W16_AllInOne_TestProgram-Copy2

January 12, 2021

[1]:
```
version = "09"
learningrate = 1e-3
learningrate1 = 3e-3
reset_scheduler_after_n_epochs = 10
```

1 Initialization

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from tqdm import tqdm

from IPython.display import display, HTML
import time
```

[3]:
```
import random
#Neural Network imports
import torch
import torch.nn as nn
import torch.optim as optim


from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error as mese
from sklearn.metrics import mean_absolute_error

from sklearn import linear_model
from sklearn.svm import SVR
```

[4]:
```
#cuda imports
ngpu = torch.cuda.device_count() # number of available gpus
```

```
device = torch.device("cuda:4") if (torch.cuda.is_available() and ngpu > 0)␣
 ↪else "cpu" #cuda:0 for gpu 0, cuda:4 for gpu 5
torch.backends.cudnn.benchmark=True # Uses cudnn auto-tuner to find the best␣
 ↪algorithm to use for your hardware

#Random Seed
random.seed(1337)
torch.manual_seed(1337)
```

[4]:
```
<torch._C.Generator at 0x7f511522c108>
```

Make all functions:

[5]:
```
class lstm(nn.Module):
    def __init__(self, hidden_state_size = 100):
        super().__init__()
        self.hidden_state_size = hidden_state_size
        self.lstm1 = nn.LSTM(1, self.hidden_state_size, batch_first=True)
        self.linear2 = nn.Linear(self.hidden_state_size, 1)

    def forward(self, X): #tensor X
        h, _ = self.lstm1( X )          # h shaped (batch, sequence,␣
 ↪hidden_layer)
        h = h[:,-1, :]                  # only need the output for the last␣
 ↪sequence

        y = self.linear2(h)             # make a prediction
        y = y + X[:,-1,-1:]         # make the output stationary
        return y.view(-1)               # like always

def init_LSTM(length_loader=1,train_for=100):
    LSTM = lstm().to(device)
    LSTM.double()

    #Training parameters:
    optimizer = optim.Adam(LSTM.parameters(), lr=learningrate)
    scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer,␣
 ↪max_lr=learningrate, steps_per_epoch=length_loader, epochs=train_for)
    criterion = nn.SmoothL1Loss()
    return LSTM, optimizer, criterion, scheduler
LSTM, optimizer, criterion, scheduler = init_LSTM()
```

[6]:
```
#Parameters:
layerSize = 128
outputSize = 1
featureSize = 170
relu = nn.ReLU()
```

```
#class maken voor NN
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(featureSize, layerSize)
        self.fc2 = nn.Linear(layerSize, outputSize)

    def forward(self, x):
        x = relu(self.fc1(x))
        x = self.fc2(x)
        return x

def init_NN():
    model = Net().to(device)
    model.float()

    #Training parameters:
    optimizer = optim.Adam(model.parameters(), lr=learningrate1)
    criterion = nn.SmoothL1Loss()
    return model, optimizer, criterion
model, optimizer, criterion = init_NN()
```

[7]:
```
def det(tensor):
    """
    Zet de tensor om van een tensor naar numpy op de CPU.
    """
    return tensor.cpu().detach().numpy()


def dim3Tensor(dft, dfs, window=7):
    """
    Make from an tensor an tensor with moving window.
    """
    #Get time shifted values and apply a moving window
    X = np.concatenate([ dfs[i:i+window].to_numpy() for i in␣
 ↪range(len(dfs)-window-1) ], axis=1).T
    #Reshape the data to the correct dimensions
    X = X.reshape(len(X), window, 1)

    #Get the target value (which is the next one in the sequence)
    y = dft[window:].to_numpy()[:-1]
    return X, y

def load_LSTM_data(house_nr):
    """
    Loads the Data for the lstm.
```

```
    and returns this.
    """
    house_nr = str(house_nr)
    if len(house_nr)==1:
        house_nr = "00"+str(house_nr)
    if len(house_nr)==2:
        house_nr = "0"+str(house_nr)
    df = pd.read_pickle('consumption_production_pickle_'+str(house_nr))
    df['hour'] = df.index.hour
    return df

