Day-ahead Electricity Demand Prediction Project: Milestone 3

Boylan Pardosi

1. Objective

The objective of this project is to provide day ahead (48 half-hourly) predictions of electricity demand using historical/today's measurements and weather data in Auckland City, represented by a substation.

2. Data Source

For the historical electricity demand, we use data provided by the NZ electricity via its EMI website (https://www.emi.ea.govt.nz/Wholesale/Datasets). The data storage contains daily demand for all connected substations in NZ, each day stored in one csv file. So we need to access each of the files and take one row which represents the substation. We are looking at 3 years of data for training, testing and validation, equivalently 365 x 3 csv files. To retrieve the wanted information there are two choices: use cloud based service such as azure data factory or download all csv using azure storage explorer and access each csv locally. For this project we will be using azure data factory for a fast and less resource consuming data retrieval.

For the weather data we can do GUI-based data query for Auckland region in here: https://www.visualcrossing.com/weather/weather-data-services/auckland/metric.

3. Data Processing

We already have demand dataframe with the following information (code 3.2):

```
## 'data.frame':    52566 obs. of  4 variables:
##  $ X   : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ Date: chr  "1/1/2021" "1/1/2021" "1/1/2021" "1/1/2021" ...
##  $ TP  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Load: num  23.4 23.4 22.6 22.1 22.2 ...
```

In the prediction model we will be using various features related to the date: month and day of week. The time/hour of the observation is already represented by trading period (TP) column. Each date have 48 TP/half hourly observation (code 3.3).

In demand prediction we also need a column to indicate if the date is a holiday. For this, I have seperately created a holiday dataframe taking holiday information between 2021 and 2023 from here: https://publicholiday.co.nz/. We load the holiday dataframe to merge with the main dataframe (code 3.4).

Now we have the following table :

Date	TP	Load	Month	Day_of_week	Is_holiday
2021-01-01	1	23.419	1	6	1
2021-01-01	2	23.409	1	6	1
2021-01-01	3	22.639	1	6	1
2021-01-01	4	22.133	1	6	1
2021-01-01	5	22.240	1	6	1
2021-01-01	6	22.060	1	6	1

Then we load the weather data (temperature and humidity) and join the raw dataframe by Date column. Since the weather have datetime column we separate the date and time and map the time to TP. Also since the weather only recorded hourly, we insert in between rows averaging the measurements, this is a reasonable way of filling the gap as those measurements will not suddenly increase/decrease significantly within an hour (code 3.5).

Date	time	temp	humidity
Length:26277	Length:26277	Min. : 2.80	Min. : 28.16
Class :character	Class :character	1st Qu.:13.10	1st Qu.: 71.51
Mode :character	Mode :character	Median :16.10	Median : 82.15
NA	NA	Mean :16.16	Mean : 80.24
NA	NA	3rd Qu.:19.00	3rd Qu.: 91.57
NA	NA	Max. :28.80	Max. :100.00

Insert rows to make halfhourly observations (code 3.6).

Create a function that replace NA with average from above and below row (code 3.7)

Now the weather table become

```r
knitr::kable(head(weather_df))
```

Date	time	temp	humidity	TP
2021-01-01	00:00:00	16.00	86.880	1
2021-01-01	00:30:00	16.20	88.450	2
2021-01-01	01:00:00	16.40	90.020	3
2021-01-01	01:30:00	16.05	90.020	4
2021-01-01	02:00:00	15.70	90.020	5
2021-01-01	02:30:00	15.75	90.505	6

Now join with the min dataframe by 'Date' and 'TP' (code 3.8)

Date	TP	Load	Month	Day_of_week	Is_holiday	time	temp	humidity
2021-01-01	1	23.419	1	6	1	00:00:00	16.00	86.880
2021-01-01	2	23.409	1	6	1	00:30:00	16.20	88.450
2021-01-01	3	22.639	1	6	1	01:00:00	16.40	90.020
2021-01-01	4	22.133	1	6	1	01:30:00	16.05	90.020
2021-01-01	5	22.240	1	6	1	02:00:00	15.70	90.020
2021-01-01	6	22.060	1	6	1	02:30:00	15.75	90.505

Load	temp	humidity
Min. : 0.001	Min. : 2.80	Min. : 28.16
1st Qu.:22.078	1st Qu.:13.15	1st Qu.: 71.43
Median :27.344	Median :16.10	Median : 82.11
Mean :28.629	Mean :16.16	Mean : 80.24
3rd Qu.:35.268	3rd Qu.:19.00	3rd Qu.: 90.96
Max. :71.433	Max. :28.80	Max. :100.00
NA's :1371	NA's :7	NA's :7

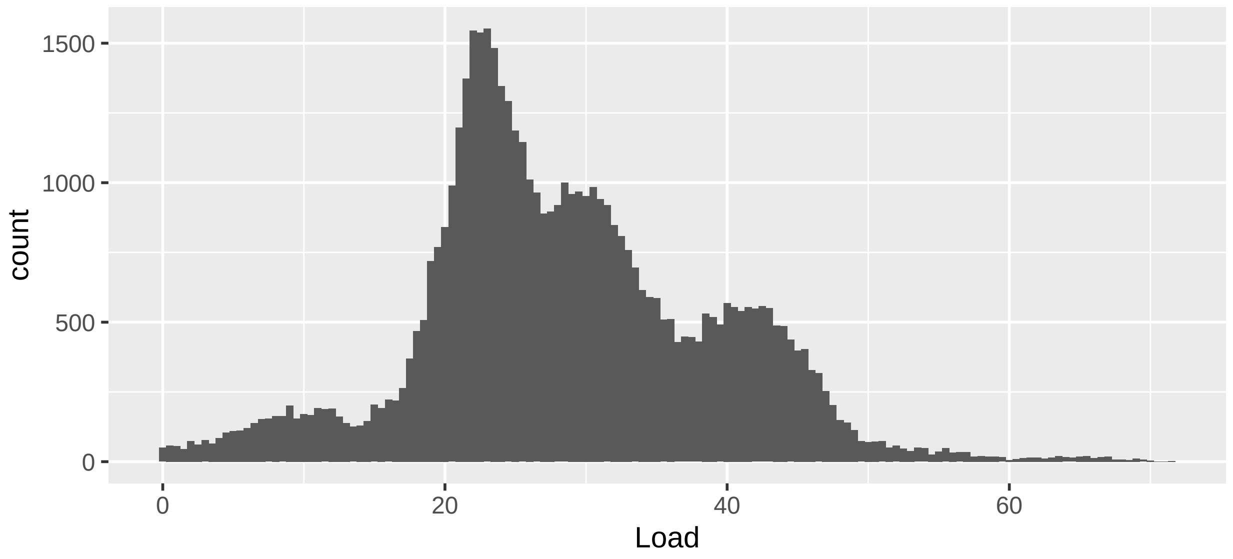
We still have NA for weather table, it happen because 2 NA occured consecutively. .Hence we will use approx from zoo to interpolate between missing data. To fill timedata we can derive from TP column (code 3.9).

