1. Workflow

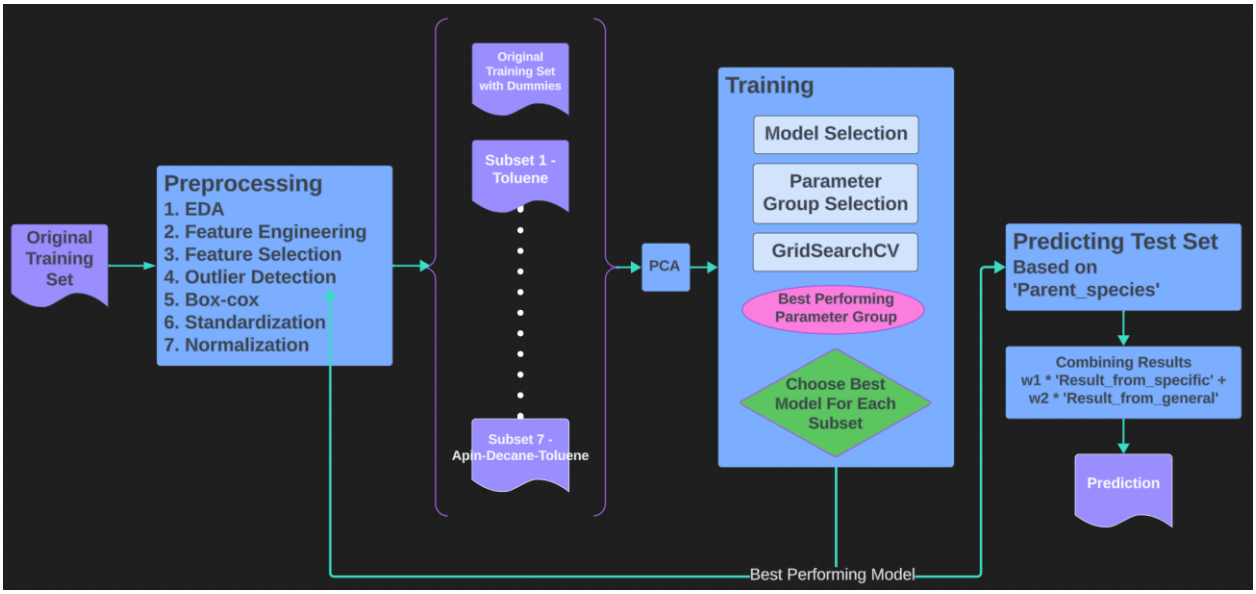

Figure 1

1.1 Data Preprocessing

- **Exploratory Data Analysis (EDA)**: We firstly performed a thorough EDA to understand the distributions of every predictor and the target, statistical characteristics of predictors, and the correlation between response and each predictor. This step as a guideline gives us a general view of the features, inspiring following works.

- **Feature Engineering**: Based on results derived from EDA, we observed that response intervals are obviously different given distinct parent species. So we decide to split the data set into subsets according to parent species, and work on each subset independently. At the same time, we also work on the original training set, in order to make an averaged prediction for the whole training set. Further steps involve creating dummy variables for the original training set, observing and analysing each subset, and removing inconsistent features for each species manually. This step helps in reducing future workload.

- **Feature Selection**: After feature engineering, we dropped features with all-0 values, since these features do not have statistical meanings and introduce extra computation.

- **Outlier Detection**: We applied Z-statistic iteratively to detect outliers in the observations. By setting up a threshold, we removed observations whose z-score exceeds the threshold.

- **Boxcox Translation**: We applied box-cox translation to reduce skewness and stabilise the variance of features. It helps in making distributions of features more close to normal, which is the assumption of many statistical methods and models.
- **Standard Scale**: We applied standard scale to make different features comparable by bringing them to a common scale with mean of 0 and a standard deviation of 1.
- **Normalisation**: We applied normalisation to make features more interpretable, since features after standard scale include negative value, while in the original training set, all features are positive.

1.2 Dimension Reduction

- **Dimensionality Reduction**: We applied PCA to reduce the dimension of the dataset while keeping the majority of variance explained. This helps in improving the efficiency of computation, removing collinearity, and improving the interpretability of our models.

1.3 Model Selection & Training

- **Model Selection**: For the original training set and each subset(splitted according to parent species), we trained several models including Linear Regression, Random Forest, and Support Vector Machine. We conducted model training iteratively with GridSearchCV, which picked up the best performing group of parameters of each model for us.
- **Model Evaluation and Validation**: We assessed performance using the R2 score and MSE during hyperparameter tuning. For each subset, we selected the best performing model.

