Supervised Algorithms Cheatsheet

1. Data Handling & Preprocessing

Pandas / IO

```
df = pd.read_csv("<path>")
df.head()
df.info()
df.describe()
df.columns
df.shape
df.drop(columns=[<col_list>], inplace=<True/False>)
df.rename(columns={<old>:<new>,...}, inplace=<True/False>)
```

Nulls & Masks

```
any(df["col"].isna())
any(df["col"].isnull())
mask = df["col"].isna()
df["col"].isnull().sum()
```

Fill / Impute

```
df["col"].fillna(value=<value>, inplace=<True/False>)

from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy=<"mean"/"median"/"most_frequent">)
X = imp.fit_transform(X)
```

Descriptive

```
mean = df["col"].mean()
median = df["col"].median()
mode = df["col"].mode()
std = df["col"].std()
qk = df["col"].quantile(<k>)
```

Transformations

```
trans = np.log1p(df["col"])
orig = np.expm1(trans)          # inverse log1p
df["col"] = np.where(<condition>, <true_val>, <false_val>)
```

Scaling & Encoding

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import MaxAbsScaler, RobustScaler
X_std = StandardScaler().fit_transform(X)
X_min = MinMaxScaler().fit_transform(X)
X_maxabs = MaxAbsScaler().fit_transform(X)
X_robust = RobustScaler().fit_transform(X)

from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
X_ord = OrdinalEncoder().fit_transform(X_cat)
X_oh = OneHotEncoder(
handle_unknown=<"ignore"/"error">
).fit_transform(X_cat).toarray()
```

Polynomial

```
from sklearn.preprocessing import PolynomialFeatures
X_poly = PolynomialFeatures(
degree=<d>, include_bias=<True/False>
).fit_transform(X)
```

Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=<float>, random_state=<int>, stratify=<y_or_None>
)
```

2. Models (Syntactic)

Pipeline

```
from sklearn.pipeline import make_pipeline
pipe = make_pipeline(<step1>, <step2>, ...)
```

Linear & Logistic

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, LogisticRegressi

LinearRegression()
Ridge(alpha=<alpha>)
Lasso(alpha=<alpha>)

LogisticRegression(
    penalty=<"l1"/"l2"/"elasticnet"/"none">,
    C=<float>,
    solver=<"liblinear"/"saga"/"lbfgs">,
    l1_ratio=<float_or_None>,
    random_state=<int>
)
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

DecisionTreeClassifier(
    criterion=<"gini"/"entropy">,
    max_depth=<int_or_None>,
    min_samples_split=<int>,
    min_samples_leaf=<int>,
    random_state=<int>
)

DecisionTreeRegressor(
    criterion=<"squared_error"/"friedman_mse"/"absolute_error">,
    max_depth=<int_or_None>,
    min_samples_split=<int>,
    random_state=<int>
)
```

Naive Bayes, KNN, SVM

```
GaussianNB()
MultinomialNB()

KNeighborsClassifier(n_neighbors=<int>, algorithm=<"auto"/"ball_tree"/"kd_tree">)

SVC(kernel=<"linear"/"poly"/"rbf"/"sigmoid">,
    C=<float>, degree=<int>, gamma=<"scale"/"auto">,
    probability=<True/False>, random_state=<int>)
```

Bagging & RandomForest

```
BaggingClassifier(estimator=<est>, n_estimators=<int>, max_samples=<float>,
                  max_features=<float>, random_state=<int>)

RandomForestClassifier(n_estimators=<int>, criterion=<"gini"/"entropy">,
                       max_depth=<int_or_None>, min_samples_split=<int>,
                       max_features=<"sqrt"/"log2"/<float>>, random_state=<int>)

BaggingRegressor(...), RandomForestRegressor(...)
```

Boosting

```
AdaBoostClassifier(
    estimator=DecisionTreeClassifier(...),
    n_estimators=<int>, learning_rate=<float>, random_state=<int>
)

AdaBoostRegressor(...)

GradientBoostingClassifier(
    n_estimators=<int>, learning_rate=<float>, max_depth=<int>,
    min_samples_split=<int>, min_samples_leaf=<int>,
    subsample=<float>, max_features=<"sqrt"/"log2">, random_state=<int>
)

GradientBoostingRegressor(...)