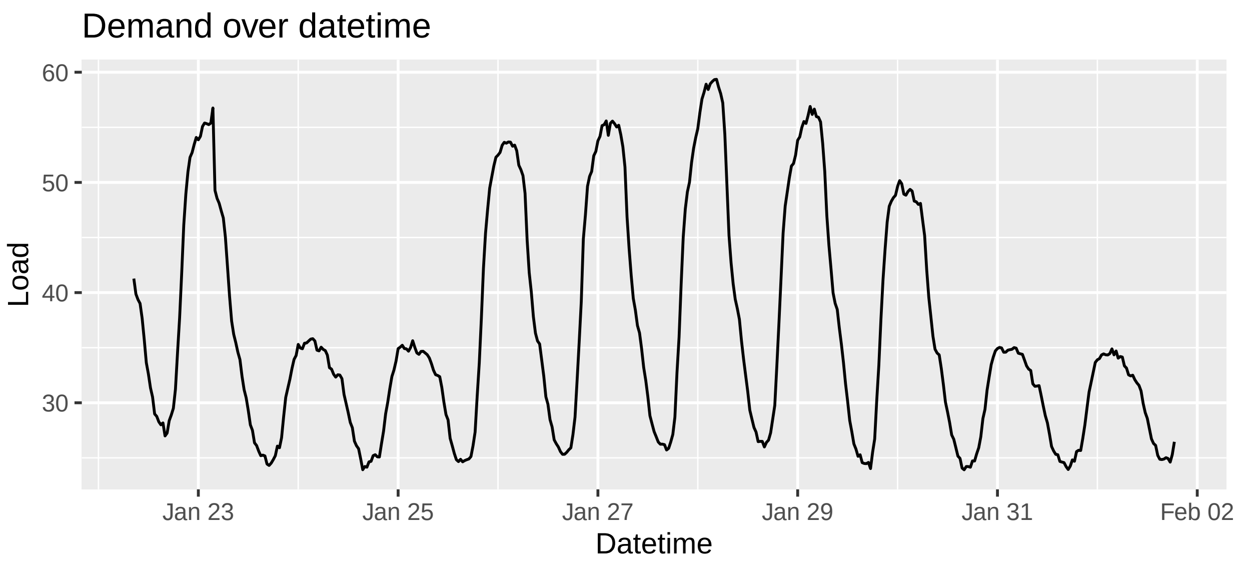
We still have 1731 NA's on demand out of 52566 = 3% of the whole observations. Since this is relatively small portion, we can do rough estimation by averaging on the same value of all features (month, day_of_week, TP, and holiday) (code 3.10).

Load	temp	humidity	Datetime
Min. : 0.001	Min. : 2.80	Min. : 28.16	Min. :2021-01-01 13:00:00
1st Qu.:22.091	1st Qu.:13.15	1st Qu.: 71.43	1st Qu.:2021-10-02 06:52:30
Median :27.363	Median :16.10	Median : 82.11	Median :2022-07-02 23:45:00
Mean :28.630	Mean :16.16	Mean : 80.24	Mean :2022-07-02 23:45:00
3rd Qu.:35.176	3rd Qu.:19.00	3rd Qu.: 90.96	3rd Qu.:2023-04-02 17:37:30
Max. :71.433	Max. :28.80	Max. :100.00	Max. :2024-01-01 12:30:00

4. Exploration

There are two main approaches for this type of prediction. First, pure time series where we only utilize previous time measurements as the predictor using selected model. Second, we use machine learning technique utilizing features such as temperature, month, day of week, time/TP as predictors. We can also combine time series analysis and additional features as explanatory variables for predictions. For time series analysis, we first explore the measurements against time and look for patterns (code 4.1).

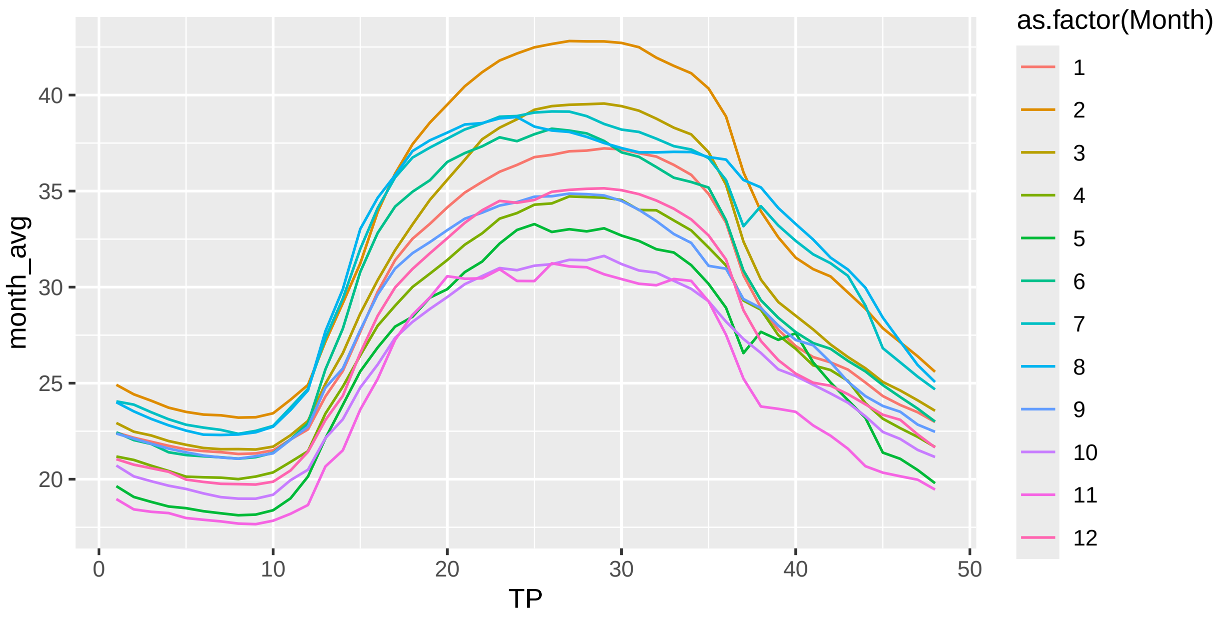




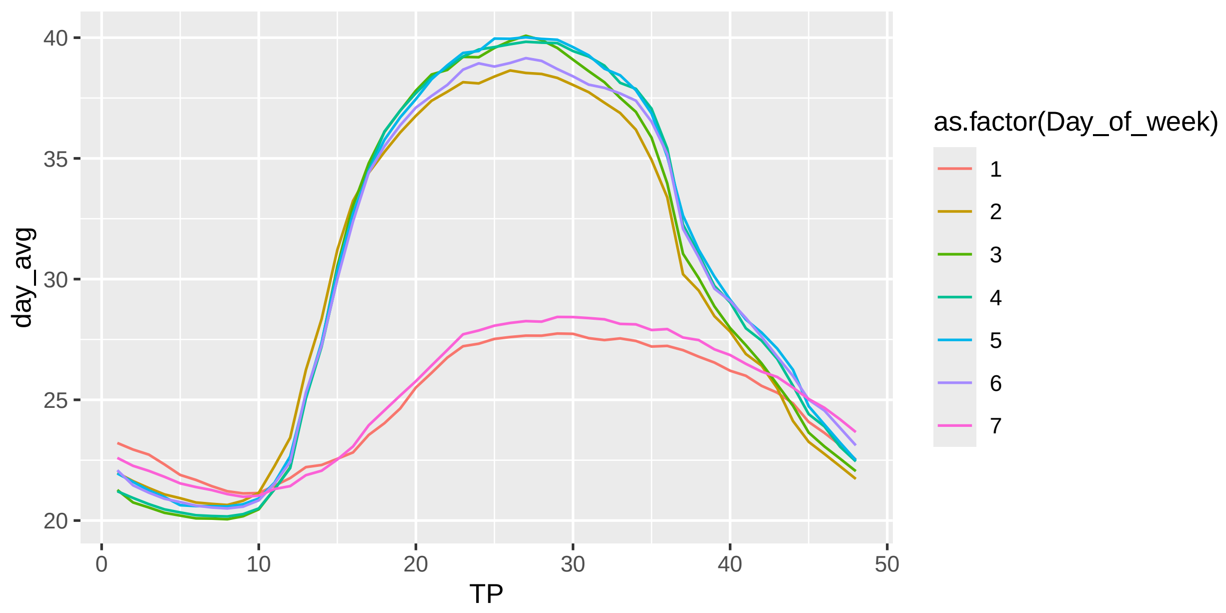
The first plot shows demand histogram which is ranging from 0 to 71 with median value of 27 MW.