1.4 Result Analysis & Prediction

- **Making Prediction**: We made the prediction by combining results from each individual subset with results from the original training set. The weight of the result from a subset depends on the size of the subset.

2. Implementation

2.1 Data Preprocessing

1. **Exploratory Data Analysis**

We firstly plot the distribution of response ‘log10(pSat_Pa)’ related to each feature, and found the distribution of response in relation to ‘parent_species’ interesting, shown below.

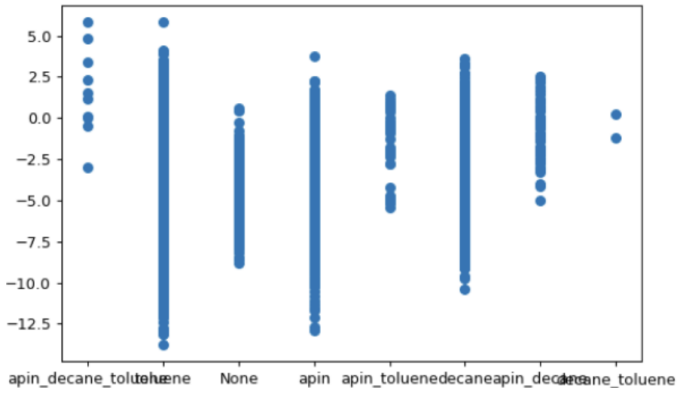

Figure 2

We observed significant differences in response intervals of different species and believed it is not a coincidence since the response values are derived from applying log10 on pSat_Pa, which has already mitigated the variance significantly.

We also noticed the overall distribution of response ‘log10(pSat_Pa)’ is close to normal distribution, shown in figure 3. This finding provides guidelines for outlier detection and model selection.

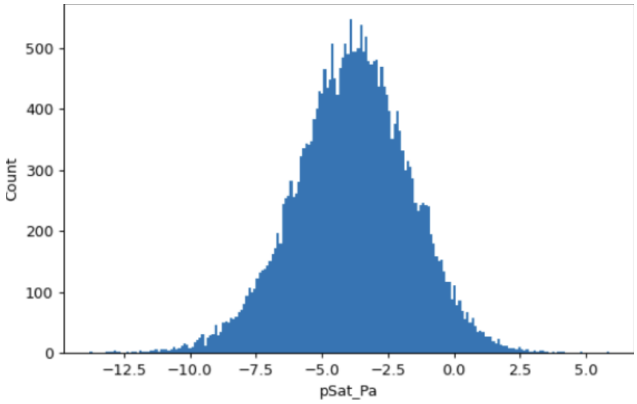

Figure 3

2. Feature Engineering

Based on knowledge derived from EDA, we splitted the training set into subsets by ‘parent_species’ and worked on each subset separately. We also kept the original training set and created dummies to replace ‘parent_species’.

Next, we analyse each subset, since all subsets are estimated and processed similarly. Our goal in this step is to remove features with low correlation with the target variable, and have inconsistent distribution in the train set and test set.

We will take the ‘decane’ subset for example in this part. We firstly analysed the correlation between ‘log10(pSat_Pa)’ and each feature. Left part of figure 4 is the heatmap representing the correlation, with numerical value on right. We set the threshold to ‘±0.1’. Features with an absolute correlation value less than 0.1 are considered to have low correlation with the target variable.

