# DATA aumentation on TERMINET SCHN dataset

In [42]:
```
import pandas as pd
import numpy as np
import torch
from torch import nn
from torch import optim
from sklearn.preprocessing import StandardScaler
from functools import partial
import deep_tabular_augmentation as dta
from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt
```

In [43]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
DATA_PATH = 'SOE_28-03-2022.xlsx - Sheet2.csv'
df = pd.read_csv(DATA_PATH)
df
```

Out[43]:

| | logID | DATE | LOCAL | NAME | DESC | VAL | QF | SOURCE |
|---|---|---|---|---|---|---|---|---|
| **0** | 13805 | 3/28/22 12:41 | N | CPU_USE | NaN | 77.86260 | 0x00001000 | PLC |
| **1** | 13808 | 3/28/22 12:41 | N | FAIL_CONF | NaN | 0.00000 | 0x00001000 | PLC |
| **2** | 13809 | 3/28/22 12:41 | N | FAIL_RTU | NaN | 0.00000 | 0x00001000 | PLC |
| **3** | 13806 | 3/28/22 12:41 | N | MEM_USE | NaN | 62.66120 | 0x00001000 | PLC |
| **4** | 13804 | 3/28/22 12:41 | N | PS1_V | NaN | 5.38353 | 0x00001000 | PLC |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **2635** | 11174 | 3/24/22 14:39 | N | MEM_USE | NaN | 34.79740 | 0x00001000 | PLC |
| **2636** | 11172 | 3/24/22 14:39 | N | PS1_V | NaN | 5.38353 | 0x00001000 | PLC |
| **2637** | 11178 | 3/24/22 14:39 | N | RTU_HEALTH | NaN | 3.00000 | 0x00001000 | PLC |
| **2638** | 11175 | 3/24/22 14:39 | N | TEMP | NaN | 31.75000 | 0x00001000 | PLC |
| **2639** | 11159 | 3/24/22 14:19 | N | CPU_USE | NaN | 59.02260 | 0x00001000 | PLC |

2640 rows × 8 columns

In [44]:
```
df[df["NAME"]=="CPU_USE"]["VAL"]
```

Out[44]:
```
0       77.8626
7       76.6798
8       77.7382
21      77.3428
28      77.8626
         ...
2611    60.2317
2612    60.2327
2625    59.9212
2632    59.1603
2639    59.0226
Name: VAL, Length: 378, dtype: float64
```

## There is the need to reshape the data grouping by same date

In [65]:
```
CPUs = df[df["NAME"] == "CPU_USE"]["VAL"].dropna().tolist()[:-1]
FAIL_CONFs = df[df["NAME"] == "FAIL_CONF"]["VAL"].dropna().tolist()
FAIL_RTUs = df[df["NAME"] == "FAIL_RTU"]["VAL"].dropna().tolist()
MEM_USEs = df[df["NAME"] == "MEM_USE"]["VAL"].dropna().tolist()
PS1_Vs = df[df["NAME"] == "PS1_V"]["VAL"].dropna().tolist()
RTU_HEALTHs = df[df["NAME"] == "RTU_HEALTH"]["VAL"].dropna().tolist()
TEMPs = df[df["NAME"] == "TEMP"]["VAL"].dropna().tolist()

print("len(CPUs):", len(CPUs))
print("len(FAIL_CONFs):", len(FAIL_CONFs))
print("len(FAIL_RTUs):", len(FAIL_RTUs))
print("len(MEM_USEs):", len(MEM_USEs))
print("len(PS1_Vs):", len(PS1_Vs))
print("len(RTU_HEALTHs):", len(RTU_HEALTHs))
print("len(TEMPs):", len(TEMPs))
```

