Caret versus Sk-learn

Caret provides one of the most comprehensive wrappers for any set of R packages and can be solely used to define an entire workflow starting from data cleaning and preprocessing, all the way through model training, prediction, and performance analysis. Plus, it is free to use. Caret can perform various data mining functionalities easier and more in a more user-friendly way. On the other hand, Scikit-learn provides the same functionalities in Python. It is also an open-source package which is free to use. Scikit-learn is designed for Data Mining and Machine Learning. Since Python is a widely-used language, it is more likely to be implemented in various applications. Scikit-learntrains models faster and sometimes more accurately due to how Python stores the data as matrices.

Feature	Caret	Python
Stratified Sampling	caret::createDataPartion() Simple partition using stratified sampling, it creates train or test first, then the other by a specified ratio	sklearn.model_selection. train_test_split() Split arrays or matrices into random train and test subsets. Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes [3].
Bootstrap resampling	caret::createResample() Multiple partitions with replacement. Can be used for model hyperparameter tuning.	sklearn.utils.resample() Resample arrays or sparse matrices in a consistent way. The default strategy implements one step of the bootstrapping procedure. Input can be arrays, lists, dataframes or scipy sparse matrices with consistent first dimension [3].
knn imputation	recipes::step_knnimpute() Learns how to impute data for each resample	sklearn.impute.KNNImputer() Imputes missing values using the weighted or unweighted mean of the desired number of nearest neighbors. The imputers have an add_indicator parameter that marks the values that were missing, which might carry some information [3].
centering	recipes::step_center() Learns how to center numeric data for each resample	sklearn.preprocessing.scale() Centering and scaling are combined in one process and transform the data to center it by removing the mean value of each feature [3].
Model tuning	caret::train() Resample training, can use bootstrapping, which can be defined in	sklearn.model_selection.GridSearchCV() Grid search with cross-validation. The performance of the selected hyper-

	trControl parameter to varify the option of the training process. Then evaluate the model performance by hyperparameter tuning, choose the optimal model with the corresponding hyperparameter settings [5].	parameters and trained model is then measured on a dedicated evaluation set that was not used during the model selection step. Finally select the optimal model [3].
bag imputation	recipes::step_bagimpute() Learn how to impute data from bagged tree model, computational expensive than knn imputation [5].	sklearn.impute.IterativeImputer() Learn from models each feature with missing values as a function of other features and uses that estimate for imputation. BaysianRidge (liner), DecisionTreeRegressor (non-linear), ExtraTreesRegressor (similar to missForest) and KneighborRegressor (KNN) are available to choose from to do round-robin regression,treating every variable as an output inturn [3].
mode imputation	recipes::step_modeimpute() Impute nominal data using the most common value [5].	sklearn.impute.SimpleImputer() with “most_frequent” parameter, fill missing values by the most frequent in the same variable, can be used both numeric and categorical data [3].
median imputation	recipes::step_medianimpute() replace missing values by the median value of the column, only on numeric data [5]	sklearn.impute.SimpleImputer() With “median” parameter, replace missing values using the median along each column. Can only be used with numeric data [3].
mean imputation	recipes::step_meanimpute() replace missing values by the mean value of the column, only on numeric data [5].	sklearn.impute.SimpleImputer() with “mean” parameter, replace missing values using the mean along each column. Can only be used with numeric data [3].
remove missing	recipes::step_naomit() Removes any rows with missing data z [5].	pandas.DataFrame.dropna() Removes rows with missing value No such feature in sk-learn package [3].
scaling	recipes::step_scale() Normalize numeric data to have a standard deviation of one [5].	sklearn.preprocessing.scale() Center to the mean and component wise scale to unit variance [3].
BoxCox	recipes::step_BoxCox() Transforms the response variable while another method, the Box-Tidwell	sklearn.preprocessing.power_transform() With method = ‘box-cox’ Does exactly the same as in caret, not accepting o or negative value [3].

	transformation, was created to estimate transformations of predictor data [5].	
YeoJohnson	recipes::step_YeoJohnson() Same method as BoxCox but also can handle zero and negative values, and computational more expensive than BoxCox [5].	sklearn.preprocessing.power_transform() With method = 'yeo-johnson' as default parameter setting, does exactly the same as in caret [3].
PCA	recipes::step_pca() Transforms a group of variables and produces a new set of artificial features to capture the max amount of information in the original variables. Variables need to be standardized, centering and scaling. Threshold can be defined as a number of variables [5].	sklearn.decomposition.SparsePCA() Finds the set of sparse components that can optimally reconstruct the data. The amount of sparseness is controllable by the coefficient of the L1 penalty, given by the parameter alpha. Can take array and it will return an array. Y can be ignored [3].
PLS	recipes::step_pls() Partial least squares feature extraction that convert numeric data into one or more new dimensions. A supervised analysis needs outcome data, and sparsity can be encouraged [5].	sklearn.pls.PLSRegression() PLSRegression inherits from PLS with mode="A" and deflation_mode="regression". Also known PLS2 or PLS in case of one dimensional response. It maximizes both the correlations between the scores and the intra-block variances [3].
date expansions	recipes::step_date() Converts date data into one or more factor or numeric variables without replacing, unless recipes::step_rm() is called [5].	pandas.to_datatime() then use dt.hour, dt.weekday_name, dt.weekday, dt.dayofyear etc. after the datatime column., to transform it manually. Sklearn doesn't have this builtin fearture [3].
umap	recipes::step_umap() Supervised and unsupervised uniform manifold approximation and projection (UMAP). Dimension reduction [4].	umap.UMAP().fit() or umap.UMAP().fit_transform() Dimension reduction technique that can be used for visualisation similarly to t-SNE, but also for general non-linear dimension reduction. Install depends upon scikit-learn [3].
weight of evidence transformation	recipes::step_woe() transform nominal data into its numerical transformation based on weights of evidence against a binary outcome [4].	category_encoders.woe.WOEEncoder() Woe is a commonly used target-based encoder in credit scoring. It's a measure of the "strength" of a grouping for separating good and bad risk as default. Binary only [3].

