Type	**Algorithm**	**Simple Meaning**	**Python Module**	**Model Creation**	**Key Parameters**
Regression	Linear Regression	Predict number using a straight line	sklearn.linear_model.LinearRegression	model = LinearRegression()	fit_intercept=True, normalize=False, copy_X=True
	Multiple Linear Regression	Predict number using many features	sklearn.linear_model.LinearRegression	model = LinearRegression()	Same as Linear Regression
	Polynomial Regression	Fit a curve instead of line	sklearn.preprocessing.PolynomialFeatures + LinearRegression	poly = PolynomialFeatures(degree=2)model = LinearRegression()	degree=2/3/..., include_bias=True
	Ridge Regression	Linear + penalty to avoid overfitting	sklearn.linear_model.Ridge	model = Ridge()	alpha=1.0, solver='auto', fit_intercept=True
	Lasso Regression	Linear + remove unimportant	sklearn.linear_model.Lasso	model = Lasso()	alpha=1.0, max_iter=1000, fit_intercept=True

Type	Algorithm	Simple Meaning	Python Module	Model Creation	Key Parameters
		features			
	Elastic Net	Ridge + Lasso combination	sklearn.linear_model.ElasticNet	model = ElasticNet()	alpha=1.0, l1_ratio=0.5, max_iter=1000
	SVR	Predict numbers using SVM	sklearn.svm.SVR	model = SVR()	kernel='rbf', C=1.0, epsilon=0.1, gamma='scale'
	Decision Tree Regression	Predict using if-else rules	sklearn.tree.DecisionTreeRegressor	model = DecisionTreeRegressor()	max_depth=None, min_samples_split=2
	Random Forest Regression	Average many trees	sklearn.ensemble.RandomForestRegressor	model = RandomForestRegressor()	n_estimators=100, max_depth=None
	Gradient Boosting Regression	Sequentially reduce error	sklearn.ensemble.GradientBoostingRegressor	model = GradientBoostingRegressor()	n_estimators=100, learning_rate=0.1, max_depth=3

Type	**Algorithm**	**Simple Meaning**	**Python Module**	**Model Creation**	**Key Parameters**
	XGBoost Regression	Optimized gradient boosting	xgboost.XGBRegressor	model = XGBRegressor()	n_estimators=100, learning_rate=0.1, max_depth=3
	KNN Regression	Average of nearest neighbors	sklearn.neighbors.KNeighborsRegressor	model = KNeighborsRegressor()	n_neighbors=5, weights='uniform'
	Bayesian Regression	Probabilistic prediction	sklearn.linear_model.BayesianRidge	model = BayesianRidge()	alpha_1=1e-6, lambda_1=1e-6
Classification	Logistic Regression	Predict category (0/1)	sklearn.linear_model.LogisticRegression	model = LogisticRegression()	penalty='l2', C=1.0, solver='lbfgs'
	KNN Classifier	Class based on neighbors	sklearn.neighbors.KNeighborsClassifier	model = KNeighborsClassifier()	n_neighbors=5, weights='uniform'
	SVM Classifier	Max margin separation	sklearn.svm.SVC	model = SVC()	kernel='rbf', C=1.0, gamma='scale'

Type	Algorithm	Simple Meaning	Python Module	Model Creation	Key Parameters
	Decision Tree Classifier	If-else rules	sklearn.tree.DecisionTreeClassifier	model = DecisionTreeClassifier()	max_depth=None, min_samples_split=2
	Random Forest Classifier	Many trees → majority vote	sklearn.ensemble.RandomForestClassifier	model = RandomForestClassifier()	n_estimators=100, max_depth=None
	Gradient Boosting Classifier	Sequential error reduction	sklearn.ensemble.GradientBoostingClassifier	model = GradientBoostingClassifier()	n_estimators=100, learning_rate=0.1
	XGBoost Classifier	Optimized boosting	xgboost.XGBClassifier	model = XGBClassifier()	n_estimators=100, learning_rate=0.1, max_depth=3
	Naive Bayes	Probabilistic classifier	sklearn.naive_bayes.GaussianNB	model = GaussianNB()	var_smoothing=1e-9
	AdaBoost	Boost weak classifiers	sklearn.ensemble.AdaBoostClassifier	model = AdaBoostClassifier()	n_estimators=50, learning_rate=1.0

Type	**Algorithm**	**Simple Meaning**	**Python Module**	**Model Creation**	**Key Parameters**
	LDA	Linear combination of features	sklearn.discriminant_analysis.LinearDiscriminantAnalysis	model = LinearDiscriminantAnalysis()	solver='svd', shrinkage=None
	QDA	Like LDA but quadratic	sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis	model = QuadraticDiscriminantAnalysis()	reg_param=0.0
	LightGBM	Fast boosting	lightgbm.LGBMClassifier	model = LGBMClassifier()	n_estimators=100, learning_rate=0.1
	CatBoost	Boosting handles categorical	catboost.CatBoostClassifier	model = CatBoostClassifier()	iterations=1000, learning_rate=0.1, depth=6
Clustering	K-Means	Group data into k clusters	sklearn.cluster.KMeans	model = KMeans()	n_clusters=8, init='k-means++', max_iter=300
	Hierarchical Clustering	Build tree of clusters	sklearn.cluster.AgglomerativeClustering	model = AgglomerativeClustering()	n_clusters=2, linkage='ward'

Type	**Algorithm**	**Simple Meaning**	**Python Module**	**Model Creation**	**Key Parameters**
	DBSCAN	Dense regions, ignore noise	sklearn.cluster.DBSCAN	model = DBSCAN()	eps=0.5, min_samples=5
	Mean Shift	Find clusters by shifting centroids	sklearn.cluster.MeanShift	model = MeanShift()	bandwidth=None, bin_seeding=False
	Gaussian Mixture Model	Soft clustering with probabilities	sklearn.mixture.GaussianMixture	model = GaussianMixture()	n_components=1, covariance_type='full'
	Affinity Propagation	Clustering by message passing	sklearn.cluster.AffinityPropagation	model = AffinityPropagation()	damping=0.5, max_iter=200
	Spectral Clustering	Graph-based clustering	sklearn.cluster.SpectralClustering	model = SpectralClustering()	n_clusters=8, affinity='rbf'
	OPTICS	Like DBSCA	sklearn.cluster.OPTICS	model = OPTICS()	min_samples=5,

Type	**Algorithm**	**Simple Meaning**	**Python Module**	**Model Creation**	**Key Parameters**
		N for varying density			max_eps=np.inf
	Birch	Efficient for large datasets	sklearn.cluster.Birch	model = Birch()	threshold=0.5, branching_factor=50
	MiniBatch K-Means	Fast K-Means with batches	sklearn.cluster.MiniBatchKMeans	model = MiniBatchKMeans()	n_clusters=8, batch_size=100