Categorical Encoding							
Encoding Method	**Description**	**Advantages**	**Disadvantages**	**Observations**	**Numpy/Pandas**	**Scikit-learn**	**Feature-Engine**

One Hot Encoding	One hot encoding, consists in encoding each categorical variable with different boolean variables (also called dummy variables) which take values 0 or 1, indicating if a category is present in an observation.
	Encoding into K or K-1 binary variable
		• Straightforward to implement
		• Makes no assumption about the distribution or categories of the categorical variable
		• Keeps all the information of the categorical variable
		• Suitable for linear models
			• Expands the feature space
			• Does not add extra information while encoding
			• Many dummy variables may be identical, introducing redundant information
				Most machine learning algorithms, consider the entire data set while being fit. Therefore, encoding categorical variables into k - 1 binary variables, is better, as it avoids introducing redundant information.
				
				Exception: One hot encoding into k dummy variables
				
				• when building tree based algorithms
				• when doing feature selection by recursive algorithms
				• when interested in determine the importance of each single category
					• K Variables: tmp = pd.get_dummies(X_train)
					• K-1 Variables: tmp = pd.get_dummies(X_train, drop_first=True)
						• from sklearn.preprocessing import OneHotEncoder
						• encoder = OneHotEncoder(categories='auto',
						drop = 'first', # to return k-1, use drop=false to return k dummies
						sparse=False,
						handle_unknown='error')
							• from feature_engine.categorical_encoders import OneHotCategoricalEncoder
							• ohe_enc = OneHotCategoricalEncoder(top_categories=None,
							variables=['sex', 'embarked'], # we can select which variables to encode
							drop_last=True) # to return k-1, false to return k

One Hot Encoding of Frequent/Top Categories	Performing one hot encoding, only considering the most frequent categories	• Straightforward to implement
		• Does not require hrs of variable exploration
		• Does not expand massively the feature space
		• Suitable for linear models
			• Does not add any information that may make the variable more predictive
			• Does not keep the information of the ignored labels
				The number of top variables is set arbitrarily. It could be 15, 10 or 5 as well. This number can be chosen arbitrarily or derived from data exploration.	Check Notebook	NA	• from feature_engine.categorical_encoders import OneHotCategoricalEncoder
							• ohe_enc = OneHotCategoricalEncoder(top_categories=10,
							# you can change this value to select more or less variables
							# we can select which variables to encode
							variables=['Neighborhood', 'Exterior1st', 'Exterior2nd'],drop_last=False)

Integer Encoding	Integer encoding consist in replacing the categories by digits from 1 to n (or 0 to n-1, depending the implementation), where n is the number of distinct categories of the variable.	• Straightforward to implement • Does not expand the feature space	• Does not capture any information about the categories labels • Not suitable for linear models.	• The numbers are assigned arbitrarily. • This encoding method allows for quick benchmarking of machine learning models.	Check Notebook	• from sklearn.preprocessing import LabelEncoder • le = LabelEncoder() • le.fit(X_train['Neighborhood']) **Note:** The LabelEncoder works one variable at the time. For encoding all the variables check notebook.	• from feature_engine.categorical_encoders import OrdinalCategoricalEncoder • ordinal_enc = OrdinalCategoricalEncoder(encoding_method='arbitrary', variables=['Neighborhood', 'Exterior1st', 'Exterior2nd']) • ordinal_enc.fit(X_train)

Count or frequency encoding	• Categories are replaced by the count or percentage of observations that show that category in the dataset.
	• Captures the representation of each label in a dataset
	• Very popular encoding method in Kaggle competitions
	• Assumption: the number observations shown by each category is predictive of the target.
		• Straightforward to implement
		• Does not expand the feature space
		• Can work well enough with tree based algorithms
			• Not suitable for linear models
			• Does not handle new categories in test set automatically
			• If 2 different categories appear the same amount of times in the dataset, that is, they appear in the same number of observations, they will be replaced by the same number: may lose valuable information.
					Check Notebook	NA	• from feature_engine.categorical_encoders import CountFrequencyCategoricalEncoder
							• count_enc = CountFrequencyCategoricalEncoder(encoding_method='count',
							# to do frequency ==> encoding_method='frequency'
							variables=['Neighborhood', 'Exterior1st', 'Exterior2nd'])
							• count_enc.fit(X_train)

