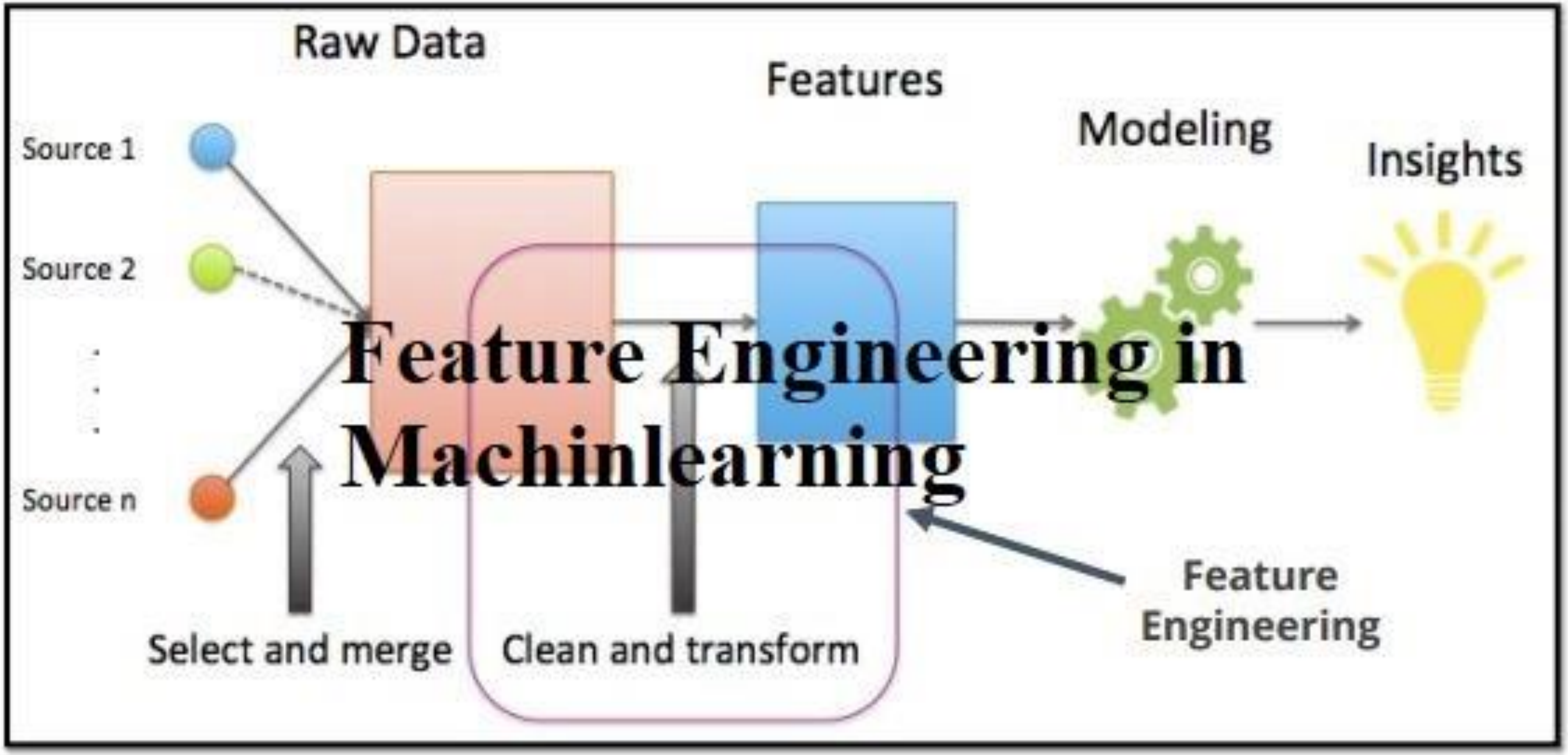
Feature Engineering

Feature Engineering in Machinlearning



Feature Engineering

Data Cleaning and Preprocessing

- **Handling Missing Values:** Strategies include imputation, deletion, or using models to predict missing values.
- **Dealing with Outliers:**

Feature Creation

- **Derived Features:** Creating new features from existing ones, like combining, binning, or polynomial features.
- **Domain Knowledge Incorporation:** Utilizing domain expertise to craft features relevant to the problem.

Feature Scaling and Normalization

- **Scaling Numerical Features:** Ensuring features are on a similar scale (e.g., Min-Max scaling, Standardization).
- **Normalization:** Bringing features to a standard scale to aid model convergence.

Handling Categorical Variables

- **One-Hot Encoding:** Transforming categorical variables into binary vectors.
- **Label Encoding:** Converting categorical values into numerical labels.

Feature Selection/Extraction

- **Removing Redundant Features:** Eliminating features that do not contribute significantly to the model.
- **Techniques:** Filter, wrapper, embedded methods, and dimensionality reduction

1. Data Cleaning and Preprocessing

- Done…

2. Feature Creation

- generating new features from existing data or domain knowledge to improve model performance
- 1. Derived Features
- **Concatenation:** Combining existing features to create new ones (e.g., combining first name and last name to create a full name feature).
- **Arithmetic Operations:** Performing mathematical operations on features (e.g., creating a 'total sales' feature by adding individual sales figures).
- 2. Binning or Discretization
- **Grouping Values:** Dividing continuous features into bins or intervals (e.g., converting age into age groups like 'child,' 'adult,' 'senior').
- **Benefits:** Handles non-linear relationships and reduces the impact of outliers.
- 3. Polynomial Features
- **Higher Order Combinations:** Generating higher-order polynomial features (e.g., squaring or cubing a feature) to capture non-linear relationships.
- **Example:** If a feature 'x' exists, creating 'x^2' or 'x^3' as additional features.
- 4. Domain-Specific Features
- **Incorporating Domain Knowledge:** Utilizing expertise in the field to engineer features specific to the problem domain.
- **Example:** In financial data, creating a feature representing profit margins or risk scores based on industry-specific criteria.

2. Feature Creation Example

- Predicting housing prices based on various features.
- **Dataset Features:** Area, number of bedrooms, number of bathrooms, location, year built, etc.
- **Derived Feature -**
- **Binning - Age of House:**
- **Polynomial Features -**
- **Domain-Specific Feature -**

2. Feature Creation Example

- Predicting housing prices based on various features.
- **Dataset Features:** Area, number of bedrooms, number of bathrooms, location, year built, etc.
- **Derived Feature - Total Area:**
  - **New Feature:** Total area combining the area of each room.
  - **Formula:** Total_Area = Area_Bedroom + Area_Bathroom + Area_LivingRoom + ...
- **Binning - Age of House:**
  - **New Feature:** Age groups for houses based on the year built.
  - **Bins:** 'New,' 'Old,' 'Historic.'
- **Polynomial Features - Interaction Terms:**
  - **New Feature:** Interaction between area and the number of bedrooms.
  - **Feature:** Area * Bedrooms.
- **Domain-Specific Feature - Walkability Score:**
  - **New Feature:** Incorporating a walkability score based on location data.
  - **Criteria:** Proximity to amenities, public transportation, and green spaces.

3. Feature Scaling and Normalization

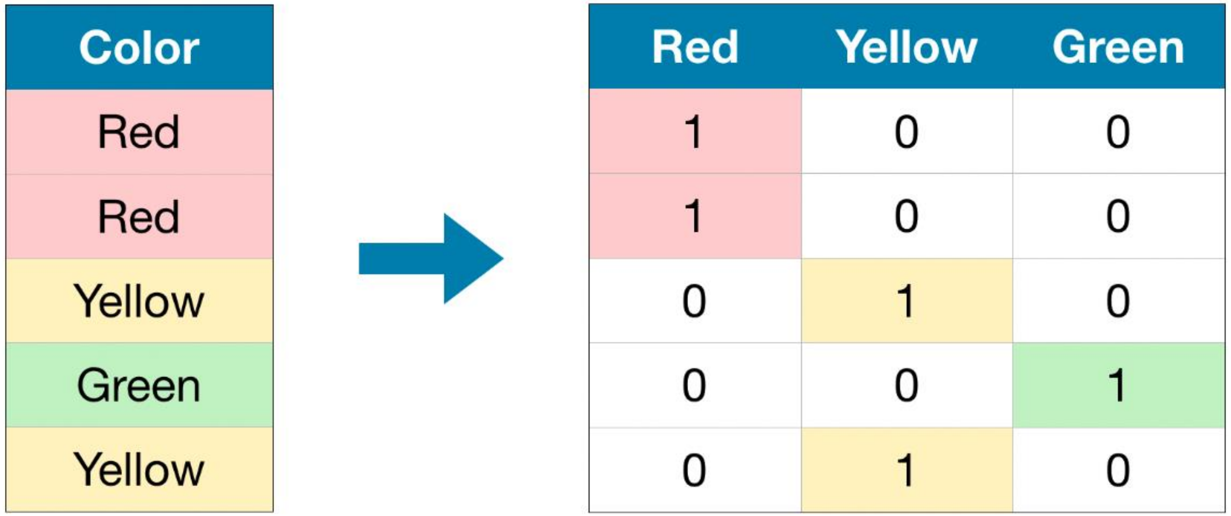
- Done…

4. Handling Categorical Values

- One-Hot Encoding
- Label Encoding
- Ordinal Encoding
- Target Encoding (Mean Encoding)
- Frequency Encoding

One hot encoding

- Each category is transformed into a binary vector where only one element is 1 (hot) while the others are 0 (cold)



Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

One hot encoding

- Predict customer preferences based on demographic data.
- **Categorical Variable:** 'Region' with categories 'North,' 'South,' 'East,' 'West