From the second plot we can see two seasonal patterns. First we have daily seasonality: it has one clear peak every day. Then we have weekly pattern, two days with less peak occurred on weekends. For further time series analysis we need analyze the autocorrelation(ACF) and partial-autocorrelation (PACF) plot to help us determine the proper model.

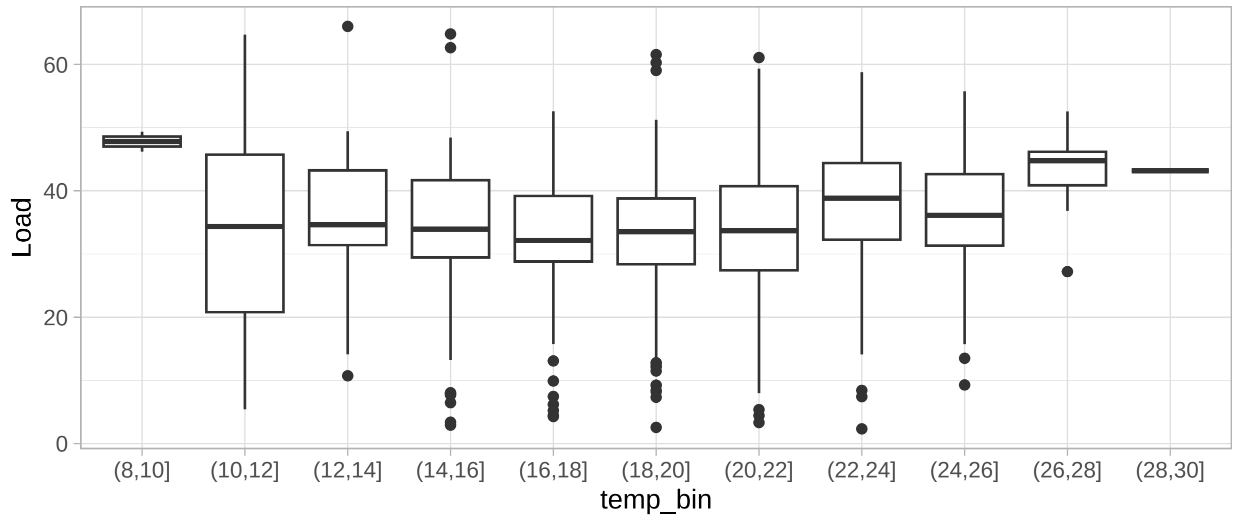
We now look at different features and see how demand is distributed along these features (code 4.2).



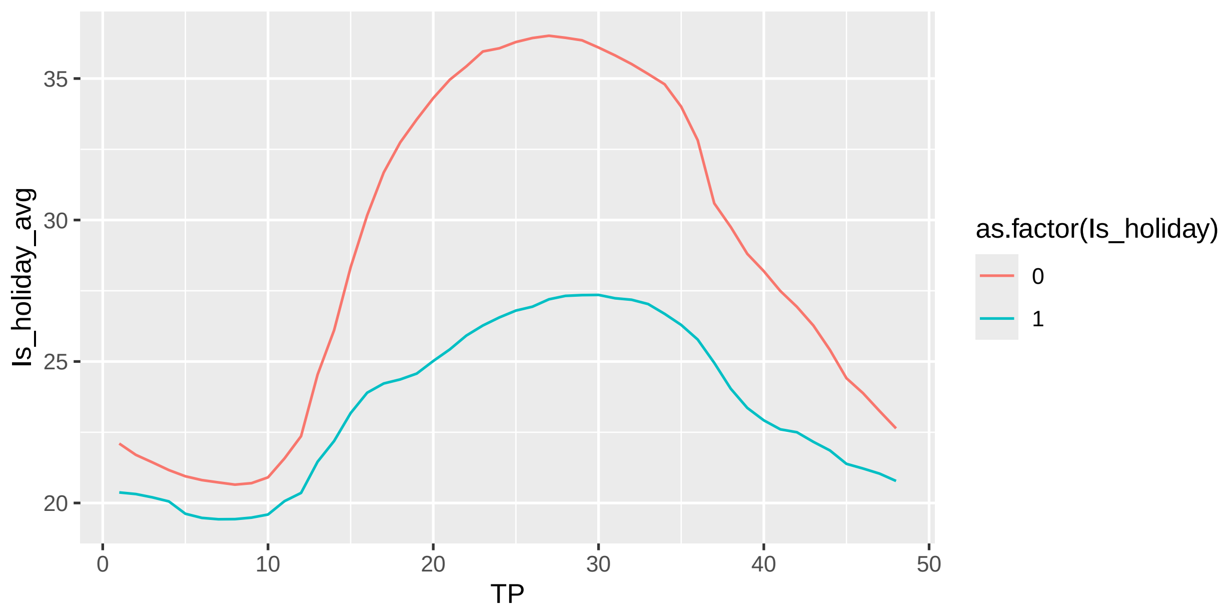
We can see that demand peaks occurred at TP 25-30. Also different month shows different average values on TP. So we can expect that month and TP are important features. We then look at day of week features (code 4.3).



It is observable that day of week 1 and 7 which are weekends have significantly lower demand. The other days seem to have similar demand. So we can also expect the weekend/weekday feature is an important predictor. Now we look at how demand distributed against temperature. We first bin the temperature to create a category feature (code 4.4).



In general, the electricity demand exhibits similar statistical patterns within the temperature range of 4 to 20 degrees Celsius. However, demand increases as temperatures rise above 20 degrees Celsius ('hot' tamperature) and decreases as temperatures fall below 4 degrees Celsius ('cold' teperature). However, we observe a significant number of outliers, indicating that temperature is not the sole influencing factor. Variables such as the trading period (TP) likely have a more substantial impact on electricity demand.



The Is_holiday feature is also important feature. As clearly shown above, mean values on holidays are siginificantly lower (code 4.5).

5. Model Development Plan

5.1. Algorithm Selection

The multi-step forecasting is basically a time-series forecasting problem, where we use the past values (time-lag values) as the predictors. However, as previously explored, we have other non time-lag features that can be important for predictions such as day of the week, month, temperature. These features cannot be included with classic time series model. Therefore we consider other algorithms that can model both time-lag and non time-lags features. There are two popular algorithms suitable for this purpose: LSTM, a RNN-based time series algorithm and XGBoost a decision tree-based algorithm that use gradient boosting for lost minimization. In addition to algorithm differences, both algorithms are differ in using inputs for the predictions.

With LSTM, each feature in the dataset is considered as a separate sequence that the model processes over time capturing temporal dependencies. In our case, this is true only for the electricity demand (dependent variable). Although the features are observed over time, we don't consider these values to be sequential. XGBoost, on the other hand, do not treat features as sequential data. In this model, We can choose time-lag features that significantly influence the future values with additional categorical and continuous predictors.

