IEOR4574 Project 2 – Forecasting Electricity Consumption Data Set Process Documentation

Yunpeng Wang
yw3954

Contents

- **Introduction**
- **Data Extraction**
- **Data Overview**
- **Data Preprocessing**
- **Target Variables**
- **Predictive Variables**
- **Pre-modeling**
- **Modeling**
- **Algorithmic Solution Results**

Introduction

- This project is essential to IEOR4574: Forecasting: A Real-World Application. In this project, we leverage time series analysis and models we learned during classes and implement them to extract insights from real-world data. More importantly, it is a learning experience
- The data we are interested in is an electricity consumption data set, which consists of 370 Portuguese clients' usage records from 2011-01-01 to 2015-01-01
- Apply agglomerative clustering algorithm to group users into clusters based on users' electricity consumption correlations
- Build two models: DeepAR and Temporal Fusion Transformer model to predict the future consumption, and more importantly, to compare the model performance to choose the better one
- Standardize a robust methodology (including data preprocessing, time series modeling, and analytical approach) for the development of forecasting that will be leveraged in the coming months

The purpose of this document is to provide a detailed technical overview of:

1. Design specifications and parameters
2. Data inclusions and exclusions
3. Predictive variable creation process
4. Reproducible model-building process

Data Extraction

The section explains the source and storage of data.

The electricity consumption data is manually pulled from UCI Machine Learning Repository. The primary source is Artur Trindade who is a researcher with Elergone Energia Lda, Leça da Palmeira, Portugal.

The format of the data is comma-separated values.

The weather data is retrieved from National Centers for Environmental Information’s (NCEI) API using Python requests. Data are records of Lisbon's(Portuguese Capital) daily maximum/ minimum/ average temperature and Lisbon’s daily precipitation. The primary source is Data Access.

The format of the stored data is comma-separated values.

To process data efficiently, data is uploaded and stored in Google Drive. Data are adjusted and integrated using Python scripts

Data Overview

The purpose of this section is to provide an overview of the electricity consumption data.

This electricity consumption data set contains all the power usage of 370 Portuguese clients (number of columns) between 01/01/2011 and 01/01/2015. The data set has no missing values. Values are in kW of each 15 min. Each column represents one client. Some clients were created after 2011. In these cases, consumption was considered zero.

The total record count (number of rows) is 140256.

Demo:

	MT_001	MT_002	MT_003	...	MT_368	MT_369	MT_370
2011-01-01 00:15:00	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
...	...	...	...	...	...	...	...
2015-01-01 00:00:00	2.538071	19.914651	1.737619	...	131.886477	673.020528	7135.135135

Data Preprocessing

This section has three parts: Data Diagnostic, Data Aggregation, and Data Clustering.

Data Diagnostics

This section has three parts: Data Diagnostic, Data Aggregation, and Data Clustering.

The first thing under scrutiny is the electricity consumption data description posted on the UCI Machine Learning Repository website. All other information is aligned with data except the statement that every year on March time change day (which has only 23 hours) the values between 1:00 am and 2:00 am are zero for all points, and Every year on October time change day (which has 25 hours) the values between 1:00 am and 2:00 am aggregate the consumption of two hours.

To prove that every year on March time change day the values between 1:00 am and 2:00 am are not zero for all points, the following codes are executed and outcomes are obtained as follows:

```
#March 13, 2011
(data.loc['2011-03-13'].sum(axis = 1)==0.0).any()
```

```
False
```

```
date = []
for index, row in data.iterrows():
    if row.sum() == 0.0:
        date.append(index)
date
```

```
[]
```

Afterward, comparisons between values in different time slots on Nov. 6th, 2011, between values in the same periods on different days, and between values in same

time span on the same date of different months are plotted out to visually check the validity of the last part of the statement. Since there is no huge boost of the values between 1:00 am and 2:00 on October time change days, it is safe to conclude that the second part is false.

Data Aggregation

The data is amassed into daily data by summing all the power usage of each client on the same date.

	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

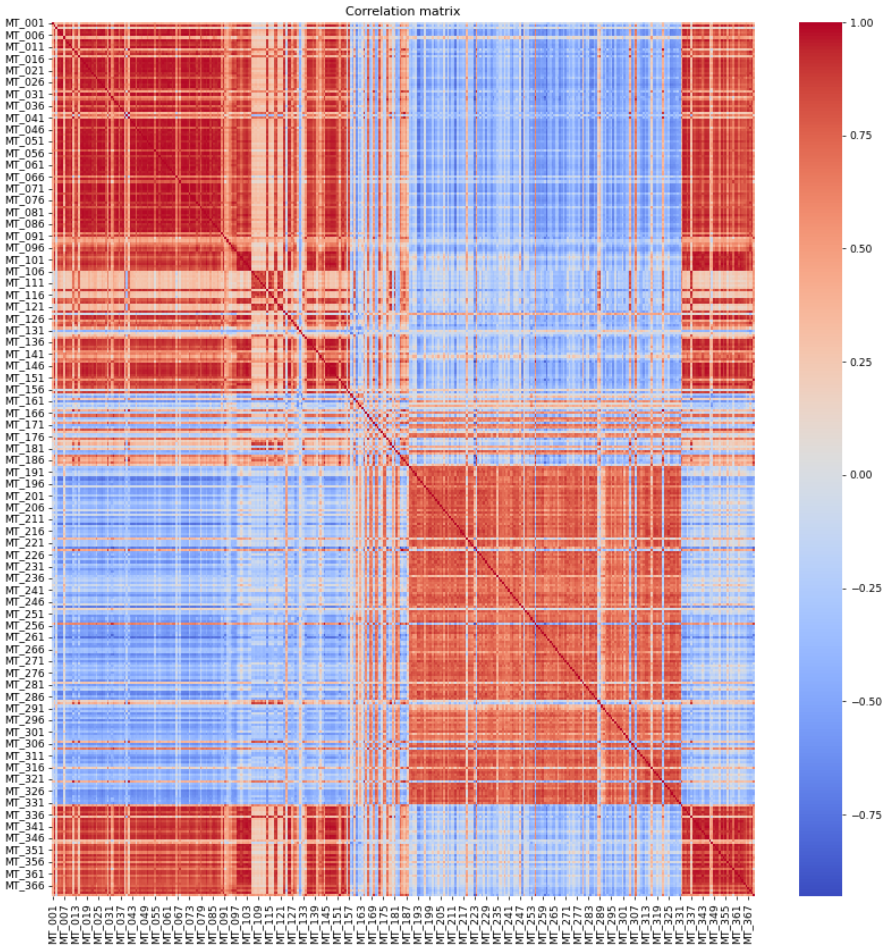
1462 rows × 370 columns

Since clients will be clustered based on their correlation, also known as resemblance, and the patterns of customers' daily electricity consumption are our primary focus of study, data will be further aggregated by clusters and be trimmed into daily average electricity consumption.

Data Clustering

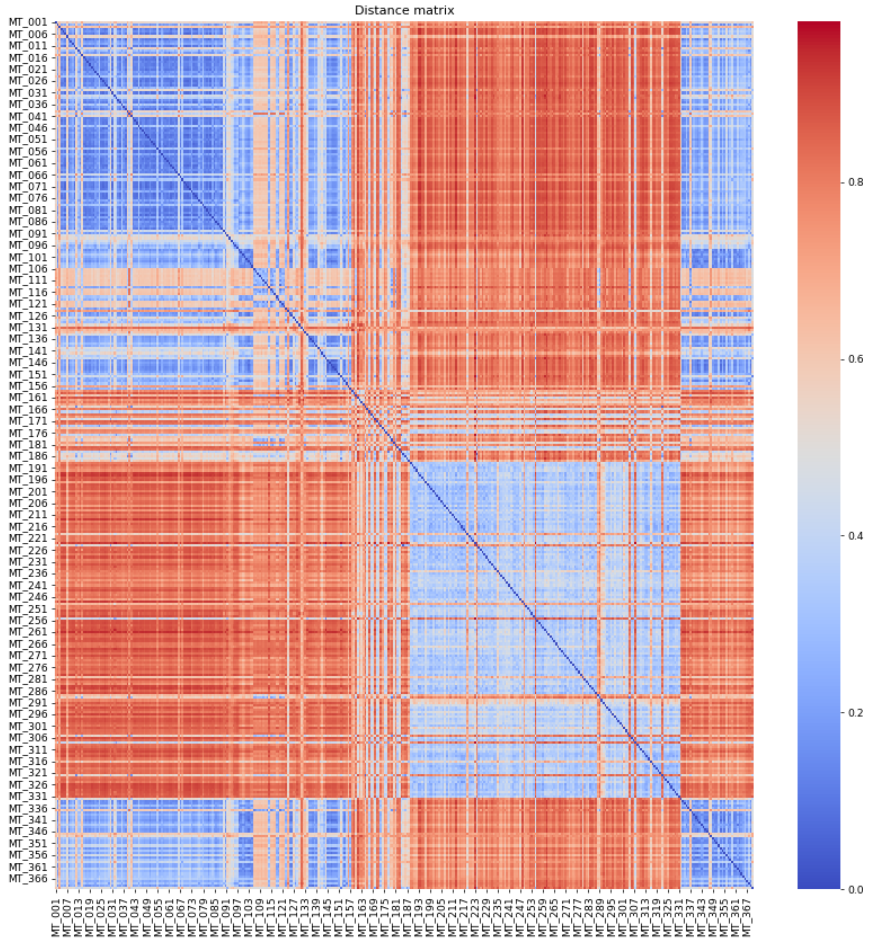
Data clustering used in this case is agglomerative hierarchical clustering, which is a bottom-up algorithm proceeded by grouping elementary objects into larger and larger clusters. Hierarchical clustering algorithms are based on the notion of distance. The definition of distance between columns is correlation-based distance.

Firstly, the correlation matrix (between columns) is calculated by calling data.corr(method = ‘pearson’). And correlation-based distance matrix is derived from the correlation matrix.



The figure above is the heatmap of the correlation matrix.

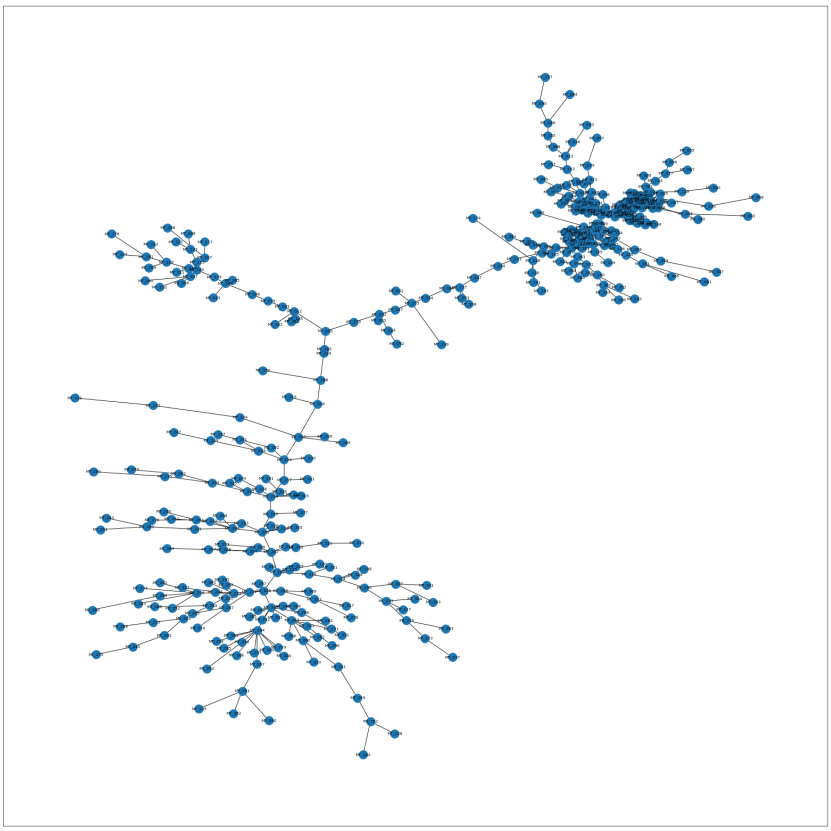
The figure below is the heatmap of the distance matrix.

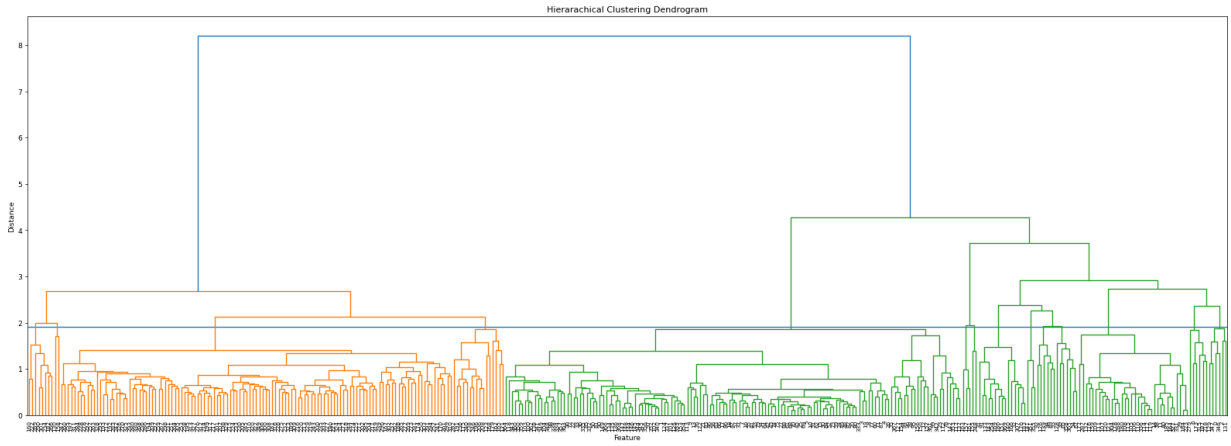


Because a distance matrix can be thought of as a weighted adjacency matrix of some graphs and it follows it can be represented as complete graphs whose nodes are features(clients) and edges length is proportional to the distance between two nodes, a minimum spanning tree is constructed and visualize distance matrix.

Then, the linkage matrix is derived from the distance matrix and a dendrogram is created based on the linkage matrix followed by choosing optimal number clusters. The logic behind choosing optimal number clusters is the proper cut on the

dendrogram should not be over-clustering clients because the fewer number of clusters, the less model in need and the better control over models, and the proper cut should neither be under-clustering because we would like to have clients within the same cluster have as similar power usage pattern as possible. Therefore, the max distance that discriminates different clusters is set to be 1.9.





Finally, the f-cluster forms flat clusters from the hierarchical clustering defined by the given linkage matrix and the given max distance limitation. The desired cluster is acquired and shown as follows:

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

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

Cluster	Feature
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

The evaluation of hierarchical clustering is delivered by observing the similarity of each cluster's client power usage plot.

There are four different data sets being bucketed:

1. original data
2. original data with standardization and outliers elimination
3. aggregated daily data
4. aggregated daily data with standardization and outliers eliminations

Each data set is through the same steps in hierarchical clustering and after cross-evaluations, we stick with the aggregated data, and previously designed

clusters (as the screenshot above shows) are good at capturing similarity so we decide to go with this one.

Target Variables

A target variable is the variable to predict as a result in algorithmic solutions.

The **average daily power usage of clients by clusters** in kW is the **target variable**.

Predictive Variables

A predictive variable is a variable used in algorithmic solutions to predict the target variable.

Variable List

1. Weather: Maximum/ minimum/ average temperature and participation of days in Lisbon

2. Seasonalities:

    a. Long period of time seasonality: Year, Month, Week, DayofYear…; sin(1,freq=A-DEC), cos(1,freq=A-DEC)...; Determined according to data trends and seasonality analysis

    b. Short period of time seasonality: Quarter, DayofMonth ...; s(2,7), s(3, 7); .... Determined according to data trends and seasonality analysis

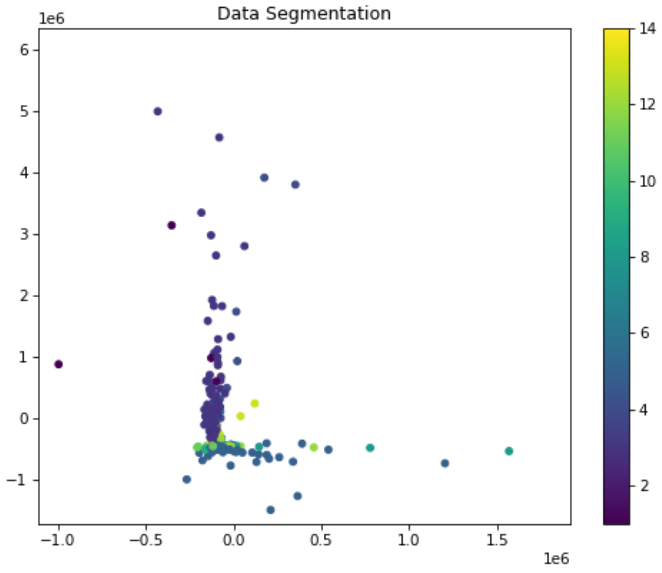
3. Trends: first, second, and third order trends over time. Determined according to data trends and seasonality analysis

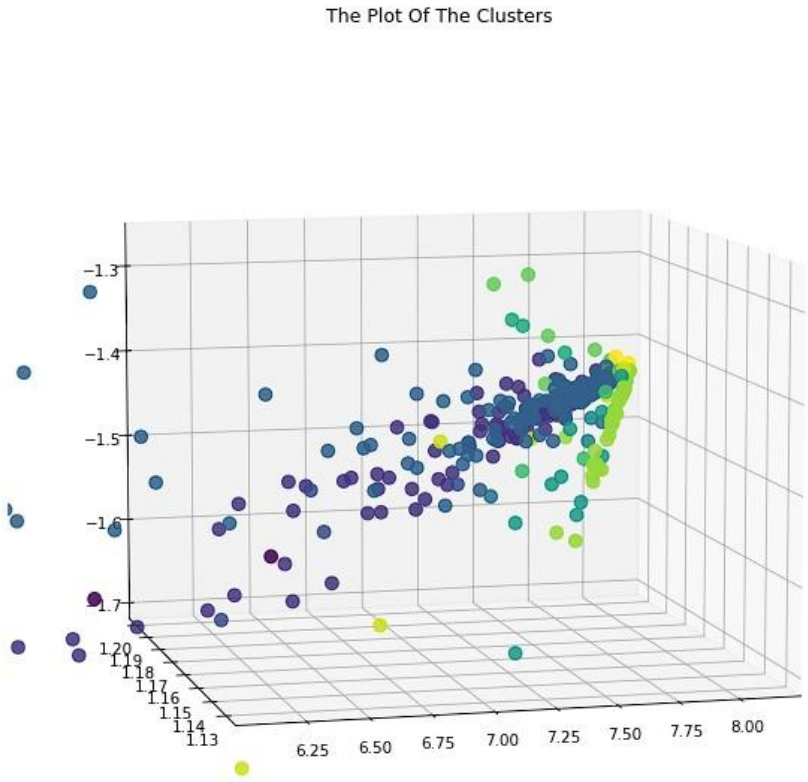
4. time series as features: lags (serial dependence). Determined according to data serial dependence analysis

5. const: constant dummies

Pre-modeling

Firstly, clients are labeled by clusters and data are further aggregated into average daily power usage by clusters. Then, we visualize our cluster in 2-D and 3-D by using PCA.



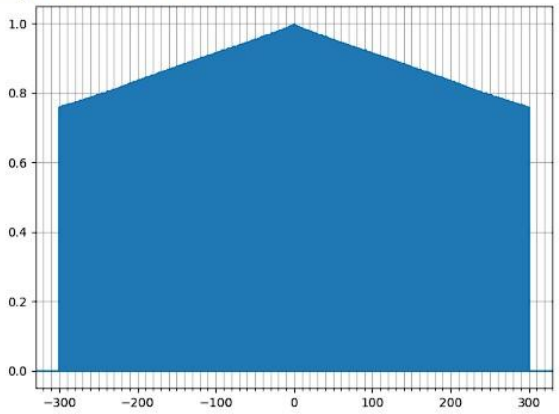


In the above 2-D and 3-D visualizations, we can clearly see that clients within the same cluster form straight lines, and most data points densely gather at some area, which makes some data points belonging to different clusters look closer than their own clusters' data points.

Secondly, Autocorrelation and Power Spectral Density are put into graphs for each cluster. They are quite handy because seasonalities or repeated behaviors in the signal at different time lengths are evident on these graphs.

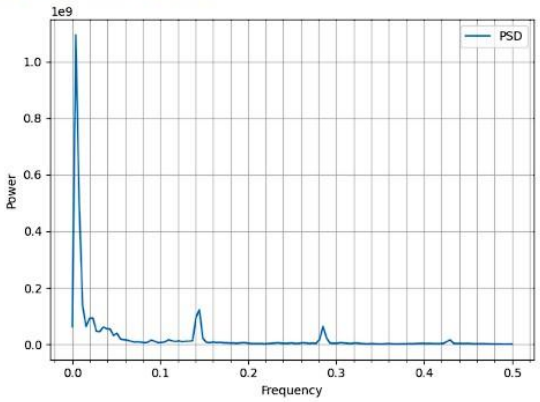
```
plt.figure(figsize=(20, 7))
fig, ax = plt.subplots()
ax.acorr(c1.values, maxlags = 300)
minor_locator = AutoMinorLocator(10)
ax.xaxis.set_minor_locator(minor_locator)
plt.grid(which='minor')
# plt.legend(['$R_X$(\u03C4)'],loc='upper left')
# plt.xlabel('\u03C4')
# plt.ylabel('Power')
ax.grid(True, which='both')
plt.tight_layout()
fig.show()
```

```
<Figure size 2000x700 with 0 Axes>
```



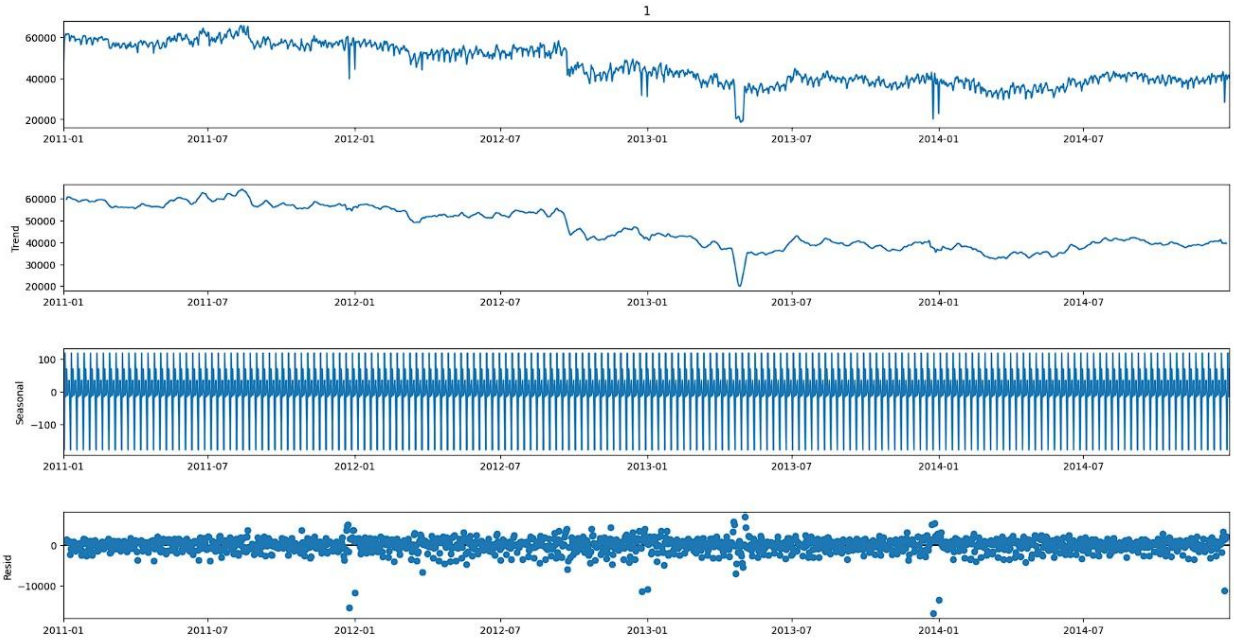

```
freqs, psd = signal.welch(c1.values)
plt.figure(figsize=(20, 7))
fig, ax = plt.subplots()
ax.plot(freqs, psd)
minor_locator = AutoMinorLocator(5)
ax.xaxis.set_minor_locator(minor_locator)
plt.grid(which='minor')
plt.legend(['PSD'])
plt.xlabel('Frequency')
plt.ylabel('Power')
ax.grid(True, which='both')
plt.tight_layout()
fig.show()
```

```
<Figure size 2000x700 with 0 Axes>
```



The graphs above indicate there exist multiple seasonalities in the first cluster time series data. On the left, as lag increases, the auto-correlation gradually decreases. On the right, it reveals a peak near the frequency of 0.002, indicating a seasonality in the sales of a year. Moreover, the second peak is located near the frequency of 0.14, and the third peak is near 0.28. Therefore, the data might have periodicity in the time scale of 8 and 3 days.

Thirdly, a function called seasonal_decompose is leveraged to further confirm seasonalities extracted from the previous step and check trends and stationarity after teasing out seasonality and trends.