One hot encoding

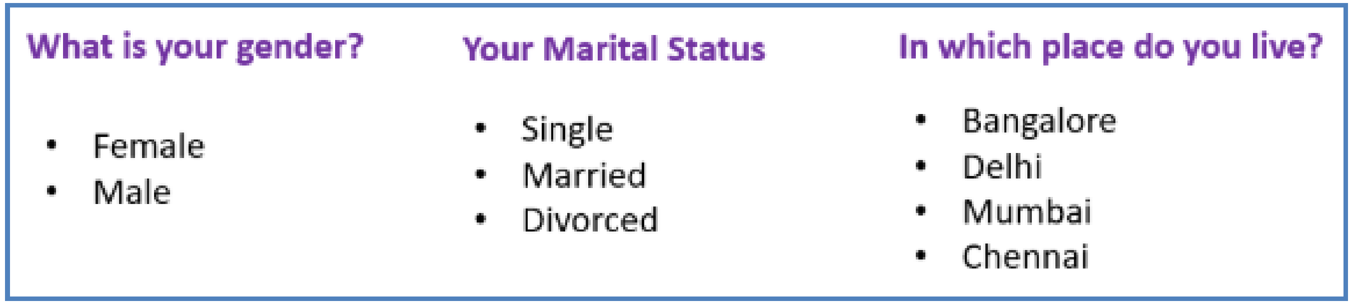
- Predict customer preferences based on demographic data.
- **Categorical Variable:** 'Region' with categories 'North,' 'South,' 'East,' 'West

Original Data	Is_North	Is_South	Is_East	Is_West
North	1	0	0	0
South	0	1	0	0
East	0	0	1	0
West	0	0	0	1

- **Advantages:**
- Preserves information without imposing ordinality.
- Compatible with various machine learning models.
- **Considerations:**
- May lead to a high number of features in cases with many categories

Label Encoding

- Assigns a unique integer to each category.
- **Example:** Assigning integers 0, 1, 2 to 'red,' 'blue,' 'green' respectively.



What is your gender?

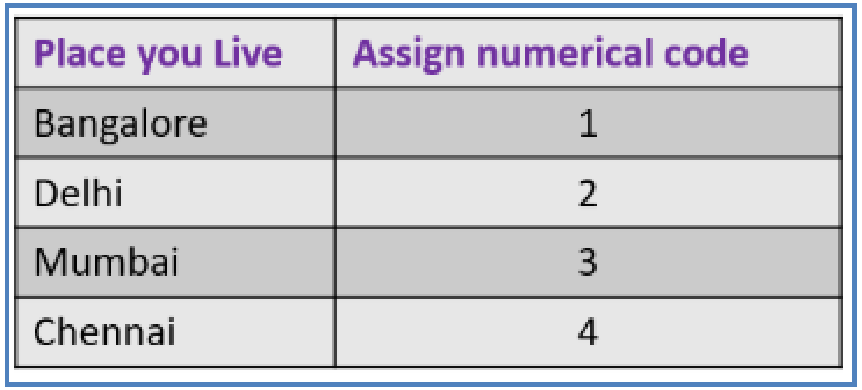
- Female
- Male

Your Marital Status

- Single
- Married
- Divorced

In which place do you live?

- Bangalore
- Delhi
- Mumbai
- Chennai

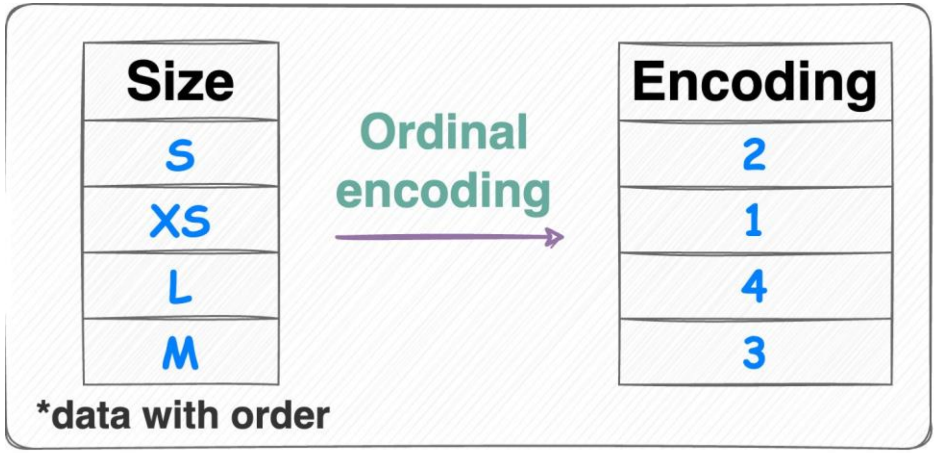


Place you Live	Assign numerical code
Bangalore	1
Delhi	2
Mumbai	3
Chennai	4

- **Advantages:**
- Simple and easy to implement.
- Reduces dimensionality compared to one-hot encoding.
- **Considerations:**
- Imposes ordinality, which might be misleading for some categorical variables.

Ordinal Encoding

- Assigns integers based on the order of categories (relevant for ordinal variables).
- **Example:** 'Low,' 'Medium,' 'High' encoded as 0, 1, 2 respectively



Size
S
XS
L
M

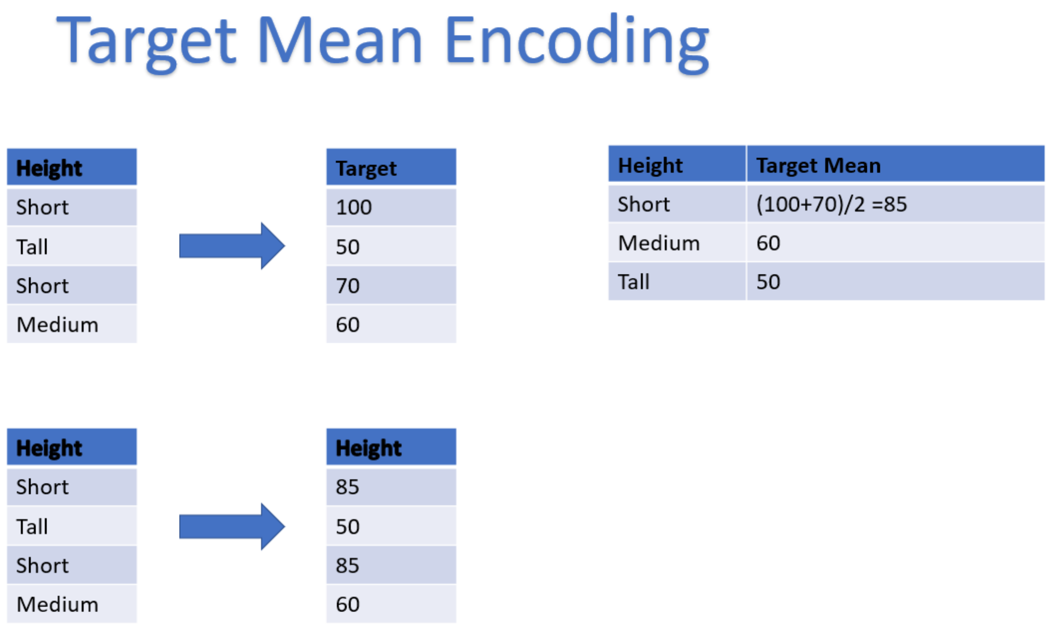
Ordinal encoding

Encoding
2
1
4
3

*data with order

Mean/Target Encoding

- Uses the target variable's mean to encode categories.
- **Example:** Replacing categories with the mean of the target variable for that category.



Target Mean Encoding

Height
Short
Tall
Short
Medium

Target
100
50
70
60

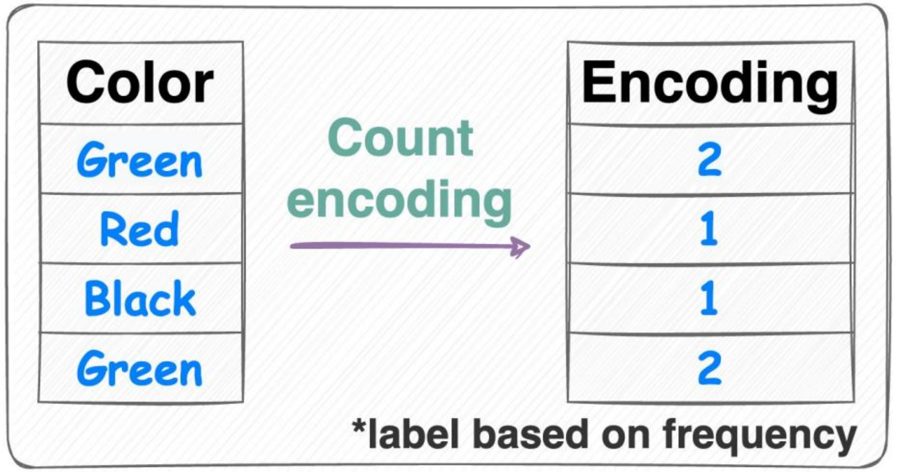
Height	Target Mean
Short	(100+70)/2 =85
Medium	60
Tall	50

Height
Short
Tall
Short
Medium

Height
85
50
85
60

Frequency Encoding

- Encodes categories based on their frequency in the dataset.
- **Example:** Replacing categories with their occurrence count.



