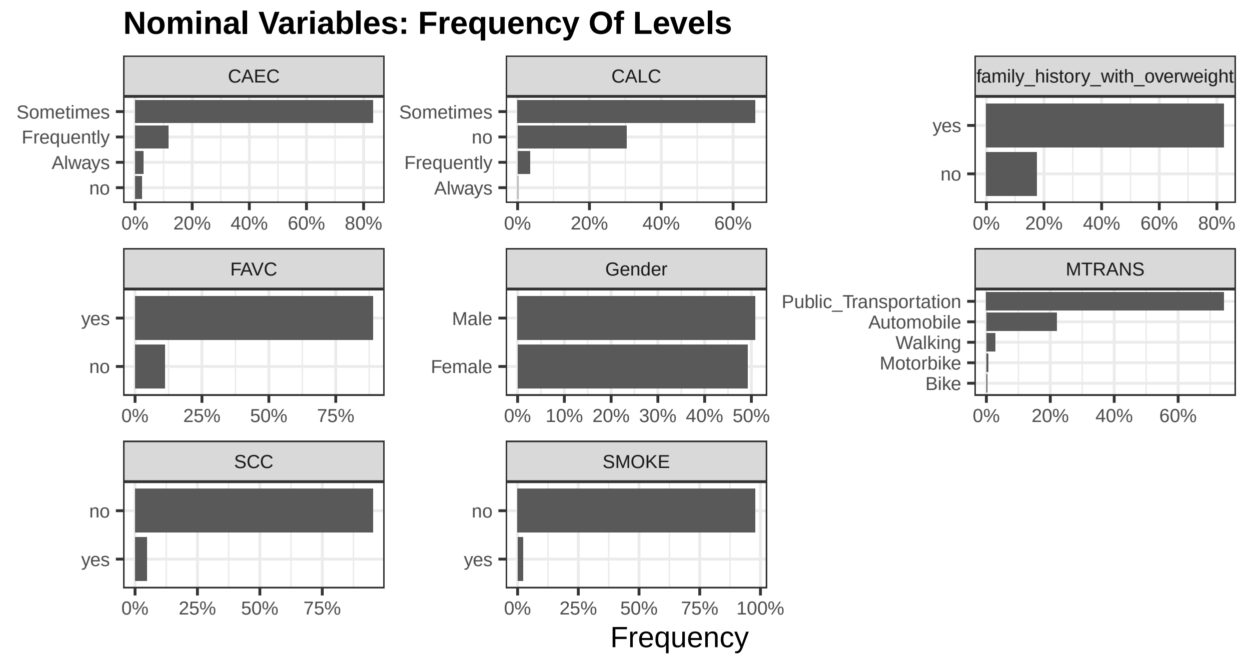

Figure 2: Counts of nominal variables

Similarly, I analysed the distribution of numerical predictors by using histograms, as shown in Figure 3, in order to check for skew or infrequent levels within ordinal variables. The target variable is not right skewed, so a log transform might not actually be that beneficial. Age and height are also well behaved. The ordinal predictors are imbalanced at times, but not to a problematic degree.

There are quite a few 18 and 21 year olds and the number 26.7 shows up quite frequently, so I checked for duplicates and lo and behold, there are duplicated rows:

```
table(data %>% duplicated())
```

```
##
## FALSE  TRUE
##  1671    17
```

The question is, whether these are actual duplicated cases or if there is a tiny chance that those are actually different individuals. I made the decision to drop them.

