Table Of Contents

Chapter 1- Business and Situation Understanding

1.1. Background

Electricity demand forecasting is an important process for companies involved in electricity sector (Feng & Ryan, 2016). Transpower, which owns and operates the electricity transmission network in New Zealand, can use the knowledge of future electricity loads to ensure a stable and reliable dispatch of electricity. In addition to formulating their bids for electricity dispatch, generation companies can use that information to plan their operation capacities and find suitable time windows for maintenance. On the other hand, retailers leverage demand forecasting to strategically plan their electricity purchases from the wholesale market. This planning enables them to optimize their retail pricing strategies, ensuring they meet their customers' needs efficiently while maintaining competitiveness in the market.

Forecast in electricity demand typically categorized in three timescales: short-term, medium-term, and long-term, each with different purposes and potentially different needs of datasets (Panapakidis, 2020). Short-term forecasts are conducted over a period ranging from hours to days which is mainly used for daily operations, scheduling and dispatch. Medium-term forecasts are carried out over weeks to months and essentially used for wider operational decisions such as fuel imports and maintenance planning. Long-term forecast utilized monthly to yearly datasets for more strategic decisions or policy-making such as power system planning and investment of renewable energy projects. This study is focused on short-term forecasting, aiming to provide precise predictions of hourly to daily electricity demand. In addition to historical load data, many studies have included climate variables as predictor (Wu, Dou, & Yue, 2020) (Gao, Niu, Ji, & Sun, 2022) (Ghiassi, Zimbra, & Saidane, 2006). Study conducted by Behmiri et. al. (2023) indicated that models that include future temperatures significantly outperform models that exclude temperatures at any time-horizon. The impact of temperature in electricity use has been studied by Yao (2022) and Li (2023) which show positive effect of hot and cold day in electricity consumptions.

As previously stated, short-term forecast is critical for *gentailers* (a term for companies that function as both generator and retailer like *Contact, Genesis, and Meridian*) to manage their plant activities

and retailing operation. If the demand turns out to be bigger than the planned output of the plants, the customers will suffer from blackouts. This is not only bad for the customers but also for the company's reputation and eventually, sales. Conversely, excess of generation can lead to damage to electrical equipment in the grid system. This will create a huge economic impact.

In this study I take the position of a manager of a business analytics team in a *gentailer*. In order to help with efficient operations, the plant and sales operation teams feel the urge to know the future demand to help them in configuring their respective operation plans. Moreover, they want this information to reflect the market dynamics and be accessible anytime they want to. Hence as a business analytics team we translate these needs as decision support tools in the form of real-time demand forecasting.

New Zealand power agencies, particularly Transpower and Electricity Authority (EA) host massive and regularly updates databases on wholesale electricity operations (e.g. https://www.emi.ea.govt.nz/). This aligns with EA's which is to promote competition in reliable supply and efficient operation (Treasury, 2004). This database will be our critical source of information.

1.2 Assessment of Situation

Current practice of planning in plant and sales operations rely heavily on managers' intuitions rather than data driven decision-making. In fact, the company has not leveraged the talent that the business analytics team has. In terms of other resources (cloud and analytics services), calculations have been made and it favors the execution of this initiative, even if it only achieves the minimum success criteria. Further situation assessment is discussed below.

1.2.1 Data Requirement

The forecast must be presented in time units that align with the market trading settings. So, it makes sense if the model is directed toward time series model or any model that has temporal nature. Short-term demand forecasting or sometimes referred as short-term load forecasting (STFL) are mostly approached with either classical statistical methods or machine learning approaches (Feng & Ryan, 2016). Among classical statistical methods is regression, which takes, for example, temperature and previous load as variables. In machine learning approaches, load prediction can be made as both time series or non-time series (or tabular) data (Zhe, Tianzhen, Han, & Piette, 2021). Autoregressive Integrated Moving Average (ARIMA) is one of the oldest methods which predict a value from its lagged

values. Long Short-term Memory (LSTM) is a type of neural network approach that works with time series model. Random-forest and Support vector machines are widely used machine learning methods in load prediction. Machine learning approaches, especially with tabular model, is sometimes preferred as it can easily allow exogenous variable such as temperature to be included in the model while in the other approaches few extra steps might be required for exogenous inclusion.

In any approaches mentioned above, historical data with 30-minutes resolution is needed to produce the real-time prediction dashboard. As we want to include temperature data as predictor temperature data both historical and forecast (for real-time forecasting) is also needed. To make sure sufficiency of data we have checked and confirmed that EA (https://www.emi.ea.govt.nz/) have the necessary (volume and resolution) load-related data which is accessible for public via Azure Storage Explorer (AZE). For temperature data https://www.visualcrossing.com/ offer free 200 API calls for hourly weather data and 7-day forecast for any city in New Zealand.

In addition to data, a fully automatic real-time dashboard will require resources such as: database server, endpoints to execute scripts, and webhost for dashboard. As the primary data is located accessible through Azure environment it will be easier if the project is executed also in Azure environment.

1.2.2 Assumptions

To make this project doable within the limited timeframe and resources, a certain demand node is selected to reflect the actual demand nodes connected to our generation plants. The demand node is assumed to be connected with the same number of customers as we don't have data related to connected customers. Hence any actual load increase/decrease coming from customer increase/decrease is assumed to be negligible. Lastly, no data cooking is needed, so far.

1.2.3 Risks

Although it's unlikely, EMI database/server may be down or database name is changed with no prior announcement, which disrupts the automatic forecasting process and ultimately the operation planning. In this case the mitigation is to prepare for a longer prediction lead time (e.g. 2 days ahead). Although this might compromise the quality of the prediction, it is still better than having no prediction at all. Risk can also come from the temperature data. We can prepare with at least one additional weather data provider and add our API calls in case the primary call fails.

1.3 Data Mining Objectives

We have translated the needs mentioned by the plant and sales operations teams into a real-time forecasting dashboard. Since forecasting is important for planning the plant operations, the dashboard must provide the forecast within sufficient lead time, so the management can work on the plant settings. In this case we chose a one day-ahead prediction. The dashboard must also provide the prediction of maximum load and when it will happen during the day, as this is important in planning the capacity to utilize for the generating plants. On the demand side, the wholesale market in New Zealand is calculated and traded per half an hour (not day-ahead as most of European and American electricity market). This made the one-day operation consist of 48 trading periods. Hence this study will use this time unit for forecasting. Ideally the model should look at all demand nodes connected to our generating points, but for simplicity and to make it more doable we will pick one demand node and do the modelling on that node.

In general, the initiative covers the following analytics purposes. *Describe:* we want to know how electricity demand changes over time and how they correspond to variables such as hour of the day, days of the week, holiday, or festive season. *Diagnose:* the analytic processes will answer why certain days or hours can have higher or lower demand. *Predictive:* estimate the future (short-term) electricity demand. *Prescriptive:* with all the data, modelling and predictions that has been conducted what should we do to meet business objectives.

1.4 Success Criteria

The success of this initiative obviously will be defined by the accuracy of the forecast. However, since there is no prediction that can achieve 100% accuracy, we have asked the plant and sales team how deviation in the prediction can impact their operations. After careful risk calculation from various perspectives (technical, operations, sales, etc.), they require that the predictions must (1) have more than 80 % probability that they are within ±15% of the actual demand and (2) mean absolute error must be less than 5 MW. Therefore, while aiming for the highest probability on the lowest error, these measures of accuracy are defined as success criteria. In the case this is not achievable we can have discussions with plant management and retailing operation how lower quality of prediction can help them.

1.5 Project Schedule

We aim to finish the project in 10 calendar weeks, starting from business formulation, deployment to reiteration. Detailed scheduling presented as follow:

Table 1Project Schedule

Job Item		Duration	Start	Finish	Personel Involved	Objectives
Number	Name					
1	**Business Understanding**	**7**	**4 Mar 24 8:00:00 AM**	**12 Mar 24 05:00:00 PM**	Data Analysts Retail Operation Plant Operation Data Engineers	Clear business objectives, solid justification (CB Analysis), clear understanding between all parties
1.1	Business Objective	3	4 Mar 24 8:00:00 AM	6 Mar 24 05:00:00 PM		
1.2	Situation Assesment	2	7 Mar 24 8:00:00 AM	8 Mar 24 05:00:00 PM		
1.3	Data Mining Objective	1	11 Mar 24 8:00:00 AM	11 Mar 24 05:00:00 PM		
1.4	Project Scheduling	1	12 Mar 24 8:00:00 AM	12 Mar 24 05:00:00 PM		
2	**Data Understanding**	**4**	**13 Mar 24 8:00:00 AM**	**18 Mar 24 05:00:00 PM**	Data Analyst Retail Operation Plant Operation Data Engineers	Determine data requirements (types, volume, frequency, source)
2.1	Data Collection	1	13 Mar 24 8:00:00 AM	13 Mar 24 05:00:00 PM		
2.2	Data Description	1	14 Mar 24 8:00:00 AM	14 Mar 24 05:00:00 PM		
2.3	Data Exploration	1	15 Mar 24 8:00:00 AM	15 Mar 24 05:00:00 PM		
2.4	Data Quality	1	18 Mar 24 8:00:00 AM	18 Mar 24 05:00:00 PM		
3	**Data Preparation**	**6**	**19 Mar 24 8:00:00 AM**	**26 Mar 24 05:00:00 PM**	Data Engineers, Data Analysts, Analytics	Create data pipelining script, Making sure data is clean and ready for modelling
3.1	Data Pipeline and Selection	2	19 Mar 24 8:00:00 AM	20 Mar 24 05:00:00 PM		
3.2	Data Cleaning	1	21 Mar 24 8:00:00 AM	21 Mar 24 05:00:00 PM		
3.3	Table Construction	1	22 Mar 24 8:00:00 AM	22 Mar 24 05:00:00 PM		
3.4	Data Integration	1	25Mar 24 8:00:00 AM	25 Mar 24 05:00:00 PM		
3.5	Data Formatting	1	26 Mar 24 8:00:00 AM	26 Mar 24 05:00:00 PM		
4	**Data Transformation**	**2**	**27 Mar 24 8:00:00 AM**	**28 Mar 24 05:00:00 PM**	Data Analysts, Analytics	Create model with more than 80% of ±15% accuracy
4.1	Data Reduction	1	27 Mar 24 8:00:00 AM	27 Mar 24 05:00:00 PM		
4.2	Data Projection	1	28 Mar 24 8:00:00 AM	28 Mar 24 05:00:00 PM		
5	**Data Mining Method**	**3**	**29 Mar 24 8:00:00 AM**	**2 Apr 24 05:00:00 PM**		
5.1	Data Mining Methods Discussion	1	29 Mar 24 8:00:00 AM	29 Mar 24 05:00:00 PM		
5.2	Data Mining Methods Selection	2	1 Apr 24 8:00:00 AM	2 Apr 24 05:00:00 PM		
6	**Data Mining Algorithm**	**6**	**3 Apr 24 8:00:00 AM**	**10 Apr 24 05:00:00 PM**		
6.1	Exploration of DM Algorithm	2	3 Apr 24 8:00:00 AM	4 Apr 24 05:00:00 PM		
6.2	Selection of DM Algorithm	2	5 Apr 24 8:00:00 AM	8 Apr 24 05:00:00 PM		
6.3	BuildThe Model	2	9 Apr 24 8:00:00 AM	10 Apr 24 05:00:00 PM		
7	**Data Mining**	**4**	**11 Apr 24 8:00:00 AM**	**16 Apr 24 05:00:00 PM**		
7.1	Train and Test Design	2	11 Apr 24 8:00:00 AM	12 Apr 24 05:00:00 PM		
7.2	Data Mining Run	1	15 Apr 24 8:00:00 AM	15 Apr 24 05:00:00 PM		
7.3	Discuss on Pattern	1	16 Apr 24 8:00:00 AM	16 Apr 24 05:00:00 PM		
8	**Interpretation**	**7**	**17 Apr 24 8:00:00 AM**	**24 Apr 24 05:00:00 PM**	Data Analysts Retail Operation Plant Operation Data Engineers Analytics	Ensure data pipelining, ensure RT model prediction with new coming
8.1	Study Mined Patterns	1	17 Apr 24 8:00:00 AM	17 Apr 24 05:00:00 PM		
8.2	Visualization	1	18 Apr 24 8:00:00 AM	18 Apr 24 05:00:00 PM		
8.3	Interpret Result	1	19 Apr 24 8:00:00 AM	19 Apr 24 05:00:00 PM		
8.4	Asses and Evaluation	1	22 Apr 24 8:00:00 AM	22 Apr 24 05:00:00 PM		
8.5	Iterate Multiple Steps	3	23 Apr 24 8:00:00 AM	24 Apr 24 05:00:00 PM		
9	**Action**	**11**	**25 Apr 24 8:00:00 AM**	**9 May 24 05:00:00 PM**	Data Engineers Analytics Retail Operation Plant Operation	Ensure dashboard that is readable for all especially for plant and retail operations Ensure seamless real-time data pipelining
9.1	Deployment	3	25 Apr 24 8:00:00 AM	29 Apr 24 05:00:00 PM		
9.2	Monitor Implementation	3	30 Apr 24 8:00:00 AM	2 May 24 05:00:00 PM		
9.3	Maintain Implementation	2	3 May 24 8:00:00 AM	6 May 24 05:00:00 PM		
9.4	Enhance Solution	3	7 May 24 8:00:00 AM	9 May 24 05:00:00 PM		

We start the project with all parties both from data and business-related departments. Up until modelling analytics people will try and look for the model to produce not only high prediction quality but also reproducible and scalable. At this point, besides the analytics people we want to keep personnel which have some level of understanding of the business (in this case the data analyst). During deployment data engineers can ask plant and retail operations if they are happy with the dashboard. Finally, we allocate two weeks to allow cycles for continuous improvement.

Chapter 2- Data Understanding

2.1. Initial Data Collection

There are two main groups of data: load data and temperature data. The Electricity Authority hosts a large volume of raw datasets that update daily, even half-hourly for certain types of data. This data is accessible publicly through AZE. To access this data we need a token, which is shared regularly on the EA website. For the real-time data pipelining Azure Data Factories can be employed. For this initial data collection, we downloaded samples of .csv data AZE. There are many climate data providers available. These providers not only provide historical but also forecast data which is important for our prediction. One of these providers https://www.visualcrossing.com/ offer climate data through API calls. Various types of climate data are available, though in this case we are only interested in temperature data.

