Electricity

– Consumption Prediction

Yunpeng Wang

Agenda

I. Introduction
II. Data Overview & Data Preprocessing
III. DeepAR/ Temporal Fusion Transformer modeling
IV. Summary




Introduction

Electricity Load Diagrams Data Set from UCI Machine Learning Repository

Data Set

This data set contains electricity consumption of 370 clients who lives in Portugal

Project Objective

Perform a time series analysis and Predict the future electricity demands

Model utilized

Amazon Sagemaker DeepAR and Temporal Fusion Transformer (TFT) models




Data Overview & Data Preprocessing

Data Overview

	MT_001	MT_002	MT_003	MT_004	MT_005	MT_006	MT_007	MT_008	MT_009	MT_010	...	MT_361	MT_362	MT_363	MT_364	MT_365	MT_366	MT_367	MT_368	MT_369	MT_370
2011-01-01 00:15:00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2011-01-01 00:30:00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2011-01-01 00:45:00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2011-01-01 01:00:00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2011-01-01 01:15:00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2014-12-31 23:00:00	2.538071	22.048364	1.737619	150.406504	85.365854	303.571429	11.305822	282.828283	68.181818	72.043011	...	276.945039	28200.0	1616.033755	1363.636364	29.986962	5.851375	697.102722	176.961603	651.026393	7621.621622
2014-12-31 23:15:00	2.538071	21.337127	1.737619	166.666667	81.707317	324.404762	11.305822	252.525253	64.685315	72.043011	...	279.800143	28300.0	1569.620253	1340.909091	29.986962	9.947338	671.641791	168.614357	669.354839	6702.702703
2014-12-31 23:30:00	2.538071	20.625889	1.737619	162.601626	82.926829	318.452381	10.175240	242.424242	61.188811	74.193548	...	284.796574	27800.0	1556.962025	1318.181818	27.379400	9.362200	670.763828	153.589316	670.087977	6864.864865
2014-12-31 23:45:00	1.269036	21.337127	1.737619	166.666667	85.365854	285.714286	10.175240	225.589226	64.685315	72.043011	...	246.252677	28000.0	1443.037975	909.090909	26.075619	4.095963	664.618086	146.911519	646.627566	6540.540541
2015-01-01 00:00:00	2.538071	19.914651	1.737619	178.861789	84.146341	279.761905	10.175240	249.158249	62.937063	69.892473	...	188.436831	27800.0	1409.282700	954.545455	27.379400	4.095963	628.621598	131.886477	673.020528	7135.135135

140256 rows × 370 columns

- 370 columns(clients)
- 2011-01-01 00:15:00 to 2015-01-01 00:00:00
- Power_usage recorded in every 15 minutes

Data Preprocessing

- check missing data
- check data aligned with data description
- check outliers (negative values, consecutive daily zero consumption clients)

Hierarchical clustering:
Agglomerative cluster in different group by clients’ daily electricity usage

Diagnostics | Aggregation | Clustering

Sum the electricity consumption data by days for each client

Data Preprocessing

—Data Diagnostic

```
data.index

DatetimeIndex(['2011-01-01 00:15:00', '2011-01-01 00:30:00',
               '2011-01-01 00:45:00', '2011-01-01 01:00:00',
               '2011-01-01 01:15:00', '2011-01-01 01:30:00',
               '2011-01-01 01:45:00', '2011-01-01 02:00:00',
               '2011-01-01 02:15:00', '2011-01-01 02:30:00',
               ...
               '2014-12-31 21:45:00', '2014-12-31 22:00:00',
               '2014-12-31 22:15:00', '2014-12-31 22:30:00',
               '2014-12-31 22:45:00', '2014-12-31 23:00:00',
               '2014-12-31 23:15:00', '2014-12-31 23:30:00',
               '2014-12-31 23:45:00', '2015-01-01 00:00:00'],
              dtype='datetime64[ns]', length=140256, freq=None)
```

- 140256 = 1461 (days' count between 2011-01-01 and 2015-01-01) * 96 (24 hours * 4 quarters)
- This means no missing time slot

- No missing data!

```
data.isnull().values.any()

False
```

Data Preprocessing

－Data Diagnostic

- Data recorded in every 15 mins

```
data.index
```

```
DatetimeIndex(['2011-01-01 00:15:00', '2011-01-01 00:30:00',
               '2011-01-01 00:45:00', '2011-01-01 01:00:00',
               '2011-01-01 01:15:00', '2011-01-01 01:30:00',
               '2011-01-01 01:45:00', '2011-01-01 02:00:00',
               '2011-01-01 02:15:00', '2011-01-01 02:30:00',
               ...
               '2014-12-31 21:45:00', '2014-12-31 22:00:00',
               '2014-12-31 22:15:00', '2014-12-31 22:30:00',
               '2014-12-31 22:45:00', '2014-12-31 23:00:00',
               '2014-12-31 23:15:00', '2014-12-31 23:30:00',
               '2014-12-31 23:45:00', '2015-01-01 00:00:00'],
              dtype='datetime64[ns]', length=140256, freq=None)
```

```
from datetime import datetime
j = "2011-01-01 00:15:00"
j = datetime.strptime(j,'%Y-%m-%d %H:%M:%S')
diff = []
for i in data.index:
    if i == j:
        continue
    else:
        diff.append(i - j)
        j = i
```

```
all(element == diff[0] for element in diff)
```

```
True
```

Data Preprocessing

–Data Diagnostic

- The argument, that every year in March time change day (which has only 23 hours) the values between 1:00 am and 2:00 am are zero for all points, is invalid.

- The argument, that every year in October time change day (which has 25 hours) the values** between 1:00 am and 2:00 am** aggregate the consumption of two hours, is hard to prove its validity. However, based on observation on data diagram, this statement is false.

Data Preprocessing

–Data Diagnostic

- No negative valued data

```
(data < 0).values.any()
```

```
False
```

```
(data.values < 0.0).any()
```

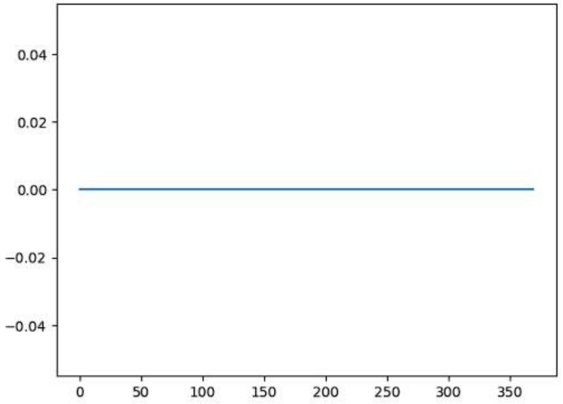
```
False
```

- No client has 30 consecutive daily zero consumption

```
consecutive_zeros = []
for i in a.columns:
    consecutive_zeros.append(len_consec_zeros(a.loc[:, i]))
```

```
plt.plot(consecutive_zeros)
```

```
[<matplotlib.lines.Line2D at 0x7f6ab02eb520>]
```




```
all(element == consecutive_zeros[0] for element in consecutive_zeros)
```

```
True
```

Data Preprocessing

—Data Aggregation

```
a = data.resample('D').sum()
a
```

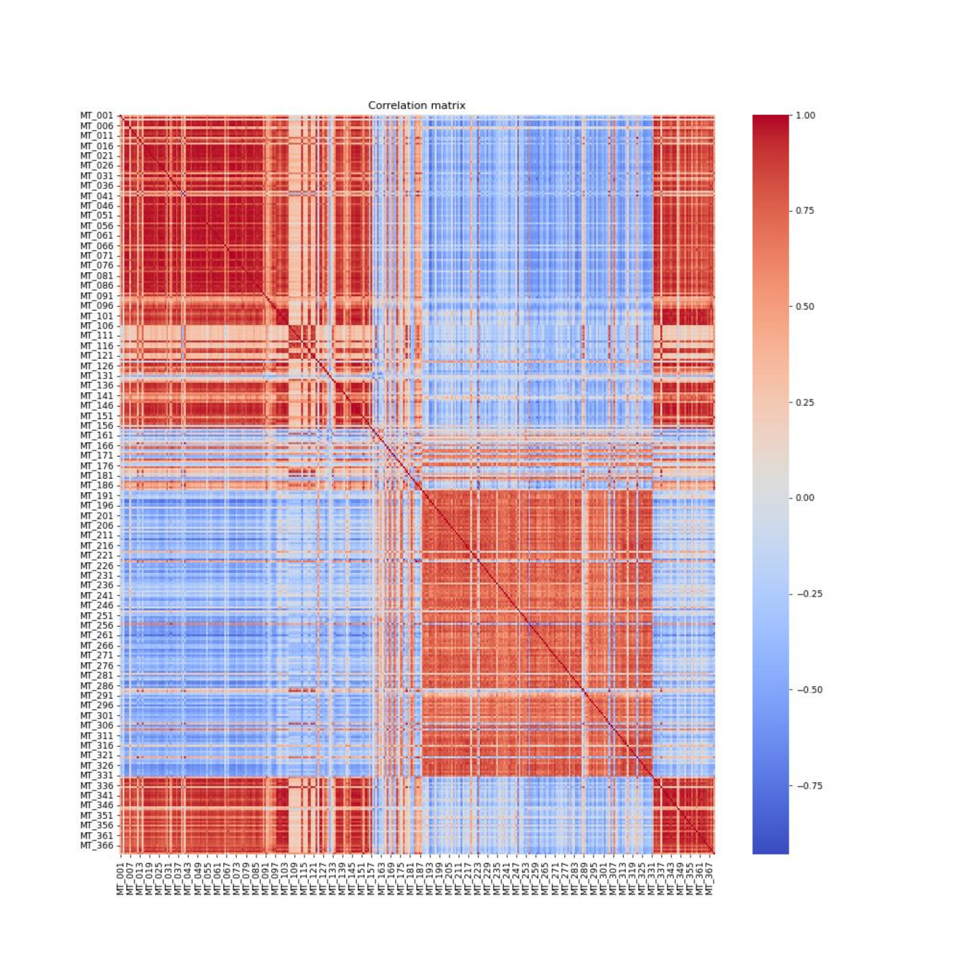
	MT_001	MT_002	MT_003	MT_004	MT_005	MT_006	MT_007	MT_008	MT_009	MT_010	...	MT_361	MT_362	MT_363	
2011-01-01	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0
2011-01-02	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0
2011-01-03	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0
2011-01-04	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0
2011-01-05	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2014-12-28	227.157360	2131.578947	151.172893	14327.235772	6776.829268	20122.023810	429.621255	25255.892256	5118.881119	4794.623656	...	28815.132049	3272100.0	220721.518987	257477
2014-12-29	248.730964	2212.660028	160.729800	14067.073171	7198.780488	22824.404762	550.593556	30286.195286	6697.552448	6337.634409	...	28825.124911	3109100.0	206852.320675	269090
2014-12-30	232.233503	2205.547653	165.073849	14290.650407	7189.024390	23880.952381	586.772188	30909.090909	6487.762238	6489.247312	...	28488.222698	2904300.0	204126.582278	263613
2014-12-31	229.695431	2273.115220	166.811468	14006.097561	7023.170732	23511.904762	690.785755	28700.336700	6211.538462	5034.408602	...	26970.735189	2748800.0	162556.962025	215886
2015-01-01	2.538071	19.914651	1.737619	178.861789	84.146341	279.761905	10.175240	249.158249	62.937063	69.892473	...	188.436831	27800.0	1409.282700	954