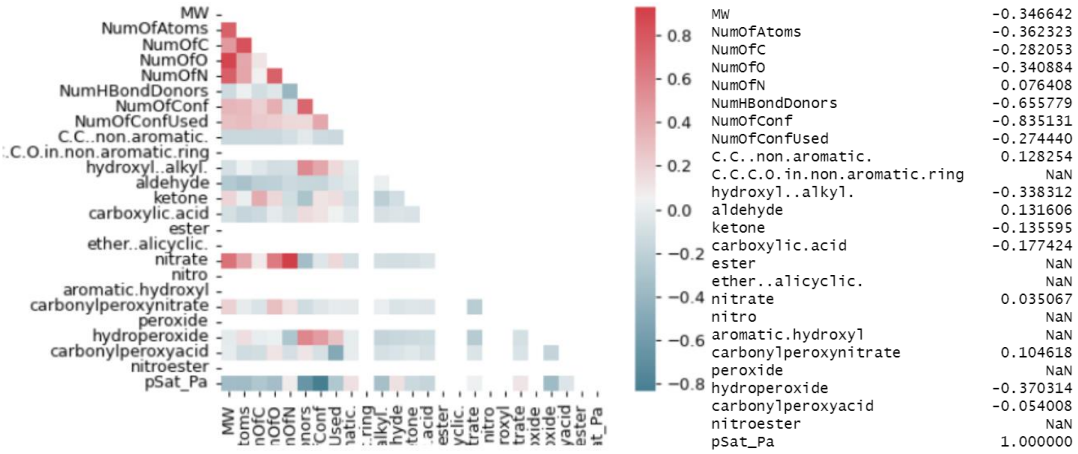

Figure 4

Then, we plot pairwise distributions of some features in the training set and test set. We believe a valid feature is more likely to have consistent distribution in both train set and test set. Take the distribution of the feature ‘NumOfC’ of ‘decane’ for example, which is shown in figure 5. Despite the existence of slight differences, the overall shape of its distribution in the train set and test set is consistent. Therefore, ‘NumOfC’ is considered a valid feature for the ‘decane’ subset.

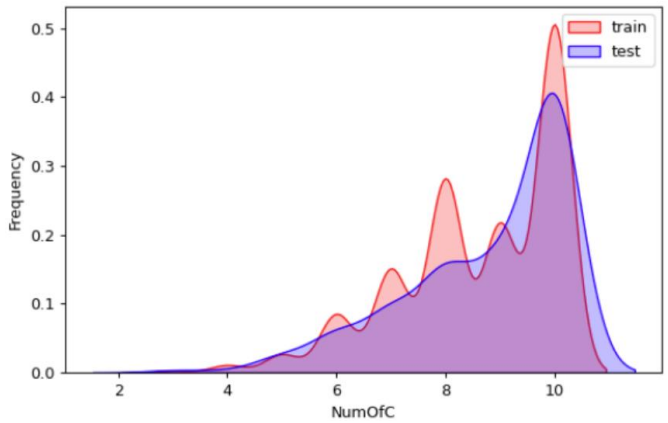

Figure 5

An example of inconsistent distribution is shown in figure 6. Still the feature ‘NumOfC’ but in the ‘None’ subset. It is considered to be inconsistent for 2 reasons:

1. *The ratio of value 7 and value 6 is close to 5:4 in the train set, and 14:5 in the test set.*
2. *{4,5,10} is observed in the train set, but not in the test set.*

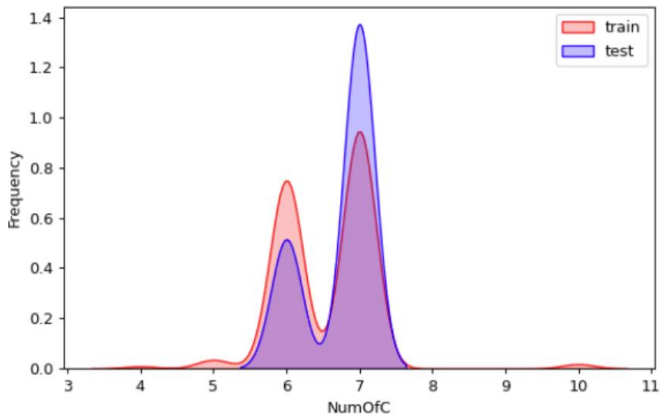

Figure 6 Distribution of NumOfC for 'None' subset

The correlation of 'NumOfC' and 'log10(pSat_Pa)' is 0.015, which is trivial as well. Therefore, 'NumOfC' is not considered a valid feature for 'None' subset and dropped.

3. Feature Selection

Still, we will take the 'decane' subset for example. We observed that some columns consist of all-0 values, 'nitroester' for example, shown in figure 7. This kind of feature does not help in prediction and introduces extra computational workload, so we removed all features with all-0 value in each subset.

	MW	NumOfAtoms	...	nitroester	pSat_Pa
count	2234.000000	2234.000000	...	2234.0	2234.000000
mean	254.497200	31.761862	...	0.0	-3.932335
std	45.287070	4.873193	...	0.0	1.964992
min	86.073165	14.000000	...	0.0	-10.382903
25%	223.105587	29.000000	...	0.0	-5.208076
50%	261.084852	32.000000	...	0.0	-3.979263
75%	291.059031	35.000000	...	0.0	-2.769820
max	338.059759	41.000000	...	0.0	3.591846

Figure 7

4. Outlier Detection

Our outlier detection function applies Z-statistic. It's noteworthy that we didn't remove outliers in the first iteration, because we detect outliers based on predictions made by the best performing model from previous runs.