2.2. Data Description

We downloaded csv files from EA database through AZE. Each of these csv's contains one day trading data of all generation and demand nodes all over New Zealand. The following pattern if used to name the files: "YYYYMMDD_DispatchNodalPricesAndVolume.csv" where YYYYMMDD denotes the date. The following table shows the description of the table.

Table 2 Data Description

Column Name	Description	Type	Range
TradingDate	Trading date	Date	Date
TradingPeriodNumber	Trading period	Integer	1-50
IntervalDatetime	The start time of the trading period	Datetime	Datetime
RunDateTime	The end time of the previous trading period	Datetime	Datetime
CaseTypeCode	Case type code	"RTD"	"RTD"
CaseID	ID that identifies trading period	Integer	18 Digits
PointOfConnectionCode	Unique code assigned for nodes in the grid	String	7 Characters
UnitCode	Unique code for generating nodes	String	4 Characters
PlantName	Name of generating plants	String	Unknown
Island	Name of the Island	String	Unknown

LoadMegawatts	Electric Load in MW	Float	Unknown
InitialMegawatts	Initial Electric Load in MW	Float	Unknown
GenerationMegawatts	Generated Power in MW	Float	Unknown
LocationFactor	Factor represent location	Float	Unknown
DollarsPerMegawatthour	Price of electricity per MWh	Float	Unknown
IsDeadFlag	Indicatenode is shut down	Binary	Binary
IsDisconnectedFlag	Indicate node is disconnected	Binary	Binary

The weather data contains a much simpler piece of information which only contains temperature, humidity and datetime for desired location (in this case, Auckland). We can customize the API calls so that it will return only the desired piece of data. The API calls require the API key which we get by signing up for subscription. During downloading this data from the database no issues were encountered. In addition to date, time, and load data itself, trading period (TP) is one of the important columns in the dataset. This column basically represents half-hourly divisions within a day. Hence for this column the range is between 1-48, TP 1 represents 00:00 AM and TP 2 represents 23:30 PM.

2.3. Data Exploration

We collect samples of one year of the load data and try to look for any pattern that can help understanding the data. We choose the demand node that represents our demand market, in this case it is denoted as HOB1101(this code is for power station located in Hobson Street) in PointOfConnectionCode column. Before doing the exploratory analysis, some transformation (mainly filtering and merging) must be done. Please see below for the initial data exploration with visualization. We can observe some patterns of electricity use across different features of the dataset.

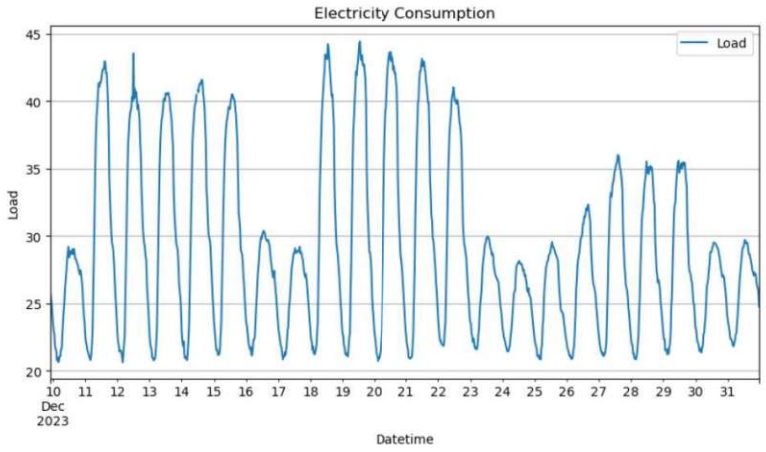


Figure 1 Load versus Datetime (sample data)

We can see two seasonal patterns shown in Figure 1. 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.

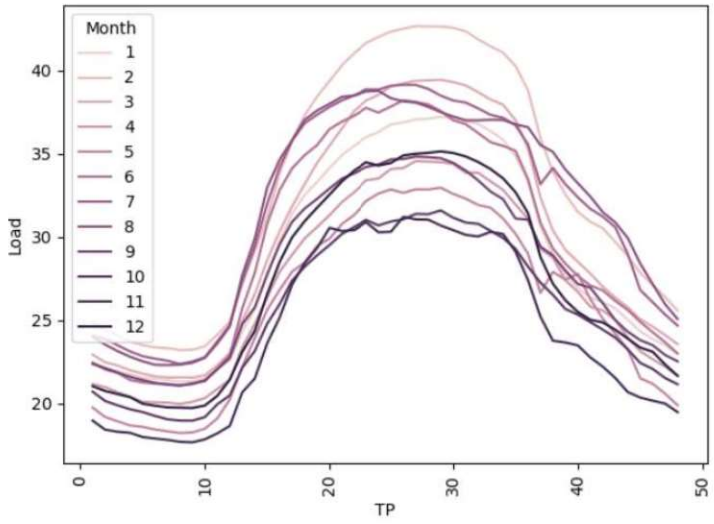


Figure 2 Monthly average load by TP

In Figure 2, we can see that demand peaks at TP 25-30 (12.30-15.00 AM). Different months show different average values on TP, where month 2 (February) shows the highest among other months. So, we can expect that month and TP are important features.

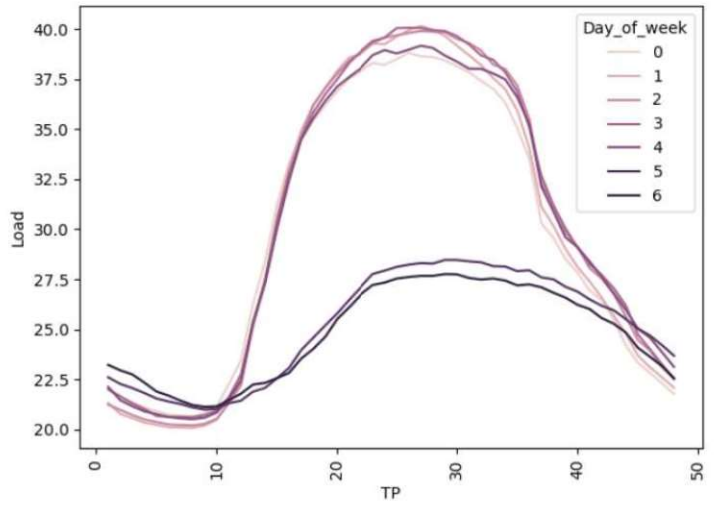


Figure 3 Daily average by TP

In Figure 3, it is observable that day of week 1 and 7 which are weekends have significantly lower load. The other days seem to have a comparable load. So, we can also expect the weekend/weekday feature to be an important predictor.

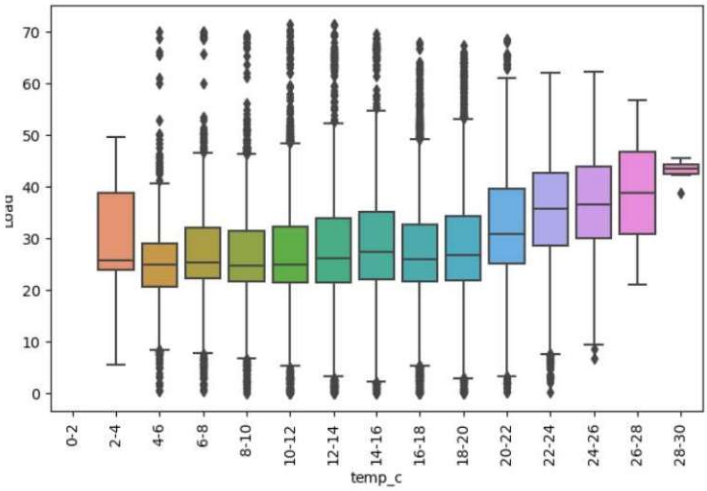


Figure 4 Load statistics by temperature ranges

As can be seen above, the mean load seems to increase as the temperature increases. This might confirm our previous observations where demand is at the highest in February (summer). There could be collinearity between month and temperature against Load. So further analysis in the analytical process should be conducted to determine the proper predictor to be used.

2.4. Data Quality

As a critical database that contains information used by companies involved in the electricity market, the data quality provided by EA is carefully maintained. In terms of data entry (null values, white spaces, blank value, and empty string), no anomality as shown below except for null values. By using python script, some null values are found and this account for about 2.6 percent of the total entries. Since we want to forecast the demand, we should address this issue which discussed in chapter 3. The lowest value is in the order of 0.001 MW and the highest is 62. The mean load is 29.9 MW. These values are considered to be valid as the use of electricity, even between the same time of the day, varies within the week. Moreover, certain events or holidays (which are to be included later) can influence the demand. For weather data, since we pull the whole table directly, we have to check if duplicate on datetime values exist. Figure 6 shows that null values on some columns. But most of

these columns are not to be used in the final data integration. Also, there are 3 duplicate entries in the datetime dataset. We will address all these issues in the next step.

summary	_c0	TP	Load
count	52560	52560	51189
mean	8759.5	24.5	28.62995959431561
stddev	5057.636463071578	13.853530889945894	10.477783506403151
min	0	1	0.001
25%	4380	13	22.078166666666664
50%	8758	24	27.343799999999998
75%	13138	36	35.266333333333336
max	17519	48	71.4325

_c0	Date	time	TP	Load	Datetime
0	0	0	0	1371	0

Figure 5 Null Values found in Load data

```
#check weather data
auckland_weather_check.show(5)
```

datetime	temp	humidity	feelslike	precip	precipprob	preciptype	snow	snowdepth	windgust	windspeed	winddir
1/01/2021 0:00	16.0	86.88	16.0	0.0	0	NULL	0	0	11.5	4.1	250.0
1/01/2021 1:00	16.4	90.02	16.4	0.0	0	NULL	0	0	11.9	4.1	184.0
1/01/2021 2:00	15.7	90.02	15.7	0.0	0	NULL	0	0	11.9	7.4	270.0
1/01/2021 3:00	15.8	90.99	15.8	0.0	0	NULL	0	0	11.5	1.8	324.0
1/01/2021 4:00	14.8	94.85	14.8	0.0	0	NULL	0	0	11.2	3.5	20.0

only showing top 5 rows

```
#check nulls weather data
check_nulls(auckland_weather_check)
```

datetime	temp	humidity	feelslike	precip	precipprob	preciptype	snow	snowdepth	windgust	windspeed	winddir
0	0	0	0	0	0	20286	1878	0	1	0	0

```
#check duplicates weather data
auckland_weather_check.groupBy("datetime").agg(count("*").alias("count")).filter(col("count") > 1).count()
```

```
3
```

Figure 6 Null values and duplicate from Weather data

Chapter 3 - Data Preparation

3.1. Initial Data Pipelining Plan

We have done initial data checking and currently deem that the data is sufficient both in quality and quantity for our business purposes. There are two main processes in this initiative. First is producing a model that can predict electricity demand, and second is deploying the model online and streaming the output of the prediction model in form of real-time dashboard. For the first process, there are two options: A) downloading all necessary historical data and processing it offline and B) use cloud/analytic service and conduct model development on that platform. At this stage option A is preferable as the data can be handled using Azure Storage Explorer and simple API call for the weather data and work out the model without the need of internet connection in addition to avoid service charge. However, for the second process, which is the data streaming process, external service which includes: database, pipelining, analytic endpoint, and webhosting, is required. For this process we will use Azure services, aligning with the current system in the company. Figure 7 shows the overall view of data pipelining plan.

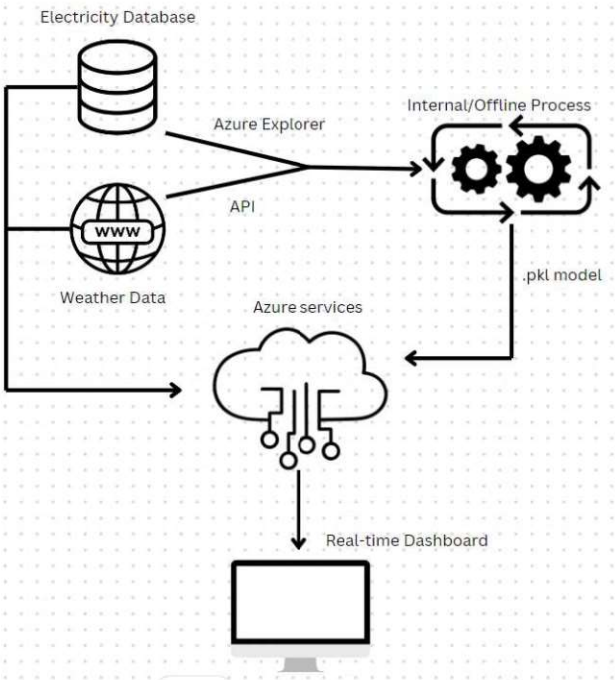


Figure 7 General Data Pipelining Plan (own source)

3.2. Gathering Relevant Data

The electricity load database located on the EA website is accessible with Azure Storage Explorer (ASE) using token that is periodically updated. The database contains various data related to electricity market operation in New Zealand. We must locate and download the csv's that are relevant to our objectives. Each csv file represents only one day of observations, hence for 2021-2023 data we need to download 1095 csv files which can be easily done with ASE. The weather data is collected via API call. Then we use python script to read all csv's and extract the relevant data filtering the 'PointOfConnection' column to our substation. This operation can be represented in Figure 8. As previously mentioned, we can pull the whole weather data for year 2021-2023 by a simple API call. We will need to make sure the join key for both data so we can integrate data properly.

