# 3011979 Practical Python for Data Sciences and Machine Learning

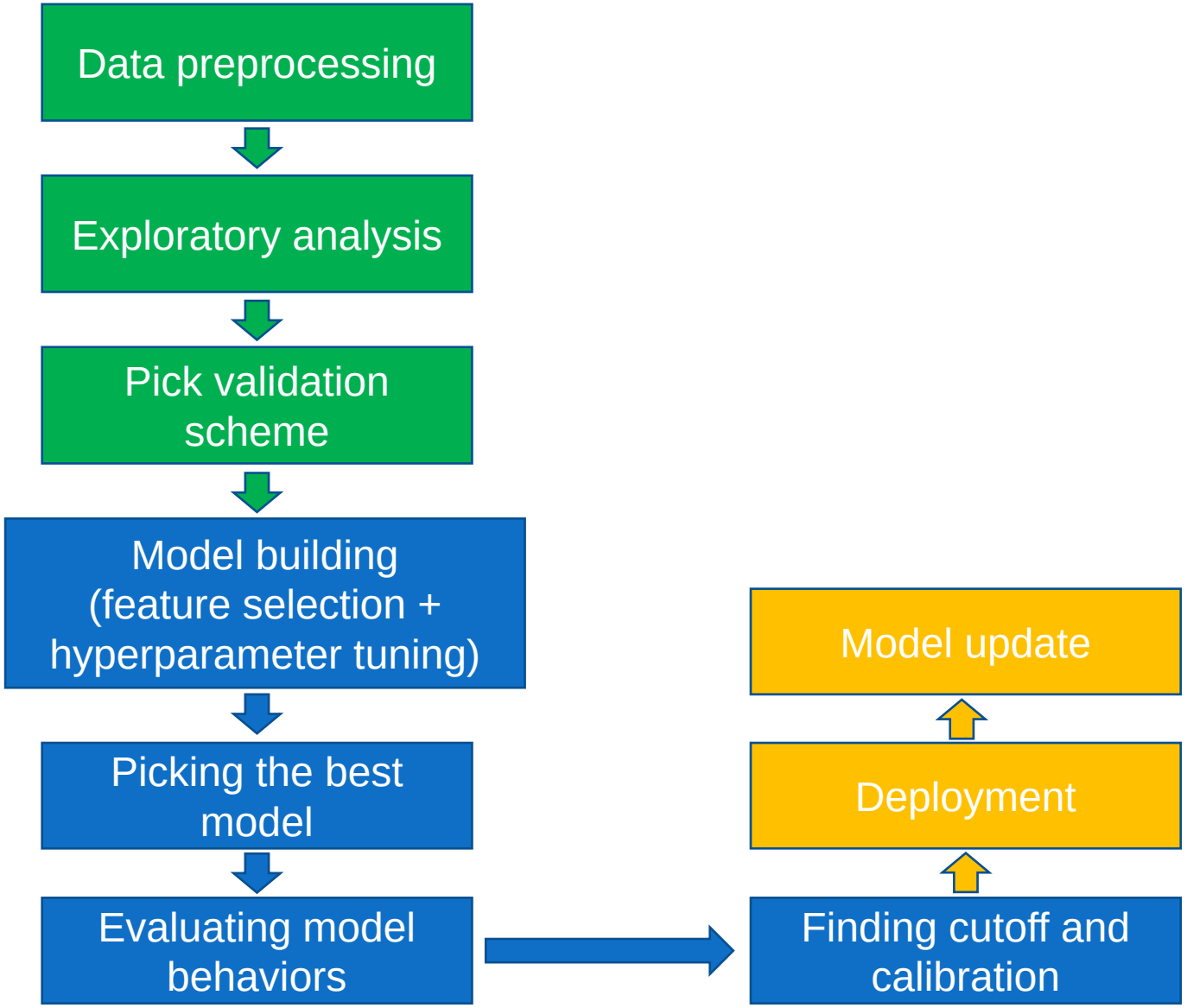
## L10: ML best practices

Mar 25th, 2022



Sira Sriswasdi, Ph.D.
Research Affairs, Faculty of Medicine
Chulalongkorn University

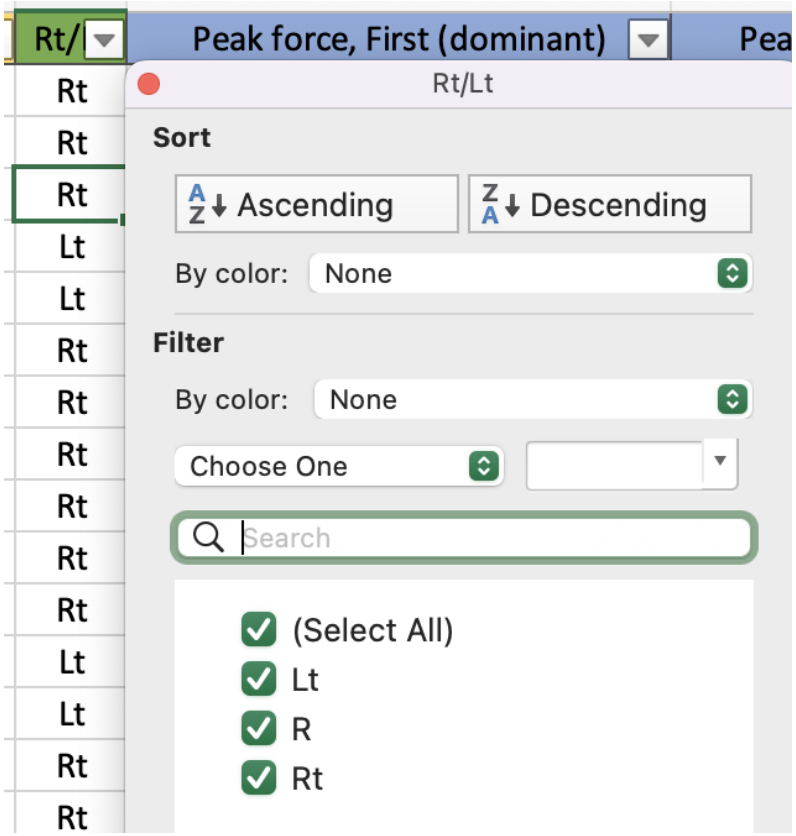
# ML development phases



# Data preprocessing

# Checking with Excel

| patient_id | age | sex | pvt_main | surg |
|---|---|---|---|---|
| N001 | 50 | 0 | 0 | 0 |
| N002 | 58 | 1 | 0 | 0 |
| N003 | 62 | 1 | 0 | 1 |
| N004 | 59 | 1 | 0 | 0 |
| N005 | 82 | 0 | NA | 0 |
| N006 | 60 | 1 | | 0 |
| N007 | 58 | 1 | 0 | 0 |




- Generate sample ID system
- Check how missing data are handled
- Check for typo in categorical variables

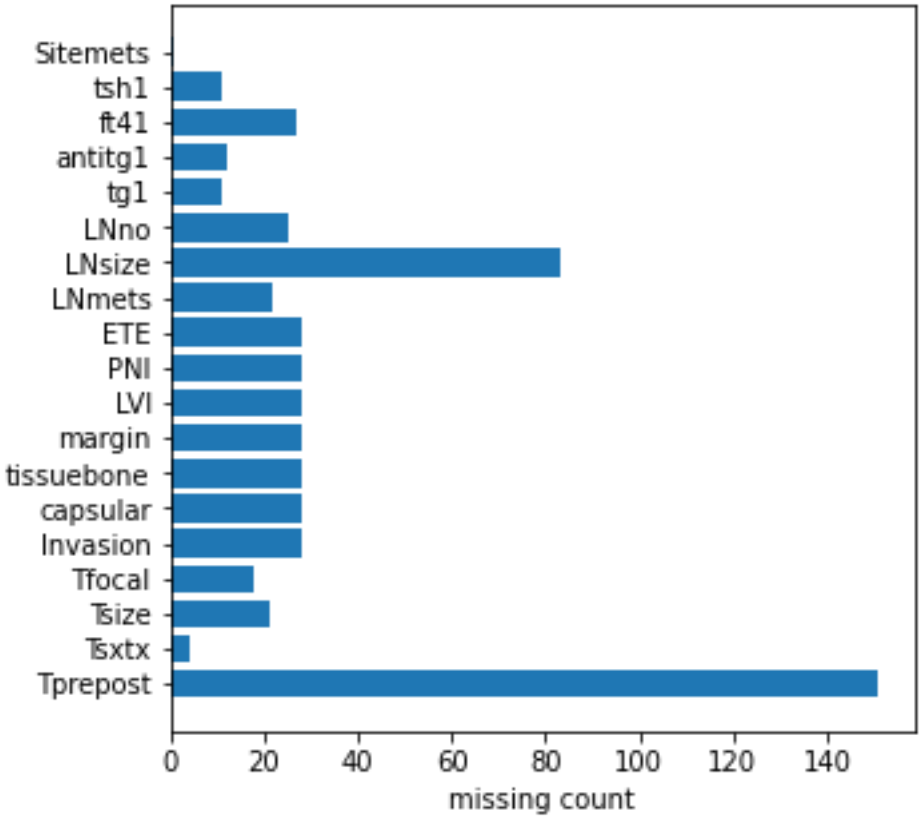
# Pandas's missing data behavior

**na_values** : *scalar, str, list-like, or dict, default None*

Additional strings to recognize as NA/NaN. If dict passed, specific per-column NA values. By default the following values are interpreted as NaN: '', '#N/A', '#N/A N/A', '#NA', '-1.#IND', '-1.#QNAN', '-NaN', '-nan', '1.#IND', '1.#QNAN', '<NA>', 'N/A', 'NA', 'NULL', 'NaN', 'n/a', 'nan', 'null'.

- Putting other text, such as "missing", in a column of numerical variable would make Pandas incorrectly treat that column as a string variable instead

# Determine the extent of missing data




- Variables with too many missing values have to be **dropped**
- Variables with few missing values may be **imputed**

# Ways to impute missing data

## sklearn.impute.SimpleImputer

*class* sklearn.impute.SimpleImputer(*, *missing_values=nan*, *strategy='mean'*, *fill_value=None*, *verbose=0*, *copy=True*, *add_indicator=False*) [source]

**strategy : *str, default='mean'***

The imputation strategy.

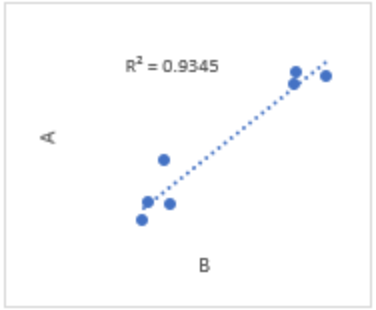
- If "mean", then replace missing values using the mean along each column. Can only be used with numeric data.
- If "median", then replace missing values using the median along each column. Can only be used with numeric data.
- If "most_frequent", then replace missing using the most frequent value along each column. Can be used with strings or numeric data. If there is more than one such value, only the smallest is returned.
- If "constant", then replace missing values with fill_value. Can be used with strings or numeric data.