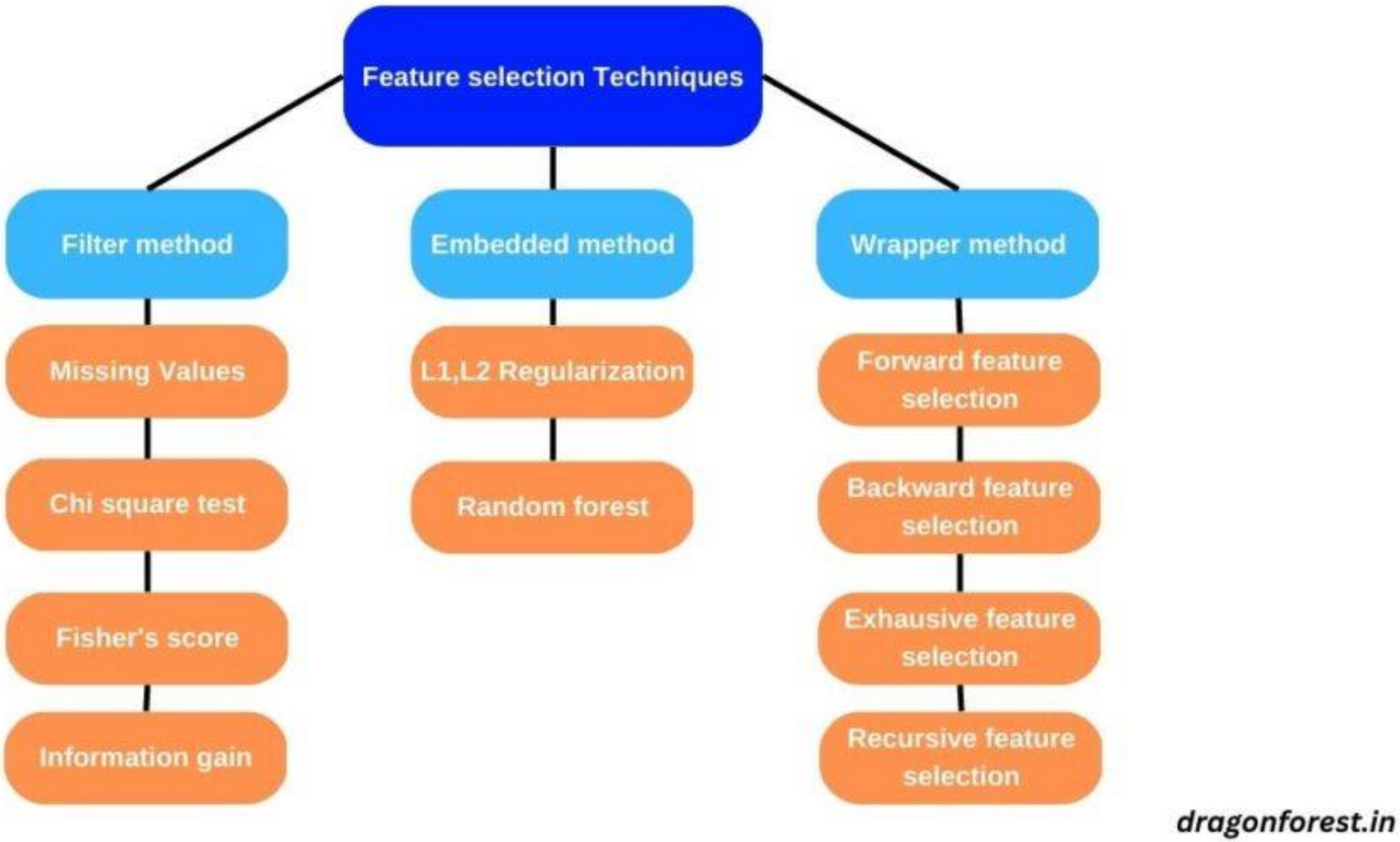
Color	Encoding
Green	2
Red	1
Black	1
Green	2

Count encoding
*label based on frequency

5. Feature Selection

- Feature selection involves choosing a subset of relevant features for use in model construction.
- **Importance:** Reduces dimensionality, improves model performance, mitigates overfitting, and enhances interpretability
- **Forward Selection:** Adds features one by one until no improvement in model performance.
- **Backward Elimination:** Starts with all features and progressively eliminates the least significant.

Feature Selection Methods


Feature selection Techniques

Filter method

Missing Values

Chi square test

Fisher's score

Information gain

Embedded method

L1,L2 Regularization

Random forest

Wrapper method

Forward feature selection

Backward feature selection

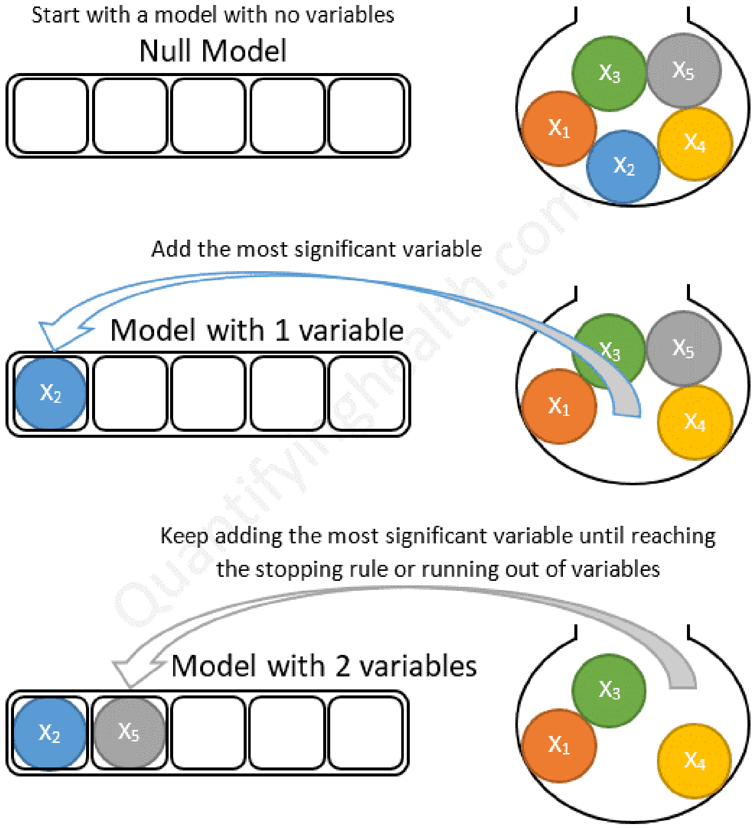
Exhausive feature selection

Recursive feature selection

dragonforest.in

Forward Selection

Forward stepwise selection example with 5 variables:



Start with a model with no variables
Null Model
X_3
X_5
X_1
X_4
X_2
Add the most significant variable
Model with 1 variable
X_2
X_3
X_5
X_1
X_4
Keep adding the most significant variable until reaching the stopping rule or running out of variables
Model with 2 variables
X_2
X_5
X_3
X_1
X_4

1. Initialization: Start with an empty set of features.

2. Iteration:

3. a. **Step 1:** Train the model using each individual feature separately and select the one that performs the best based on a chosen evaluation metric (e.g., accuracy, F1 score, etc.) on a validation set through cross-validation.

Model 1: Train using only the 'age' feature.

Model 2: Train using only the 'income' feature.

Model 3: Train using only the 'gender' feature.

4. b. **Select the Best Feature:** Choose the feature that yields the highest performance metric (e.g., accuracy) among all the models.

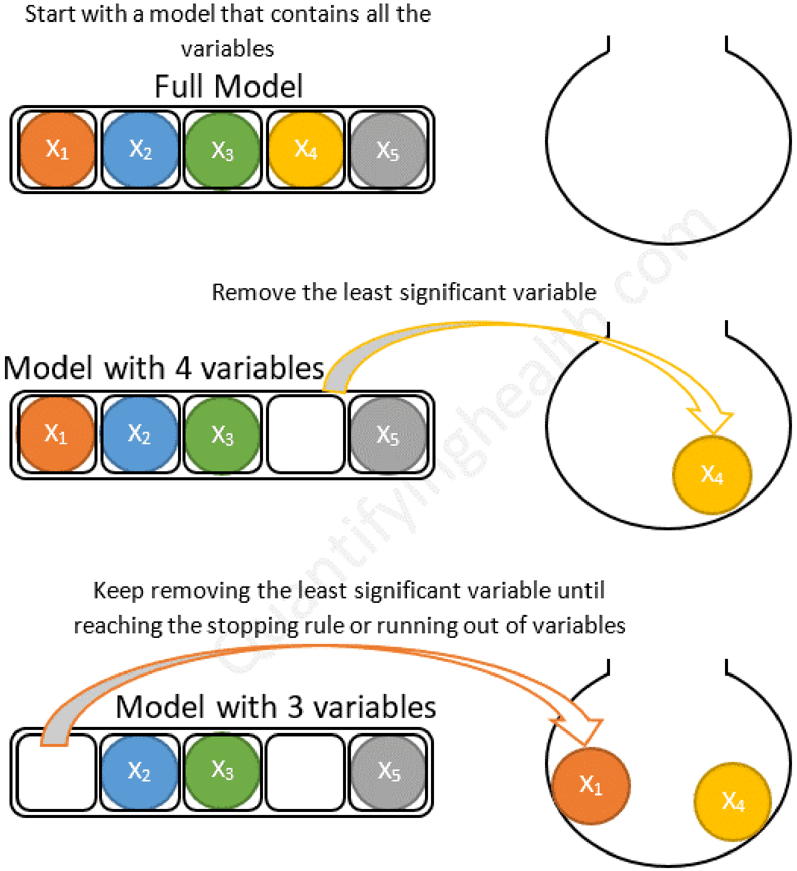
5. c. **Add the Best Feature:** Add the selected feature to the set of chosen features.

6. d. **Iterate:** Repeat steps (a) through (c) until adding more features does not significantly improve the model performance.