Target guided encodings - Ordered ordinal encoding	Categories are replaced by integers from 1 to k, where k is the number of distinct categories in the variable, but this numbering is informed by the mean of the target for each category.	• Straightforward to implement
		• Does not expand the feature space
		• Capture information within the category, therefore creating more predictive features
		• Creates monotonic relationship between categories and target
		Note: Monotonic does not mean strictly linear. Monotonic means that it increases constantly, or it decreases constantly.
		Specific to WOE:
		• It orders the categories on a "logistic" scale which is natural for logistic regression
		• The transformed variables can then be compared because they are on the same scale. Therefore, it is possible to determine which one is more predictive.
			• May lead to over-fitting
			• Difficult to implement together with cross-validation with current libraries
				• the encoding is guided by the target, and
				• they create a monotonic relationship between the variable and the target.
				Monotonicity
				A monotonic relationship is a relationship that does one of the following:
				• (1) as the value of one variable increases, so does the value of the other variable; or
				• (2) as the value of one variable increases, the value of the other variable decreases
				• These methods can be also used on numerical variables, after discretisation. This creates a monotonic relationship between the numerical variable and the target, and therefore improves the performance of linear models
				• Replacing categorical labels with Pandas code and this method will generate missing values for categories present in the test set that were not seen in the training set. Therefore it is extremely important to handle rare labels before-hand.
					Check Notebook	NA	• from feature_engine.categorical_encoders import OrdinalCategoricalEncoder
							• ordinal_enc = OrdinalCategoricalEncoder(encoding_method='ordered',
							# NOTE that we indicate ordered in the encoding_method, otherwise it assings numbers arbitrarily
							variables=['Neighborhood', 'Exterior1st', 'Exterior2nd'])

Target guided encodings - Mean encoding	Mean encoding implies replacing the category by the average target value for that category.	Same as Ordered Ordinal Encoding	Same as Ordered Ordinal Encoding	Same as Ordered Ordinal Encoding	Check Notebook	NA	• from feature_engine.categorical_encoders import MeanCategoricalEncoder • mean_enc = MeanCategoricalEncoder(variables=['cabin', 'sex', 'embarked']) • mean_enc.fit(X_train, y_train)
Target guided encodings - Probability Ratio Encoding	For each category, we calculate the mean of target=1, that is the probability of the target being 1 (P(1)), and the probability of the target=0 (P(0)). And then, we calculate the ratio P(1)/P(0), and replace the categories by that ratio. **Note:** These encoding is suitable for classification problems only, where the target is binary	Same as Ordered Ordinal Encoding	Same as Ordered Ordinal Encoding	Same as Ordered Ordinal Encoding	Check Notebook	NA	• from feature_engine.categorical_encoders import WoERatioCategoricalEncoder • ratio_enc = WoERatioCategoricalEncoder(encoding_method = 'ratio', variables=['cabin', 'sex', 'embarked']) • ration_enc.fit(X_train, y_train)
Target guided encodings - Weight of evidence	WoE = ln (Distribution of Goods / Distribution of bads) WoE = ln (p(1) / p(0)) **Note:** WoE is well suited for Logistic Regression	Same as Ordered Ordinal Encoding	Same as Ordered Ordinal Encoding	Same as Ordered Ordinal Encoding	Check Notebook	NA	• from feature_engine.categorical_encoders import WoERatioCategoricalEncoder • woe_enc = WoERatioCategoricalEncoder(encoding_method = ' woe', variables=['cabin', 'sex', 'embarked']) • woe_enc.fit(X_train, y_train)

Rare Label Encoding	Rare labels are those that appear only in a tiny proportion of the observations in a dataset
	Scenario for re-grouping:
	• One predominant category
	• A small number of categories
	• High cardinality
		• Grouping categories into rare for variables that show low cardinality may or may not improve model performance, however, we tend to re-group them into a new category to smooth model deployment.
		• Grouping categories into rare for variables with high cardinality, tends to improve model performance as well.
				• Grouping infrequent labels or categories under a new category called 'Rare' or 'Other' is the common practice in machine learning for business
				• Rare labels should be identified in the training set only.
					Check Notebook	NA	• from feature_engine.categorical_encoders import RareLabelCategoricalEncoder
							• rare_encoder = RareLabelCategoricalEncoder(
							tol=0.05, # minimal percentage to be considered non-rare
							n_categories=4, # minimal number of categories the variable should have to re-group rare categories
							variables=['Neighborhood', 'Exterior1st', 'Exterior2nd',
							'MasVnrType', 'ExterQual', 'BsmtCond'] # variables to re-group)