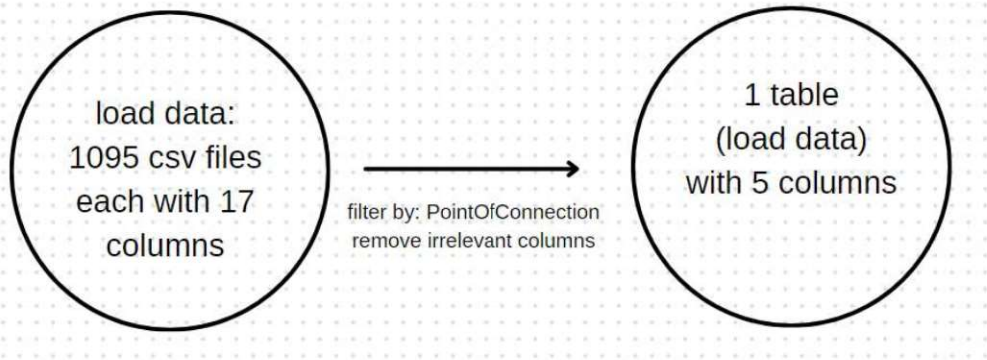


Figure 8 Gathering load data

3.3. Data Cleaning

As mentioned in chapter 2, we need to address the problem of null values and duplicates. We cannot just delete these observations altogether now because we want the data frame to have a complete date time structure since we are working with time series model. Hence, we have to fill out the null values so we can have a proper time structured data. To better reflect the actual measurement, we fill the null values with the average values of the same values of other features such as day, month, holiday, and hour (represented by TP). For example, if load value is null at datetime 20/01/2022 Monday, 01:00, then we will this up by averaging values that are in month January with day of week Monday and time 01:00. But this is only possible when we have these additional features in the

dataset. Thus, after adding these features which are discussed in the next sub-section, we can fill the null values. Additionally, we can make the data tidier by simplifying the decimal point from currently 6 to 2. This will help the algorithm reduce calculation complexity in addition to better human readability of the data. Figure 9 and Figure 10 show how load dataset has changed from unclean data.

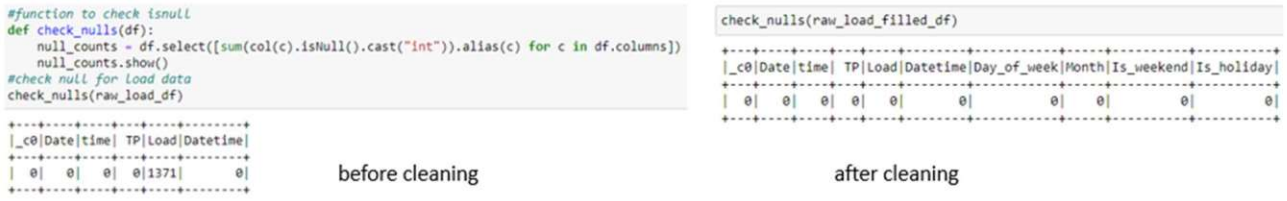


Figure 9 Load data information before and after cleaning

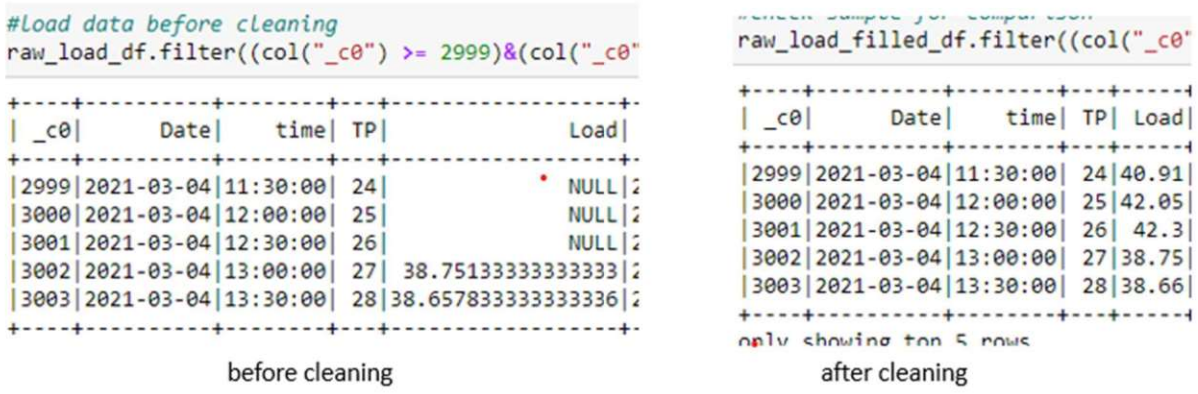


Figure 10 Load data samples before and after cleaning

There are two main issues in the weather data. First, there are duplication of time index which can make improper time structure of the data. Second, each row represents one hour observation while the load data is half-hourly. To deal with this we can use *drop_duplicates* and concatenate it to the same data frame with 30 minutes lag (see python script Load and Clean weather data for detail).

```
#check duplicates weather data before cleaning
auckland_weather_check.groupBy("datetime").agg(count("*").alias("count")).filter(col("count") > 1).count()
```

```
3
```

```
#check duplicates weather data after cleaning
auckland_weather_2020to2023.groupBy("datetime").agg(count("*").alias("count")).filter(col("count") > 1).count()
```

```
0
```

Figure 11 Weather data duplicates before and after cleaning

```
+----------------+----+--------+
|        datetime|temp|humidity|
+----------------+----+--------+
|01/01/2021 00:00|16.0|   86.88|
|01/01/2021 01:00|16.4|   90.02|
|01/01/2021 02:00|15.7|   90.02|
|01/01/2021 03:00|15.8|   90.99|
|01/01/2021 04:00|14.8|   94.85|
+----------------+----+--------+
```

before cleaning

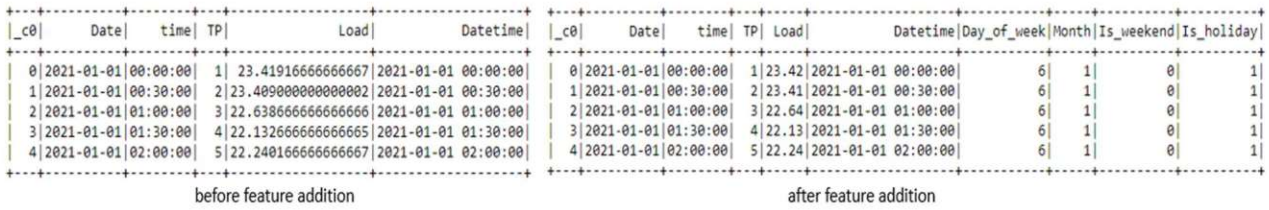
```
+----+--------+-------------------+
|temp|humidity|           Datetime|
+----+--------+-------------------+
|16.0|   86.88|2021-01-01 00:00:00|
|16.0|   86.88|2021-01-01 00:30:00|
|16.4|   90.02|2021-01-01 01:00:00|
|16.4|   90.02|2021-01-01 01:30:00|
|15.7|   90.02|2021-01-01 02:00:00|
+----+--------+-------------------+
```

after cleaning

Figure 12 Weather data glimpse before and after cleaning

3.4. Adding New Features

To predict electricity demand with pure time series analysis, we can rely only on the time index itself as the 'predictor'. However, solving it with machine learning type methods, it is important to have predictors that are related to seasonality instead of only the time index. In addition to that, we include external factor, which in this case is temperature. For this purpose, we add features like month, day of week, weekday/weekend, holiday, and temperature in addition to existing feature which is time. We can simply derive most of these features from the existing feature, which is the date. Only holiday and temperature features are retrieved from out of the load data. The holiday data is gathered from a website that provides information on public holidays that occurred in Auckland such as New Year, Easter, Auckland Anniversary Day, Anzac Day, among others.



_c0	Date	time	TP	Load	Datetime
0	2021-01-01	00:00:00	1	23.41916666666667	2021-01-01 00:00:00
1	2021-01-01	00:30:00	2	23.409000000000002	2021-01-01 00:30:00
2	2021-01-01	01:00:00	3	22.638666666666666	2021-01-01 01:00:00
3	2021-01-01	01:30:00	4	22.132666666666665	2021-01-01 01:30:00
4	2021-01-01	02:00:00	5	22.240166666666667	2021-01-01 02:00:00

before feature addition

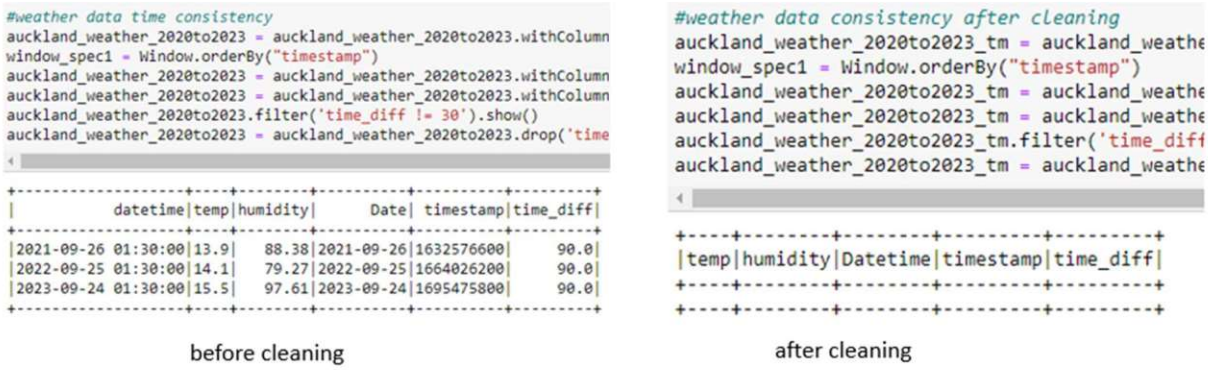
_c0	Date	time	TP	Load	Datetime	Day_of_week	Month	Is_weekend	Is_holiday
0	2021-01-01	00:00:00	1	23.42	2021-01-01 00:00:00	6	1	0	1
1	2021-01-01	00:30:00	2	23.41	2021-01-01 00:30:00	6	1	0	1
2	2021-01-01	01:00:00	3	22.64	2021-01-01 01:00:00	6	1	0	1
3	2021-01-01	01:30:00	4	22.13	2021-01-01 01:30:00	6	1	0	1
4	2021-01-01	02:00:00	5	22.24	2021-01-01 02:00:00	6	1	0	1

after feature addition

Figure 13 Feature addition in load data

3.5. Integrating Tables

Now we have two data tables, the load and weather table. We can join them into a main data frame to be easily handled in the analytic process. It is clear that the join key will be the datetime column. Hence before we join them we have to make sure that: 1) they have the same datetime data type, 2) they have the same datetime structure (half hourly from 00:00) 3)They have the same length of row. After checking the dataset with script, it is found that there are some gaps in the weather data. This can create null values after merging. To overcome this, we will use the datetime from load data to ensure it has the same datetime entries with load table and fill out values with the nearest available value. Result of this process shown in Figure 14 which shows a small addition of entries. Figure 15 shows similarity in data type of the merging key (datetime) and the number of entries. Figure 16 shows final data table after integration.



```
#weather data time consistency
auckland_weather_2020to2023 = auckland_weather_2020to2023.withColumn
window_spec1 = Window.orderBy("timestamp")
auckland_weather_2020to2023 = auckland_weather_2020to2023.withColumn
auckland_weather_2020to2023 = auckland_weather_2020to2023.withColumn
auckland_weather_2020to2023.filter('time_diff != 30').show()
auckland_weather_2020to2023 = auckland_weather_2020to2023.drop('time
```

datetime	temp	humidity	Date	timestamp	time_diff
2021-09-26 01:30:00	13.9	88.38	2021-09-26	1632576600	90.0
2022-09-25 01:30:00	14.1	79.27	2022-09-25	1664026200	90.0
2023-09-24 01:30:00	15.5	97.61	2023-09-24	1695475800	90.0

before cleaning

```
#weather data consistency after cleaning
auckland_weather_2020to2023_tm = auckland_weathe
window_spec1 = Window.orderBy("timestamp")
auckland_weather_2020to2023_tm = auckland_weathe
auckland_weather_2020to2023_tm = auckland_weathe
auckland_weather_2020to2023_tm.filter('time_diff
auckland_weather_2020to2023_tm = auckland_weathe
```

temp	humidity	Datetime	timestamp	time_diff

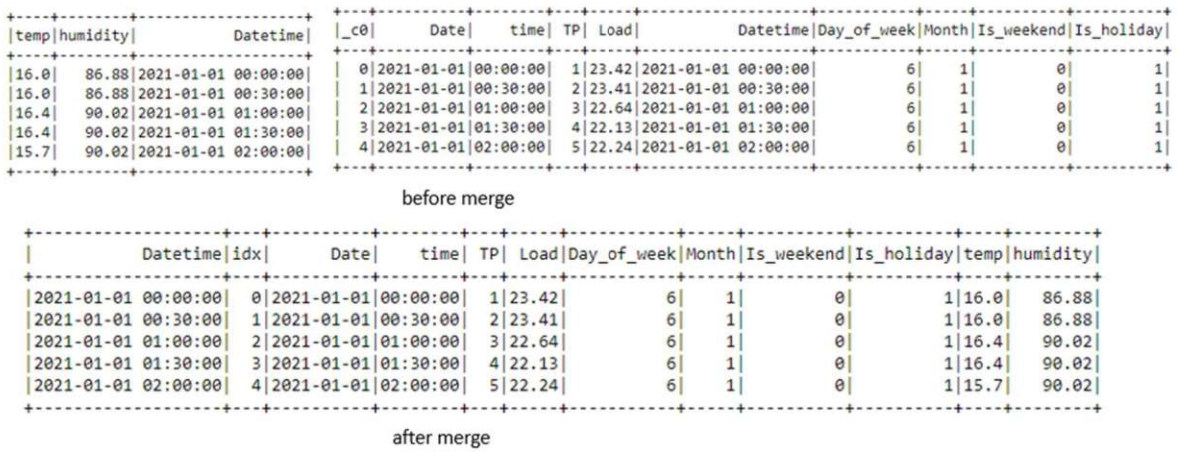
after cleaning

Figure 14 Weather data consistency before and after pre-integration process (shows inconsistency removed after cleaning)

```
auckland_weather_2020to2023_tm.printSchema()
raw_load_filled_df.printSchema()
```

```
root
 |-- temp: double (nullable = true)
 |-- humidity: double (nullable = true)
 |-- Datetime: timestamp (nullable = true)

root
 |-- _c0: string (nullable = true)
 |-- Date: date (nullable = true)
 |-- time: string (nullable = true)
 |-- TP: string (nullable = true)
 |-- Load: double (nullable = true)
 |-- Datetime: timestamp (nullable = true)
 |-- Day_of_week: integer (nullable = true)
 |-- Month: integer (nullable = true)
 |-- Is_weekend: integer (nullable = false)
 |-- Is_holiday: long (nullable = true)
```

