01 LINEAR REGRESSION

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

02 LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(C=1.0, penalty='l2')
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

Hyperparameters:

- C (Inverse of regularization strength, default=1.0)
- penalty (Regularization term, default='l2')

03 DECISION TREES

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(max_depth=None, min_samples_split=2,
min_samples_leaf=1)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

Hyperparameters:

- max_depth (Maximum depth of the tree, default=None)
- min_samples_split (Minimum number of samples required to split an internal node, default=2)
- min_samples_leaf (Minimum number of samples required to be at a leaf node, default=1)

04 RANDOM FOREST

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, max_depth=None,
min_samples_split=2, min_samples_leaf=1)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

Hyperparameters:

- n_estimators (Number of trees in the forest, default=100)
- max_depth (Maximum depth of the trees, default=None)
- min_samples_split (Minimum number of samples required to split an internal node, default=2)
- min_samples_leaf (Minimum number of samples required to be at a leaf node, default=1)

05 SUPPORT VECTOR MACHINES

```
from sklearn.svm import SVC
model = SVC(C=1.0, kernel='rbf')
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

Hyperparameters:

- C (Regularization parameter, default=1.0)
- kernel (Kernel function, default='rbf')

06 K-NEAREST NEIGHBORS

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5,
weights='uniform')
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

Hyperparameters:

- n_neighbors (Number of neighbors, default=5)
- weights (Weight function used in prediction, default='uniform')

07 K-MEANS CLUSTERING

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=8, init='k-means++')
model.fit(X_train)
predictions = model.predict(X_test)
```

Hyperparameters:

- n_clusters (Number of clusters, default=8)
- init (Method for initialization, default='k-means++')

08 PRINCIPAL COMPONENT ANALYSIS

```
from sklearn.decomposition import PCA
model = PCA(n_components=None)
model.fit(X_train)
transformed_data = model.transform(X_test)
```

Hyperparameters:

- n_components (Number of components to keep, default=None)

09 GRADIENT BOOSTING (XGBOOST)

```
from xgboost import XGBClassifier
model = XGBClassifier(n_estimators=100, learning_rate=0.1,
max_depth=3)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

Hyperparameters:

- n_estimators (Number of boosting rounds, default=100)
- learning_rate (Step size shrinkage to prevent overfitting, default=0.1)
- max_depth (Maximum depth of a tree, default=3)

10 NAIVE BAYES

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

11 CROSS-VALIDATION

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Creating a Random Forest model
rf_model = RandomForestClassifier(n_estimators=100,
max_depth=None, min_samples_split=2,
min_samples_leaf=1)

# Performing cross-validation
cv_scores = cross_val_score(rf_model, X, y, cv=5)

# Displaying cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Score:", np.mean(cv_scores))
```

In this example, we use cross_val_score to perform 5-fold cross-validation for a Random Forest classifier.

12 GRID SEARCH

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Creating a Random Forest model
rf_model = RandomForestClassifier()

# Defining the parameter grid for grid search
param_grid_rf = {
   'n_estimators': [50, 100, 200],
   'max_depth': [None, 10, 20],
   'min_samples_split': [2, 5, 10],
   'min_samples_leaf': [1, 2, 4]
}

# Creating a GridSearchCV object
grid_search_rf = GridSearchCV(rf_model, param_grid_rf, cv=5,
scoring='accuracy')

# Fitting the grid search to the data
grid_search_rf.fit(X, y)

# Displaying the best parameters and corresponding score
print("Random Forest - Best Parameters:", grid_search_rf.best_params_)
print("Random Forest - Best Score:", grid_search_rf.best_score_)
```