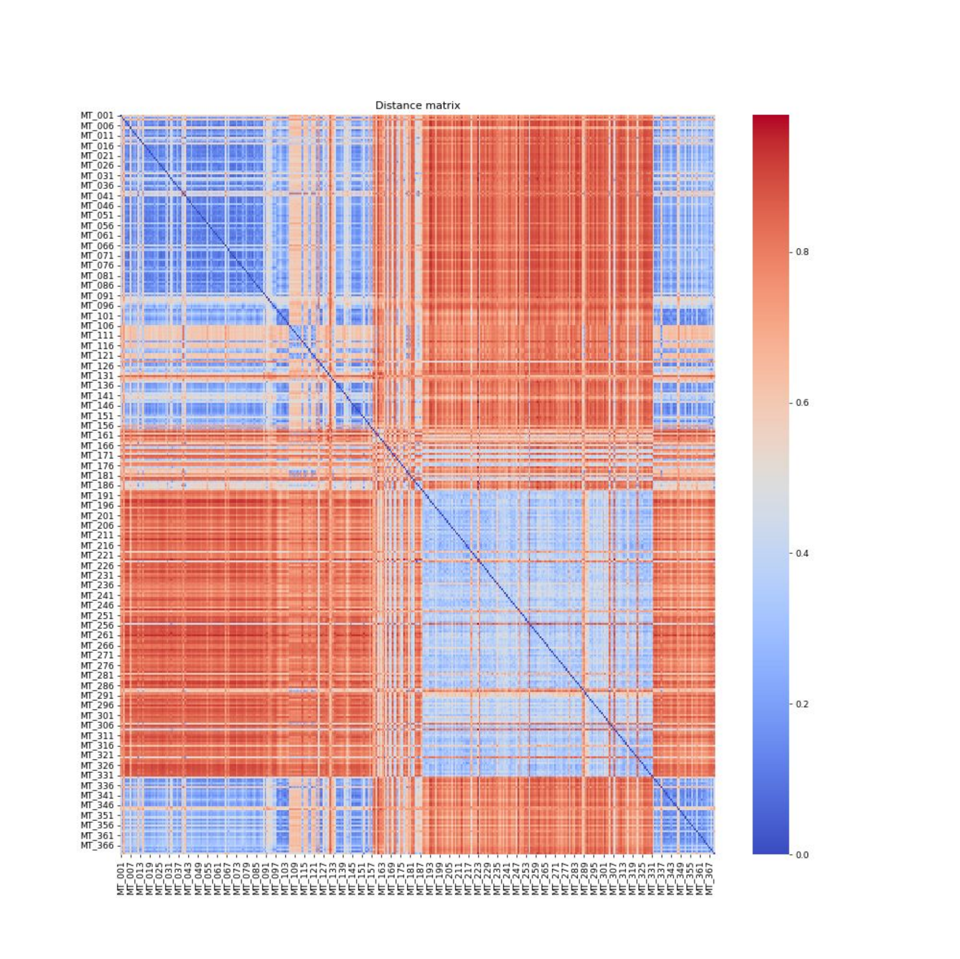
1462 rows × 370 columns

Data Preprocessing

–Data Clustering

```
corr_mat_a = a.corr(method='pearson')
dist_a = np.sqrt(0.5*(1-corr_mat_a))
```

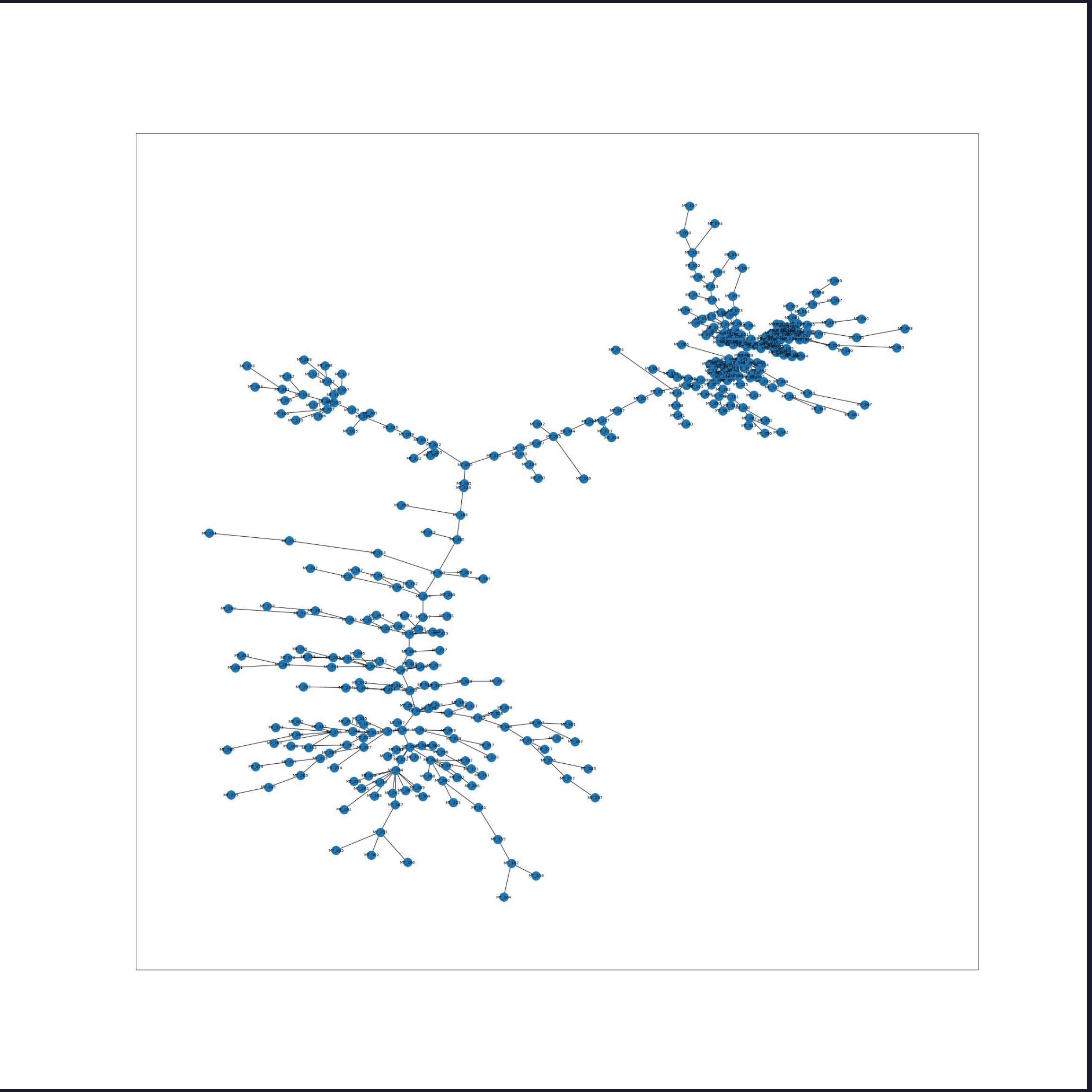




Data Preprocessing

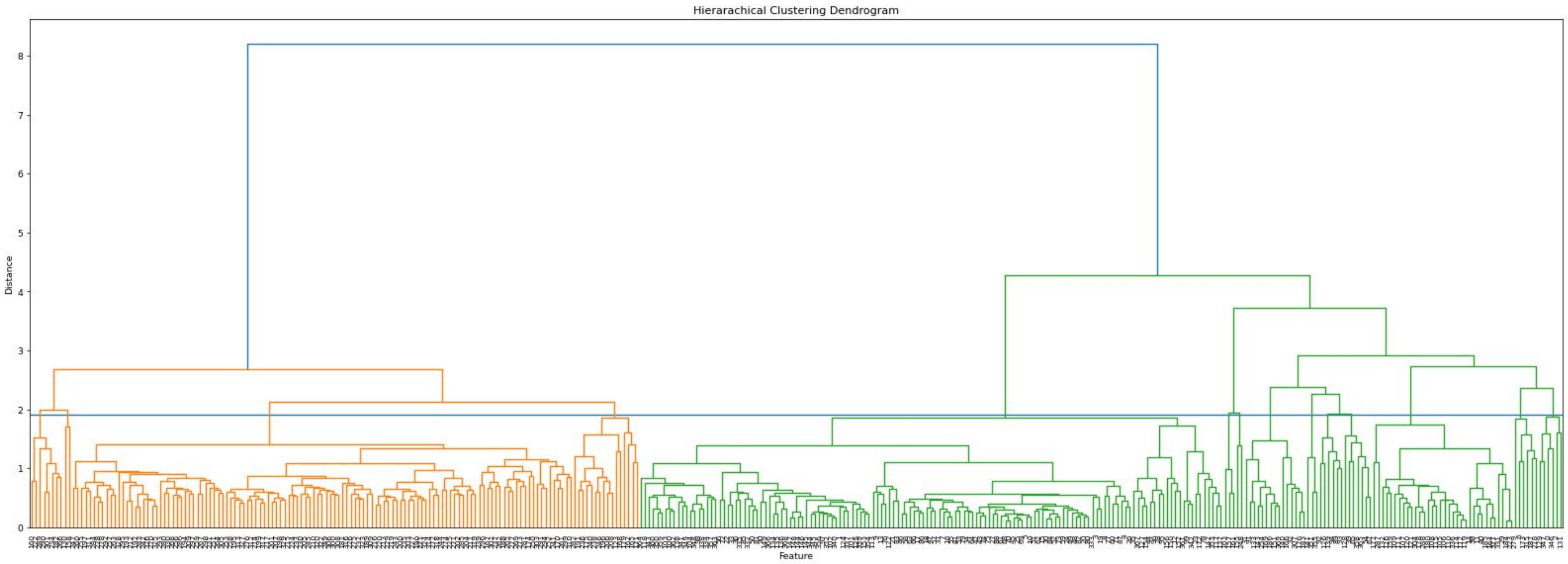
–Data Clustering

- minimum spanning tree structure incorporates hierarchical relationships



Data Preprocessing
–Data Clustering

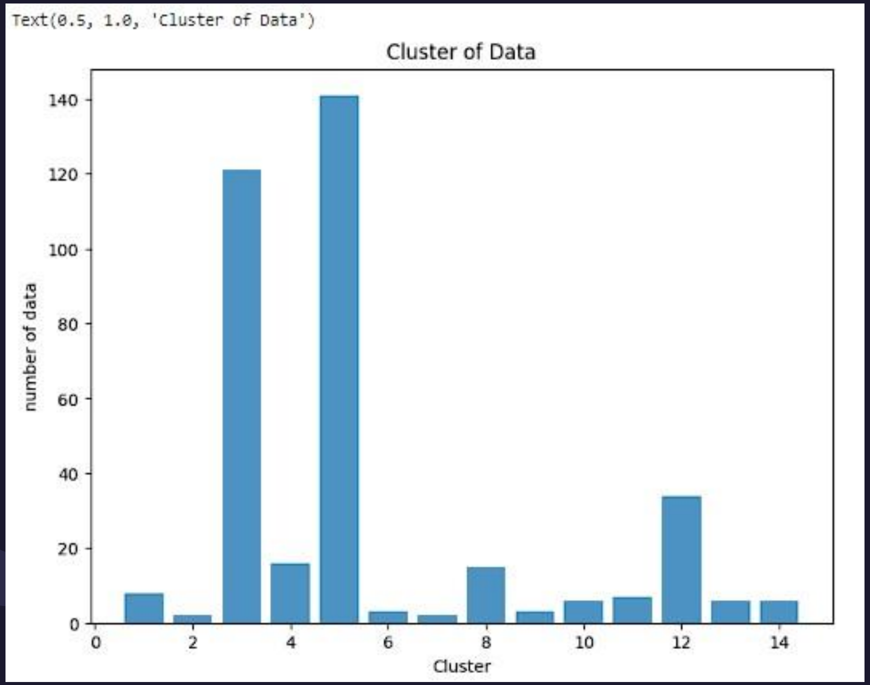
- Correlation-based Clustering Dendrogram



Data Preprocessing

–Data Clustering

- max_d = 1.9, decided by evaluating dendrogram visually
- Total number of clusters: 14



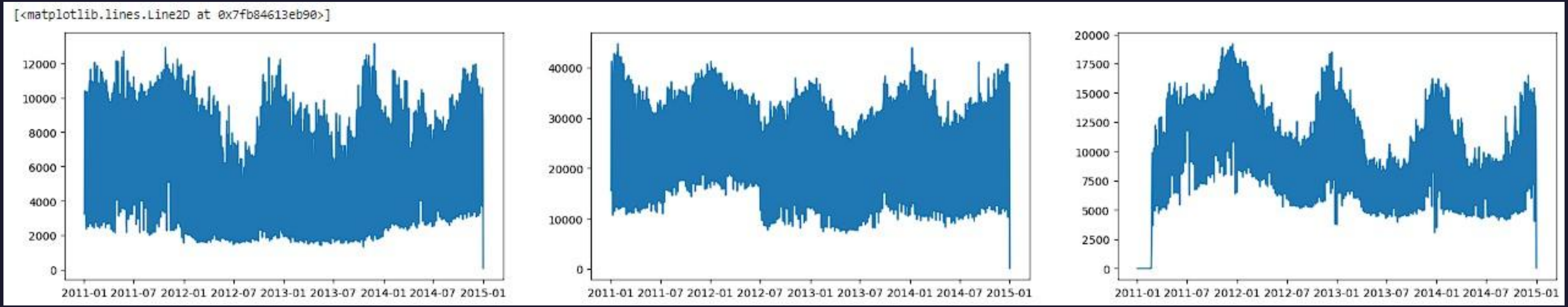

```python
from scipy.cluster.hierarchy import fcluster
max_d = 1.9
clusters_a = fcluster(link_a, t=max_d, criterion='distance')
```

```python
df_clust_a= pd.DataFrame({'Cluster':clusters_a, 'Feature':a.columns.values.astype('str')})
df_clust_a.groupby('Cluster').count()
```