def LSTM_split_df(df):
    """
    Splits the dataframe in train test validate parts.
    """
    trdf = df.loc['2019-01':'2019-08'].filter(['consumption'])
    vadf = df.loc['2019-09':'2019-11'].filter(['consumption'])
    tedf = df.loc['2019-12':].filter(['consumption'])
    return trdf, vadf, tedf

def LSTM_data_scaler(df1,df2,df3):
    scaler_TR = StandardScaler()
    scaler_TR.fit(df1)
    df1 = pd.DataFrame(scaler_TR.transform(df1), columns=['consumption'])

    scaler_VA = StandardScaler()
    scaler_VA.fit(df2)
    df2 = pd.DataFrame(scaler_VA.transform(df2), columns=['consumption'])

    scaler_TE = StandardScaler()
    scaler_TE.fit(df3)
    df3 = pd.DataFrame(scaler_TE.transform(df3), columns=['consumption'])
    return df1,df2,df3

def create_sCons(idf):
    """
    Create the desired shifted columns.
    """
    idf['sCons'] = idf.consumption.shift(24)
    idf = idf.dropna(axis=0)
    odf = idf.filter(['sCons'])
    idf = idf.drop(['sCons'], axis=1)
    return idf, odf

def LSTM_create_shifted_col(df1, df2, df3):
    """
    Create the desired shifted columns.
```

```
    """
    return create_sCons(df1), create_sCons(df2), create_sCons(df3)

def LSTM_create_tensors(n1,n2,n3,n4,n5,n6):
    train_X_t = torch.from_numpy(np.array(n1)).to(device).double()
    train_y_t = torch.from_numpy(np.array(n2)).to(device).double()

    valid_X_t = torch.from_numpy(np.array(n3)).to(device).double()
    valid_y_t = torch.from_numpy(np.array(n4)).to(device).double()

    test_X_t = torch.from_numpy(np.array(n5)).to(device).double()
    test_y_t = torch.from_numpy(np.array(n6)).to(device).double()
    return train_X_t, train_y_t, valid_X_t, valid_y_t, test_X_t, test_y_t

def calculate_metrics_for_model(output,target):
    """
    Calculates all the desired evaluation metrics for the model.
    """
    yhat = det(output)
    y = det(target)
    actual, pred = np.array(y), np.array(yhat)

    mae = mean_absolute_error(yhat, y)
    mse = mese(yhat, y)
    mape = np.mean(np.abs((actual - pred) / actual)) * 100
    r2 = r2_score(yhat, y)
    return [mae, mse, mape, r2]

def train_LSTM_Nepoch(data, target,N):
    """
    Train the LSTM for one epoch.
    """
    LSTM, optimizer, criterion, scheduler = init_LSTM(len(train_loader))
    LSTM.train()
    optimizer.zero_grad()
    for i in range(N):
        output = LSTM(data)

        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        scheduler.step()

        if i % reset_scheduler_after_n_epochs == 0:
            scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer,␣
 ↪max_lr=learningrate, steps_per_epoch=len(train_loader),␣
 ↪epochs=reset_scheduler_after_n_epochs)
```

```
            pass
    return calculate_metrics_for_model(output, target)

def validate_LSTM(data, target):
    """
    Validate the LSTM on unseen data and give back the evaluation metrics.
    """
    LSTM.eval()
    optimizer.zero_grad()

    outputV = LSTM(data)
    return calculate_metrics_for_model(outputV,target)

def GetGlobalData(nr):
    """
    Get the data for the MVLR, SVR and NN.
    """
    return pd.read_pickle('House_'+str(nr)).drop(pd.
 ↪read_pickle('House_'+str(nr)).columns[-1], axis=1)

def NormalScaler(df):
    """
    Scale the data according to the normal method.
    """
    #scale the data
    #X:
    scalerx = StandardScaler()
    scalerx.fit(df.loc[:,~df.columns.isin(["consumption"])])
    scaled_dataX = scalerx.transform(df.loc[:,~df.columns.
 ↪isin(["consumption"])]).tolist()
    #Y:
    scalery = StandardScaler()
    scalery.fit(df.loc[:,df.columns.isin(["consumption"])])
    datay = scalery.transform(df.loc[:,df.columns.isin(["consumption"])])
    return datay,scaled_dataX

def Split_Normal(dataX,dataY):
    """
    Split the data according to the method.
    """
    #split the data
    train_X = dataX[0:5800]
    train_y = dataY[0:5800].reshape(-1,1)