In our project, we detect outliers in following steps:

i. Fit the training set to the best scored model obtained from previous run

ii. Make predictions given features on the training set, calculate residuals of prediction.

Iii. Calculate mean and standard deviation of residuals.

iv. Calculate Z-score of residuals.

v. Observations whose abs(Z-score) is larger than 4 times standard deviation will be considered as outliers.

We continuously select the model used in outlier detection based on following rules:

i. If the score is improved after the last run, we will apply the new model.

ii. If the score is not improved, we will use the previous model, but with different sigma, seeking to find a better outlier evaluation method.

Figure 8,9,10 displays the selected outliers from the 'toluene' subset from a run. It shows that our outlier detection function is working properly, by removing observations at the 'border' of the observation group. By configuring the value of '*sigma*', we are able to control the proportion of outliers.

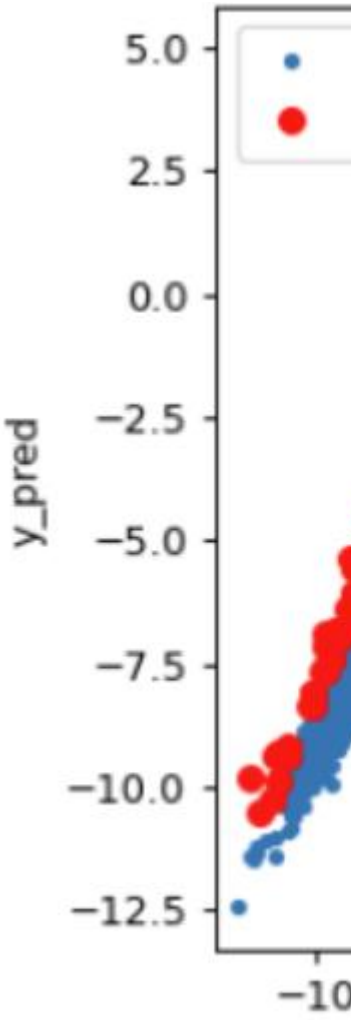

Figure 8

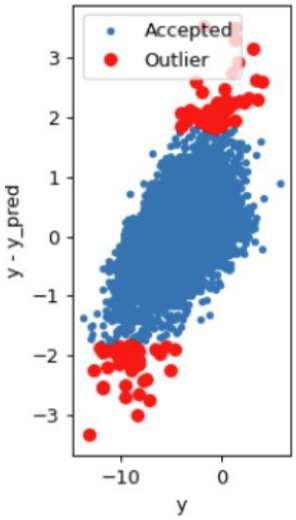

Figure 9

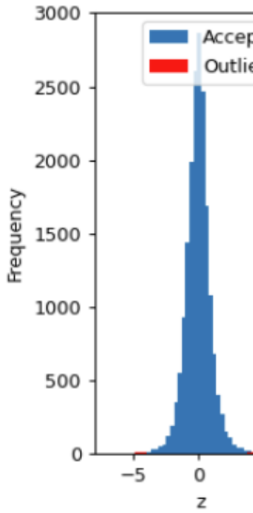

Figure 10

5. Scaling

Figure 11 displays our scaling method. We applied box-cox transformation to features firstly since we had observed normally distributed targets during EDA,

and we observed skewly distributed features. For example, figure 12 displays a skewly distributed feature 'NumOfConf' in the 'apin' subset, while the distribution of response 'pSat_Pa' of the 'apin' subset is close to normal distribution. Applying box-cox can stabilise variance and reduce skewness.