7. Final Model: Use the selected set of features to train the final model.

Backward Elimination

Backward stepwise selection example with 5 variables:



Start with a model that contains all the variables

Full Model

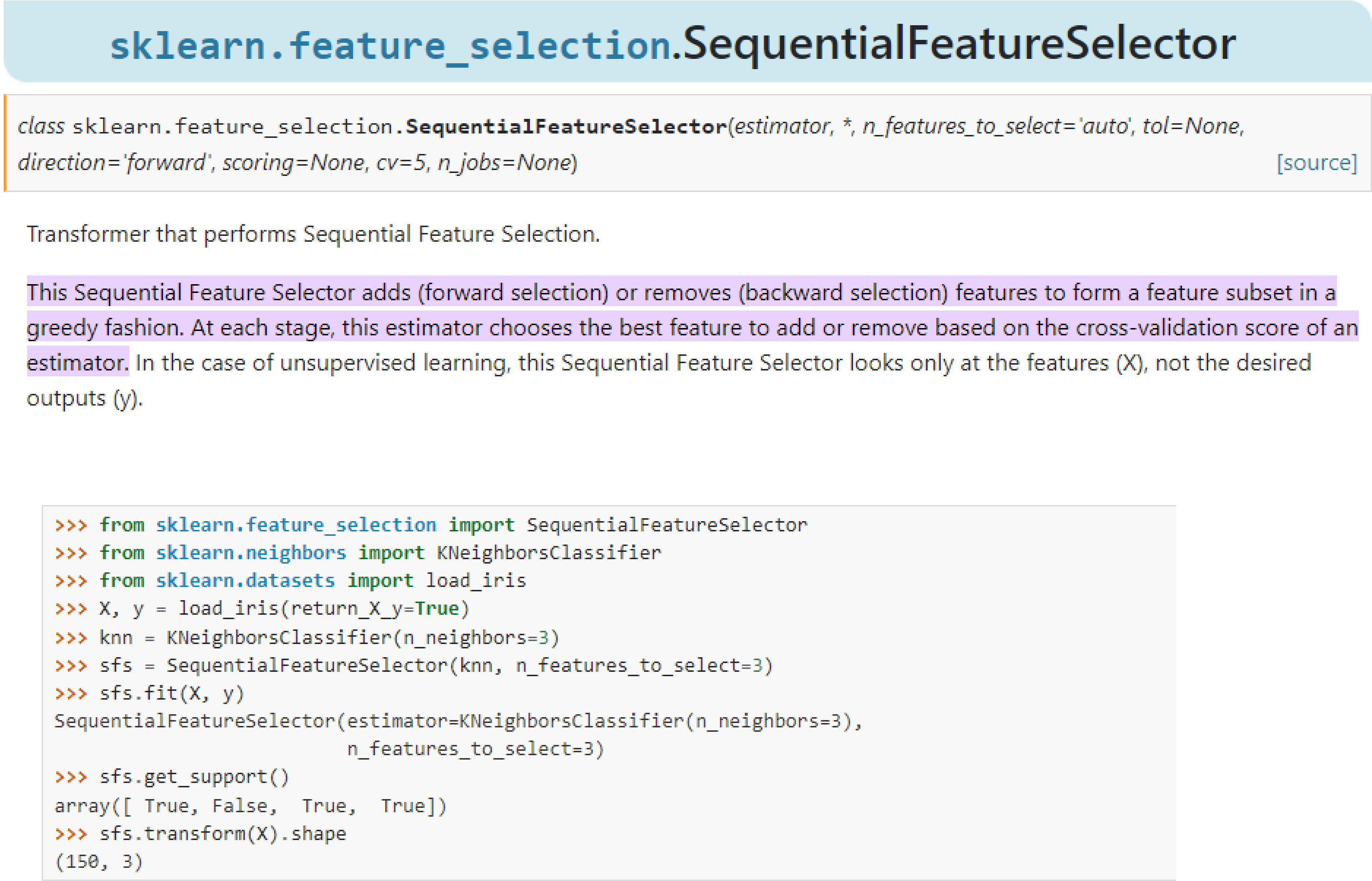
Remove the least significant variable

Model with 4 variables

Keep removing the least significant variable until reaching the stopping rule or running out of variables

Model with 3 variables

1. **Initialization:** Begin with all available features included.
2. **Iteration:**
3. a. **Step 1:** Train the model using all features.
4. b. **Evaluate Model:** Assess the performance of the model using a chosen evaluation metric (e.g., mean squared error, R-squared) on a validation set through cross-validation.
5. c. **Remove One Feature:** Remove one feature (one at a time) from the set of features and train the model again without that particular feature.
6. d. **Evaluate Model without Feature:** Evaluate the performance of the updated model (without the removed feature) using the same evaluation metric.
7. e. **Criterion for Removal:** Remove the feature whose absence causes the least impact or degradation in the model's performance.
8. f. **Iterate:** Repeat steps (a) through (e) until further removal of features does not significantly degrade the model's performance or until a predefined stopping criterion is met.
9. **Final Model:** Use the selected subset of features to train the final model.



sklearn.feature_selection.SequentialFeatureSelector

class sklearn.feature_selection.**SequentialFeatureSelector**(*estimator*, ***, *n_features_to_select='auto'*, *tol=None*, *direction='forward'*, *scoring=None*, *cv=5*, *n_jobs=None*) [source]

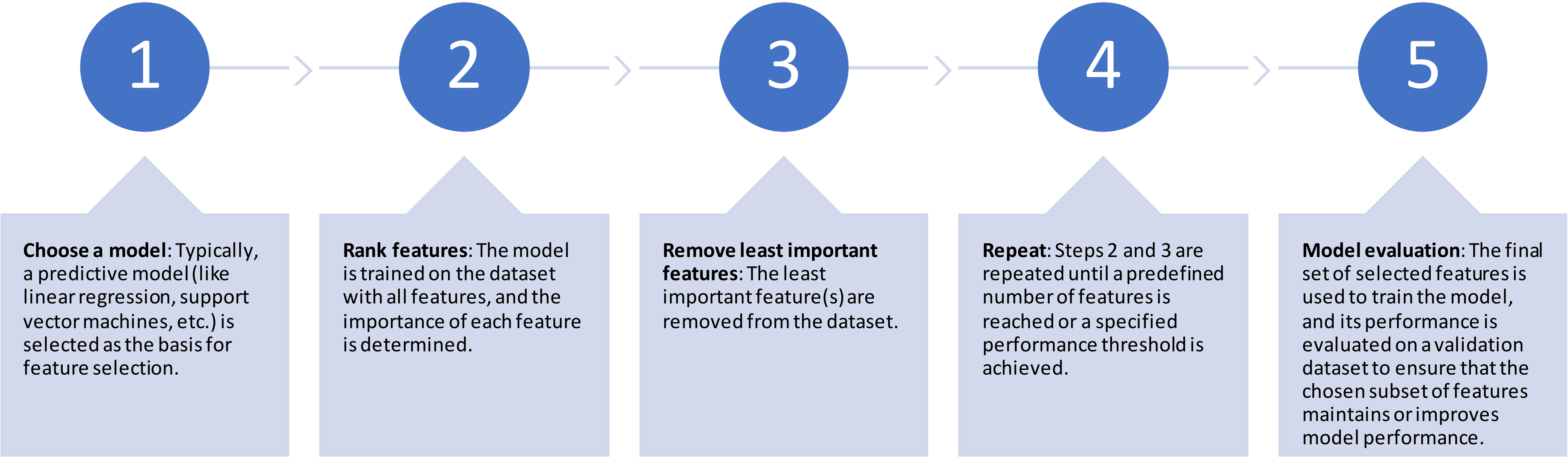
Transformer that performs Sequential Feature Selection.

This Sequential Feature Selector adds (forward selection) or removes (backward selection) features to form a feature subset in a greedy fashion. At each stage, this estimator chooses the best feature to add or remove based on the cross-validation score of an estimator. In the case of unsupervised learning, this Sequential Feature Selector looks only at the features (X), not the desired outputs (y).

```
>>> from sklearn.feature_selection import SequentialFeatureSelector
>>> from sklearn.neighbors import KNeighborsClassifier
>>> from sklearn.datasets import load_iris
>>> X, y = load_iris(return_X_y=True)
>>> knn = KNeighborsClassifier(n_neighbors=3)
>>> sfs = SequentialFeatureSelector(knn, n_features_to_select=3)
>>> sfs.fit(X, y)
SequentialFeatureSelector(estimator=KNeighborsClassifier(n_neighbors=3),
                          n_features_to_select=3)
>>> sfs.get_support()
array([ True, False,  True,  True])
>>> sfs.transform(X).shape
(150, 3)
```

- https://www.kaggle.com/code/jurk06/forward-and-backward-subset-selection-method

Recursive Feature Elimination



1

Choose a model: Typically, a predictive model (like linear regression, support vector machines, etc.) is selected as the basis for feature selection.

2

Rank features: The model is trained on the dataset with all features, and the importance of each feature is determined.

3

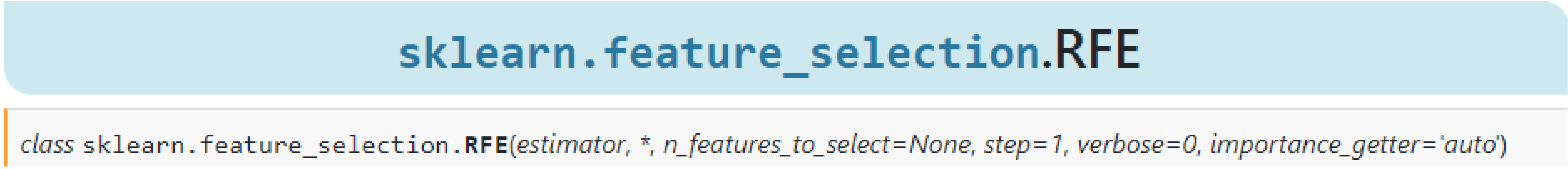
Remove least important features: The least important feature(s) are removed from the dataset.

4

Repeat: Steps 2 and 3 are repeated until a predefined number of features is reached or a specified performance threshold is achieved.