    valid_X = dataX[5800:7952]
    valid_y = dataY[5800:7952].reshape(-1,1)
```

```
    test_X = dataX[7952:8663]
    test_y = dataY[7952:8663].reshape(-1,1)
    return train_X, train_y, valid_X, valid_y, test_X, test_y

def MakeTrainLoader(train_X, train_y, valid_X, valid_y, test_X, test_y):
    """
    Function for making the dataloaders.
    This is mainly for the NN.
    """
    #Make tensors from the numpy arrays.
    train_X_t = torch.from_numpy(np.array(train_X)).to(device).float()
    train_y_t = torch.from_numpy(np.array(train_y)).to(device).float()

    valid_X_t = torch.from_numpy(np.array(valid_X)).to(device).float()
    valid_y_t = torch.from_numpy(np.array(valid_y)).to(device).float()

    test_X_t = torch.from_numpy(np.array(test_X)).to(device).float()
    test_y_t = torch.from_numpy(np.array(test_y)).to(device).float()

    #Tensor Datasets
    train_set = torch.utils.data.TensorDataset(train_X_t, train_y_t)
    valid_set = torch.utils.data.TensorDataset(valid_y_t, valid_X_t)
    test_set = torch.utils.data.TensorDataset(test_y_t, test_X_t)

    #Tensor DataLoaders
    train_loader = torch.utils.data.DataLoader(train_set, batch_size=64,␣
↪shuffle=False, num_workers = 0)#, pin_memory=True)
    valid_loader = torch.utils.data.DataLoader(valid_set, batch_size=64,␣
↪shuffle=False, num_workers = 0)#, pin_memory=True)
    test_loader = torch.utils.data.DataLoader(test_set, batch_size=64,␣
↪shuffle=False, num_workers = 0)
    return train_loader, valid_loader, test_loader

def train_NN_Nepochs(train_loader,N):
    """
    Training function for the training of the NN.
    """
    model, optimizer, criterion = init_NN()
    model.train()
    for i in range(N):
        for batch_idx, data_target in enumerate(train_loader):
            data = data_target[0]
            target = data_target[1]
            data = data.view(-1, data.shape[1])
            optimizer.zero_grad()

            output = model(data)
```

```
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
    return calculate_metrics_for_model(output, target)

def validate_NN(d1,target):
    """
    Validation function for the training of the NN.
    """
    model.eval()
    d1 = torch.from_numpy(np.array(d1)).to(device).float()
    target = torch.from_numpy(np.array(target)).to(device).float()
    output = model(d1)
    return calculate_metrics_for_model(output, target)

def train_MVLR(train_X,train_y):
    """
    Function to train the MVLR.
    """
    regr = linear_model.LinearRegression()
    regr.fit(train_X,train_y[:,0])
    yhat = regr.predict(train_X)

    target = torch.from_numpy(np.array(train_y)).to(device).float()
    yhat = torch.from_numpy(np.array(yhat)).to(device).float()
    return calculate_metrics_for_model(yhat,target), regr

def validate_MVLR(d1,d2,regr):
    """
    Validate the MVLR with the input data.
    """
    yhat = regr.predict(d1)
    target = d2

    target = torch.from_numpy(np.array(target)).to(device).float()
    yhat = torch.from_numpy(np.array(yhat)).to(device).float()
    return calculate_metrics_for_model(yhat,target)

def train_SVR(train_X,train_y):
    """
    Function to train the MVLR.
    """
    regr = SVR()
    regr.fit(train_X,train_y[:,0])
    yhat = regr.predict(train_X)

    target = torch.from_numpy(np.array(train_y)).to(device).float()[:,0]
```

```
    yhat = torch.from_numpy(np.array(yhat)).to(device).float()
    return calculate_metrics_for_model(yhat,target), regr

def validate_SVR(d1,d2,regr):
    """
    Validate the MVLR with the input data.
    """
    yhat = regr.predict(d1)
    target = d2

    target = torch.from_numpy(np.array(target)).to(device).float()
    yhat = torch.from_numpy(np.array(yhat)).to(device).float()
    return calculate_metrics_for_model(yhat,target)
```

2 Main loop:

[8]:
```
#settings:
show_every = 10
train_for = 50 #LSTM
NNtrain_for = 250
learningrate = 1e-3
learningrate1 = 3e-3
houses = [28,37,40,42,105,115,56,51,58,70,99,100]
stationary = True