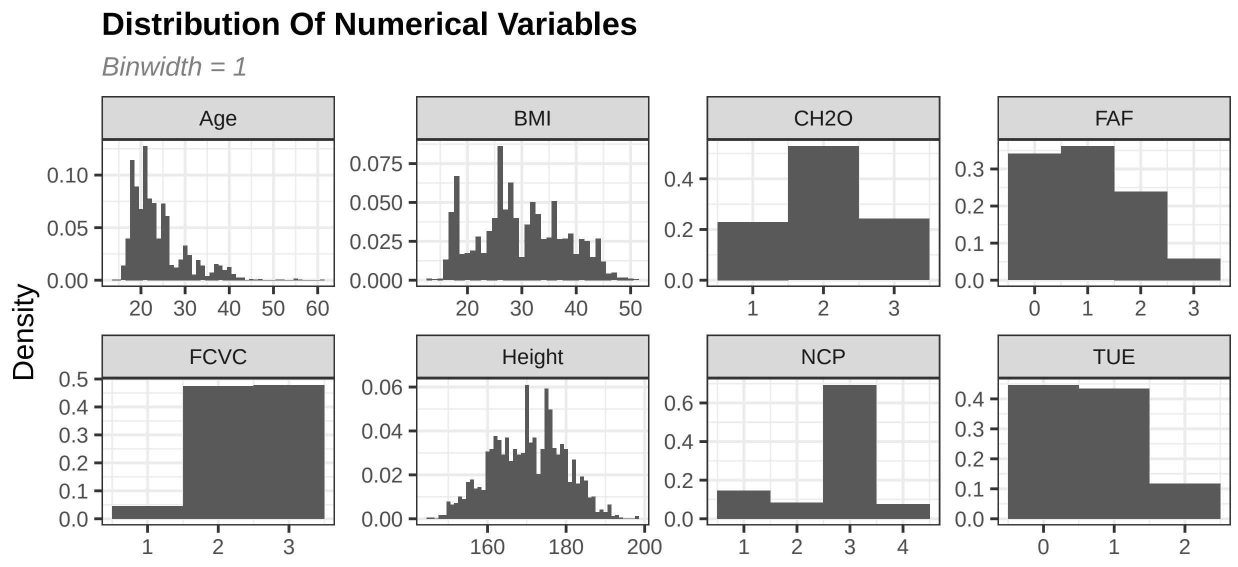

Figure 3: Distribution of numerical variables

2.4 Useless predictors

Useless predictors might be random or zero variance variables. In my preprocessing pipeline, I am including a zero variance filter for every model variation that I'm trying, to the second case is covered. Nonetheless, I visualise the relationship of the predictors with the target using boxplots and scatter plots in Figure 4 and Figure 5.

The main takeaways are:

- Nominal predictors: There don't seem to be random variables without any relation, though gender and smoking are not highly predictive by themselves.
- Numeric predictors: None of the numeric predictors have a tight relationship with BMI, however there are slight trends. In combination with the nominal predictors, there might be interactions and non-linearities that can be exploited from more flexible methods. Discretisation of height and age might be helpful to some models.

2.5 Outliers

From the distribution charts, it looks like there are no visible outliers in any variables that would have to be dealt with, though there are a few old people. These can be dealt with using discretisation into bigger buckets, which I tried during modelling. It remains to be seen whether it is actually beneficial for the model performance - as that is the only thing that really counts.

