```
!pip install torch_geometric
!pip install easydict
!pip install gdown
!gdown --id '1fog9Q3rHdRN73CsYm_tGB2EDDx5V1_4q' --output data.zip
!unzip -o data.zip
```

```
Collecting torch_geometric
  Downloading torch_geometric-2.6.1-py3-none-any.whl.metadata (63 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.1/63.1 kB 1.2 MB/s eta 0:00:00
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from torch_geometric) (3.10.8)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch_geometric) (2024.6.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch_geometric) (3.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torch_geometric) (1.26.4)
Requirement already satisfied: psutil>=5.8.0 in /usr/local/lib/python3.10/dist-packages (from torch_geometric) (5.9.5)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from torch_geometric) (3.1.4)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torch_geometric) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from torch_geometric) (4.66.5)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geome
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric) (
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric) (24.
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric
Requirement already satisfied: yarl<2.0,>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geome
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch_geometric) (2.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->torch_geo
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torch_geometric) (3.1
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torch_geometric
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torch_geometric
Requirement already satisfied: typing-extensions>=4.1.0 in /usr/local/lib/python3.10/dist-packages (from multidict<7.0,>=4.5
Downloading torch_geometric-2.6.1-py3-none-any.whl (1.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 13.0 MB/s eta 0:00:00
Installing collected packages: torch_geometric
Successfully installed torch_geometric-2.6.1
Requirement already satisfied: easydict in /usr/local/lib/python3.10/dist-packages (1.13)
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (5.2.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.12.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.16.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.5)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.6)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gd
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdow
/usr/local/lib/python3.10/dist-packages/gdown/__main__.py:140: FutureWarning: Option `--id` was deprecated in version 4.3.1
  warnings.warn(
Downloading...
From (original): https://drive.google.com/uc?id=1fog9Q3rHdRN73CsYm_tGB2EDDx5V1_4q
From (redirected): https://drive.google.com/uc?id=1fog9Q3rHdRN73CsYm_tGB2EDDx5V1_4q&confirm=t&uuid=bef4c103-977c-4c4b-8281-8
To: /content/data.zip
100% 61.9M/61.9M [00:01<00:00, 48.5MB/s]
Archive:  data.zip
   creating: dataset/
  inflating: dataset/features_test.pkl
  inflating: dataset/features_train.pkl
  inflating: dataset/graph_test.pkl
  inflating: dataset/graph_train.pkl
```

Generate | 10 random numbers using numpy | Close

```
import easydict
import numpy as np
import pandas as pd
import pickle as pkl
import os
import json
import torch
import torch.nn as nn
import torch_geometric as tg
from torch_geometric.nn import GCNConv
from torch_geometric.data import Data
from torch_geometric.loader import DataLoader
import scipy.sparse as sp
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
```

```
from sklearn.metrics import mean_squared_error



def to_device(batch, device):
    batch["idx"] = batch["idx"].to(device)
    batch["hist_graphs"] = [x.to(device) for x in batch["hist_graphs"]]
    batch["labels"] = batch["labels"].to(device)
    return batch

class AssetBatch(torch.utils.data.Dataset):
    def __init__(self, args, adjs, features, labels, start, end):
        super(AssetBatch, self).__init__()
        self.args = args
        self.adjs = adjs
        self.features = features
        self.labels = labels
        self.hist_time_steps = args.hist_time_steps
        self.start = start
        self.end = end

    def __len__(self):
        return self.end - self.start + 1

    def __getitem__(self, index):
        idx = index + self.start
        hist_graphs = []
        for i in range(idx - self.hist_time_steps, idx):
            x = torch.Tensor(self.features[i])
            edge_index, edge_weight = tg.utils.from_scipy_sparse_matrix(self.adjs[i])
            graph = Data(x=x, edge_index=edge_index, edge_weight=edge_weight)
            hist_graphs.append(graph)
        sample = {
            'idx': torch.tensor(idx, dtype=torch.long),
            'hist_graphs': hist_graphs,
            'labels': torch.tensor(self.labels[idx], dtype=torch.float)
        }
        return sample



class AssetDataset():
    def __init__(self, args, features, adjs, mode="train"):
        self.args = args
        self.mode = mode
        self.max_steps = len(adjs)
        self.hist_time_steps = args.hist_time_steps

        if mode == "train":
            self.labels = [self._extract_labels(feat) for feat in features]
            self.features = [self._extract_features(feat) for feat in features]
            self.features = self._preprocess_features(self.features)
            self.adjs = [self._preprocess_adj(adj) for adj in adjs]
            self._split_data()
        elif mode == "test":
            self.labels = [np.zeros(features[0].shape[0])] * len(features)
            self.features = [self._extract_features(feat) for feat in features]