- **Mean** or **median** for numerical variables (inspect distribution)
- **Mode** for categorical variables
- Leave blank (let XGBoost model learns)

# Multivariate imputation

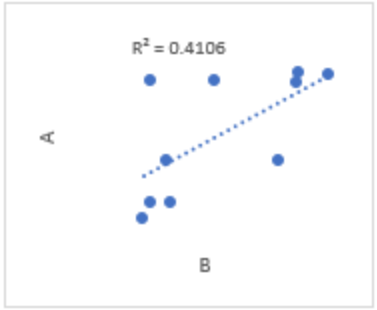
Missing data is in red. There is a strong correlation between A and B, so let's try to impute A using B and C.

| A | B | C |
|---|---|---|
| 0.93 | 1.40 | 1.53 |
| 0.24 | 0.46 | 0.76 |
| | 0.80 | |
| 0.95 | 1.24 | 1.46 |
| 0.23 | 0.57 | |
| 0.90 | | 1.28 |
| 0.15 | 0.42 | |
| 0.47 | 0.54 | 0.63 |
| | 1.14 | |
| 0.89 | 1.23 | 1.45 |



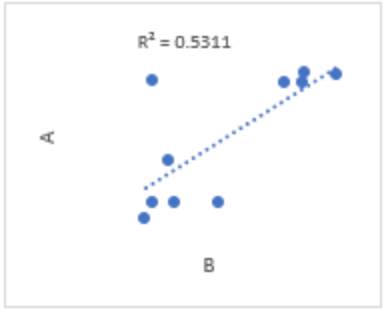

Missing data is filled in randomly. This dillutes the correlations, but allows us to impute using all available data.

| A | B | C |
|---|---|---|
| 0.93 | 1.40 | 1.53 |
| 0.24 | 0.46 | 0.76 |
| 0.90 | 0.80 | 1.53 |
| 0.95 | 1.24 | 1.46 |
| 0.23 | 0.57 | 1.28 |
| 0.90 | 0.46 | 1.28 |
| 0.15 | 0.42 | 1.53 |
| 0.47 | 0.54 | 0.63 |
| 0.47 | 1.14 | 1.28 |
| 0.89 | 1.23 | 1.45 |



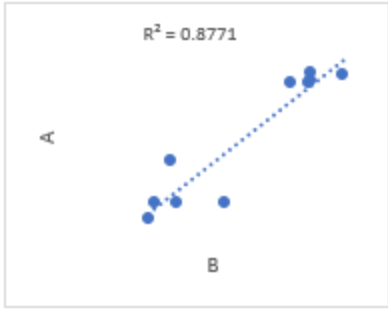

A random forest is used to predict A with B and C. Notice the correlation between A and B improved.

| A | B | C |
|---|---|---|
| 0.93 | 1.40 | 1.53 |
| 0.24 | 0.46 | 0.76 |
| 0.24 | 0.80 | 1.53 |
| 0.95 | 1.24 | 1.46 |
| 0.23 | 0.57 | 1.28 |
| 0.90 | 0.46 | 1.28 |
| 0.15 | 0.42 | 1.53 |
| 0.47 | 0.54 | 0.63 |
| 0.89 | 1.14 | 1.28 |
| 0.89 | 1.23 | 1.45 |




After Imputing B using A and C, we have achieved a correlation between A and B much closer to the original data.

| A | B | C |
|---|---|---|
| 0.93 | 1.40 | 1.53 |
| 0.24 | 0.46 | 0.76 |
| 0.24 | 0.80 | 1.53 |
| 0.95 | 1.24 | 1.46 |
| 0.23 | 0.57 | 1.28 |
| 0.90 | 1.24 | 1.28 |
| 0.15 | 0.42 | 1.53 |
| 0.47 | 0.54 | 0.63 |
| 0.89 | 1.14 | 1.28 |
| 0.89 | 1.23 | 1.45 |




- Treat target variable as the output
- Use the remaining variables as input
- Train machine learning models to predict the missing values

# Predictive mean matching

The predicted value of A (E[A|B,C]) is shown to the left. We are interested in imputing the bold missing value below

| E[A\|B,C] | A | B | C |
|---|---|---|---|
| 0.73 | 0.93 | 1.40 | 1.53 |
| 0.62 | 0.24 | 0.46 | 0.76 |
| 0.60 | | 0.80 | 1.53 |
| 1.39 | 0.95 | 1.24 | 1.46 |
| 0.36 | 0.23 | 0.57 | 1.28 |
| 1.27 | 0.90 | 0.46 | 1.28 |
| 0.15 | 0.15 | 0.42 | 1.53 |
| 0.65 | 0.47 | 0.54 | 0.63 |
| 1.20 | | 1.14 | 1.28 |
| 1.24 | 0.89 | 1.23 | 1.45 |

Our predicted value for the first missing sample is 0.60. The closest predicted value is 0.62. We find the closest values for all of our missing samples.

| E[A\|B,C] | A | B | C |
|---|---|---|---|
| 0.73 | 0.93 | 1.40 | 1.53 |
| 0.62 | 0.24 | 0.46 | 0.76 |
| 0.60 | | 0.80 | 1.53 |
| 1.39 | 0.95 | 1.24 | 1.46 |
| 0.36 | 0.23 | 0.57 | 1.28 |
| 1.27 | 0.90 | 0.46 | 1.28 |
| 0.15 | 0.15 | 0.42 | 1.53 |
| 0.65 | 0.47 | 0.54 | 0.63 |
| 1.20 | | 1.14 | 1.28 |
| 1.24 | 0.89 | 1.23 | 1.45 |

We then impute the value corresponding to the original data.

| E[A\|B,C] | A | B | C |
|---|---|---|---|
| 0.73 | 0.93 | 1.40 | 1.53 |
| 0.62 | 0.24 | 0.46 | 0.76 |
| 0.60 | 0.24 | 0.80 | 1.53 |
| 1.39 | 0.95 | 1.24 | 1.46 |
| 0.36 | 0.23 | 0.57 | 1.28 |
| 1.27 | 0.90 | 0.46 | 1.28 |
| 0.15 | 0.15 | 0.42 | 1.53 |
| 0.65 | 0.47 | 0.54 | 0.63 |
| 1.20 | 0.89 | 1.14 | 1.28 |
| 1.24 | 0.89 | 1.23 | 1.45 |

- Use predicted values to identify observed data points with similar predictions
- Impute with observed values from other data points

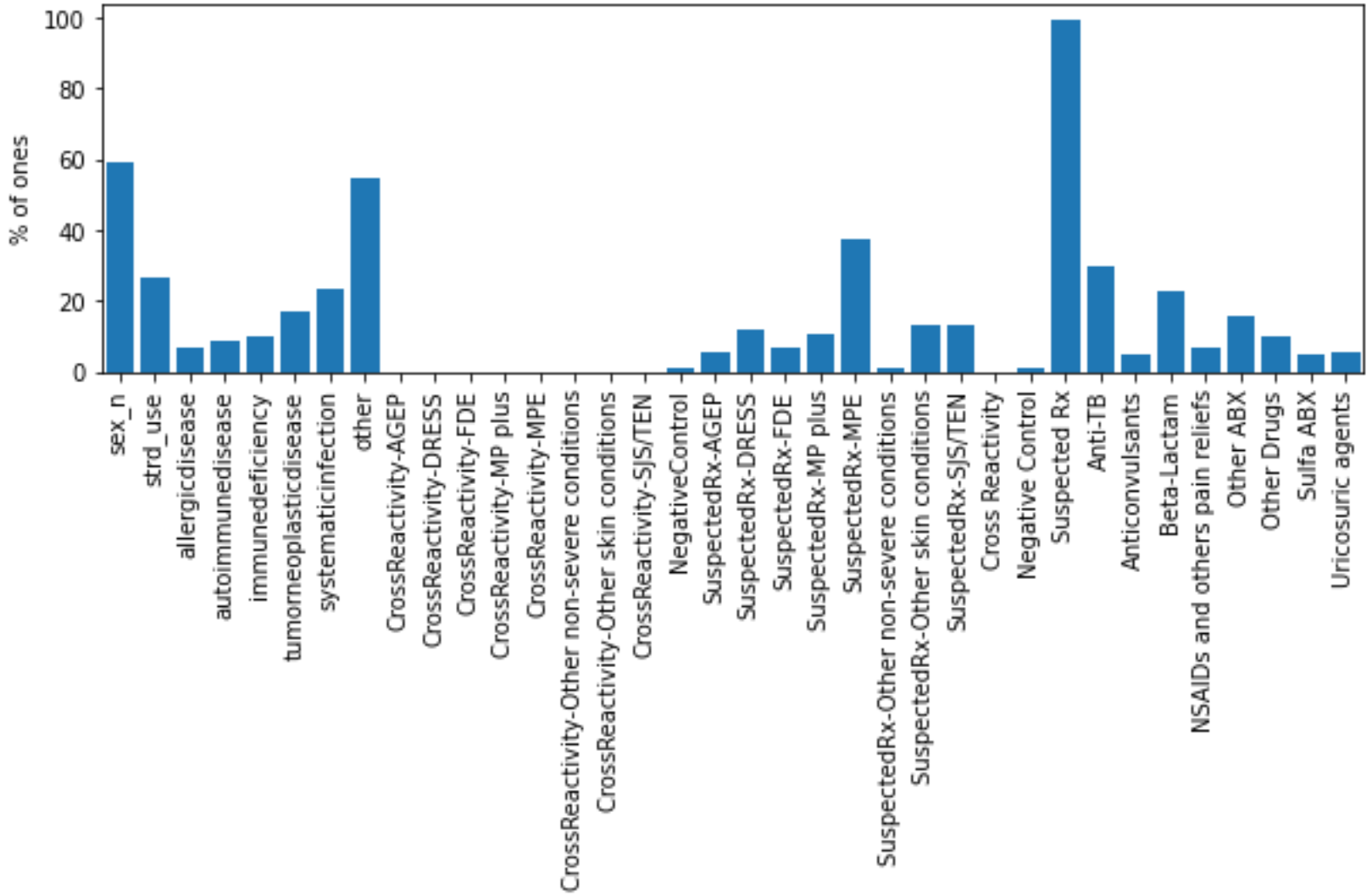
# Transform categorical and ordinal variables

| Sitemets | Sitemets:1 | Sitemets:2 | Sitemets:3 | Sitemets:4 |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 |

| Severity | normal to mild | mild to moderate | moderate to severe |
|---|---|---|---|
| normal | 0 | 0 | 0 |
| mild | 1 | 0 | 0 |
| moderate | 1 | 1 | 0 |
| severe | 1 | 1 | 1 |

- Each category is independent from another
- Ordinal variables have order
  - Tree model can use as is, linear model needs human help

# Some binary variables may be dropped




- Binary variables that are mostly 0 or 1 are uninformative

# Feature transformation can help

## sklearn.preprocessing.PowerTransformer

*class* sklearn.preprocessing.PowerTransformer(*method='yeo-johnson', *, standardize=True, copy=True*) [source]

Apply a power transform featurewise to make data more Gaussian-like.

Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like. This is useful for modeling issues related to heteroscedasticity (non-constant variance), or other situations where normality is desired.

Currently, PowerTransformer supports the Box-Cox transform and the Yeo-Johnson transform. The optimal parameter for stabilizing variance and minimizing skewness is estimated through maximum likelihood.