```
len(CPUs): 377
len(FAIL_CONFs): 377
len(FAIL_RTUs): 377
len(MEM_USEs): 377
len(PS1_Vs): 377
len(RTU_HEALTHs): 377
len(TEMPs): 377
```

In [70]:
```
df_dict = {"CPUs": CPUs, "FAIL_CONFs": FAIL_CONFs, "FAIL_RTUs": FAIL_RTUs, "ME
df_reshaped = pd.DataFrame.from_dict(df_dict)
df_reshaped.head()
```

Out[70]:

| | CPUs | FAIL_CONFs | FAIL_RTUs | MEM_USEs | PS1_Vs | RTU_HEALTHs | TEMPs |
|---|---|---|---|---|---|---|---|
| **0** | 77.8626 | 0.0 | 0.0 | 62.6612 | 5.38353 | 3.0 | 31.125 |
| **1** | 76.6798 | 0.0 | 0.0 | 62.6854 | 5.38353 | 1.0 | 31.187 |
| **2** | 77.7382 | 0.0 | 0.0 | 62.6848 | 5.38453 | 3.0 | 31.188 |
| **3** | 77.3428 | 0.0 | 0.0 | 62.6844 | 5.40841 | 3.0 | 31.124 |
| **4** | 77.8626 | 0.0 | 0.0 | 62.6628 | 5.38353 | 3.0 | 31.000 |

# Data augmentation

In [74]:
```
X_train, X_test = train_test_split(df_reshaped, test_size=0.1, random_state=42
x_scaler = StandardScaler()
X_train_scaled = x_scaler.fit_transform(X_train)
X_test_scaled = x_scaler.transform(X_test)
pd.DataFrame(X_train_scaled, columns=list(df_reshaped.columns)).head()
```

Out[74]:

| | CPUs | FAIL_CONFs | FAIL_RTUs | MEM_USEs | PS1_Vs | RTU_HEALTHs | TEMPs |
|---|---|---|---|---|---|---|---|
| **0** | -0.256360 | 0.0 | 0.0 | -0.554650 | 0.378668 | 0.335658 | -0.548251 |
| **1** | -0.165282 | 0.0 | 0.0 | 0.402059 | 0.378668 | 0.335658 | -0.816868 |
| **2** | 0.468665 | 0.0 | 0.0 | 0.995027 | -2.099245 | 0.335658 | -0.552583 |
| **3** | 0.769275 | 0.0 | 0.0 | 0.567372 | 0.378668 | 0.335658 | 1.349398 |
| **4** | 0.784632 | 0.0 | 0.0 | 0.569413 | 0.286484 | 0.335658 | -0.279634 |

In [77]:
```
df_reshaped.shape
```

Out[77]: (377, 7)

In [79]:
```
datasets = dta.create_datasets_no_target_var(X_train_scaled, X_test_scaled)
data = dta.DataBunch(*dta.create_loaders(datasets, bs=1024))
```

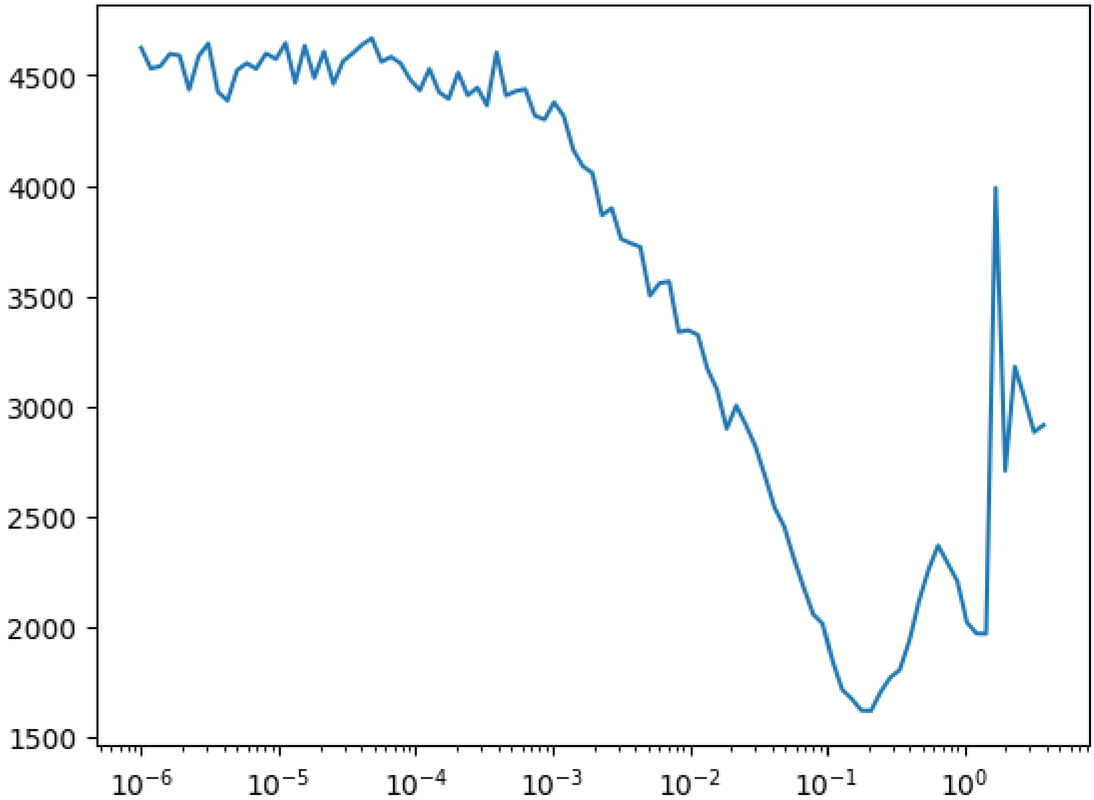
Now we have to define the Variational Encoder Architecture (here: 50->12->12->5->12->12->50) and then use the LearningRate Finder to tell us the best Learning rate:

In [84]:
```
D_in = X_train_scaled.shape[1]
VAE_arch = [50, 12, 12]
df_cols = list(df_reshaped.columns)