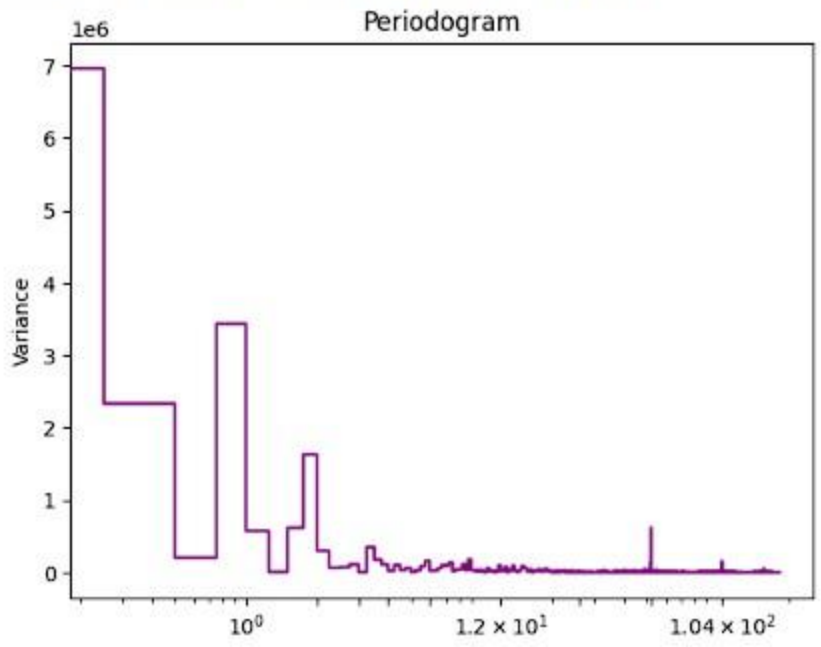


As a result, by implementing **seasonal_decompose** and setting the parameter of period equal to 8, the above group of graphs is manufactured. Obviously, there are lots of ups and downs in the trend subfigure and residuals of data after teasing seasonality and trend are approximately normally distributed with a mean equal to 0. Thence, seasonal trends, and stationary residuals are deduced.

Next, we take advantage of the periodogram and create Fourier features in accordance with the strength of the frequencies in the time series revealed by the periodogram.

```python
def plot_periodogram(ts, detrend='linear', ax=None):
    from scipy.signal import periodogram
    fs = pd.Timedelta("1Y") / pd.Timedelta("1D")
    freqencies, spectrum = periodogram(
        ts,
        fs=fs,
        detrend=detrend,
        window="boxcar",
        scaling='spectrum',
    )
    if ax is None:
        _, ax = plt.subplots()
    ax.step(freqencies, spectrum, color="purple")
    ax.set_xscale("log")
    ax.set_xticks([1, 2, 3, 4, 6, 12, 26, 52, 104])
    ax.ticklabel_format(axis="y", style="sci", scilimits=(0, 0))
    ax.set_ylabel("Variance")
    ax.set_title("Periodogram")
    return ax
plot_periodogram(c1.values)
```

```
<Axes: title={'center': 'Periodogram'}, ylabel='Variance'>
```



From left to right, the periodogram drops off after 3 on x-axis. That was why we chose three Fourier pairs to model the annual season.

Then, all trends and seasonalities deduced above are added to each cluster's dataframe via running a function DeterministicProcess and DatetimeIndex functions from Pandas.

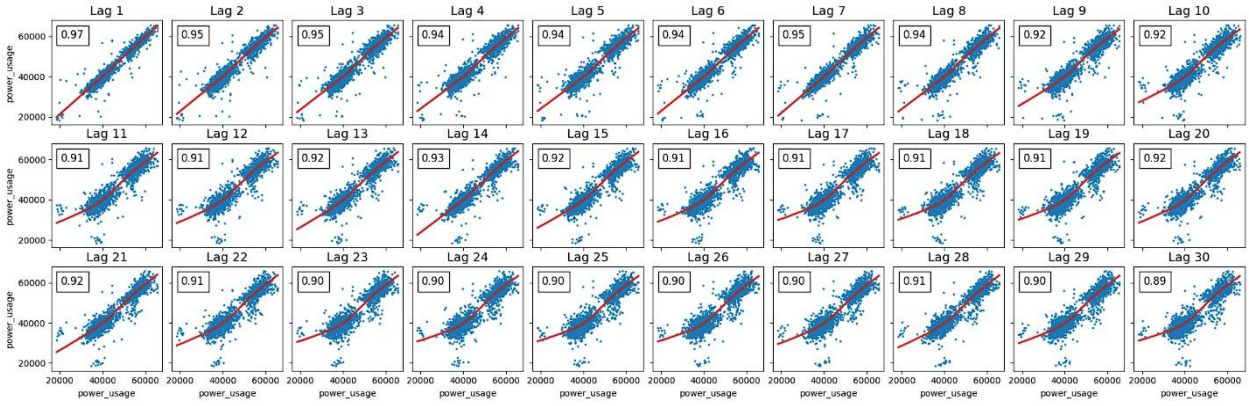
```
from statsmodels.tsa.deterministic import CalendarFourier, DeterministicProcess

fourier = CalendarFourier(freq="A", order=3)
dp = DeterministicProcess(
index= c1.index,
period = 8,
constant=True,               # dummy feature for bias (y-intercept)
order=1,
seasonal=True,
additional_terms=[fourier],  # annual seasonality (fourier)
drop=True,                   # drop terms to avoid collinearity
)
c1 = pd.concat([c1,dp.in_sample()], axis = 1)
```

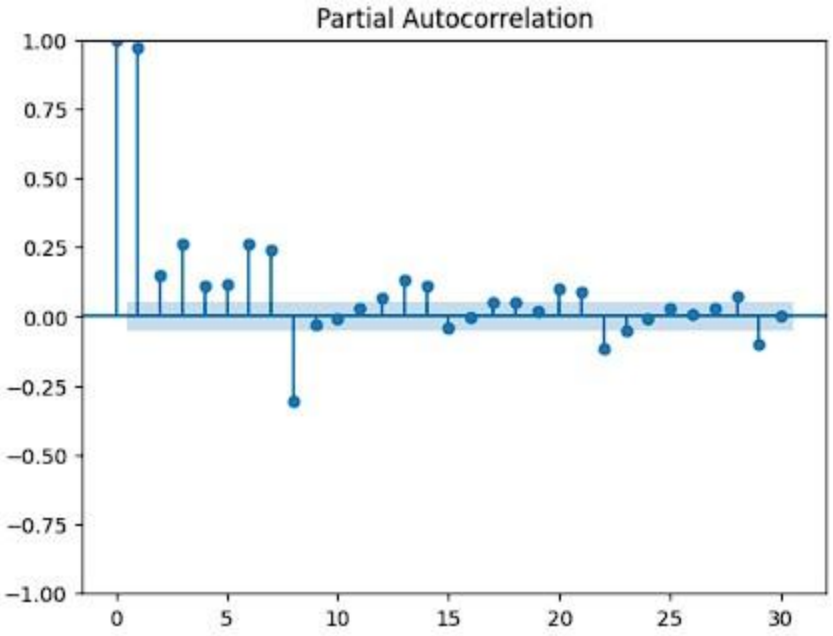
```
c1.rename(columns={1: 'power_usage'}, inplace = True)
```

```
c1.columns
```

```
Index(['power_usage', 'const', 'trend', 's(2,8)', 's(3,8)', 's(4,8)', 's(5,8)',
       's(6,8)', 's(7,8)', 's(8,8)', 'sin(1,freq=A-DEC)', 'cos(1,freq=A-DEC)',
       'sin(2,freq=A-DEC)', 'cos(2,freq=A-DEC)', 'sin(3,freq=A-DEC)',
       'cos(3,freq=A-DEC)'],
      dtype='object')
```

Afterwards, lag and partial autocorrelation plots make serial dependence in the time series become apparent and help us choose which lag features to use. With little effort in analyzing these plots, we add favourable lags features to dataframes.





The lag plots indicate that the relationship of daily electricity consumption to its lags is mostly linear, while the partial autocorrelations suggest the dependence can be captured using lags 1, 2, 3, 4, 5, 6, 7, and 8. We lag a time series in Pandas with the shift method.

Last but not least, weather features are appended to data frames manually.

For more information about other clusters' analysis, please check out outputs on Jupyter notebook 'TFT.ipynb'.

Modeling

Introduction

DeepAR and Temporal Fusion Transformer are built to study daily electricity consumption patterns and predict future daily data. Implementation of these two models gives us great insights into deep learning in forecasting time series.

Prior to constructing and training models, available data are split into train, validation, and test datasets. Then, DeepAR is deployed on Amazon Web Service and TFT is deployed on Google Colab. Both models are trained on 14 clusters' datasets and tested after training. MAPE is the metric used to evaluate the performance of models.

Dataset feeded to DeepAR is labeled average daily data with weather data appended. Yet, dataset gone to TFT are preprocessed in the way described in **Pre-modeling** section. Each cluster has its own unique trend, seasonality and lags features. Thus, there are 14 TFT models created corresponding to 14 modified clusters data.

The figures used below are showcases of one or two models, for comprehensive demonstration graphs, please check out outputs on corresponding Jupyter notebooks 'TFT.ipynb' and 'DeepAR.ipynb'.

DeepAR

The Amazon SageMaker DeepAR forecasting algorithm is a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN), more precisely, autoregressive recurrent network. DeepAR outperforms the standard ARIMA and exponential smoothing methods when the dataset contains hundreds of related time series.

After trying to deploy DeepAR on Google Colab and encountering with lots of Python libraries' incompatibility issues and lack of computing resources, I turn to Amazon SageMaker, which a fully managed machine learning service on Amazon Web Service(AWS). Following tutorials, we finish initial setting up work, including creating a Sagemaker domain, uploading datasets into S3 (scalable storage in the cloud) and opening up a notebook in Sagemakes studio. By the way, the data given to the model is the clusters' daily power usage data without feature engineering.

Subsequently, weather data is merged into the original data first. processed data is split into three-year train data, and one-year test data for evaluating the trained model. Validation data is omitted in train data and cross-validation is skipped because of time constraints.

```
freq = "D"

# we predict for 7*4*3 days
prediction_length = 7 * 4 * 3

# we also use 7*4*3 days as context length, this is the number of state updates accomplished before making predictions
context_length = 7 * 4 * 3

start_dataset = pd.Timestamp("2011-01-01", freq=freq)
end_training = pd.Timestamp("2014-01-01", freq=freq)
```

We specify here the portion of the data that is used for training: the model sees data from 2011-01-01 to 2014-01-01 for training.

```
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
```

As test data, we will consider time series extending beyond the training range: these will be used for computing test scores, by using the trained model to forecast their trailing 84 days(approximately three months) and comparing predictions with actual values. To evaluate our model performance over three months, we generate test data extending to 1, 2, 3, and 4 quarters beyond the training range. This way we perform the rolling evaluation of our model.

Now that we have the data files locally, we copy them to S3 where DeepAR can access them.

Then, the estimator is defined as follows and launches the training job.

```
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,
)
```

To expedite process, default parameters are leveraged for every optional parameter in this case. Moreover, additional time features, in our case daily temperature and precipitation in Lisbon are assigned to the model via the dynamic_feat field.

Given that the trained model is obtained, we deploy it to an endpoint so that predictions can be performed.

```
predictor_new_features = estimator_new_features.deploy(
    initial_instance_count=1, instance_type="ml.m5.large", predictor_cls=DeepARPredictor
)
```

```
INFO:sagemaker:Creating model with name: deepar-electricity-demo-new-features-2023-06-08-22-42-22-899
INFO:sagemaker:Creating endpoint-config with name deepar-electricity-demo-new-features-2023-06-08-22-42-22-899
INFO:sagemaker:Creating endpoint with name deepar-electricity-demo-new-features-2023-06-08-22-42-22-899
------!
```

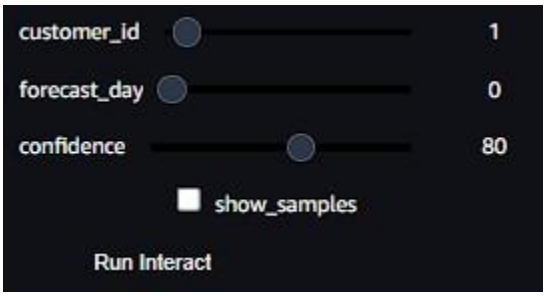
```
predictor_new_features.predict(
    ts=timeseries1[1][:-100],
    dynamic_feat=[w['TAVG'].tolist()],
    quantiles=[0.1, 0.5, 0.9],
).head()
```

	0.1	0.5	0.9
2014-09-23	3742.401611	4154.849609	4612.723145
2014-09-24	4042.995117	4518.472656	4945.676758
2014-09-25	3745.682129	4210.323242	4534.091309
2014-09-26	3187.028320	3673.031250	4158.256836
2014-09-27	3371.047852	3823.166260	4208.417969

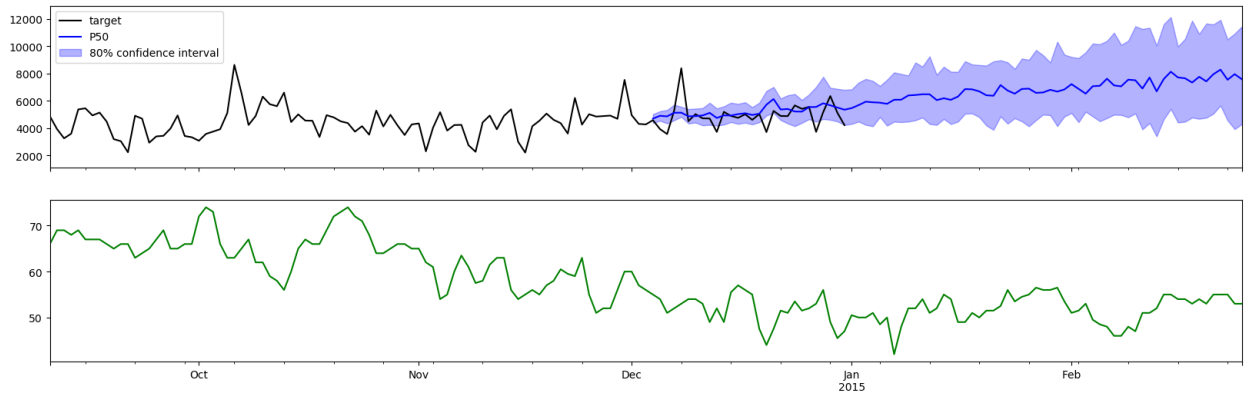
The figure above shows predictor object generates predictions.

To better visualize the predictions out of our model, the function as follows is defined.

```
@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,
    )
```

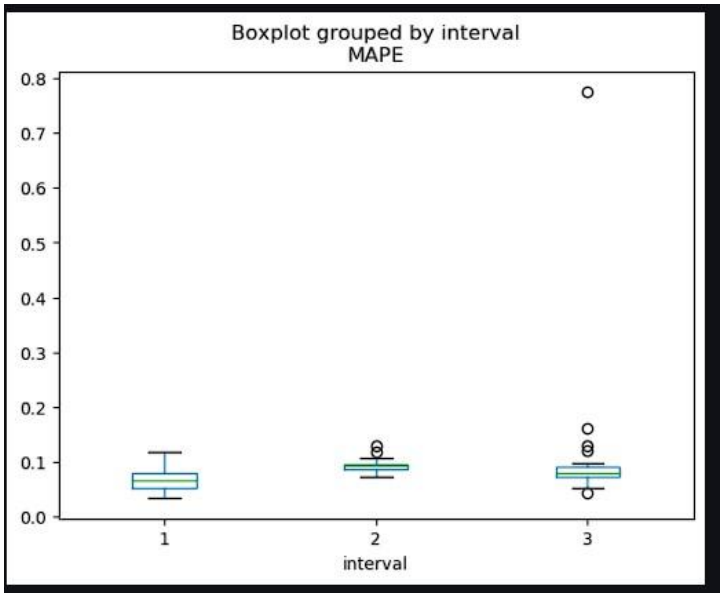


We can interact with the function, to look at the forecast of any cluster at any point in (future) time.



As for the evaluation metric, we choose Absolute Percentage Error (MAPE) because it works for analyzing large sets of data and requires the use of dataset values other than zero.

The testing results must be divided into three periods where they have a time span of 28 days. For each region report MAPE, a box plot of errors is used to show the spread.




```
FMAPE.mean()

0.08931150398980939
```

Temporal Fusion Transformer (TFT)

Temporal Fusion Transformer (TFT) is a transformer-based model that leverages self-attention to capture the complex temporal dynamics of multiple time sequences. Among notable Deep Learning time-series models, TFT stands out because it supports various types of features.

We would like to design a unique model for each cluster data set. We start off by creating DataLoaders and separating data into train, validation, and test data. The reasons behind passing our data TimeSeriesDataSet format which is immensely useful are:

- It spares us from writing our own Dataloader.
- We can specify how TFT will handle the dataset's features.

Our model uses a lookback window of 360 days to predict the power usage of the next 180 days.

Time-varying features are predictive features.

Most cluster data is split into three-year train data, half-year validation data, and half-year test data. However, for those clusters who have 0 electricity consumption in the previous 365 days, we ditch the first-year data and the model sees the data from 2012-01-01 to 2014-01-01 as train data.

```
c12c = []
for i in c12.columns[1:-2]:
    c12c.append(i)
c12 = c12.fillna(0)
training12 = TimeSeriesDataSet(
    c12['2012-01-01':'2014-01-01'],
    time_idx="days_from_start",
    target="power_usage",
    group_ids=["cluster"],
    min_encoder_length=max_encoder_length // 2,
    max_encoder_length=max_encoder_length,
    min_prediction_length=1,
    max_prediction_length=max_prediction_length,
    static_categoricals=["cluster"],
    time_varying_known_reals=c12c,
    time_varying_unknown_reals=['power_usage'],
    target_normalizer= None,
    add_relative_time_idx=True,
    add_target_scales=True,
    add_encoder_length=True,
)
```

```
validation12 = TimeSeriesDataSet.from_dataset(training12, c12[:-180], predict=True, stop_randomization=True)
test12 = TimeSeriesDataSet.from_dataset(validation12, c12, predict=True, stop_randomization=True)
train_dataloader12 = training12.to_dataloader(train=True, batch_size=batch_size, num_workers=0)
val_dataloader12 = validation12.to_dataloader(train=False, batch_size=batch_size * 3, num_workers=0)
test_dataloader12 = test12.to_dataloader(train=False, batch_size=batch_size * 3, num_workers=0)
```

Next, we proceed with baseline models. Although it is named a naive predictor, baseline models can be very surprising and outperform other fancier models.

```
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
```

TFT models are trained using the Trainer interface from PyTorch Lightning.

```
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)
```

```
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)
```

For cluster1 data, after 54 epochs, Early stopping kicks in and halts the training process. A model with as less validation loss as possible is created.

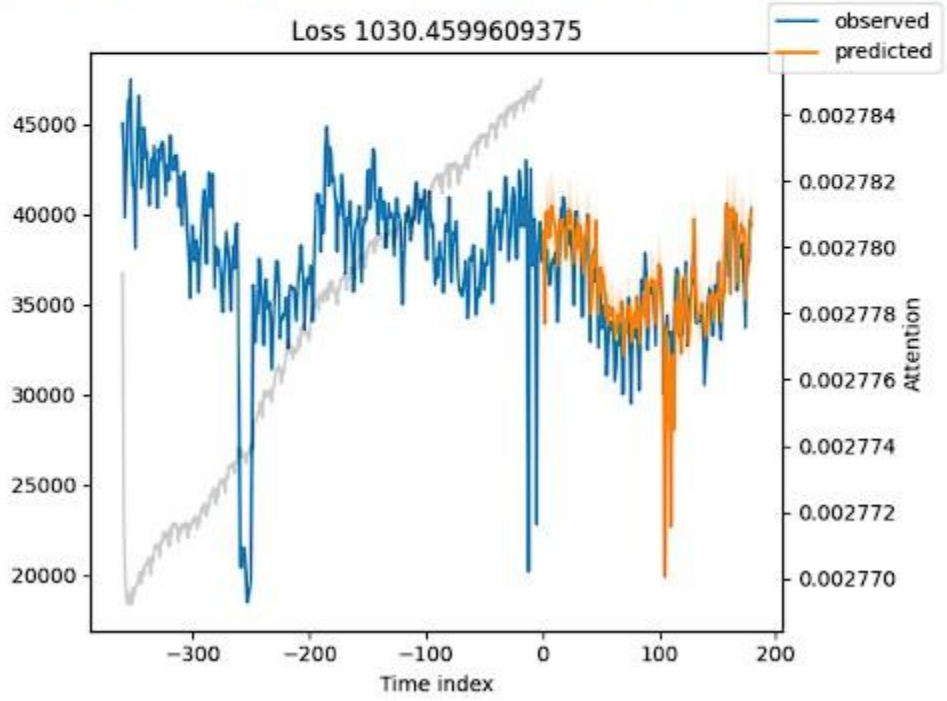
Trained models with the best number of epochs are saved for later and we perform predictions and evperiod.

```
best_model_path1 = trainer1.checkpoint_callback.best_model_path
print(best_model_path1)
best_tft1 = TemporalFusionTransformer.load_from_checkpoint(best_model_path1)
actuals1_p = torch.cat([y[0] for x, y in iter(val_dataloader1)]).to('cuda')
predictions1 = best_tft1.predict(val_dataloader1).to('cuda')
((actuals1_p - predictions1)/(actuals1_p)).abs().mean().item()
```

```
lightning_logs/lightning_logs/version_1/checkpoints/epoch=54-step=1045.ckpt
```

```
raw_predictions1= best_tft1.predict(val_dataloader1, mode="raw", return_x=True)
best_tft1.plot_prediction(raw_predictions1.x, raw_predictions1.output, idx=0, add_loss_to_title=True)
```

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

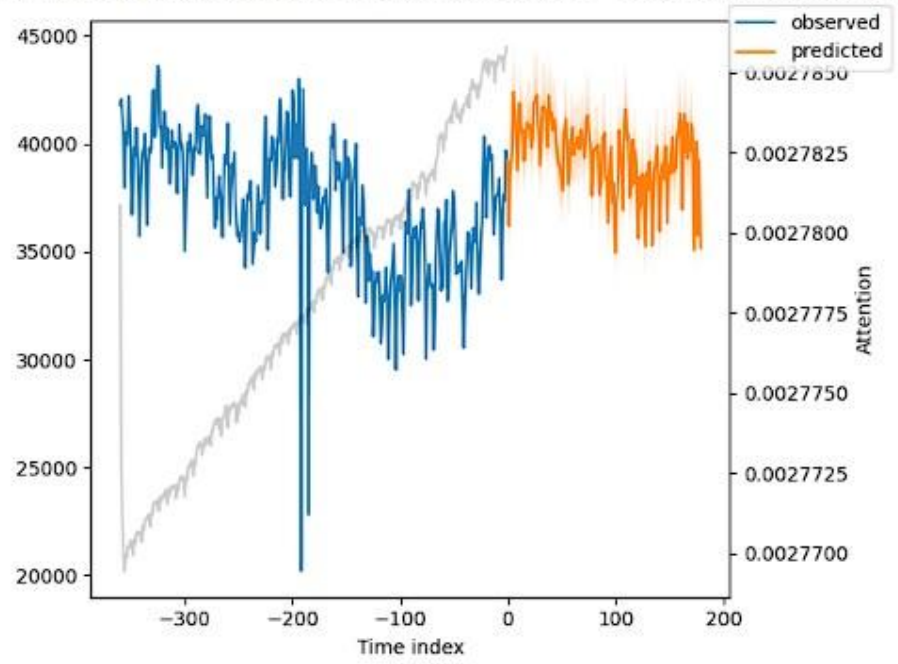


The graph above represents the visualization of predictions on validation data.

The graph below illustrates predictions on test data.

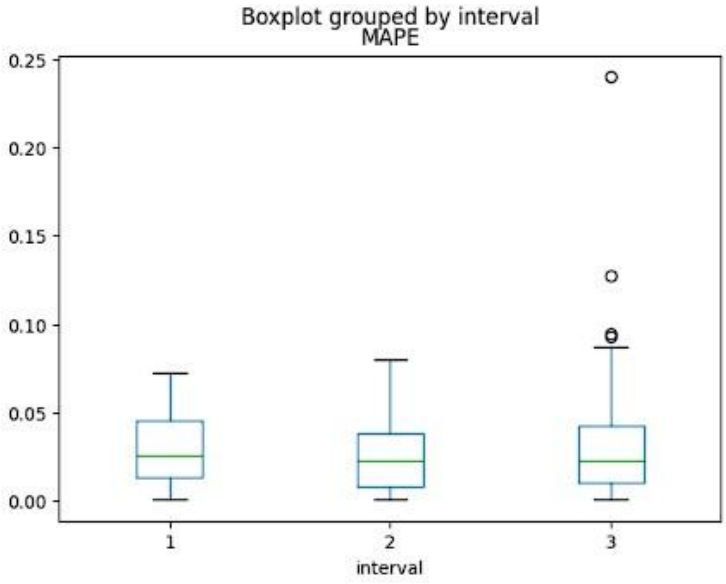
```
new_raw_predictions1 = best_tft1.predict(test_dataloader1, mode="raw", return_x=True)
best_tft1.plot_prediction(new_raw_predictions1.x, new_raw_predictions1.output, idx=0, show_future_observed=False)
```

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



As for the evaluation metric, we choose Absolute Percentage Error (MAPE) because it works for analyzing large sets of data and requires the use of dataset values other than zero.

The testing results must be divided into three periods where they have a time span of 60 days. For each region's MAPE, a box plot of errors is used to show the spread.



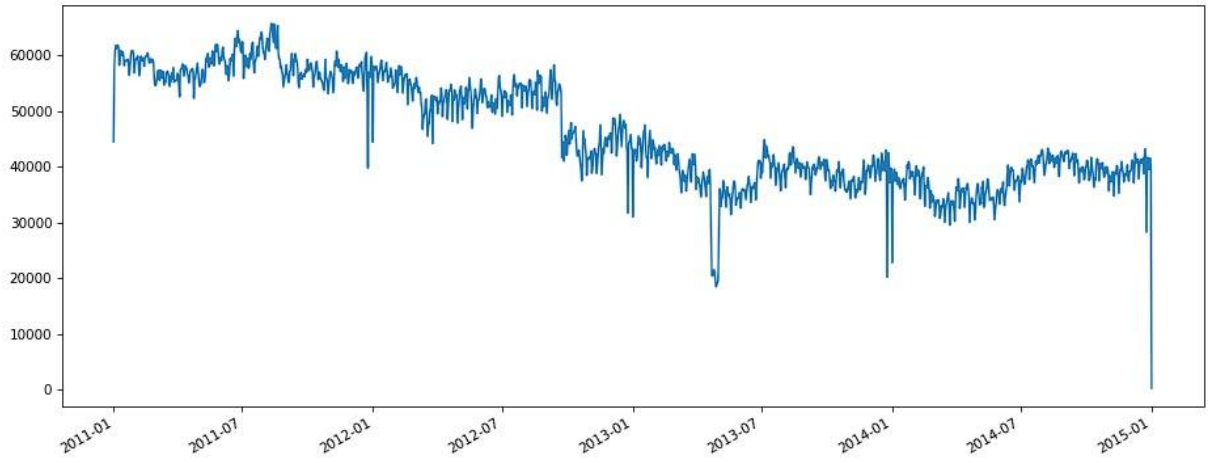

```
mape1 = cluster1['MAPE'].mean()
mape1
```

```
0.029311845577712728
```

Overall, both DeepAR and TFT perform extraordinarily well according to our modeling results. DeepAR is so far ahead of TFT due to its easy deployment and implementation. However, TFT has a more favorable performance over DeepAR because TFT's MAPE is 7.0% while DeepAR's MAPE is 8.9%.