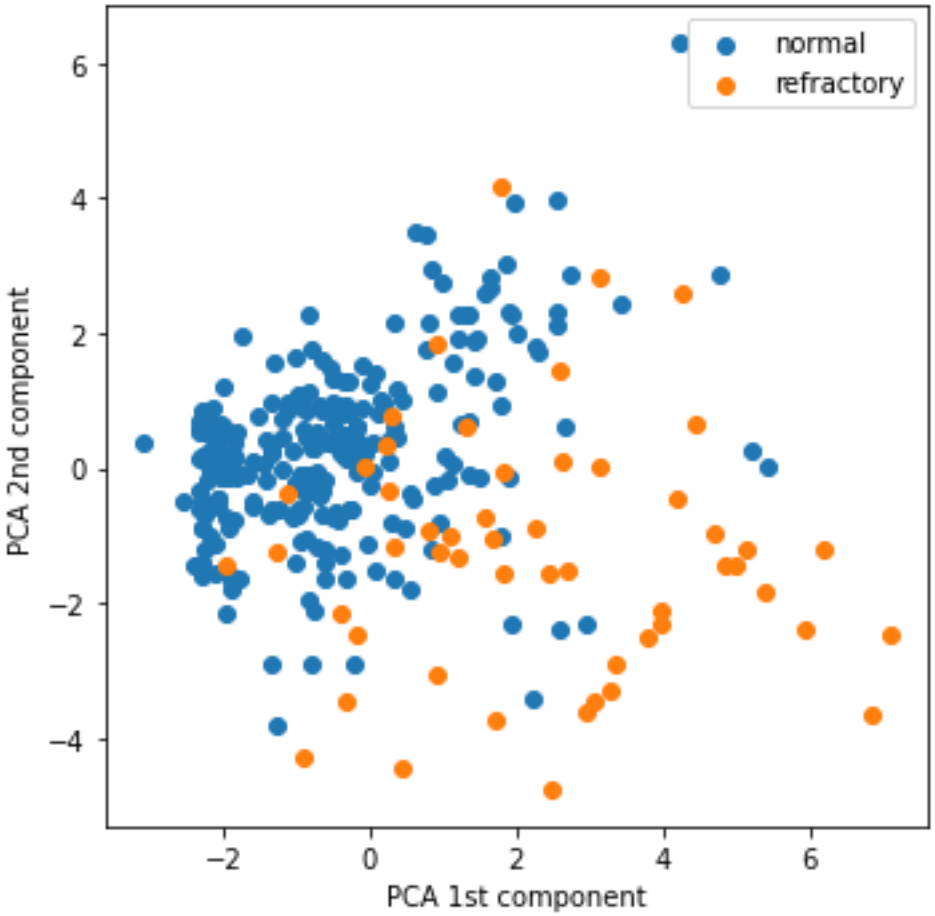
Box-Cox requires input data to be strictly positive, while Yeo-Johnson supports both positive or negative data.

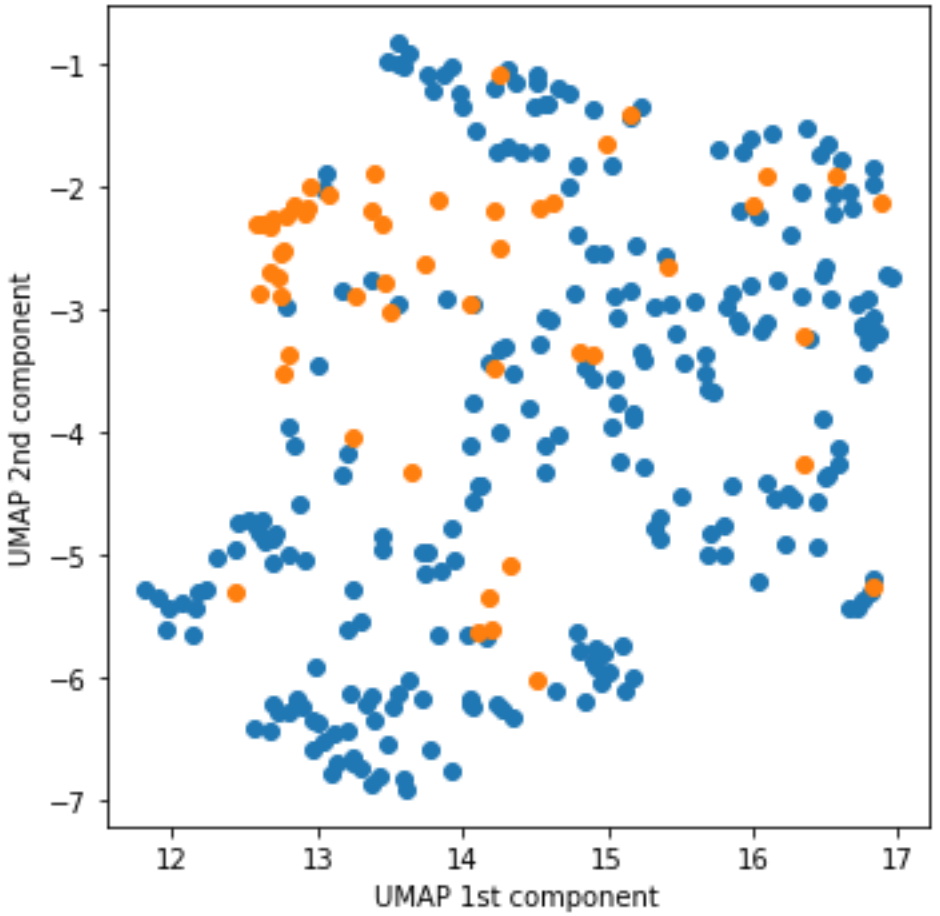
By default, zero-mean, unit-variance normalization is applied to the transformed data.

- A simple log-transform or sqrt-transform can reduce the impact of outliers on linear models
- Check count or intensity data

# Exploratory data analysis

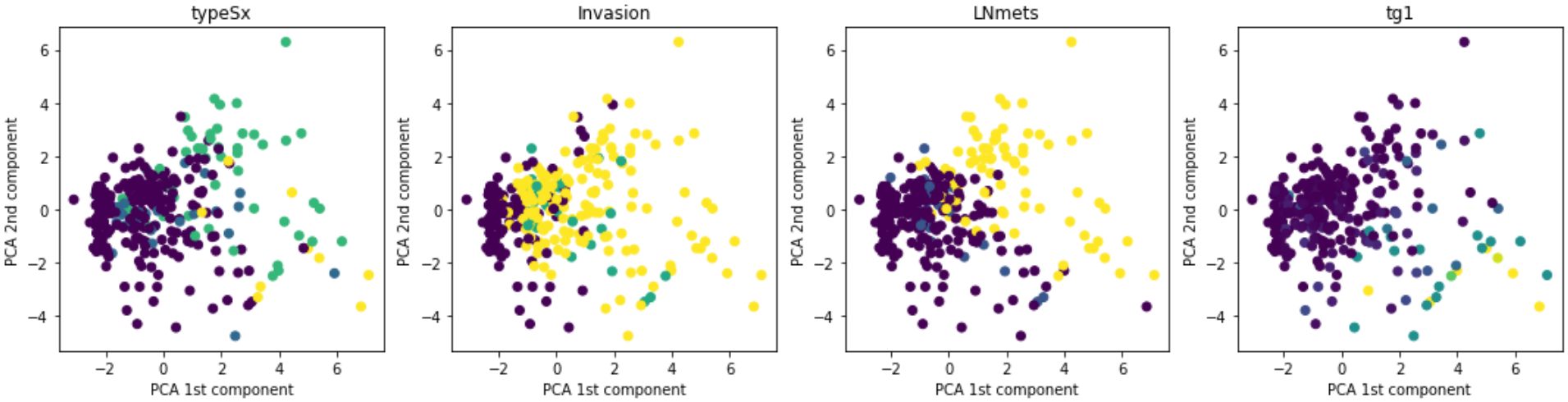
# Visualize sample distribution




- Separation between classes
- Look for outliers and subgroups

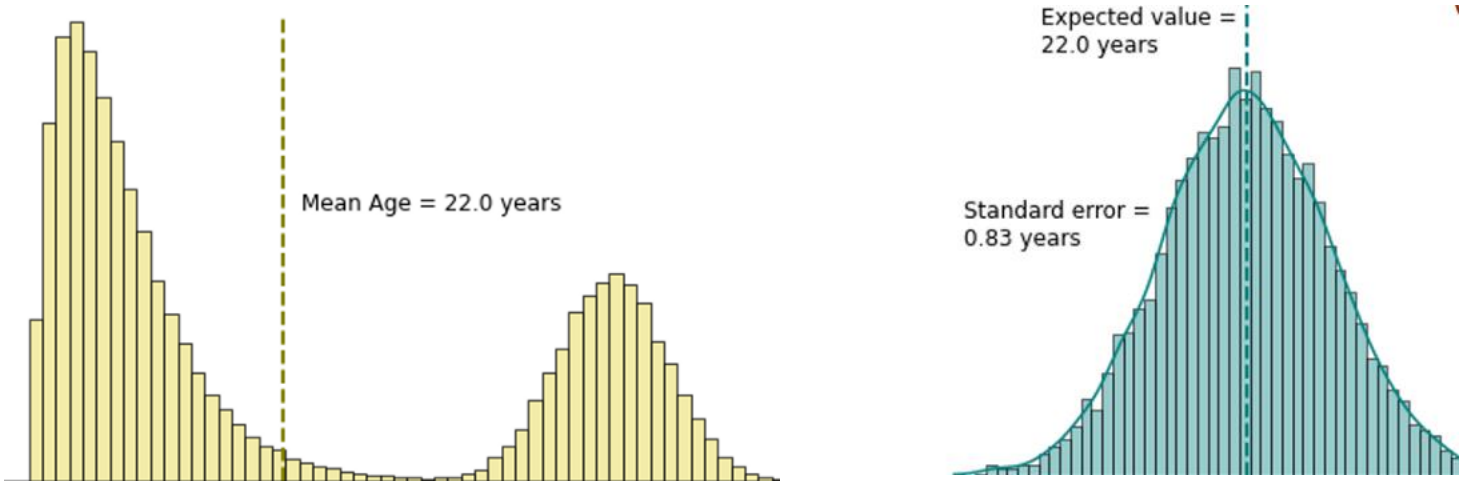
# Identify features that correlate with label




- Visually = PCA loadings
- Statistically
  - Correlation
  - Mann-Whitney U test
  - ANOVA
  - Fisher's exact test
  - Chi-squared test

| | Count | Fisher P-value |
|---|---|---|
| **Anti-TB** | 37.0 | 0.640711 |
| **Anticonvulsants** | 6.0 | 0.618221 |
| **Beta-Lactam** | 28.0 | 0.611501 |
| **NSAIDs and others pain reliefs** | 8.0 | 0.014954 |
| **Other ABX** | 19.0 | 1.000000 |
| **Other Drugs** | 12.0 | 0.732384 |
| **Sulfa ABX** | 6.0 | 0.618221 |
| **Uricosuric agents** | 7.0 | 1.000000 |

# Always use visualization with statistics



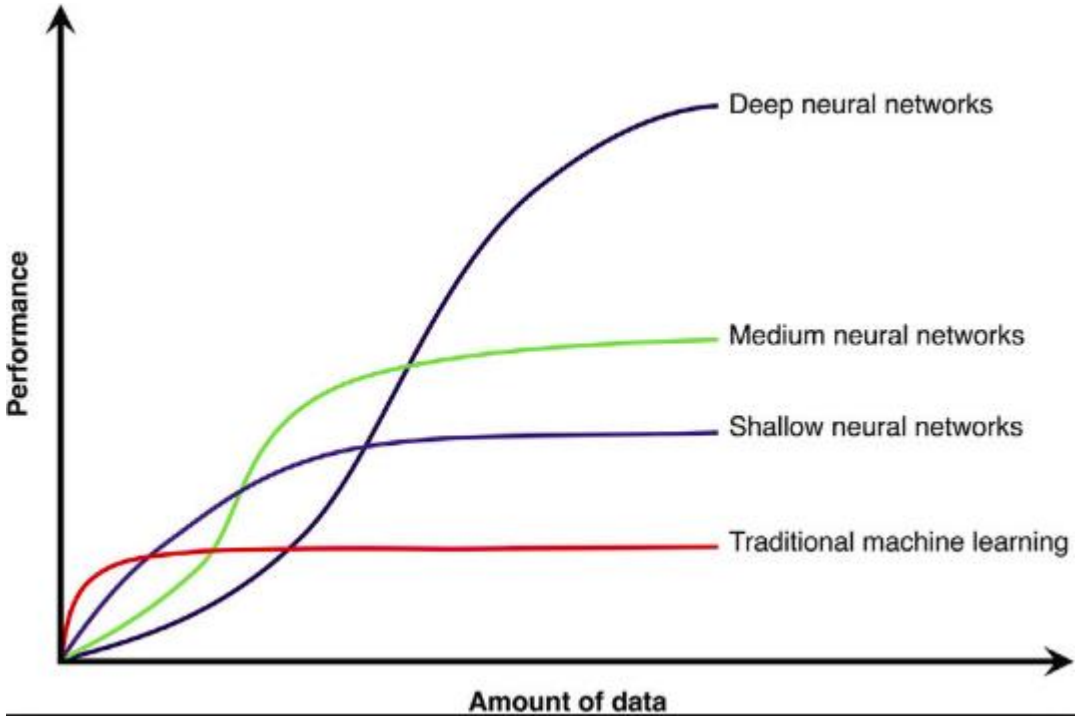

- Statistical test of means would fail to detect the difference between a bimodal distribution and a unimodal distribution with similar mean
- Check variable statistics (such as mean and SD) together with test results
  - SD of bimodal distribution would be much higher