	Feature
Cluster	
1	8
2	2
3	121
4	16
5	141
6	3
7	2
8	15
9	3
10	6
11	7
12	34
13	6
14	6

Data Preprocessing
—Data Clustering

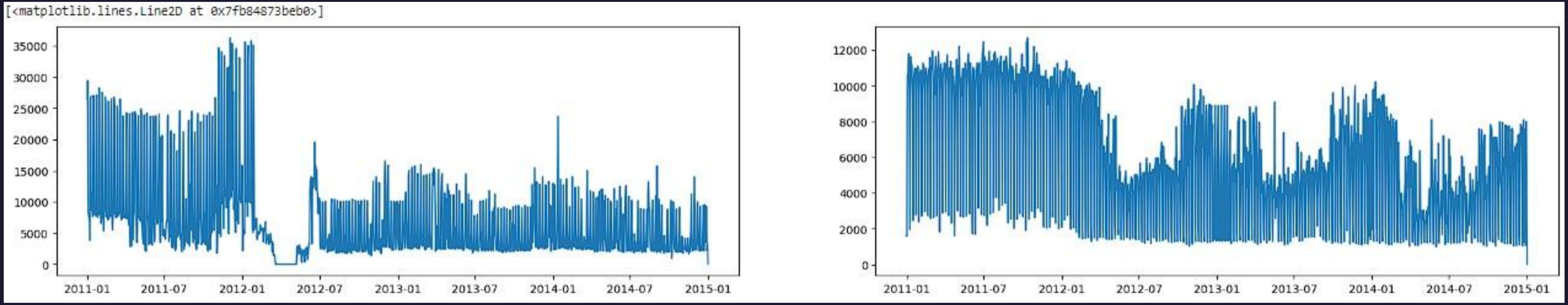
Visually check if customers within the same cluster have similar electricity consumption pattern



Cluster 6

Data Preprocessing
–Data Clustering

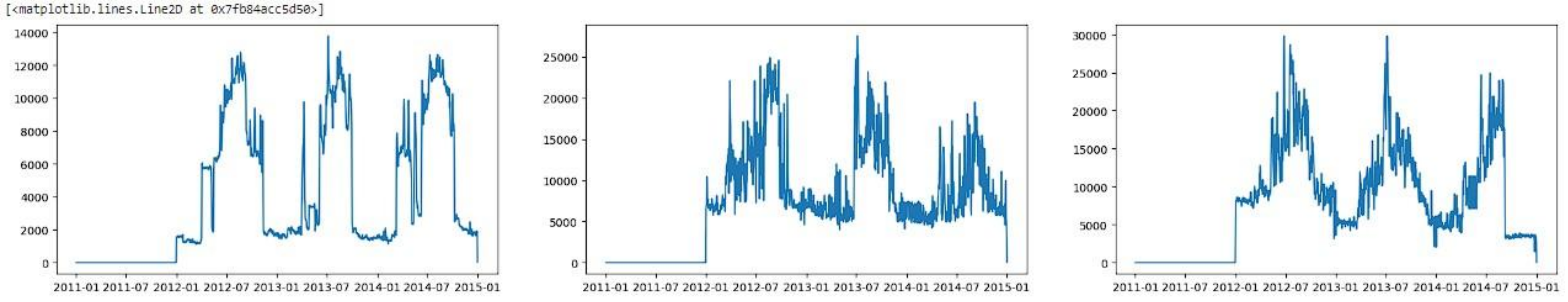
Visually check if customers within the same cluster have similar electricity consumption pattern



Cluster 7

Data Preprocessing
–Data Clustering

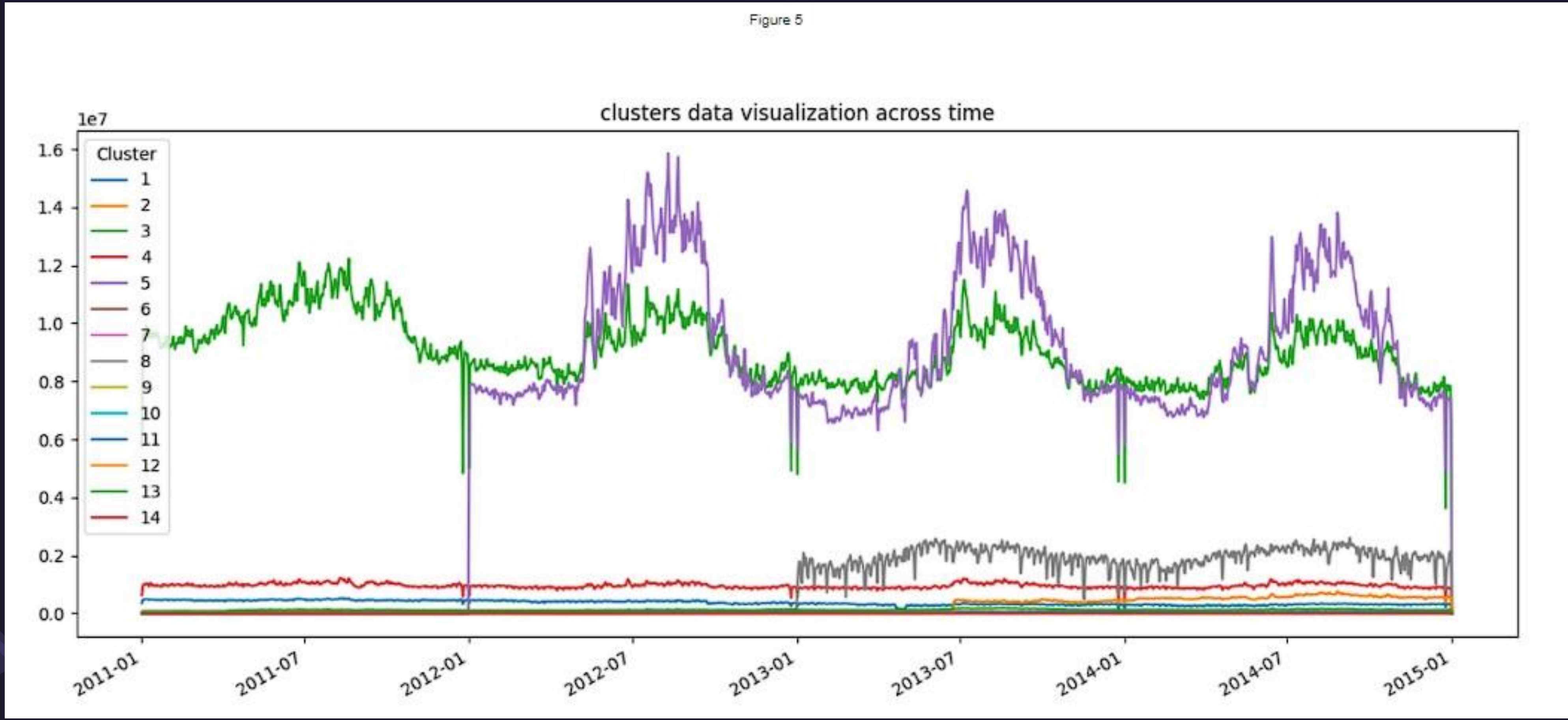
Visually check if customers within the same cluster have similar electricity consumption pattern