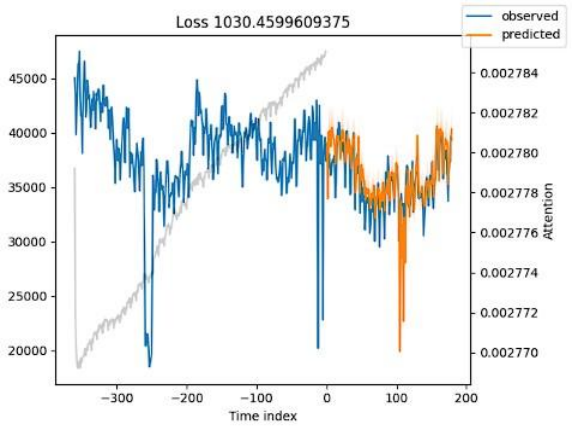
Algorithmic Solution Results

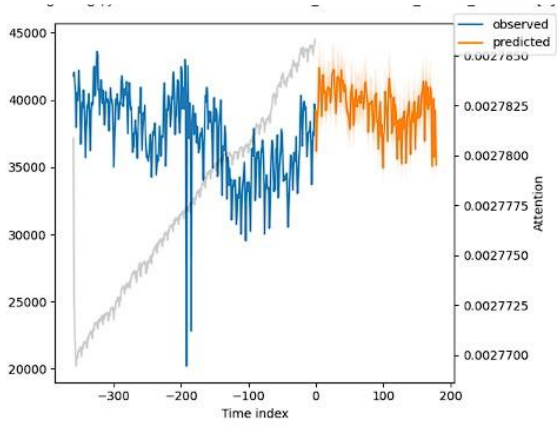
Cluster1



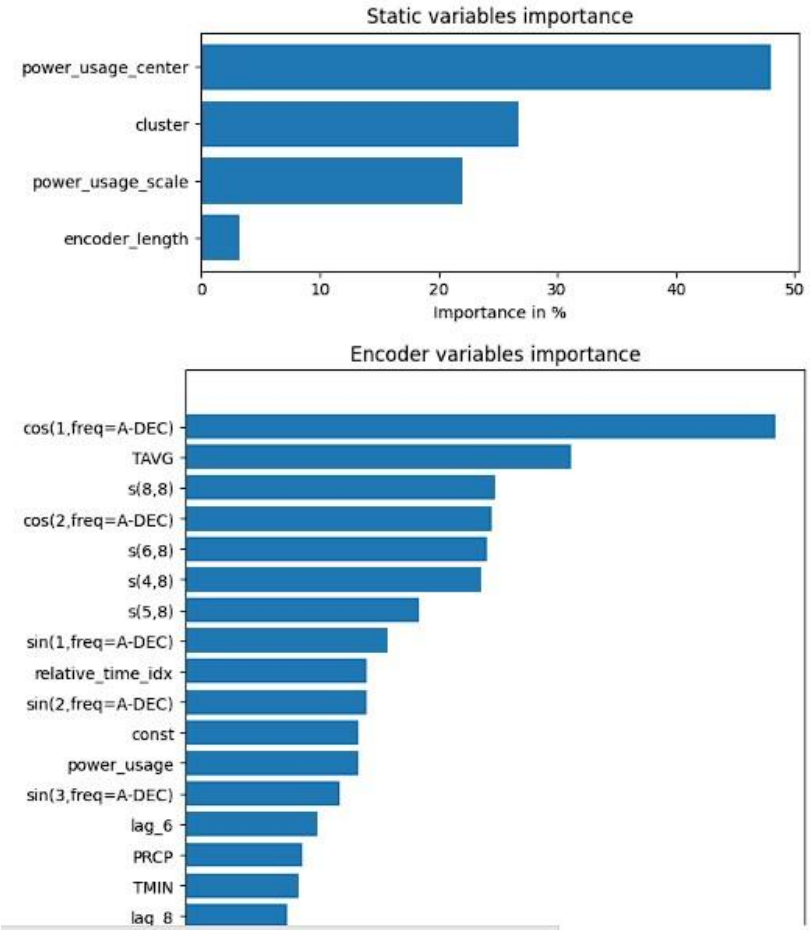
Above is the plot of Cluster 1's data. The scale of the data is relatively large. There is an obvious downward trend in the long run. Daily power usage fluctuates frequently but not drastically. In each year, data peaks during summer and reaches the valley low in January. Data around June 2013 is an anomaly in the data because power usage declines moderately and then bounces back to its norm.

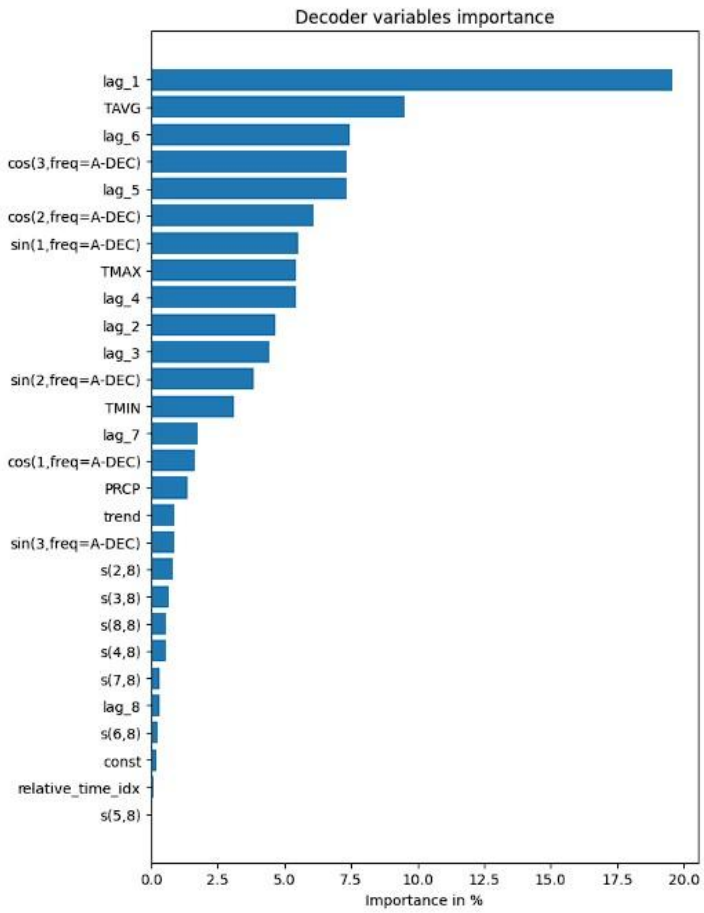
The plots below are the TFT's predictions over the validation and test dataset. This model captures the trend and seasonality of the historical data precisely and even reproduces a little bit of oscillation. However, there is one part in the left graph where the prediction deviates from the true values. There is a sudden drop in data at 100 days from 2015-01-01. The reason behind this anomaly is worth further investigation.



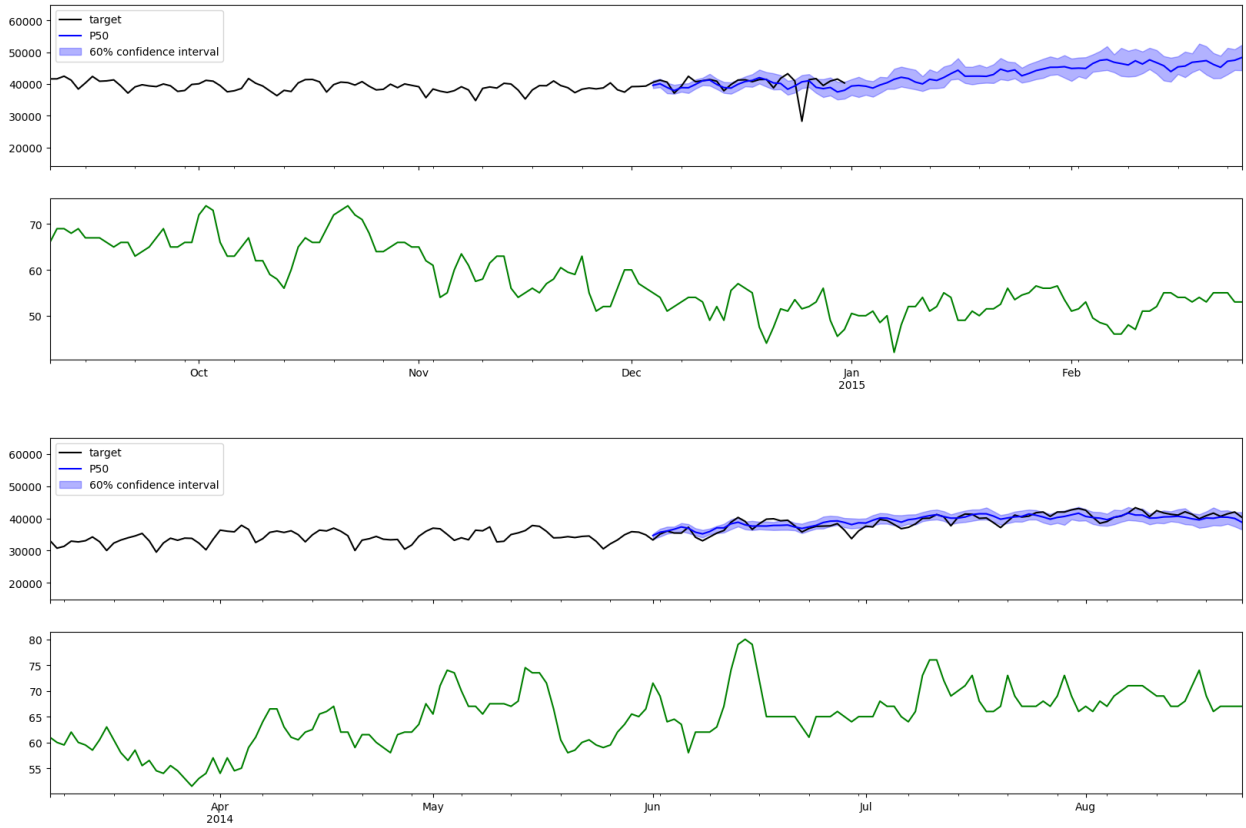


The figures below are feature importances estimated by TFT.

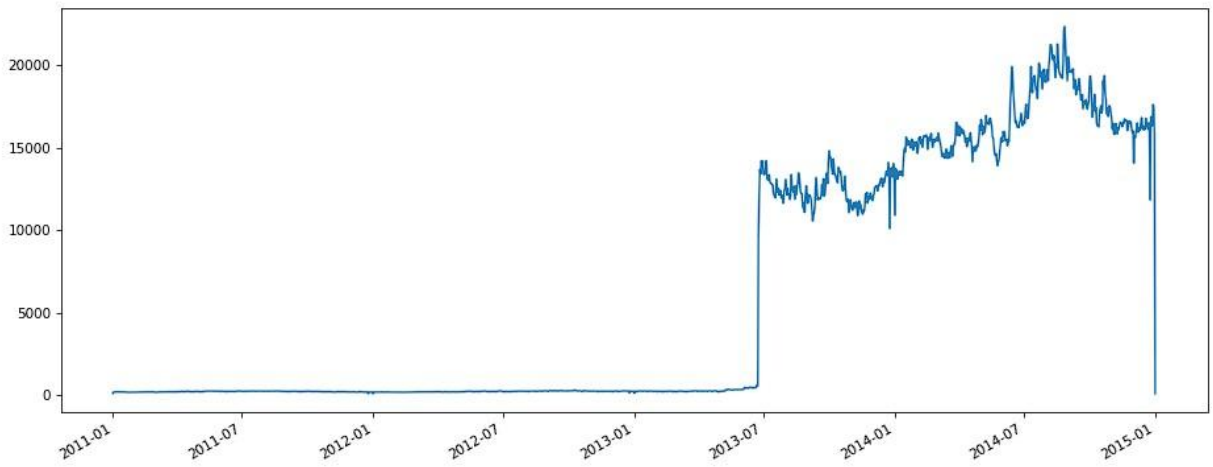




The plots below are DeepAR's predictions over the test set and out-of-sample. They show that DeepAR fits Cluster 1's data well and can make reliable predictions.

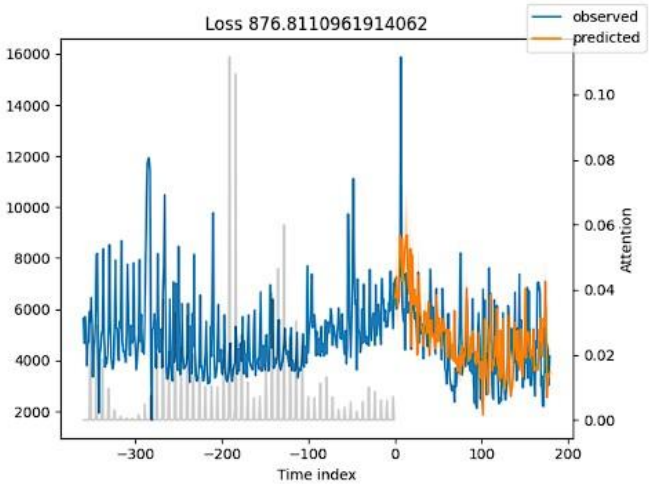


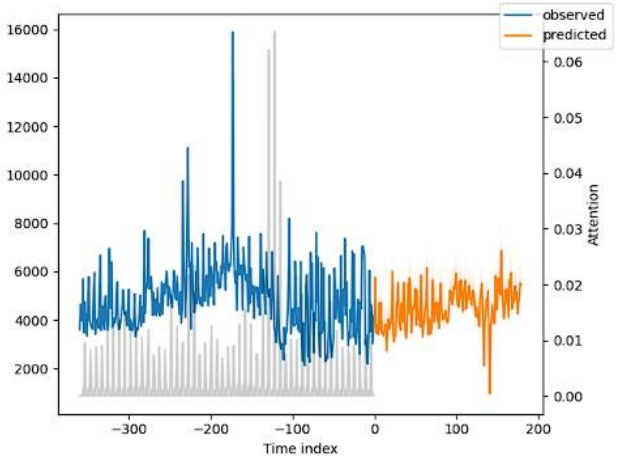
Cluster 2



Above is the plot of Cluster 2's data. The scale of the data is modest. There is no electricity consumption of Cluster 2 in the first two and half years. Power usage

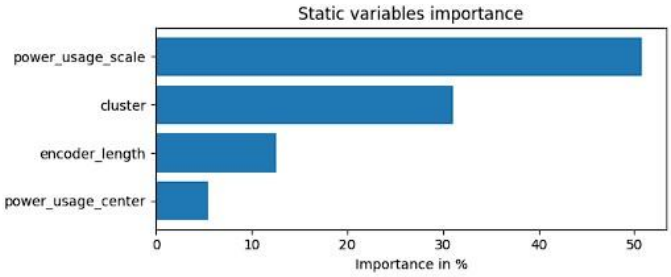
has an upward trend from 2013-07 to 2014-07, and then it goes down fast. Daily power usage fluctuates frequently but not drastically.

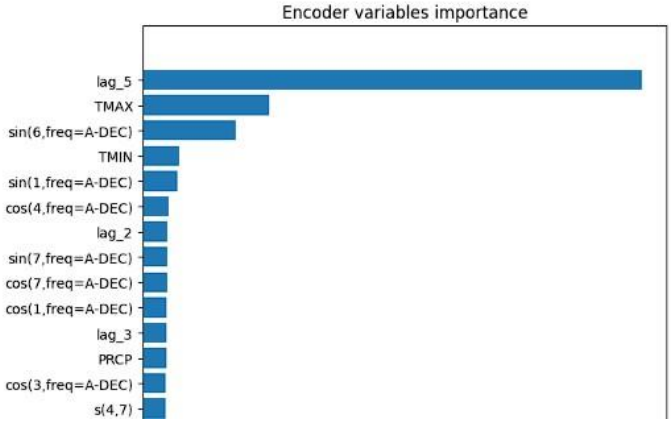


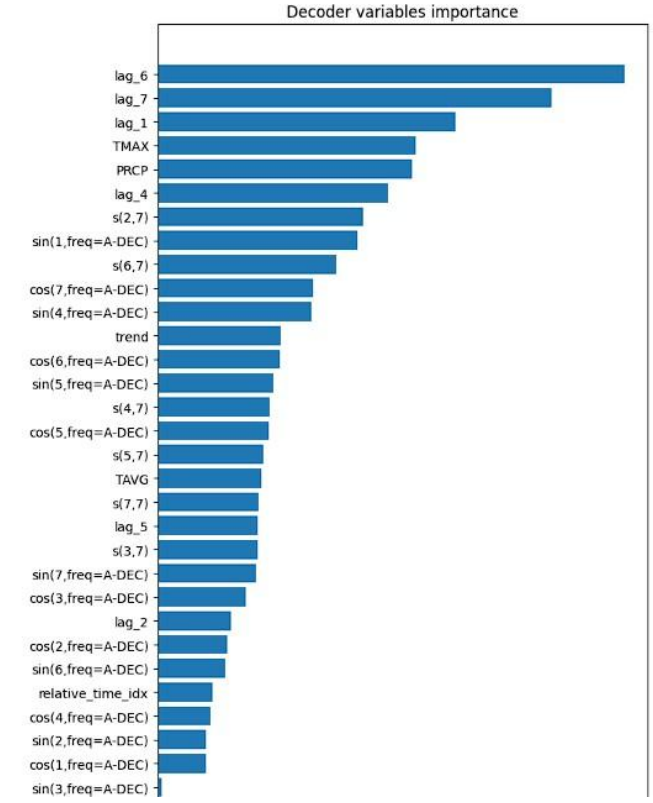


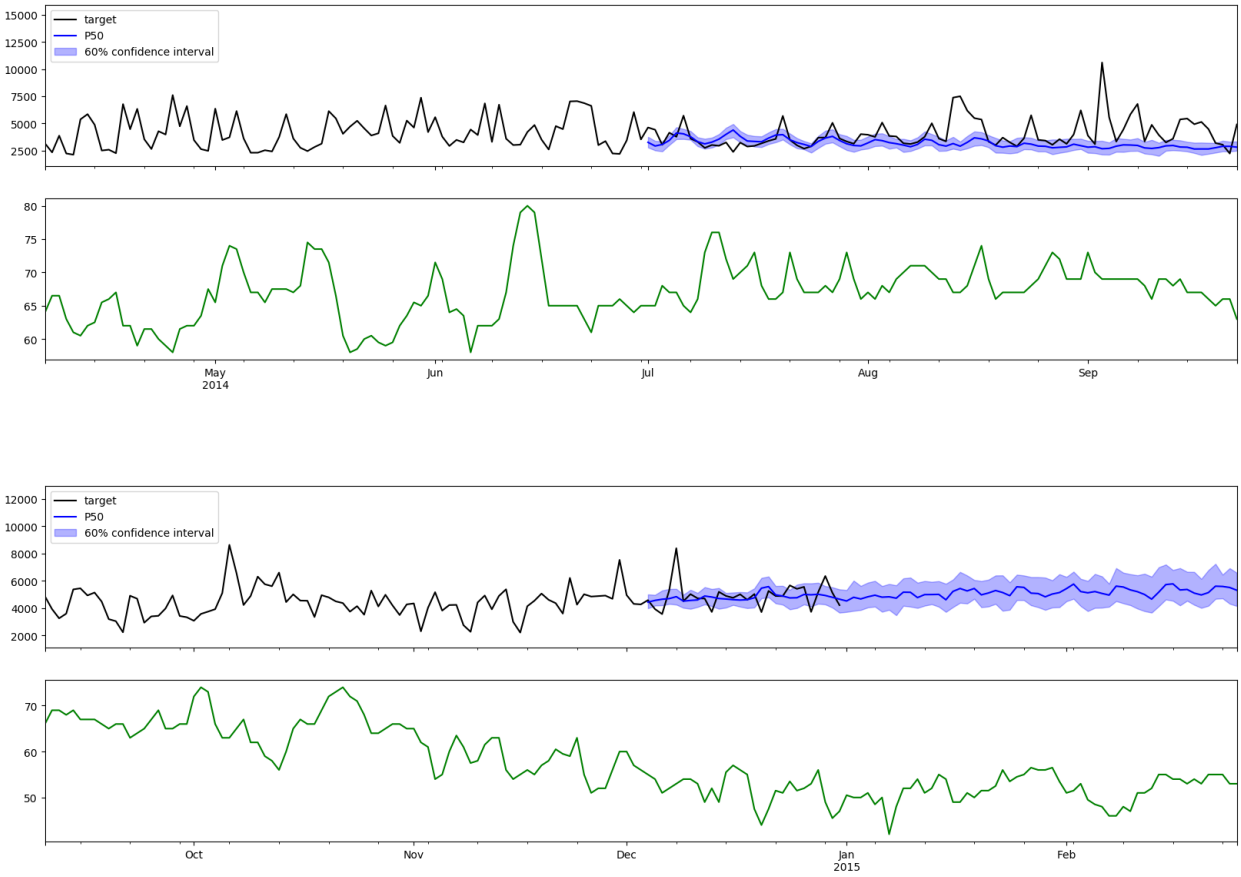
The plots above are the TFT's predictions over the validation and test dataset. This model captures the trend of the historical data. But it does not cope with hidden special events' effects and fluctuations.

The figures below are feature importances estimated by TFT.


Static variables importance
power_usage_scale
cluster
encoder_length
power_usage_center
0
10
20
30
40
50
Importance in %

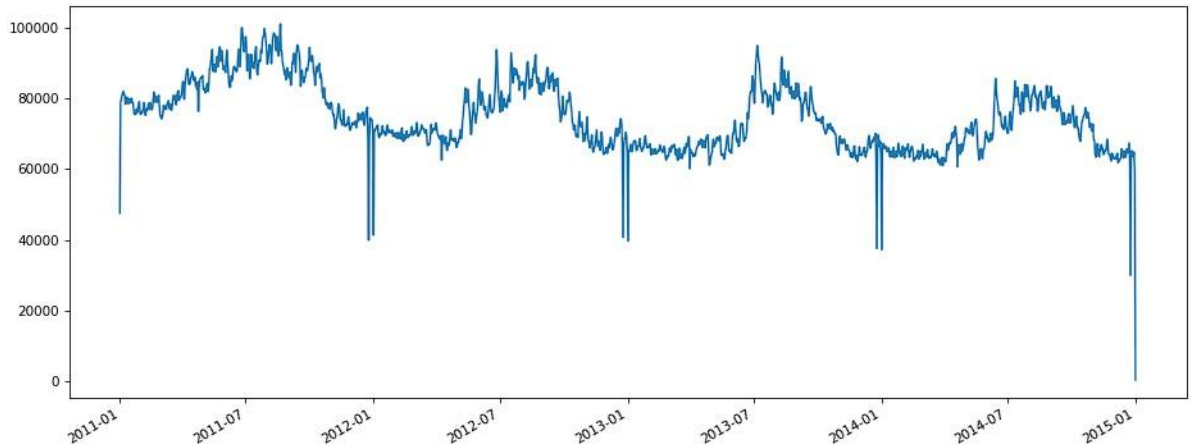

Encoder variables importance
lag_5
TMAX
sin(6,freq=A-DEC)
TMIN
sin(1,freq=A-DEC)
cos(4,freq=A-DEC)
lag_2
sin(7,freq=A-DEC)
cos(7,freq=A-DEC)
cos(1,freq=A-DEC)
lag_3
PRCP
cos(3,freq=A-DEC)
s(4,7)


Decoder variables importance
lag_6
lag_7
lag_1
TMAX
PRCP
lag_4
s(2,7)
sin(1,freq=A-DEC)
s(6,7)
cos(7,freq=A-DEC)
sin(4,freq=A-DEC)
trend
cos(6,freq=A-DEC)
sin(5,freq=A-DEC)
s(4,7)
cos(5,freq=A-DEC)
s(5,7)
TAVG
s(7,7)
lag_5
s(3,7)
sin(7,freq=A-DEC)
cos(3,freq=A-DEC)
lag_2
cos(2,freq=A-DEC)
sin(6,freq=A-DEC)
relative_time_idx
cos(4,freq=A-DEC)
sin(2,freq=A-DEC)
cos(1,freq=A-DEC)
sin(3,freq=A-DEC)

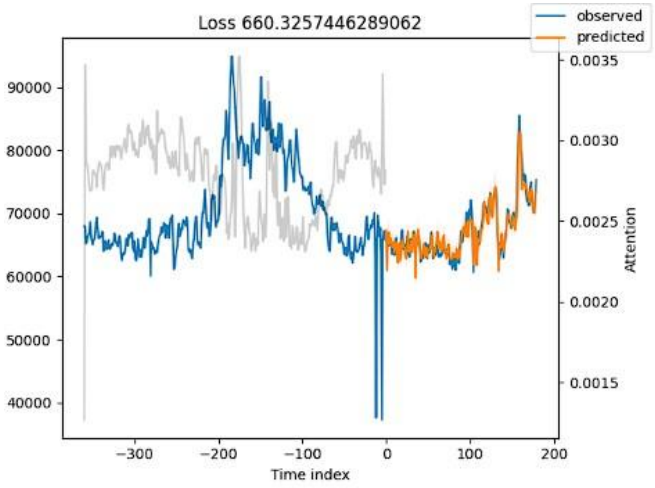


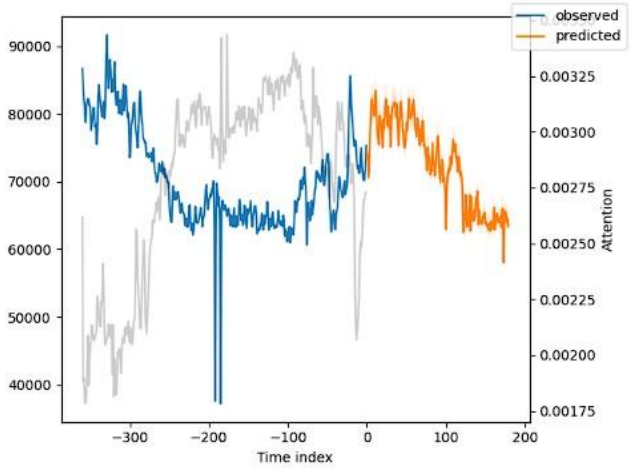
The plots above are DeepAR's predictions over the test set and out-of-sample. Not desirable but acceptable for forecasting.