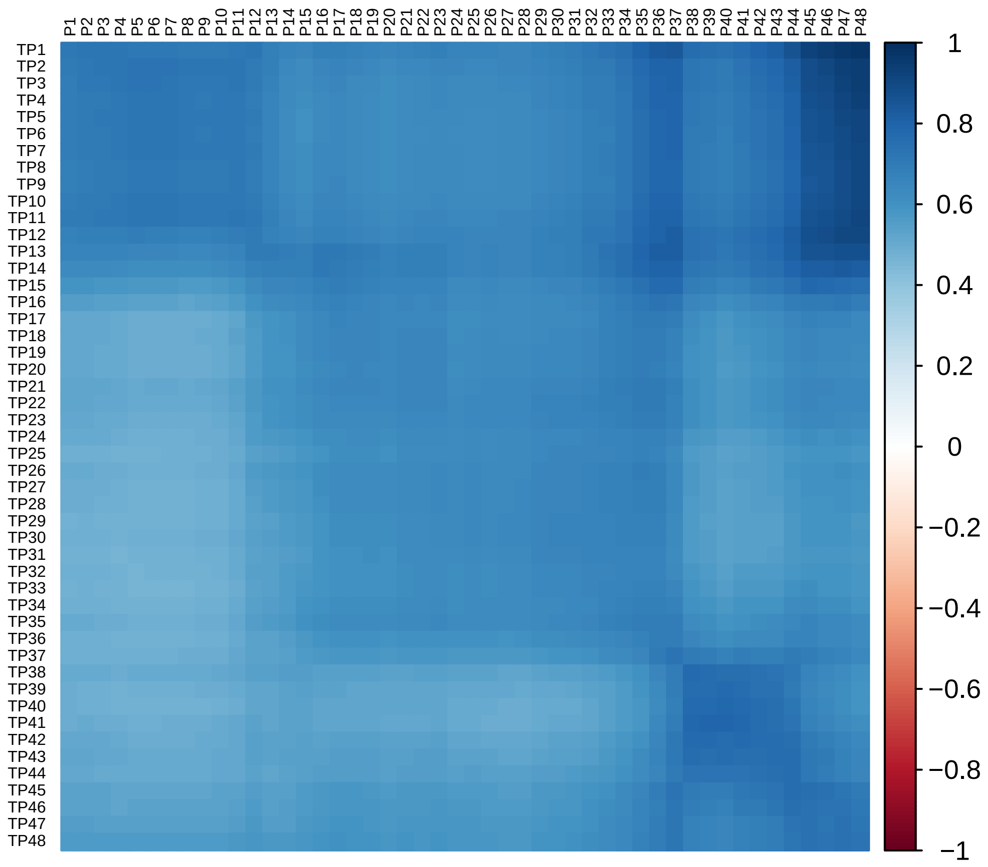
Additionally Some studies such as Grinsztajn et.al., 2022, shows that XGBoost perform better in tabular dataset, which as we use multiple non time-lag features, true for our particular case. Thus, we will use XGBoost regression model for the main prediction model while also develop the LSTM model for comparison. With XGBoost, we basically create separate multiple prediction targets each representing different trading period (code 5.1). Each of the prediction targets can potentially have different time-lag features and hyperparameters which will be determined via parameter tuning (discussed below).

5.2 Feature Selection

5.2.1 Time-lag features

As shown in the exploration we can see there are at least two cycles in the time series: daily and weekly cycles. Hence to exploit the daily cycles for our prediction we can use values from the previous day as the main regressor. To identify how these values correlated to the values from previous day on different time/trading period , we can check the correlation matrix. For this purpose we create a new dataframe containing only the dependent variable, and pivot it into daily observations, so each row has 48 demand values each representing different half-hourly time. We can use the feature TP as the pivoted column as it also represent the time (TP1 for 00:00, TP2 for 00:30, until TP48 for 23:30).

The following plot shows correlation matrix between today's values and previous values (TP for today and P for the previous day) (Code 5.1).



As can be seen above, the P =48 seems to have good correlation with the values of the next day especially for TP1 to TP16. For TP17 onward, the values from the same time of the previous day seems to have better correlation. Thus for each of the prediction target we use P48 and P=x (where x is the TP to predict) as the time-lag features. For example for TP = 5 we use P48 and P5 and for TP = 20 we use P48 and P20. There could be multicollinearity between these time-lag predictors, however since XGBoost is a tree-based algorithm we can allow this multicollinearity to exist.

5.2.2 Non time-lag features

As previously mentioned, the features: Month, Day_of_week, temperature (binned), holiday are important factors in determining the demand. We will include these categorical variables as additional features (code 5.2). As we use values from the previous day as the main regressor, we add the holiday status of the previous day to help the model to adjust the prediction.

5.2.3 Weather Data Exclusion

Including the weather data in the model can increase the prediction of electricity demand accuracy. However, this require good weather forecasting accuracy, which in reality cannot be guaranteed. Hence, we will consider two scenarios, one with weather data and other without weather data. If the model with weather data does not perform significantly better, it is more reasonable to use the model without the weather data. Additionally, we know that the feature Month can be correlated with temperature.

5.4 Parameter Tuning

We will conduct parameter tuning to find the best parameters for each 48 prediction targets. These are some of the main parameters that can control the tree structures, learning rate and regularization. We define the best parameters as parameters that provide the minimum MSE using cross validation. This can be done with gridsearch xgb.cv. To ensure the hyperparameter tuning can be conducted in reasonable computation time, we only consider 3 or 4 values for each of the paramaters (Code 5.3).

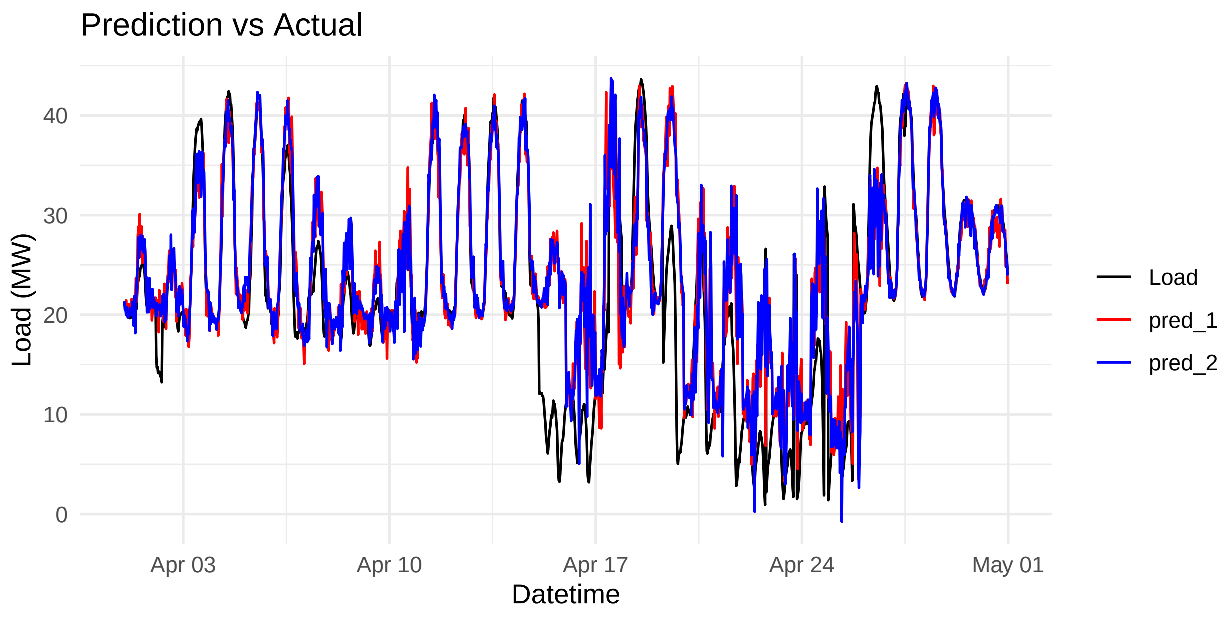
5.5 Model Comparison with LSTM

To ensure our XGBoost model robustness we can build LSTM model for comparison. As previously mentioned, with LSTM, we treat all features as sequential data. As opposed to the XGBoost model with multiple prediction targets, here the multi-step prediction can be done in one go, with 48 sequential prediction for each day. The general steps for this model is first to split for training and testing. Then we transform the variables: for example we can use MinMax scaling for demand values, and one-hot encoding for categorical features. Then we continue with training and testing. Finally, we gather the error measurements (MSE, MAPE, and MAE) and compare it with the XGBoost model.