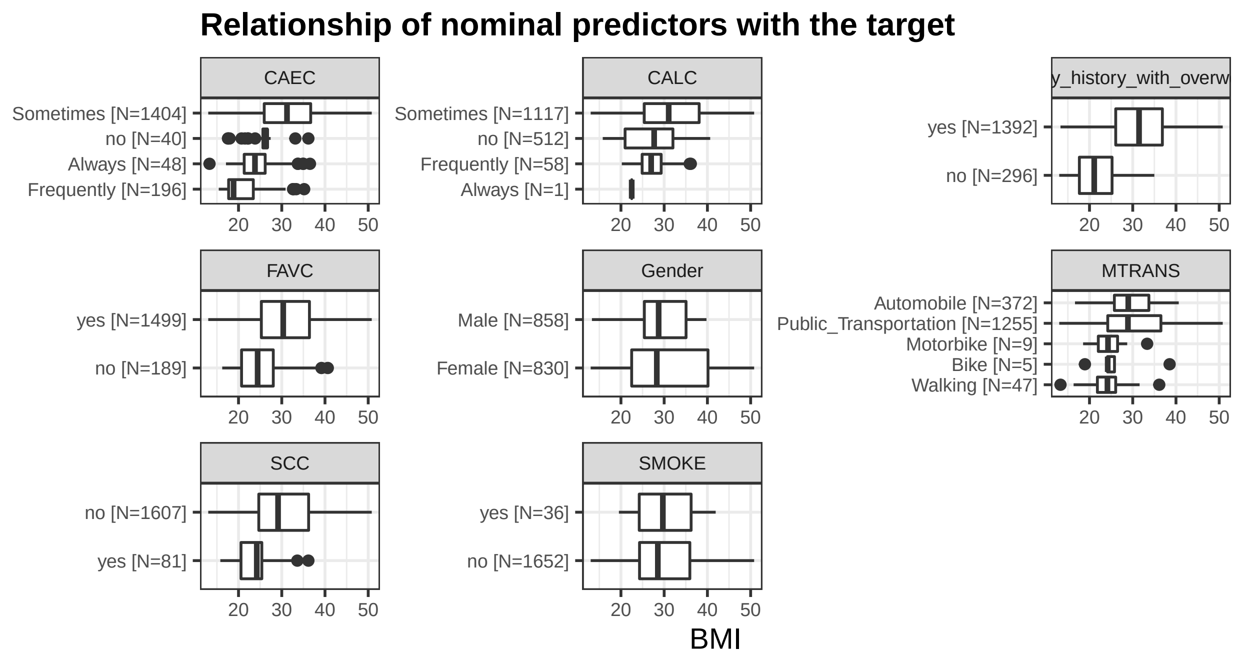

Figure 4: Relation of nominal predictors with the target variable

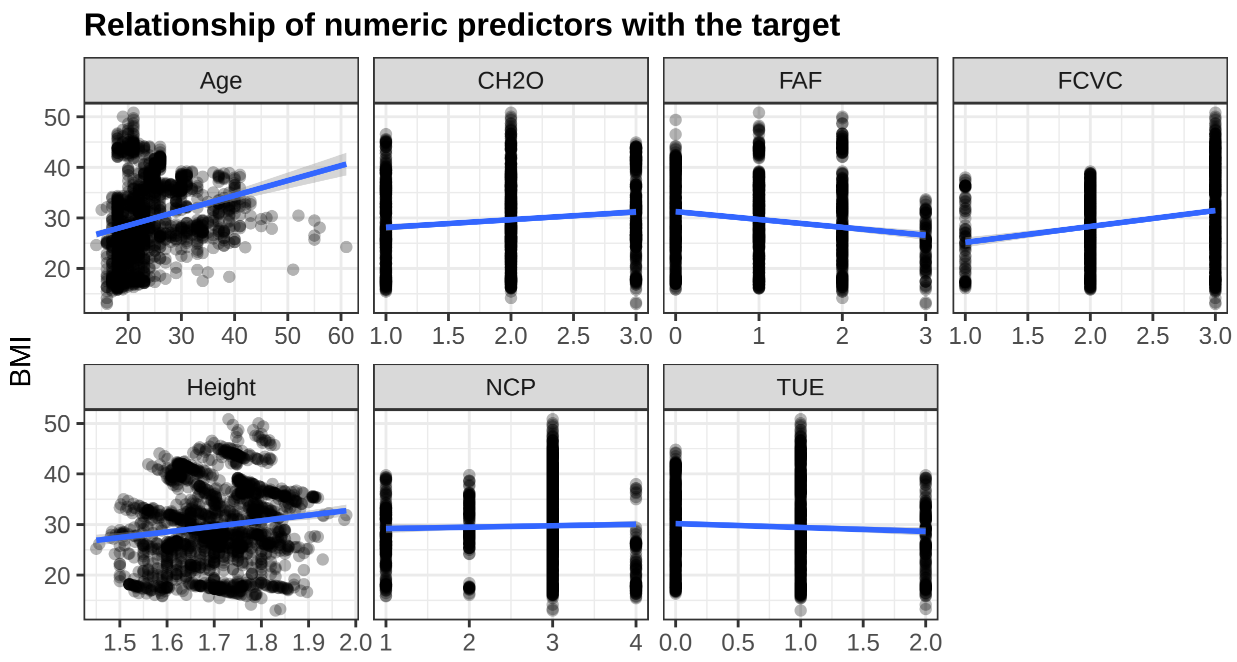

Figure 5: Relation of numeric predictors with the target variable

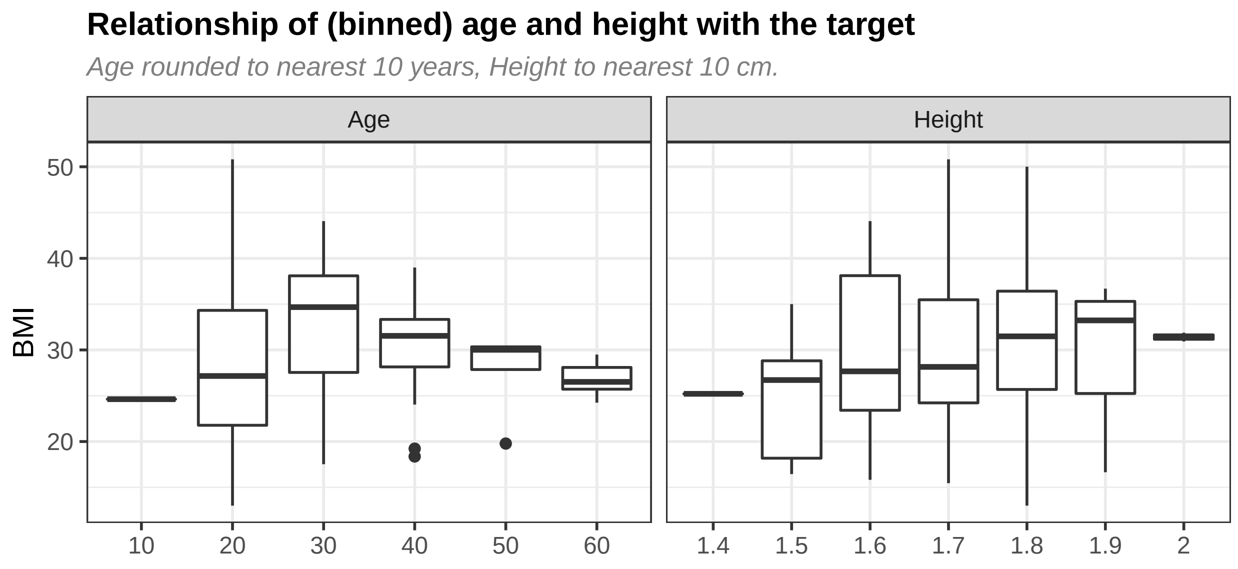

Figure 6: Relation of binned numeric predictors with the target variable

3 Question 2

Describe your preferred prediction method/approach. Did you transform variables in training and test data? Did you partition the data? How did you select the tuning parameters? Which estimation method did you use? Why did you select this method/approach for your preferred specification?

3.1 Specification and Splits

Before modelling, I split the data with the target into training and testing and then split the training data further into 5 folds for cross validated hyperparameter tuning. Tuning several hundred hyperparameter combinations on the five folds, I selected the hyperparameter combination with the best cross-validated R^2 on the holdout within the folds and trained the model on the full training set, in order to maximise the amount of data before I evaluated each model on the holdout data set I set aside previously. I called the model scripts from one single "parent" script (using `source`), in order to have the same splits and folds for the different model types and eliminate randomness when comparing between them.

3.2 Preprocessing/Transformation of the best candidate

My preferred approach is using Gradient Boosting (XGBoost) allowing for new factor levels, removing zero variance predictors and creating dummy variables from nominal predictors. I tried five different preprocessing approaches, as can be seen in Table 1, and compared the out-of-sample R^2 (on the test set, which wasn't used for hyperparameter tuning and fitting the training data). Surprisingly, the model with the least preprocessing worked best for XGBoost. Not only was this

Table 1: Different preprocessing approaches for gradient boosting

Script Number	Gradient Boosting
In all scripts	Allow new factor levels Remove zero variance predictors Dummies for nominal predictors