#stats savelist:
MVLR_stats = pd.DataFrame()#MVLR
SVR_stats = pd.DataFrame()#SVR
NN_stats = pd.DataFrame()#NN
LSTM_stats = pd.DataFrame()#LSTM

#Learning loop:
for i in tqdm(range(len(houses))):
    """
    Data loading...
    """
    print("NOW PREPARING DATA:")
    window_size = 7
    house_number = houses[i]

    #LSTM:
    lstm_df = load_LSTM_data(house_number)
    train_LSTM, valid_LSTM, test_LSTM = LSTM_split_df(lstm_df)
    train_LSTM, valid_LSTM, test_LSTM = LSTM_data_scaler(train_LSTM,␣
↪valid_LSTM, test_LSTM)
```

```
   (train_x_LSTM, train_y_LSTM), (valid_x_LSTM, valid_y_LSTM), (test_x_LSTM,␣
↪test_y_LSTM) = LSTM_create_shifted_col(train_LSTM, valid_LSTM, test_LSTM)
   train_x_LSTM, train_y_LSTM = dim3Tensor(train_x_LSTM, train_y_LSTM,␣
↪window_size); valid_x_LSTM, valid_y_LSTM = dim3Tensor(valid_x_LSTM,␣
↪valid_y_LSTM, window_size); test_x_LSTM, test_y_LSTM =␣
↪dim3Tensor(test_x_LSTM, test_y_LSTM, window_size);
   train_X_t_LSTM, train_y_t_LSTM, valid_X_t_LSTM, valid_y_t_LSTM,␣
↪test_X_t_LSTM, test_y_t_LSTM = LSTM_create_tensors(train_x_LSTM,␣
↪train_y_LSTM, valid_x_LSTM, valid_y_LSTM, test_x_LSTM, test_y_LSTM)

   #MVLR, SVR, NN:
   df = GetGlobalData(house_number)
   Y_data, X_data  = NormalScaler(df)
   train_X, train_y, valid_X, valid_y, test_X, test_y =␣
↪Split_Normal(X_data,Y_data)
   train_loader, validation_loader, test_loader = MakeTrainLoader(train_X,␣
↪train_y, valid_X, valid_y, test_X, test_y)

   """
   Training
   """
   print("NOW TRAINING:")
   #MVLR:
   MVLR_train_stats, mvlr = train_MVLR(train_X,train_y)

   #SVR:
   SVR_train_stats, svr = train_SVR(train_X,train_y)

   #NN:
   NN_train_stats = train_NN_Nepochs(train_loader,NNtrain_for)

   #LSTM:
   data = train_X_t_LSTM; target = train_y_t_LSTM.view(-1);
   LSTM_train_stats = train_LSTM_Nepoch(data,target,train_for)


   """
   Evaluation
   """
   print("NOW EVALUATING:")
   #MVLR:
   MVLR_valid_stats = validate_MVLR(valid_X,valid_y,mvlr)

   #SVR:
   SVR_valid_stats = validate_SVR(valid_X,valid_y,svr)

   #NN:
```

```
    NN_valid_stats = validate_NN(valid_X,valid_y)

    #LSTM:
    dataV = valid_X_t_LSTM; targetV = valid_y_t_LSTM.view(-1);
    LSTM_valid_stats = validate_LSTM(dataV, targetV)


    """
    Testing
    """
    print("NOW TESTING:")
    #MVLR:
    MVLR_test_stats = validate_MVLR(test_X,test_y,mvlr)

    #SVR:
    SVR_test_stats = validate_SVR(test_X,test_y,svr)

    #NN:
    NN_test_stats = validate_NN(test_X,test_y)

    #LSTM:
    dataTest = test_X_t_LSTM; targetTest = test_y_t_LSTM.view(-1);
    LSTM_test_stats = validate_LSTM(dataTest, targetTest)


    """
    Save metrics
    """
    print("NOW CALCULATING METRICS:")
    #MVLR:
    index =␣
↪["MAE_train","MSE_train","MAPE_train","R2_train","MAE_valid","MSE_valid","MAPE_valid","R2_v
    New_Stats = pd.DataFrame(MVLR_train_stats+MVLR_valid_stats+MVLR_test_stats,␣
↪index=index, columns=[str(house_number)])
    MVLR_stats = pd.concat([MVLR_stats,New_Stats],axis=1)