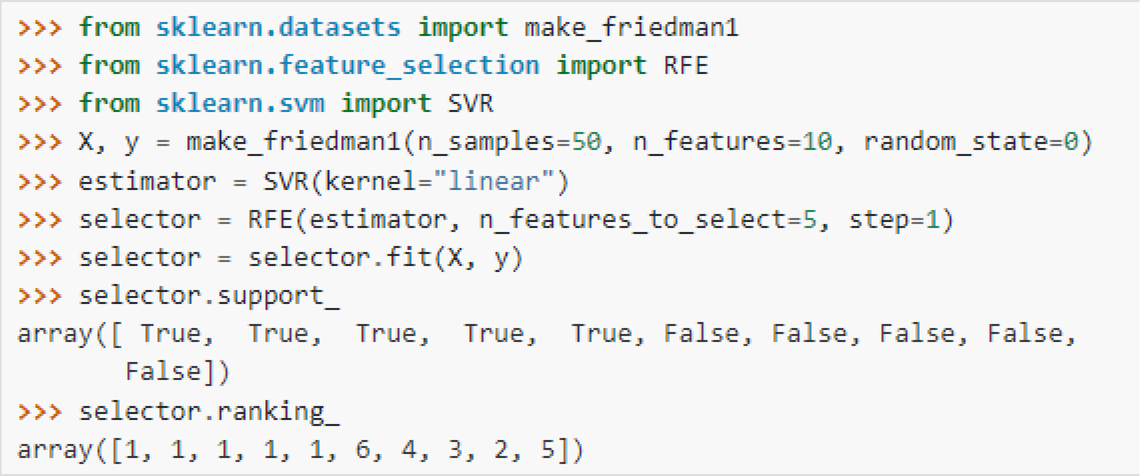
5

Model evaluation: The final set of selected features is used to train the model, and its performance is evaluated on a validation dataset to ensure that the chosen subset of features maintains or improves model performance.



sklearn.feature_selection.RFE

class sklearn.feature_selection.**RFE**(*estimator, *, n_features_to_select=None, step=1, verbose=0, importance_getter='auto'*)

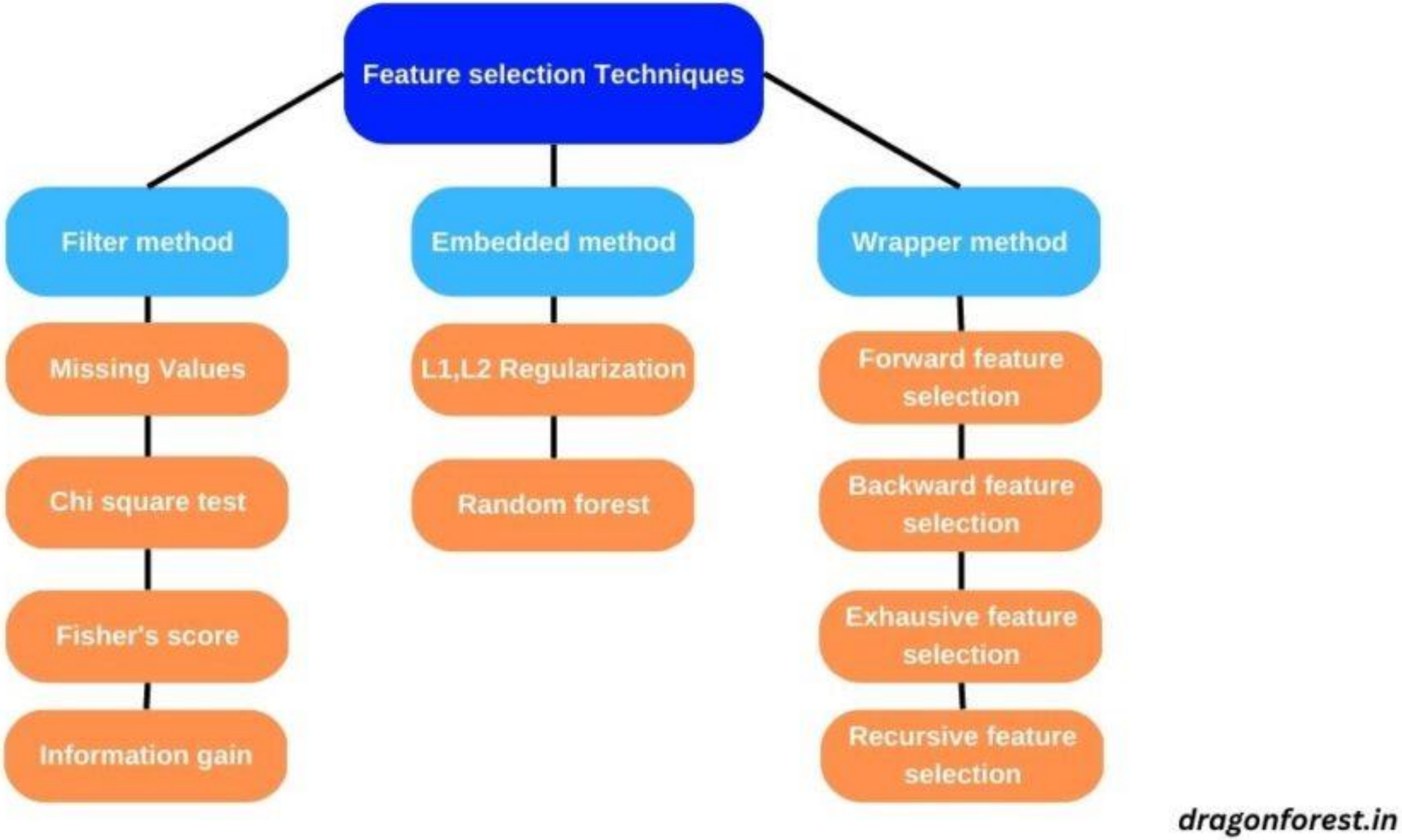


```
>>> from sklearn.datasets import make_friedman1
>>> from sklearn.feature_selection import RFE
>>> from sklearn.svm import SVR
>>> X, y = make_friedman1(n_samples=50, n_features=10, random_state=0)
>>> estimator = SVR(kernel="linear")
>>> selector = RFE(estimator, n_features_to_select=5, step=1)
>>> selector = selector.fit(X, y)
>>> selector.support_
array([ True,  True,  True,  True,  True, False, False, False, False,
       False])
>>> selector.ranking_
array([1, 1, 1, 1, 1, 6, 4, 3, 2, 5])
```

Exhaustive Feature Selection

- find the best subset of features by evaluating all possible feature combinations

Feature Selection Methods



Feature selection Techniques

Filter method

Missing Values

Chi square test

Fisher's score

Information gain

Embedded method

L1,L2 Regularization

Random forest

Wrapper method

Forward feature selection

Backward feature selection

Exhausive feature selection

Recursive feature selection

dragonforest.in

Feature Selection using Random Forest

Training the Random Forest Model: The Random Forest algorithm builds multiple decision trees by using bootstrapped samples of the dataset and randomly selecting a subset of features for each tree.

Feature Importance Calculation: During the training of each tree in the Random Forest, the algorithm measures how much each feature decreases the impurity (e.g., Gini impurity for classification) across all the trees in the forest. This measure of importance is accumulated and averaged across all trees to determine the overall importance of each feature.

Ranking or Selection: Once the model is trained, the feature importances can be extracted. These importances provide a score for each feature, indicating how much each feature contributes to the model's predictive performance.

Feature Selection: Based on the feature importances, you can then choose to select a subset of the most important features for further modeling or analysis.

L1 L2 Regularization

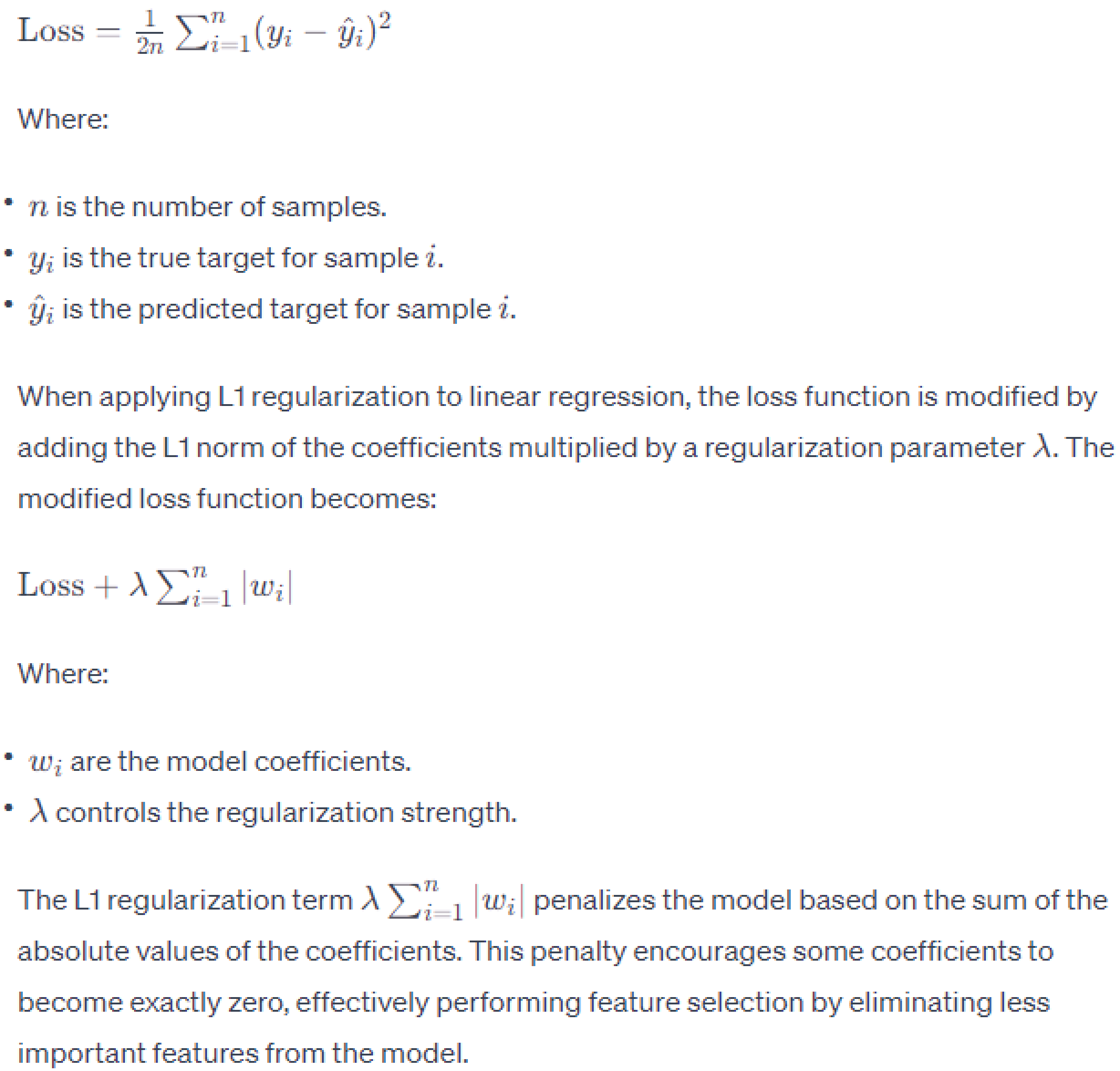
- To prevent overfitting by adding a penalty term to the model's loss function. Its primary goal is to discourage the learning algorithm from fitting the training data too closely, thereby promoting simpler models that generalize better to unseen data
- **L1 Regularization**:
  - L1 regularization adds the absolute value of the magnitude of coefficients as a penalty term to the loss function.
  - It encourages sparsity in the model by pushing some coefficients to exactly zero, effectively performing feature selection.
  - The optimization objective with L1 regularization is:

$$\text{Loss} + \lambda \textstyle\sum_{i=1}^{n} |w_i|,$$

  - **L2 Regularization**:
  - L2 regularization adds the squared magnitude of coefficients as a penalty term to the loss function.
  - It encourages the weights to be small but does not usually force them to be exactly zero.
  - The optimization objective with L2 regularization is:

$$\text{Loss} + \lambda \textstyle\sum_{i=1}^{n} w_i^2,$$

How L1 regularization works?