            self.features = self._preprocess_features(self.features)
            self.adjs = [self._preprocess_adj(adj) for adj in adjs]
            self._prepare_test_data()

    def _extract_labels(self, features):
        features = np.array(features.todense())
        labels = features[:, 0]
        return labels

    def _extract_features(self, features):
        features = np.array(features.todense())
        if self.mode == "train":
            features = features[:, 1:]
        return features

    def _preprocess_features(self, features_list):
        stacked_features = np.vstack(features_list)
        stacked_features = np.nan_to_num(stacked_features)

        if self.args.scaler == 'standard':
            scaler = StandardScaler()
```

```
        elif self.args.scaler == 'minmax':
            scaler = MinMaxScaler()
        elif self.args.scaler == 'robust':
            scaler = RobustScaler()
        else:
            scaler = StandardScaler()
        stacked_features = scaler.fit_transform(stacked_features)
        if self.args.pca_components > 0:
            pca = PCA(n_components=self.args.pca_components)
            stacked_features = pca.fit_transform(stacked_features)
        num_samples = len(features_list)
        split_features = np.split(stacked_features, num_samples)
        return split_features

    def _preprocess_adj(self, adj):
        if self.args.adj_norm:
            rowsum = np.array(adj.sum(1))
            rowsum[rowsum == 0] = 1
            r_inv = sp.diags(np.power(rowsum, -0.5).flatten(), dtype=np.float32)
            adj_normalized = adj.dot(r_inv).transpose().dot(r_inv)
            return adj_normalized
        return adj

    def _split_data(self):
        train_start = self.hist_time_steps
        valid_start = int(self.max_steps * self.args.train_proportion)

        train = AssetBatch(self.args, self.adjs, self.features, self.labels, train_start, valid_start - 1)
        valid = AssetBatch(self.args, self.adjs, self.features, self.labels, valid_start, self.max_steps - 1)

        self.train = DataLoader(train, shuffle=True, batch_size=self.args.batch_size, collate_fn=lambda x: x[0])
        self.valid = DataLoader(valid, shuffle=False, batch_size=self.args.batch_size, collate_fn=lambda x: x[0])

        print('Dataset splits:')
        print(f'{len(train)} train samples from {train_start} to {valid_start - 1}')
        print(f'{len(valid)} valid samples from {valid_start} to {self.max_steps - 1}')

    def _prepare_test_data(self):
        test_start = self.hist_time_steps
        test = AssetBatch(self.args, self.adjs, self.features, self.labels, test_start, self.max_steps - 1)
        self.test = DataLoader(test, shuffle=False, batch_size=self.args.batch_size, collate_fn=lambda x: x[0])
        print('Dataset info:')
        print(f'{len(test)} test samples from {test_start} to {self.max_steps - 1}')
```

Generate | create a dataframe with 2 columns and 10 rows | Close

```
class TGCNModel(nn.Module):
    def __init__(self, args, node_features):
        super(TGCNModel, self).__init__()
        self.args = args
        self.num_time_steps = args.hist_time_steps
        self.node_features = node_features
        self.hidden_dim = args.hidden_dim

        self.gcns = nn.ModuleList([
            GCNConv(self.node_features, self.hidden_dim)
            for _ in range(self.num_time_steps)
        ])

        self.gru = nn.GRU(self.hidden_dim, self.hidden_dim, batch_first=True)
        self.fc = nn.Linear(self.hidden_dim, 1)
        self.dropout = nn.Dropout(args.dropout)

    def forward(self, graphs):
        gcn_outputs = []
        for t in range(self.num_time_steps):
            x = graphs[t].x
            edge_index = graphs[t].edge_index
            h = self.gcns[t](x, edge_index)
            h = torch.relu(h)
            gcn_outputs.append(h.unsqueeze(1))

        gcn_outputs = torch.cat(gcn_outputs, dim=1)
        gcn_outputs = self.dropout(gcn_outputs)
        gru_out, _ = self.gru(gcn_outputs)
```

```
            final_out = gru_out[:, -1, :]

            y = self.fc(final_out).flatten()
            return y

        def get_loss(self, data):
            graphs = data['hist_graphs']
            labels = data['labels'].squeeze(0)
            pred = self.forward(graphs)
            if pred.shape != labels.shape:
                print(f"Shape mismatch after squeeze: pred.shape = {pred.shape}, labels.shape = {labels.shape}")
            loss_fn = nn.MSELoss()
            loss = loss_fn(pred, labels)
            return loss, pred.detach().cpu().numpy(), labels.detach().cpu().numpy()

        def predict(self, data):
            graphs = data['hist_graphs']
            predictions = self.forward(graphs)
            return predictions.detach().cpu().numpy()


    if __name__ == "__main__":

        args = easydict.EasyDict({
            "hist_time_steps": 12,
            "epochs": 100,
            "batch_size": 1,
            "feat_norm": True,
            "adj_norm": True,
            "early_stop": 15,
            "train_proportion": 0.8,
            "learning_rate": 0.001,
            "weight_decay": 0.0001,
            "hidden_dim": 64,
            "dropout": 0.3,
            "scaler": 'standard',
            "pca_components": 50,
        })
        RESULTS_DIR = "result/"
        os.makedirs(RESULTS_DIR, exist_ok=True)
        with open(os.path.join(RESULTS_DIR, 'args.txt'), 'w') as f:
            json.dump(args.__dict__, f, indent=2)

        print('Results directory:', RESULTS_DIR)
        print("Start Training...")