Figure 15 Comparison between load and weather data



```
+----+--------+-------------------+
|temp|humidity|           Datetime|
+----+--------+-------------------+
|16.0|   86.88|2021-01-01 00:00:00|
|16.0|   86.88|2021-01-01 00:30:00|
|16.4|   90.02|2021-01-01 01:00:00|
|16.4|   90.02|2021-01-01 01:30:00|
|15.7|   90.02|2021-01-01 02:00:00|
+----+--------+-------------------+
```

```
+---+----------+--------+---+-----+-------------------+-----------+-----+----------+----------+
|_c0|      Date|    time| TP| Load|           Datetime|Day_of_week|Month|Is_weekend|Is_holiday|
+---+----------+--------+---+-----+-------------------+-----------+-----+----------+----------+
|  0|2021-01-01|00:00:00|  1|23.42|2021-01-01 00:00:00|          6|    1|         0|         1|
|  1|2021-01-01|00:30:00|  2|23.41|2021-01-01 00:30:00|          6|    1|         0|         1|
|  2|2021-01-01|01:00:00|  3|22.64|2021-01-01 01:00:00|          6|    1|         0|         1|
|  3|2021-01-01|01:30:00|  4|22.13|2021-01-01 01:30:00|          6|    1|         0|         1|
|  4|2021-01-01|02:00:00|  5|22.24|2021-01-01 02:00:00|          6|    1|         0|         1|
+---+----------+--------+---+-----+-------------------+-----------+-----+----------+----------+
```

before merge

```
+-------------------+---+----------+--------+---+-----+-----------+-----+----------+----------+----+--------+
|           Datetime|idx|      Date|    time| TP| Load|Day_of_week|Month|Is_weekend|Is_holiday|temp|humidity|
+-------------------+---+----------+--------+---+-----+-----------+-----+----------+----------+----+--------+
|2021-01-01 00:00:00|  0|2021-01-01|00:00:00|  1|23.42|          6|    1|         0|         1|16.0|   86.88|
|2021-01-01 00:30:00|  1|2021-01-01|00:30:00|  2|23.41|          6|    1|         0|         1|16.0|   86.88|
|2021-01-01 01:00:00|  2|2021-01-01|01:00:00|  3|22.64|          6|    1|         0|         1|16.4|   90.02|
|2021-01-01 01:30:00|  3|2021-01-01|01:30:00|  4|22.13|          6|    1|         0|         1|16.4|   90.02|
|2021-01-01 02:00:00|  4|2021-01-01|02:00:00|  5|22.24|          6|    1|         0|         1|15.7|   90.02|
+-------------------+---+----------+--------+---+-----+-----------+-----+----------+----------+----+--------+
```

after merge

Figure 16 Data table before and after integration

3.6 Formatting Data

In this particular project, the CSV format is chosen because the data sources are primarily available in this format. CSV files are particularly advantageous due to their simplicity and efficiency in handling large datasets. They are straightforward to generate from various applications, easy to import into data processing tools, and are supported by nearly all data mining and machine learning software, making them an ideal choice for this project. Hence, in our case reformatting and converting filetype for integration is not applicable for the time being.

Chapter 4 - Data Transformation

4.1. Data Reduction

Due to the nature of the problem, we have significantly reduced the data from the original sources from the previous steps. As previously mentioned, the original datasets consist of 1095 csv files, each with 17 columns and roughly 72000 rows, only for the load data. Since we have specifically determined the substation for this project only 48 out of 57000 rows are selected for each csv files that represent daily observations (see Figure 17). Horizontally, we have also reduced it from originally 17 columns to only 3 columns (date time, TP, and Load) before developing it to 8 columns with additional features (see Figure 18).

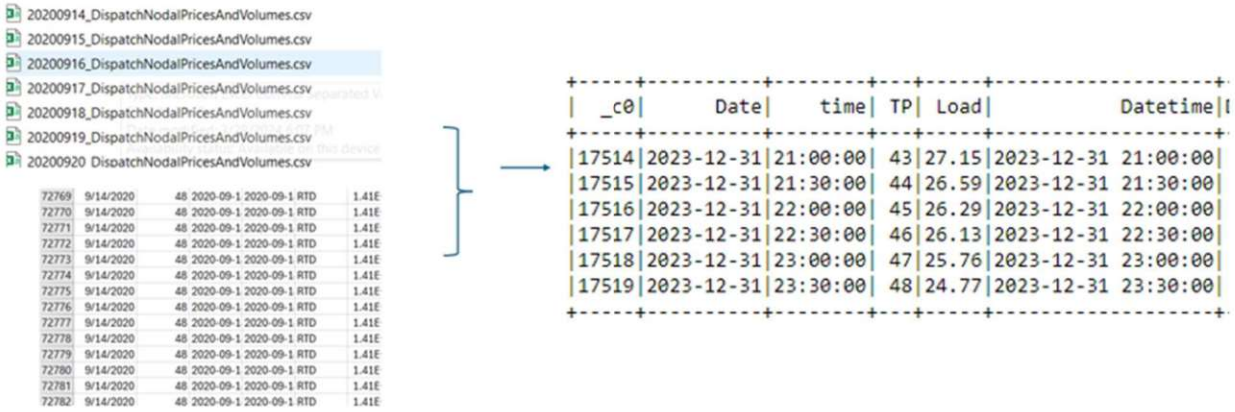


Figure 17 Vertical reduction of load data

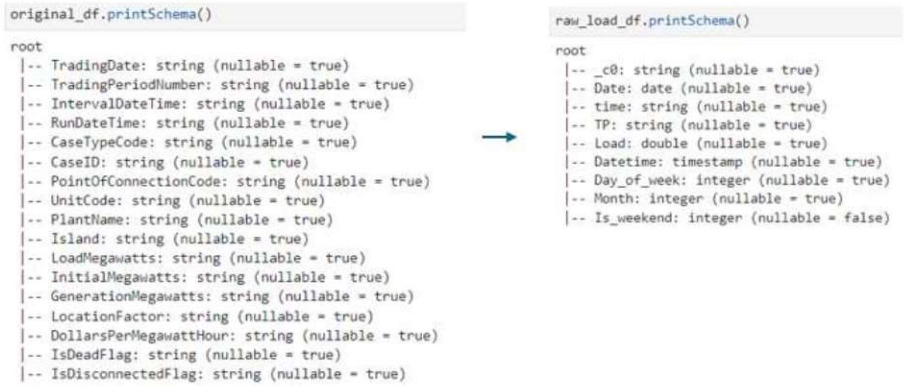
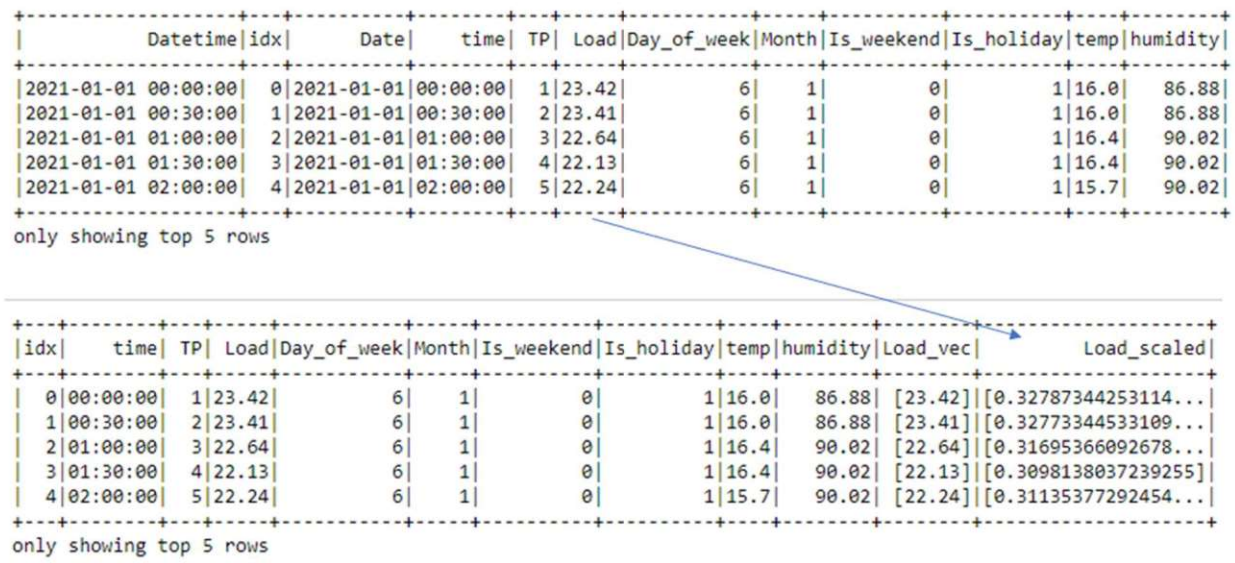


Figure 18 Horizontal Reduction of Load data

We cannot reduce vertically any further because as a time series problem, reducing it vertically will compromise the time index structure of the dataset. We potentially can reduce some feature as we analyze into the algorithm, hence this will be done in the next step.

4.2. Projection and Transformation

In terms of the target variable, the necessity for data transformation significantly depends on the data mining algorithm implemented. For example, when using traditional time series analysis techniques, transformations such as logarithmic scaling or differencing may be important to achieve stationarity of the series, which is a prerequisite for the effective application of many statistical models. Also algorithms like recurrent neural network algorithm (RNN) may need normalization of target variable so it can reach convergence more quickly. On the contrary, when utilizing machine learning approaches like XGBoost, such transformations are not mandatory. XGBoost and similar algorithms can handle non-stationary data effectively as they are designed to learn from raw values through their robust, tree-based structures. Consequently, our strategy regarding data transformation is directly aligned with the specific requirements and capabilities of the chosen algorithm, ensuring that our methodology is both appropriate and efficient for the task. Hence, at this point we do not conduct projection, and will do it as necessary or as the model requires in the next chapter. However, to give an example, we can apply min-max scaling for the load values, as shown in the following figure. In the case of utilizing LSTM, we can use these transformed values for training data and converting it back for actual prediction value.



Datetime	idx	Date	time	TP	Load	Day_of_week	Month	Is_weekend	Is_holiday	temp	humidity
2021-01-01 00:00:00	0	2021-01-01	00:00:00	1	23.42	6	1	0	1	16.0	86.88
2021-01-01 00:30:00	1	2021-01-01	00:30:00	2	23.41	6	1	0	1	16.0	86.88
2021-01-01 01:00:00	2	2021-01-01	01:00:00	3	22.64	6	1	0	1	16.4	90.02
2021-01-01 01:30:00	3	2021-01-01	01:30:00	4	22.13	6	1	0	1	16.4	90.02
2021-01-01 02:00:00	4	2021-01-01	02:00:00	5	22.24	6	1	0	1	15.7	90.02

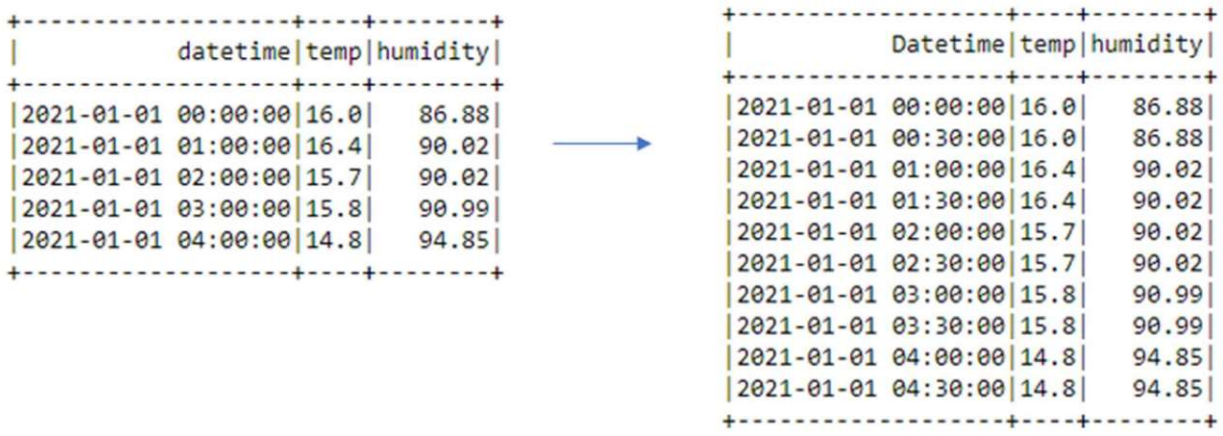
only showing top 5 rows

idx	time	TP	Load	Day_of_week	Month	Is_weekend	Is_holiday	temp	humidity	Load_vec	Load_scaled
0	00:00:00	1	23.42	6	1	0	1	16.0	86.88	[23.42]	[0.32787344253114...]
1	00:30:00	2	23.41	6	1	0	1	16.0	86.88	[23.41]	[0.32773344533109...]
2	01:00:00	3	22.64	6	1	0	1	16.4	90.02	[22.64]	[0.31695366092678...]
3	01:30:00	4	22.13	6	1	0	1	16.4	90.02	[22.13]	[0.3098138037239255]
4	02:00:00	5	22.24	6	1	0	1	15.7	90.02	[22.24]	[0.31135377292454...]

only showing top 5 rows

Figure 19 Example of Transformation by Min-Max scaling

We do need to transform the time index of the weather data to align with the time structure that will be used in the prediction. This is done by inserting rows in between original rows and filling the values with the previous observation. Figure 20 shows transformed data table by modifying the time index from previously hourly to half hourly observations.



datetime	temp	humidity
2021-01-01 00:00:00	16.0	86.88
2021-01-01 01:00:00	16.4	90.02
2021-01-01 02:00:00	15.7	90.02
2021-01-01 03:00:00	15.8	90.99
2021-01-01 04:00:00	14.8	94.85

Datetime	temp	humidity
2021-01-01 00:00:00	16.0	86.88
2021-01-01 00:30:00	16.0	86.88
2021-01-01 01:00:00	16.4	90.02
2021-01-01 01:30:00	16.4	90.02
2021-01-01 02:00:00	15.7	90.02
2021-01-01 02:30:00	15.7	90.02
2021-01-01 03:00:00	15.8	90.99
2021-01-01 03:30:00	15.8	90.99
2021-01-01 04:00:00	14.8	94.85
2021-01-01 04:30:00	14.8	94.85

Figure 20 Transformation on weather data.