Cluster 3



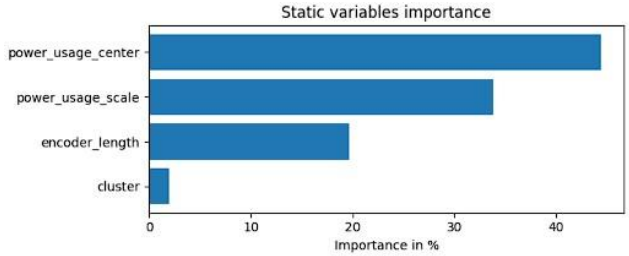
Above is the plot of Cluster 3's data. The scale of the data is relatively large. There is no obvious trend but strong annual seasonality. Daily power usage fluctuates frequently but not drastically. In each year, data peaks during summer and reaches the valley low in January.

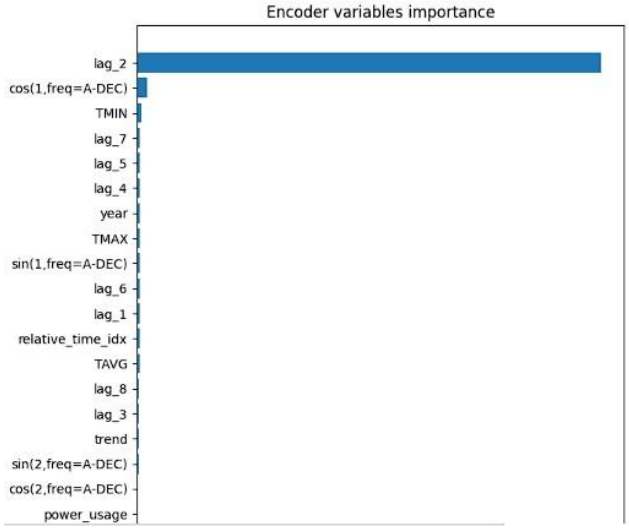


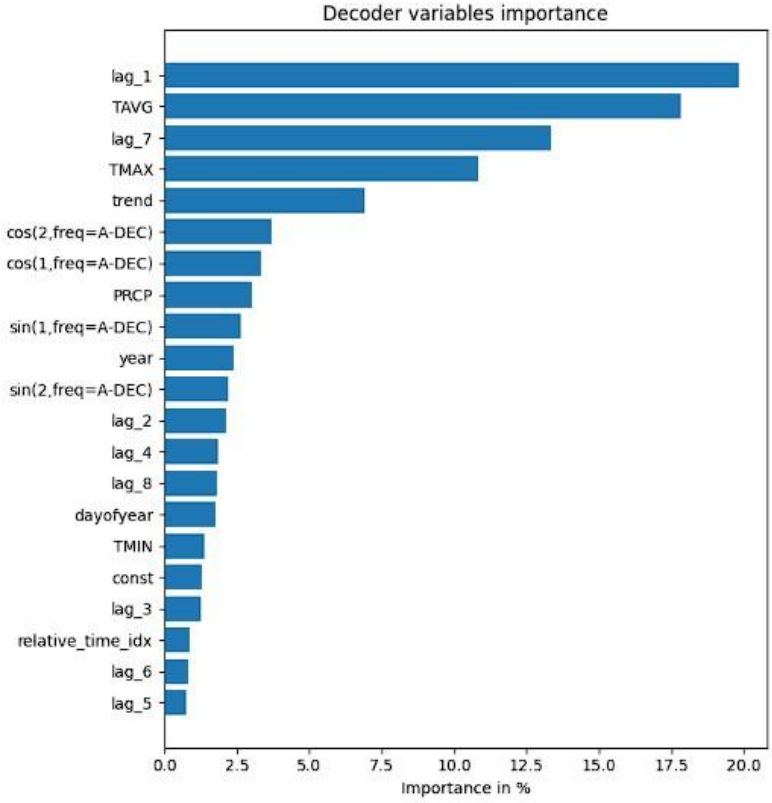


The plots above are the TFT's predictions over the validation and test dataset. This model mimics the seasonality of the historical data precisely. It is dependable for forecasting.

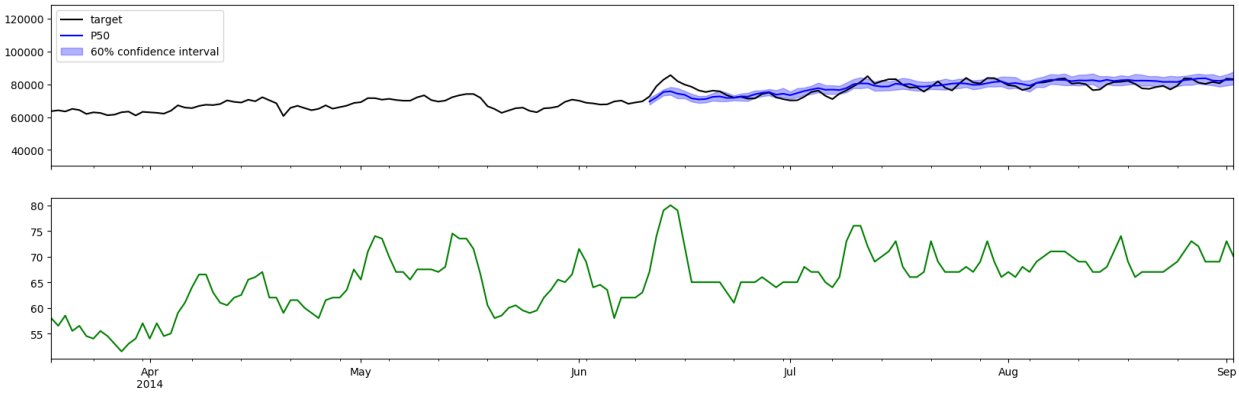
The figures below are feature importances estimated by TFT.

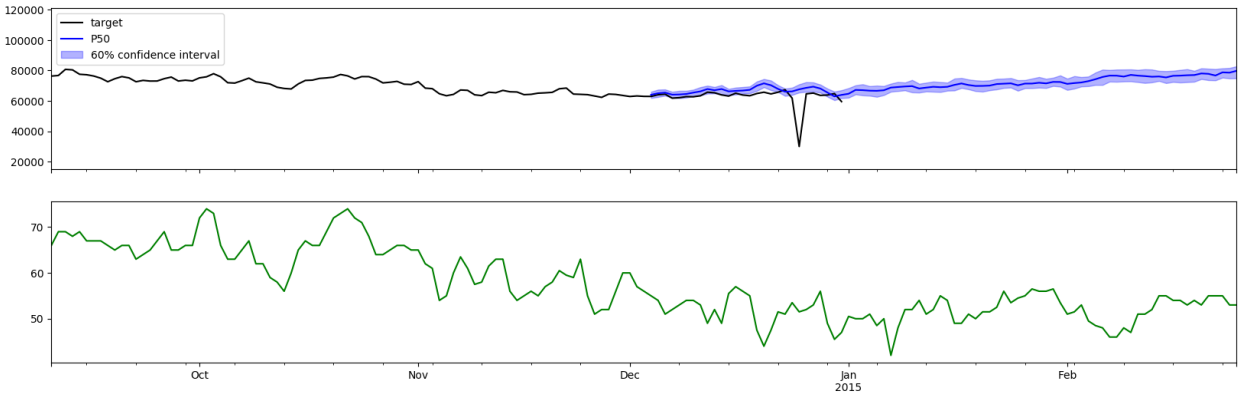

Static variables importance
power_usage_center
power_usage_scale
encoder_length
cluster
0
10
20
30
40
Importance in %


Encoder variables importance
lag_2
cos(1,freq=A-DEC)
TMIN
lag_7
lag_5
lag_4
year
TMAX
sin(1,freq=A-DEC)
lag_6
lag_1
relative_time_idx
TAVG
lag_8
lag_3
trend
sin(2,freq=A-DEC)
cos(2,freq=A-DEC)
power_usage

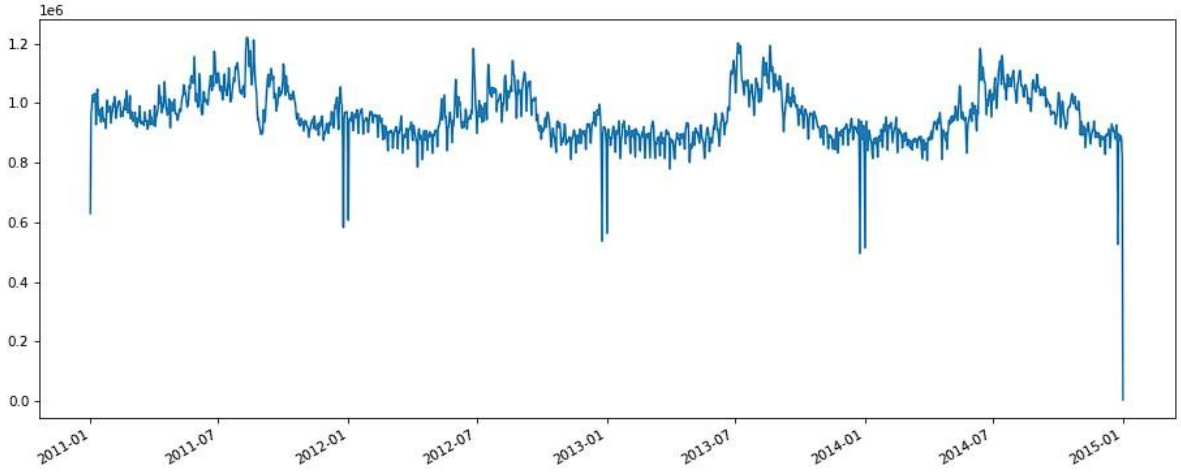

Decoder variables importance
lag_1
TAVG
lag_7
TMAX
trend
cos(2,freq=A-DEC)
cos(1,freq=A-DEC)
PRCP
sin(1,freq=A-DEC)
year
sin(2,freq=A-DEC)
lag_2
lag_4
lag_8
dayofyear
TMIN
const
lag_3
relative_time_idx
lag_6
lag_5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Importance in %

The graphs below demonstrate DeepAR's predictions over the test set and out-of-sample. They show that DeepAR fits Cluster 3's data well and can make reliable predictions.

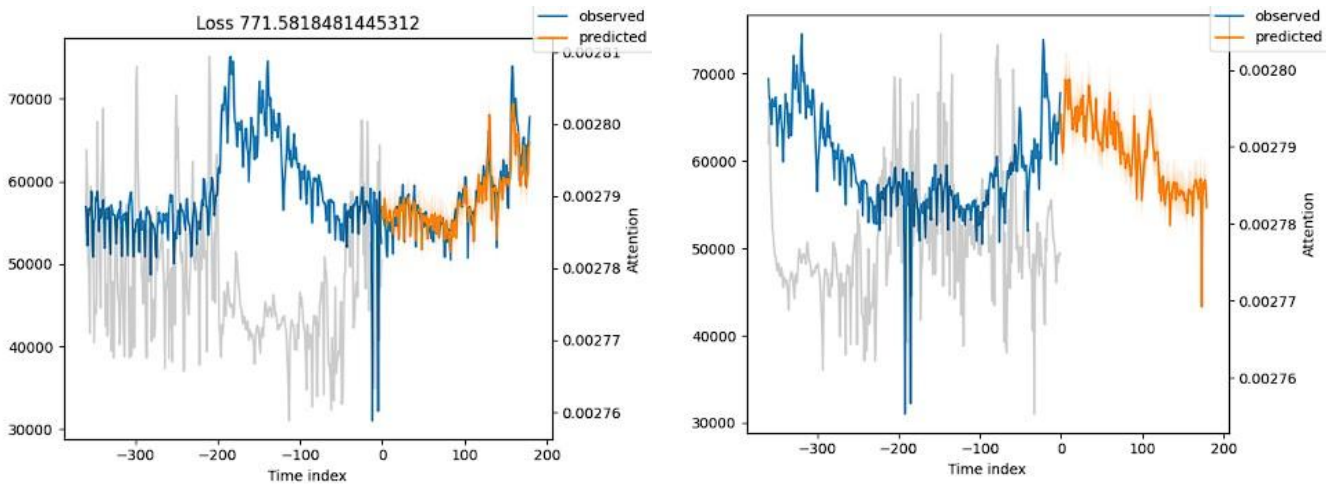




Cluster 4

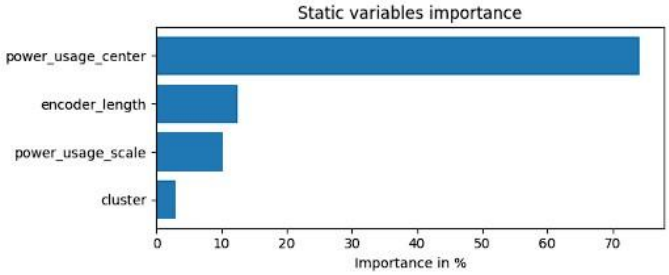


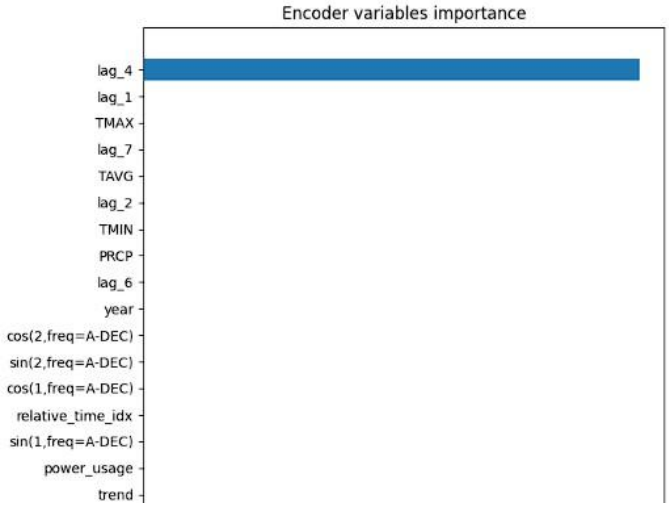
Above is the plot of Cluster 4's data. The scale of the data is incredibly large. There is no obvious trend but strong annual seasonality. Daily power usage fluctuates frequently but not drastically. In each year, data peaks during summer and reaches the valley low in January.

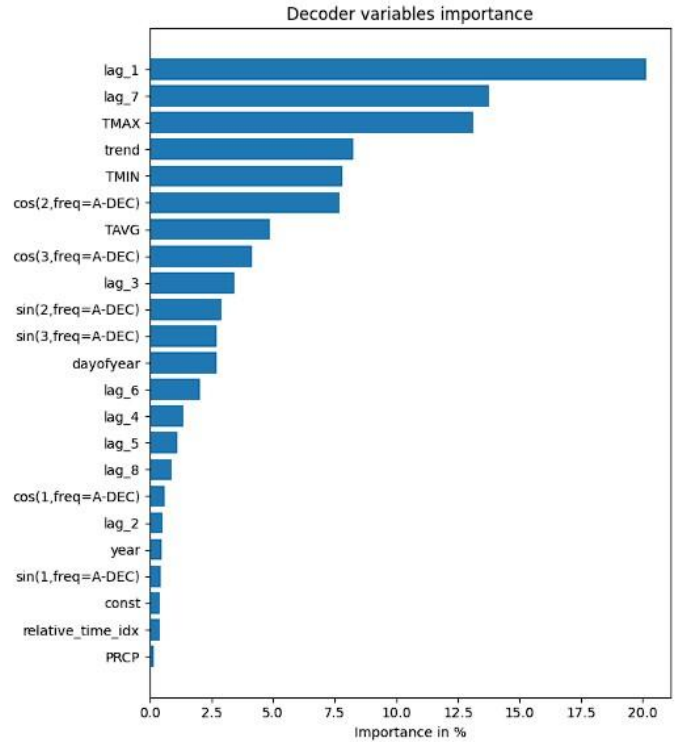


The diagrams above are the TFT's predictions over the validation and test dataset. This model mimics the seasonality of the historical data precisely. It is dependable for forecasting.

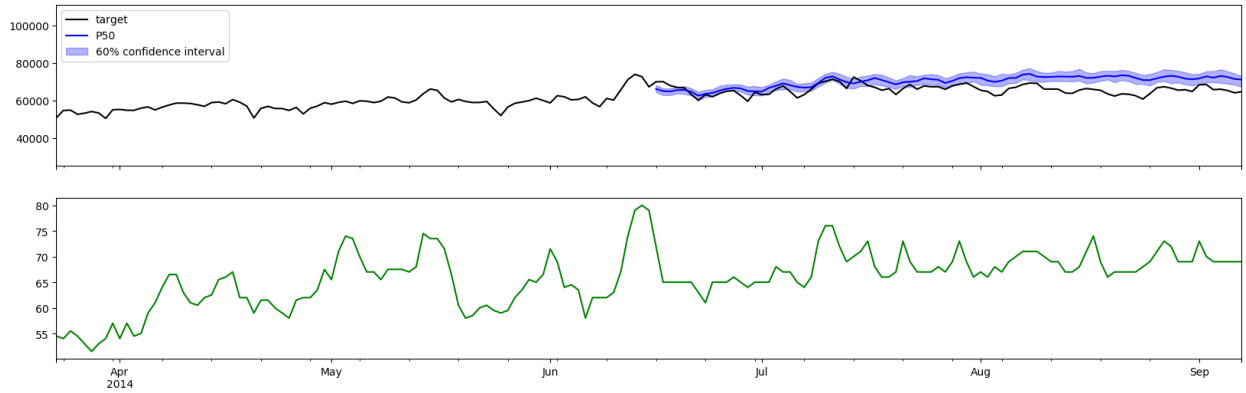
The figures below are feature importances estimated by TFT.

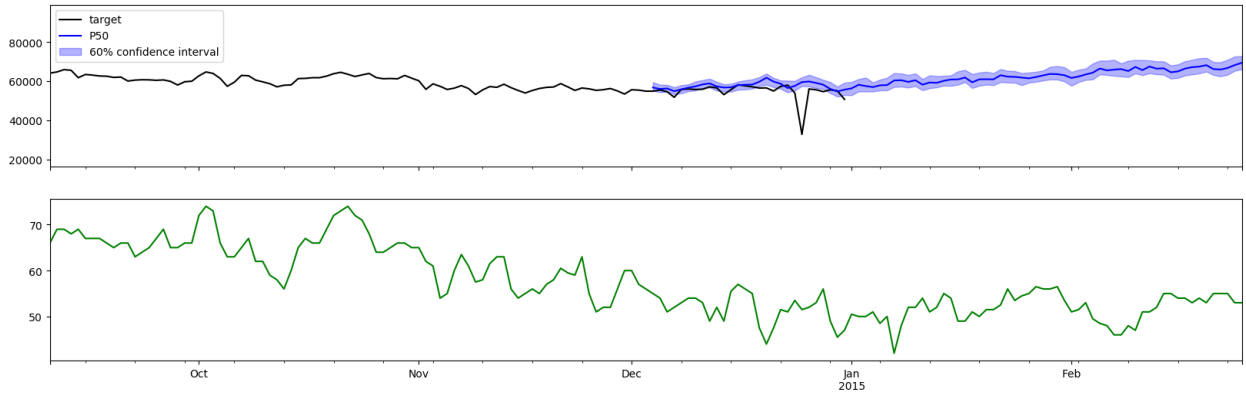

Static variables importance
power_usage_center
encoder_length
power_usage_scale
cluster
0
10
20
30
40
50
60
70
Importance in %


Encoder variables importance
lag_4
lag_1
TMAX
lag_7
TAVG
lag_2
TMIN
PRCP
lag_6
year
cos(2,freq=A-DEC)
sin(2,freq=A-DEC)
cos(1,freq=A-DEC)
relative_time_idx
sin(1,freq=A-DEC)
power_usage
trend

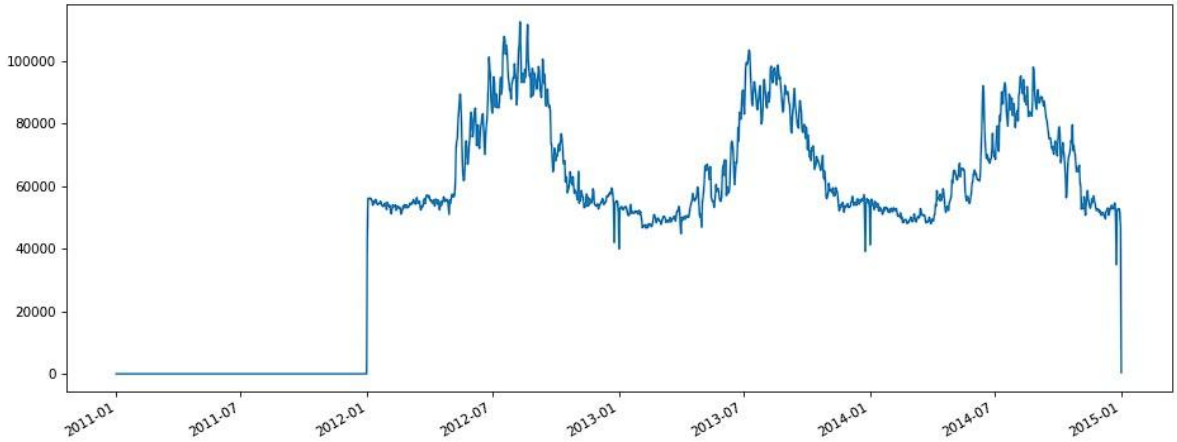

Decoder variables importance
lag_1
lag_7
TMAX
trend
TMIN
cos(2,freq=A-DEC)
TAVG
cos(3,freq=A-DEC)
lag_3
sin(2,freq=A-DEC)
sin(3,freq=A-DEC)
dayofyear
lag_6
lag_4
lag_5
lag_8
cos(1,freq=A-DEC)
lag_2
year
sin(1,freq=A-DEC)
const
relative_time_idx
PRCP
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Importance in %

The graphs below demonstrate DeepAR's predictions over the test set and out-of-sample. They show that DeepAR fits Cluster 4's data well and can make reliable predictions.

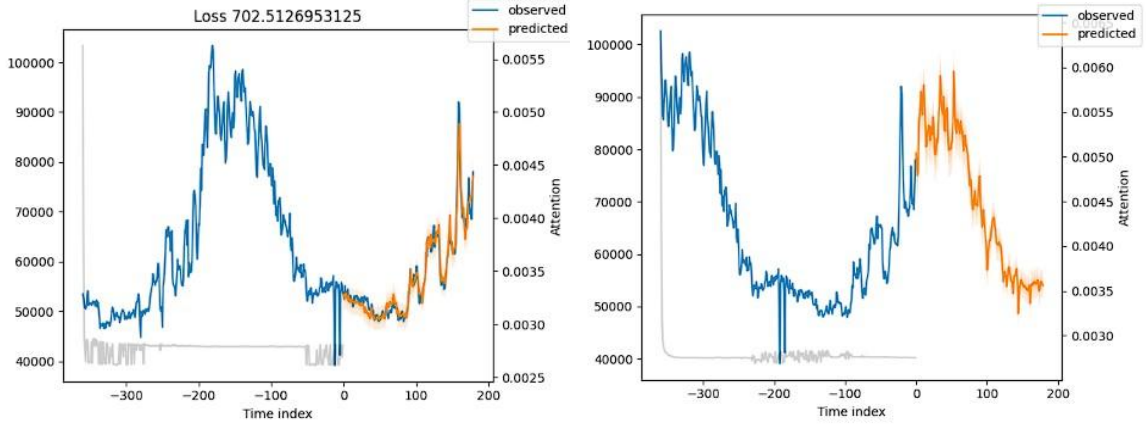




Cluster 5

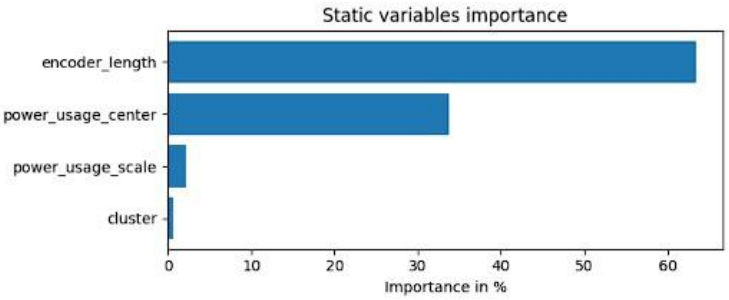


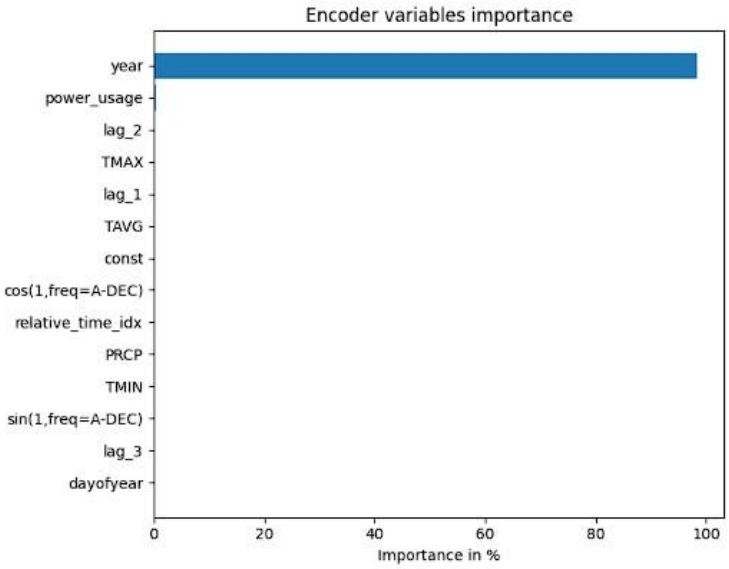
Above is the plot of Cluster 5's data. The scale of the data is reasonably large. There is no electricity consumption of Cluster 5 in the first year. There is a very minimal downward trend but strong annual seasonality. Daily power usage fluctuates frequently but not drastically. In each year, data peaks during summer and reaches the valley low in January.