# Pick validation scheme

# Recap roles of train-val-test

- Training set = for helping the model find the right coefficients, branching conditions, etc.
  - Only need to match the complexity of the task and variation in data
  - Small training set can be ok – **underfit risk**
- Validation set = for selecting the best model
  - Small validation set can be ok in cross-validation scheme
  - Small validation set can lead to **overfit risk**
- Test set = for representing actual usage situation
  - Must capture the distribution of data that the model will face
  - Small test set is **not** ok

# You may not need all those training data




- Classical models are not data-hungry
- A quick test can be performed on a fixed, sizeable test set + bootstrapping of the training set

# Validation scheme choices

**cv : *int, cross-validation generator or an iterable, default=None***

Determines the cross-validation splitting strategy. Possible inputs for cv are:

- None, to use the default 5-fold cross validation,
- integer, to specify the number of folds in a `(Stratified)KFold`,
- CV splitter,
- An iterable yielding (train, test) splits as arrays of indices.

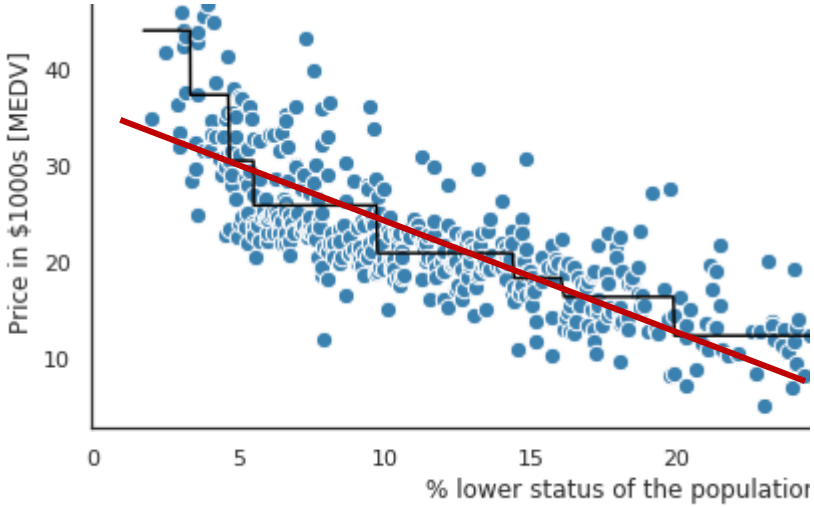
For integer/None inputs, if the estimator is a classifier and `y` is either binary or multiclass, `StratifiedKFold` is used. In all other cases, `KFold` is used. These splitters are instantiated with `shuffle=False` so the splits will be the same across calls.

Refer User Guide for the various cross-validation strategies that can be used here.

- Test or no test?
  - No test is better than small test
- Cross-validation or bootstrapping
  - Statistically the same
  - Both can be integrated with GridSearchCV
  - Bootstrap is useful when you need large validation

# Model building

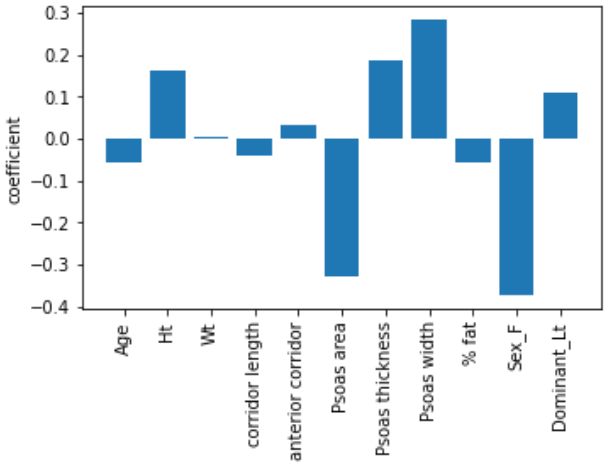
# Try several different model families



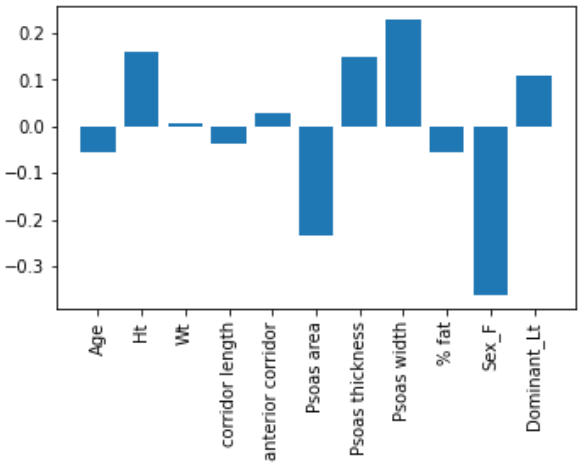

- Linear model
  - Try both Ridge and Lasso
  - Tune regularization strength a bit
- Support vector machine
  - Usually not needed, try RBF kernel if you have a lot of data
- Tree model
  - An untuned Random Forest with plenty of trees is typically ok
  - Tune *max_feature* based on your data

# Tuning regularization strength

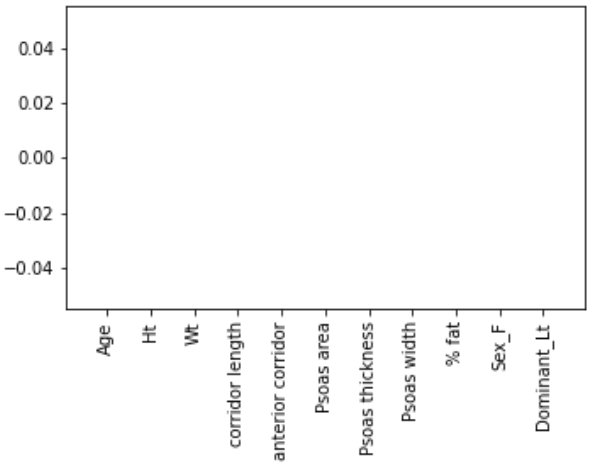
Unpenalized model




Ridge model

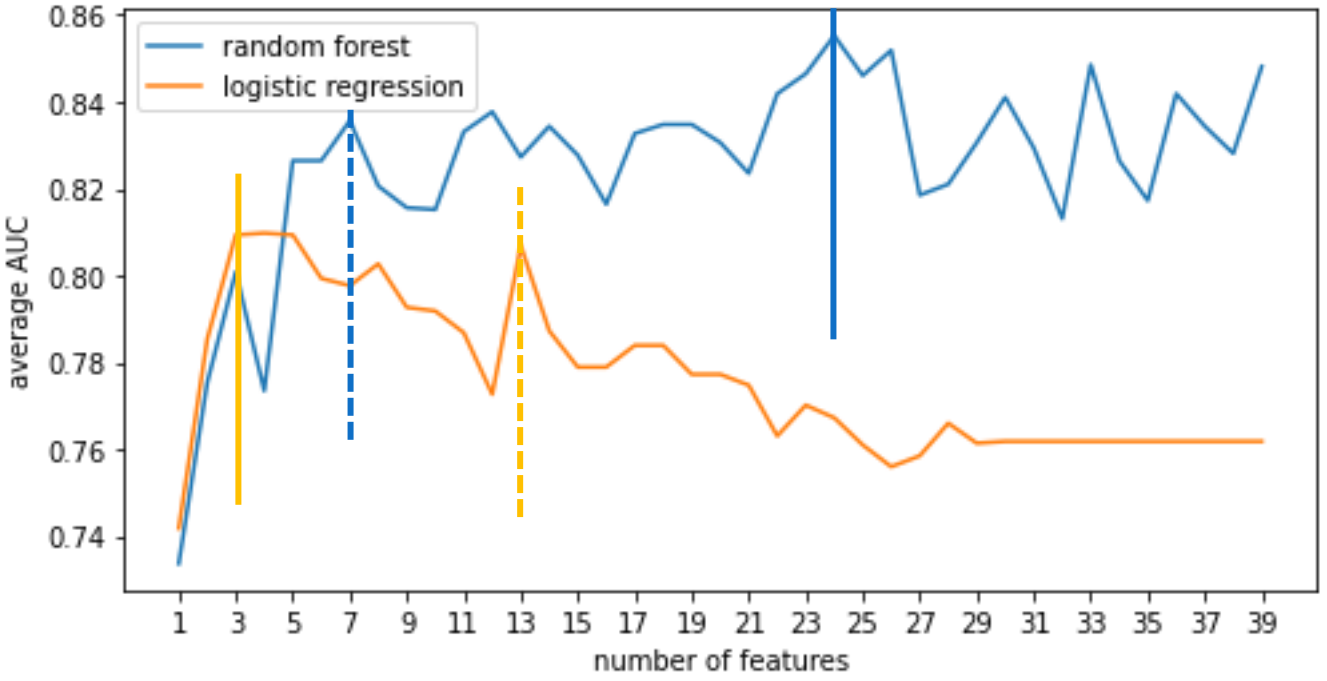



Lasso model




- Lasso pushes coefficients to zero more strongly
- The strength C is specific to each dataset

# Feature selection




- Backward (recursive) feature elimination is recommended
- For radiomics with 100-1000 features, speed up with univariate statistics and large step size
  - Then perform *step* = 1 feature in another round if needed

# sklearn's RFECV

## sklearn.feature_selection.RFECV

*class* sklearn.feature_selection.RFECV(*estimator*, ***, *step=1*, *min_features_to_select=1*, *cv=None*, *scoring=None*, *verbose=0*, *n_jobs=None*, *importance_getter='auto'*) [source]

Recursive feature elimination with cross-validation to select the number of features.

See glossary entry for cross-validation estimator.

Read more in the User Guide.

**Parameters:**

**n_jobs : *int, default=None***

Number of jobs to run in parallel. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

**step : *int or float, default=1***

If greater than or equal to 1, then `step` corresponds to the (integer) number of features to remove at each iteration. If within (0.0, 1.0), then `step` corresponds to the percentage (rounded down) of features to remove at each iteration. Note that the last iteration may remove fewer than `step` features in order to reach `min_features_to_select`.