dummy encoding	recipes::step_dummy() Converts nominal variable to a set of binary variables. Each nominal variable will have nlevels-1 binary variables. Output is more than 1 column if not binary. Converts ordinal predictors to a set of numerical variables that represent its order (and order squared, cubed etc as a set of orthogonal polynomials). Not working on high cardinality variables [5].	sklearn.preprocessing.OneHotEncoder() with drop='first'. Encode categorical features as a one-hot numeric array. The input to this transformer should be an array-like of integers or strings. Output is in one column, but it's an array list. Can't handle Y [3]. sklearn.preprocessing.OrdinalEncoder() The features are converted to ordinal integers. This results in a single column of integers (0 to n_categories - 1) per feature. The input to this transformer should be an array-like of integers or strings [3]. learn.preprocessing.labelEncoder() Is used to label outcome variable [3].
one hot binary encoding	recipes::step_dummy() one_hot=TRUE Converts nominal variable to a set of binary variables. Each nominal variable will have nlevels binary variables. Output is more than one column [5]	sklearn.preprocessing.OneHotEncoder() Encode categorical features as a one-hot numeric array. The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. It creates a binary column for each category and returns a sparse matrix or dense array. Can't handle Y [3].
hash encoding	recipes::step_texthash() Transform a text variable into a new set of numerical variables. Done by using MurmurHash3 method, computation efficient [4].	sklearn.feature_extraction.FeatureHasher ()This turns sequences of symbolic feature names (strings) into scipy.sparse matrices, using a hash function to compute the matrix column corresponding to a name. Only 1 column with matrix. Hash function employed 32-bit MurmurHash3 method , computational efficient [3].
Detecting / dealing with near-zero-variance of predictors	recipes::step_nzv() One simple step that will potentially remove variables that are highly sparse and unbalanced. Can set freqCut and uniqueCut for the cutoff threshold [4].	sklearn.feature_selection.VarianceThresh old() Feature selector that removes all low-variance features. Can use for unsupervised learning. Allow NaN, threshold parameter default as 0. It can take array and returns array [3].
Detecting / dealing with linear combinations of predictors	recipes::step_lincomb() Remove numeric variables that have linear combinations between them. This algorithm might need to apply more than one time [4].	sklearn.decomposition.PCA() Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. Doesn't support sparse data. Takes array and return to its origin space. Can ignore Y

		by set y=NONE, input data need to be centered [3].
Detecting / dealing with high correlation of predictors	recipes::step_corr() Remove variables that have large absolute correlations with other variables less than threshold. Unique value column ignored, if variable missing not much and inadequate value is chosen, will be ignored [4].	There is no such function can handle this kind of feature, we can use corr() to calculate the correlation matrix, then set a threshold and create a function manully to drop one of the highly correlated column [3].
ICA	recipes::step_ICA() A transformation of a group of variables that produces a new set of artificial features or components. It assumes that the variables are mixtures of a set of distinct, non-Gaussian signals and attempts to transform the data to isolate these signals. Center and scale needed priorly [4].	sklearn.decomposition.FastICA() is used to decompose a multivariate dataset in a set of successive orthogonal components that explain a maximum amount of the variance. It can take array and will return array [3].
Dealing with novel levels of nominal predictors in unseen data	recipes::step_tokenize() Split a character string into smaller parts as a token list for further analyse. It won't convert them to number until further step taken. Now we can employ hash encodeing to convert to numerics. Text encoding can handle novel levels without troubles [4].	sklearn.feature_extraction.text.CountVectorizer() Convert a collection of text documents to a matrix of token counts. If no feature selection analyse provided, number of features will equal to vocabulary size. Now we can apply hash encoding to convert them to numeric. This text encoder pipeline can handle novel levels without troubles [3].
Transforming nominal variables to binary	recipes::step_dummy() Converts nominal variable to a set of binary variables. Each nominal variable will have nlevels-1 binary variables. Output is more than 1 column if not binary. Either One_hot=TRUE/FALSE. Dummy encoding can also deal with ordinal variables but won't be binary [4].	sklearn.preprocessing.OneHotEncoder() Encode categorical features as a one-hot numeric array. The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. It creates a binary column for each category and returns a sparse matrix or dense array. Can't handle Y [3].
Documentation of the characteristics of available methods	More packages required to install than scikit-learn before training, but caret has more methods available than scikit-learn. Hyperparameter tuning can be done within the caret::train(), by specify a trControl() inside, with bootstrapping. Switch between the methods to train is easier, easy syntax. Resampling and compare_models make model compare easier. Struggled with large dataset, cannot train different models and	Less packages required before training. Less methods available, but still a good coverage through supervised learning (regression and classification) and unsupervised learning (e.g. clustering, PCA etc.). It also well cover hyperparameter tuning and dataset transformation, includes preprocessing, and feature engineering. Gooding handling on large dataset and text. Its training methods are easier for starter to handle without much

	different datasets at a same time. Not good for handling text, image and deep learning [2].	statistic background. Designed for machine learning, training speed is faster and can handle intensive computation more efficient as it stores data as matrices. But its API is less flexible than caret [3].