6. Result

6.1 XGBoost Models

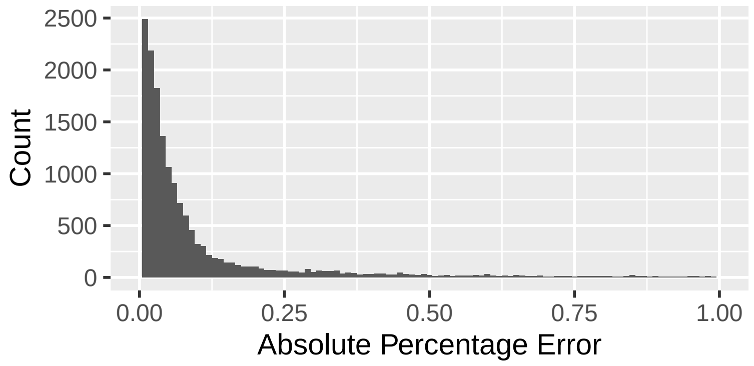
The following plot show samples of actual vs prediction where pred_1 for the model with weather data and pred_2 for the model without weather. Both can recognize the weekly (lower loads in the weekend) and daily patterns (peak and low point on certain time of the day). They seem to perform quite similarly.



Error measurements :

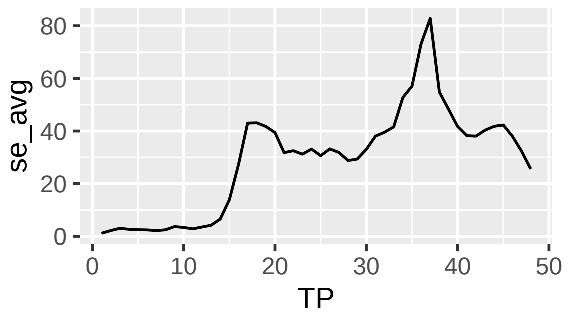
ape_1	ape_2	se_1	se_2
Min. : 0.00	Min. : 0.00	Min. : 0.00	Min. : 0.00
1st Qu.: 0.02	1st Qu.: 0.02	1st Qu.: 0.26	1st Qu.: 0.23
Median : 0.05	Median : 0.04	Median : 1.66	Median : 1.40
Mean : 0.23	Mean : 0.23	Mean : 29.02	Mean : 29.31
3rd Qu.: 0.11	3rd Qu.: 0.10	3rd Qu.: 9.17	3rd Qu.: 8.25
Max. :59.38	Max. :60.61	Max. :1605.97	Max. :1759.06
NA's :35040	NA's :35040	NA's :35040	NA's :35040

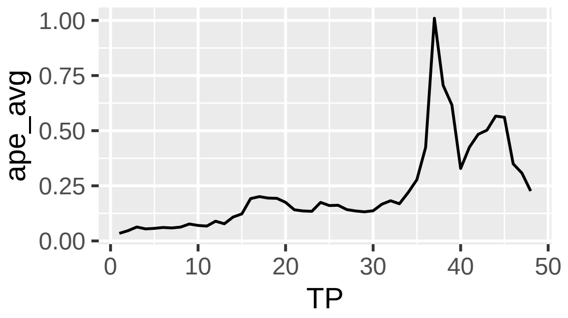
```
## Warning: Removed 35703 rows containing non-finite outside the scale range
## (`stat_bin()`).

## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_bar()`).
```



The error measurements from both models are similar, which suggest that Month and weather data carry the same information or are far less significant to the the time-lag features. It is noticeable that in both absolute percentage error and squared error there are wide gaps between the medians and the means. For example for the APE, the median is 4 to 5% while the mean is 23%. This means that most of the time errors are quite small, but at low occurrence the error become very high, probably at the hours with extreme (very low or very high) load. As can be seen above the APE is distributed mainly on the very low value, with low occurrences for APE greater than 0.25.

We can conclude that the model without weather data is more preferable, because both model perform equally and we don't need to be bothered by the weather data inaccuracy.





The plot above shows how error related to the TP (represent time of the day). In general, the error increases as it gets further away from the last observation, but seems to be more noticeably unpredictable at TP35-37.This represent the hour 17:00-19:00 which is after office hour, where activities are shifting from office to domestic (cooking, water heating, etc).

Feature Importance TP1

Feature	Gain	Cover	Frequency
time_lag_1	0.9228029	0.6046208	0.5287356
time_lag_2	0.0644412	0.2227885	0.2413793
temp	0.0047175	0.0558707	0.0651341
humidity	0.0032212	0.0362742	0.0613027
Month	0.0027678	0.0448896	0.0536398

Feature	Gain	Cover	Frequency
Day_of_week	0.0020495	0.0355562	0.0498084

Feature Importance TP30

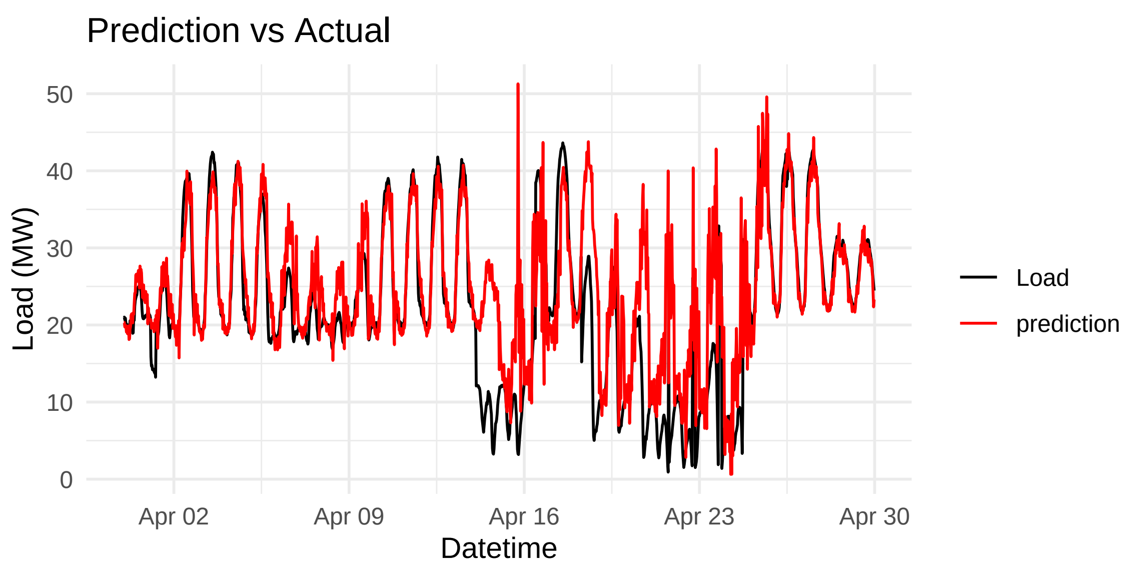
Feature	Gain	Cover	Frequency
time_lag_1	0.3623068	0.2478517	0.2335526
time_lag_2	0.3356857	0.2559714	0.2203947
Day_of_week	0.1865392	0.1446292	0.1842105
temp	0.0443041	0.1450681	0.1315789
humidity	0.0323268	0.1193347	0.1381579
Month	0.0247420	0.0475167	0.0625000
Is_holiday	0.0112284	0.0294179	0.0197368
Is_holiday_prev	0.0028670	0.0102102	0.0098684

As it goes further away from the last observation, the time-lag value of the same time from previous day (time-lag 2) become increasingly more important than the last known observation (time-lag 1). This shows quite strong daily patterns in the time series.Also as we predict further into the future, the model depends more to non-lag features hence the features contributes more evenly. The weather features contribution is insignificant in the low TP compared to Month, while in the TP 30 we see almost equal contribution of Month and Weather features. This suggest that adding weather data not necessarily improve the prediction performance.