- Increase *step* for dataset with many features

# Full model tuning

## Use GridSearchCV to tune Random Forest models

Choices of metrics: https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

```
base_rf = RandomForestClassifier(n_estimators = 500, random_state = 3011979)
param_grid = {'max_features': [0.3, 0.6], ## fraction of features considered
              'max_depth': [10, None], ## tree depth
              'min_samples_split': [2, 10]} ## minimum number of samples at each split

grid_rf = GridSearchCV(estimator = base_rf, param_grid = param_grid,
                       scoring = ['accuracy', 'f1_weighted', 'roc_auc_ovr_weighted'],
                       refit = False, cv = 5)
```

```
grid_rf.fit(X_train_std, y_train)
```

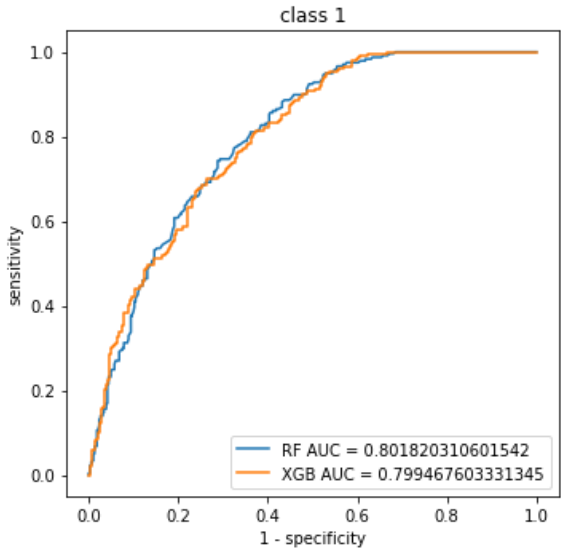
**n_jobs : *int, default=None***

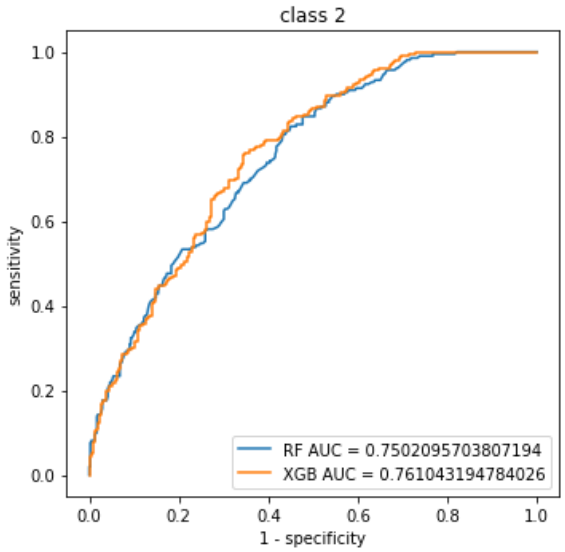
Number of jobs to run in parallel. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

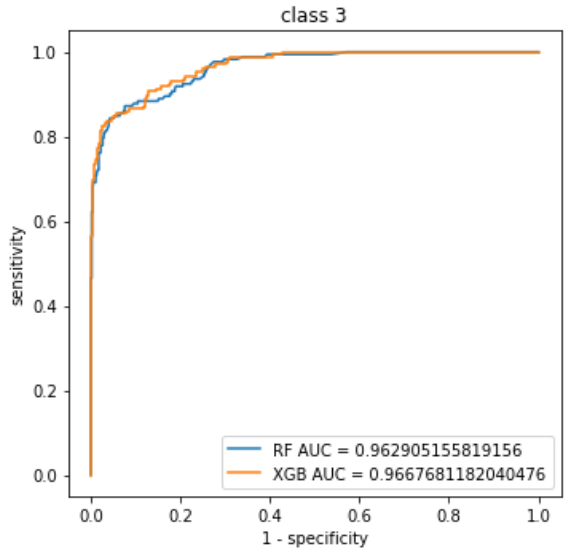
- This is the most time-consuming part of the code
- Increase *n_jobs* in GridSearchCV to speed up
- Include multiple performance metrics