1	Nothing additional
2	Normalise numeric
3	Normalise numeric Log scale target
4	Lump together infrequent levels
5	Log scale target Discretise height and age

setting better than all other XGBoost models, but it also dominated all other SVM and Elastic Net models on the holdout set (using mean absolute error, mean absolute percentage error, root mean squared error and R^2). Therefore, I chose this model and fit the best specification on the entire data with the target variable as a final step, in order to maximise data input before making predictions on the holdout without the target.

The performance of each of the five different XGBoost models can be seen in Figure 7. Clearly, *gb1*, which is the first script, worked best with an OOS R^2 of 0.855 and the lowest RMSE. A short point on the OOS metric: As the assignment is graded on R^2, I went with R^2 to select the best model from tuning. However, in practice, if the cost of making large errors is worse than making small errors, using MSE might be more advisable. If all errors are equally bad, then MAE might be better. The point is that adapting the metric to the use case at hand is important, but that I went with R^2 as this is the metric that the predictions are evaluated on for the assignment.

3.3 Tuning

All final five XGBoost methods were similarly tuned, in order to enable comparison between each preprocessing approach. Specifically, I tuned:

- *trees*: An integer for the number of trees contained in the ensemble.
- *tree_depth*: An integer for the maximum depth of the tree (i.e. number of splits).
- *min_n*: An integer for the minimum number of data points in a node that is required for the node to be split further.
- *loss_reduction*: A number for the reduction in the loss function required to split further.
- *sample_size*: A number for the number (or proportion) of data that is exposed to the fitting routine.
- *mtry*: A number for the number (or proportion) of predictors that will be randomly sampled at each split when creating the tree models.

