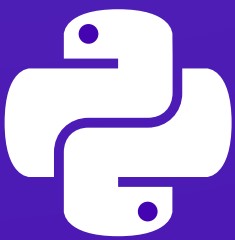


Python series



scikit-learn Fundamentals

With Code Examples

Data Preprocessing

Clean and prepare your data for machine learning with scikit-learn's preprocessing tools.



```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = [[1, 2], [3, 4]]
X_scaled = scaler.fit_transform(X)
print(X_scaled)
```

Results:



```
[ [-1.  -1. ]
 [ 1.   1. ]]
```

follow for more

Train-Test Split

Split your data into training and testing sets for model evaluation.

```
from sklearn.model_selection import train_test_split
X = [[1, 2], [3, 4], [5, 6], [7, 8]]
y = [0, 1, 1, 0]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)
print(X_train)
print(X_test)
```

Results:

```
[[5, 6], [1, 2]]
[[3, 4], [7, 8]]
```

Swipe next →

Linear Regression

Implement linear regression for predicting continuous target variables.



```
from sklearn.linear_model import LinearRegression
X = [[1], [2], [3], [4]]
y = [2, 4, 6, 8]
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict([[5]])
print(y_pred)
```

Results:



```
[10.]
```

follow for more

Logistic Regression

Classify binary outcomes using logistic regression.



```
from sklearn.linear_model import LogisticRegression
X = [[0, 0], [1, 1], [2, 2], [3, 3]]
y = [0, 0, 1, 1]
model = LogisticRegression()
model.fit(X, y)
y_pred = model.predict([[2, 2]])
print(y_pred)
```

Results:



```
[1]
```

Swipe next →

K-Nearest Neighbors

Use the k-nearest neighbors algorithm for classification and regression tasks.



```
from sklearn.neighbors import KNeighborsClassifier
X = [[0, 0], [1, 1], [2, 2], [3, 3]]
y = [0, 0, 1, 1]
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X, y)
y_pred = model.predict([[2, 2]])
print(y_pred)
```

Results:



```
[1]
```

follow for more

Decision Trees

Build decision trees for classification and regression problems.



```
from sklearn.tree import DecisionTreeClassifier
X = [[0, 0], [1, 1], [2, 2], [3, 3]]
y = [0, 0, 1, 1]
model = DecisionTreeClassifier()
model.fit(X, y)
y_pred = model.predict([[2, 2]])
print(y_pred)
```

Results:



```
[1]
```

Swipe next →

Random Forests

Ensemble multiple decision trees into powerful random forest models.



```
from sklearn.ensemble import RandomForestClassifier
X = [[0, 0], [1, 1], [2, 2], [3, 3]]
y = [0, 0, 1, 1]
model = RandomForestClassifier(n_estimators=10)
model.fit(X, y)
y_pred = model.predict([[2, 2]])
print(y_pred)
```

Results:



```
[1]
```

Swipe next →

Cross-Validation

Evaluate model performance using cross-validation techniques.

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
X = [[0, 0], [1, 1], [2, 2], [3, 3]]
y = [0, 0, 1, 1]
model = LogisticRegression()
scores = cross_val_score(model, X, y, cv=3)
print(f"Accuracy: {scores.mean():.2f} +/- {scores.std():.2f}")
```

Results:

```
Accuracy: 1.00 +/- 0.00
```

Model Evaluation Metrics

Assess model performance with various evaluation metrics.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
X = [[0, 0], [1, 1], [2, 2], [3, 3]]
y = [0, 0, 1, 1]
model = LogisticRegression()
model.fit(X, y)
y_pred = model.predict(X)
accuracy = accuracy_score(y, y_pred)
precision = precision_score(y, y_pred)
recall = recall_score(y, y_pred)
f1 = f1_score(y, y_pred)
print(f"Accuracy: {accuracy:.2f}, Precision: {precision:.2f}, Recall: {recall:.2f}, F1-score: {f1:.2f}")
```


Results:

```
Accuracy: 1.00, Precision: 1.00, Recall: 1.00, F1-score: 1.00
```

follow for more

Model Persistence

Save and load trained models for future use.



```
import joblib
from sklearn.linear_model import LogisticRegression
X = [[0, 0], [1, 1], [2, 2], [3, 3]]
y = [0, 0, 1, 1]
model = LogisticRegression()
model.fit(X, y)

# Save the model
joblib.dump(model, 'model.joblib')

# Load the model
loaded_model = joblib.load('model.joblib')
y_pred = loaded_model.predict([[2, 2]])
print(y_pred)
```

Results:




```
[1]
```

Swipe next →


Feature Selection

Identify the most relevant features in your dataset using scikit-learn's feature selection methods.



```
from sklearn.feature_selection import SelectKBest, f_classif
X = [[0, 0], [1, 1], [2, 2], [3, 3]]
y = [0, 0, 1, 1]
selector = SelectKBest(f_classif, k=1)
X_new = selector.fit_transform(X, y)
print(X_new)
```

Results:



```
[[0.]
 [1.]
 [2.]
 [3.]]
```

Dimensionality Reduction

Reduce the number of features in your dataset using techniques like Principal Component Analysis (PCA).



```
from sklearn.decomposition import PCA
X = [[0, 0], [1, 1], [2, 2], [3, 3]]
pca = PCA(n_components=1)
X_pca = pca.fit_transform(X)
print(X_pca)
```

Results:



```
[ [-1.41421356]
  [-0.70710678]
  [
```