We then apply 0-mean standardisation and normalisation on the training set. We standardise the data since we don't want the difference in variance of features to affect the result of PCA, as larger variance contributes more to the result.
We applied normalisation to make the data more interpretable as we observed that all features are positive in the original dataset.

```
def boxcox(train,test):
  col_transform = test.columns
  for col in ['MW','NumOfAtoms','NumOfC','NumOfO','NumOfN','NumHBondDonors'
,'NumOfConf','NumOfConfUsed']:
    train.loc[:,col],_ = stats.boxcox(train.loc[:,col]+1)
    test.loc[:,col],_ = stats.boxcox(test.loc[:,col]+1)
  return [train,test]

def scale_z_score(train,test):
  def scale(col):
    return (col-np.mean(col))/np.std(col)
  scale_cols = [col for col in test.columns]
  train[scale_cols] = train[scale_cols].apply(scale,axis=0)
  test[scale_cols] = test[scale_cols].apply(scale,axis=0)
  return [train,test]

def scale_minmax(train,test):
  def scale(col):
    return (col-col.min())/(col.max()-col.min())
  scale_cols = [col for col in test.columns]
  train[scale_cols] = train[scale_cols].apply(scale,axis=0)
  test[scale_cols] = test[scale_cols].apply(scale,axis=0)
  return [train,test]
```

Figure 11

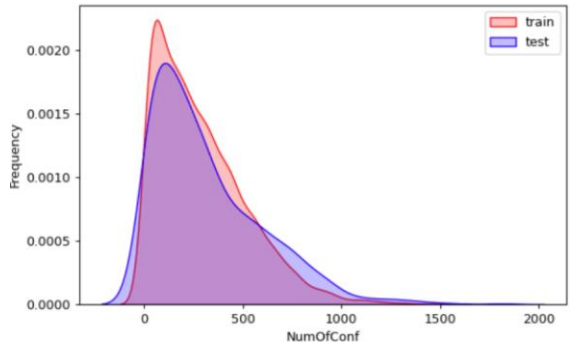

Figure 12

2.2 Dimension Reduction

1. PCA

For dimension reduction, we firstly applied PCA with the maximum expected number of components. We decided to capture 99% variance in predictors of each subset. The plot shown in figure 13 represents the result of PCA on the 'toluene' subset from an iteration. We observed that for the 'toluene' subset, 14

components will be able to explain 99% of variance in predictors, a significant reduction from 21 original features.

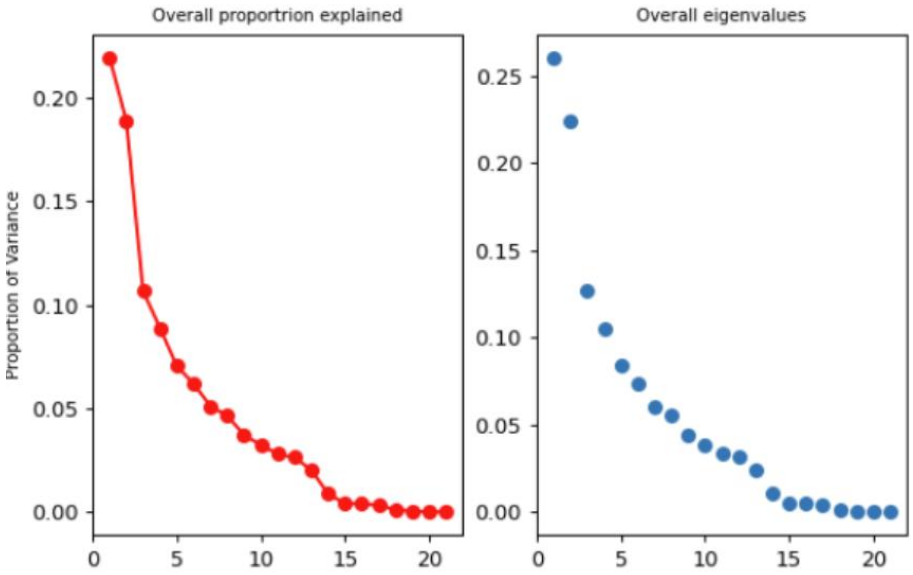

Figure 13

2.3 Model Training

1. Model Selection

As mentioned in section 2.3, we took several models into consideration, including Linear Regression, Random Forest, and Support Vector Machine.

We selected Linear Regression as it's simple, efficient, and suitable for a large dataset. At the same time, coefficients derived from linear models are interpretable.

We selected Random Forest since it naturally reduces overfitting and improves accuracy through the combination of multiple decision trees. At the same time, RF is suitable for large datasets.

We selected SVM since it works well in high-dimensional spaces. At the same time, we could try out different kernels to capture both linear and non-linear relationships in the data. For example, using kernels 'linear' and 'poly' to capture linear relationships, 'rbf' and 'sigmoid' for non-linear complex relationships.

2. Hyperparameter Tuning

As mentioned in section 2.3, we applied GridSearchCV for hyperparameter tuning, as it significantly increased the efficiency in finding the best performing parameter group for a model.