$\text{Loss} = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

Where:

- n is the number of samples.
- y_i is the true target for sample i.
- $\hat{y}_i$ is the predicted target for sample i.

When applying L1 regularization to linear regression, the loss function is modified by adding the L1 norm of the coefficients multiplied by a regularization parameter λ. The modified loss function becomes:

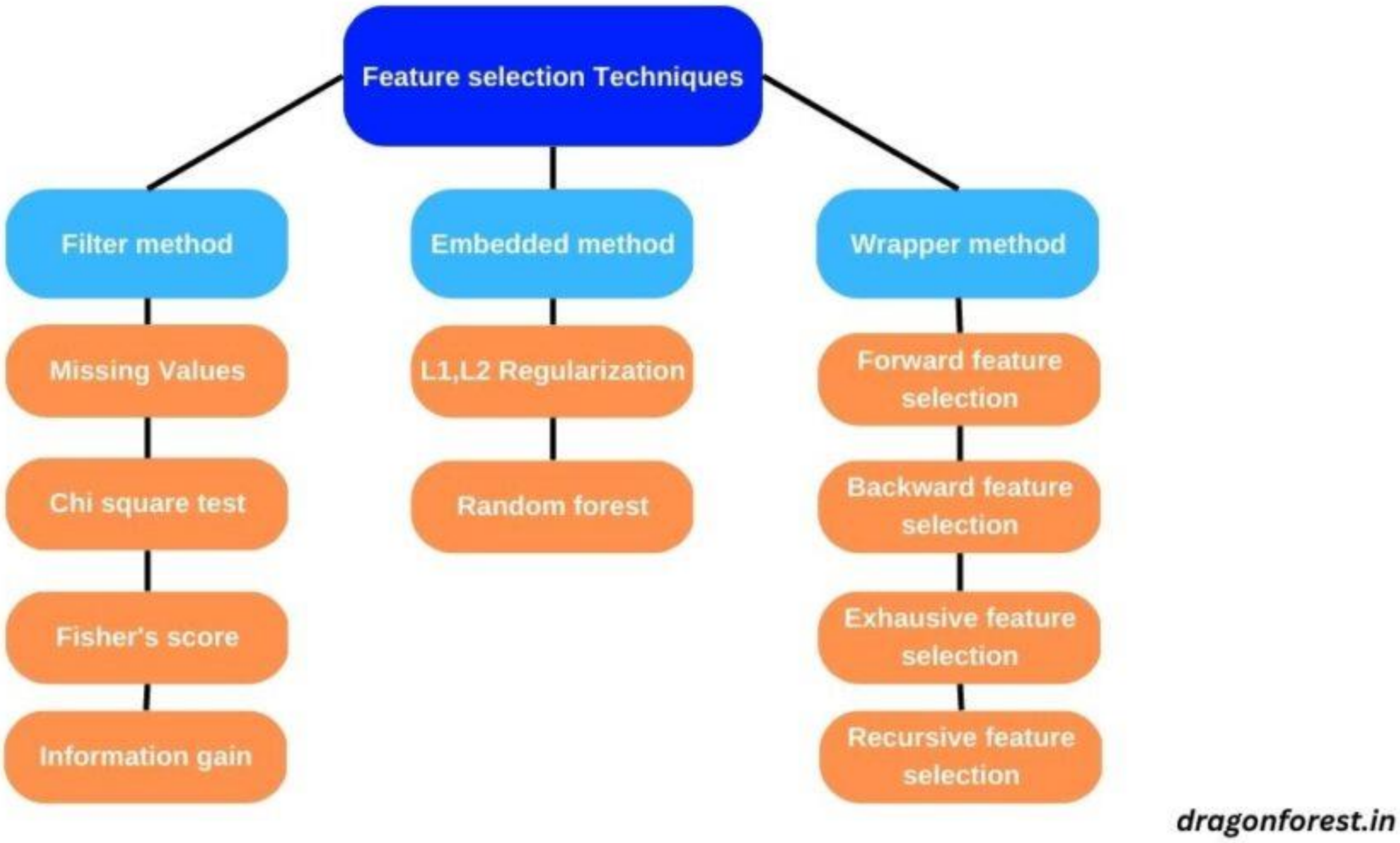
$\text{Loss} + \lambda\sum_{i=1}^{n}|w_i|$

Where:

- w_i are the model coefficients.
- λ controls the regularization strength.

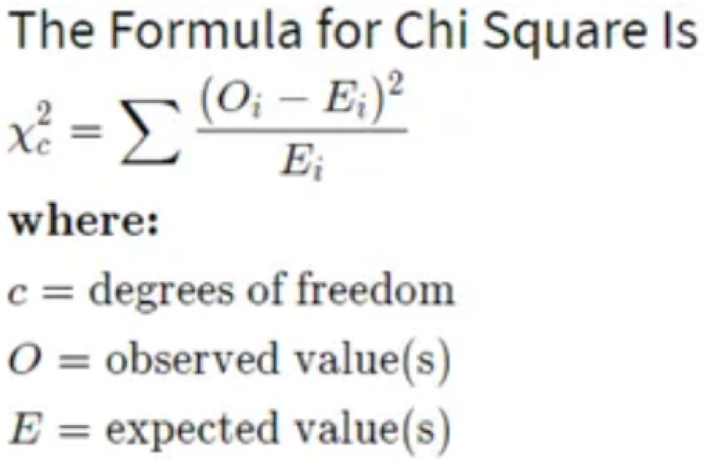
The L1 regularization term $\lambda\sum_{i=1}^{n}|w_i|$ penalizes the model based on the sum of the absolute values of the coefficients. This penalty encourages some coefficients to become exactly zero, effectively performing feature selection by eliminating less important features from the model.

Feature Selection Methods



Chi square test based feature selection

A chi-square test is used in statistics to test the independence of two events. Given the data of two variables, we can get observed count O and expected count E. Chi-Square measures how expected count E and observed count O deviates each other.



The Formula for Chi Square Is

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

where:

$c =$ degrees of freedom

$O =$ observed value(s)

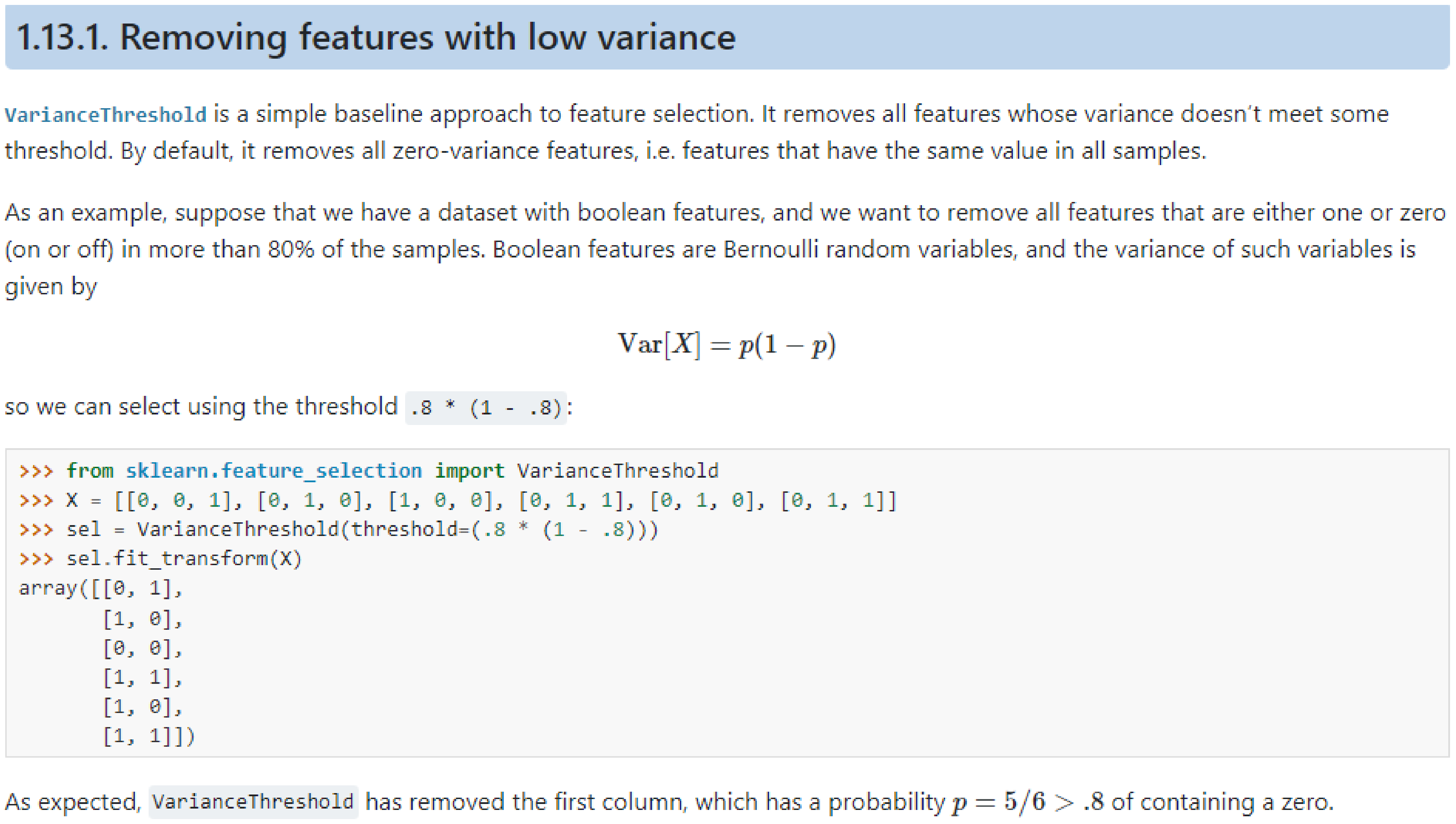
$E =$ expected value(s)