Chapter 5 - Data Mining Methods

5.1. Data Mining Methods

5.1.1 Data Mining Methods in General

According to Fayyad et al., in their framework of Knowledge Discovery in Databases (KDD) (Fayyad, Gregory, & Padhraic, 1996), data mining is defined as the core analysis step that results in the discovery of interpretable patterns from large datasets. Data mining methods are broadly categorized into supervised and unsupervised learning. Supervised Learning involves training a model on a labeled dataset, where the target outcome is known. Supervised learning encompasses these types of methods (Shmueli, Bruce, Yahav, Patel, & Lichtendahl Jr, 2020, p. 11):

1. Prediction: This method involves estimating an unknown value. Prediction models are typically used for continuous output variables (regression tasks). Algorithms that can be used for this method are linear regression, regression trees, and neural network.
2. Classification: This method targets categorical output variables. It assigns data points to predefined categories based on learned relationships from the training data. Algorithms that can be used for this method are k-nearest neighbor, naïve bayes, logistic regression, and neural network.
3. Time Series Forecasting: This method is particularly designed for data that is indexed in time order. It is a predictive modeling technique that analyzes past trends and seasonal patterns to forecast future outcomes. We can use regression based and smoothing based algorithms in this method.

On the other hand, unsupervised learning does not require labeled data. It is used to identify patterns or internal structures within data where the outcomes are not specifically known in advance. This method is mainly used for clustering problems and may not be applicable for our problem.

5.1.2 Data Mining Objective and Selection Considerations

As discussed in the first chapter, our main objectives are:

1. Provide short-term electricity consumption predictions: 48-step ahead half hourly measurements.
2. The errors of this predictions must be within acceptable range: At least 80% of the predictions must be within 15% of MAPE and MAE of 5MW

With the above objective there are certain measurements of prediction inaccuracies that must be met which means the method must generate continuous prediction values rather than categorical. We also have to consider the data type and availability since some algorithms such as time series forecasting require perfect time-indexed structure. Fortunately, the available data have all the requirements satisfied for this algorithm.

5.2. Data Mining Methods Selection

It is clear that the one day ahead half hourly forecasting is naturally a time series problem, hence time series forecasting is the main option. However, we can frame the problem as ordinary prediction problem, only it is done in subsequent/ time-structured fashion. With this in mind and the considerations mentioned above we can conclude that both prediction and time series forecasting methods are highly relevant:

- Prediction: We can predict the future electricity demand based on various predictors such as historical demand and weather conditions. Since there are continuous and categorical predictors available, we can both use regression-based and tree-based algorithms for prediction.
- Time Series Forecasting: Given the nature of electricity demand, which is heavily time-dependent and subject to seasonal variations, time series forecasting is particularly suitable. This approach can leverage historical demand data to predict future values, considering trends, cyclic behavior, and seasonal patterns.

These supervised data mining methods satisfy both data availability consideration as well as data mining objectives. We clearly cannot use clustering and classification types of algorithms in our problem because as defined in the business objectives, we have to work with continuous target variables with defined error measurement. Some potential algorithms of these methods are linear regression, neural network, and time series analysis.

Chapter 6 - Data Mining Algorithms

6.1 Exploratory Analysis of DM Algorithms

6.1.1 Potential DM Algorithms to Explore

We have previously decided to explore algorithms within the **prediction method** and **time series forecasting method**. The main difference between these two methods is the way we frame the predictions. If we only consider the features of the target variable without formulating and exploiting the relationship between time and measurements historically, then we basically use prediction method. In this method we have options like linear regression, regression trees, and neural network. However, naturally, we can frame the prediction as time series forecasting by acknowledging that there are relationships between measurements and the time-lags (e.g. value at t highly correlated with value at t+7). We can also use algorithms that combine the strength of these two types of algorithms, that is exploiting the knowledge of direct relationship between time-lag values as well as using features such as day, time, temperature, etc for prediction. We identify families of algorithms to explore and how we can use it in our project, shown in in Table 3.

Table 3 Potential Algorithms to Explore

Methods	Family of Algorithms	How does it work for our context?	Algorithms
Prediction	Linear Regression	Use continuous feature such as time and temperature to predict future load with multivariate linear regression	Multivariate Linear Regression
	Tree Based Regression	Use continuous (e.g. time and temperature) and categorical features (e.g. day, month) future load with decision trees-based algorithms	Random Forrest, Gboost, XGboost
	Neural Nets Regression	Use continuous (e.g. time and temperature) and categorical features (e.g. day, month) future load with neural network	Neural Network
Time Series Forecasting	Time Series Analysis	Explore time series and find seasonality and trends to and develop time series model predict future load	ARMA, ARIMA, SARIMA
	Smoothing Based	Explore time series and find mathematical model that fit time series curve	Exponential Smoothing
Combination	Algorithms that can combine time series and prediction	Combine our knowledge in time series analysis (seasonality, trend etc) with continuous and categorical features to predict future load	XGboost with time-lag feature

6.1.2 Exploratory Analysis for Algorithm Selection

To help us select the appropriate algorithm for our project, we conducted exploratory analysis. Basically, this exploration is trying to answer these questions: Is there significant correlation between load and continuous type predictors? Is there correlation between load and categorical type predictors? Is there significant trend and seasonality that we can recognize from the time series? For this purpose, we can recall again some visuals from the previous chapter. In addition to that, we present autocorrelation (ACF) and partial autocorrelation factor (PACF) for load time series with various differencing values.

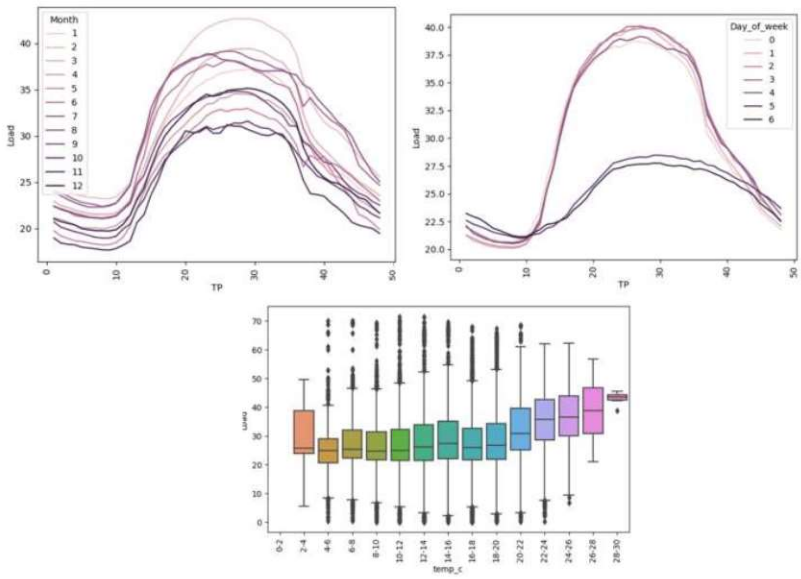


Figure 21 EDA Visualization

Upon analyzing Figure 21, we see pattern of load along TP values. This pattern is clearly not linear. The Month feature also influences loads, as shown in the top right plot, where certain month can have higher or lower than other month along TP values. Similarly, the day of week feature influence load, but most notably if it categorized into weekend (day=1,7) and weekdays(day=2-6). The bottom figure shows there is an almost linear relationship between load and temperature.

We try to identify seasonality and correlations between time-lag values from the ACF and PACF. Before that, we first try some differencing values to transform the time series into stationarity (Figure 22). Then we use this transformation to plot ACF and PACF and find correlations. From Figure 22and Figure 23 we can conclude that firstly, there is a clear seasonality which in this case is 48. Secondly,

there is significant correlation found in time-lag 1 and time-lag 48. Please note that, as pyspark do not have library to handle time-series model we use R for this exploration.

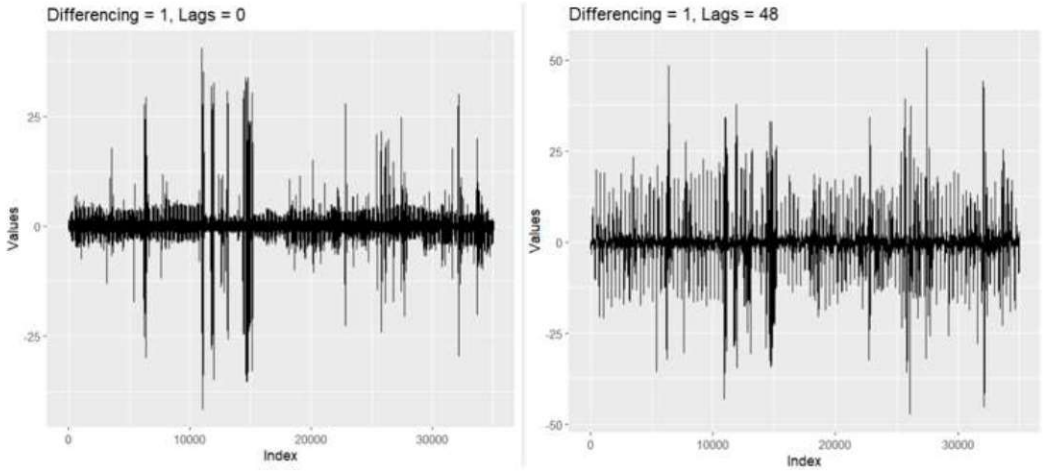


Figure 22 Differencing for stationarity.

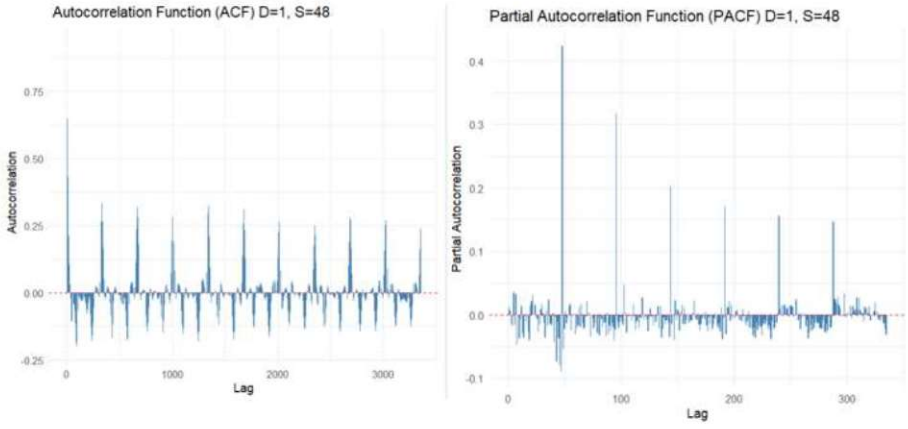


Figure 23 PACF and ACF plot of transformed time series

6.2 Exploratory Analysis for Algorithm Selection

Based on the above exploration we conclude that there we have both categorical and continuous that can be used as predictors. Moreover, there are time-series patterns that we can exploit to help increase the accuracy of the predictions. Hence it is reasonable to use algorithms that can exploit time-lag and other features as predictors. In this case we have two options: XGBoost with additional time-lag features and LSTM. We summarize our rationale for selecting these algorithms in Table 4.

Table 4 Algorithm preferences from exploratory analysis

Methods	Family of Algorithms	Advantage	Disadvantage	Algorithms
Prediction	Linear Regression	Linear correlation temp vs load	Non-linearity of other features, categorical predictor	Multivariate Linear Regression
	Tree Based Regression	Categorical and continuous variable corelate with target	Can not exploit time-series information	Random Forest, XGboost
	Neural Nets Regression	Categorical and continuous variable corelate with target	Can not exploit time-series information	Neural Network
Time Series Forecasting	Time Series Analysis	Time series pattern and seasonality exist	Can not exploit external features	ARMA, ARIMA, SARIMA
	Smoothing Based	Time series pattern and seasonality exist	Can not exploit external features	Exponential Smoothing
Combination	Algorithms that can combine time series and prediction	Categorical and continuous variable corelate with target, can exploit time-series information	Complex model (Increased computation time and resource)	XGboost with time-lag features, LSTM

6.3 Model Development

We now have two algorithms to pick: XGBoost with time-lag feature and LSTM. XGBoost algorithm which is based on gradient decent, was developed mainly for tabular data. Meanwhile LSTM, a RNN based, was designed to model temporal sequences (Sak, Senior, & Beaufays, 2014), hence it is closely related to time series problem. It should be noted that our data is now largely tabular because of many features that have been added in addition to the original time series data.

At this point we prefer XGBoost because of its flexibility in multi-step prediction. In XGBoost we can do multi-step prediction recursively (e.g. use t+1 prediction as input for t+2 prediction) or use different time-lag for different steps of prediction, hence we can use only actual values as inputs. Applying the latter in LSTM requires configuring the neural network which may involve complex modification to the network architecture. Moreover, researchers recommended the use of XGBoost in mainly tabular data over neural network-based algorithm (Grinsztajn.Léo, Oyallon, & Gaël, 2022).