Figure 14 displays the code of using GridSearchCV in the model training function.

```
def train_model(model, param_grid=[], X=[], y=[], splits=5, repeats=2):
  rkfold = RepeatedKFold(n_splits=splits, n_repeats=repeats)
  gsearch = GridSearchCV(model, param_grid, cv=rkfold,
                         scoring=r2_scorer,
                         verbose=2, return_train_score=True)
  gsearch.fit(X,y)
  model = gsearch.best_estimator_
  best_idx = gsearch.best_index_
  grid_results = pd.DataFrame(gsearch.cv_results_)
```

Figure 14

Figure 15 displays an example of transferring a model and its parameter group into the function. The 'param_grid' is a set of key-value pairs, where the key is the name of the parameter and value is a list of all values of that parameter. Each unique combination of parameters will be trained twice, each time through a 10-fold cross-validation.

```
  svr = LinearSVR()
  param_grid1 = {
    'C': [0.1,0.2,0.3,0.5,1],  # Regularization parameter
    'max_iter': [1500,2000,2500], # Maximum number of iterations
    'tol': [0.05,0.1,0.2,0.5,1] # Tolerance for stopping criteria
  }
  y_prediction = train_model(svr,param_grid=param_grid1,X=X,y=y,splits=10, repeats
=2)
```

Figure 15

The result of the training is shown in figure 16. For each parameter group, the best performing one will be selected based on the overall R2 score.

```
----------------------
LinearSVR(C=0.5, max_iter=1500, tol=0.2)
R2score= 0.7910859030996735
cross_val: mean= 0.5047081137026963 , std= 0.4960534285746889
```

Figure 16

3. Model Comparison

For each subset, we compared the R2 score for each model, and recorded them in each iteration. The process of finding the best-performing model with an optimised parameter group is repeated until a certain combination of parameters is found and proved to be better than others. We will skip the finding process in the final report and jump directly to the result:

Dataset	Optimal Model	Optimal Parameter Group	R2 Score
'Toluene'	Random Forest	n_estimator=500, max_features=9, min_sample_split=11, max_depth=21, min_sample_leaf=2	0.833
'None'	Random Forest	n_estimators=500, max_depth=25, max_features=12, min_samples_leaf=2, min_samples_split=5,	0.898
'apin'	Random Forest	n_estimator=650, max_features=6, min_sample_split=14, max_depth=25, min_sample_leaf=2	0.852

‘decane’	LinearSVR	C=0.5, max_iter=500, tol=0.05	0.804
‘apin_toluene’	LinearSVR	C=0.3, max_iter=500, tol=0.05	0.740
‘apin_decane’	LinearSVR	C=0.5, max_iter=2500, tol=0.05	0.816
‘apin_decane_toluene’	SVR	kernel=linear	0.930
All	LinearSVR	C=0.1, max_iter=1500, tol=0.05	0.701

2.4 Prediction

We made the final prediction by linearly combining results derived from each subset with results from the model trained on overall observations. Table below shows the weight of different ‘parent_species’. We defined weight according to the size of each subset, in order to avoid overfitting and bias. For example, the size of the ‘toluene’ subset is really large with 20000+ observations, so we gave large weight to results derived from the ‘toluene’ subset, and small weight to the results predicted by the general model. A counter-example is how we obtain the results for ‘apin_decane_toluene’, whose subset size is really small.

Final prediction	Species_Weight	Overall_Weight
‘Toluene’	0.7	0.3
‘None’	0.5	0.5
‘apin’	0.7	0.3
‘decane’	0.4	0.6
‘apin_toluene’	0.3	0.7
‘apin_decane’	0.3	0.7
‘apin_decane_toluene’	0.2	0.8

4.2 Discussion

1. **Analysis**

After discussion, together we found some possible methods to improve the performance of our model.

I. Taking more models into consideration. For example, ElasticNet or Boosting.
II. Adding an interaction term. Since there are 30 features in the original data set, and we lack awareness of the correlations between different features, we haven't included interaction terms in this project.
III. We took an iterative method in outlier detection, but we did not notice that the action of removing outliers itself had an impact on following work. This is like a one-way channel, where the final result relies on all previous steps. However, we can't promise that all previous runs are optimised. Therefore, we should add a control group which does not remove any observations.
IV. During PCA, we forced 99% variance to be captured, which may include noise or specificities of the training data as well.