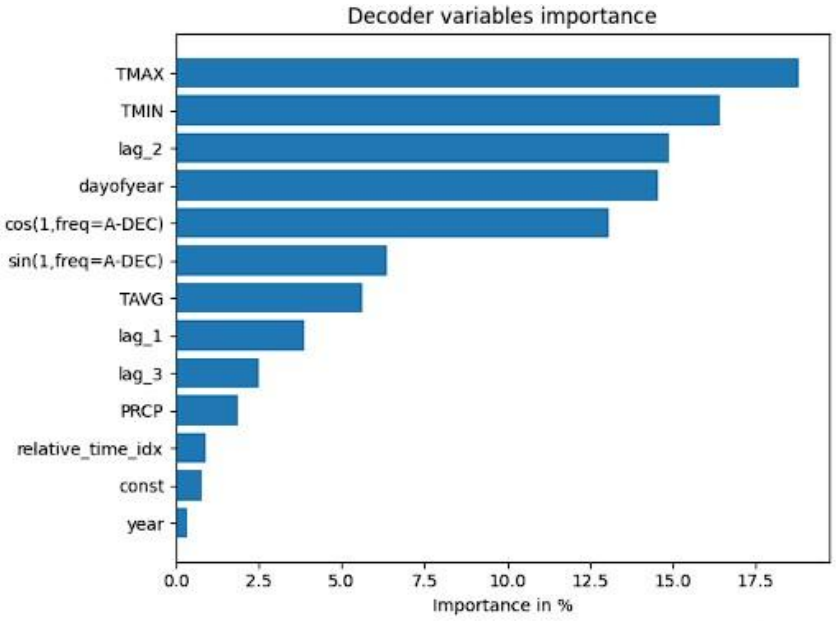


The diagrams above are the TFT's predictions over the validation and test dataset. This model mimics the seasonality of the historical data precisely. It is dependable for forecasting.

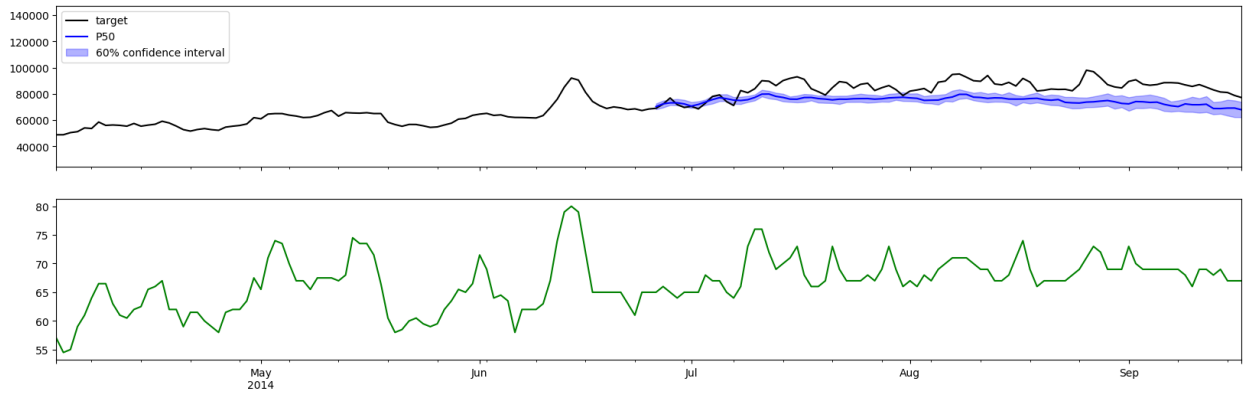
The figures below are feature importances estimated by TFT.

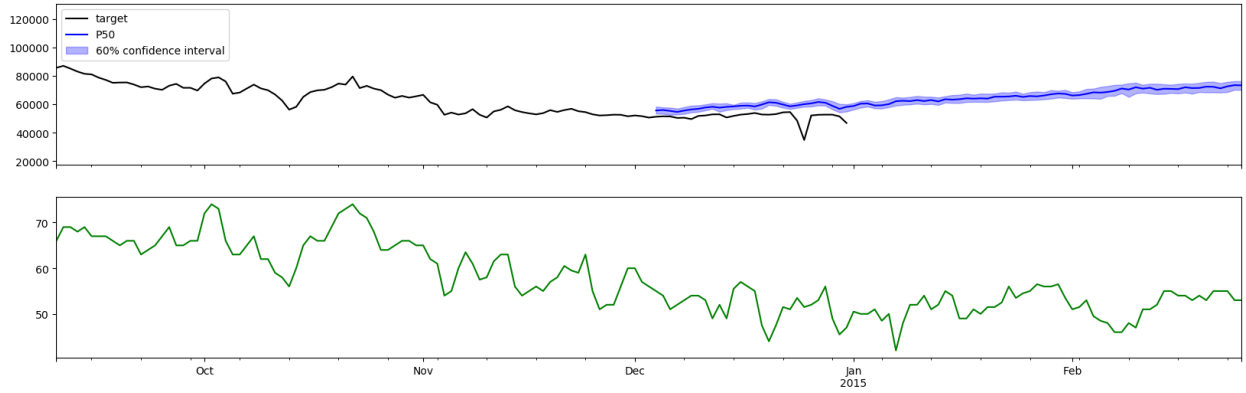




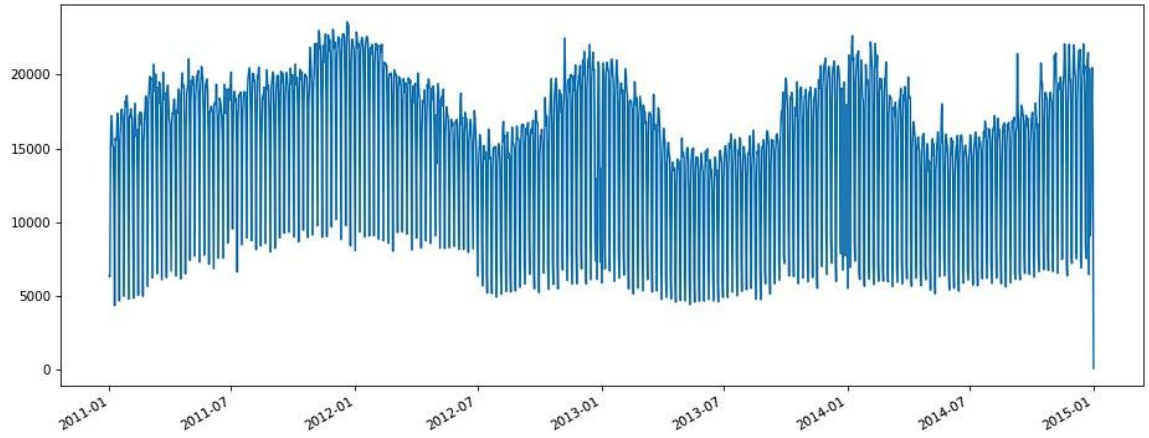


The graphs below demonstrate DeepAR's predictions over the test set and out-of-sample. It is a decent model but could be better.

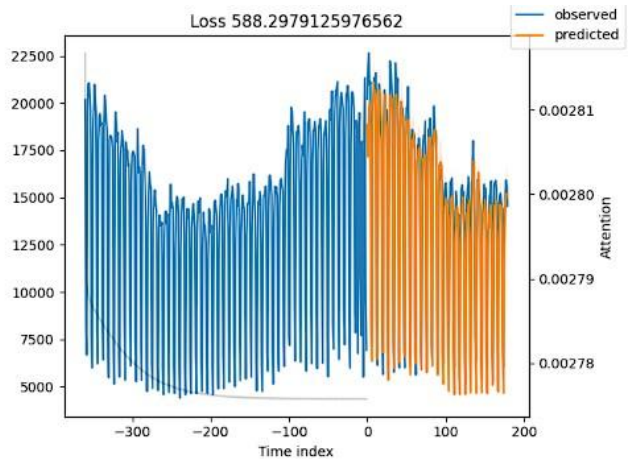


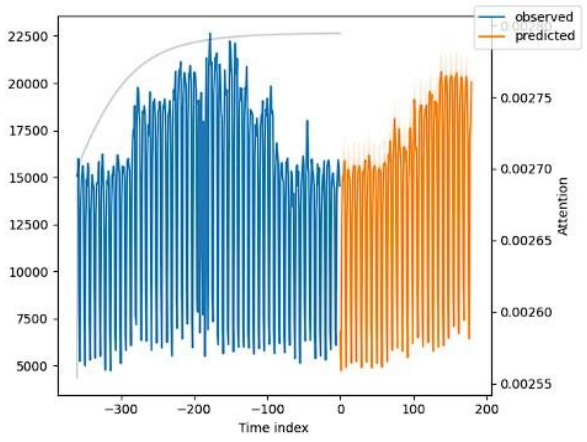


Cluster 6



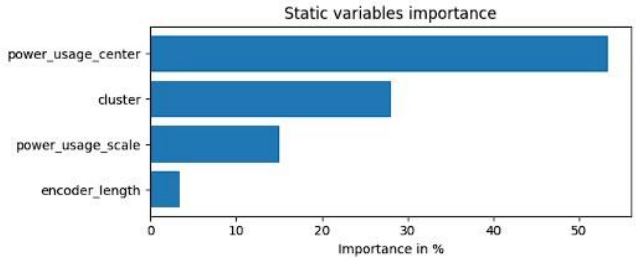
Above is the plot of Cluster 6's data. The scale of the data is reasonably small. There is an upward trend in the first year. However, data shows no trend but a strong annual seasonality for the rest of the time. Daily power usage fluctuates drastically. In each year, data peaks during winter and reaches the valley low during summer.

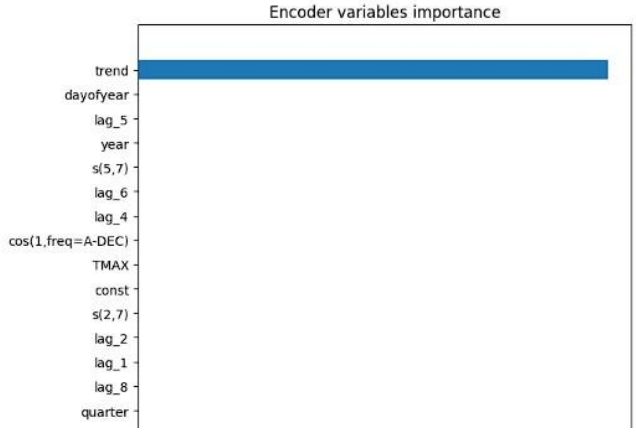


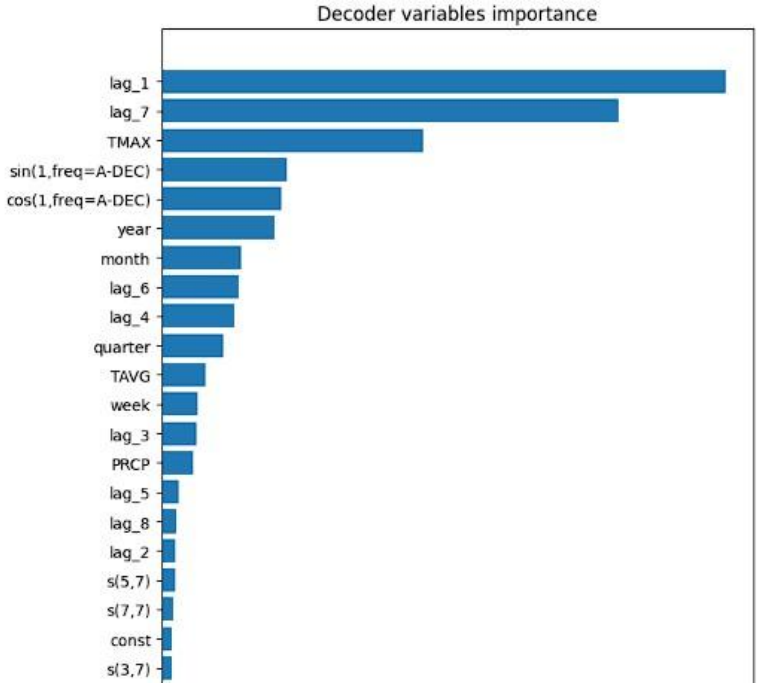


The diagrams above are the TFT's predictions over the validation and test dataset. This model mimics the historical data precisely. It is dependable for forecasting.

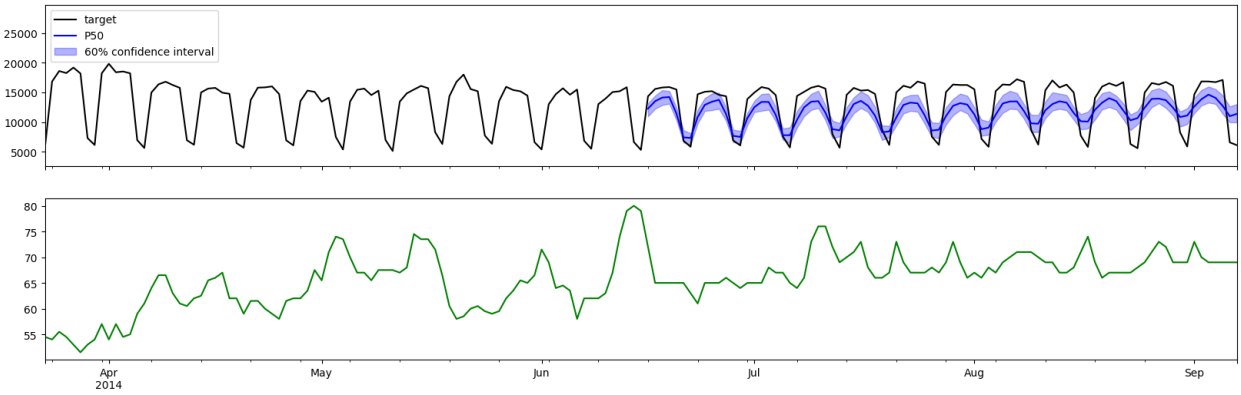
The figures below are feature importances estimated by TFT.

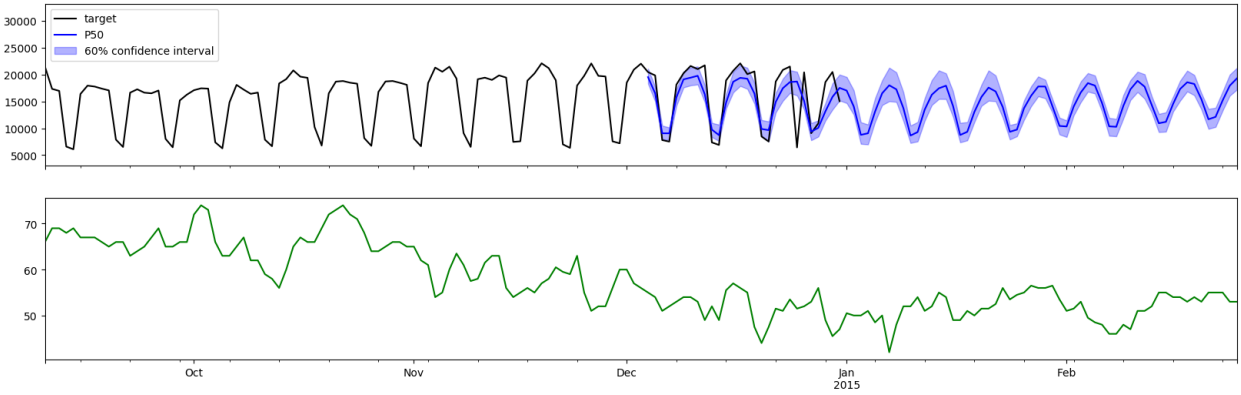




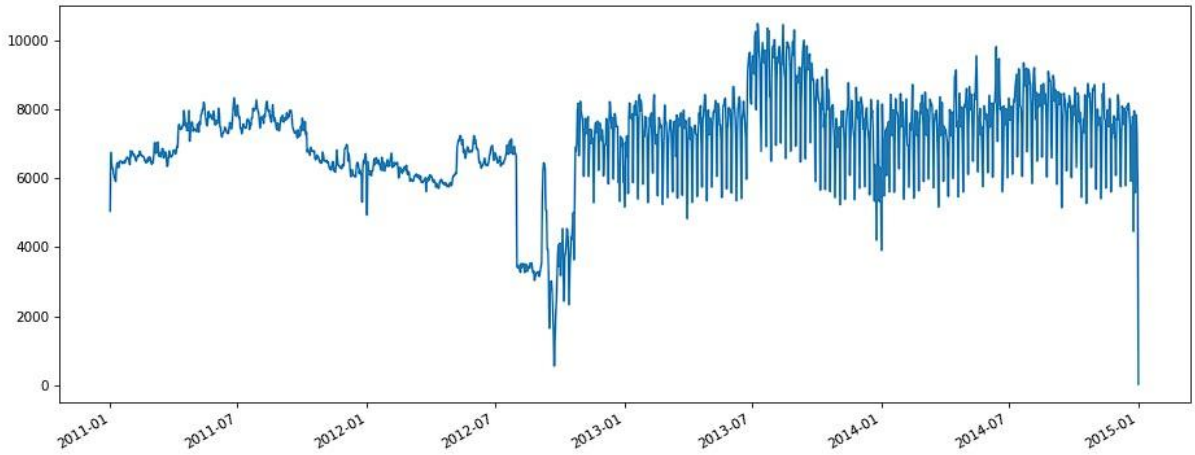


The graphs below demonstrate DeepAR's predictions over the test set and out-of-sample. They show that DeepAR captures the periodicity of Cluster 6's data well and can make reliable predictions.

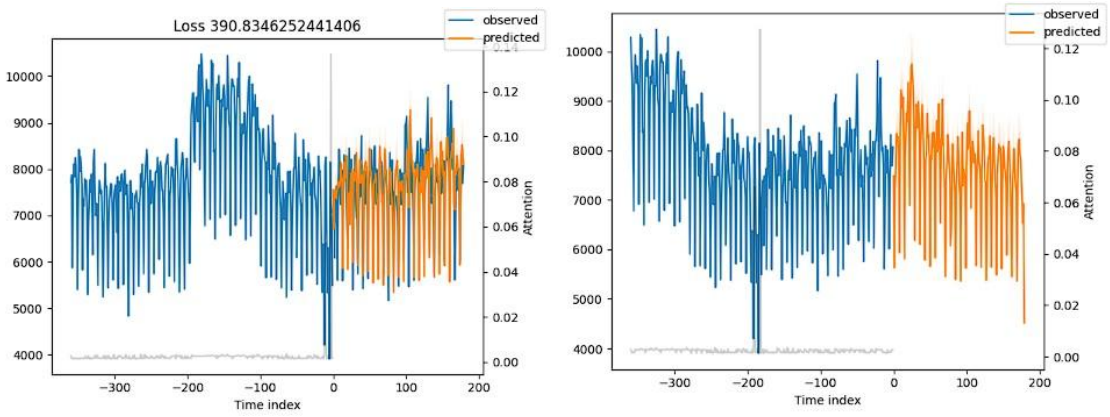

target
P50
60% confidence interval
25000
20000
15000
10000
5000
80
75
70
65
60
55
Apr
2014
May
Jun
Jul
Aug
Sep


target
P50
60% confidence interval
30000
25000
20000
15000
10000
5000
70
60
50
Oct
Nov
Dec
Jan
2015
Feb

Cluster 7

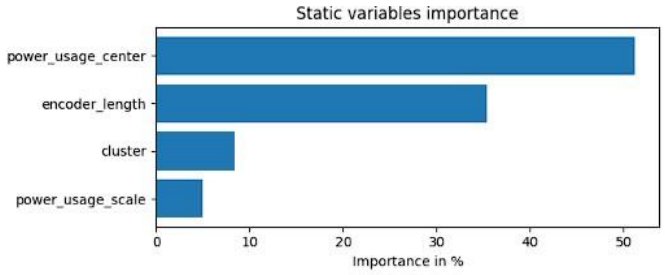


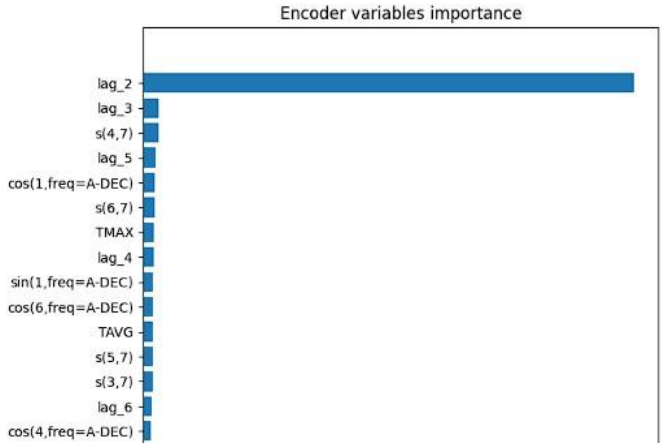
Above is the plot of Cluster 7's data. The scale of the data is very small. There is no regular pattern that could be obtained via observation. Daily power usage fluctuates mildly in the first year and ten months and then shrinks drastically. Then they back to 7000kW per day level.

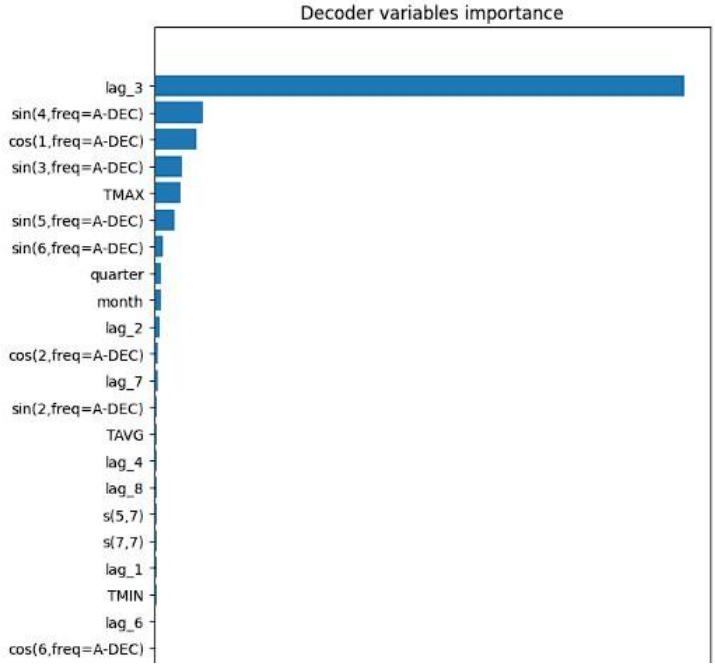


The diagrams above are the TFT's predictions over the validation and test dataset. This model mimics the ups and downs of historical data precisely. It is dependable for forecasting.

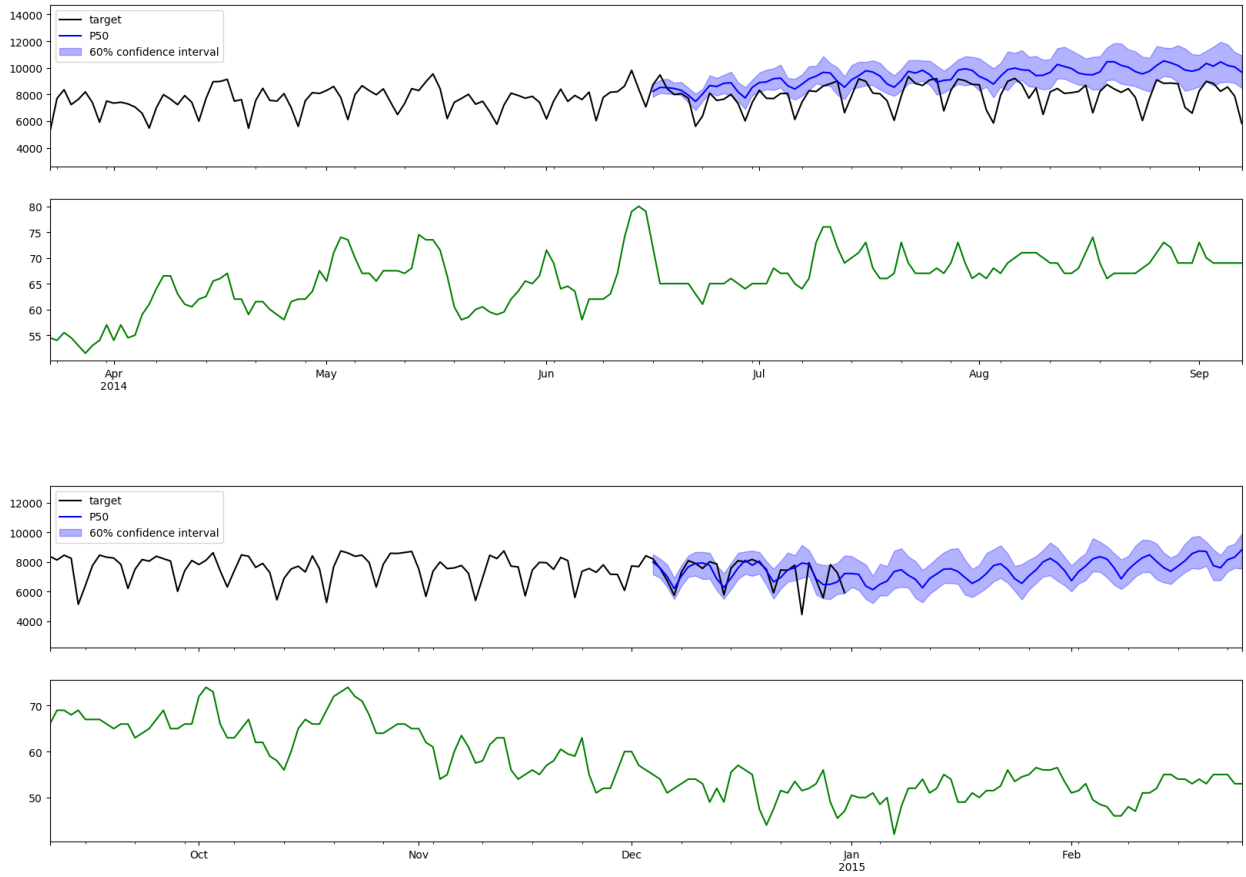
The figures below are feature importances estimated by TFT.