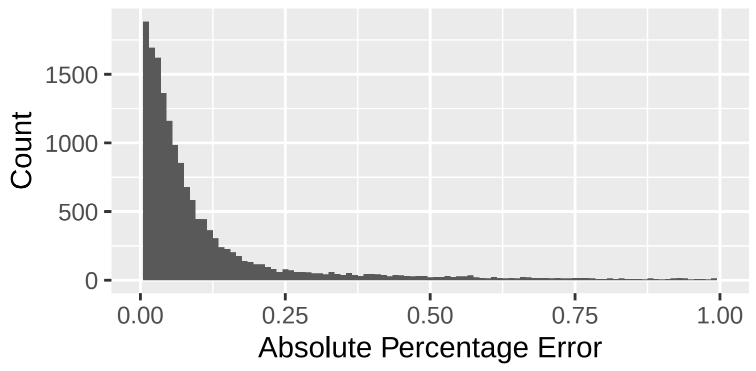
6.2 LSTM Model

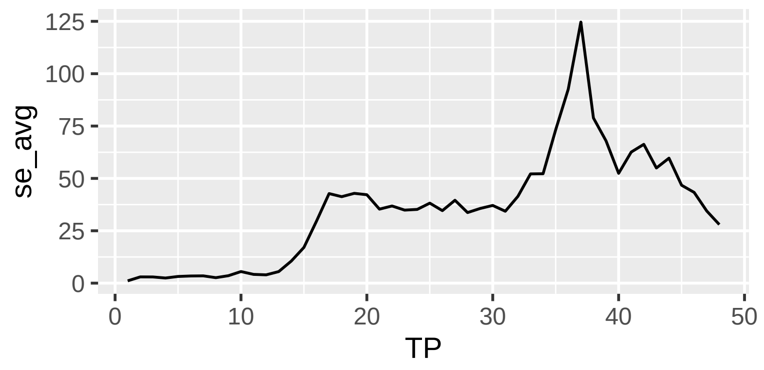
To ensure our model robustness we compare with LSTM model. Although, our problem is a time series problem, we frame it as tabular problem as we can easily incorporate time-lag features that can provide important contextual information about past patterns at specific times. Features like the day of the week and holidays can be encoded and included directly in the model. So in this case the temporal dependencies captured by LSTM are between the time-lag values that we use as features. We use the LSTM layer with 50 units and 'relu' activation to capture temporal patterns and non-linear relationships in the data. The dense layer with a single unit outputs the final prediction value. Training the model for 50 epochs with a batch size of 32 is chosen to balance between sufficient training iterations for convergence and computational efficiency.

This model produced the following result :



se	ape
Min. : 0.0000	Min. : 0.00000
1st Qu.: 0.4022	1st Qu.: 0.02418
Median : 2.5594	Median : 0.05599
Mean : 35.3817	Mean : 0.26275
3rd Qu.: 14.5196	3rd Qu.: 0.12859
Max. :2304.5917	Max. :56.49364





We can see that this model produces comparable result with XGBoost model, with slightly lower performance. We also have the same error patterns as shown in above plots (error distributions and error across prediction target). By comparing with result from a widely used algorithm such as LSTM, we are confident that the XGBoost model is sufficiently robust and accurate.