Cluster 9

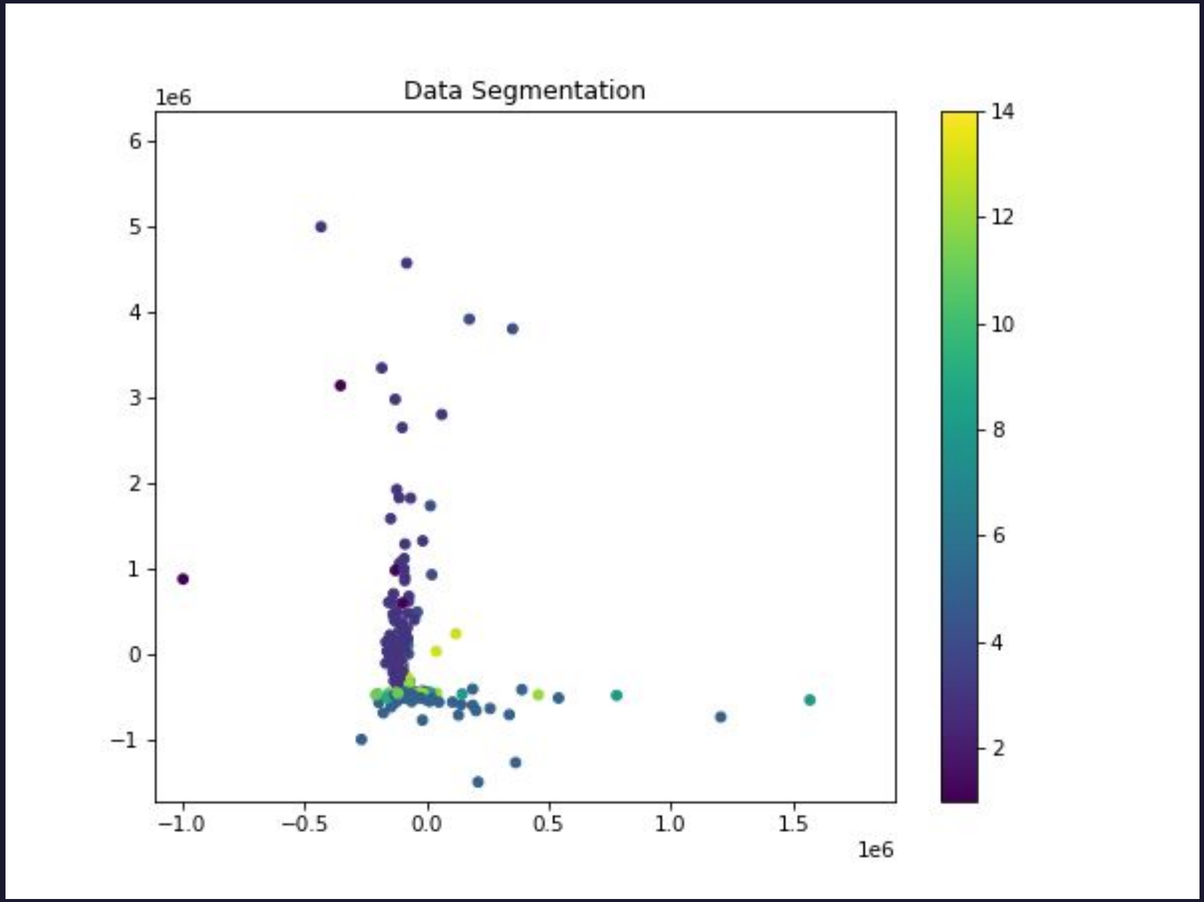
Data Preprocessing

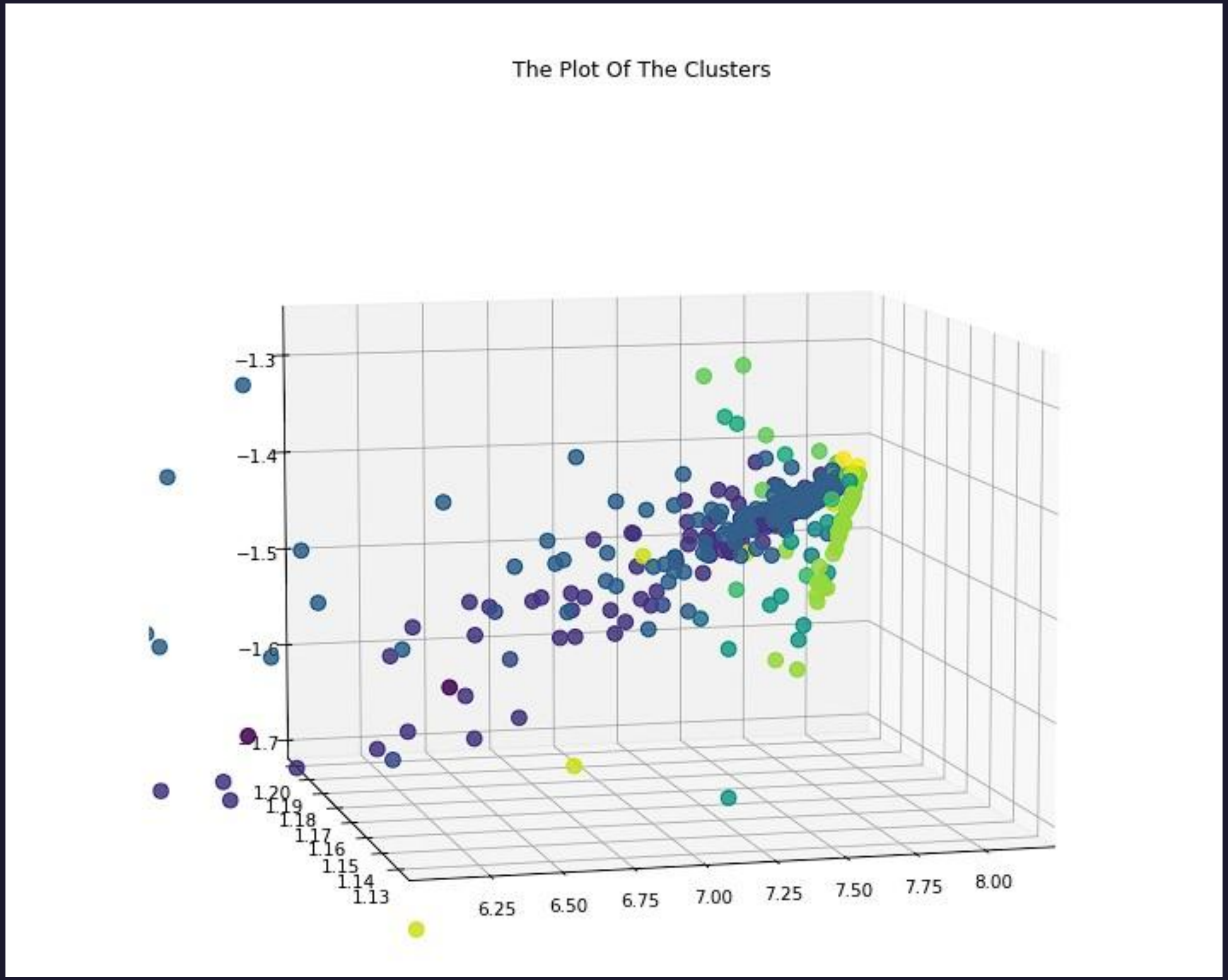
—Data Visualization



Data Preprocessing

—Data Visualization (PCA)








DeepAR/ Temporal Fusion Transformer modeling(TFT)

DeepAR modeling

– Introduction

- Amazon SageMaker is a fully managed machine learning service on Amazon Web Service(AWS)
- Amazon SageMaker provides machine learning (ML) capabilities that are purpose-built for data scientists and developers to prepare, build, train, and deploy high-quality ML models efficiently.
- The Amazon SageMaker DeepAR forecasting algorithm is a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN).
- DeepAR outperforms the standard ARIMA and ETS methods when the dataset contains hundreds of related time series.

DeepAR modeling

```
training_data_new_features = [
    {
        "start": str(start_dataset),
        "target": ts[
            start_dataset : end_training - timedelta(days=1)
        ].tolist(),
        "dynamic_feat": [w['TAVG'][start_dataset:end_training - timedelta(days=1)].tolist()],
    }
    for ts in timeseries1
]
print(len(training_data_new_features))
```

```
14
```

```
num_test_windows = 4

test_data_new_features = [
    {
        "start": str(start_dataset),
        "target": ts[start_dataset : end_training + timedelta(days=k * prediction_length)].tolist(),
        "dynamic_feat": [w['TAVG'][start_dataset:end_training + timedelta(days=k * prediction_length)].tolist()],
    }
    for k in range(1, num_test_windows + 1)
    for ts in timeseries1
]
print(len(test_data_new_features))
```

```
56
```

```
estimator_new_features = sagemaker.estimator.Estimator(
    image_uri=image_name,
    sagemaker_session=sagemaker_session,
    role=role,
    train_instance_count=1,
    train_instance_type="ml.c4.2xlarge",
    base_job_name="deepar-electricity-demo-new-features",
    output_path=s3_output_path_new_features,
)

hyperparameters = {
    "time_freq": freq,
    "epochs": "1000",
    "early_stopping_patience": "20",
    "mini_batch_size": "64",
    "learning_rate": "5E-4",
    "context_length": str(context_length),
    "prediction_length": str(prediction_length),
    "num_dynamic_feat": "auto",  # this will use the `dynamic_feat` field if
}
estimator_new_features.set_hyperparameters(**hyperparameters)

estimator_new_features.fit(
    inputs={
        "train": "{}/train/".format(s3_data_path_new_features),
        "test": "{}/test/".format(s3_data_path_new_features),
    },
    wait=True,
)
```

DeepAR modeling

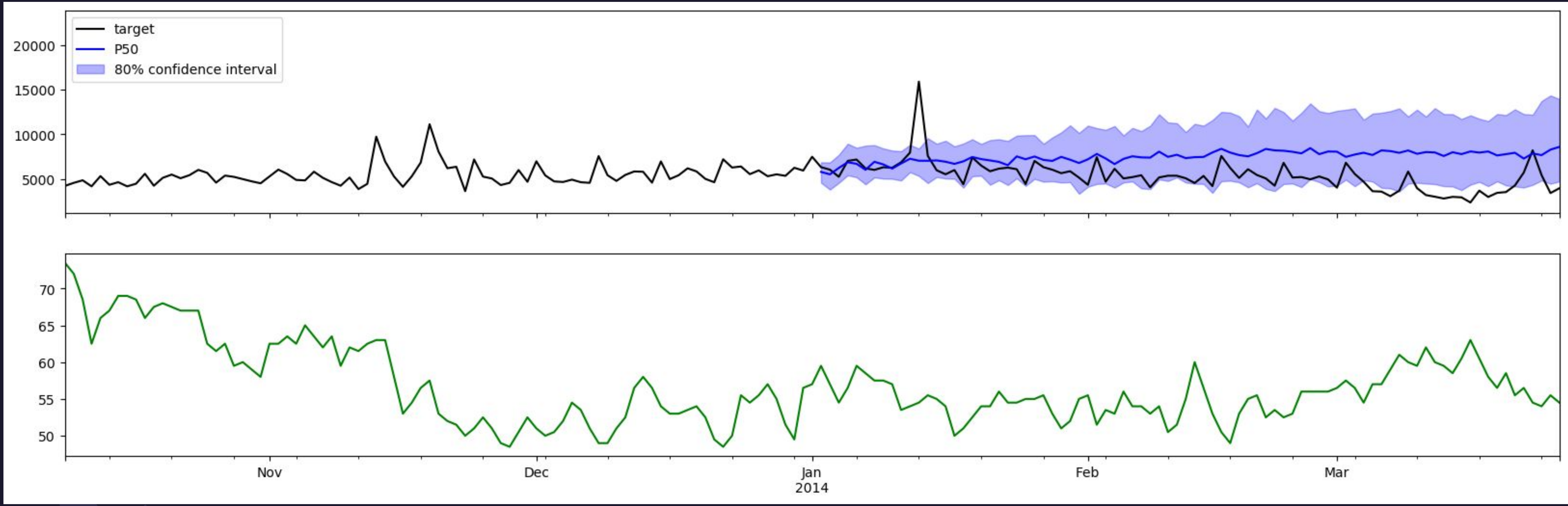
```
@interact_manual(
    customer_id=IntSlider(min=0, max=13, value=1, style=style),
    forecast_day=IntSlider(min=0, max=336, value=10, style=style),
    confidence=IntSlider(min=60, max=95, value=80, step=5, style=style),
    history_weeks_plot=IntSlider(min=1, max=10, value=1, style=style),
    show_samples=Checkbox(value=False),
    continuous_update=False,
)
def plot_interact(customer_id, forecast_day, confidence, show_samples):
    forecast_date = end_training + datetime.timedelta(days=forecast_day)
    ts = timeseries1[customer_id]
    freq = ts.index.freq
    target = ts[start_dataset : forecast_date + prediction_length * freq]
    dynamic_feat = [w['TAVG'].tolist()]
    plot(
        predictor_new_features,
        target_ts=target,
        dynamic_feat=dynamic_feat,
        forecast_date=forecast_date,
        show_samples=show_samples,
        plot_history=7 * 12,
        confidence=confidence,
    )
```