model = dta.Autoencoder(D_in, VAE_arch, latent_dim=5).to(device)
opt = optim.Adam(model.parameters(), lr=0.01)
loss_func = dta.customLoss()
learn = dta.Learner(model, opt, loss_func, data, cols=df_cols)

run = dta.Runner(cb_funcs=[dta.LR_Find, dta.Recorder])
run.fit(100, learn)
```

In [88]:
```
run.recorder.plot(skip_last=5)
```




So we can set up a desirable learning rate and scheduler for our learning rate:

In [90]:
```
sched = dta.combine_scheds([0.3, 0.7], [dta.sched_cos(0.01, 0.1), dta.sched_co
```

Train the model

In [92]:
```
cbfs = [partial(dta.LossTracker, show_every=50), dta.Recorder, partial(dta.Par
model = dta.Autoencoder(D_in, VAE_arch, latent_dim=20).to(device)
opt = optim.Adam(model.parameters(), lr=0.01)
learn = dta.Learner(model, opt, loss_func, data, cols=df_cols)
run = dta.Runner(cb_funcs=cbfs)
run.fit(400, learn)
```

```
epoch: 50
train loss is: 4874.25341796875
validation loss is: 261.2872314453125
epoch: 100
train loss is: 2041.269287109375
validation loss is: 239.14183044433594
epoch: 150
train loss is: 1683.766357421875
validation loss is: 205.6879119873047
epoch: 200
train loss is: 1526.854248046875
validation loss is: 200.50686645507812
epoch: 250
train loss is: 1434.8919677734375
validation loss is: 189.18687438964844
epoch: 300
train loss is: 1375.500244140625
validation loss is: 181.38125610351562
epoch: 350
train loss is: 1332.343505859375
validation loss is: 176.0098114013672
epoch: 400
train loss is: 1298.6165771484375
validation loss is: 172.45680236816406
```

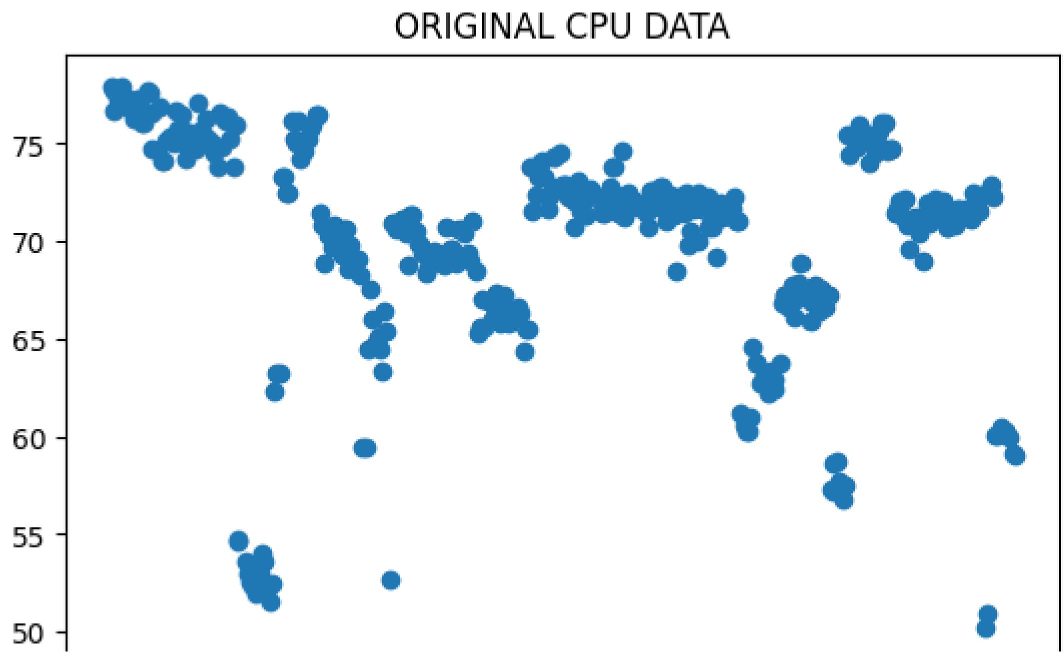
In [135]:
```
data_points = 1000
ake = run.predict_df(learn, no_samples=new_data_points, scaler=x_scaler)