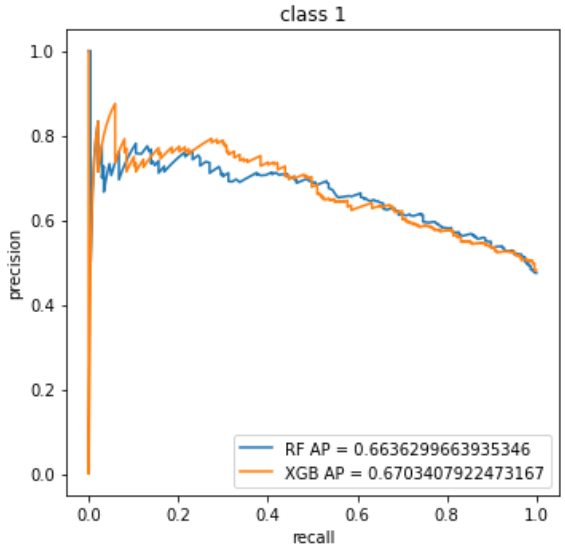
# Finding the best models

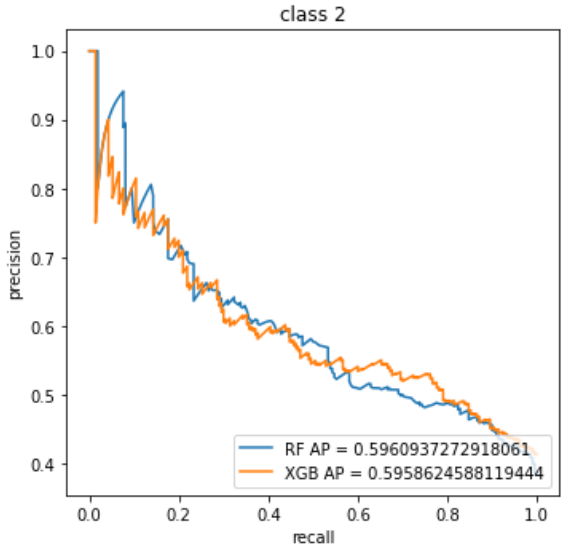
# Starts with ROC and PRC

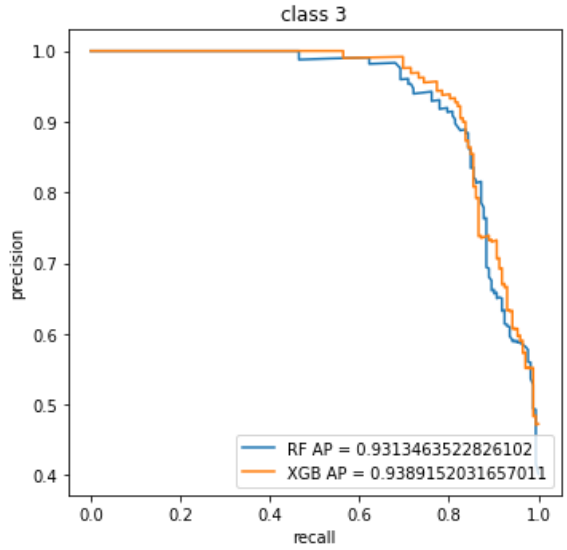


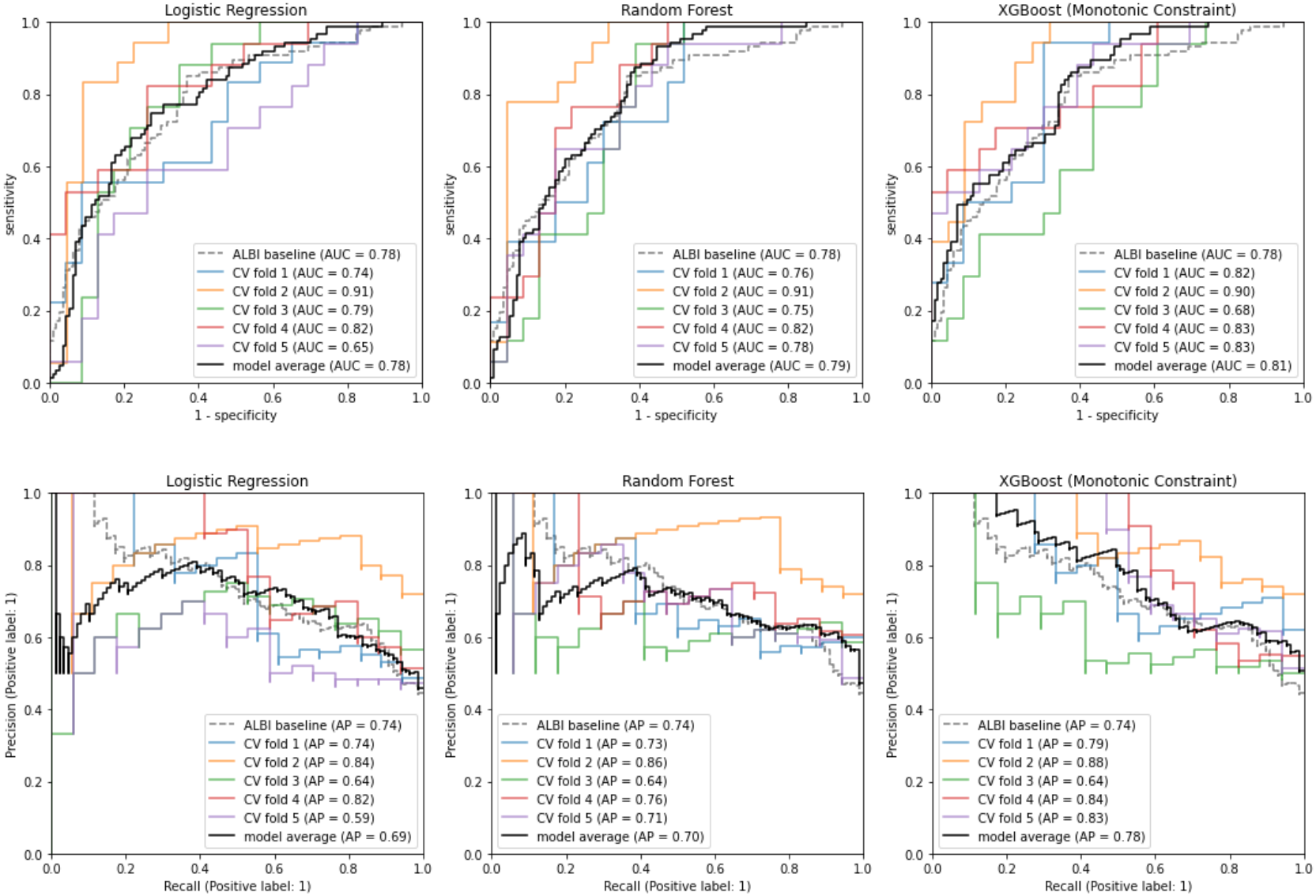



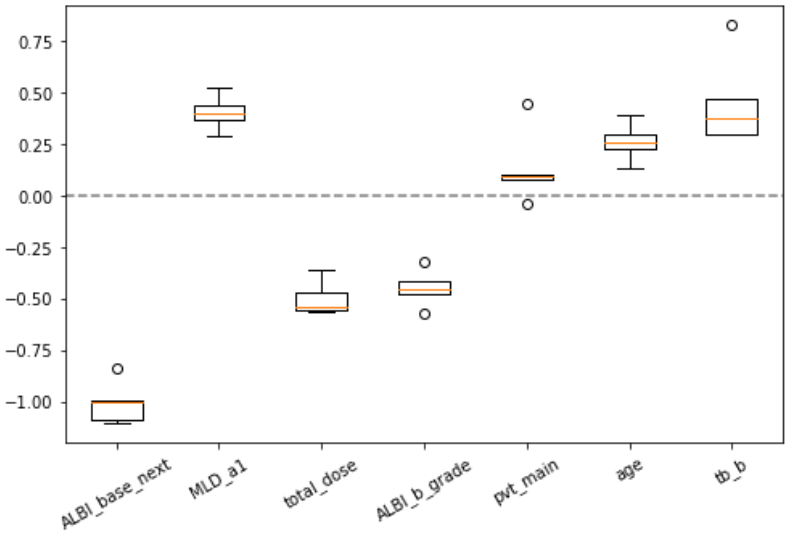
# Compare across CV folds



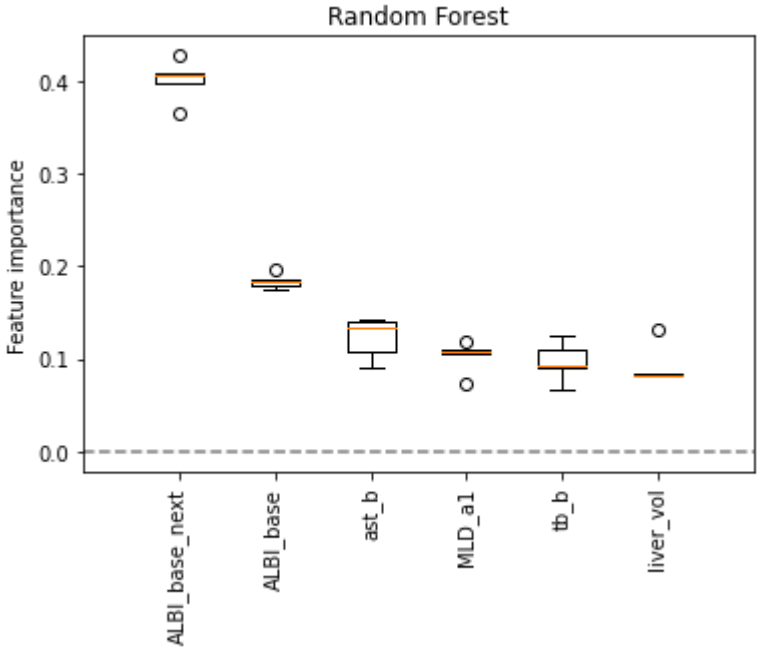
# Explaining model behaviors

# Feature importance – when included

Ridge model coefficients

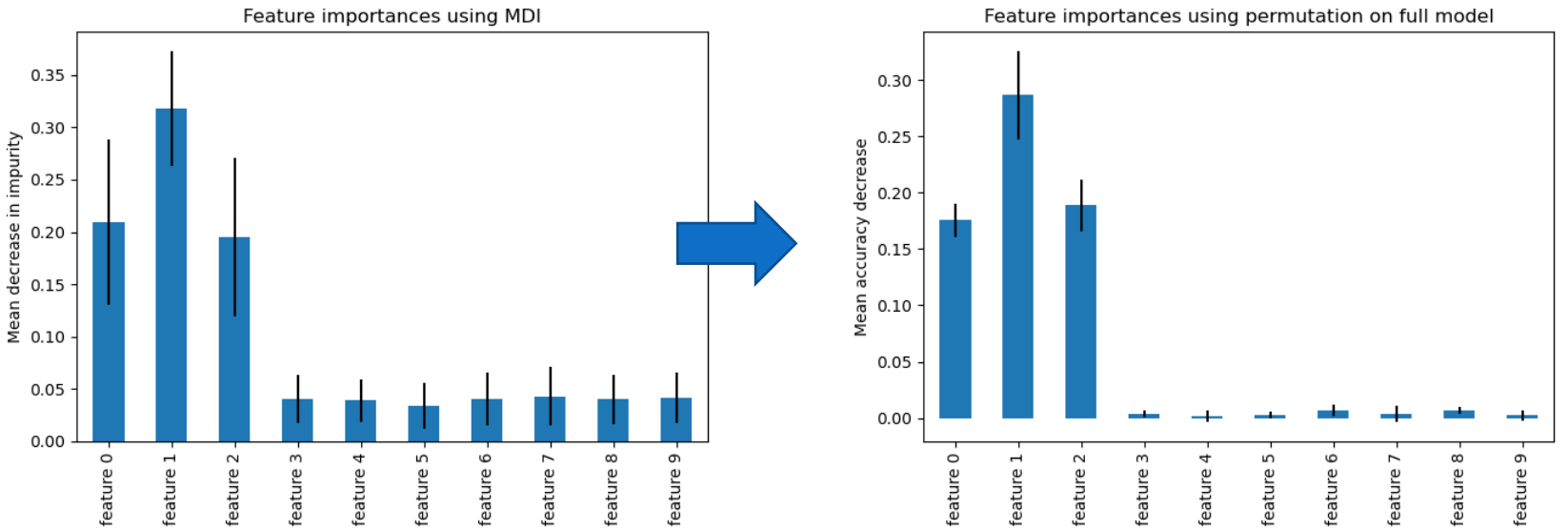



Feature importance




- High absolute coefficient in linear model
- High reduction in impurity in tree model
  - Does not tell us how increase in feature value changes the prediction

# Feature importance – when excluded

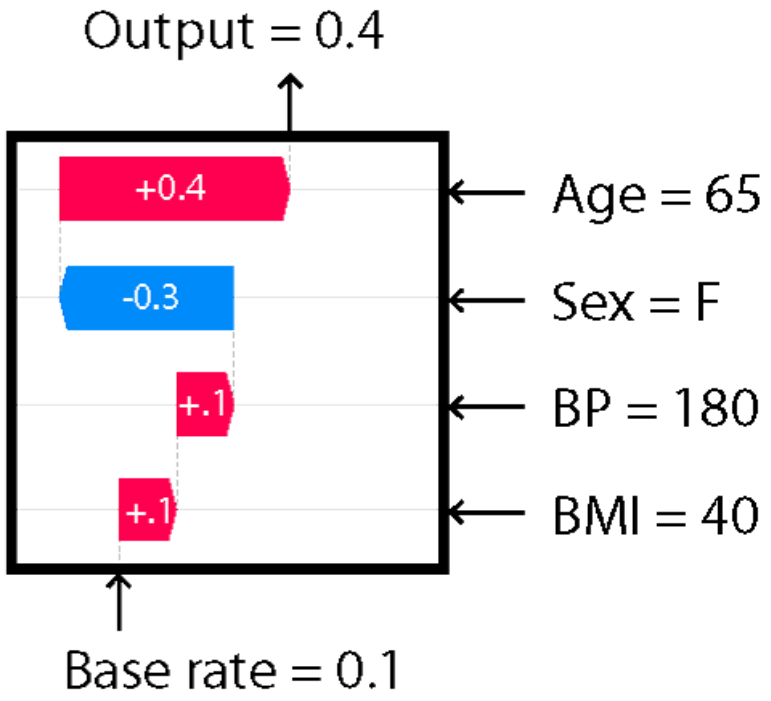



## sklearn.inspection.permutation_importance

sklearn.inspection.permutation_importance(*estimator, X, y, *, scoring=None, n_repeats=5, n_jobs=None, random_state=None, sample_weight=None, max_samples=1.0*) [source]

- Measure importance by randomizing the values of a feature, or dropping a feature from the model and computing the **drop in performance**

# Feature importance – on individual sample



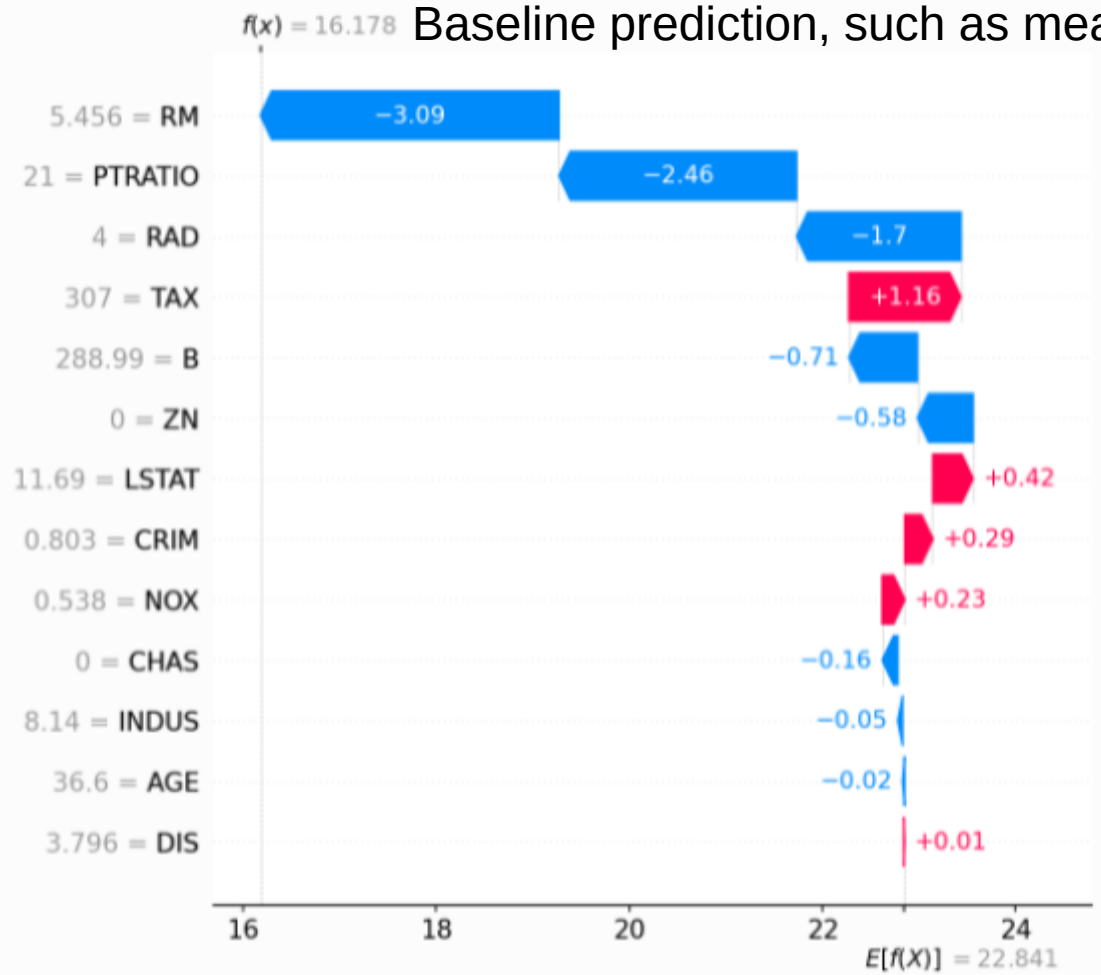

SHAP

```
pip install shap
or
conda install -c conda-forge shap
```

- **Shapley value** = total gain generated by cooperation
- Prediction shift from *f(no information)* to $f(x_1, x_2, \ldots, x_n)$
- $\varphi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} [v(S \cup \{i\}) - v(S)]$
  - Calculated by adding $x_i$ to all feature combinations without $x_i$

# Total Shapley = sum of individual features

```
[6]: # the waterfall_plot shows how we get from shap_values.base_values to model.predict(X)[sample_ind]
     shap.plots.waterfall(shap_values[sample_ind], max_display=14)
```