- https://towardsdatascience.com/chi-square-test-for-feature-selection-in-machine-learning-206b1f0b8223

- https://www.kdnuggets.com/2023/06/advanced-feature-selection-techniques-machine-learning-models.html
- https://ranasinghiitkgp.medium.com/implementing-feature-selection-methods-for-machine-learning-bfa2e4b4e02

Fisher’s Score

- Fisher's Discriminant Ratio, commonly known as Fisher's Score, is a feature selection approach that ranks features based on their ability to differentiate various classes in a dataset. It may be used for continuous features in a classification problem.
- Fisher's Score is calculated as the ratio of between-class and within-class variance. A higher Fisher's Score implies the characteristic is more discriminative and valuable for classification.



1.13.1. Removing features with low variance

VarianceThreshold is a simple baseline approach to feature selection. It removes all features whose variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e. features that have the same value in all samples.

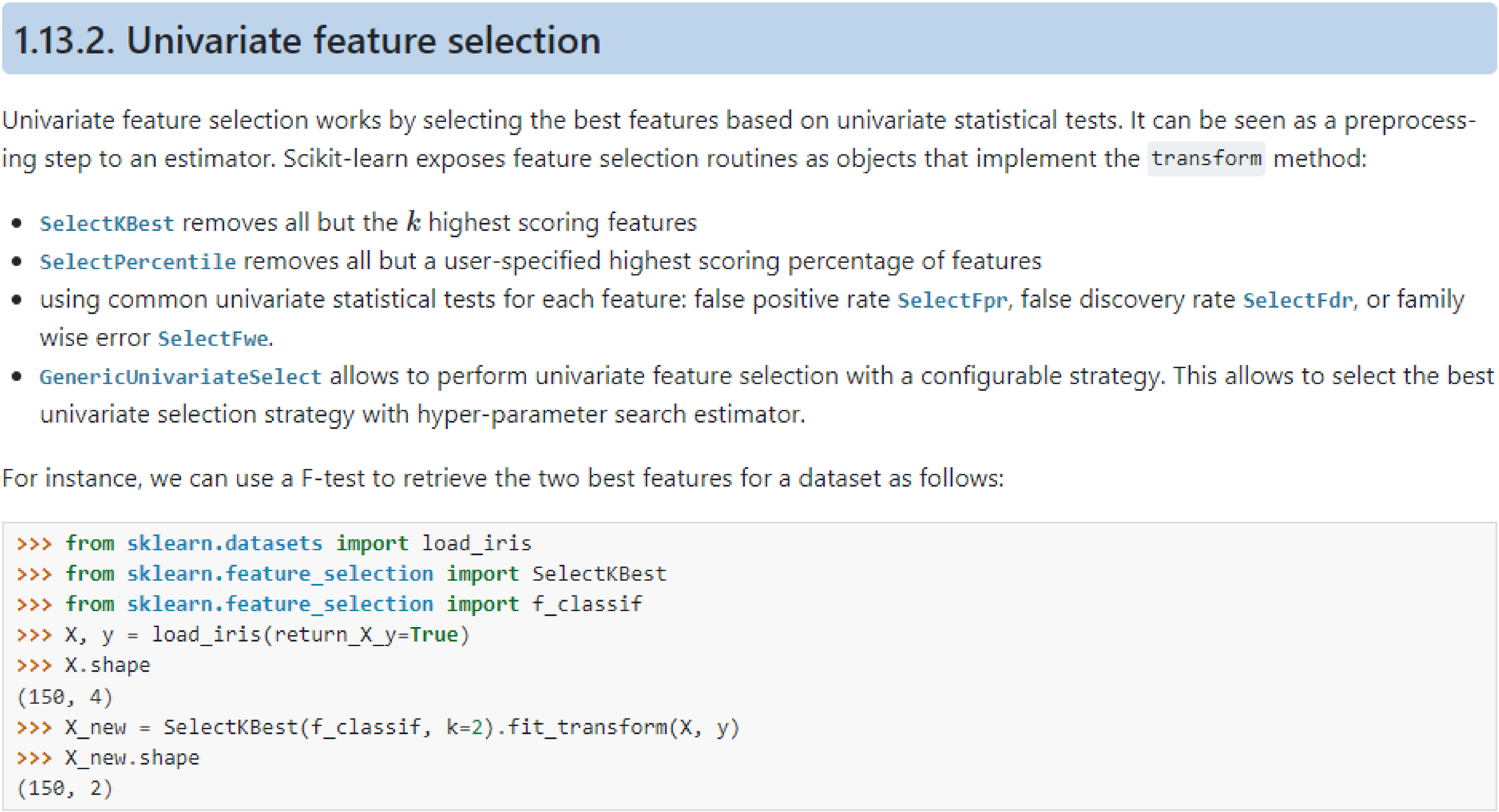
As an example, suppose that we have a dataset with boolean features, and we want to remove all features that are either one or zero (on or off) in more than 80% of the samples. Boolean features are Bernoulli random variables, and the variance of such variables is given by

$$\mathrm{Var}[X] = p(1 - p)$$

so we can select using the threshold `.8 * (1 - .8)`:

```
>>> from sklearn.feature_selection import VarianceThreshold
>>> X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
>>> sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
>>> sel.fit_transform(X)
array([[0, 1],
       [1, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [1, 1]])
```

As expected, `VarianceThreshold` has removed the first column, which has a probability $p = 5/6 > .8$ of containing a zero.



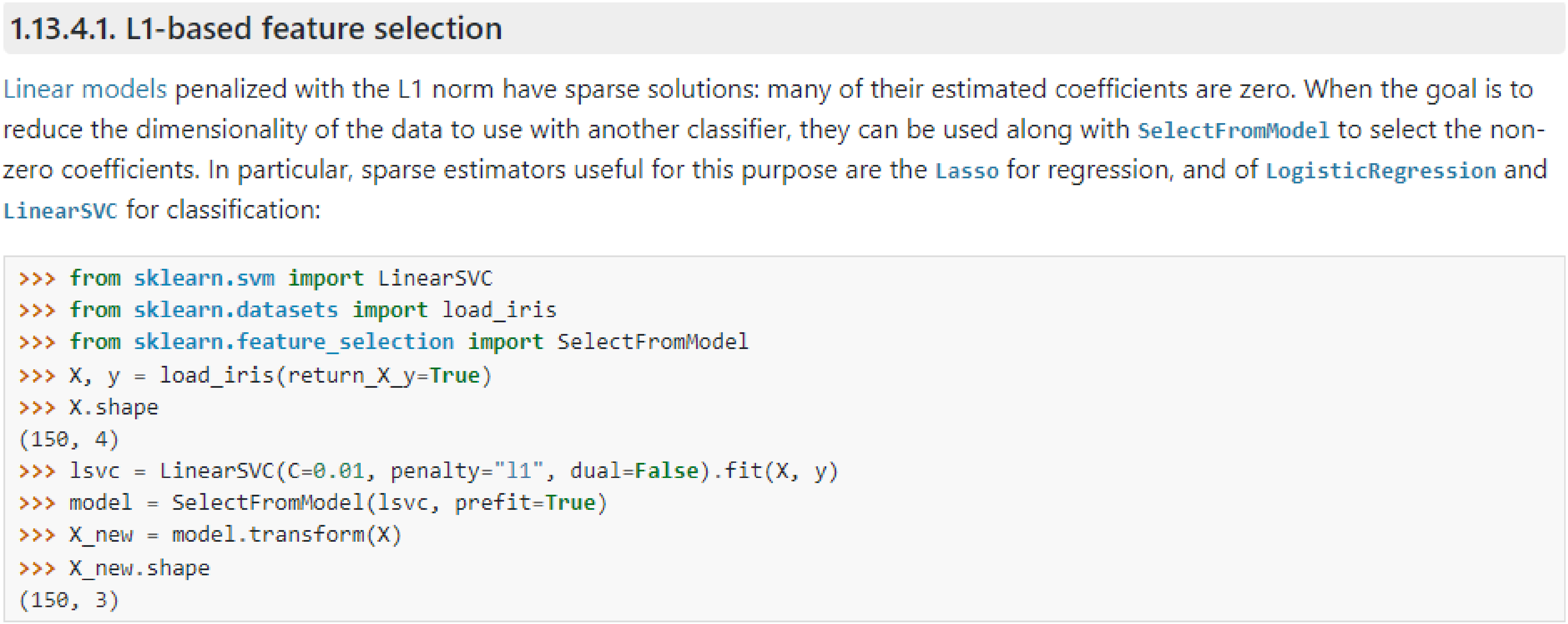
1.13.2. Univariate feature selection

Univariate feature selection works by selecting the best features based on univariate statistical tests. It can be seen as a preprocessing step to an estimator. Scikit-learn exposes feature selection routines as objects that implement the `transform` method:

- `SelectKBest` removes all but the k highest scoring features
- `SelectPercentile` removes all but a user-specified highest scoring percentage of features
- using common univariate statistical tests for each feature: false positive rate `SelectFpr`, false discovery rate `SelectFdr`, or family wise error `SelectFwe`.
- `GenericUnivariateSelect` allows to perform univariate feature selection with a configurable strategy. This allows to select the best univariate selection strategy with hyper-parameter search estimator.

