Scikit-learn (sklearn) Cheat Sheet

1. Library Structure Overview

sklearn is modular and organized as follows:

```
sklearn/
├──── datasets            # Sample data, loading utilities
├──── model_selection     # Splitting, cross-validation, hyperparam tuning
├──── preprocessing       # Scaling, encoding, imputation, feature engineering
├──── feature_selection   # Selecting informative features
├──── pipeline            # Chaining steps into workflows
├──── metrics             # Scoring and evaluation functions
├──── linear_model        # Linear/logistic regressions, elastic net, etc.
├──── ensemble            # RandomForest, GradientBoosting, etc.
├──── tree                # DecisionTreeClassifier/Regressor
├──── neighbors           # k-NN algorithms
├──── svm                 # Support Vector Machines
├──── naive_bayes         # Naive Bayes variants
├──── cluster             # Clustering (KMeans, DBSCAN, etc.)
├──── decomposition       # PCA, TruncatedSVD, etc.
├──── manifold            # Advanced dimensionality reduction
├──── compose             # ColumnTransformer, FeatureUnion
└──── inspection          # Model explanation (permutation_importance, etc.)
```

2. Typical Machine Learning Workflow

Step 1: Import Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

Step 2: Load Data

```
from sklearn import datasets
X, y = datasets.load_iris(return_X_y=True)
```

Or from DataFrame:

```
df = pd.read_csv('file.csv')
X = df.drop('target', axis=1).values
y = df['target'].values
```

Step 3: Split Data

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

Step 4: Preprocessing

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 5: Build & Train Model

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train_scaled, y_train)
```

Step 6: Predict & Evaluate

```
y_pred = clf.predict(X_test_scaled)
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

3. Core Modules and Tasks

A. Datasets

```
from sklearn.datasets import load_digits, fetch_openml
digits = load_digits()
```

```
# Or fetch larger datasets
X, y = fetch_openml('adult', as_frame=True, return_X_y=True)
```

B. Preprocessing/Feature Engineering

```
from sklearn.preprocessing import MinMaxScaler, LabelEncoder

# Scaling
scaler = MinMaxScaler()  # 0-1 scaling
X_scaled = scaler.fit_transform(X)

# Encoding
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)
```

- **Pipelines** for sequential transformations:

  ```
  from sklearn.pipeline import Pipeline
  pipe = Pipeline([
      ('scaler', StandardScaler()),
      ('clf', RandomForestClassifier())
  ])
  pipe.fit(X_train, y_train)
  ```

C. Model Selection

Task	Function	Example
Train/test split	`train_test_split`	As above
Cross-validation	`cross_val_score`	`cross_val_score(clf, X, y, cv=5)`
Grid search	`GridSearchCV`	
Random search	`RandomizedSearchCV`	

Example: Hyperparameter Search

```
from sklearn.model_selection import GridSearchCV
```

```
params = {'max_depth': [3, None], 'n_estimators': [10, 100, 200]}
gs = GridSearchCV(RandomForestClassifier(), params, cv=5)
gs.fit(X_train, y_train)
print("Best params:", gs.best_params_)
```

D. Main Model Families

Supervised Learning:

- **Classification:** `LogisticRegression, RandomForestClassifier, SVC, GradientBoostingClassifier, KNeighborsClassifier, GaussianNB, MLPClassifier`
- **Regression:** `LinearRegression, Ridge, Lasso, SVR, RandomForestRegressor, GradientBoostingRegressor`

Unsupervised Learning:

- **Clustering:** `KMeans, DBSCAN, AgglomerativeClustering, GaussianMixture`
- **Dimensionality Reduction:** `PCA, TruncatedSVD, NMF, TSNE, Isomap`
- **Feature Selection:** `SelectKBest, SelectFromModel, RFECV`

4. Evaluation Metrics

Classification:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score
```

Regression:

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

Confusion Matrix and Reports:

```
from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(y_true, y_pred))
print(classification_report(y_true, y_pred))
```

5. Advanced Tools

Pipelines & ColumnTransformer

- **Pipeline** for automated sequences:

  ```
  from sklearn.pipeline import make_pipeline
  pipe = make_pipeline(StandardScaler(), PCA(n_components=2), LogisticRegression())
  ```

- **ColumnTransformer** for feature-wise preprocessing:

  ```
  from sklearn.compose import ColumnTransformer
  from sklearn.impute import SimpleImputer
  numeric_features = ['age', 'income']
  categorical_features = ['gender', 'city']
  ct = ColumnTransformer([
      ('num', StandardScaler(), numeric_features),
      ('cat', OneHotEncoder(), categorical_features)
  ])
  X_trans = ct.fit_transform(df)
  ```

Model Export & Persistence

```
from joblib import dump, load
dump(clf, 'rf_model.joblib')
clf_loaded = load('rf_model.joblib')
```

Custom Transformers & Advanced Pipeline

```
from sklearn.base import BaseEstimator, TransformerMixin

class MyTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        # learn something from X
        return self
    def transform(self, X):
        # modify X
        return X
```

Model Interpretation

```
from sklearn.inspection import permutation_importance
result = permutation_importance(clf, X_test, y_test)
importances = result.importances_mean
```

- Plot `feature_importances_` for tree-based models.

6. Special Topics and Pro Tips

- **Imbalanced Data:** `from imblearn.over_sampling import SMOTE`
- **Probabilities:** Use `predict_proba` for estimated class probabilities.
- **Chaining:** You can chain transformers and classifiers for end-to-end automation.
- **Parallelism:** Many models accept `n_jobs=-1` for multicore training/testing.

7. Example: End-to-End Workflow

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.pipeline import Pipeline

# Load data
df = pd.read_csv('titanic.csv')
y = df['Survived']
X = df.drop(['Survived', 'Name', 'Ticket'], axis=1)

# Preprocessing
num_features = ['Age', 'Fare']
cat_features = ['Sex', 'Embarked']

preprocessor = ColumnTransformer([
    ('num', StandardScaler(), num_features),
    ('cat', OneHotEncoder(), cat_features)
])

# Build pipeline
pipe = Pipeline([
    ('prep', preprocessor),
    ('clf', GradientBoostingClassifier())
])

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

# Hyperparameter tuning
params = {'clf__n_estimators': [100,200], 'clf__learning_rate': [0.01,0.1]}
gs = GridSearchCV(pipe, params, cv=5)
gs.fit(X_train, y_train)

print("Best parameters:", gs.best_params_)
print("Test accuracy:", gs.score(X_test, y_test))
```

8. Useful Links and References

- Official documentation/tutorials: `scikit-learn.org`
- User Guide: Explanations and usage of each estimator and transformer
- Example Gallery: Dozens of annotated code examples
- Community cookbooks and notebooks

9. Summary Table: Common Scikit-learn Objects

Module	Key Classes/Functions	Usage
`model_selection`	`train_test_split`, `GridSearchCV`, `cross_val_score`	Splits, hyperparam search, CV
`preprocessing`	`StandardScaler`, `OneHotEncoder`, `LabelEncoder`, `SimpleImputer`	Prep/input wrangling
`metrics`	`accuracy_score`, `mean_squared_error`, `roc_auc_score`	Model evaluation
`ensemble`	`RandomForestClassifier`, `GradientBoostingClassifier`	Tree-based ensembles

`pipeline`	`Pipeline`, `make_pipeline`	Combine models & preprocessors
`compose`	`ColumnTransformer`	Parallel preprocessing per feature type
`inspection`	`permutation_importance`	Feature importances, explanation
`decomposition`	`PCA`, `TruncatedSVD`	Dimensionality reduction

10. Extra Advanced: Custom Scoring, Cross-validator, and Model Stacking

- **Custom scorer:**

```
from sklearn.metrics import make_scorer
scorer = make_scorer(f1_score, average='macro')
```

- **Nested cross-validation:**

```
from sklearn.model_selection import cross_val_score
cross_val_score(gs, X, y, cv=5)
```

- **Stack multiple models:**

```
from sklearn.ensemble import StackingClassifier
stack = StackingClassifier(
    estimators=[('lr', LogisticRegression()), ('rf', RandomForestClassifier())],
    final_estimator=GradientBoostingClassifier()
)
stack.fit(X_train, y_train)
```