6.3.1 Feature Engineering: Time-lag features

We have previously added features that we think can be good predictors for the load values. However, we have not created the time-lag features. For this purpose, we have to modify the data frame by adding these time-lag features. The questions will be: 1) what time-lags will be used? and 2) since

there will be 48 steps of prediction, do they use the same time-lags? To answer this question, we have to analyze the correlation matrix.

```
#create correlation matrix and visualize
vector_col = "variables"
assembler = VectorAssembler(inputCols=cols, outputCol=vector_col)
main_TP_vect = assembler.transform(main_TP_wide_df).select(vector_col)
#correlation matrix
cormat = Correlation.corr(main_TP_vect, vector_col).collect()[0][0]
#to dense matrix
cormat_array = cormat.toArray()
```

```
#visualize, focus only second quadrant
matrix_size = cormat_array.shape[0]
half_size = matrix_size // 2
second_quadrant = cormat_array[:half_size, half_size:]

plt.figure(figsize=(14, 12))
sns.heatmap(second_quadrant, annot=True, cmap='coolwarm', vmin=-1, vmax=1,
            xticklabels=cols[half_size:], yticklabels=cols[:half_size],fmt='.2f',annot_kws={'size':6})
plt.title("Second Quadrant of the Correlation Matrix")
plt.show()
```

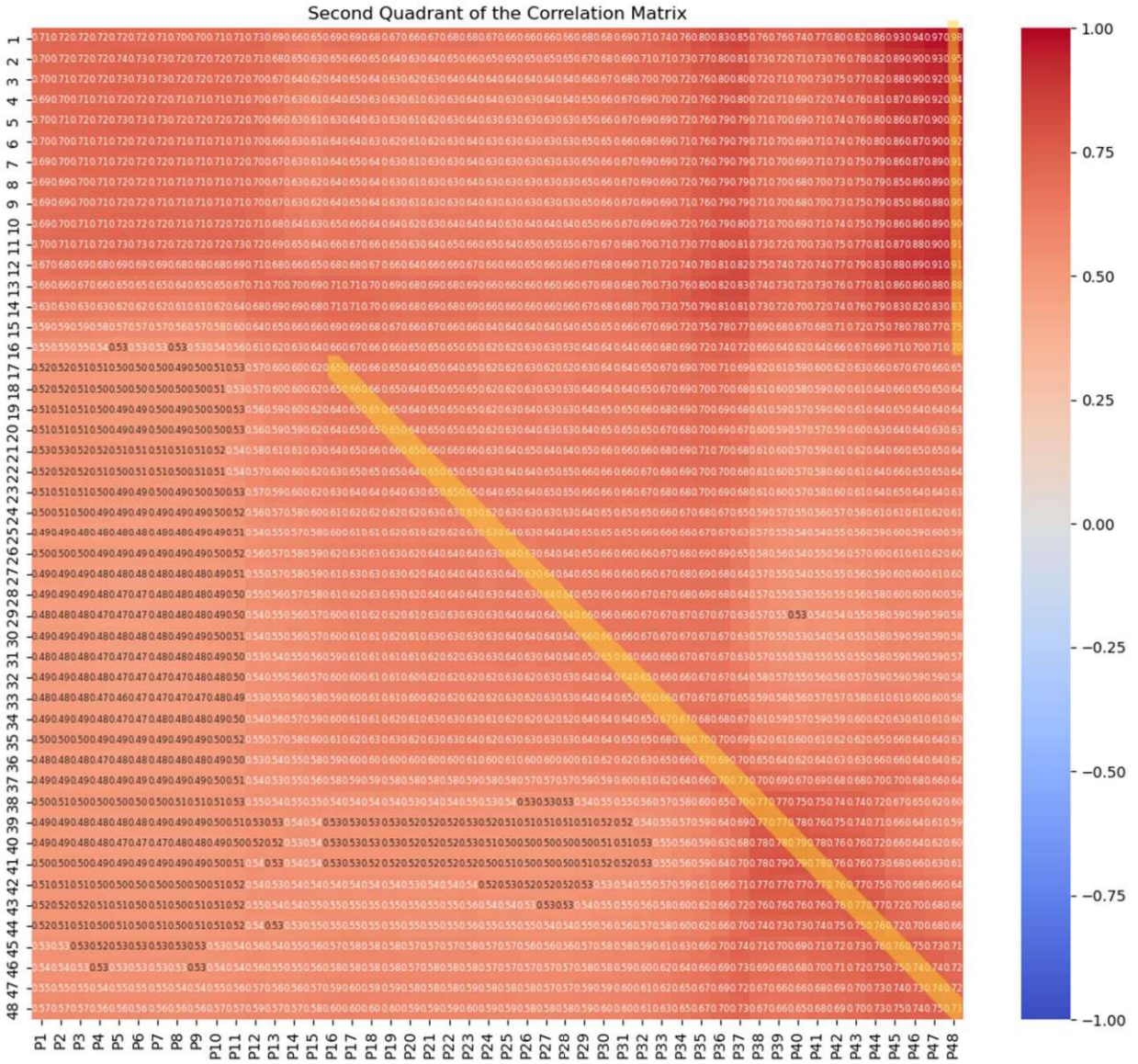


Figure 24 Correlation matrix on time-lag features (higher correlation highlighted with yellow mark).

Figure 24 shows correlation matrix between time-lag features. For clarification, TP-X denotes the load value at trading period X, while P-X denotes the load values at trading period X on the previous day. As can be noticed P 48 is highly correlated with TP1-TP16. In other words, value at TP 48 is the best time-lag predictor for the next day's trading period 1 to 16. However, TP48 correlation seems to decrease significantly for P17 onward. Generally, it can be observed that for TP17 onward, the same trading periods from the previous day have the highest correlations. Hence, we can consider two scenarios regarding time-lag features. First is to use only the most strongly correlated feature, second is to use 2 time-lag features. For example in the second scenario, to predict TP1 we use P48 and P1 as the time-lag features.

6.3.2 Feature Engineering: Additional Feature

Since we use the previous day values as the main predictor, we can help the model accuracy by providing information about the change/difference of certain features between the predicted and the previous day, particularly for features that are not showing in regular fashion. To give an example, instead using the *Is_holiday* value of the predicted day, we feed the model with the *Is_holiday* values of both the previous day and the predicted day. In this way, we help the model on how much to increase or decrease the load value since generally holidays have lower values than non-holidays. A simpler way is to create a feature that represents the change of values rather than feeding the model with two features. We can apply the same principle to Is_weekend. However, we already have *day_of_week* feature which can inform the model if the previous day is a weekend or weekday by knowing the predicted *day_of_week* value. Hence, we can simplify the used features by removing Is_weekend and use *day_of_week* instead. Table 5 shows features used for predictions.

```
#Add previous holiday information features
w = Window.orderBy("Datetime")
main_df = main_df.withColumn("Is_holiday_prev", lag("Is_holiday", 48).over(window_spec))
```

```
main_df.dropna().show(5)
```

```
+-------------------+---+----------+--------+---+-----+-----------+-----+----------+----------+----+--------+--------------+
|           Datetime|idx|      Date|    time| TP| Load|Day_of_week|Month|Is_weekend|Is_holiday|temp|humidity|Is_holiday_prev|
+-------------------+---+----------+--------+---+-----+-----------+-----+----------+----------+----+--------+--------------+
|2021-01-02 00:00:00| 48|2021-01-02|00:00:00|  1|22.79|          7|    1|         1|         1|18.0|    87.5|             1|
|2021-01-02 00:30:00| 49|2021-01-02|00:30:00|  2|22.69|          7|    1|         1|         1|18.0|    87.5|             1|
|2021-01-02 01:00:00| 50|2021-01-02|01:00:00|  3|21.94|          7|    1|         1|         1|17.8|   88.79|             1|
|2021-01-02 01:30:00| 51|2021-01-02|01:30:00|  4|21.56|          7|    1|         1|         1|17.8|   88.79|             1|
|2021-01-02 02:00:00| 52|2021-01-02|02:00:00|  5|21.08|          7|    1|         1|         1|18.0|   88.19|             1|
+-------------------+---+----------+--------+---+-----+-----------+-----+----------+----------+----+--------+--------------+
only showing top 5 rows
```

Figure 25 Adding Is_holiday_prev feature.

Table 5 Used Features

Target Variables	Time-lag Features First Scenario	Time-lag Features Second Scenario	Other features
TP1 to TP16	P48	Use both P1 to P48 and P48	Month, Day of week, Temperature, Humidity, Is_holiday, Is_holiday_prev (categorical)
TP17 to TP48	P17 to P48		

6.3.3 Model Parameters

One of the critical model parameters in XGBoost is the n_estimators (number of trees). This parameter sets the number of trees (boosted trees) to be created in the model. More trees can model more complex patterns but can also lead to overfitting, hence we set it default 100. Another important parameter is *eta* which reflects the learning rate of the model. We use a balance 0.3 for *eta* because we don't want to produce overfitting model but at the same time, we want each tree to make substantial contribution to the prediction. Finally, we use default regularization parameters (lambda and alpha) for now, and we will fine tune as we evaluate model performance.

6.3.4 Reshaping Data Frames

Since we have 48 number of predictions with different predictors, we create 48 data frames from the main data frames. First, we flatten the data into daily observations (hence each row has 48 Load values) with wide pivot. Then for each 48 data frames we filter based on the trading period and the used time-lag features as previously determined. Figure 26 shows first few rows of data frame for trading period 1. Notice the NA value which is expected as we are in the first of the dataset hence, we don't have the previous value as predictor. Now that we basically make it an ordinary prediction dataset (not exactly a time series) we can dismiss the null values.

```
#create multiple target and time-lag features for both scnearios
#scenario1: use strongest correlated
#scenario2: use multiple time-lag features
df_list = {}
w = Window.orderBy("Date")
load_48 = main_df.filter(col('TP')==48).select('Date','Load')
load_48_prev = load_48.withColumn('Lag_a', lag('Load', 1).over(w))
load_48_prev = load_48_prev.select('Date','Lag_a')
for i in range(1,49) :
    df_list[i] = main_df.filter(col('TP')==i)
    if i<=16 :
        df_list[i] = df_list[i].join(load_48_prev, on="Date", how="inner").withColumnRenamed('Lag_a','Lag_1')
        df_list[i] = df_list[i].withColumn('Lag_2', lag('Load', 1).over(w))
    else:
        df_list[i] = df_list[i].join(load_48_prev, on="Date", how="inner").withColumnRenamed('Lag_a','Lag_2')
        df_list[i] = df_list[i].withColumn('Lag_1', lag('Load', 1).over(w))
```

```
df_list[1].drop('Datetime').show(5)
```

Date	idx	time	TP	Load	Day_of_week	Month	Is_weekend	Is_holiday	temp	humidity	Is_holiday_prev	Lag_2	Lag_1
2021-01-01	0	00:00:00	1	23.42	6	1	0	1	16.0	86.88	NULL	NULL	NULL
2021-01-02	48	00:00:00	1	22.79	7	1	1	1	18.0	87.5	1	23.42	22.78
2021-01-03	96	00:00:00	1	22.83	1	1	1	0	17.1	99.3	1	22.79	22.77
2021-01-04	144	00:00:00	1	22.26	2	1	0	1	19.0	92.91	0	22.83	22.97
2021-01-05	192	00:00:00	1	22.34	3	1	0	0	18.9	88.2	1	22.26	23.29

Figure 26 Data frame for TP=1

```
#Prediction function
def xgb_model(df,features,params):
    data = df.dropna()
    assembler = VectorAssembler(inputCols=features, outputCol="features")
    data = assembler.transform(data)
    #split
    train_data = data.filter(year(col("Date")) <= 2022)
    test_data = data.filter(year(col("Date")) > 2022)

    #create model
    xgb_regressor = SparkXGBRegressor(
    features_col="features",
    label_col="Load",
    device="cuda",
    objective='reg:squarederror',
    num_round=50,
    learning_rate=params['eta'],
    max_depth=params['max_depth'],
    reg_lambda=params['lambda'],
    verbosity=0)
    #train
    xgb_model = xgb_regressor.fit(train_data)
    #predict
    predictions = xgb_model.transform(test_data)

    return predictions
```

Figure 27 Function to run the model for each of 48 predictions.

Chapter 7 - Data Mining

7.1. Logical Test Design

Training, testing, and splitting data are fundamental practices in machine learning that help to evaluate the performance of a model in a robust, reliable, and replicable way (Hastie, Tibshirani, & Friedman, 2013). The training data is used to fit and tune the parameters of the machine learning model while the testing data is used to evaluate the model performance. The training and test data splitting is set to 2:1 as we have 3 years of observations, and we want to have complete and balanced information related to time and seasonal features. Hence the first 2 (2021 and 2022) years are for training and the year 2023 data for testing. We did not include data from earlier year (2020) due to data incompleteness, also including too distant data will introduce issues like population and economic growth/decline that can compromise accuracy. After testing, we gather the performance form the error measurements to look for possible improvement.

7.2. Running the Model

7.2.1 Multi-step Train and Test Process

Since we have 48 predictions (each representing trading period) to be made each round (day), we must create a loop that produces a reshaped to daily data frames, training processes, and predictions. Before the training process is conducted, the model must be informed about features that are categorical. After finishing the loop, predictions are gathered and reshaped again to half-hourly data frames before merged with the main data frame. From this merged data frame, we can calculate the aggregated error measurements across all prediction steps. Figure 28 Shows the code that represents this process. Figure 29 shows the sample of result from initial parameter.

```
#Do predictions  for TP1 to TP48 -- First attempt
#Export each to csv to avoid loosing data in the process as it took sometime to complete all pr
features_in = ['Lag_1','Day_of_week','Month','Is_holiday','Is_holiday_prev','temp','humidity']
for i in range(1,49):
    print(f'Start training and testing for TP{i}')
    df = df_list[i]
    params={'eta': 0.3, 'lambda': 0, 'max_depth': 6}
    xgb_pred_0 = xgb_model(df,features_in,params)
    df_to_merge_0 = xgb_pred_0.select('prediction','TP','Date')
    df_to_merge_0.toPandas().to_csv(f'0_xgb_pred{i}.csv')
    print(f'Finish training and testing for TP{i}')
```

```
#add error columns and calculate overall performance
main_xgb_pred_df_0 = main_xgb_pred_df_0.dropna().withColumn("abs_error", spark_abs(col('Load') - col("prediction")))
main_xgb_pred_df_0 = main_xgb_pred_df_0.dropna().withColumn("absp_error", spark_abs(col('Load') - col("prediction")) / col('Loa
total_count = main_xgb_pred_df_0.select('prediction').count()
xgb_mape_15_0 = (main_xgb_pred_df_0.filter(col("absp_error") < 15).count())/total_count
mae_0 = main_xgb_pred_df_0.select(avg("abs_error")).first()[0]
mape_0 = main_xgb_pred_df_0.select(avg("absp_error")).first()[0]
```

Figure 28 Multi-step train and testing with error measurements.