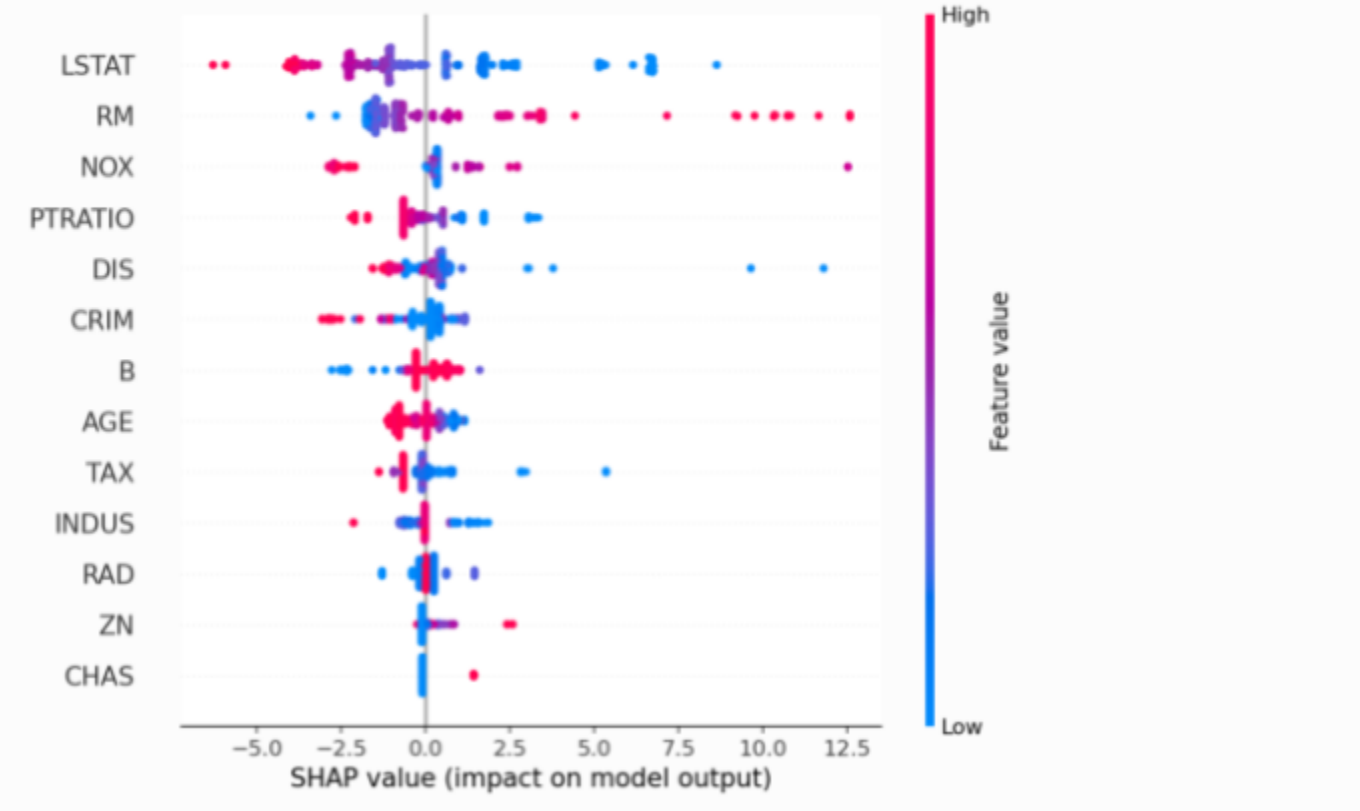



Baseline prediction, such as mean

Prediction for this sample

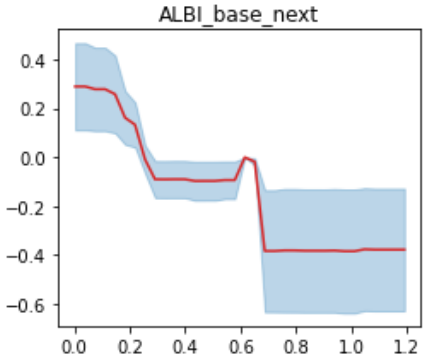
# Model-level Shapley

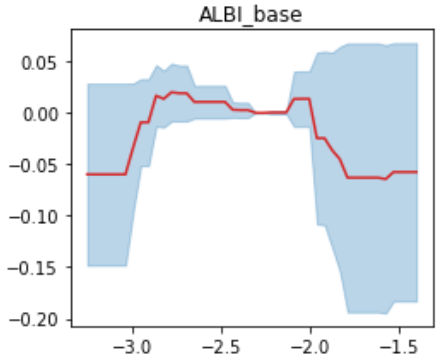
```
# the waterfall_plot shows how we get from explainer.expected_value to model.predict(X)[sample_ind]
shap.plots.beeswarm(shap_values_ebm, max_display=14)
```

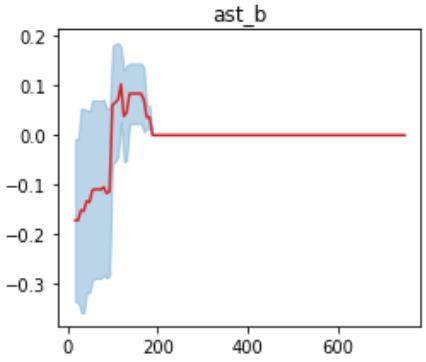


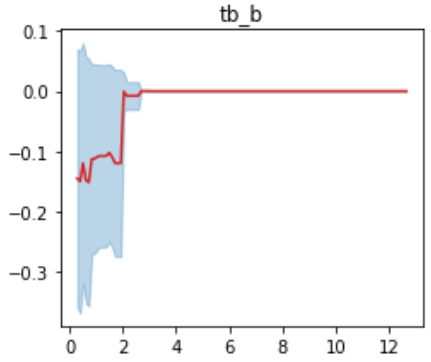

- Each dot = one sample
- Show how increase in feature value changes prediction

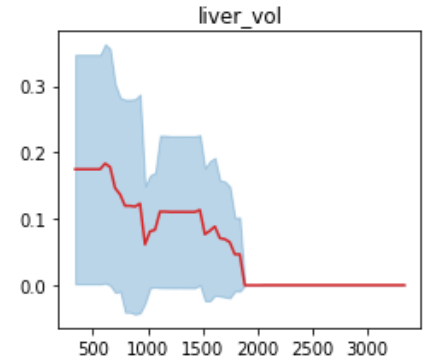
# Another way for showing model behaviors

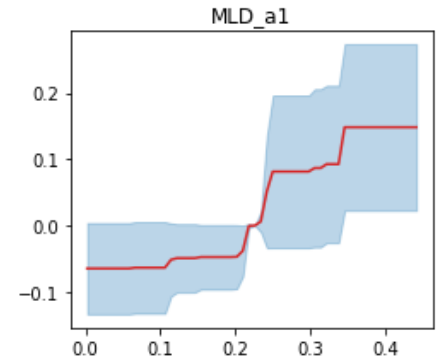



| patient_id | age | sex | pvt_main | surg | tace | cmt | rt_tech | total_dose | no_fx | dose_fx |
|---|---|---|---|---|---|---|---|---|---|---|
| N001 | 50 | 0 | 0 | 0 | 1 | 0 | 2 | 20 | 5 | 4 |

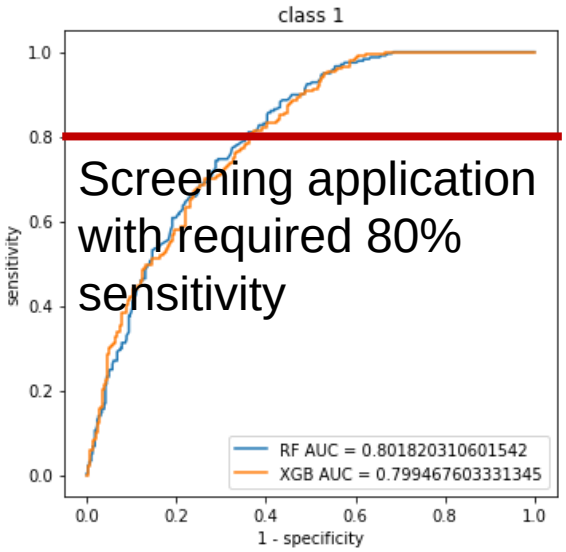
Change *total_dose* to 5, 10, 15, …, 50

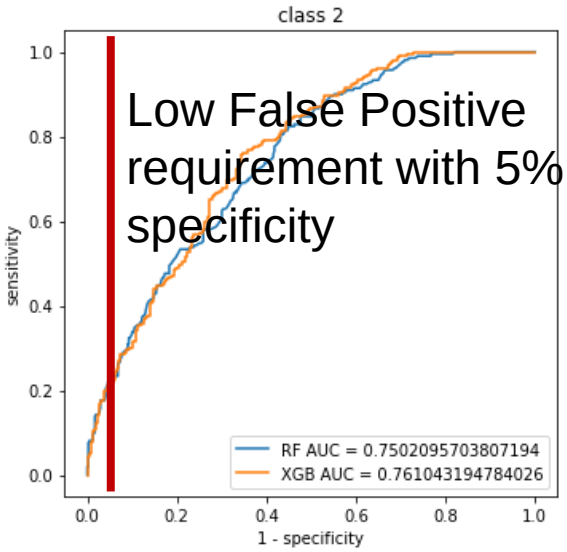
Record changes in prediction

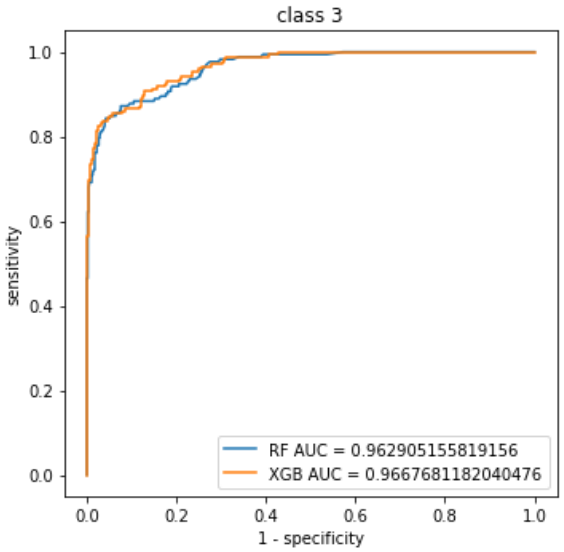
- For each feature, replacing the value in each sample and calculating the change in prediction relative to the mean