Interactive interface – forecast of any customer at any point in (future) time

DeepAR modeling

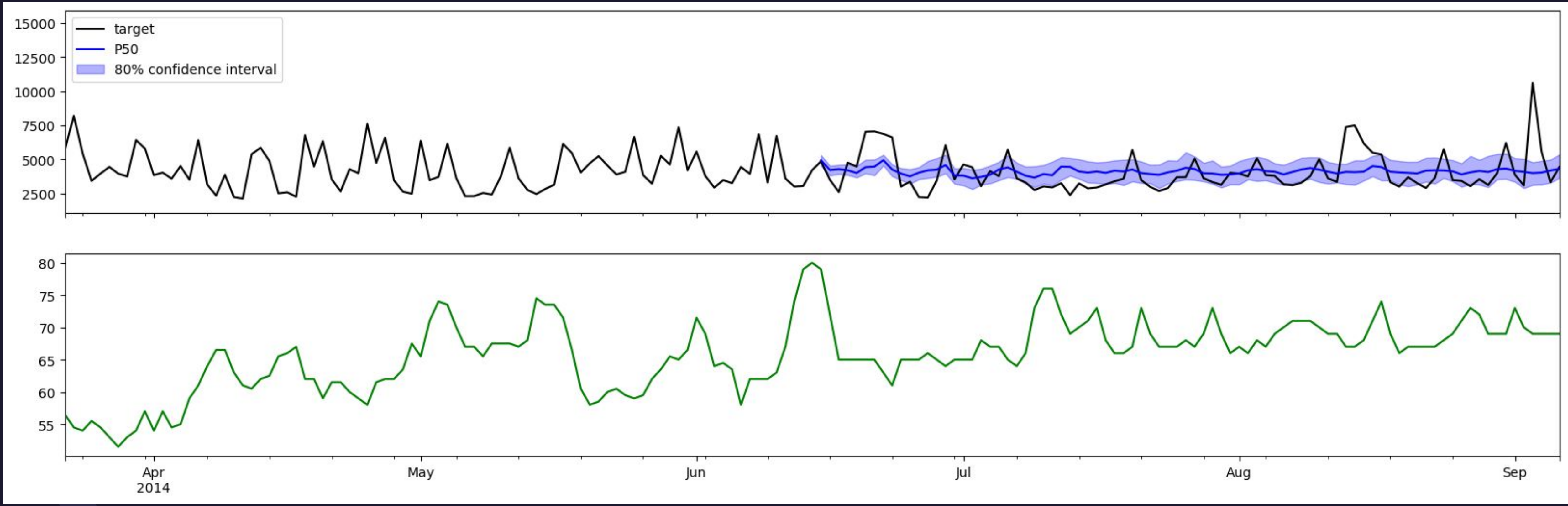
– Results

Cluster 1 validation



DeepAR modeling
– Results

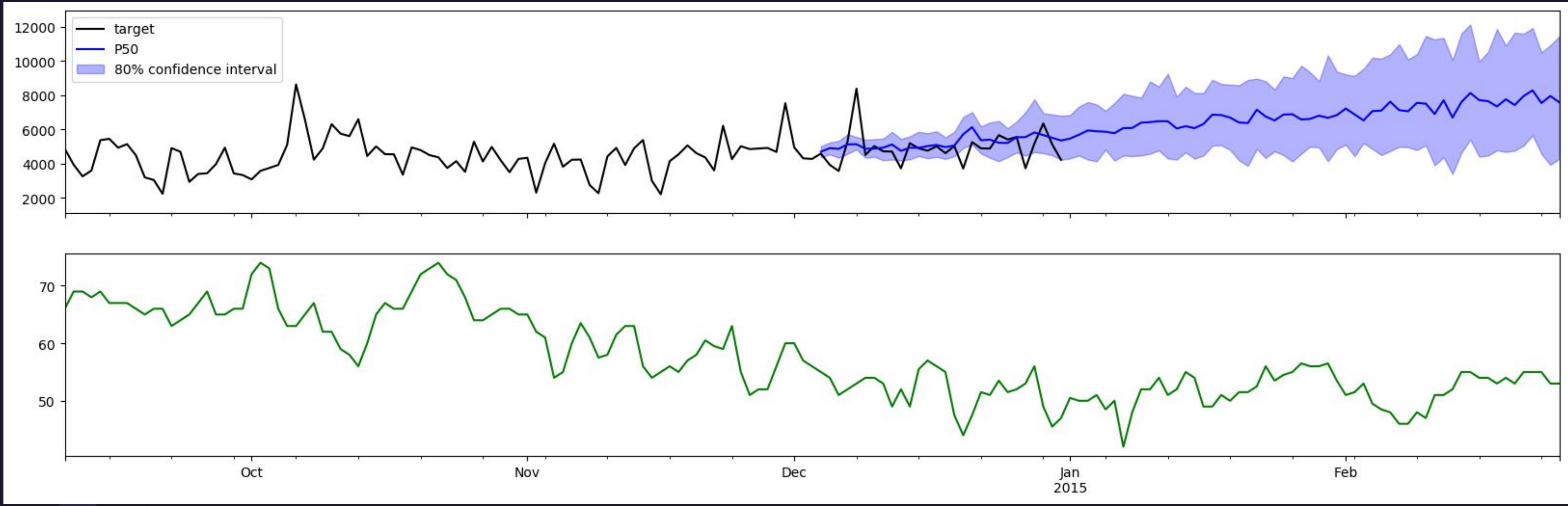
Cluster 1 test



DeepAR modeling

– Results

Cluster 1 future prediction



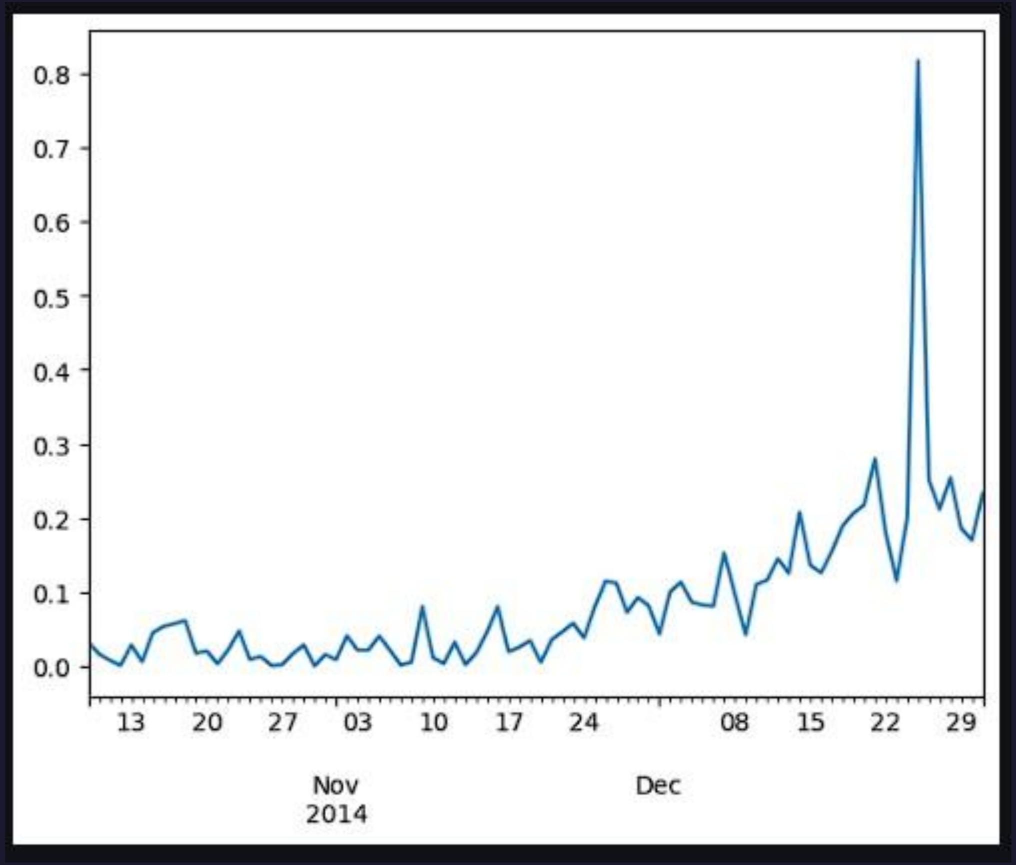
DeepAR modeling
– Results

Cluster 1

```
overall[0].MAPE.mean()
```

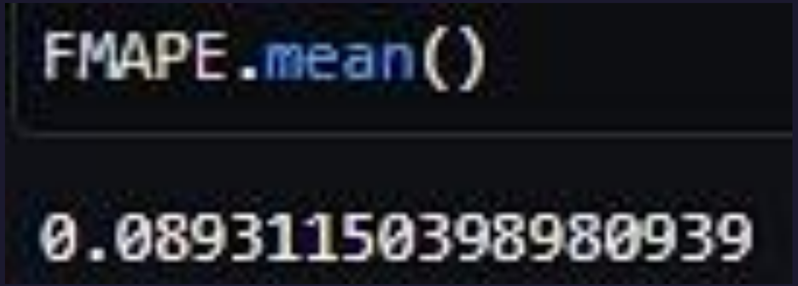
```
0.08568281996792973
```

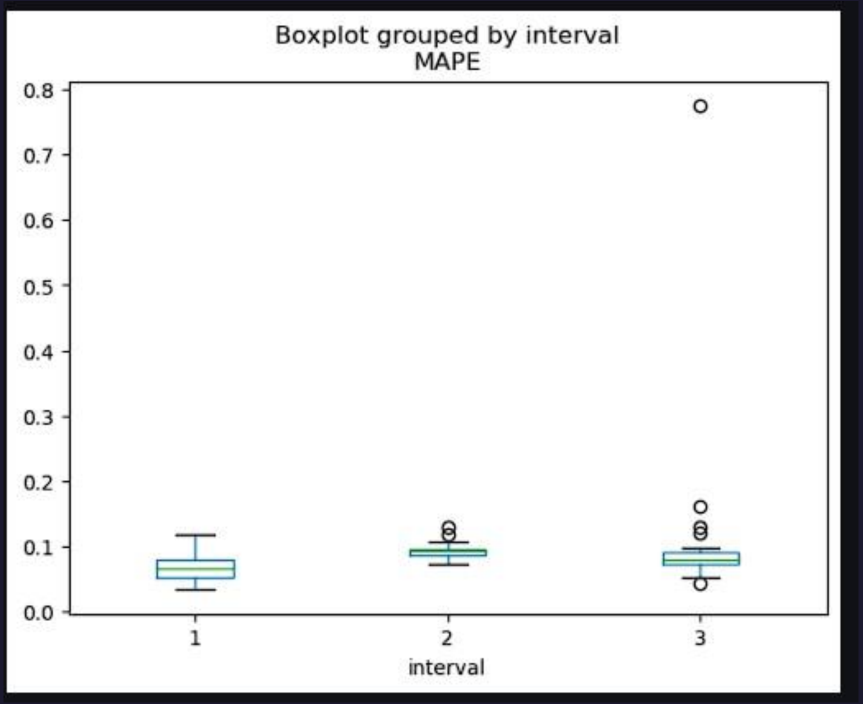
Above is MAPE on test set
Right is MAPE on test set plot over time



DeepAR modeling
– Results

- For other clusters' results, please check out outputs on Jupyter Notebook and technical documentation
- Overall MAPE are demonstrated as graph below





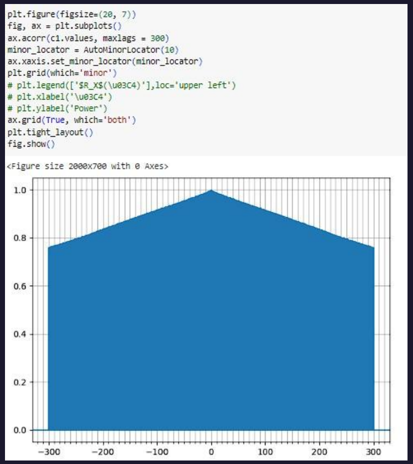
Temporal Fusion Transformer modeling – Introduction

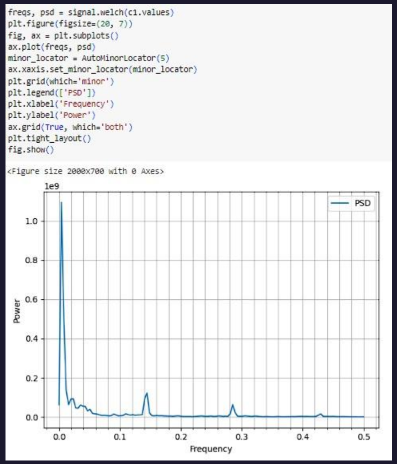
- Temporal Fusion Transformer (TFT) is a transformer-based model that leverages self-attention to capture the complex temporal dynamics of multiple time sequences

- TFT supports:
  - multiple time series
  - multi-horizon forecasting
  - heterogeneous features
  - interpretable predictions

Temporal Fusion Transformer modeling

– Pre-modeling (feature engineering)

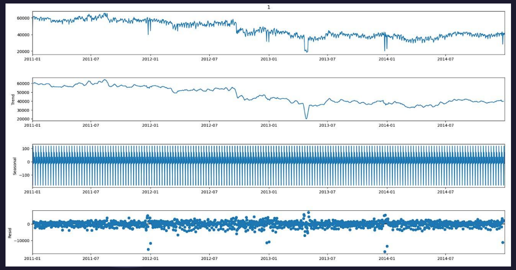




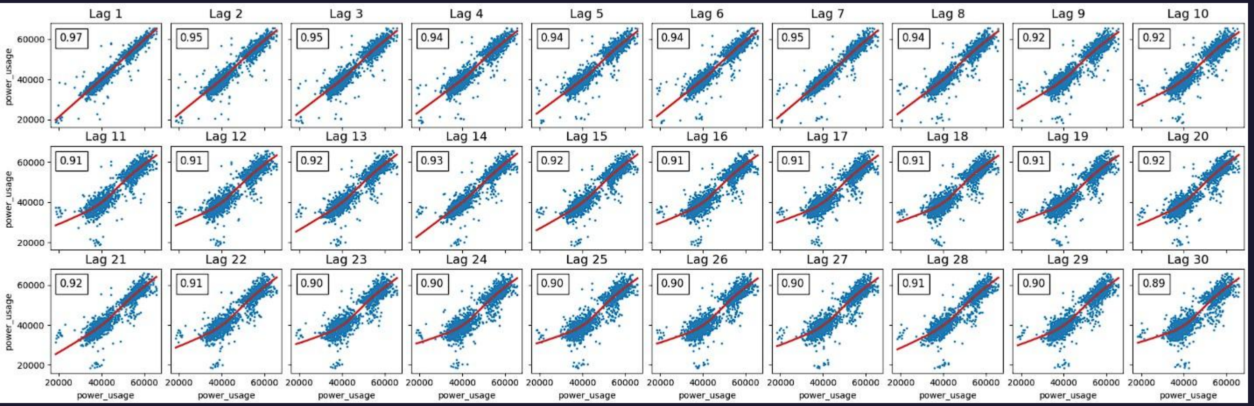
- ***DeterministicProcess, CalendarFourier*** from statsmodels.tsa.deterministic
- Merge Lisbon's daily avg/max/min temperature and precipitation
- ***plot_pacf*** from statsmodels.graphics.tsaplots