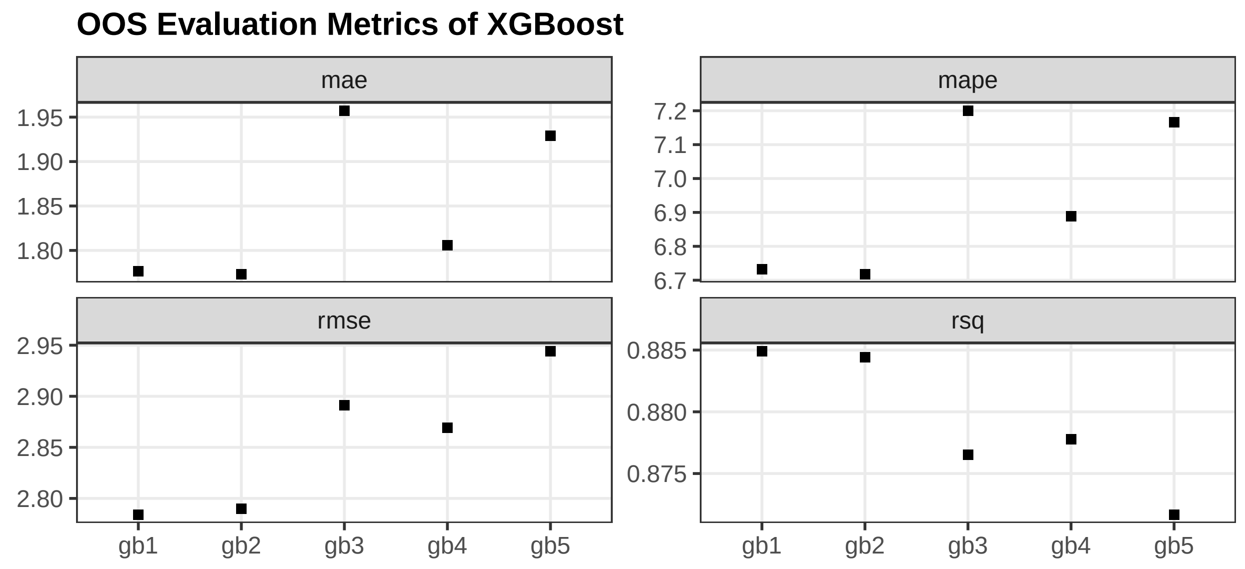

Figure 7: Out-of-sample evaluation metrics for XGBoost

- *learn_rate*: A number for the rate at which the boosting algorithm adapts from iteration-to-iteration (shrinkage parameter).

Where applicable, I also tuned the *threshold* in the preprocessing pipeline, which determines the levels below which infrequent factor levels get lumped together. I used the folds from the cross validation approach on the training data in order to compute stable performance metrics. The values I provided for the parameters came from a space-filling design (latin hypercube). The space-filling design is particularly useful in cases like this, where the hyperparameter space has high dimensionality, as it helps to achieve sufficient coverage of the space at a computationally feasible level of hyperparameter combinations. Additionally, I used parallel processing enabled by the `doParallel` package, in order to speed up the tuning process using 7 of the 8 cores on my CPU.

Table 2: Different preprocessing approaches for Elastic Net and SVM

Script Number	SVM	Elastic Net
In all scripts	Allow new factor levels Remove zero variance predictors Normalise numeric predictors	Allow new factor levels Normalise numeric predictors Dummies for nominal predictors Remove zero variance predictors
1	Nothing additional	Nothing additional
2	Dummies for nominal predictors	Lump together infrequent levels
3	Lump together infrequent levels	Discretise height and age
4	Log scale target	Log scale target
5	Discretise height and age Lump together infrequent levels	Interactions between all predictors
6	Log scale target Lump together infrequent levels	Natural splines for Age and Height
7		Natural splines for Age and Height Interactions between all predictors

4 Question 3

Besides your preferred specification, did you test additional variable transformations?

As mentioned in Question 2, I wrote different preprocessing pipelines in different scripts and then called them from the parent file. I used the same approach for the other models as well (Elastic Net and SVM), which can be seen in Table 2.

4.1 SVM Preprocessing Approaches

For SVM, normalising numeric predictors is required to optimise the margin, therefore I included it in all approaches. Additionally, I removed zero variance predictors and allowed for new factor levels that have not been encountered in the training process.

As seen in Figure 8, creating dummies from nominal variables hurt the OOS performance. Similarly, discretising height and age did not work well for SVM. The best one is *svm3*, with best OOS metrics across the board and it featured lumping infrequent levels in factors together. SVM was really different here, as it benefitted greatly from this step, as opposed to XGBoost for example.