    #SVR:
    index =␣
↪["MAE_train","MSE_train","MAPE_train","R2_train","MAE_valid","MSE_valid","MAPE_valid","R2_v
    New_Stats = pd.DataFrame(SVR_train_stats+SVR_valid_stats+SVR_test_stats,␣
↪index=index, columns=[str(house_number)])
    SVR_stats = pd.concat([SVR_stats,New_Stats],axis=1)

    #NN:
    index =␣
↪["MAE_train","MSE_train","MAPE_train","R2_train","MAE_valid","MSE_valid","MAPE_valid","R2_v
```

```
    New_Stats = pd.DataFrame(NN_train_stats+NN_valid_stats+NN_test_stats,␣
↪index=index, columns=[str(house_number)])
    NN_stats = pd.concat([NN_stats,New_Stats],axis=1)

    #LSTM:
    index =␣
↪["MAE_train","MSE_train","MAPE_train","R2_train","MAE_valid","MSE_valid","MAPE_valid","R2_v
    New_Stats = pd.DataFrame(LSTM_train_stats+LSTM_valid_stats+LSTM_test_stats,␣
↪index=index, columns=[str(house_number)])
    LSTM_stats = pd.concat([LSTM_stats,New_Stats],axis=1)
```

```
  0%|          | 0/12 [00:00<?, ?it/s]
NOW PREPARING DATA:
NOW TRAINING:
NOW EVALUATING:
NOW TESTING:
  8%|          | 1/12 [01:11<13:06, 71.50s/it]
NOW CALCULATING METRICS:
NOW PREPARING DATA:
NOW TRAINING:
NOW EVALUATING:
NOW TESTING:
 17%|         | 2/12 [02:20<11:46, 70.69s/it]
NOW CALCULATING METRICS:
NOW PREPARING DATA:
NOW TRAINING:
NOW EVALUATING:
NOW TESTING:
 25%|         | 3/12 [03:30<10:35, 70.59s/it]
NOW CALCULATING METRICS:
NOW PREPARING DATA:
NOW TRAINING:
NOW EVALUATING:
NOW TESTING:
 33%|        | 4/12 [04:40<09:23, 70.49s/it]
NOW CALCULATING METRICS:
NOW PREPARING DATA:
NOW TRAINING:
NOW EVALUATING:
NOW TESTING:
 42%|        | 5/12 [05:59<08:31, 73.03s/it]
```

```
NOW CALCULATING METRICS:
NOW PREPARING DATA:
NOW TRAINING:
NOW EVALUATING:
NOW TESTING:
 50%|        | 6/12 [07:22<07:34, 75.82s/it]
NOW CALCULATING METRICS:
NOW PREPARING DATA:
NOW TRAINING:
NOW EVALUATING:
NOW TESTING:
 58%|        | 7/12 [08:35<06:15, 75.11s/it]
NOW CALCULATING METRICS:
NOW PREPARING DATA:
NOW TRAINING:
NOW EVALUATING:
NOW TESTING:
 67%|       | 8/12 [09:46<04:55, 73.92s/it]
NOW CALCULATING METRICS:
NOW PREPARING DATA:
NOW TRAINING:
NOW EVALUATING:
NOW TESTING:
 75%|       | 9/12 [10:56<03:38, 72.73s/it]
NOW CALCULATING METRICS:
NOW PREPARING DATA:
NOW TRAINING:
NOW EVALUATING:
NOW TESTING:
 83%|      | 10/12 [12:06<02:23, 71.76s/it]
NOW CALCULATING METRICS:
NOW PREPARING DATA:
NOW TRAINING:
NOW EVALUATING:
NOW TESTING:
 92%|     | 11/12 [13:17<01:11, 71.61s/it]
NOW CALCULATING METRICS:
NOW PREPARING DATA:
NOW TRAINING:
NOW EVALUATING:
NOW TESTING:
```