        with open('dataset/graph_train.pkl', 'rb') as file:
            adjs = pkl.load(file)
        with open('dataset/features_train.pkl', 'rb') as file:
            feats = pkl.load(file)

        feat_dim = feats[0].shape[1] - 1
        num_nodes = adjs[0].shape[0]

        print('Total time steps:', len(adjs))
        print('Total number of assets:', num_nodes)
        print('Total number of features:', feat_dim)

        device = torch.device("cuda" if torch.cuda.is_available() else 'cpu')
        dataloader = AssetDataset(args, feats, adjs, "train")

        model = TGCNModel(args, args.pca_components if args.pca_components > 0 else feat_dim).to(device)
        optimizer = torch.optim.Adam(model.parameters(), lr=args.learning_rate, weight_decay=args.weight_decay)
        scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=5, factor=0.5)


        best_val_loss = float('inf')
        patience_counter = 0
        for epoch in range(args.epochs):
            model.train()
            train_losses = []
            for train_data in dataloader.train:
                train_data = to_device(train_data, device)
                optimizer.zero_grad()
                loss, _, _ = model.get_loss(train_data)
                loss.backward()
                torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=2.0)
```

```
            optimizer.step()
            train_losses.append(loss.item())


        model.eval()
        val_losses = []
        val_preds = []
        val_labels = []
        with torch.no_grad():
            for val_data in dataloader.valid:
                val_data = to_device(val_data, device)
                val_loss, preds, labels = model.get_loss(val_data)
                val_losses.append(val_loss.item())
                val_preds.extend(preds)
                val_labels.extend(labels)

        avg_train_loss = np.mean(train_losses)
        avg_val_loss = np.mean(val_losses)


        val_mse = mean_squared_error(val_labels, val_preds)

        scheduler.step(avg_val_loss)

        if avg_val_loss < best_val_loss:
            best_val_loss = avg_val_loss
            torch.save(model.state_dict(), os.path.join(RESULTS_DIR, "model.pt"))
            patience_counter = 0
        else:
            patience_counter += 1
            if patience_counter >= args.early_stop:
                print("Early stopping triggered.")
                break

        print(f"Epoch {epoch+1}/{args.epochs}, LR: {optimizer.param_groups[0]['lr']:.6f}, "
              f"Train Loss: {avg_train_loss:.6f}, Val Loss: {avg_val_loss:.6f}, Val MSE: {val_mse:.6f}")

    print("Start Testing...")

    model.load_state_dict(torch.load(os.path.join(RESULTS_DIR, "model.pt"), map_location=device))
    model.eval()


    with open('dataset/graph_test.pkl', 'rb') as file:
        adjs_test = pkl.load(file)
    with open('dataset/features_test.pkl', 'rb') as file:
        feats_test = pkl.load(file)
    feat_dim_test = feats_test[0].shape[1]


    if args.pca_components > 0:
        assert args.pca_components == model.node_features, "Feature dimensions do not match between training and testing data."
    else:
        assert feat_dim_test == model.node_features, "Feature dimensions do not match between training and testing data."

    dataloader_test = AssetDataset(args, feats_test, adjs_test, "test")


    test_preds = []
    with torch.no_grad():
        for test_data in dataloader_test.test:
            test_data = to_device(test_data, device)
            test_pred = model.predict(test_data)
            test_preds.append(test_pred)


    pred = pd.DataFrame(test_preds)
    stacked_pred = pred.stack().reset_index(drop=True)
    submission = pd.DataFrame({'Id': np.arange(len(stacked_pred)), 'Label': stacked_pred})
    submission.to_csv('submission10.csv', index=False)
    print("Submission file created: submission.csv")
```

```
Results directory: result/
Start Training...