Data seasonality and trend analysis

Derived variable creation



Autocorrelation
Power Spectral Density
Seasonality decomposition



Temporal Fusion Transformer modeling

```python
early_stop_callback = EarlyStopping(monitor="val_loss", min_delta=1, patience=10, verbose=True, mode="min")
lr_logger = LearningRateMonitor()
logger = TensorBoardLogger("lightning_logs")
trainer1 = pl.Trainer(
    max_epochs=100,
    accelerator='gpu',
    devices=1,
    enable_model_summary=True,
    gradient_clip_val=0.1,
    callbacks=[lr_logger, early_stop_callback],
    logger=logger)
tft1 = TemporalFusionTransformer.from_dataset(
    training1,
    learning_rate=0.1,
    hidden_size=160,
    attention_head_size=4,
    dropout=0.1,
    hidden_continuous_size=160,
    output_size=7,  # there are 7 quantiles by default: [0.02, 0.1, 0.25, 0.5, 0.75, 0.9, 0.98]
    loss=QuantileLoss(),
    log_interval=10,
    reduce_on_plateau_patience=4)
```

```python
trainer1.fit(tft1,
    train_dataloaders=train_dataloader1,
    val_dataloaders=val_dataloader1)
```

```
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:
   | Name                               | Type                            | Params
----------------------------------------------------------------------------------------
0  | loss                               | QuantileLoss                    | 0
1  | logging_metrics                    | ModuleList                      | 0
2  | input_embeddings                   | MultiEmbedding                  | 1
3  | prescalers                         | ModuleDict                      | 10.2 K
4  | static_variable_selection          | VariableSelectionNetwork        | 313 K
5  | encoder_variable_selection         | VariableSelectionNetwork        | 3.1 M
6  | decoder_variable_selection         | VariableSelectionNetwork        | 3.0 M
7  | static_context_variable_selection  | GatedResidualNetwork            | 103 K
8  | static_context_initial_hidden_lstm | GatedResidualNetwork            | 103 K
9  | static_context_initial_cell_lstm   | GatedResidualNetwork            | 103 K
10 | static_context_enrichment          | GatedResidualNetwork            | 103 K
11 | lstm_encoder                       | LSTM                            | 206 K
12 | lstm_decoder                       | LSTM                            | 206 K
13 | post_lstm_gate_encoder             | GatedLinearUnit                 | 51.5 K
14 | post_lstm_add_norm_encoder         | AddNorm                         | 320
15 | static_enrichment                  | GatedResidualNetwork            | 128 K
16 | multihead_attn                     | InterpretableMultiHeadAttention | 64.4 K
17 | post_attn_gate_norm                | GateAddNorm                     | 51.8 K
18 | pos_wise_ff                        | GatedResidualNetwork            | 103 K
19 | pre_output_gate_norm               | GateAddNorm                     | 51.8 K
20 | output_layer                       | Linear                          | 1.1 K
----------------------------------------------------------------------------------------
7.8 M     Trainable params
0         Non-trainable params
7.8 M     Total params
31.071    Total estimated model params size (MB)
```

Temporal Fusion Transformer modeling – Results

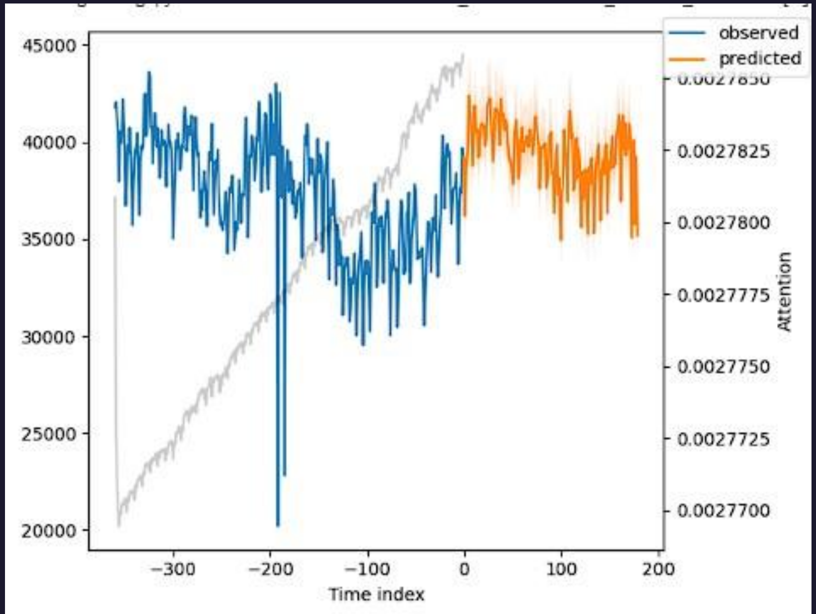
Cluster 1

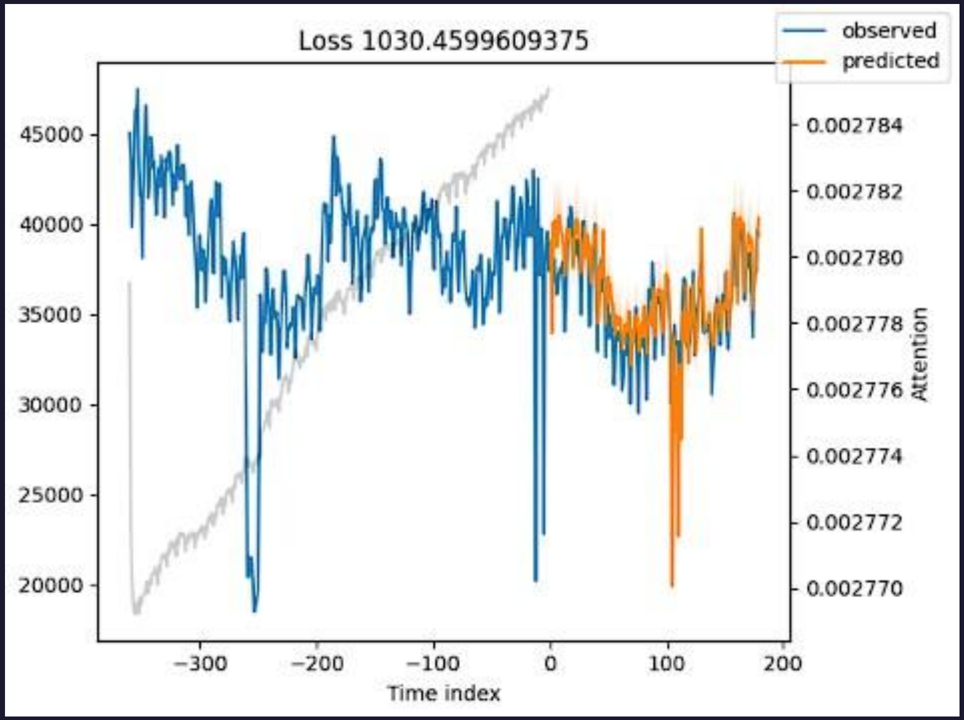
Baseline Model Outcome on validation

```
actuals1 = torch.cat([y for x, (y, weight) in iter(val_dataloader1)]).to('cuda')
baseline_predictions1 = Baseline().predict(val_dataloader1).to('cuda')
((actuals1 - baseline_predictions1)/(actuals1)).abs().mean().item()

WARNING: Missing logger folder: /content/lightning_logs
WARNING:lightning.pytorch.loggers.tensorboard:Missing logger folder: /content/lightning_logs
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
0.07389553636312485
```

test



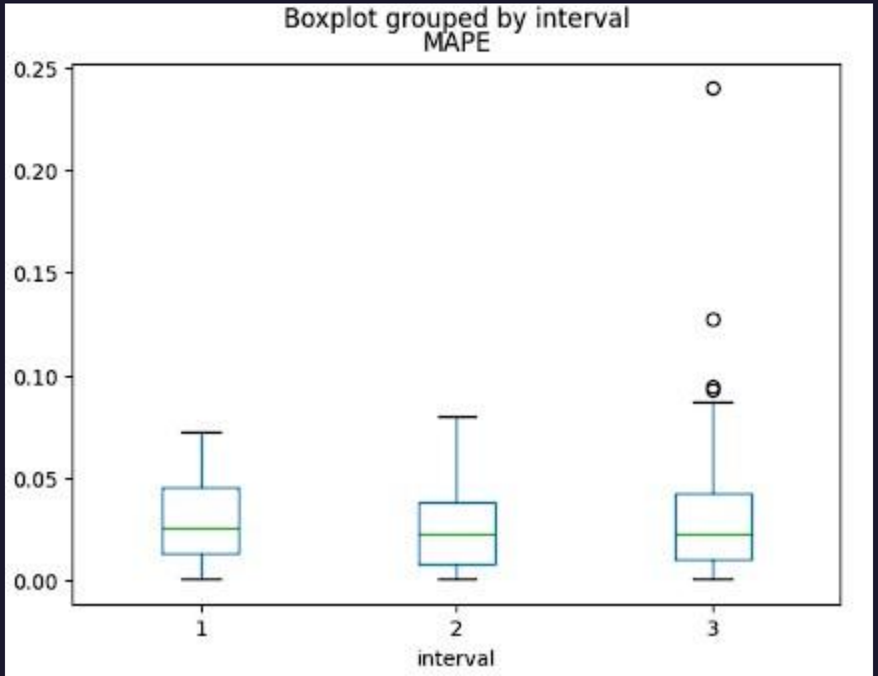


validation

Temporal Fusion Transformer modeling – Results

Cluster 1

Boxplot of MAPE on test set




```
mape1 = cluster1['MAPE'].mean()
mape1
```

```
0.029311845577712728
```

MAPE on test set

Temporal Fusion Transformer modeling – Results

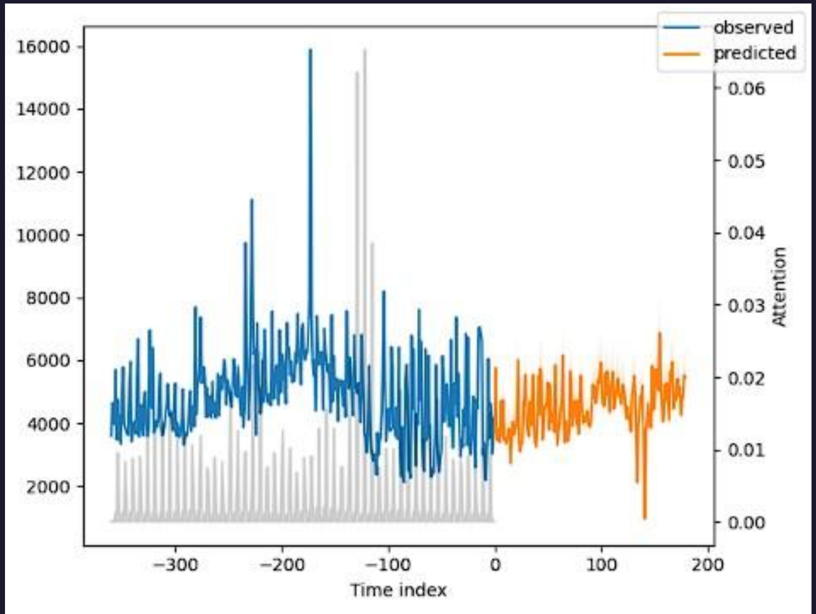
Cluster 2

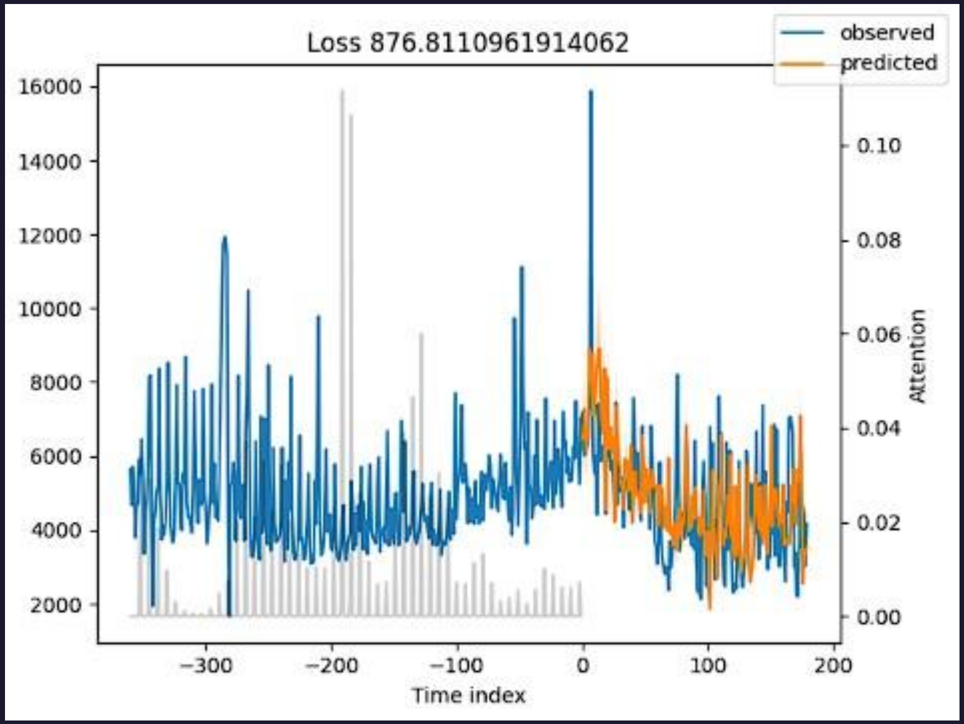
Baseline Model Outcome on validation

```
actuals2 = torch.cat([y for x, (y, weight) in iter(val_dataloader2)]).to('cuda')
baseline_predictions2 = Baseline().predict(val_dataloader2).to('cuda')
((actuals2 - baseline_predictions2)/(actuals2)).abs().mean().item()

WARNING: Missing logger folder: /content/lightning_logs
WARNING:lightning.pytorch.loggers.tensorboard:Missing logger folder: /content/lightning_logs
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
0.6881226301193237
```