Static variables importance
power_usage_center
encoder_length
cluster
power_usage_scale
0
10
20
30
40
50
Importance in %

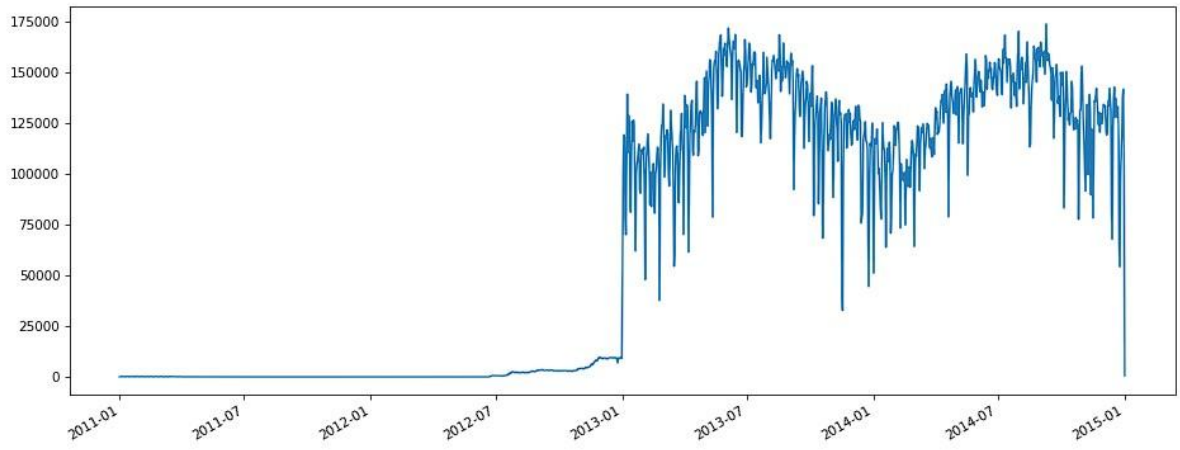

Encoder variables importance
lag_2
lag_3
s(4,7)
lag_5
cos(1,freq=A-DEC)
s(6,7)
TMAX
lag_4
sin(1,freq=A-DEC)
cos(6,freq=A-DEC)
TAVG
s(5,7)
s(3,7)
lag_6
cos(4,freq=A-DEC)


Decoder variables importance
lag_3
sin(4,freq=A-DEC)
cos(1,freq=A-DEC)
sin(3,freq=A-DEC)
TMAX
sin(5,freq=A-DEC)
sin(6,freq=A-DEC)
quarter
month
lag_2
cos(2,freq=A-DEC)
lag_7
sin(2,freq=A-DEC)
TAVG
lag_4
lag_8
s(5,7)
s(7,7)
lag_1
TMIN
lag_6
cos(6,freq=A-DEC)

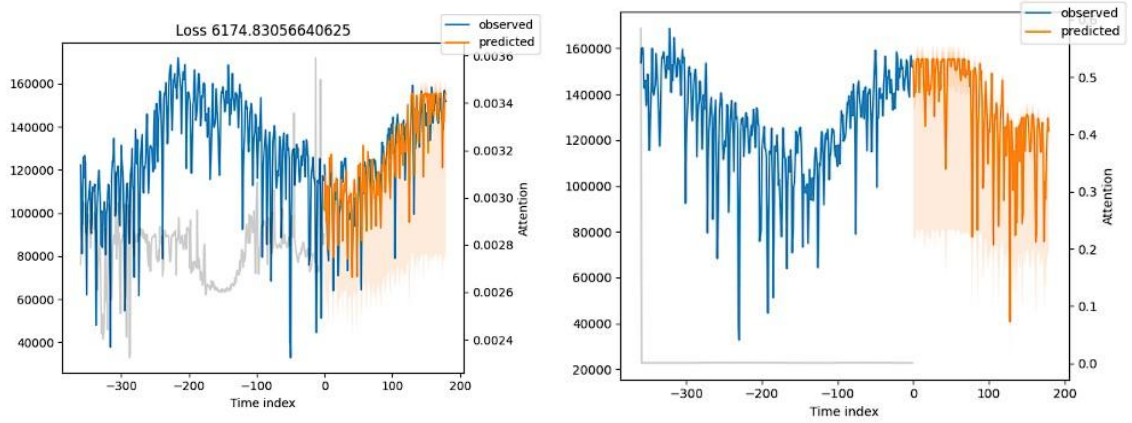
The graphs below demonstrate DeepAR's predictions over the test set and out-of-sample. They show that DeepAR captures the periodicity of Cluster 7's data and there is a lot of room for improvement.



Cluster 8

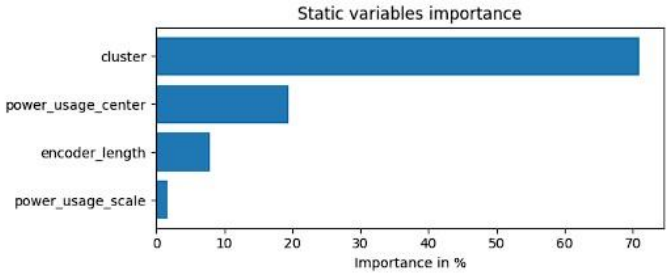


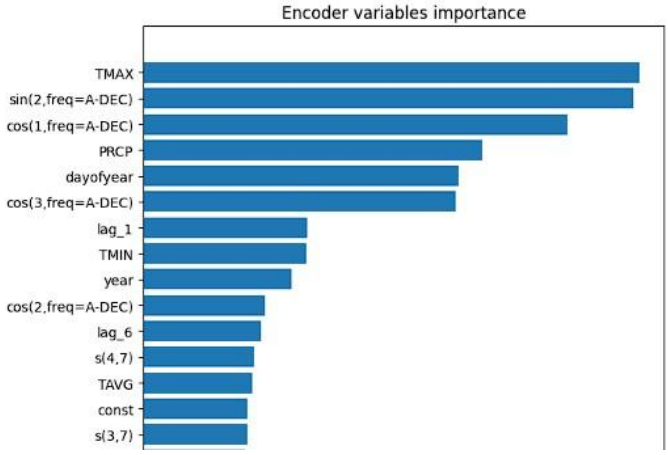
Above is the plot of Cluster 8's data. The scale of the data is the biggest among other clusters' data. There is no electricity consumption in the two years. There is no obvious trend but strong annual seasonality. Daily power usage fluctuates drastically. In each year, data peaks during summer and reaches the valley low during winter.

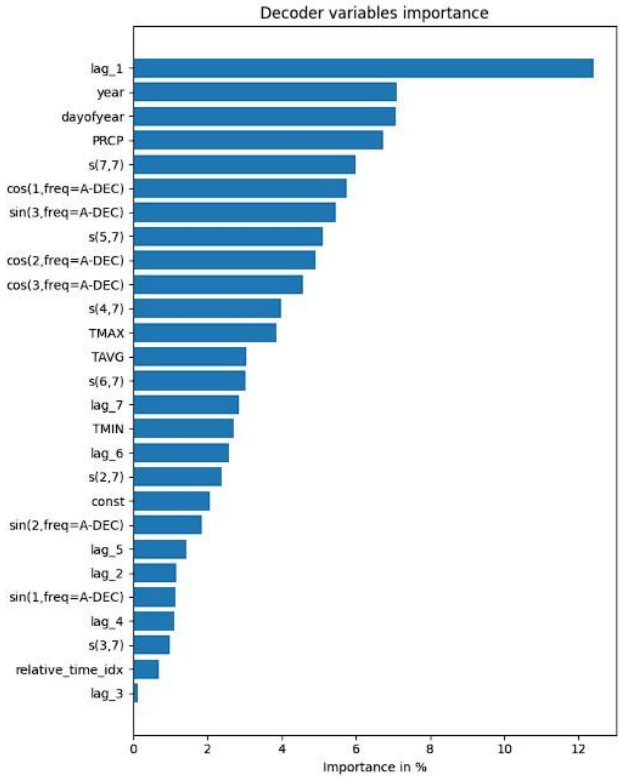


The diagrams above are the TFT's predictions over the validation and test dataset. This model learns the patterns of historical time series data and is capable of predicting accurately.

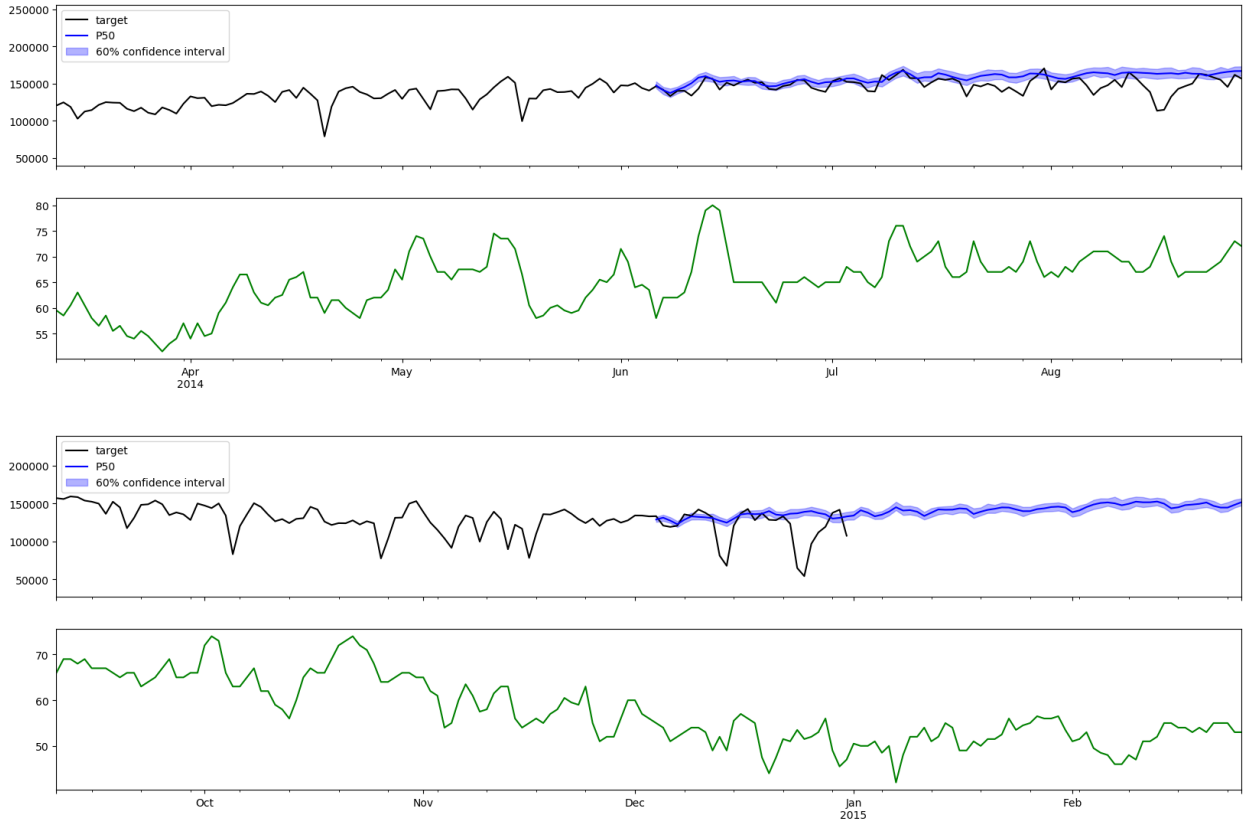
The figures below are feature importances estimated by TFT.



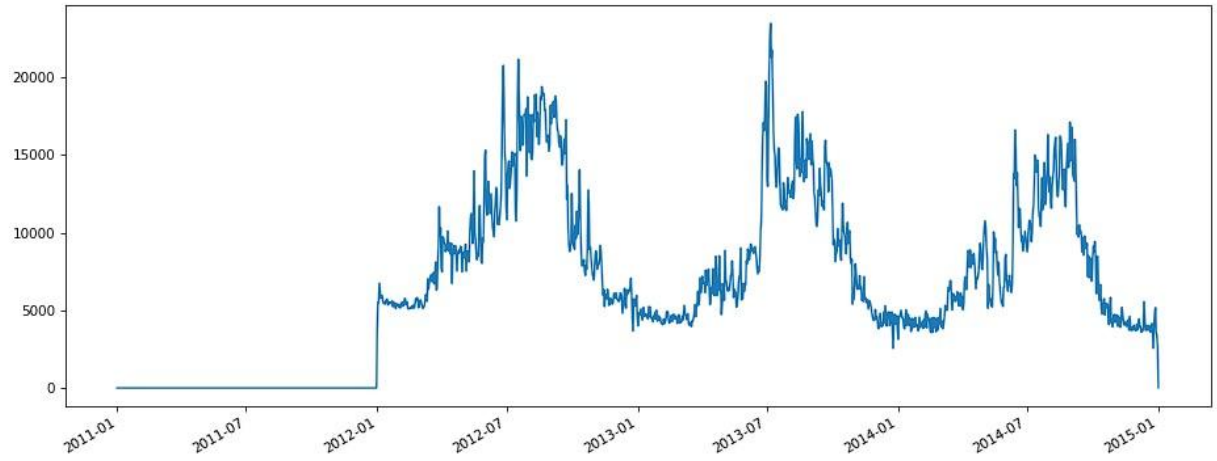




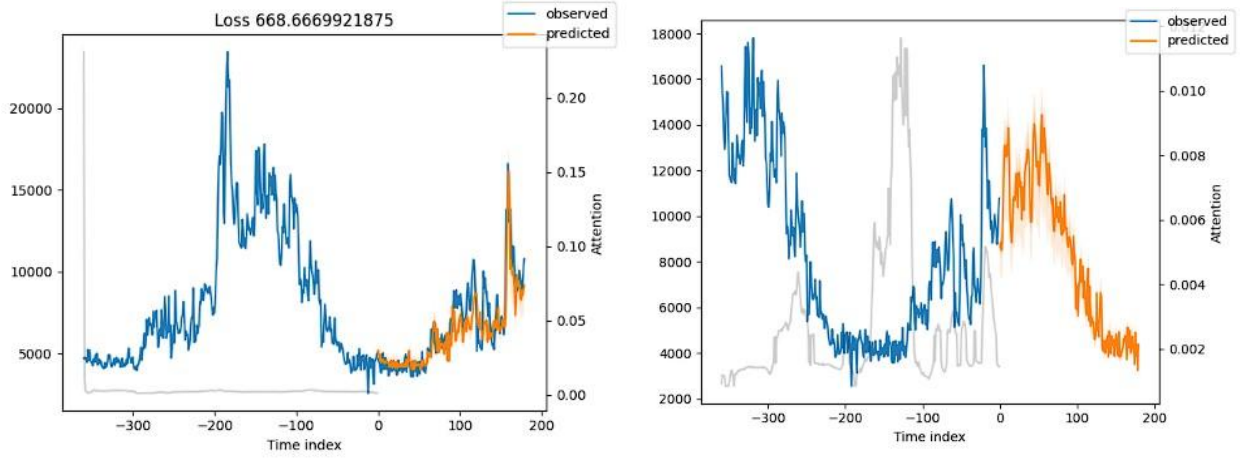
The graphs below demonstrate DeepAR's predictions over the test set and out-of-sample. They show that DeepAR captures the periodicity of Cluster 8's data and there is a lot of room for improvement.



Cluster 9

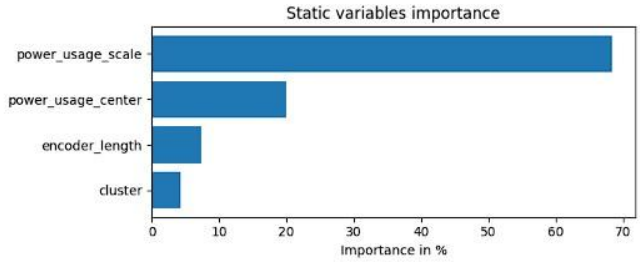


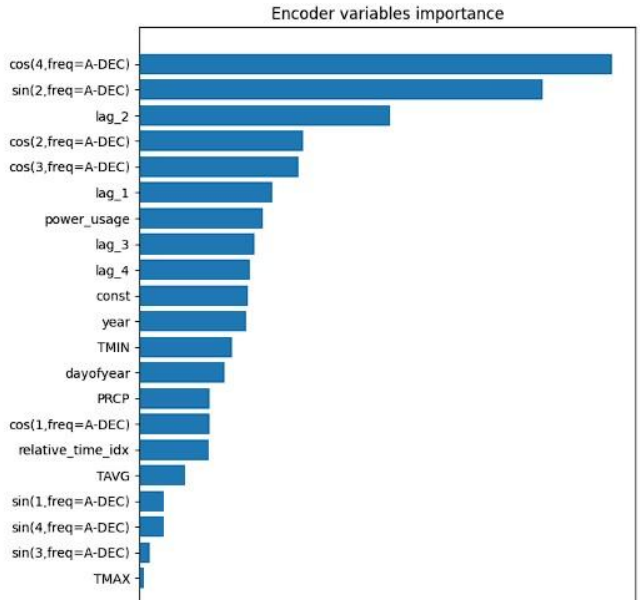
Above is the plot of Cluster 9's data. The scale of the data is moderate. There is no electricity consumption in the first year. There is a very minimal downward trend but strong annual seasonality. Daily power usage fluctuates frequently and drastically. In each year, data peaks during summer and reaches the valley low during winter.

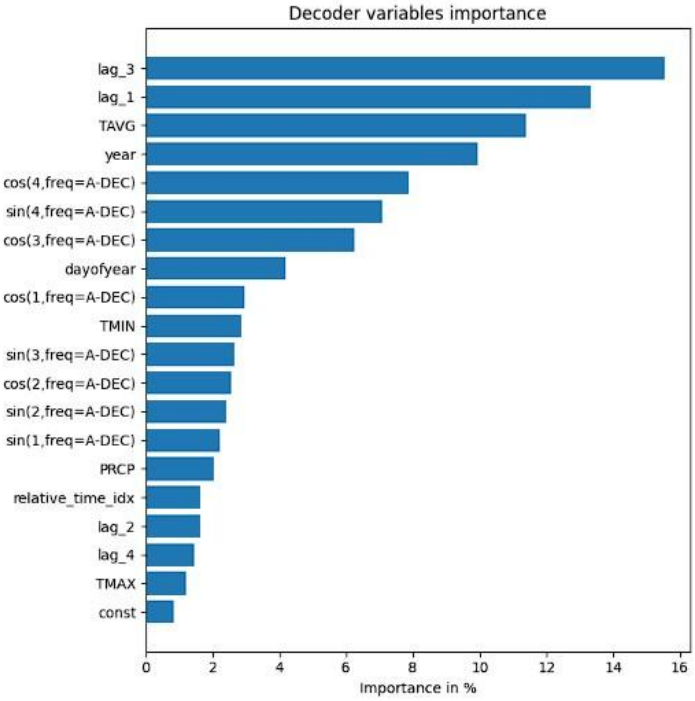


The diagrams above are the TFT's predictions over the validation and test dataset. This model grasps trends and seasonality of historical data perfectly.

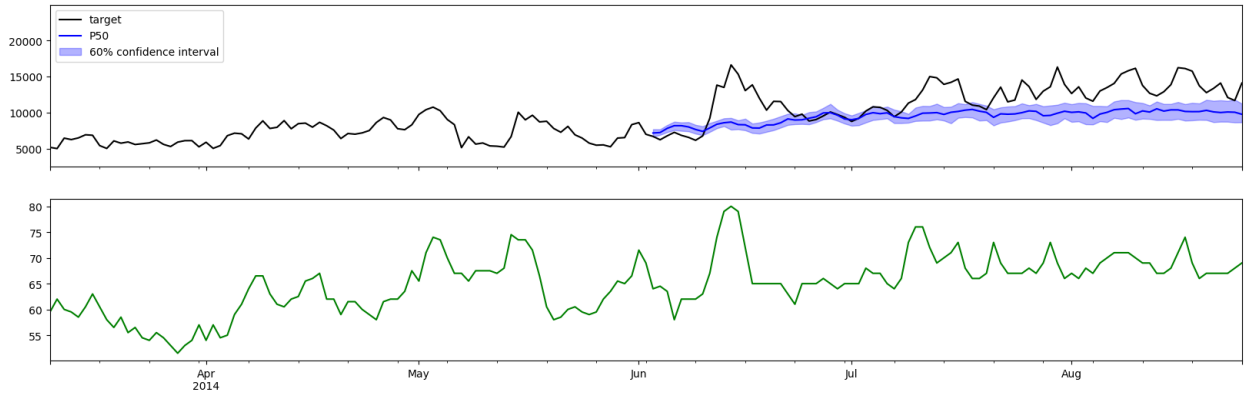
The figures below are feature importances estimated by TFT.

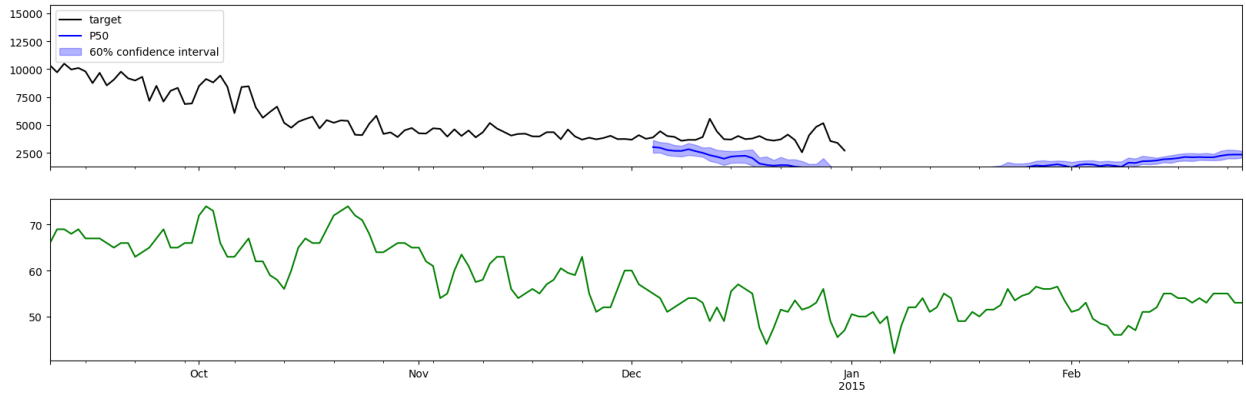

Static variables importance
power_usage_scale
power_usage_center
encoder_length
cluster
0
10
20
30
40
50
60
70
Importance in %


Encoder variables importance
cos(4,freq=A-DEC)
sin(2,freq=A-DEC)
lag_2
cos(2,freq=A-DEC)
cos(3,freq=A-DEC)
lag_1
power_usage
lag_3
lag_4
const
year
TMIN
dayofyear
PRCP
cos(1,freq=A-DEC)
relative_time_idx
TAVG
sin(1,freq=A-DEC)
sin(4,freq=A-DEC)
sin(3,freq=A-DEC)
TMAX

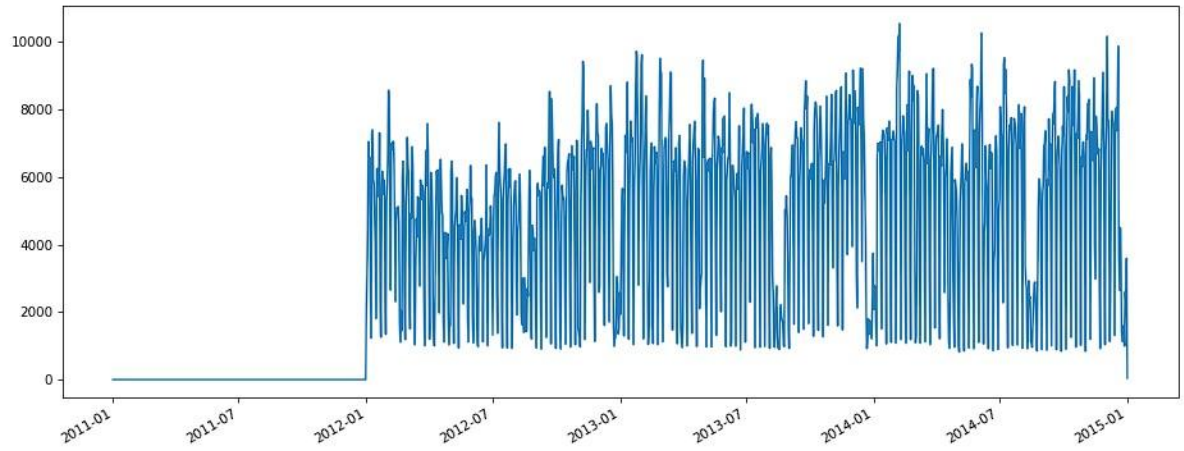

Decoder variables importance
lag_3
lag_1
TAVG
year
cos(4,freq=A-DEC)
sin(4,freq=A-DEC)
cos(3,freq=A-DEC)
dayofyear
cos(1,freq=A-DEC)
TMIN
sin(3,freq=A-DEC)
cos(2,freq=A-DEC)
sin(2,freq=A-DEC)
sin(1,freq=A-DEC)
PRCP
relative_time_idx
lag_2
lag_4
TMAX
const
0
2
4
6
8
10
12
14
16
Importance in %

The graphs below demonstrate DeepAR's predictions over the test set and out-of-sample. They show that DeepAR captures the periodic trend of Cluster 9's data and there is a lot of room for improvement.