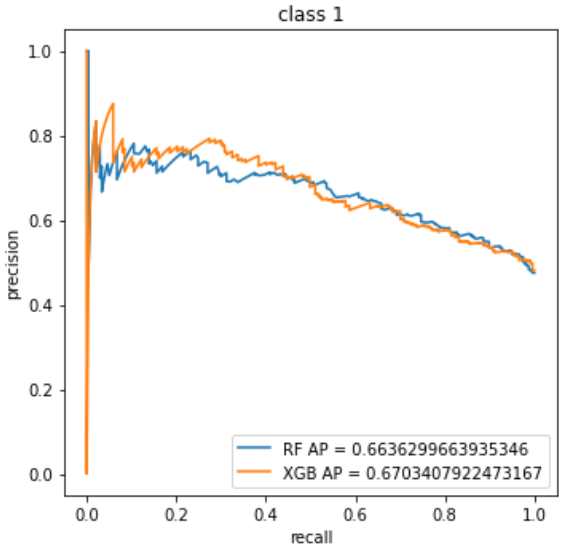
# Probability cutoff and calibration

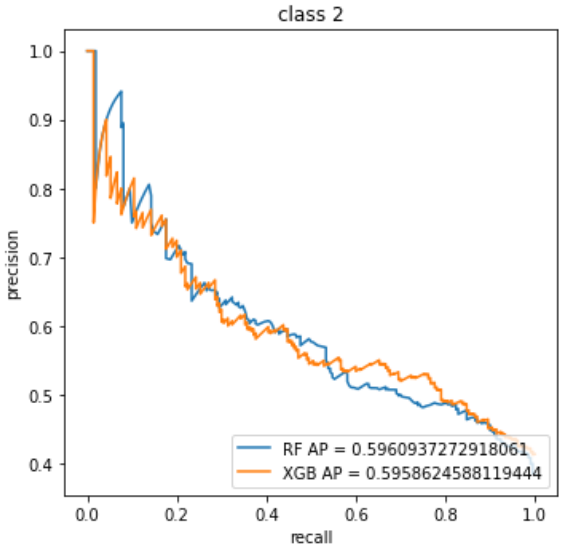
# Cutoff based on user criteria

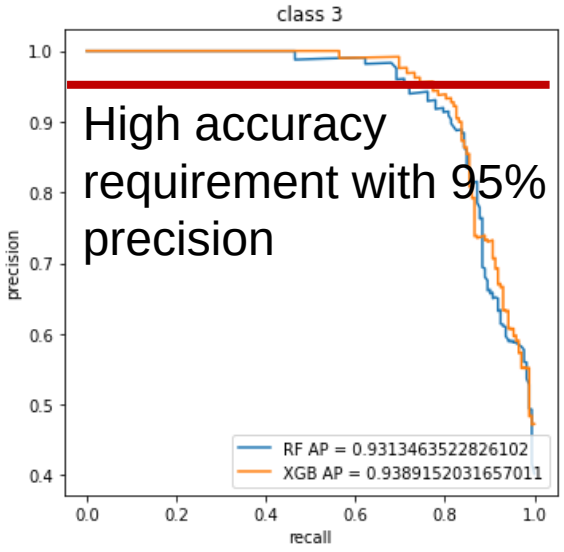


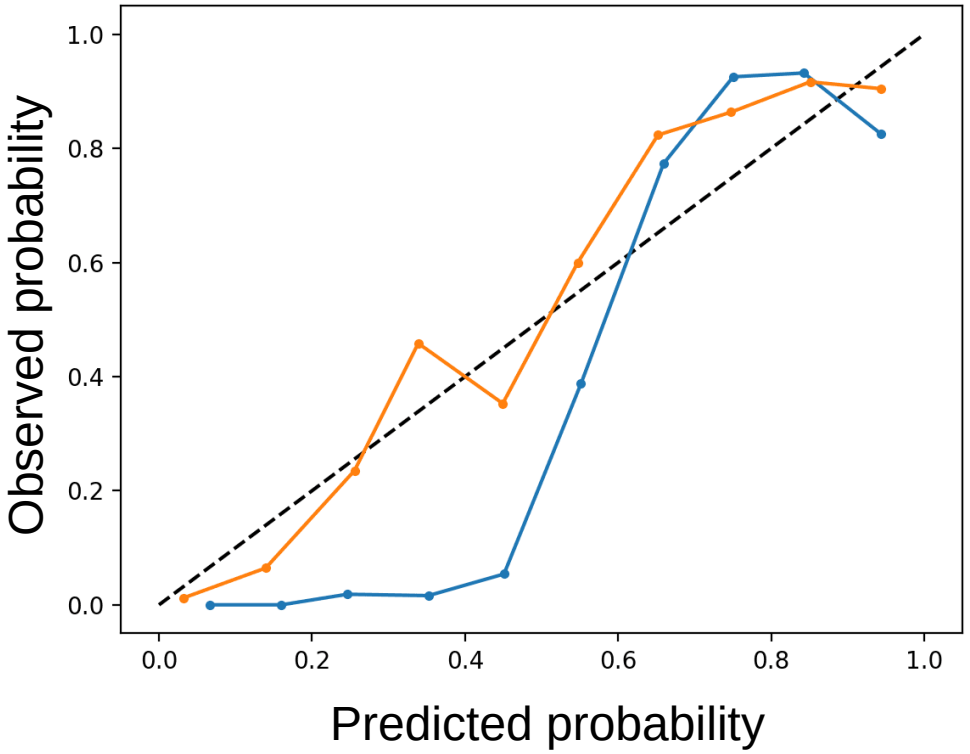



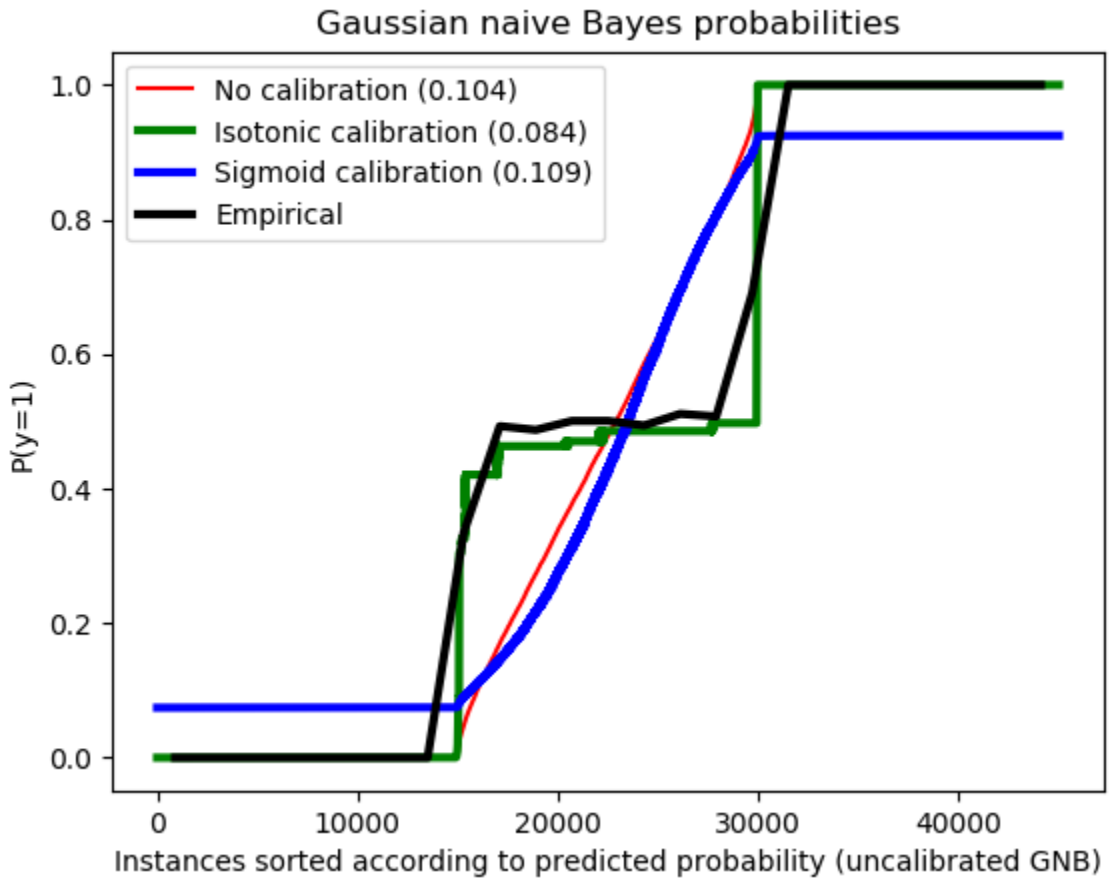



# Interpretation of predicted score




- Calibrated models = those whose prediction reflect the actual probability of being positive
  - 70 out of 100 samples predicted with 0.7 are positive
  - 30 out of 100 samples predicted with 0.3 are positive

# Calibration




- Calibration = shift predicted probability with a monotonic function toward observed probability
- Isotonic method requires a large validation set

# sklearn's calibration method

## sklearn.calibration.CalibratedClassifierCV

*class* sklearn.calibration.CalibratedClassifierCV(*base_estimator=None, *, method='sigmoid', cv=None, n_jobs=None, ensemble=True*) [source]

Probability calibration with isotonic regression or logistic regression.

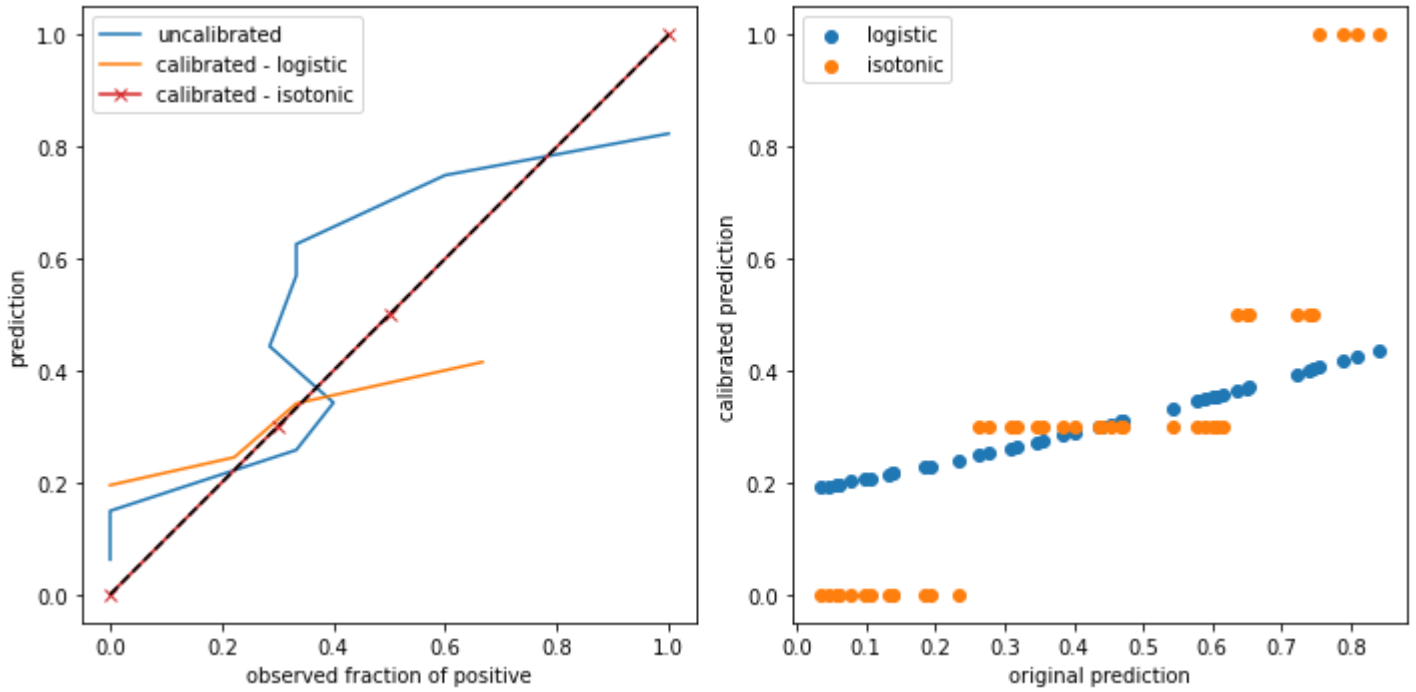
**ensemble : *bool, default=True***

Determines how the calibrator is fitted when `cv` is not `'prefit'`. Ignored if `cv='prefit'`.

If `True`, the `base_estimator` is fitted using training data and calibrated using testing data, for each `cv` fold. The final estimator is an ensemble of `n_cv` fitted classifier and calibrator pairs, where `n_cv` is the number of cross-validation folds. The output is the average predicted probabilities of all pairs.

- Return a model, which is an ensemble (default)
- Predicted probabilities are calibrated on validation sets
- Recommend the default 'sigmoid' method unless you have more than several hundred samples in the validation set

# Calibration with small dataset




- Isotonic method overfit the calibration
- Not enough data points to fit model throughout the range

# Any question?