7. Appendix (R Code)

```r
#3.1
library(tidyverse)
library(extrafont)
library(hms)
library(zoo)
library(ggplot2)
library(Matrix)
library(tibble)
library(superml)
```

```r
#3.2
file='C:/Users/boypa/OneDrive - The University of Auckland/Project/Data/raw_df.csv'
raw_df = read.csv(file)
str(raw_df)
```

```r
#3.3
raw_df <- raw_df %>% mutate(Date=as_date(Date, format="%m/%d/%Y"))
raw_m1_df <- raw_df %>% mutate(Month = month(Date), Day_of_week = wday(Date))
```

```r
#3.4
file2='C:/Users/boypa/OneDrive - The University of Auckland/Project/Data/holidays_df.csv'
holiday_df = read.csv(file2)
holiday_df <- holiday_df %>% mutate(Date=as.Date(Date, format = '%Y-%m-%d'))
raw_m2_df <- raw_m1_df %>% left_join(holiday_df, by='Date')
raw_m2_df <- raw_m2_df %>% select(-X.x,-X.y)
knitr::kable(head(weather_df))
```

```r
#3.5
file3='C:/Users/boypa/OneDrive - The University of Auckland/Project/Data/auckland_weather_2021_2023.csv
weather_df <- read.csv(file3)
weather_df <- weather_df %>% select(datetime,temp,humidity)
weather_df <- weather_df %>% distinct(datetime, .keep_all = TRUE)
weather_df <- weather_df %>% separate(col = datetime,
                                      into = c('Date','time'), sep = 'T')
knitr::kable(summary(weather_df))
```

```r
#3.6
#fill NA
weather_df <- weather_df %>% mutate(Date=as.Date(Date), time=as_hms(time))
#add TP with map: "00:00:00" :1, "01:00:00" :3 , ... "23:30:00" :47
weather_df <- weather_df %>% mutate(TP = 1+2*hour(time))
#add new rows
new_rows_df <- weather_df %>%
  mutate(Date=Date,time=time+as.numeric(
    as.difftime(1800,unit='secs')), TP=TP+1, temp=NA, humidity=NA)
comb_df <- bind_rows(weather_df,new_rows_df)
weather_df <- comb_df %>% arrange(Date,time)
```

```r
#3.7
fill_avg <- function(df,col) {
    df_r <- df %>%
    mutate(above := lag({{col}}),
           below := lead({{col}})) %>%
    mutate({{col}} := ifelse(is.na({{col}}), (above + below) / 2, {{col}})) %>%
    select(-above, -below)
    return(df_r)
}
weather_df <- weather_df %>% fill_avg(temp)
weather_df <- weather_df %>% fill_avg(humidity)
knitr::kable(head(weather_df))
```

```r
#3.8
main_df <- raw_m2_df %>% left_join(weather_df, by=c("Date","TP"))
knitr::kable(head(main_df))
knitr::kable(summary(main_df[c('Load','temp','humidity')]))
```

```r
#3.9
idna = which(is.na(main_df$time))
main_df[idna,"time"] = as_hms((main_df[idna,"TP"]-1)*3600/2)
main_df$temp = na.approx(main_df$temp, na.rm=FALSE)
main_df$humidity = na.approx(main_df$humidity, na.rm=FALSE)
#last observation
main_df[nrow(main_df),"temp"] = main_df[nrow(main_df)-1,"temp"]
main_df[nrow(main_df),"humidity"] = main_df[nrow(main_df)-1,"humidity"]
```

```r
#3.10
main_complete_df <- main_df %>%
  group_by(Month, TP, Day_of_week, Is_holiday) %>%
  mutate(Load = ifelse(is.na(Load), mean(Load, na.rm = TRUE), Load)) %>%
  ungroup()
#add datetime column
main_complete_df  <- main_complete_df %>% mutate(Datetime= as.POSIXct(Date, tz='UTC') + time)
knitr::kable(summary(main_complete_df[c('Load','temp','humidity','Datetime')]))
```

```r
#4.1
ggplot(main_complete_df,aes(x=Load)) + geom_histogram(binwidth = 0.5)
ggplot(main_complete_df[1000:1500,], aes(x = Datetime, y = Load)) + geom_line()+
  ggtitle("Demand over datetime")+xlab("Datetime")+ylab("Load")
```

```r
#4.2
main_complete_df %>% group_by(Month,TP) %>%
  summarise(month_avg=mean(Load), .groups = 'drop') %>%
  ggplot(aes(x=TP, y=month_avg, color=as.factor(Month)))+geom_line()
```

```r
#4.3
main_complete_df %>% group_by(Day_of_week,TP) %>%
  summarise(day_avg=mean(Load), .groups = 'drop') %>%
  ggplot(aes(x=TP, y=day_avg, color=as.factor(Day_of_week)))+geom_line()
```

```
#4.4
main_complete_tb_df <- main_complete_df %>% mutate(temp_bin=cut(temp,
                                breaks = seq(2,30,2)))
main_complete_tb_df %>%  ggplot(aes(x=temp_bin,y=Load)) + geom_boxplot()+theme_light()
```

```
#4.5
main_complete_df %>% group_by(Is_holiday,TP) %>%
  summarise(Is_holiday_avg=mean(Load), .groups = 'drop') %>%
  ggplot(aes(x=TP, y=Is_holiday_avg, color=as.factor(Is_holiday)))+geom_line()
```

```
#5.1
#Create daily observation dataframe
main_daily_df <- main_complete_df %>% select(-c('Month','Day_of_week','Is_holiday','time','temp','humidi

#Create daily observation dataframe of the previous day
main_daily_prev_df <- main_complete_df %>% select(-c('Month','Day_of_week','Is_holiday','time','temp','I

#merge and remove first daily observation
main_daily_df <- main_daily_df[-1,]
main_daily_prev_df <- main_daily_prev_df[-1,]
cor_df <- cbind(select(main_daily_df,-Date),select(main_daily_prev_df,-Date))

#corelation matrix
cor_mat <- cor(cor_df,use="complete.obs")
cor_mat <- cor_mat[1:48,49:96]

#plot correlation matrix
corrplot(cor_mat, method = "color",
         tl.col = "black", tl.srt = 90, diag = TRUE,tl.cex=0.5)
```

```
#5.2
#to categorical feature
cat_feature <- c('Month','Day_of_week','Is_holiday','Is_holiday_prev')
main_complete_df[cat_feature] <- lapply(main_complete_df[cat_feature],factor)
#add  Is_holiday_prev
main_complete_df <- main_complete_df %>% mutate(Is_holiday_prev=lag(Is_holiday,48))
main_complete_df[1:48,'Is_holiday_prev'] <- 0 # for first 48 rows
```

```
#5.3
#Function to tune hyperparameters with XGBoost Cross Validation
run_xgb_cv <- function(data_x,data_y,grid_df){
  #reformat for xgb
  data_x <- as.matrix(sapply(data_x, as.numeric))
  data_y <- as.matrix(sapply(data_y, as.numeric))
  dt_train <- xgb.DMatrix(data=data_x,label=data_y)
  #list to store rsme
  rmse_mean_list = rep(0,nrow(grid_df))
  #loop for params combinations
    for (k in 1:nrow(grid_df)) {
      params <- list(
      booster = "gbtree",
      objective = "reg:squarederror",
```

```r
      eta = grid_df[k,'eta'],
      max_depth = grid_df[k,'max_depth'],
      lambda = grid_df[k,'lambda']
      )
      cv_model <- xgb.cv(params = params, data = dt_train,
                         nfold = 5, nrounds = 50, verbose = FALSE )
      rmse_mean_list[k] <- cv_model$evaluation_log$test_rmse_mean %>% mean()
    }
    #take the minimum rsme, use the coresponding params as output
    return(grid_df[which.min(rmse_mean_list),])
}

#Do hyperparameter tuning for all 48 prediction targets
best_params  <-  data.frame(eta=rep(0,48),max_depth=rep(0,48),lambda=rep(0,48))
#list of parameters, put in a grid as df
etas = c(0.1,0.3,0.6)
max_depths = c(3,6,9)
lambdas = c(3,6)
grid_df <- expand.grid(eta=etas,max_depth=max_depths,lambda=lambdas)
split_at <- floor((nrow(df_x_list[[1]]))*2/3)
for (i in 1: length(df_x_list)) {
  x_train <- df_x_list[[i]][2:split_at,]
  y_train <- df_y_list[[i]][2:split_at,]
  best_param <- run_xgb_cv (x_train,y_train,grid_df)
  best_params[i,] <- best_param
}
#save best params to csv
write.csv(best_params,'best_params.csv')
```

```r
#5.4
#to categorical feature
cat_feature <- c('Month','Day_of_week','Is_holiday','Is_holiday_prev')
main_complete_df[cat_feature] <- lapply(main_complete_df[cat_feature],factor)
#create multiple dataframes, each representing different TP
df_x_list <- list()
df_y_list <- list()
df_date <- main_complete_df %>% filter(TP==48) %>% select(Date)
feature_used <- c('Month','Day_of_week','temp','humidity'
              ,'Is_holiday','Is_holiday_prev')
for (i in 1:48) {
  time_lag_1 <- main_complete_df %>% filter(TP==48) %>% select(Load) %>% lag(1)
  time_lag_2 <- main_complete_df %>% filter(TP==i) %>% select(Load) %>% lag(1)
  df_x_list[[i]] <- main_complete_df %>% filter(TP==i) %>% select(all_of(feature_used))
  df_x_list[[i]] <- df_x_list[[i]] %>% mutate(time_lag_1=time_lag_1$Load,
                                              time_lag_2=time_lag_2$Load )
  df_y_list[[i]] <-  main_complete_df %>% filter(TP==i) %>% select(Load)
}
```

```r
#5.5
#splitting point
nrow_df <- nrow(df_x_list[[1]])
split_at <- floor((nrow_df)*2/3)
```

```r
#Do the train and prediction
best_params <- read.csv('best_params.csv')
predict_xgb_1_df <- list()
predict_xgb_2_df <- list()
predict_xgb_1_df <- lapply(1:48, function(x) data.frame(Date=df_date$Date[(split_at+1):nrow_df],
                                      pred_1=0)) #With weather data
predict_xgb_2_df <- lapply(1:48, function(x) data.frame(Date=df_date$Date[(split_at+1):nrow_df],
                                      pred_2=0))  #without weather data
#importance matrix
importance_mat_list <- list()

for (i in 1:48) {
  #add TP for later merging
  predict_xgb_1_df[[i]] <- predict_xgb_1_df[[i]] %>% mutate(TP=i)
  predict_xgb_2_df[[i]] <- predict_xgb_2_df[[i]] %>% mutate(TP=i)
  x_train_1 <- df_x_list[[i]][2:split_at,]
  x_train_2 <- df_x_list[[i]][2:split_at,] %>% select(-c(temp,humidity))
  x_test_1 <- df_x_list[[i]][(split_at+1):nrow_df,]
  x_test_2 <- df_x_list[[i]][(split_at+1):nrow_df,] %>% select(-c(temp,humidity))
  y_train <- df_y_list[[i]][2:split_at,]

  #convert to matrix form for xgboost
  train_data_1 <- as.matrix(sapply(x_train_1, as.numeric))
  train_data_2 <- as.matrix(sapply(x_train_2, as.numeric))
  label_data <- as.matrix(sapply(y_train, as.numeric))
  test_data_1 <- as.matrix(sapply(x_test_1, as.numeric))
  test_data_2 <- as.matrix(sapply(x_test_2, as.numeric))

  dtrain_1 <- xgb.DMatrix(data = train_data_1, label = label_data)
  dtrain_2 <- xgb.DMatrix(data = train_data_2, label = label_data)
  dtest_1 <- xgb.DMatrix(test_data_1)
  dtest_2 <- xgb.DMatrix(test_data_2)

  #use params from best_params
  params <- list(
    booster = "gbtree",
    objective = "reg:squarederror",
    eta = best_params[i,"eta"],
    max_depth = best_params[i,"max_depth"],
    lambda = best_params[i,"lambda"],
    min_child_weight = 1,
    subsample = 0.8,
    colsample_bytree = 0.8
    )

  xgb_model_1 <- xgb.train(params = params, data = dtrain_1, nrounds = 50)
  predict_xgb_1_df[[i]] <- predict_xgb_1_df[[i]] %>% mutate(pred_1=predict(xgb_model_1,dtest_1))
  xgb_model_2 <- xgb.train(params = params, data = dtrain_2, nrounds = 50)
  predict_xgb_2_df[[i]] <- predict_xgb_2_df[[i]] %>% mutate(pred_2=predict(xgb_model_2,dtest_2))

  #save importance matrix
  importance_mat_list[[i]] <- xgb.importance(feature_names = colnames(train_data_1), model = xgb_model_
}
```

```
#save sample
write_csv(importance_mat_list[[1]],'importance_TP1.csv')
write_csv(importance_mat_list[[30]],'importance_TP30.csv')

#aggregate predictions
agg_pred_1_df <- bind_rows(predict_xgb_1_df)
agg_pred_2_df <- bind_rows(predict_xgb_2_df)

#merge predictions from all targets to half-hourly dataframe
main_with_pred_df<-left_join(main_complete_df,agg_pred_1_df,by=c('Date','TP')) %>%
  left_join(agg_pred_2_df,by=c('Date','TP'))

main_with_pred_df <- main_with_pred_df %>% mutate(se_1=(pred_1-Load)^2,
                                                  se_2=(pred_2-Load)^2,
                                                  ape_1=abs(pred_1-Load)/Load,
                                                  ape_2=abs(pred_2-Load)/Load)
write_csv(main_with_pred_df,'main_with_pred_df.csv')
```