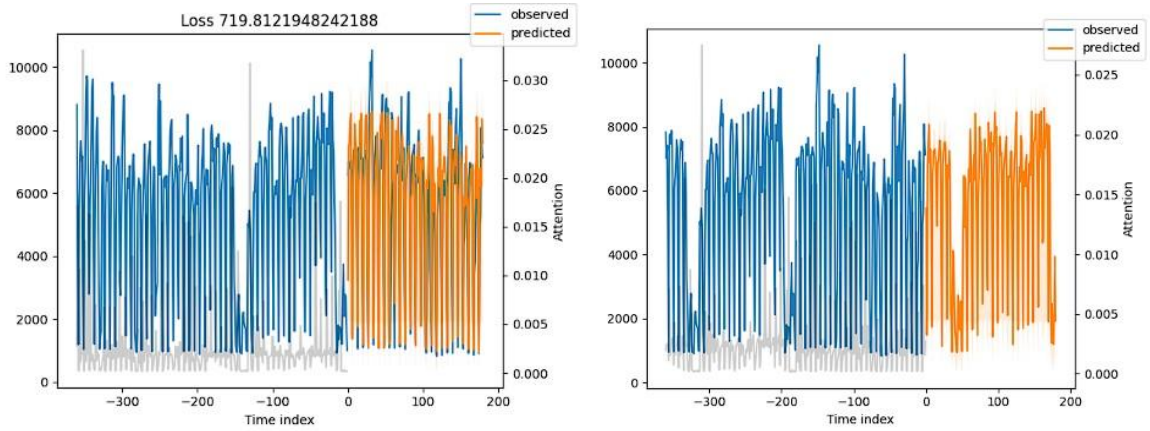




Cluster 10

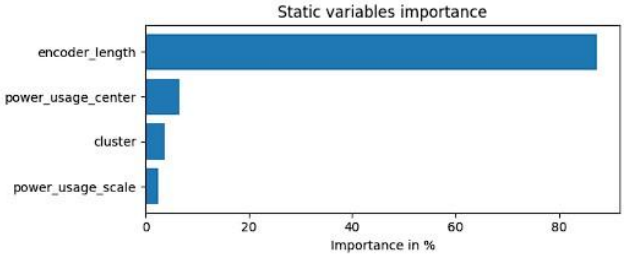


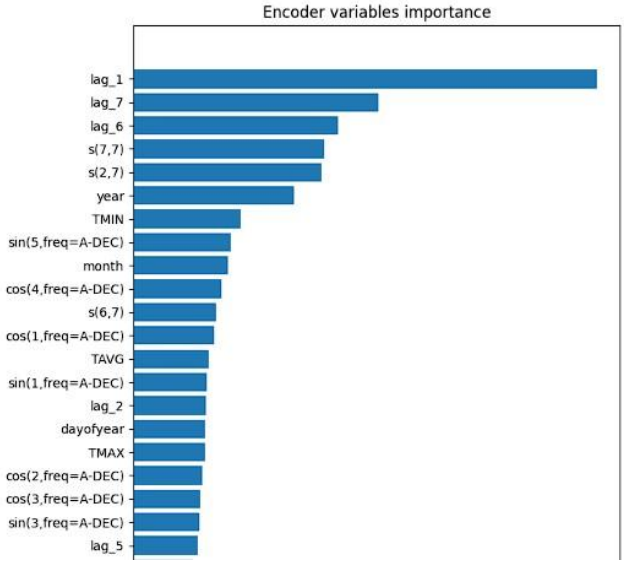
Above is the plot of Cluster 10's data. The scale of the data is relatively small. There is no electricity consumption in the first year. There is neither an obvious trend nor seasonality. Daily power usage fluctuates very frequently and drastically.

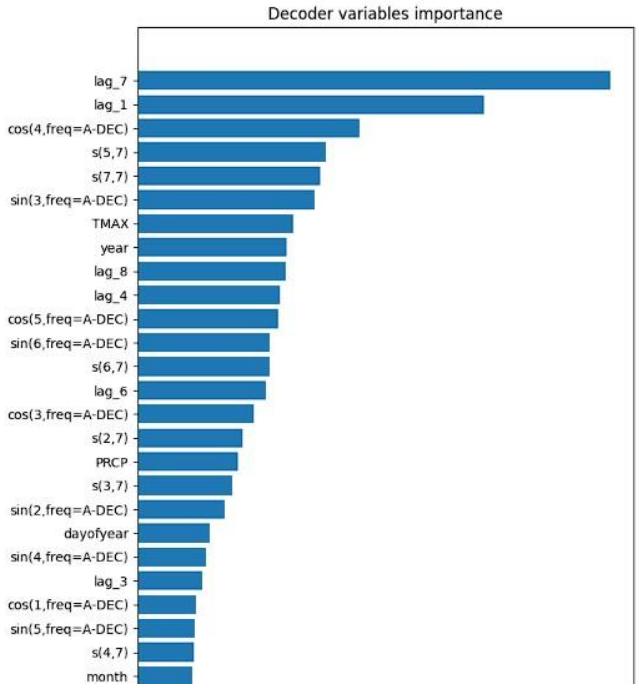


The plots above are the TFT's predictions over the validation and test dataset. This model mimics the characteristics of the historical data. Still, ups and downs in the original data are changed too drastically and frequently to learn comprehensively.

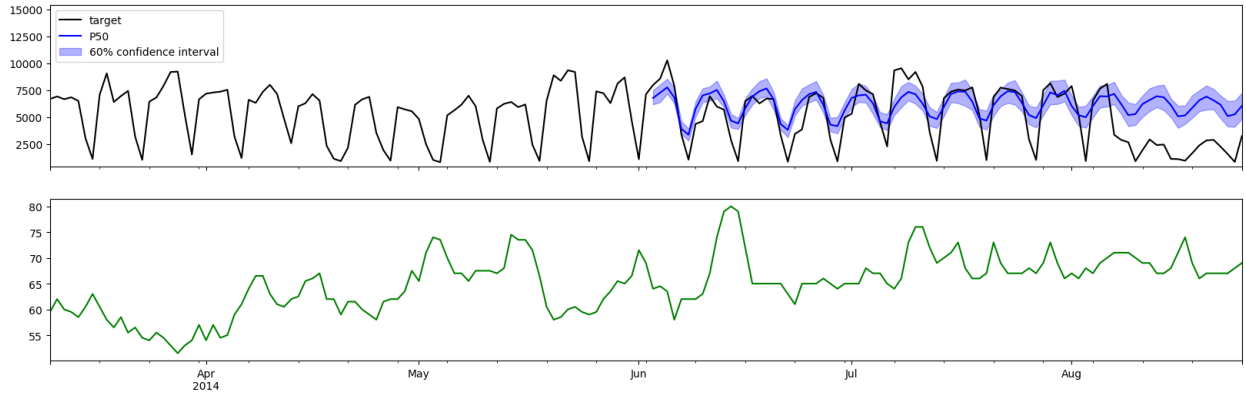
The figures below are feature importances estimated by TFT.

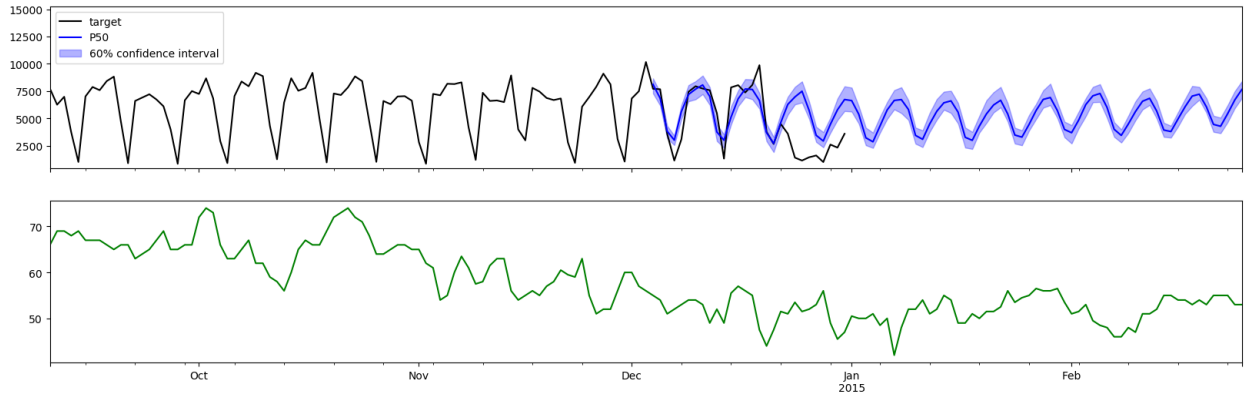

Static variables importance
encoder_length
power_usage_center
cluster
power_usage_scale
0
20
40
60
80
Importance in %


Encoder variables importance
lag_1
lag_7
lag_6
s(7,7)
s(2,7)
year
TMIN
sin(5,freq=A-DEC)
month
cos(4,freq=A-DEC)
s(6,7)
cos(1,freq=A-DEC)
TAVG
sin(1,freq=A-DEC)
lag_2
dayofyear
TMAX
cos(2,freq=A-DEC)
cos(3,freq=A-DEC)
sin(3,freq=A-DEC)
lag_5

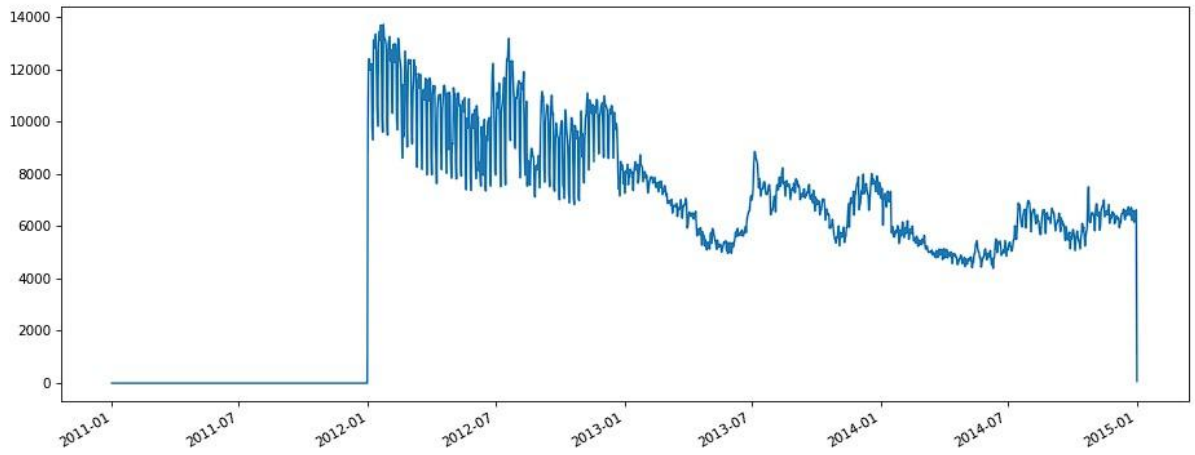

Decoder variables importance
lag_7
lag_1
cos(4,freq=A-DEC)
s(5,7)
s(7,7)
sin(3,freq=A-DEC)
TMAX
year
lag_8
lag_4
cos(5,freq=A-DEC)
sin(6,freq=A-DEC)
s(6,7)
lag_6
cos(3,freq=A-DEC)
s(2,7)
PRCP
s(3,7)
sin(2,freq=A-DEC)
dayofyear
sin(4,freq=A-DEC)
lag_3
cos(1,freq=A-DEC)
sin(5,freq=A-DEC)
s(4,7)
month

The plots below demonstrate DeepAR's predictions over the test set and out-of-sample. They show that DeepAR captures the periodic trend of Cluster 10's data.

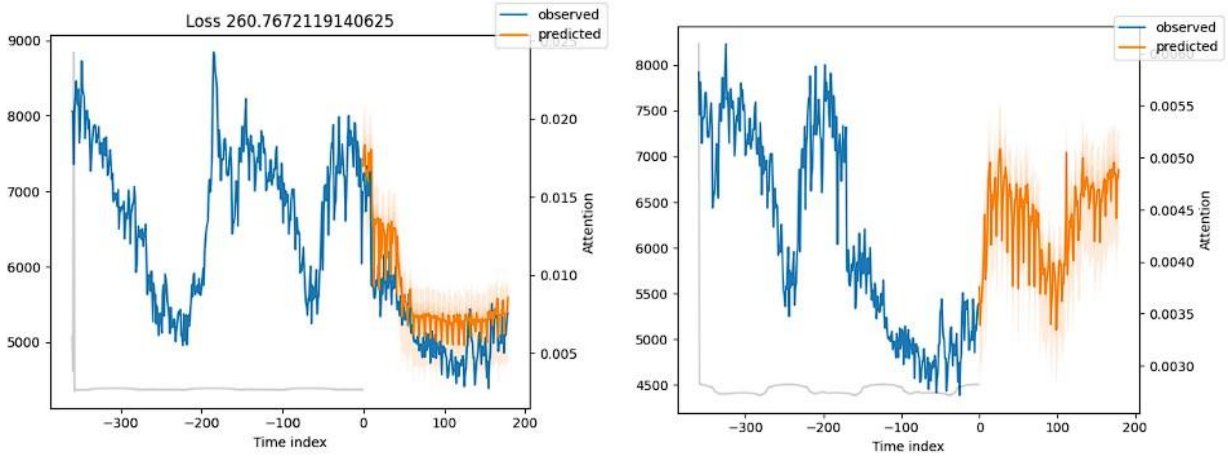




Cluster 11

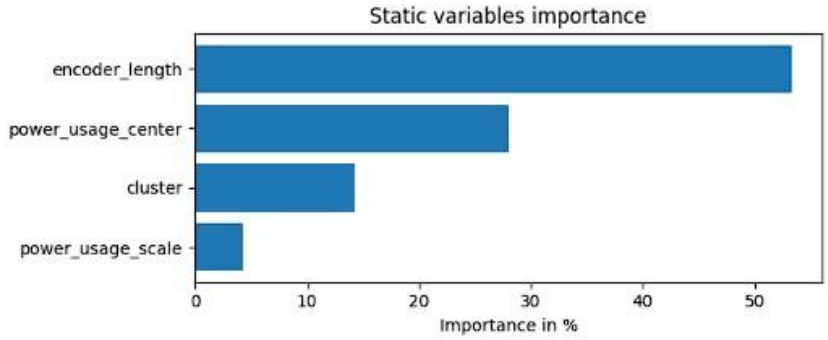


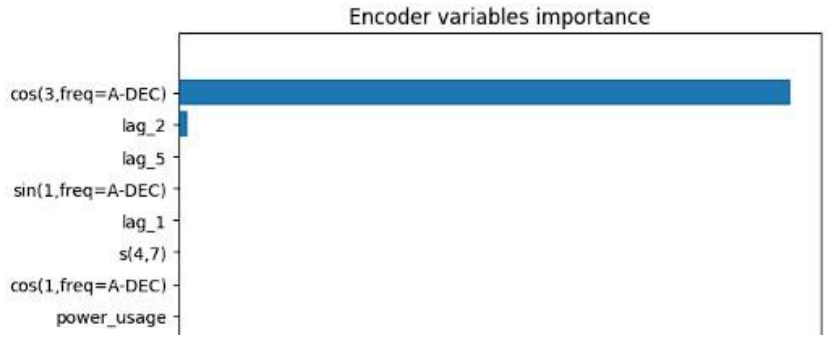
Above is the plot of Cluster 11's data. The scale of the data is moderate. There is no electricity consumption in the first year. There is a downward trend and annual seasonality. Daily power usage fluctuates drastically from 2012-01 to 2013-01. For other periods of time, data changes gradually. In each year, data peaks during summer and reaches the valley low during winter.

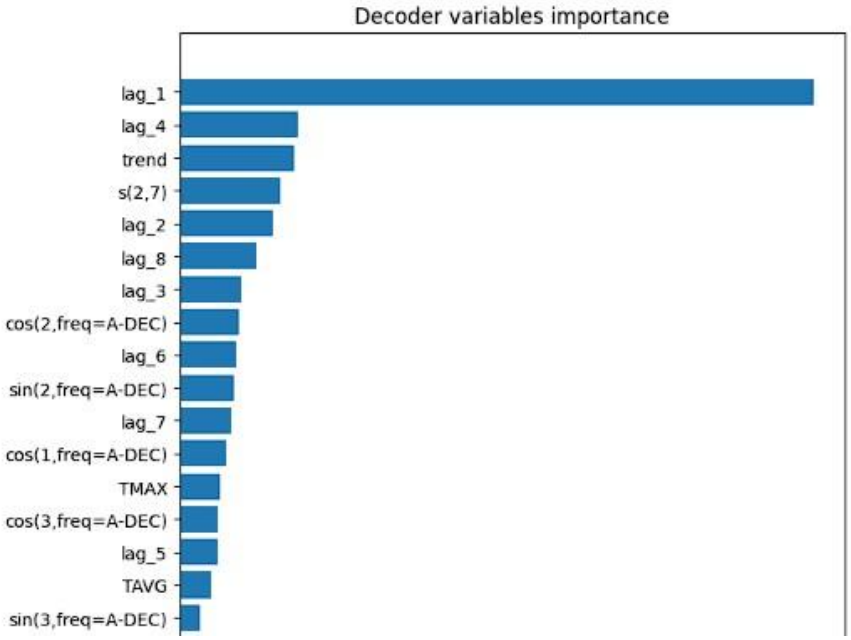


The diagrams above are the TFT's predictions over the validation and test dataset. This model grasps trends and seasonality of historical data perfectly. It is dependable for forecasting.

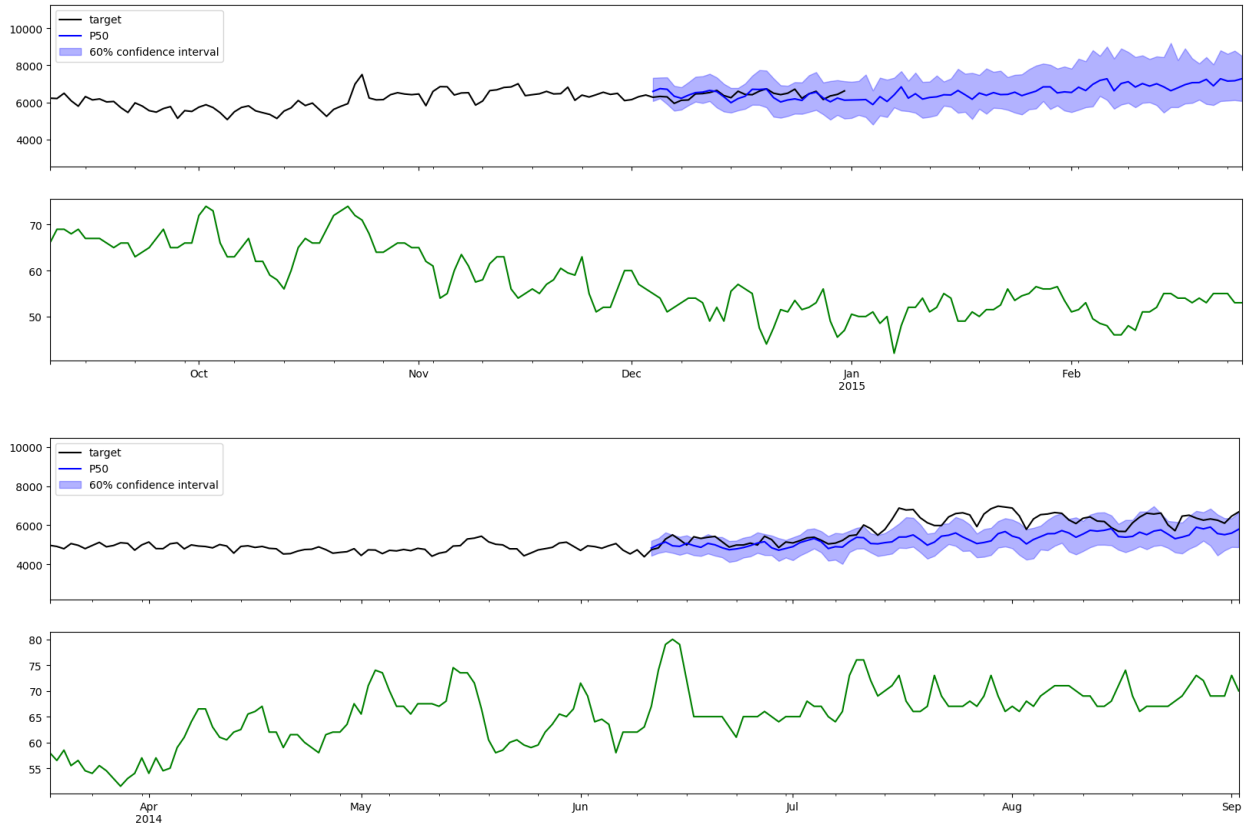
The figures below are feature importances estimated by TFT.



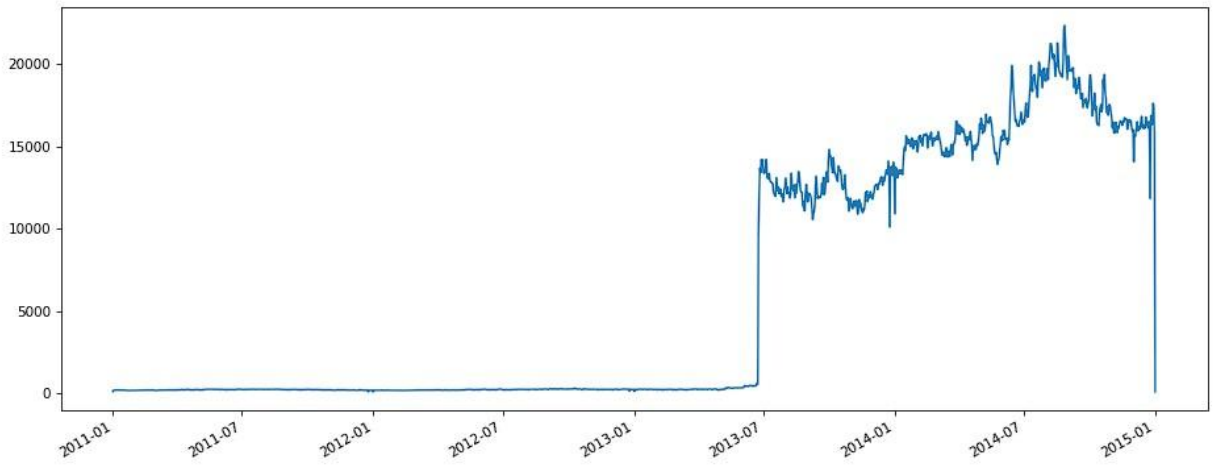




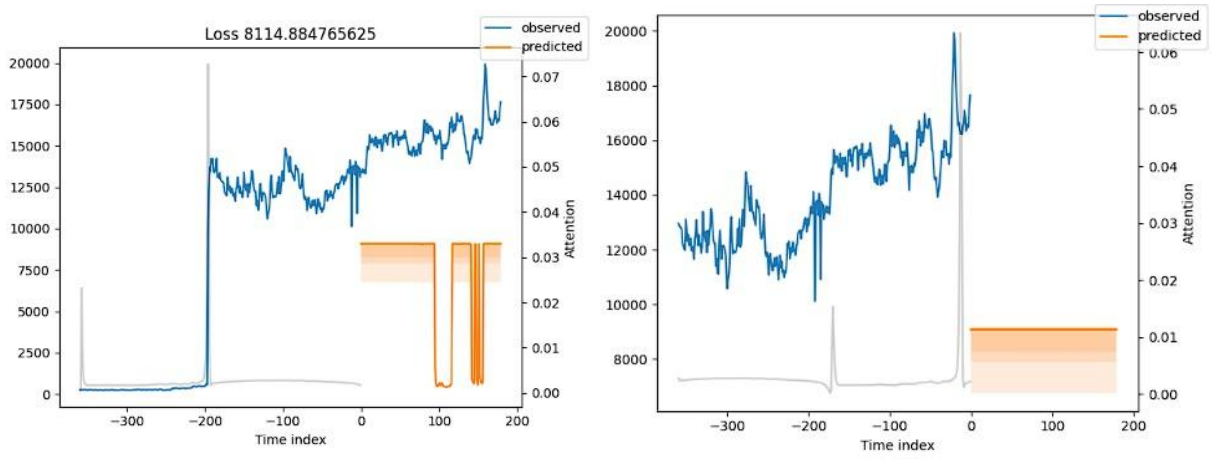
The plots below demonstrate DeepAR's predictions over the test set and out-of-sample. They show that DeepAR captures the trend and seasonality of Cluster 11's data well and can make reliable predictions.



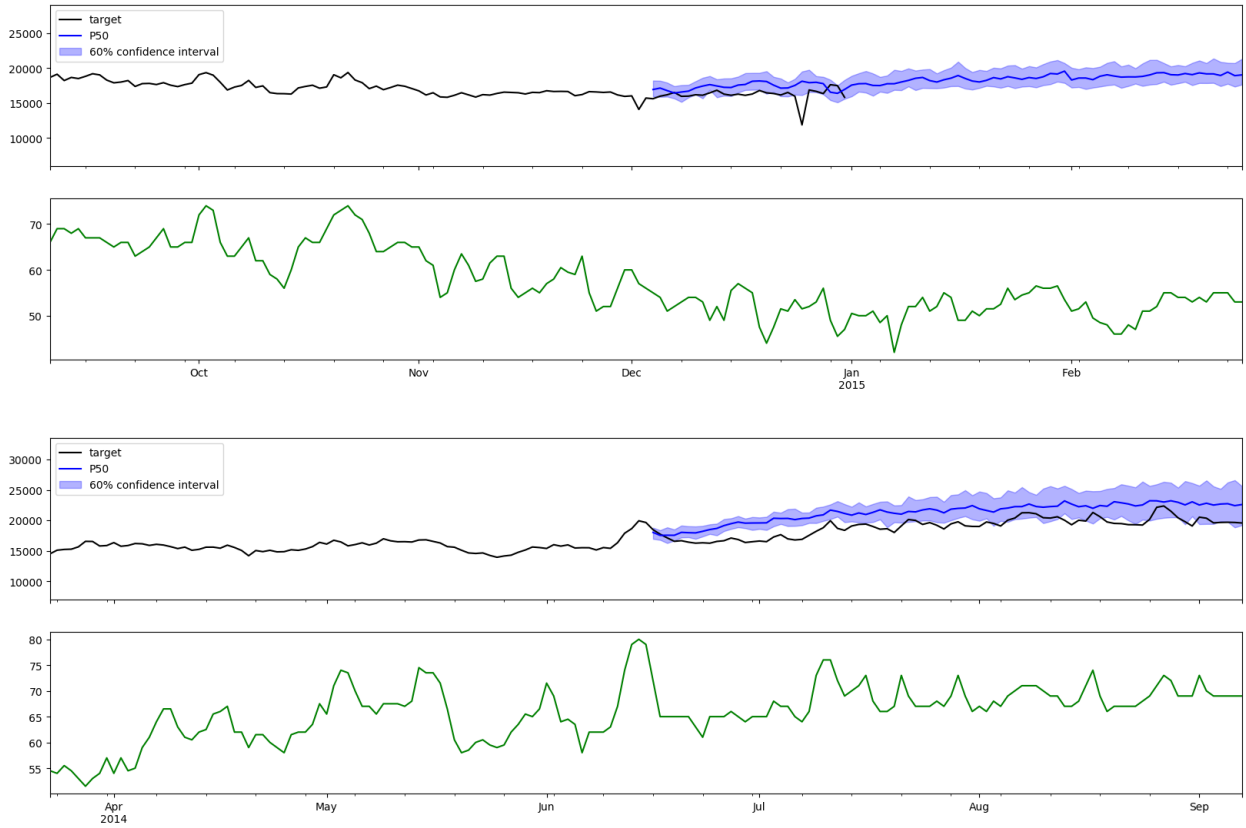
Cluster 12



Above is the plot of Cluster 12's data. The scale of the data is moderate. There is no electricity consumption in the two years. There is an up-then-down trend and no obvious annual seasonality. Daily power usage fluctuates mildly.