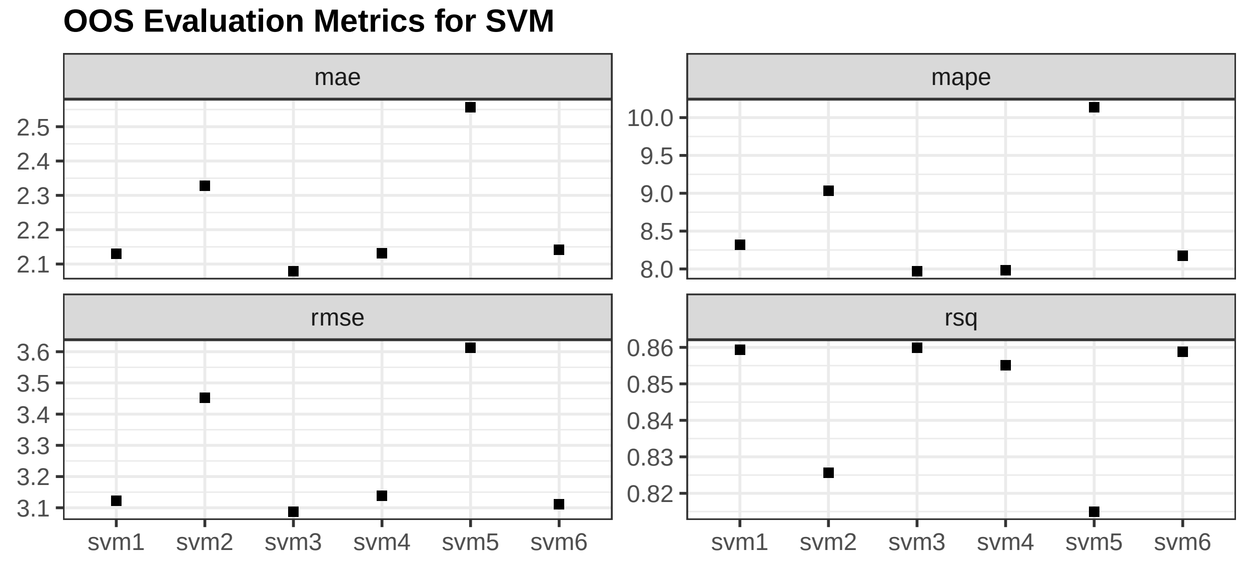

Figure 8: Out-of-sample evaluation metrics for SVM

4.2 Elastic Net Preprocessing Approaches

For the elastic net, normalisation is also required, as the penalties will not treat all predictors equally otherwise. Additionally, elastic net needs dummy encoding as well (though that could not be one-hot encoding like in gradient boosting due to multicollinearity, though Lasso should be able to deal with that). Additionally, I also allowed for new factor levels and removed zero variance predictors.

As can be seen in Figure 9, the biggest jump was made by including interaction terms for all predictors after dummy encoding - it was a huge spike. Including natural splines for age and height helped as well. The best model is *lin7*, which includes both interaction terms and natural splines and works by far the best, almost coming near SVM.

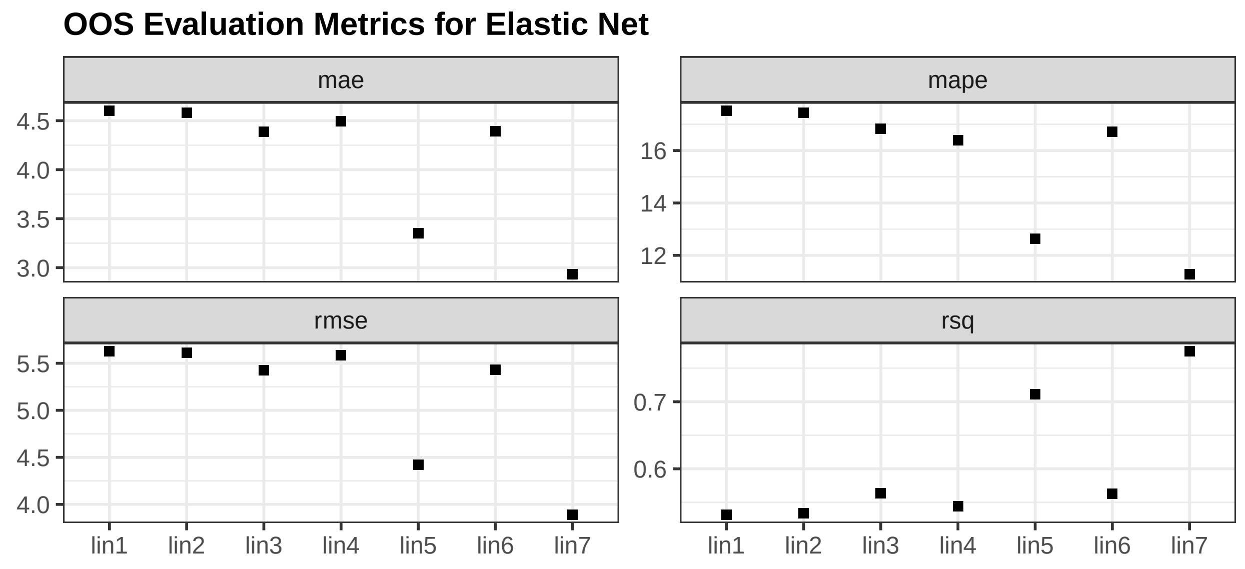

Figure 9: Out-of-sample evaluation metrics for Elastic Net

5 Question 4

Besides your preferred specification, did you test additional ways to select the tuning parameters and/or partition the sample?