test



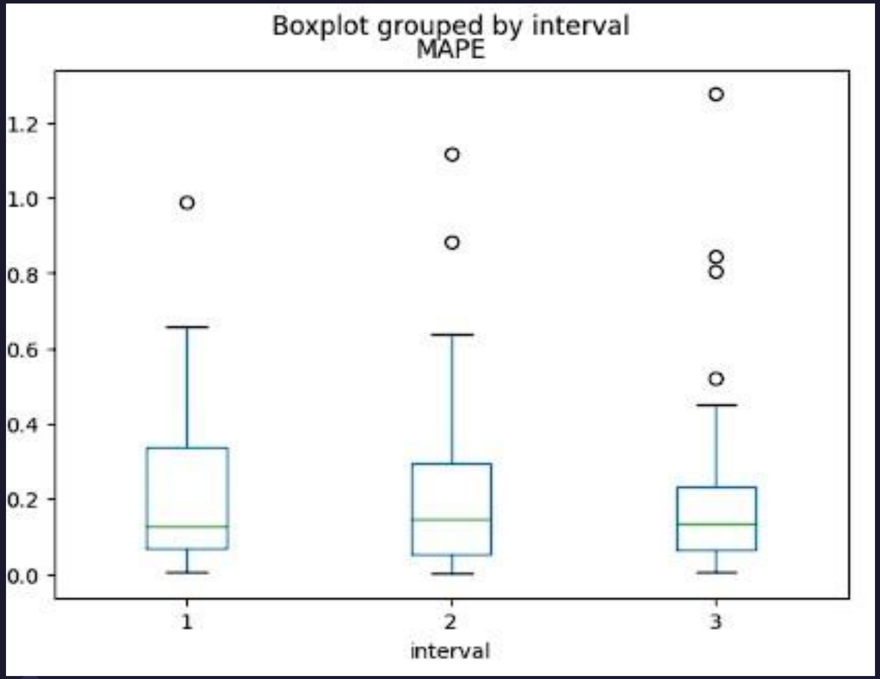


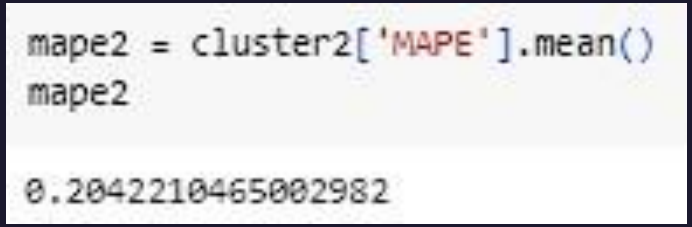
validation

Temporal Fusion Transformer modeling – Results

Cluster 2

Boxplot of MAPE on test set





```
mape2 = cluster2['MAPE'].mean()
mape2
```

```
0.2042210465002982
```

MAPE on test set

Temporal Fusion Transformer modeling – Results

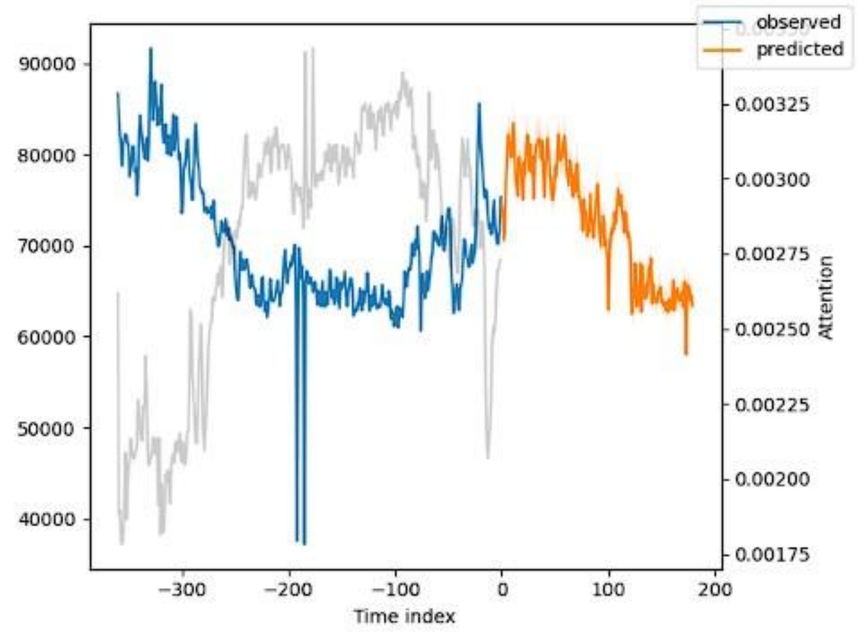
Cluster 3

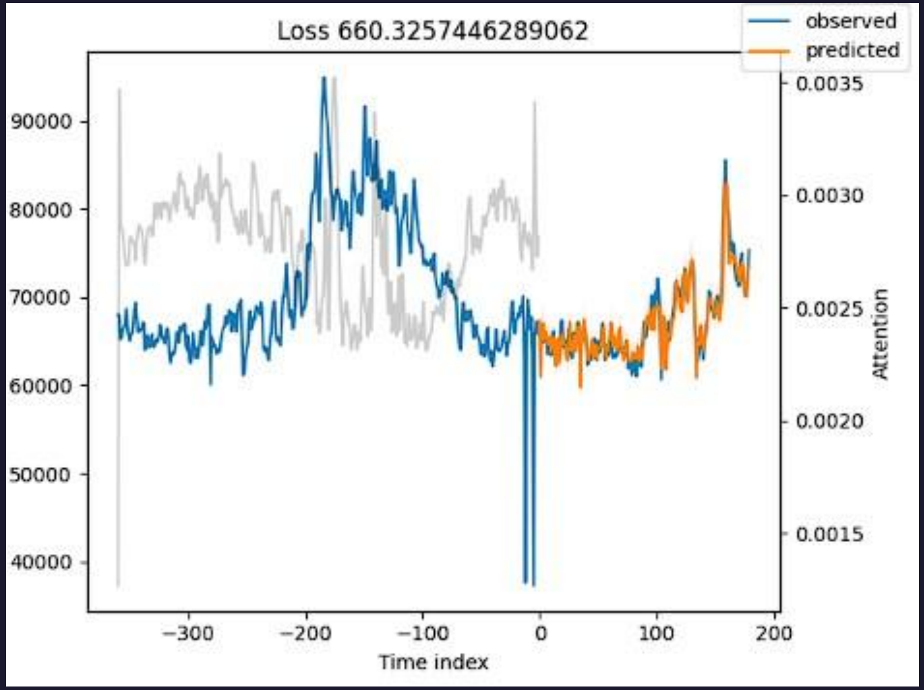
Baseline Model Outcome on validation

```
actuals3 = torch.cat([y for x, (y, weight) in iter(val_dataloader3)]).to('cuda')
baseline_predictions3 = Baseline().predict(val_dataloader3).to('cuda')
((actuals3 - baseline_predictions3)/(actuals3)).abs().mean().item()

INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
0.04628966748714447
```

test





validation

Temporal Fusion Transformer modeling – Results

Cluster 3

Boxplot of MAPE on test set

```
mape3 = cluster3['MAPE'].mean()
mape3
```

```
0.02463250434958997
```

MAPE on test set

Temporal Fusion Transformer modeling
– Results

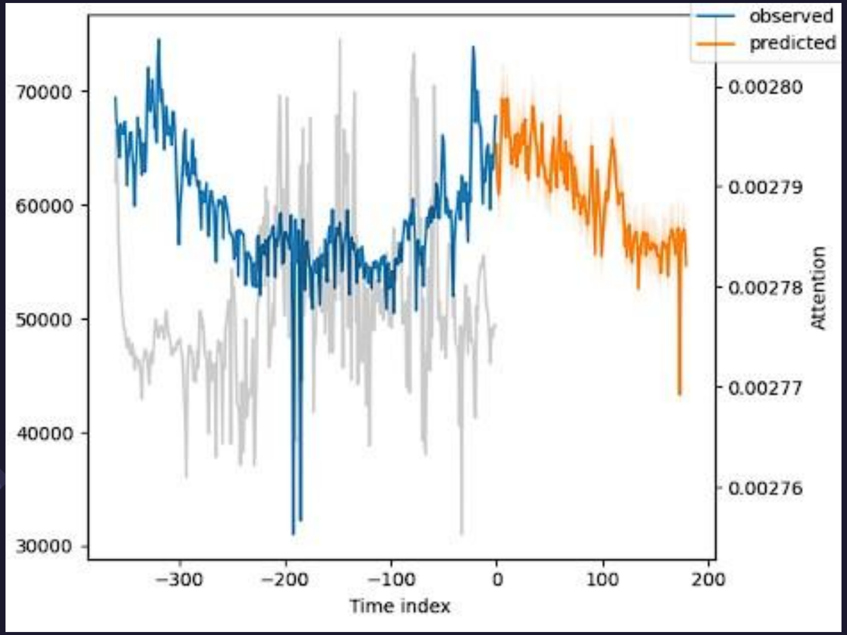
Cluster 4

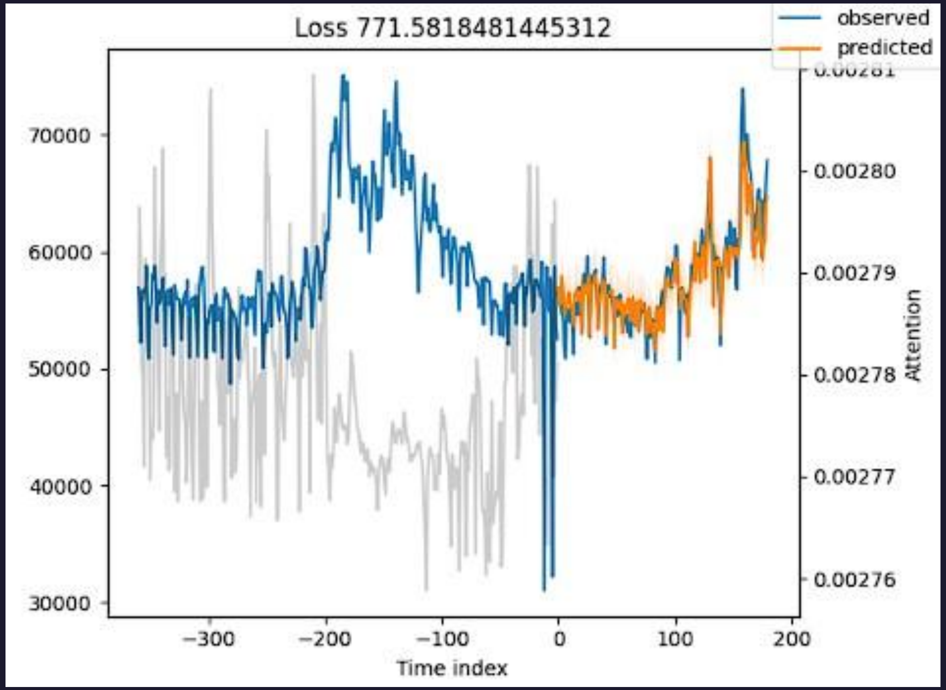
Baseline Model Outcome on validation

```
actuals4 = torch.cat([y for x, (y, weight) in iter(val_dataloader4)]).to('cuda')
baseline_predictions4 = Baseline().predict(val_dataloader4).to('cuda')
((actuals4 - baseline_predictions4)/(actuals4)).abs().mean().item()

INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
0.0896955281496048
```