For instance, we can use a F-test to retrieve the two best features for a dataset as follows:

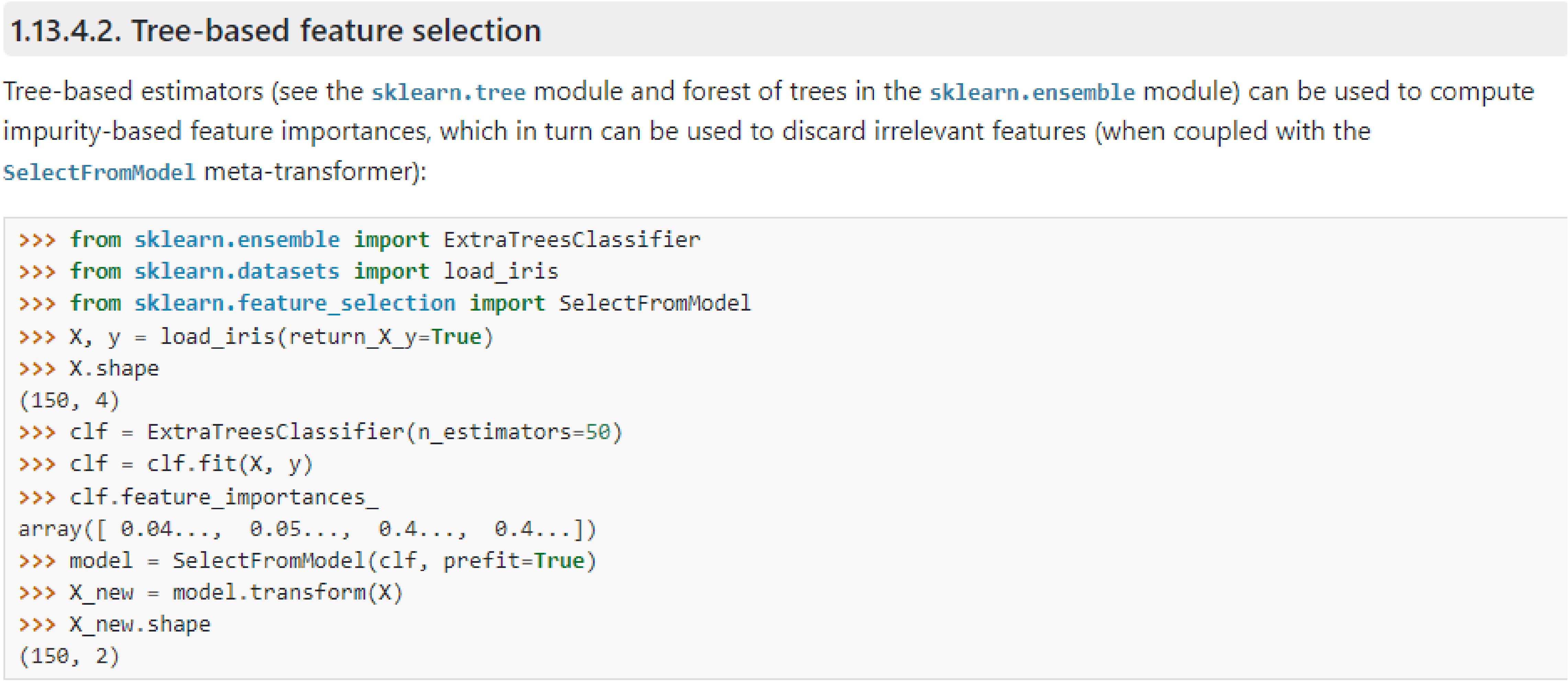
```
>>> from sklearn.datasets import load_iris
>>> from sklearn.feature_selection import SelectKBest
>>> from sklearn.feature_selection import f_classif
>>> X, y = load_iris(return_X_y=True)
>>> X.shape
(150, 4)
>>> X_new = SelectKBest(f_classif, k=2).fit_transform(X, y)
>>> X_new.shape
(150, 2)
```



1.13.4.1. L1-based feature selection

Linear models penalized with the L1 norm have sparse solutions: many of their estimated coefficients are zero. When the goal is to reduce the dimensionality of the data to use with another classifier, they can be used along with `SelectFromModel` to select the non-zero coefficients. In particular, sparse estimators useful for this purpose are the `Lasso` for regression, and of `LogisticRegression` and `LinearSVC` for classification:

```
>>> from sklearn.svm import LinearSVC
>>> from sklearn.datasets import load_iris
>>> from sklearn.feature_selection import SelectFromModel
>>> X, y = load_iris(return_X_y=True)
>>> X.shape
(150, 4)
>>> lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X, y)
>>> model = SelectFromModel(lsvc, prefit=True)
>>> X_new = model.transform(X)
>>> X_new.shape
(150, 3)
```



1.13.4.2. Tree-based feature selection

Tree-based estimators (see the `sklearn.tree` module and forest of trees in the `sklearn.ensemble` module) can be used to compute impurity-based feature importances, which in turn can be used to discard irrelevant features (when coupled with the `SelectFromModel` meta-transformer):

```
>>> from sklearn.ensemble import ExtraTreesClassifier
>>> from sklearn.datasets import load_iris
>>> from sklearn.feature_selection import SelectFromModel
>>> X, y = load_iris(return_X_y=True)
>>> X.shape
(150, 4)
>>> clf = ExtraTreesClassifier(n_estimators=50)
>>> clf = clf.fit(X, y)
>>> clf.feature_importances_
array([ 0.04...,  0.05...,  0.4...,  0.4...])
>>> model = SelectFromModel(clf, prefit=True)
>>> X_new = model.transform(X)
>>> X_new.shape
(150, 2)
```

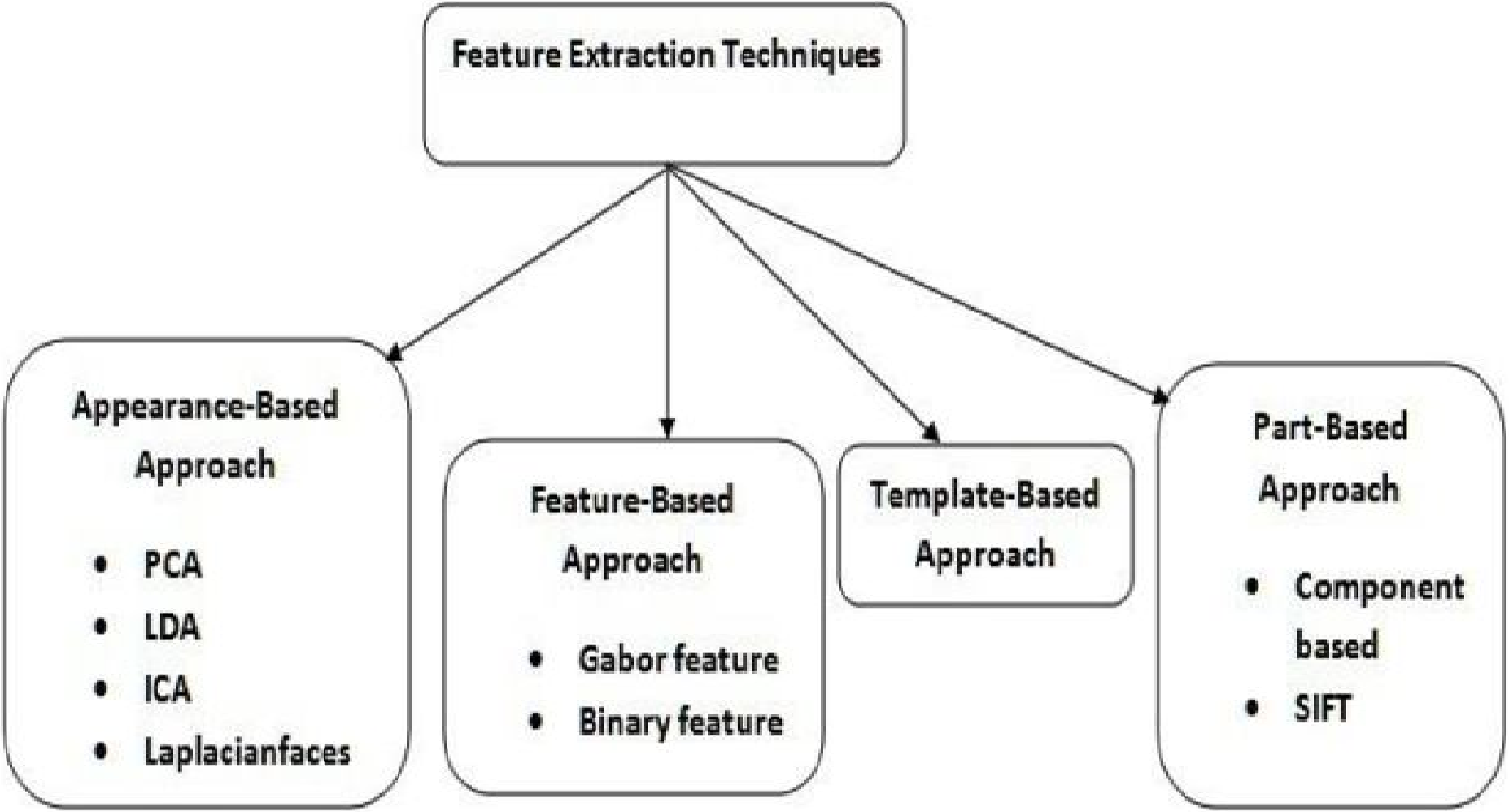


Fig. 1 Classification of Various Feature Extraction Techniques