**Generalized Linear Models:**

**Linear Regression:**

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

**Logistic Regression:**

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

**Ridge Regression:**

from sklearn.linear\_model import Ridge

model = Ridge(alpha=1.0)

**Lasso Regression:**

from sklearn.linear\_model import Lasso

model = Lasso(alpha=1.0)

**ElasticNet:**

from sklearn.linear\_model import ElasticNet

model = ElasticNet(alpha=1.0, l1\_ratio=0.5)

**Support Vector Machines (SVM):**

**SVM Classifier:**

from sklearn.svm import SVC

model = SVC(kernel='rbf')

**SVM Regressor**:

from sklearn.svm import SVR

model = SVR(kernel='linear')

**Nearest Neighbors:**

#### k-Nearest Neighbors (k-NN):

from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor

knn\_classifier = KNeighborsClassifier(n\_neighbors=5)

knn\_regressor = KNeighborsRegressor(n\_neighbors=5)

**Decision Trees:**

**Decision Tree Classifier:**

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()

**Decision Tree Regressor:**

from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()

**Ensemble Methods:**

**Random Forests:**

from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor

rf\_classifier = RandomForestClassifier(n\_estimators=100)

rf\_regressor = RandomForestRegressor(n\_estimators=100)

**Gradient Boosting:**

from sklearn.ensemble import GradientBoostingClassifier,GradientBoostingRegressor

gb\_classifier = GradientBoostingClassifier(n\_estimators=100)

gb\_regressor = GradientBoostingRegressor(n\_estimators=100)

**AdaBoost:**

from sklearn.ensemble import AdaBoostClassifier, AdaBoostRegressor

adb\_classifier = AdaBoostClassifier(n\_estimators=100)

adb\_regressor = AdaBoostRegressor(n\_estimators=100)

**Extra Trees Classifier/Regressor:**

from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor

et\_classifier = ExtraTreesClassifier(n\_estimators=100)

et\_regressor = ExtraTreesRegressor(n\_estimators=100)

**Naive Bayes:**

**Gaussian Naive Bayes:**

from sklearn.naive\_bayes import GaussianNB

model = GaussianNB()

**Multinomial Naive Bayes:**

from sklearn.naive\_bayes import MultinomialNB

model = MultinomialNB()

**Bernoulli Naive Bayes:**

from sklearn.naive\_bayes import BernoulliNB

model = BernoulliNB()

**Neural Network Models:**

**Multi-layer Perceptron (MLP) Classifier/Regressor:**

from sklearn.neural\_network import MLPClassifier, MLPRegressor

mlp\_classifier = MLPClassifier(hidden\_layer\_sizes=(100, ), max\_iter=1000)

mlp\_regressor = MLPRegressor(hidden\_layer\_sizes=(100, ), max\_iter=1000)

**Unsupervised Learning Models:**

**Clustering:**

from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering, MeanShift

kmeans = KMeans(n\_clusters=3)

dbscan = DBSCAN(eps=0.5, min\_samples=5)

agg\_clustering = AgglomerativeClustering(n\_clusters=3)

meanshift = MeanShift()

**Dimensionality Reduction:**

from sklearn.decomposition import PCA, TruncatedSVD, FastICA

pca = PCA(n\_components=2)

tsvd = TruncatedSVD(n\_components=2)

ica = FastICA(n\_components=2)

**Model Selection and Evaluation:**

**Cross-validation techniques:**

from sklearn.model\_selection import cross\_val\_score, GridSearchCV, RandomizedSearchCV

scores = cross\_val\_score(model, X, y, cv=5)

grid\_search = GridSearchCV(estimator=model, param\_grid={}, cv=5)

random\_search = RandomizedSearchCV(estimator=model, param\_distributions={}, cv=5)

**Model evaluation metrics:**

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

**Preprocessing and Utilities:**

**Feature preprocessing:**

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder, LabelEncoder

scaler = StandardScaler()

minmax\_scaler = MinMaxScaler()

onehot\_encoder = OneHotEncoder()

label\_encoder = LabelEncoder()

**Imputation:**

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')

**Model Pipelines:**

from sklearn.pipeline import Pipeline, FeatureUnion

pipeline = Pipeline(steps=[('scaler', StandardScaler()), ('model', model)])

feature\_union = FeatureUnion([('pca', PCA()), ('tsvd', TruncatedSVD())])