LSTM Model

```
#Create Multiple DF for each Prediction Target
#create multiple dataframes, each representing different TP
df_x_train <- list()
df_y_train <- list()
df_x_test <- list()
df_y_test <- list()
feature_used <- c('Month','Day_of_week',
                  'Is_holiday','Is_holiday_prev','Date','Load')
scale_params <- list()
for (i in 1:48) {
  time_lag_1 <- main_complete_df %>% filter(TP==48) %>% select(Load) %>% lag(1)
  time_lag_2 <- main_complete_df %>% filter(TP==i) %>% select(Load) %>% lag(1)
  df_x_m <- main_complete_df %>% filter(TP==i) %>% select(all_of(feature_used))
  df_x_m <- df_x_m %>% mutate(time_lag_1=time_lag_1$Load,
                              time_lag_2=time_lag_2$Load )%>% na.omit()
  df_x_train[[i]]  <-  df_x_m %>% filter(year(Date)<=2022) %>% select(-c('Load'))
  df_x_test[[i]] <- df_x_m %>% filter(year(Date)>2022) %>% select(-c('Load'))
  df_y_train[[i]] <-  df_x_m %>% filter(year(Date)<=2022) %>% select(Load)
  df_y_test[[i]] <-  df_x_m %>% filter(year(Date)>2022) %>% select(Load)
  scale_params[[i]] = c(min(df_y_train[[i]], na.rm = TRUE),max(df_y_train[[i]], na.rm = TRUE))
}
```

```
#Function for scaling
scale_func <- function(x,min_val,max_val) {
  (x - min_val) / (max_val - min_val)
}

inverse_scale <- function(x, min_val, max_val) {
  x * (max_val - min_val) + min_val
}

#preprocess data
preprocess_data <- function(tp) {
  dummy_encode <- dummyVars("~Month+Day_of_week", data = df_x_train[[tp]])

  x_train_encode <- predict(dummy_encode,newdata = df_x_train[[tp]])
  x_test_encode <- predict(dummy_encode,newdata = df_x_test[[tp]])
  x_train <-cbind(x_train_encode, df_x_train[[tp]])%>% select(-c('Date','Month','Day_of_week'))
  x_test <-cbind(x_test_encode, df_x_test[[tp]])%>% select(-c('Date','Month','Day_of_week'))
  x_train <- array(data.matrix(x_train), dim = c(nrow(x_train), 1, ncol(x_train)))
  x_test <- array(data.matrix(x_test), dim = c(nrow(x_test), 1, ncol(x_test)))
  y_train <- df_y_train[[tp]] %>% mutate(Load = scale_func(Load, scale_params[[i]][1],scale_params[[i]]

  return(list(x_train = x_train, y_train = y_train$Load, x_test=x_test))
}
```

```
#Training function
train_lstm_model <- function(x_train, y_train) {
  model <- keras_model_sequential() %>%
    layer_lstm(units = 50, activation = 'relu') %>%
    layer_dense(units = 1)

  model %>% compile(
    loss = 'mean_squared_error',
    optimizer = optimizer_adam()
  )
  history <- model %>% fit(
    x_train, y_train,
    epochs = 50,
    batch_size = 32,
    validation_split = 0.2
  )
  return(model)
}
```

```
df_pred_lstm_list <- list()
for (i in 1:48) {
  processed_df <- preprocess_data(i)
  lstm_model <- train_lstm_model(processed_df$x_train,processed_df$y_train)
  predictions_scaled <- lstm_model %>% predict(processed_df$x_test)
  predictions <- predictions_scaled %>% inverse_scale(scale_params[[i]][1],scale_params[[i]][2])
  df_pred_lstm <- as.data.frame(predictions)
  colnames(df_pred_lstm) <- c('prediction')
  df_pred_lstm_list[[i]] <- cbind(df_x_test[[i]],df_pred_lstm)
  print('Finish...')
}
```

```
#combine prediction
df_pred_lstm_model <- df_pred_lstm_list[[1]] %>% mutate(TP=1)
df_pred_lstm_model <- cbind(df_pred_lstm_model,df_y_test[[1]])
for (i in 2:48) {
  df_to_merge <- df_pred_lstm_list[[i]] %>% mutate(TP=i)
  df_to_merge <- cbind(df_to_merge,df_y_test[[i]])
  df_pred_lstm_model <- df_pred_lstm_model %>% rbind(df_to_merge)
}
df_pred_lstm_model <- df_pred_lstm_model %>% arrange(Date,TP)
```

```r
#add error columns calculate mean
df_pred_lstm_model <- df_pred_lstm_model %>% mutate(se = (Load-prediction)^2, pe =abs(Load-prediction)/
write.csv(df_pred_lstm_model, 'lstm_model_prediction.csv')
lstm_error_table <- df_pred_lstm_model %>% select(c('se','pe'))
knitr::kable(summary(lstm_error_table))
```