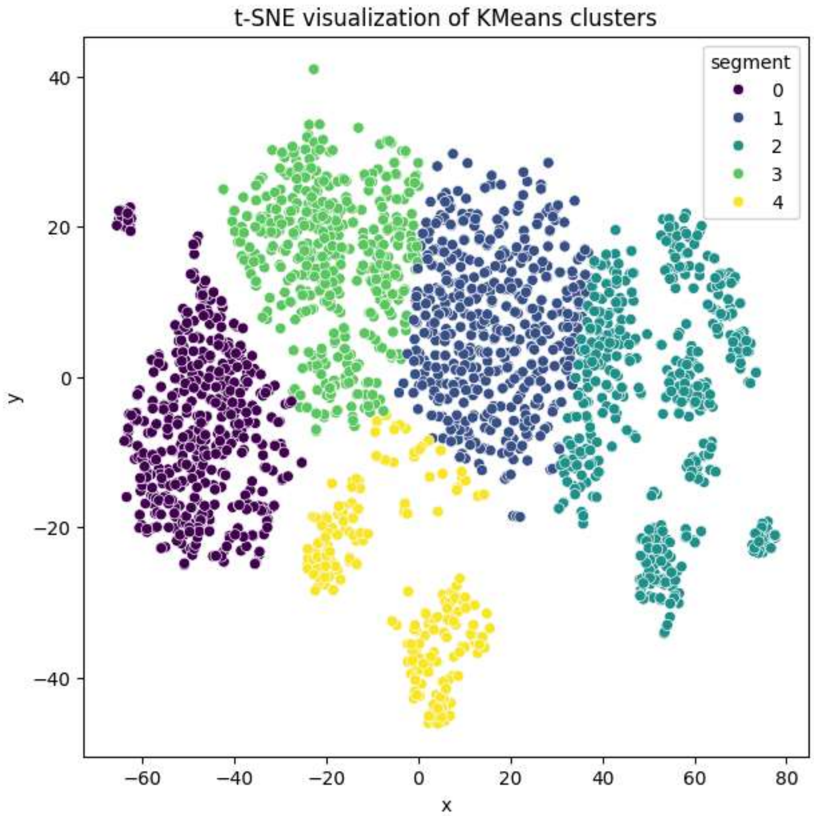
```
import torch
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans

import warnings
warnings.filterwarnings('ignore')

# 1. Load the dataset
df = pd.read_csv('/content/new.csv')

# 2. Handle missing values
df.dropna(inplace=True)

# 3. Extract day, month, and year from 'Dt_Customer'
parts = df["Dt_Customer"].str.split("-", n=3, expand=True)
df["day"] = parts[0].astype('int')
df["month"] = parts[1].astype('int')
df["year"] = parts[2].astype('int')

# 4. Drop original date and specified columns
df.drop(['Dt_Customer', 'Z_CostContact', 'Z_Revenue'], axis=1, inplace=True)

# 5. Identify object type columns and apply Label Encoding
object_cols = df.select_dtypes(include='object').columns
for col in object_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])

# 6. Convert to PyTorch tensor
data_tensor = torch.tensor(df.values, dtype=torch.float32)

# 7. Standardize data using PyTorch
mean = torch.mean(data_tensor, dim=0)
std = torch.std(data_tensor, dim=0)
std[std == 0] = 1e-7
scaled_data_tensor = (data_tensor - mean) / std

# 8. Apply t-SNE for dimensionality reduction (using scikit-learn as no pure PyTorch equivalent was used)
scaled_data_np = scaled_data_tensor.numpy()
model_tsne = TSNE(n_components=2, random_state=0)
tsne_data_np = model_tsne.fit_transform(scaled_data_np)
tsne_data_pytorch = torch.tensor(tsne_data_np, dtype=torch.float32)

# 9. Apply KMeans clustering (using scikit-learn)
tsne_data_np_for_kmeans = tsne_data_pytorch.numpy()
model_kmeans = KMeans(n_clusters=5, random_state=22, n_init=10)
segments_np = model_kmeans.fit_predict(tsne_data_np_for_kmeans)
segments_pytorch = torch.tensor(segments_np, dtype=torch.int64)

# 10. Convert tensors to NumPy for visualization and create DataFrame
tsne_data_np_for_plot = tsne_data_pytorch.numpy()
segments_np_for_plot = segments_pytorch.numpy()
df_tsne = pd.DataFrame({'x': tsne_data_np_for_plot[:, 0], 'y': tsne_data_np_for_plot[:, 1], 'segment': segments_np_for_plot})

# 11. Generate and display the scatter plot
plt.figure(figsize=(7, 7))
sb.scatterplot(x='x', y='y', hue='segment', data=df_tsne, palette='viridis')
plt.title('t-SNE visualization of KMeans clusters')
plt.show()

# Optional: Display shapes and types for verification
print("--- Verification of Shapes and Data Types ---")
print(f"Processed DataFrame shape: {df.shape}")
print(f"PyTorch tensor after preprocessing (data_tensor) shape: {data_tensor.shape}, dtype: {data_tensor.dtype}")
print(f"PyTorch tensor after scaling (scaled_data_tensor) shape: {scaled_data_tensor.shape}, dtype: {scaled_data_tensor.dtype}")
print(f"PyTorch tensor after t-SNE (tsne_data_pytorch) shape: {tsne_data_pytorch.shape}, dtype: {tsne_data_pytorch.dtype}")
print(f"PyTorch tensor after KMea                                                        pytorch.dtype}")
print(f"DataFrame for plotting (d
```

What can I help you build?




```
--- Verification of Shapes and Data Types ---
Processed DataFrame shape: (2216, 29)
PyTorch tensor after preprocessing (data_tensor) shape: torch.Size([2216, 29]), dtype: torch.float32
PyTorch tensor after scaling (scaled_data_tensor) shape: torch.Size([2216, 29]), dtype: torch.float32
PyTorch tensor after t-SNE (tsne_data_pytorch) shape: torch.Size([2216, 2]), dtype: torch.float32
PyTorch tensor after KMeans (segments_pytorch) shape: torch.Size([2216]), dtype: torch.int64
DataFrame for plotting (df_tsne) shape: (2216, 3)
```