```
100%|      | 12/12 [14:28<00:00, 72.36s/it]
NOW CALCULATING METRICS:
```

[9]:
```
(NN_stats)
```

[9]:

	28	37	40	42	105	\
MAE_train	0.093142	0.085364	0.044794	0.050784	0.089386	
MSE_train	0.016239	0.011258	0.002907	0.004050	0.012355	
MAPE_train	26.148751	20.203789	11.817272	14.224532	24.566419	
R2_train	0.960514	0.978729	0.995401	0.982154	0.946734	
MAE_valid	0.688622	0.647286	0.813625	0.795903	0.833871	
MSE_valid	0.965112	0.754525	1.109359	1.087577	1.154629	
MAPE_valid	179.580331	147.158170	117.958450	139.106369	167.964530	
R2_valid	-21.921885	-21.281836	-23.828306	-19.660788	-20.865523	
MAE_test	0.719268	0.962482	0.940798	1.014340	0.970317	
MSE_test	1.160039	1.610535	1.943202	1.759829	1.579856	
MAPE_test	179.058349	136.640155	175.697792	283.392215	170.228887	
R2_test	-25.119544	-22.352327	-22.566970	-23.580334	-26.833155	

	115	56	51	58	70	\
MAE_train	0.113080	0.124174	0.090566	0.223820	0.082721	
MSE_train	0.018677	0.024419	0.013623	0.065298	0.011911	
MAPE_train	115.117085	25.901738	33.439812	60.970002	19.006905	
R2_train	0.954482	0.909361	0.915257	0.833082	0.953794	
MAE_valid	0.717299	0.723298	0.716555	0.639183	0.647115	
MSE_valid	0.994318	0.857033	0.832910	0.810421	0.826835	
MAPE_valid	195.373559	155.453491	249.712849	170.532346	208.078074	
R2_valid	-19.179400	-20.992832	-22.165890	-20.140392	-21.838509	
MAE_test	0.715385	0.764402	0.745136	0.717373	0.812174	
MSE_test	1.126163	1.020298	1.135821	1.186757	1.111065	
MAPE_test	358.141661	268.253779	166.103113	165.929484	111.435187	
R2_test	-22.175455	-22.771117	-19.997040	-22.732971	-22.026415	

	99	100
MAE_train	0.080945	0.120064
MSE_train	0.013771	0.021671
MAPE_train	13.111131	20.950356
R2_train	0.950635	0.893066
MAE_valid	0.715830	0.566421
MSE_valid	0.897513	0.715925
MAPE_valid	184.961033	223.106098
R2_valid	-18.581681	-19.501309
MAE_test	0.771613	0.442072
MSE_test	1.332527	0.652804
MAPE_test	163.309443	391.861439

```
R2_test     -24.179119  -21.508725
```

[10]:
```
(MVLR_stats)
```

[10]:
```
                    28          37          40          42          105  \
MAE_train     0.539776    0.487047    0.594334    0.618426    0.426398
MSE_train     0.677175    0.550612    0.650719    0.712479    0.439539
MAPE_train  372.762942  250.491691  261.619091  201.672292  285.611725
R2_train     -1.295701   -0.168206   -1.826845   -3.758616   -0.013595
MAE_valid     0.546119    0.483264    0.631514    0.647428    0.497675
MSE_valid     0.705909    0.532079    0.746581    0.822979    0.564836
MAPE_valid  389.660311  315.295458  162.011409  202.558827  375.348783
R2_valid     -1.631268   -1.073369   -1.594368   -2.492699   -0.123531
MAE_test      0.585810    0.608748    0.757381    0.838018    0.616078
MSE_test      0.779223    0.743084    1.132437    1.224808    0.691351
MAPE_test   365.024376  271.421313  304.437566  388.840294  384.319949
R2_test      -2.435056   -0.436935   -1.526976   -3.009745   -0.237296

                    115          56          51          58          70  \
MAE_train     0.554327    0.513016    0.537692    0.538506    0.544396
MSE_train     0.601086    0.539342    0.589090    0.660183    0.632755
MAPE_train  422.807264  252.926421  593.043470  338.957524  807.910442
R2_train     -0.656304   -0.403507   -0.347404   -0.837246   -0.529622
MAE_valid     0.568231    0.535316    0.510911    0.511881    0.503247
MSE_valid     0.665854    0.613655    0.553135    0.609081    0.584460
MAPE_valid  383.839583  263.016319  496.258211  333.443546  465.401363
R2_valid     -1.067319   -1.272496   -0.718725   -1.309194   -1.367590
MAE_test      0.577983    0.579907    0.629616    0.599253    0.643138
MSE_test      0.674005    0.600554    0.727486    0.769395    0.741737
MAPE_test  1062.242699  522.188711  308.340454  336.805892  160.431600
R2_test      -1.663687   -2.183115   -1.369704   -2.001718   -1.564057