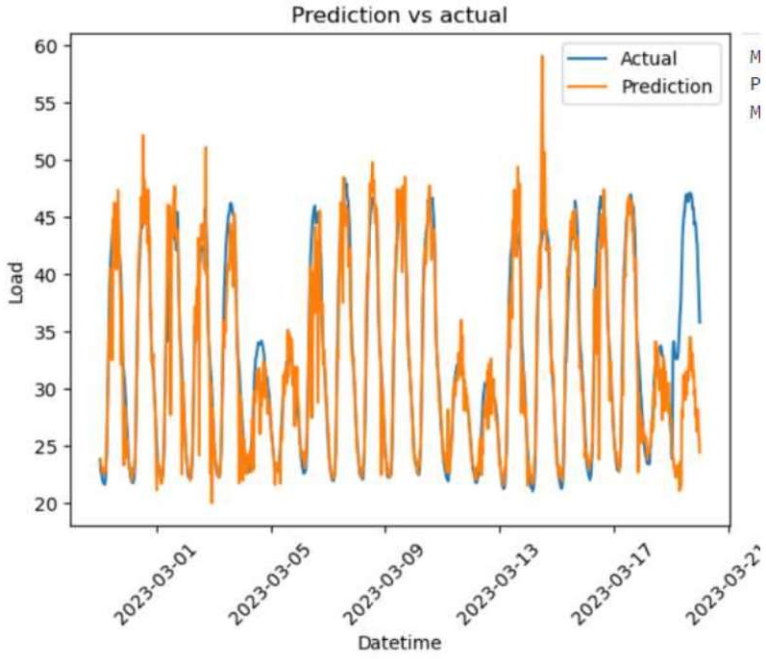


Figure 29 Plot sample of initial prediction and error measurements (without parameter tuning)

7.2.2 Computation Time

For real-time multi-step electricity forecasting, sometimes it is required to update and train models while the real time prediction is running. Reliable computation time during training phase ensure model can be refined and adapted promptly in response to changing conditions. This allows for integration of the latest data that reflects trends and anomalies to enhance accuracy and reliability of the predictions. The process of training and testing for all targets was completed in 3 hours. However, it must be noted that spark xgboost was primarily intended for distributed machine learning process with the use of multiple parallel CPUs or GPUs. Hence, the use of pyspark is only sensible if the volume of the data is big requiring multiple use or CPU's or GPU's. In this project we assume that we will utilize multiple GPU's.

7.2.3 Cross Validation and Hyperparameter tuning with *ParamGridBuilder*

We can check multiple combinations of the parameters to be used in the model to find combinations that result in the best performance. But doing this manually is inefficient. Additionally, there are 48 prediction targets with potentially different ideal parameters. *ParamGridBuilder* can help us with this process. It searches for the best parameters by conducting cross validation within the training datasets, i.e. splitting the training dataset into *k* folds of testing and training set and trying all possible combinations of parameters. The following figure shows the best parameters for all 48 targets.

```
best_params = {}
features_in = ['Lag_2','Day_of_week','Month','Is_holiday','Is_holiday_prev','temp','humidity']
for i in range(1,49) :
    train_data = df_list[i].filter(year(col("Datetime"))<=2022).dropna()
    xgb = SparkXGBRegressor(features_col=features_in, label_col="Load",  device="cuda")
    paramGrid = (ParamGridBuilder()
             .addGrid(xgb.learning_rate, [0.01, 0.1, 0.2])
             .addGrid(xgb.max_depth, [3, 5, 7])
             .addGrid(xgb.reg_lambda, [0.1, 1, 10])
             .build())
    evaluator = RegressionEvaluator(labelCol="Load", predictionCol="prediction", metricName="rmse")
    crossval = CrossValidator(estimator=xgb,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=3)
    cvModel = crossval.fit(train_data)
    best_params[i] = [cvModel.bestModel.getOrDefault('learning_rate'),cvModel.bestModel.getOrDefault('max_depth'),
                      cvModel.bestModel.getOrDefault('reg_lambda')]
#save best param to json
with open('dataset/bestparams.json', 'w') as json_file:
    json.dump(bestparams, json_file)
```

```
{1: {'eta': 0.2, 'lambda': 25, 'max_depth': 6},
 2: {'eta': 0.3, 'lambda': 25, 'max_depth': 3},
 3: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 4: {'eta': 0.3, 'lambda': 25, 'max_depth': 3},
 5: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 6: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 7: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 8: {'eta': 0.2, 'lambda': 5, 'max_depth': 3},
 9: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 10: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 11: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 12: {'eta': 0.3, 'lambda': 25, 'max_depth': 3},
 13: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 14: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 15: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 16: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 17: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 18: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 19: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 20: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 21: {'eta': 0.1, 'lambda': 1, 'max_depth': 3},
 22: {'eta': 0.1, 'lambda': 1, 'max_depth': 3},
 23: {'eta': 0.1, 'lambda': 1, 'max_depth': 3},
 24: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
```

```
bestparams[i] =  grid_search.best_params_
 25: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 26: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 27: {'eta': 0.1, 'lambda': 1, 'max_depth': 3},
 28: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 29: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 30: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 31: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 32: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 33: {'eta': 0.1, 'lambda': 1, 'max_depth': 3},
 34: {'eta': 0.1, 'lambda': 1, 'max_depth': 3},
 35: {'eta': 0.1, 'lambda': 1, 'max_depth': 3},
 36: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 37: {'eta': 0.1, 'lambda': 1, 'max_depth': 3},
 37: {'eta': 0.1, 'lambda': 1, 'max_depth': 3},
 38: {'eta': 0.2, 'lambda': 25, 'max_depth': 3},
 39: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 40: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 41: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 42: {'eta': 0.2, 'lambda': 5, 'max_depth': 3},
 43: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 44: {'eta': 0.1, 'lambda': 5, 'max_depth': 3},
 45: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 46: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 47: {'eta': 0.1, 'lambda': 25, 'max_depth': 3},
 48: {'eta': 0.1, 'lambda': 25, 'max_depth': 3}}
```

Figure 30 Best Parameters for 48 Targets determined via parameter tuning

7.3. Model's Output Patterns

As can be seen in the Figure 29, the model can fit the actual values quite well. First, it recognizes the hourly trends, where it consists of two low points and one peak point for each day. Second, it follows the weekly trajectory as well (notice the two lower peak points occurred after 5 high points, which represent weekend days). However, we see the prediction line fluctuates sharply more than the actual values which look smoother. This could be a sign of overfitting. By applying more regularization via lambda and tuning parameters like learning rate and tree depth, this can be minimized.

7.3.1 Result with Using Parameter Tuning and Multiple time-lag Features (second scenarios)

As previously mentioned, the parameter tuning produces the best parameters for each of 48 targets. Figure 30 shows how different target can have different best parameters. For the parameter *max_depth,* almost all targets use the lowest possible value 3 except for TP1 where it uses 6 instead. There are variances of the parameter *lambda*, with the majority using 5 or 25 which are the higher side. Lastly, almost all targets use the low value for *eta* 0.1. We can see the patterns of the parameters used tend to go with the side that produce more regularization, hence minimizing the overfitting. We also use the second scenario for the time-lag features, so now we use two time-lag features. Applying these changes producing smoother lines which look more similar to the actual values (see Figure 31 compared to Figure 29). Additionally, there are improvements in error measurements.

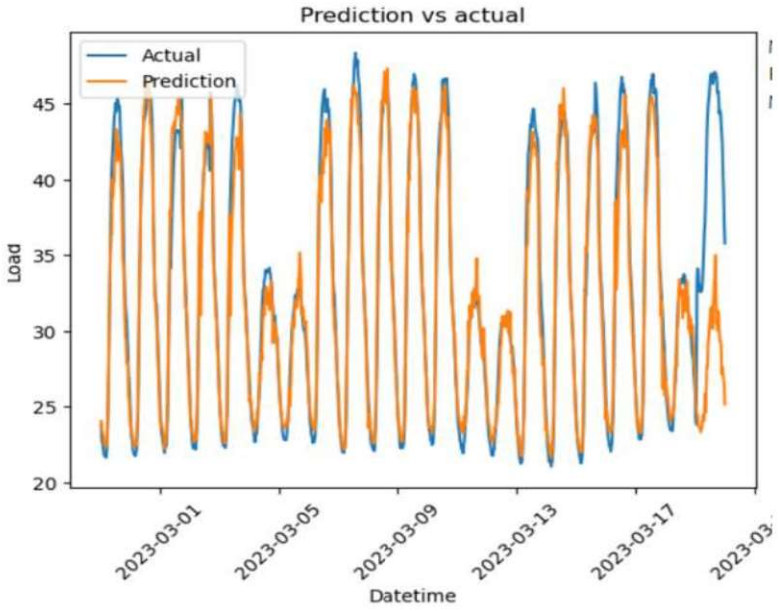


Figure 31Prediction samples using the best parameters with parameter tuning

7.3.2 Error Patterns

To evaluate the performance of our model we have to look at the measurements error. But visually we can see some pattern of error that reflects model's performance. As can be observed from Figure 31, the error is relatively high for trading periods with extreme (low and high) demand but most notably on the peak points. After inspecting other months, we found the same pattern. Hence this must be addressed when evaluating the model (next chapter). From the business objective perspective, we have defined our performance with MAE and MAPE error measurements. Figure 32 summarizes the error measurements of our model. We can see that although the MAE is satisfied (less than 5MW), the P[MAPE<15%] are lower than 80%. So overall, the current model has not satisfied the business objective.

```
print(f'MAPE : {mape_0_1}') #Mean average percentage error
print(f'Proportion APE<15% : {xgb_mape_15_0_1}') #Proportion of absolute percentage error less than 15%
print(f'MAE : {mae_0_1}')#Mean Absolute Error
```

```
MAPE : 0.26319963703348653
Proportion APE<15% : 0.7884132420091324
MAE : 3.1687566498234956
```

Figure 32 Error Measurements Summary

7.3.3 Output Documentation

The prediction for each prediction target was done separately with their own data frames. Hence, to have the aggregated performance of our model, we have to integrate outputs from all prediction steps. This process involves merging the predictions with date and TP as the merging keys in the main data frame. We put the training and testing predictions in different columns. After the integration, we can evaluate the aggregated error measurements for testing data. Figure 33 shows the updated main data frame.

```
-----------------+------------------+--------------------+
|      Date| TP|          Datetime|idx|    time| Load|Day_of_week|Month|Is_weekend|Is_holiday|temp|humidity|Is_holiday_prev|
prediction|         abs_error|          absp_error|
+----------+---+------------------+---+--------+-----+-----------+-----+----------+----------+----+--------+---------------+-
-----------------+------------------+--------------------+
|2023-01-01|  1|2023-01-01 00:00:00|  0|00:00:00|11.45|          1|    1|         1|         1|18.9|   84.06|              0|
11.84469985961914|0.39469985961914134| 0.03447160346018702|
|2023-01-01|  2|2023-01-01 00:30:00|  1|00:30:00|11.91|          1|    1|         1|         1|18.9|   84.06|              0|1
2.347430229187012| 0.4374302291870116|0.036727978940974945|
|2023-01-01|  3|2023-01-01 01:00:00|  2|01:00:00|12.05|          1|    1|         1|         1|18.8|   83.99|              0|1
3.235966682434082| 1.1859666824340813| 0.09842047157129305|
|2023-01-01|  4|2023-01-01 01:30:00|  3|01:30:00|12.16|          1|    1|         1|         1|18.8|   83.99|              0|1
```

Figure 33 Main data frame updated with predictions and error measurements.

Chapter 8 - Interpretation

8.1 Data Mining Result Discussion

As shown in the previous chapter the prediction model performed reasonably well but still does not yet satisfy our business criteria (Figure 32). We will look at the prediction performance across different features to ensure that it reflects the result from exploratory and descriptive analysis previously done and identify potential improvement in the model. Before that we look at feature importances first. Figure 34 shows a sample of feature importances (TP20). As expected, the time-lag feature is the main predictor. *Day_of_week* is the next important predictor. This is understandable, as the model recognized that the peaks for certain days (weekend days) are lower than others, shown and discussed in previous section. Weather features: temperature and humidity, are also important predictors, more than Month. The Month feature may not provide additional predictive value as the weather already captures the load pattern. Which give us the option to drop one of these features, in addition to avoid multicollinearity. Lastly *the Is_holiday* and *Is_holiday_prev* have relatively low importance. This is because of the low occurrence of holiday. ML models typically learn better from features that have a wide range of variations and occur frequently enough to establish a clear pattern.

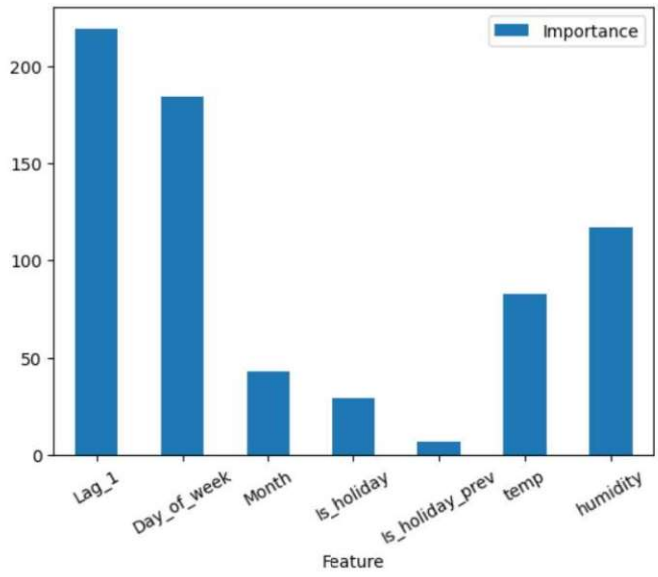


Figure 34 Feature Importance

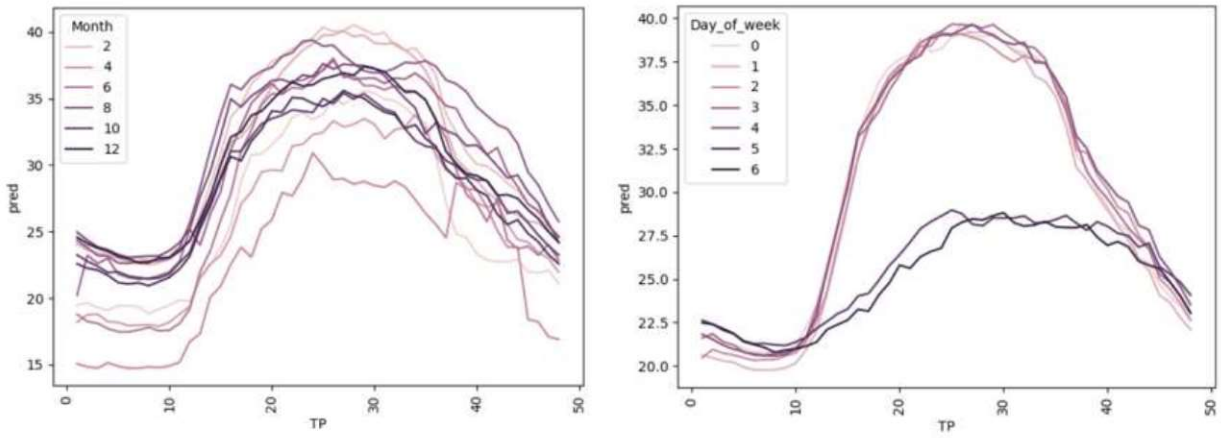


Figure 35 Prediction values across TP average on.

