This code trains a simple linear regression model using PyTorch. Here's a breakdown, focusing on how the data is prepared for PyTorch:

Data Loading and Preprocessing

```
df = pd.read_excel('body-fat-brozek.xlsx')
X = df.iloc[:, :-1].values
X = (X - X.mean(axis=0)) / X.std(axis=0)  # Z-score normalization
ys = df.iloc[:, -1].values
```

- Loads a dataset from an Excel file into a Pandas DataFrame.
- `X` : all columns except the last one (features).
- `ys` : last column (target).
- Features are **standardized**: zero mean and unit variance.

Conversion to PyTorch Tensors

```
def to_tensor(X):
    return torch.tensor(X, dtype=torch.float32)
```

- Wraps `torch.tensor(...)` to ensure all inputs are converted to 32-bit floats, the default for PyTorch models.

Train-Test Split and Tensor Conversion

```
X_train, X_test, ys_train, ys_test = (
    to_tensor(d) for d in train_test_split(X, ys, test_size=0.5, random_state=42)
)
X_train = to_tensor(X_train)
ys_train = to_tensor(ys_train).view(-1, 1)
X_test = to_tensor(X_test)
ys_test = to_tensor(ys_test).view(-1, 1)
```

- Uses `train_test_split` to divide data 50/50.
- Converts each split to PyTorch tensors.
- `.view(-1, 1)` reshapes the targets to a column vector shape (needed for PyTorch's `nn.Linear`, which expects 2D inputs).

Model and Optimizer Setup

```
model = nn.Linear(X.shape[1], 1)
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
```

- `nn.Linear`: a linear regression model with `X.shape[1]` input features and 1 output.
- `SGD` optimizer updates weights using stochastic gradient descent.
- Alternative `Adam` optimizer is commented out.

Training Loop

```
for epoch in range(10000):
    optimizer.zero_grad()
    loss = nn.MSELoss()(model(X_train), ys_train)
    loss.backward()
    optimizer.step()
```

- For 10,000 epochs:
    - Computes predictions and MSE loss.
    - Backpropagates gradients.
    - Updates weights using the optimizer.
    - Prints loss every 1000 epochs.

Evaluation and Feature Importance

```
with torch.no_grad():
    mae = torch.mean(torch.abs(model(X_test) - ys_test)).item()
    ...
```

- Disables gradient tracking (`torch.no_grad()`).
- Computes **Mean Absolute Error** on the test set.
- Retrieves learned weights: `model.weight.data` .
- Prints feature importances sorted by absolute weight.

Summary of PyTorch-specific parts:

- Converts NumPy arrays to `torch.tensor(...)` , using `.float32` .
- Model is built with `nn.Linear` .
- Loss is `nn.MSELoss()` .
- Optimization via `torch.optim.SGD` .
- Training includes `zero_grad()` , `backward()` , and `step()` .
- Evaluation uses `.no_grad()` and `.item()` for scalar conversion.

Let me know if you'd like a diagram or a Jupyter-ready version.