```
import pandas as pd
import numpy as np
import torch
from torch import nn
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import StandardScaler
from opacus import PrivacyEngine

# Parameters
NUM_CLIENTS = 3
ROUNDS = 3
EPOCHS = 5
BATCH_SIZE = 16
NOISE_MULTIPLIER = 1.0
MAX_GRAD_NORM = 1.0

# Load marketing dataset
df = pd.read_csv("marketing_data.csv")

# Split features and labels
features = df[['age', 'visits', 'clicks', 'time_on_site', 'purchases']].values
labels = df['converted'].values

# Split data among clients
client_data = np.array_split(np.column_stack((features, labels)), NUM_CLIENTS)

# Prepare DataLoader
def prepare_dataloader(X, y):
    X_tensor = torch.tensor(X, dtype=torch.float32)
    y_tensor = torch.tensor(y, dtype=torch.long)
    return DataLoader(TensorDataset(X_tensor, y_tensor), batch_size=BATCH_SIZE, shuffle=True)

# Define logistic regression model
class LogisticModel(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.linear = nn.Linear(input_dim, 2)

    def forward(self, x):
        return self.linear(x)

# Train model with differential privacy
def train_with_privacy(model, dataloader):
    optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
    criterion = nn.CrossEntropyLoss()
    privacy_engine = PrivacyEngine()

    model, optimizer, dataloader = privacy_engine.make_private(
        module=model,
        optimizer=optimizer,
        data_loader=dataloader,
        noise_multiplier=NOISE_MULTIPLIER,
        max_grad_norm=MAX_GRAD_NORM,
    )
```

```
    model.train()
    for _ in range(EPOCHS):
        for x_batch, y_batch in dataloader:
            optimizer.zero_grad()
            output = model(x_batch)
            loss = criterion(output, y_batch)
            loss.backward()
            optimizer.step()

    return model

# Federated learning process
def federated_learning():
    global_model = LogisticModel(input_dim=5)
    global_weights = global_model.state_dict()

    for round in range(ROUNDS):
        print(f"Round {round + 1}")
        local_models = []

        for idx, client in enumerate(client_data):
            X = client[:, :-1]
            y = client[:, -1].astype(int)
            scaler = StandardScaler()
            X = scaler.fit_transform(X)

            dataloader = prepare_dataloader(X, y)
            local_model = LogisticModel(input_dim=5)
            local_model.load_state_dict(global_weights)

            trained_model = train_with_privacy(local_model, dataloader)
            local_models.append(trained_model.state_dict())

        # Aggregate models
        new_state_dict = global_model.state_dict()
        for key in new_state_dict:
            new_state_dict[key] = torch.stack([m[key] for m in local_models], dim=0).mean(dim=0)
        global_model.load_state_dict(new_state_dict)

    return global_model

# Run training
if __name__ == "__main__":
    model = federated_learning()
    print("Training complete. Final model ready for personalized and privacy-preserving predictions.")
```