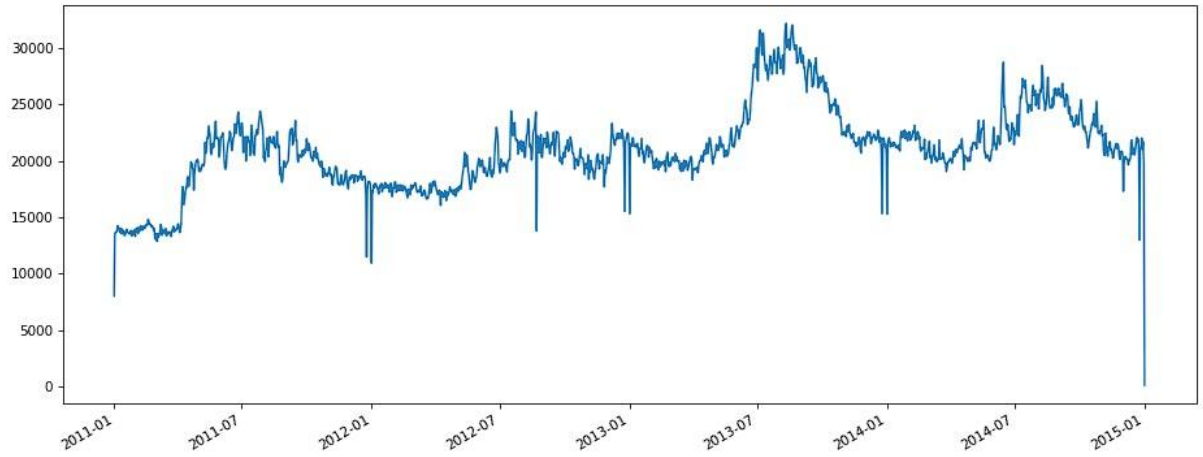


The diagrams above are the TFT's predictions over the validation and test dataset. This model fails to understand trends and seasonality of historical data. It is not suitable for forecasting future time series data.

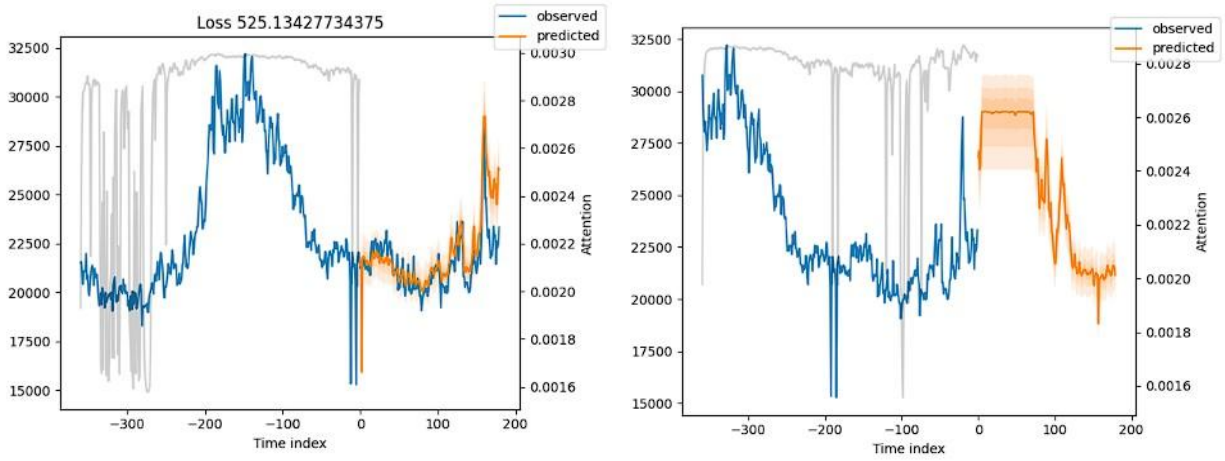


The plots above demonstrate DeepAR's predictions over the test set and out-of-sample. They show that DeepAR captures the trend and seasonality of Cluster 12's data well and can make reliable predictions.

Cluster 13

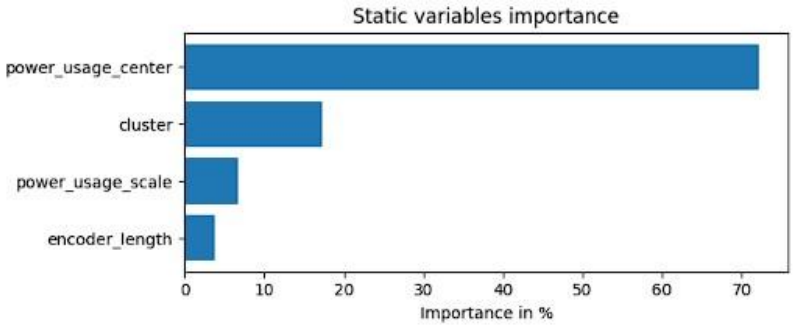


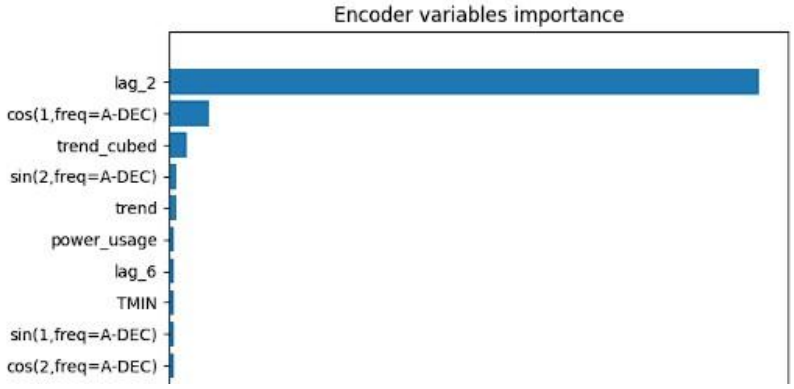
Above is the plot of Cluster 13's data. The scale of the data is moderate. There is a minimal upward trend but strong annual seasonality. Daily power usage fluctuates frequently but not drastically. In each year, data peaks during summer and reaches the valley low in January.

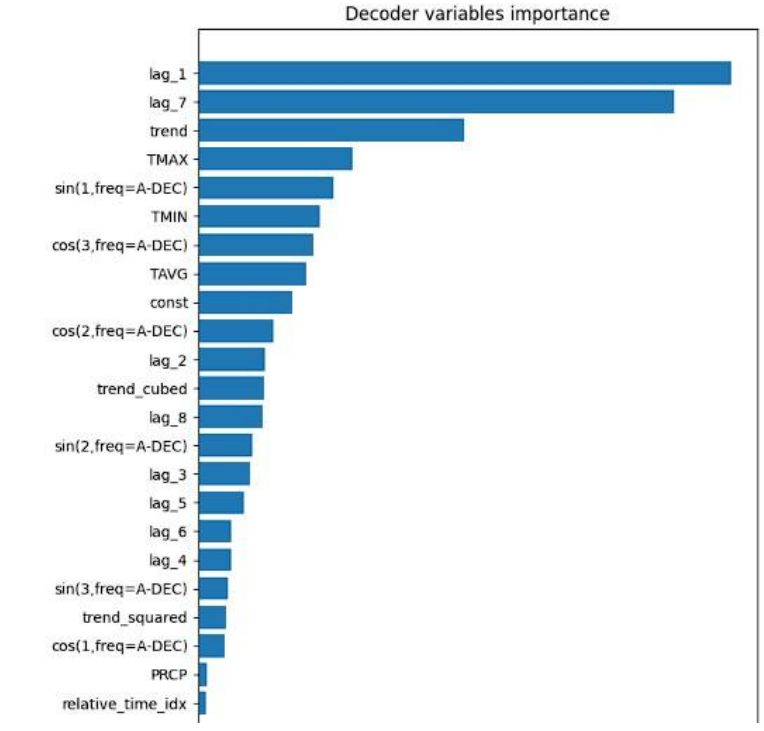


The diagrams above are the TFT's predictions over the validation and test dataset. This model mimics the trend and seasonality of historical data precisely. It is dependable for forecasting.

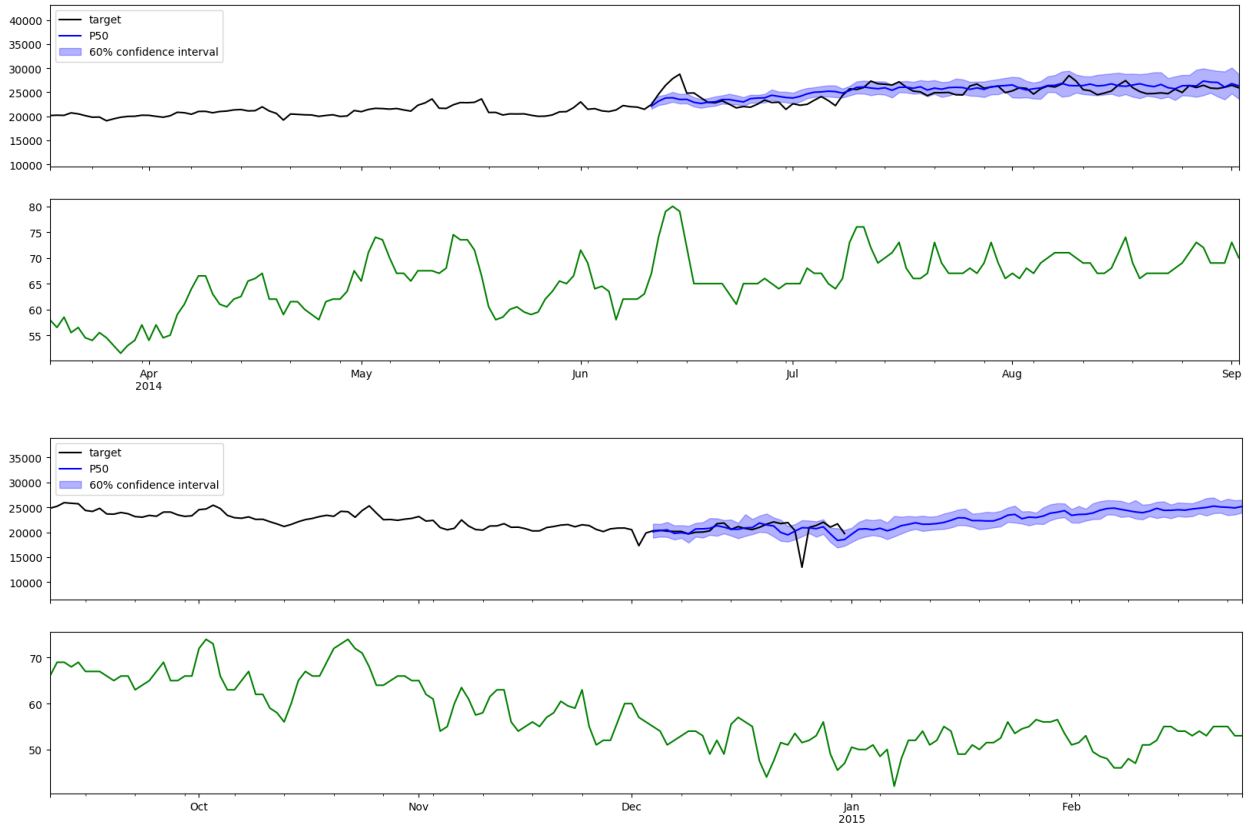
The figures below are feature importances estimated by TFT.


Static variables importance
power_usage_center
cluster
power_usage_scale
encoder_length
0
10
20
30
40
50
60
70
Importance in %

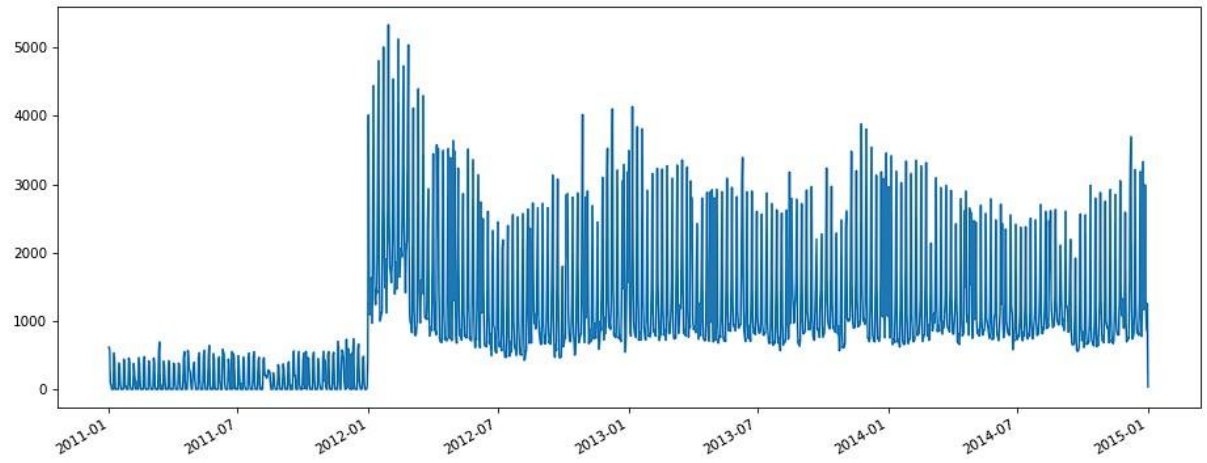

Encoder variables importance
lag_2
cos(1,freq=A-DEC)
trend_cubed
sin(2,freq=A-DEC)
trend
power_usage
lag_6
TMIN
sin(1,freq=A-DEC)
cos(2,freq=A-DEC)


Decoder variables importance
lag_1
lag_7
trend
TMAX
sin(1,freq=A-DEC)
TMIN
cos(3,freq=A-DEC)
TAVG
const
cos(2,freq=A-DEC)
lag_2
trend_cubed
lag_8
sin(2,freq=A-DEC)
lag_3
lag_5
lag_6
lag_4
sin(3,freq=A-DEC)
trend_squared
cos(1,freq=A-DEC)
PRCP
relative_time_idx

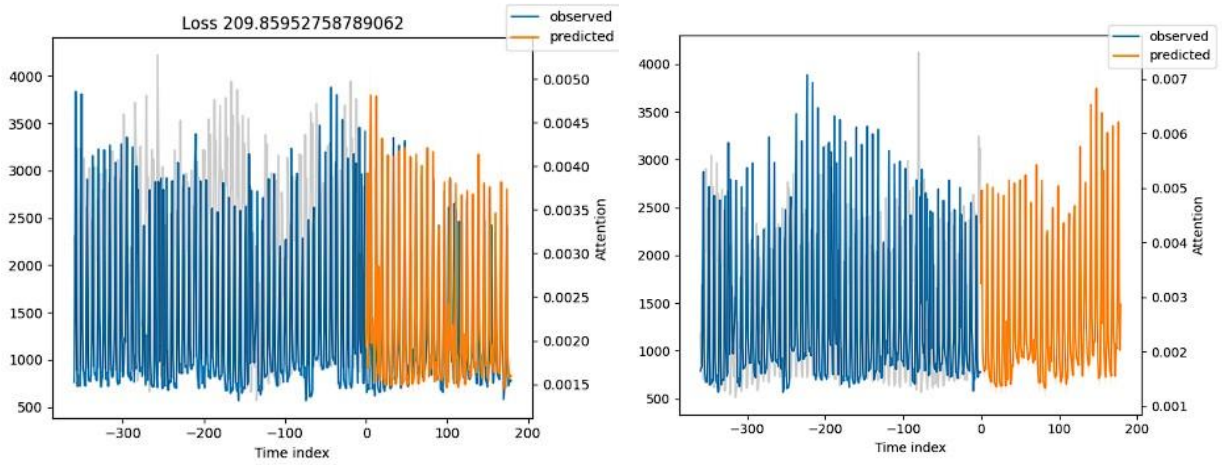
The plots below demonstrate DeepAR's predictions over the test set and out-of-sample. They show that DeepAR captures the trend and seasonality of Cluster 13's data well and can make reliable predictions.



Cluster 14

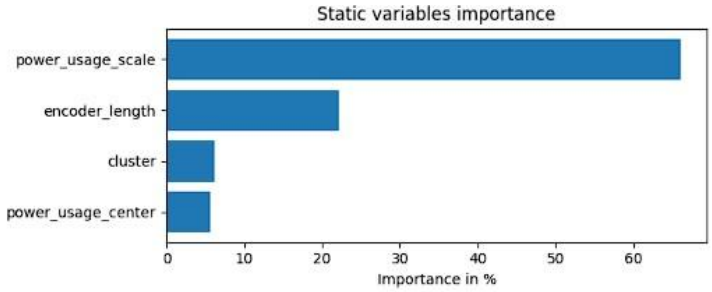


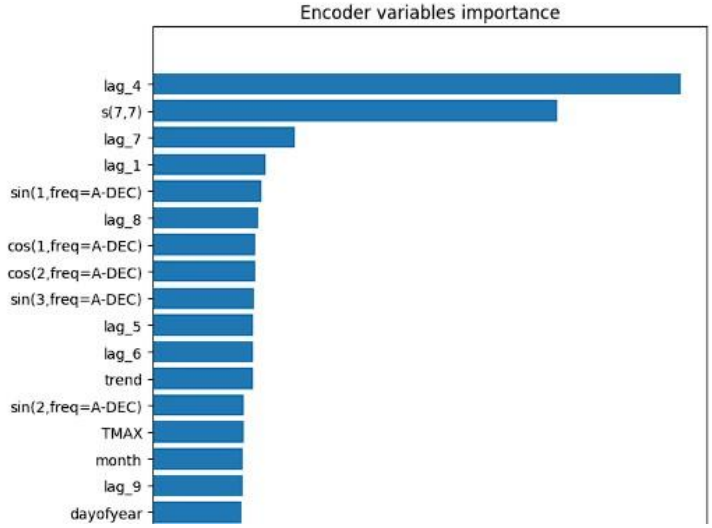
Above is the plot of Cluster 14's data. The scale of the data is tiny. There is no trend or seasonality. Daily power usage fluctuates very frequently and drastically.

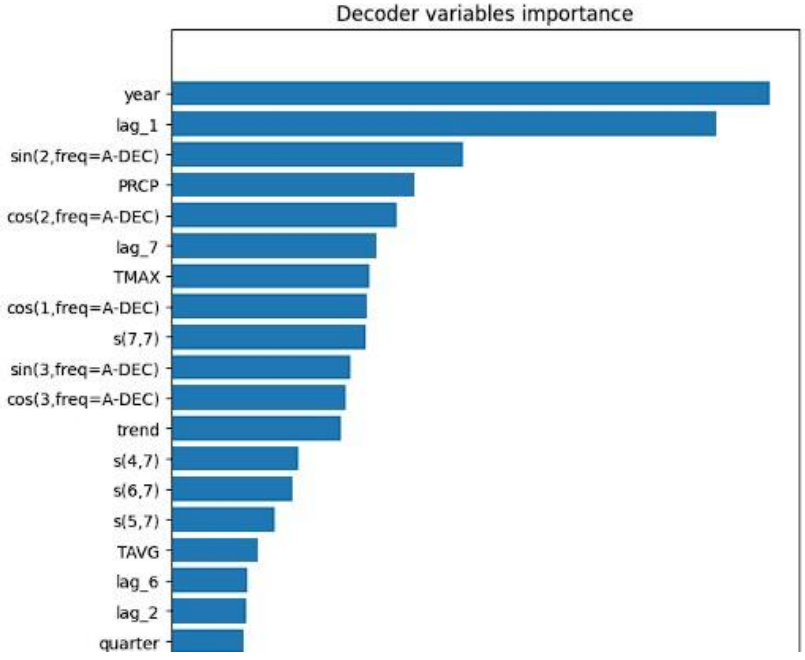


The plots above are the TFT's predictions over the validation and test dataset. This model mimics the characteristics of the historical data. Still, ups and downs in the original data are changed too drastically and frequently to learn comprehensively.

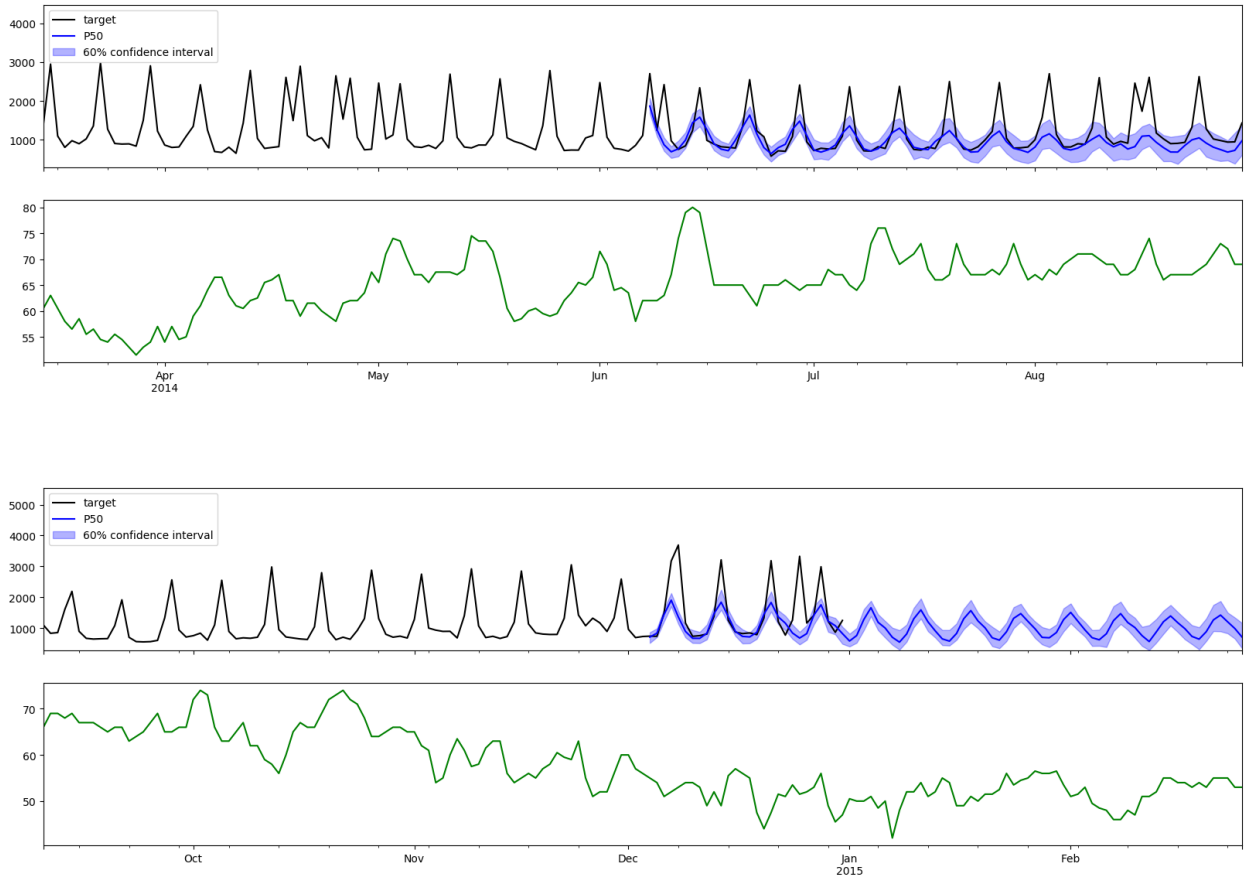
The figures below are feature importances estimated by TFT.


Static variables importance
power_usage_scale
encoder_length
cluster
power_usage_center
0
10
20
30
40
50
60
Importance in %


Encoder variables importance
lag_4
s(7,7)
lag_7
lag_1
sin(1,freq=A-DEC)
lag_8
cos(1,freq=A-DEC)
cos(2,freq=A-DEC)
sin(3,freq=A-DEC)
lag_5
lag_6
trend
sin(2,freq=A-DEC)
TMAX
month
lag_9
dayofyear


Decoder variables importance
year
lag_1
sin(2,freq=A-DEC)
PRCP
cos(2,freq=A-DEC)
lag_7
TMAX
cos(1,freq=A-DEC)
s(7,7)
sin(3,freq=A-DEC)
cos(3,freq=A-DEC)
trend
s(4,7)
s(6,7)
s(5,7)
TAVG
lag_6
lag_2
quarter

The plots below demonstrate DeepAR's predictions over the test set and out-of-sample. They show that DeepAR captures the trend and seasonality of Cluster 14's data decently and there is still a lot of room for improvement.



Next Step

Hyperparameter tuning is definitely worthy of further study. It would not only potentially improve the accuracy of predictions, but speed up the training of deep learning models.

Try to work on a one-fit-all TFT model instead of multiple one-to-one TFT models. It would be rather convenient to tune hyperparameters and make comparisons among different models' performances. It would also be interesting to study the importance of feature engineering to TFT in the context of forecasting time series data.