test



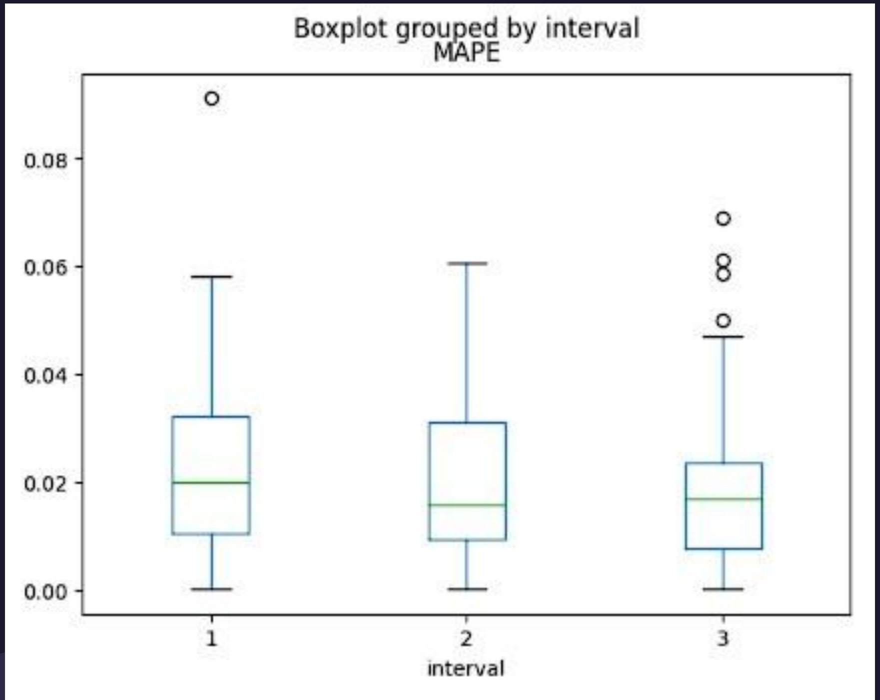


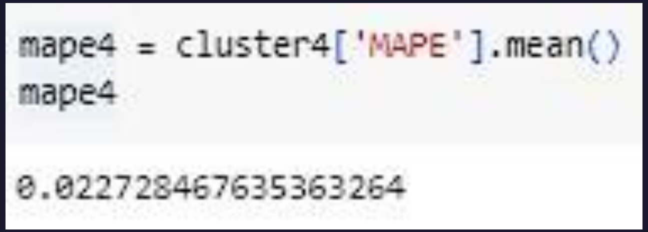
validation

Temporal Fusion Transformer modeling – Results

Cluster 4

Boxplot of MAPE on test set





MAPE on test set

Temporal Fusion Transformer modeling – Results

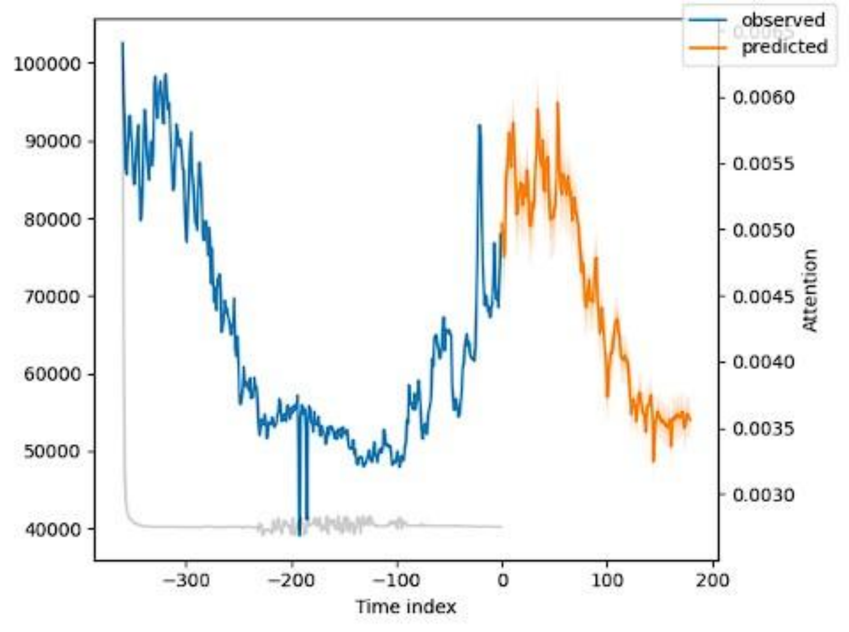
Cluster 5

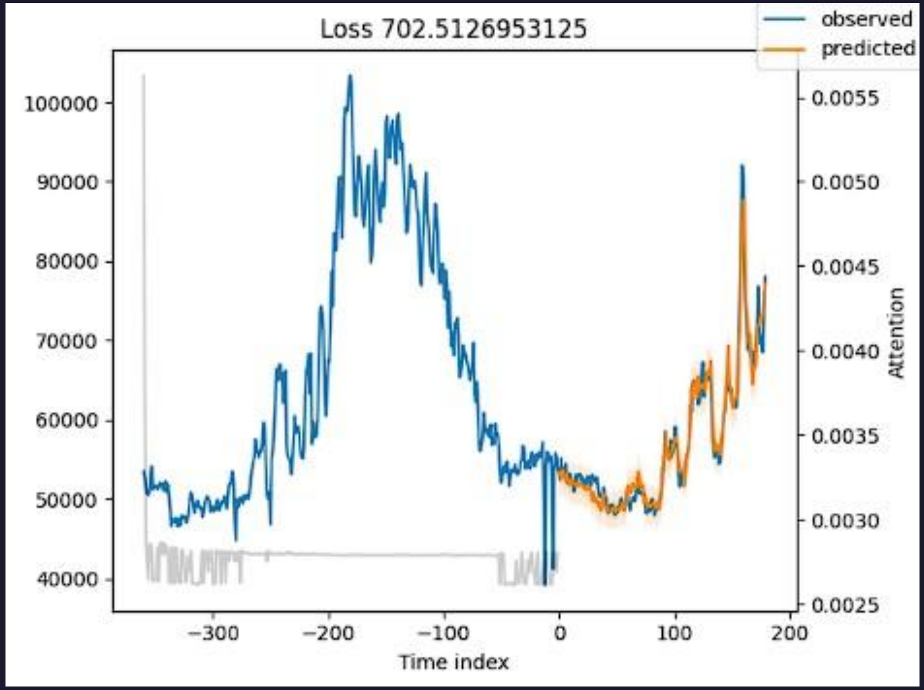
Baseline Model Outcome on validation

```
actuals5 = torch.cat([y for x, (y, weight) in iter(val_dataloader5)]).to('cuda')
baseline_predictions5 = Baseline().predict(val_dataloader5).to('cuda')
((actuals5 - baseline_predictions5)/(actuals5)).abs().mean().item()

INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
0.10071036964654922
```

test



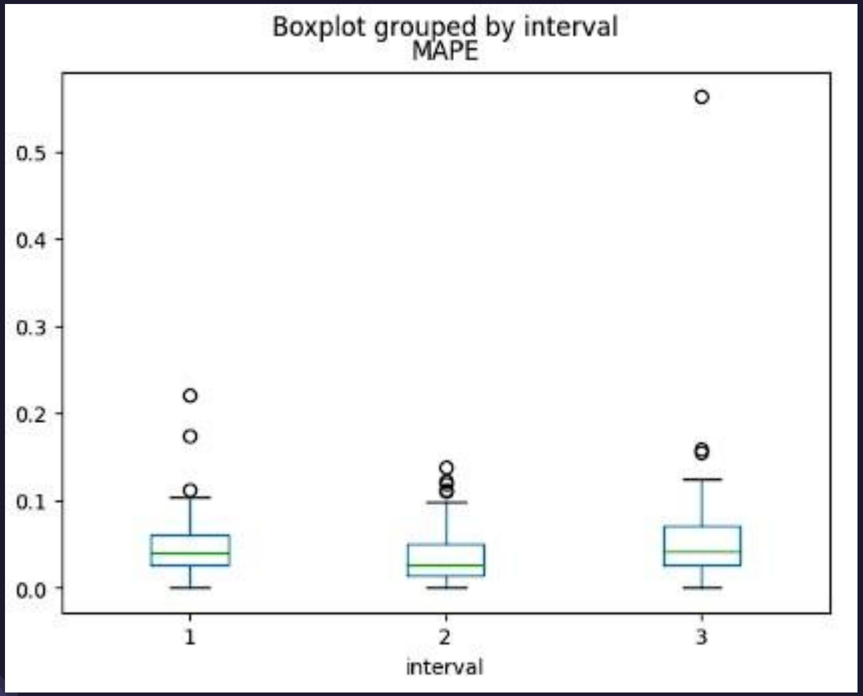


validation

Temporal Fusion Transformer modeling – Results

Cluster 5

Boxplot of MAPE on test set




```
mape5 = cluster5['MAPE'].mean()
mape5

0.042938305456993425
```

MAPE on test set

Temporal Fusion Transformer modeling – Results

- For other clusters' results, please check out outputs on Jupyter Notebook and technical documentation
- Overall MAPE are 0.07007176169746014, calculated by taking weighted average over MAPE of clusters' data

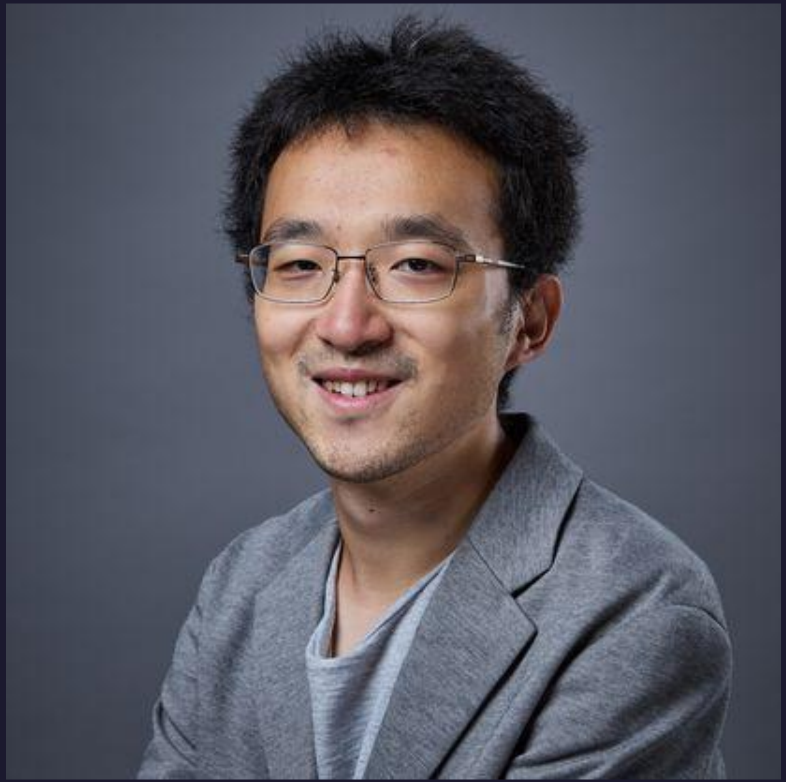



Summary

Summary

- According to our modeling results, Temporal Fusion Transformer (TFT) performs very well on forecasting electricity consumption with the assistance of hierarchical clustering. Its accuracy of forecasting is around 7.0% while DeepAR's MAPE is around 8.9%, which is also a reasonably good score.
- Main advantage of TFT are the interpretability of features' importance and the ability to foretell out-of-sample data even without features' out-of-sample data.
- DeepAR has the edge on TFT in terms of easy implementation and efficient learning and predicting for all clusters at once.
- Further commitment on choosing optimal parameters like epochs, batch_size and learning rate is worthy looking into.

One man gang



Yunpeng Wang