5.1 Splits

Using the most simple model specifications for each model type, I experimented with stratified resampling and found that setting the strata to be BMI, in order to preserve the distributions of the target in both train and test split benefitted the out-of-sample performance. This seemed logical, as the model performance we hope to achieve stands entirely on the assumptions that the relations between predictors and target remain the same, going from training to deployment in praxis, where we do not observe the target beforehand.

Furthermore, I tried a higher number of folds for cross validation than 5, but as the standard error of the performance metrics was not excessively high, I decided to go for the benefit of making the training times faster and allow me to cycle through more hyperparameter combinations, thus benefitting the overall modelling process. Additionally, I played with the proportion of the train/test split, but decided to keep the default of 75%, as it seemed to be a good enough tradeoff between ability to evaluate and ability to train the model, given the amount of data that we have.

5.2 Tuning

I tried using a regular grid as opposed to a space-filling design, but the benefit of covering the hyperparameter space more optimally was very noticeable in the training time and made training much more efficient.

6 Question 5

Besides your preferred specification, did you test additional estimation methods/approaches?

As already mentioned above, besides Gradient Boosting, I tried Support Vector Machines and Elastic Net with the preprocessing/transformation steps outlined in Question 3.

For the SVM, I used a radial basis function and tuned both the cost parameter and the sigma of the function with the space-filling design on a few hundreds of combinations, outlined above. For the Elastic Net, I used the *glmnet* engine and tuned the linear mixture parameter between Ridge and Lasso, as well as the penalty parameter lambda.

As can be seen in Figure 10, XGBoost performed the best out of all three model types on the holdout data, which is the only metric that should be considered. From Figure 10, it also becomes apparent that all models beat the baseline of a featureless predictor.

7 Question 6

Did you use or test methods/approaches that were not covered in the lecture?

The Support Vector Machine using a radial basis function and gradient boosting algorithm I used were not covered in the lecture.

7.1 Stacked Model

Additionally, I created a blended model of candidates from the tuning process of the best XGBoost model, which are then linearly stacked using a lasso approach to further increase predictive performance of the best model. The performance of the *gb_blended* model can be seen in Figure 10 - it is only marginally better than *gb1*. In practice, it must be debated, whether the increased complexity and computational expensiveness is worth the potential marginal increase in performance. On top of that, not only is it more expensive to train, but also more difficult to explain to potential stakeholders. For this reason, I decided to forego the potential minimal marginal benefit of using the more verbose model and go with the best XGBoost specification (*gb1*) that I have presented in Question 2 and can also be seen in Figure 10.

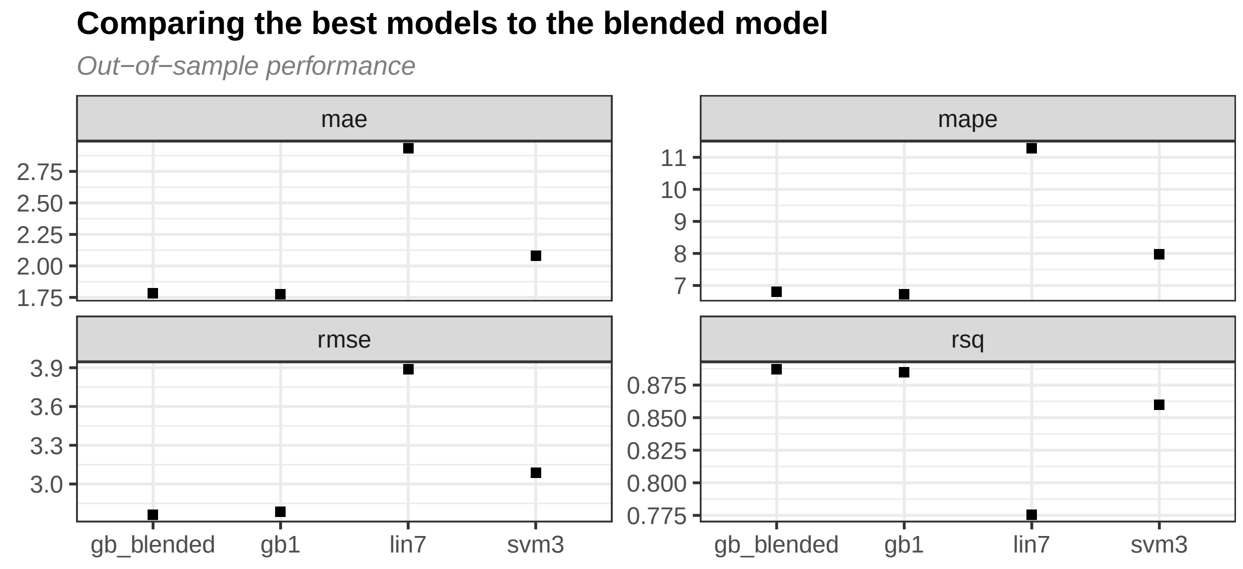

Figure 10: Out-of-sample evaluation metrics for the best models within each model type