Total time steps: 175
```

```
Total number of assets: 372
Total number of features: 93
<ipython-input-4-aa648a117313>:59: RuntimeWarning: invalid value encountered in power
  r_inv = sp.diags(np.power(rowsum, -0.5).flatten(), dtype=np.float32)
Dataset splits:
128 train samples from 12 to 139
35 valid samples from 140 to 174
Epoch 1/100, LR: 0.001000, Train Loss: 0.008819, Val Loss: 0.005419, Val MSE: 0.005419
Epoch 2/100, LR: 0.001000, Train Loss: 0.007979, Val Loss: 0.005509, Val MSE: 0.005509
Epoch 3/100, LR: 0.001000, Train Loss: 0.007991, Val Loss: 0.005374, Val MSE: 0.005374
Epoch 4/100, LR: 0.001000, Train Loss: 0.007834, Val Loss: 0.005380, Val MSE: 0.005380
Epoch 5/100, LR: 0.001000, Train Loss: 0.007884, Val Loss: 0.005497, Val MSE: 0.005497
Epoch 6/100, LR: 0.001000, Train Loss: 0.007777, Val Loss: 0.005673, Val MSE: 0.005673
Epoch 7/100, LR: 0.001000, Train Loss: 0.007657, Val Loss: 0.005401, Val MSE: 0.005401
Epoch 8/100, LR: 0.001000, Train Loss: 0.007760, Val Loss: 0.005673, Val MSE: 0.005673
Epoch 9/100, LR: 0.000500, Train Loss: 0.007762, Val Loss: 0.005402, Val MSE: 0.005402
Epoch 10/100, LR: 0.000500, Train Loss: 0.007689, Val Loss: 0.005376, Val MSE: 0.005376
Epoch 11/100, LR: 0.000500, Train Loss: 0.007629, Val Loss: 0.005493, Val MSE: 0.005493
Epoch 12/100, LR: 0.000500, Train Loss: 0.007633, Val Loss: 0.005371, Val MSE: 0.005371
Epoch 13/100, LR: 0.000500, Train Loss: 0.007654, Val Loss: 0.005414, Val MSE: 0.005414
Epoch 14/100, LR: 0.000500, Train Loss: 0.007603, Val Loss: 0.005382, Val MSE: 0.005382
Epoch 15/100, LR: 0.000500, Train Loss: 0.007616, Val Loss: 0.005637, Val MSE: 0.005637
Epoch 16/100, LR: 0.000500, Train Loss: 0.007710, Val Loss: 0.005440, Val MSE: 0.005440
Epoch 17/100, LR: 0.000500, Train Loss: 0.007589, Val Loss: 0.005399, Val MSE: 0.005399
Epoch 18/100, LR: 0.000250, Train Loss: 0.007599, Val Loss: 0.005375, Val MSE: 0.005375
Epoch 19/100, LR: 0.000250, Train Loss: 0.007554, Val Loss: 0.005419, Val MSE: 0.005419
Epoch 20/100, LR: 0.000250, Train Loss: 0.007565, Val Loss: 0.005385, Val MSE: 0.005385
Epoch 21/100, LR: 0.000250, Train Loss: 0.007558, Val Loss: 0.005428, Val MSE: 0.005428
Epoch 22/100, LR: 0.000250, Train Loss: 0.007578, Val Loss: 0.005409, Val MSE: 0.005409
Epoch 23/100, LR: 0.000250, Train Loss: 0.007525, Val Loss: 0.005386, Val MSE: 0.005386
Epoch 24/100, LR: 0.000125, Train Loss: 0.007521, Val Loss: 0.005431, Val MSE: 0.005431
Epoch 25/100, LR: 0.000125, Train Loss: 0.007488, Val Loss: 0.005387, Val MSE: 0.005387
Epoch 26/100, LR: 0.000125, Train Loss: 0.007489, Val Loss: 0.005410, Val MSE: 0.005410
Early stopping triggered.
Start Testing...
<ipython-input-12-fedf6bb6d674>:98: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default
  model.load_state_dict(torch.load(os.path.join(RESULTS_DIR, "model.pt"), map_location=device))
<ipython-input-4-aa648a117313>:59: RuntimeWarning: invalid value encountered in power
  r_inv = sp.diags(np.power(rowsum, -0.5).flatten(), dtype=np.float32)
Dataset info:
21 test samples from 12 to 32
Submission file created: submission.csv
```