XGBClassifier(
    use_label_encoder=<False/True>, eval_metric=<"logloss">,
    n_estimators=<int>, max_depth=<int>, learning_rate=<float>,
    subsample=<float>, random_state=<int>
)

XGBRegressor(...)
```

Stacking

```
StackingClassifier(
    estimators=[("<name>", <estimator>), ...],
    final_estimator=<estimator>, cv=<int>, n_jobs=<int>
)

StackingRegressor(...)
```

General

```
model.fit(<X_train>, <y_train>)
y_pred = model.predict(<X_test>)
y_proba = model.predict_proba(<X_test>)[:, <class_index>]
score = model.score(<X_test>, <y_test>)
```

3. Validation & Hyperparam Search

K-Fold & CV

```
from sklearn.model_selection import KFold, cross_val_score, cross_validate

KFold(n_splits=<int>, shuffle=<True/False>, random_state=<int>)

cross_val_score(estimator=<model>,
X=<X>, y=<y>,
cv=<int_or_KFold>, scoring=<metric>)
```

Grid / Randomized Search

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

GridSearchCV(estimator=<model>,
             param_grid={<param>: [<values>], ...},
             cv=<int_or_KFold>,
             scoring=<metric>,
             n_jobs=<int>)

RandomizedSearchCV(estimator=<model>,
                   param_distributions={<param>: [<values>], ...},
                   n_iter=<int>, cv=<int>, scoring=<metric>, n_jobs=<int>)
```

4. Metrics & Visualization

Classification Metrics

```
from sklearn.metrics import (
  accuracy_score, precision_score, recall_score, f1_score,
  confusion_matrix, classification_report, roc_auc_score,
  roc_curve, average_precision_score
)

accuracy_score(<y_true>, <y_pred>)
precision_score(<y_true>, <y_pred>, average=<"binary"/"micro"/"macro"/"weighted">)
recall_score(<y_true>, <y_pred>)
f1_score(<y_true>, <y_pred>)
confusion_matrix(<y_true>, <y_pred>)
classification_report(<y_true>, <y_pred>)
```

Probabilistic / ROC

```
y_proba = model.predict_proba(<X_test>)[:, <class_index>]
roc_auc_score(<y_true>, <y_score>)
fpr, tpr, thr = roc_curve(<y_true>, <y_score>)
```

Regression Metrics

```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

r2_score(<y_true>, <y_pred>)
adjusted_r2 = 1 - (1 - r2)*(n - 1)/(n - p - 1)
mean_squared_error(<y_true>, <y_pred>)
mean_absolute_error(<y_true>, <y_pred>)
```

Visualization (Syntactic)

```
sns.heatmap(<conf_matrix>, annot=<True/False>, fmt="<fmt>", cmap=<"cmap">,
            xticklabels=<labels>, yticklabels=<labels>)

fpr, tpr, thr = roc_curve(<y_true>, <y_score>)
plt.plot(fpr, tpr); plt.plot([0,1],[0,1], linestyle="<ls>")

sns.regplot(x=<y_true>, y=<y_pred>, ci=<int_or_None>)
sns.scatterplot(x=<x>, y=<y>, hue=<col_or_None>)
plt.bar(<feature_names>, <coef_or_importance>)
```

5. Available Metric Strings (sklearn)

```
# Classification:
"accuracy", "precision", "recall", "f1",
"f1_micro", "f1_macro", "f1_weighted",
"precision_micro", "precision_macro", "precision_weighted",
"recall_micro", "recall_macro", "recall_weighted",
"roc_auc", "average_precision"

# Regression:
"r2",
"neg_mean_squared_error",
"neg_root_mean_squared_error",
"neg_mean_absolute_error",
"neg_median_absolute_error",
"explained_variance"
```

6. Miscellaneous (NumPy & Pandas)

```
# NumPy
np.sqrt(<x>), np.exp(<x>), np.log(<x>)
np.log1p(<x>), np.expm1(<x>)
np.mean(<x>), np.median(<x>), np.std(<x>), np.var(<x>)
np.max(<x>), np.min(<x>), np.argmax(<x>), np.argmin(<x>)
np.unique(<x>), np.dot(<a>,<b>)
np.linalg.inv(<A>), np.linalg.det(<A>)
np.corrcoef(<x>,<y>)
np.linspace(<start>,<stop>,<num>), np.arange(<start>,<stop>,<step>)
np.random.rand(<n>), np.random.randn(<n>), np.random.seed(<int>)

# Pandas quick
df.head(<n>), df.tail(<n>), df.sample(<n>)
df.isnull().sum(), df.dropna(axis=<0_or_1>)
df.groupby(<cols>).agg({<col>: <func>})
```