list = list(df_reshaped.std()/5)
ake_with_noise = run.predict_with_noise_df(learn, no_samples=new_data_points, n
ake_with_noise.head()
```

Out[135]:

| | CPUs | FAIL_CONFs | FAIL_RTUs | MEM_USEs | PS1_Vs | RTU_HEALTHs | TEMPs |
|---|---|---|---|---|---|---|---|
| **0** | 80.135983 | 10.0 | 10.0 | 73.681166 | 15.382381 | 12.789905 | 41.477023 |
| **1** | 78.493321 | 10.0 | 10.0 | 71.674377 | 15.385260 | 12.984066 | 41.550801 |
| **2** | 81.278700 | 10.0 | 10.0 | 71.098531 | 15.381668 | 12.901114 | 41.482864 |
| **3** | 78.717785 | 10.0 | 10.0 | 70.603800 | 15.380286 | 13.215952 | 41.557542 |
| **4** | 81.259381 | 10.0 | 10.0 | 77.248663 | 15.379779 | 13.212596 | 41.363957 |

## Comparison between CPU data original and augmented

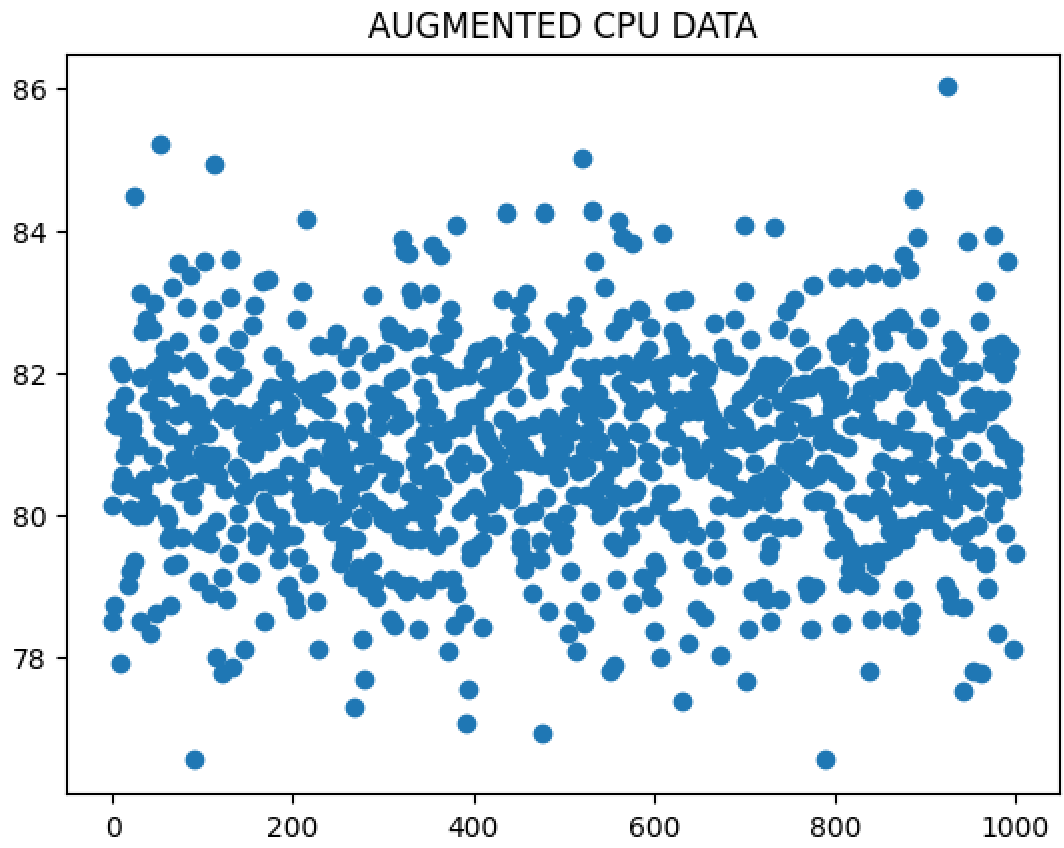
In [137]:
```
plt.scatter(df[df["NAME"]=="CPU_USE"]["VAL"].index.tolist(),df[df["NAME"]=="CP
plt.title("ORIGINAL CPU DATA")
plt.show()
```




In [140]:
```
df[df["NAME"]=="CPU_USE"]["VAL"].describe()
```

Out[140]:
```
count    378.000000
mean      69.665522
std        5.948844
min       43.395200
25%       67.193950
50%       71.260800
75%       73.333050
max       77.862600
Name: VAL, dtype: float64
```

In [142]:
```
plt.scatter(df_fake_with_noise["CPUs"].index.tolist(),df_fake_with_noise["CPUs
plt.title("AUGMENTED CPU DATA")
plt.show()
```



In [144]:
```
df_fake_with_noise["CPUs"].describe()
```

Out[144]:
```
count    1000.000000
mean       80.913865
std         1.358482
min        76.535951
25%        80.052997
50%        80.951565
75%        81.829621
max        86.018452
Name: CPUs, dtype: float64
```