                    99         100
MAE_train     0.585016    0.540455
MSE_train     0.711168    0.677399
MAPE_train  339.733505  285.979939
R2_train     -1.845736   -0.501867
MAE_valid     0.547148    0.462932
MSE_valid     0.653039    0.547252
MAPE_valid  413.987064  472.522831
R2_valid     -1.296782   -1.303908
MAE_test      0.693992    0.393509
MSE_test      1.017638    0.452002
MAPE_test   342.065954  849.510670
R2_test      -3.652084   -2.897339
```

[11]:
```
SVR_stats
```

[11]:

	28	37	40	42	105	\
MAE_train	0.276727	0.278257	0.300789	0.305370	0.232311	
MSE_train	0.380702	0.337298	0.311145	0.365504	0.234521	
MAPE_train	84.879118	62.266392	83.291656	80.413681	79.881161	
R2_train	-0.124635	0.315557	-0.012419	-0.614268	0.498649	
MAE_valid	0.507002	0.450078	0.628380	0.642011	0.560314	
MSE_valid	0.686971	0.557940	0.816743	0.936991	0.653306	
MAPE_valid	410.972500	327.959371	140.531075	171.987736	312.973809	
R2_valid	-1.359576	-1.271324	-4.067409	-7.953262	-1.134099	
MAE_test	0.597613	0.633618	0.793601	0.877509	0.710654	
MSE_test	0.787205	0.843558	1.477454	1.463107	0.874619	
MAPE_test	387.276030	254.948664	179.160440	221.490717	303.822279	
R2_test	-2.779046	-0.792251	-13.239907	-19.774878	-1.931432	

	115	56	51	58	70	\
MAE_train	0.295125	0.283173	0.302390	0.286905	0.310675	
MSE_train	0.321464	0.305294	0.339180	0.387236	0.384081	
MAPE_train	119.038558	68.282992	202.916551	86.871278	172.923613	
R2_train	0.226565	0.348702	0.236173	-0.022270	0.065745	
MAE_valid	0.554719	0.517377	0.486813	0.473341	0.486178	
MSE_valid	0.699144	0.653934	0.588591	0.633805	0.659440	
MAPE_valid	369.857812	267.802310	529.386902	359.456778	457.405472	
R2_valid	-1.440748	-1.867370	-1.143395	-1.267273	-3.107773	
MAE_test	0.598833	0.597459	0.612799	0.597962	0.660978	
MSE_test	0.733443	0.669165	0.751899	0.807331	0.813254	
MAPE_test	881.889153	506.443977	262.520504	321.102309	151.570201	
R2_test	-3.246958	-1.701439	-2.226823	-1.958748	-1.641215	

	99	100
MAE_train	0.303954	0.295758
MSE_train	0.408382	0.410984
MAPE_train	89.331871	81.153083
R2_train	-0.299458	0.125979
MAE_valid	0.510639	0.439388
MSE_valid	0.687648	0.551771
MAPE_valid	413.977242	548.037052
R2_valid	-2.045747	-0.842302
MAE_test	0.658883	0.444745
MSE_test	1.104124	0.480519
MAPE_test	222.248292	1088.957024
R2_test	-9.888161	-1.761574

[12]:
```
LSTM_stats
```

[12]:

	28	37	40	42	105	\
MAE_train	0.801216	0.674397	0.840092	0.892379	0.729738	
MSE_train	1.567037	1.222761	1.534189	1.680891	1.239169	

```
MAPE_train  1505.288920  253.155119  394.280669  316.342115  299.511297
R2_train      -0.572703   -0.238018   -0.524647   -0.675856   -0.312994
MAE_valid      0.826287    0.758647    0.820958    0.833089    0.775359
MSE_valid      1.595593    1.476596    1.529042    1.572665    1.360227
MAPE_valid   525.755758  361.544840  342.262619  335.589447  282.519522
R2_valid      -0.623602   -0.502556   -0.504593   -0.583000   -0.362309
MAE_test       0.905530    0.786979    0.847474    0.939782    0.845244
MSE_test       1.749422    1.265668    1.596610    1.679947    1.345584
MAPE_test    492.233166  276.060807  254.984764  264.064464  404.653650
R2_test       -0.698950   -0.248797   -0.534539   -0.694085   -0.366391