8 Appendices

8.1 Appendix A: Additional Figure on Variable Importance

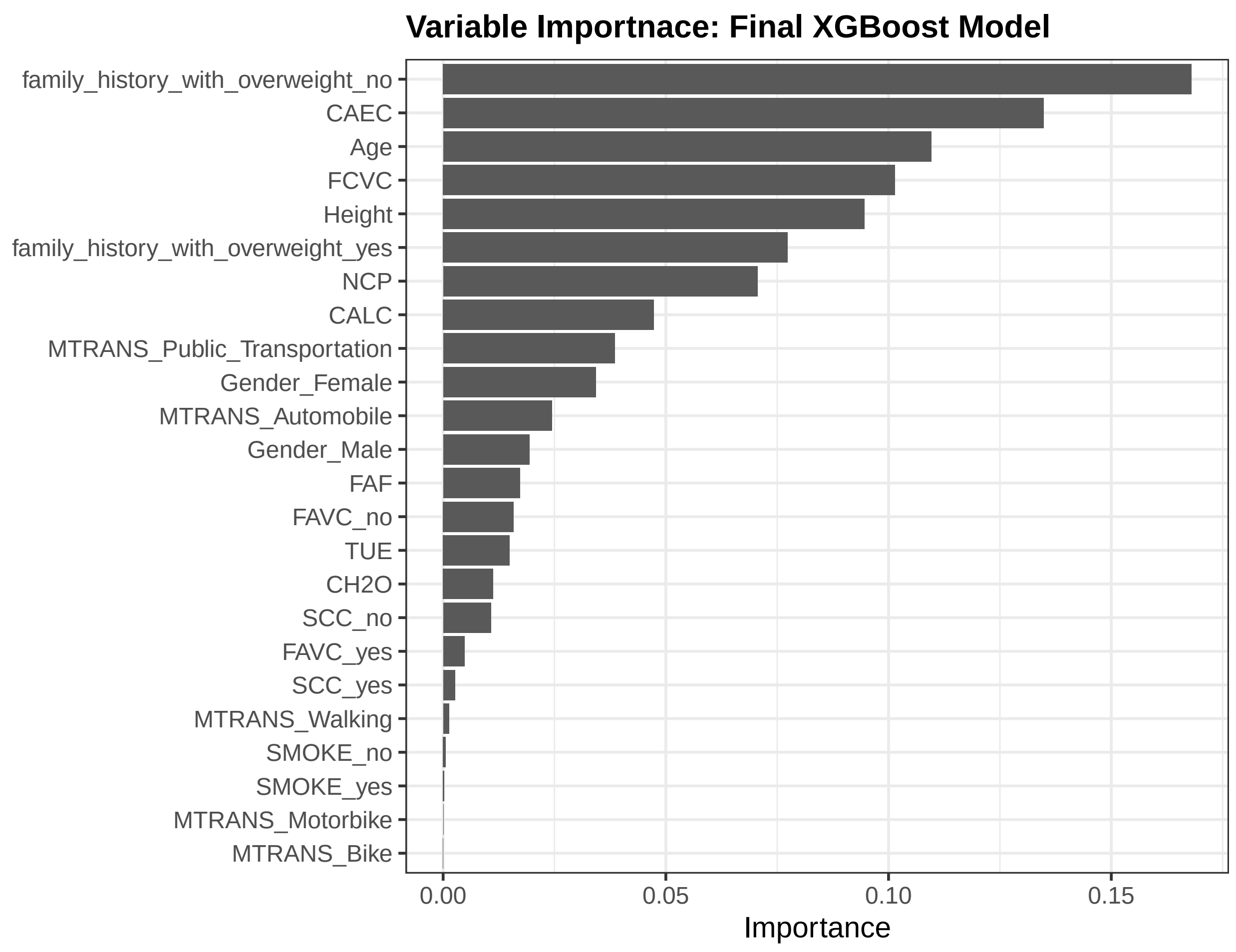

Figure 11: Variable importance (Gini Impurity) for the final gradient boosting model