	from imblearn. over_sampling import	Description / Steps
Random Oversampling	RandomOverSampler	- extract at random observation of the majority class until a certain balancing ratio is reached - it is a naive technique (requires no assumption)
SMOTE	SMOTE	1) Isolate the minority class 2) Determine how many new samples need to be generated and select from which original samples the new one will be generated 3) For each chosen minority observation, find k-nearest neighbors 4) Determine the L2-distance between the observation and k-nearest neighbors 5) The distance has to be multiplied by a factor (a random number [0;1]) formula: new sample = or_smpl - f * (or_smpl - minor_nbr)
SMOTE-NC (Nominal Continuous)	SMOTENC(categorical_features=[n, ...])	- extands the functunality of SMOTE to categorical variables (ADASYN cannot do this) having a categorical feature (column): 1) from all numerical features, calculate median(sum(std(all numerical features))) 2) proceed with SMOTE putting this median value as an L2-distance between feature values 3) when putting the new-generated sample to the place, assign the most frequent categorical feature values to it
Borderline SMOTE	BorderlineSMOTE	- create new sample only from the original observations that are the closets to the borderline with the majority class 1) fit KNN with *all* dataset 2) find and ignore observations from the minority class which K-ns belongs to the majority class (noise and irrelevant) 3) find and ignore observations from the minority class if most of their neighbors are from the minority class (safe and easy to classify) 4) Select the observations of the minority class if most of their neighbors are from the majority class 5) fit KNN to the minority class observations Next - division into 2 variants
Borderline SMOTE (variant 1)	BorderlineSMOTE(kind='borderline-1',)	6) as a regular SMOTE: a new sample to be between original observations of the *minority* class

Borderline SMOTE (variant 2)	BorderlineSMOTE(kind='borderline-2',)	6) a new sample to be between original observations of the *majority* class
K-Means SMOTE	KMeansSMOTE	- for clusters 1) determine clusters -- K-means algorithms to the whole dataset 2) select clusters where the % of the minority classes is above a threshold (typically 0.5) 3) weight the cluster -- how many new samples to create in each cluster L2mean = Mean L2 between minority observations density = (number of minority observations / L2mean) * number of feature sparsity = 1 / density cluster sparsity = sparsity / sum() 4) calculate the number of the synthetic samples to be generated for each cluster: g(i) = cs(i)*G, where cs(i) = cluster sparsity, G = total num of samples to generate, g(i) = num of samples to generate from cluster i
ADASYN	ADASYN	- synthetic data is more generated from *all* observations that are harder to classify (this is the main difference beetwen this one and SMOTE) 1) determine the balancing ratio: X(minority) / X(majority) 2) determine the number og samples to generate: G = (X(majority) - X(minority)) * factor 3) train KNN using the *entire* dataset to find closest K-ns for each observation of minority class 4) determine the weighting r: D/K, where D = neighbors from the majority class, K = neighbors 5) normalize r: r/sum(r-s) 6) calculate the number of the synthetic samples to be generated for each observation of minority class: g(i) = r(i) * G, where r(i) = weight for observation i, G = total num of samples to generate, g(i) = num of samples to generate from observation i 7) for each minority class x(i) generate g(i) synthetic samples formula: new sample = or_smpl - f * (or_smpl - *any*_nbr)