                   115         56          51          58          70  \
MAE_train      0.837738    0.789482    0.728637    0.747396    0.752254
MSE_train      1.468918    1.325196    1.225353    1.465135    1.428758
MAPE_train  1188.914157  400.904908  425.335048  369.407404  336.497228
R2_train      -0.434074   -0.438977   -0.297915   -0.458357   -0.449170
MAE_valid      0.857247    0.890307    0.780276    0.800333    0.789230
MSE_valid      1.570158    1.741660    1.405798    1.604110    1.765823
MAPE_valid   513.454756  461.088177  719.127897  1221.929490  436.251870
R2_valid      -0.547536   -0.738205   -0.418085   -0.606303   -0.755228
MAE_test       0.950815    0.955439    0.915303    0.858847    0.936715
MSE_test       1.727394    1.947341    1.583934    1.663366    1.643358
MAPE_test    229.327720  787.189624  467.433130  596.200090  846.378516
R2_test       -0.711865   -0.894008   -0.538744   -0.722833   -0.670029

                   99         100
MAE_train      0.832181    0.681621
MSE_train      1.552895    1.248868
MAPE_train  532.709128  362.735830
R2_train      -0.601468   -0.382725
MAE_valid      0.789553    0.729572
MSE_valid      1.449562    1.536009
MAPE_valid  477.224885  590.717876
R2_valid      -0.452772   -0.535136
MAE_test       0.906124    0.726834
MSE_test       1.767754    1.830458
MAPE_test   479.242091  346.972896
R2_test       -0.839436   -0.866999
```

[13]:
```
LSTM_stats.to_pickle("LSTM_statistics")
SVR_stats.to_pickle("SVR_statistics")
MVLR_stats.to_pickle("MVLR_statistics")
NN_stats.to_pickle("NN_statistics")
```

[14]:
```
pd.read_pickle("LSTM_statistics").to_excel("LSTM.xlsx")
pd.read_pickle("SVR_statistics").to_excel("SVR.xlsx")
pd.read_pickle("MVLR_statistics").to_excel("MVLR.xlsx")
```

```
pd.read_pickle("NN_statistics").to_excel("NN.xlsx")
```

3 summarize stats with mean

[15]:
```
(NN_stats).mean(axis=1)
```

[15]:
```
MAE_train       0.099903
MSE_train       0.018015
MAPE_train     32.121483
R2_train        0.939434
MAE_valid       0.708751
MSE_valid       0.917180
MAPE_valid    178.248775
R2_valid      -20.829863
MAE_test        0.797947
MSE_test        1.301575
MAPE_test     214.170959
R2_test       -22.986931
dtype: float64
```

[16]:
```
(MVLR_stats).mean(axis=1)
```

[16]:
```
MAE_train       0.539949
MSE_train       0.620129
MAPE_train    367.793026
R2_train       -1.015387
MAE_valid       0.537139
MSE_valid       0.633238
MAPE_valid    356.111976
R2_valid       -1.270937
MAE_test        0.626953
MSE_test        0.796143
MAPE_test     441.302456
R2_test        -1.914809
dtype: float64
```

[17]:
```
(LSTM_stats).mean(axis=1)
```

[17]:
```
MAE_train       0.775594
MSE_train       1.413264
MAPE_train    532.090152
R2_train       -0.448909
MAE_valid       0.804238
MSE_valid       1.550604
MAPE_valid    522.288928
```

```
R2_valid       -0.552444
MAE_test        0.881257
MSE_test        1.650070
MAPE_test     453.728410
R2_test        -0.648890
dtype: float64
```

[18]:
```
(SVR_stats).mean(axis=1)
```

[18]:
```
MAE_train       0.289286
MSE_train       0.348816
MAPE_train    100.937496
R2_train        0.062027
MAE_valid       0.521353
MSE_valid       0.677190
MAPE_valid    359.195671
R2_valid       -2.291690
MAE_test        0.648721
MSE_test        0.900473
MAPE_test     398.452466
R2_test        -5.078536
dtype: float64
```