📚 Tutorial: Linear Regression with `sklearn.linear_model.LinearRegression`

Step 1: Import Necessary Libraries

First, you need to import the required libraries. Make sure you have Scikit-learn, NumPy, and Pandas installed. If not, you can install them using pip:

```
pip install numpy pandas scikit-learn
```

Now, import the libraries in your Python script or Jupyter Notebook:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

Step 2: Load the Dataset

For this tutorial, let's use a simple synthetic dataset. You can also load a real dataset using Pandas.

```
# Create a synthetic dataset
data = {
    'Height': [160, 170, 180, 190, 200, 165, 175, 185],
    'Weight': [60, 70, 80, 90, 100, 65, 75, 85]
}

# Convert to DataFrame
df = pd.DataFrame(data)
print(df)
```

Step 3: Prepare the Data

Separate the features (independent variables) and the target (dependent variable). In this case, let's predict `Weight` based on `Height`.

```
# Features and target variable
X = df[['Height']]  # Features
y = df['Weight']    # Target variable
```

Step 4: Split the Data

Split the dataset into training and testing sets. This helps evaluate the model's performance on unseen data.

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

print("Training set:")
print(X_train, y_train)
print("Testing set:")
print(X_test, y_test)
```

Step 5: Create and Train the Model

Now, create a linear regression model and fit it to the training data.

```
# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
```

Step 6: Make Predictions

Use the trained model to make predictions on the test set.

```
# Make predictions
y_pred = model.predict(X_test)
print("Predicted weights:", y_pred)
```

Step 7: Evaluate the Model

Evaluate the model's performance using metrics like Mean Squared Error (MSE) and R-squared (R^2).

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

Step 8: Visualize the Results

Visualize the regression line and the data points.

```
# Plotting
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression line')
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.title('Linear Regression: Height vs Weight')
plt.legend()
plt.show()
```

Complete Code Example

Here's the complete code for the tutorial:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Create a synthetic dataset
data = {
    'Height': [160, 170, 180, 190, 200, 165, 175, 185],
    'Weight': [60, 70, 80, 90, 100, 65, 75, 85]
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Features and target variable
X = df[['Height']]
y = df['Weight']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)

print("Mean Squared Error:", mse)

# Plotting
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression line')
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.title('Linear Regression: Height vs Weight')
plt.legend()
plt.show()
```

Conclusion

This tutorial provides a comprehensive overview of how to perform linear regression using Scikit-learn. You can adapt this workflow to different datasets and features as needed. Happy coding!

`sklearn.linear_model.LinearRegression` Summary

Key Arguments

1. **fit_intercept** (default=True):
    - Determines whether to calculate the intercept for the model. If set to False, the model will be forced through the origin.
2. **normalize** (default=False):
    - If True, the regressors (X) are normalized before regression. Useful when features are on different scales.
3. **n_jobs** (default=None):
    - The number of jobs to use for computation. If set to -1, it uses all available cores, which can speed up computations for large datasets.

Key Points

- **Model Fitting**:
    - Use the `fit(X, y)` method to train the model, where (X) is the feature matrix and (y) is the target vector.
- **Making Predictions**:
    - After fitting, use the `predict(X)` method to generate predictions for new data.
- **Coefficients and Intercept**:
    - Access the model coefficients via the `coef_` attribute and the intercept via the `intercept_` attribute after fitting.
- **Performance Evaluation**:
    - Evaluate model performance using metrics like Mean Squared Error (MSE) and R-squared (R^2).
- **Assumptions**:
    - Assumes linearity between features and target, independence of errors, homoscedasticity, and normality of errors.

Example Usage

Here's a simple example of using `LinearRegression`:

```
from sklearn.linear_model import LinearRegression
import numpy as np

# Sample data
X = np.array([[1], [2], [3], [4]])
y = np.array([2, 3, 5, 7])
```

```
# Create and fit the model
model = LinearRegression()
model.fit(X, y)

# Make predictions
predictions = model.predict(np.array([[5], [6]]))

# Output results
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("Predictions:", predictions)
```

This streamlined overview focuses on the most relevant arguments and points for practical usage of `sklearn.linear_model.LinearRegression`.