Figure 35 shows the spread of predicted values across *TP,* averaging on month and day of week, reflects similar pattern with the actual values as shown previously in Chapter 2.

8.2 Visualizations for Model Analysis

8.2.1 Prediction Results

We present visualizations of predictions vs actual in different scales. From analyzing these visualizations, we can see how well the model predicts consumption and where it tends to deviate. Plotting with different scales can also reveal consumption patterns and help assess whether the model can recognize these patterns.

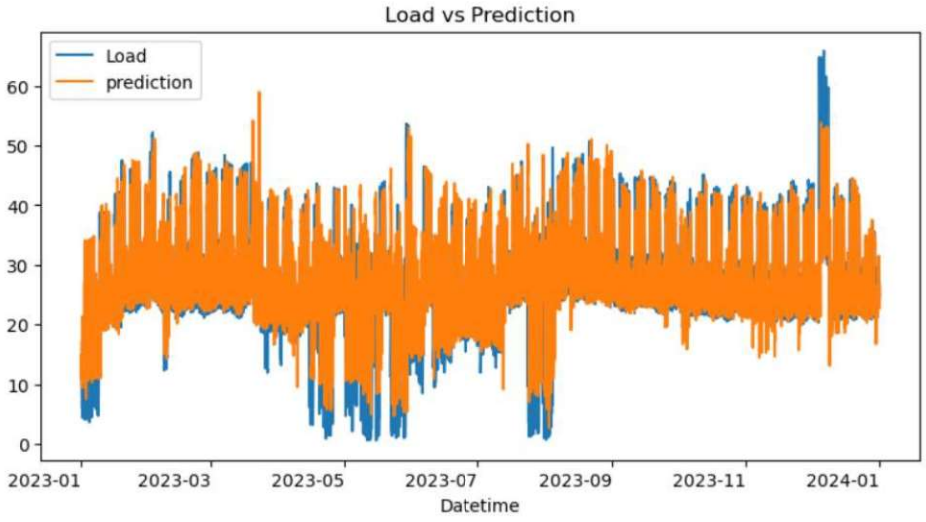


Figure 36 Predictions vs actual yearly scale

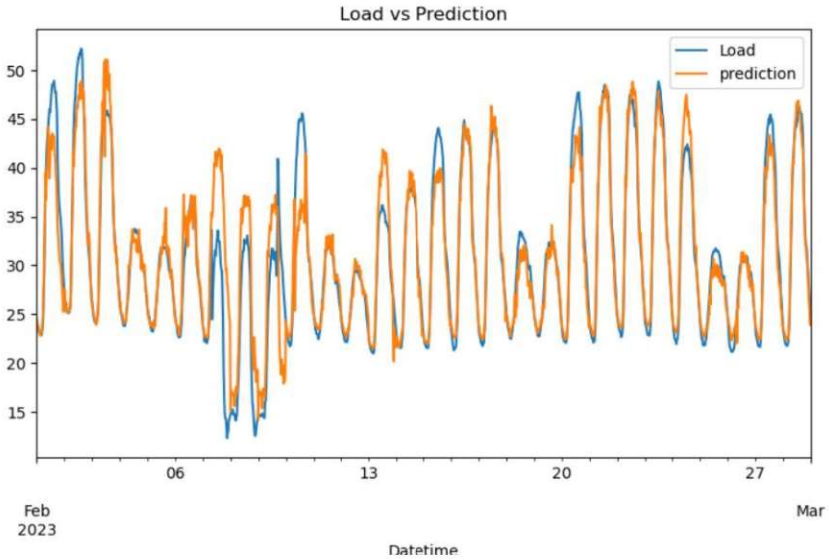


Figure 37 Predictions vs actual: monthly scale

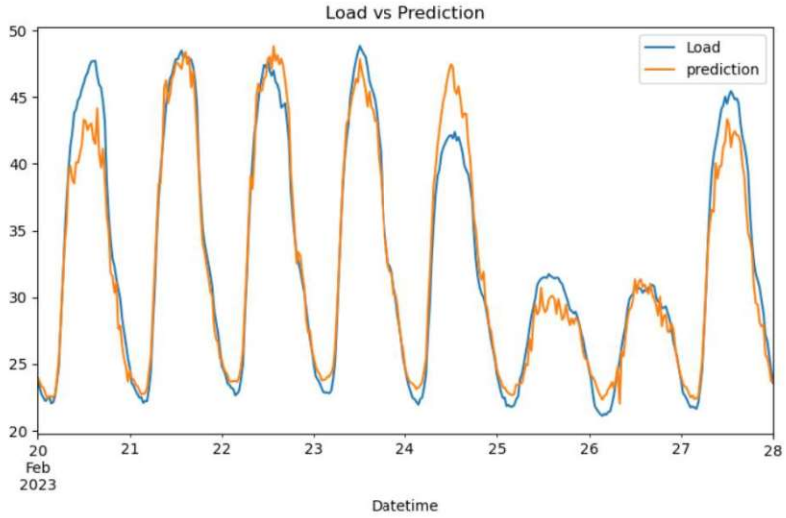


Figure 38 Predictions vs actual: weekly-daily scale.

8.2.1 Prediction Errors Exploration

In this section we will look at visualizations on prediction error at different features to analyze how much certain values contribute to the overall error. This can help in interpreting the model's behavior and identifying potential biases. Moreover, it can guide feature engineering efforts by highlighting which features are most impactful for improving model performance. We first start with the histogram on the overall error, followed by error distributions on different features.

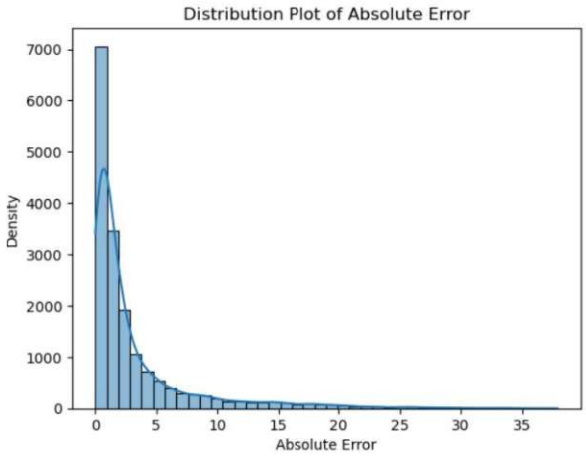


Figure 39 Absolute prediction error histogram.

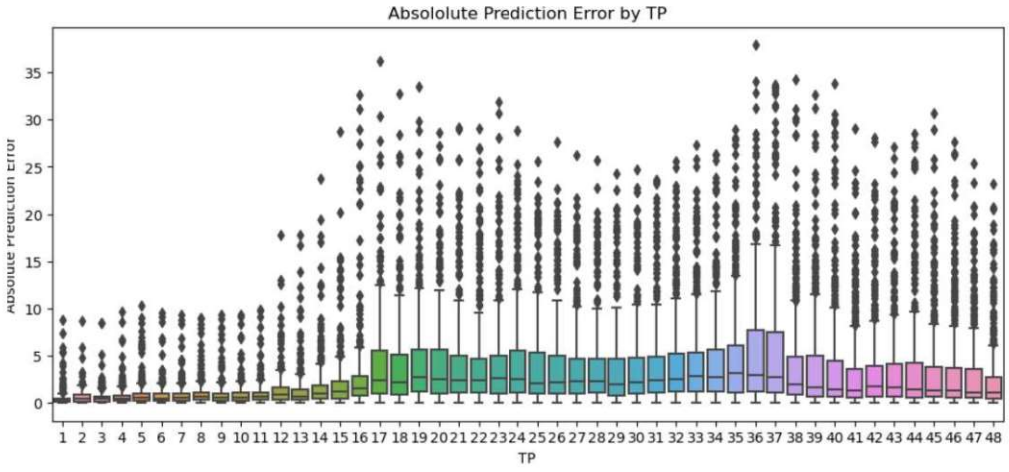


Figure 40 Absolute prediction error on trading periods.

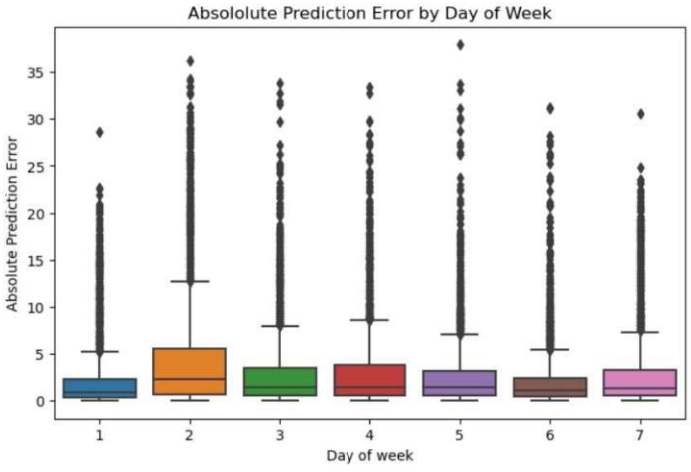


Figure 41 Absolute prediction error on different days.

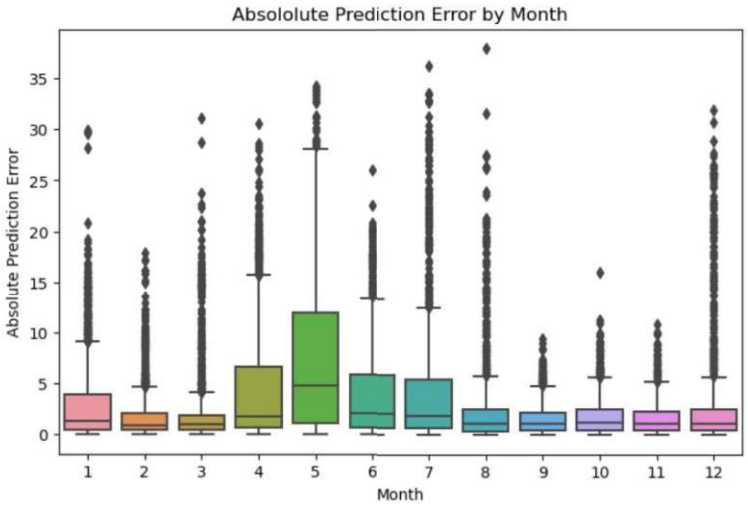


Figure 42 Absolute prediction error on different months.

8.3 Interpretation

8.3.1 Prediction Result vs Actual

Figure 36 shows that the predictions perform reasonably well recognizing the seasonal patterns over the year. However, for certain months it tends to over forecast for periods with lower consumption. From weekly perspective (Figure 37 and Figure 38), the model can recognize weekend days where it produces lower predictions consistent with the actual values. The model also can recognize the daily patterns where it has lower values in the midnight to morning and peaking at around afternoon (Figure 38). Despite the ability to capture this pattern, performance measure is not yet satisfied.

From Figure 38 we see that the prediction shows more fluctuations than the actual values which are smoother. The irregularities of the prediction in comparison to the actual values can be a sign of overfitting. We have previously addressed this issue by using parameter tuning to find the most suitable parameters including lambda values that can overcome overfitting. Although these irregularities are minimized now, they are still noticeable. To further obtain the best parameters more accurately, we can use more advanced tools parameter tuning such as Bayesian Optimization (Nguyen, 2019). However, this process is resource intensive and might not be practical for this project.

8.3.1 Prediction Error Exploration

Figure 39 shows that our absolute prediction error distributions seem to follow gamma distribution where the mode is less than 1 MW which is far less than the objective which is 5 MW. There are some extreme prediction error values but with very small frequency. Looking from trading period perspective (Figure 40), prediction errors seem to increase as TP increases but most notably on TP's where consumptions are at the peak. This is understandable because the further the target from the current point, the harder it is to predict. The higher error on the peak TPs is also explainable as bigger magnitude also produces bigger absolute error. To improve prediction performance on these targets we can consider scenario 2 for the time-lag features, use two time-lag features: last value and value of the same time from the previous day.

The error distribution over the different days of the week is more even than other features. We have discussed that the model can recognize the weekly pattern. However, we can notice that predictions on Monday (day of week =2) seem to have higher error, compared to the other days possibly due to a change of consumption behavior from the weekend to the workweek. Moreover, the time-lag feature, which is the most important predictor, is actually taken from weekend observation (Sunday).

The predictions error on the month feature seems to be uneven with months 4 to 7 tending to have higher error. Aligned with the multicollinearity issue previously stated, we have the option to drop the month feature and use the weather features instead.

8.4 Model Evaluation

In this project we use two error measurements, absolute error and absolute percentage error. The absolute error is used because of its direct effect on plant operation, while the percentage error is a relative measure that reflects prediction consistency on different magnitude. As defined in the business objective the model must be able to predict with a mean absolute error of 5MW and 80% probability of prediction within range ±15% of actual. Additionally, from time computation perspective the model must conduct the training and testing process within reasonable duration to anticipate for model update in the real-time prediction environment. Table 6 summarize the model evaluation. Since not all of the metrics are fully satisfied, we need to improve the model.

Table 6 Model Evaluation Summary

	Metrics		
	MAE	P(MAPE<15%)	Computation Duration
Target	<5	>80%	Reliable
Performance	2.73	77%	Reliable
Achieved?	Yes	No	Yes

8.5 Multiple Iteration

8.5.1 Improve Current Model

Summarizing our analysis in 8.2 and 8.3 we can do the following for model improvement:

1. Use multiple time-lag features.
2. Drop the Month feature and use the most important feature in weather data.
3. Use more advanced parameter tuning.

For point 1 we have previously added 2 time-lag features, now we can use both for the improved model. For point number 2, we can easily apply this by slightly modify the code, so we have options to drop any feature in the dataset for XGBoost model (see Figure 43). For point number 3, due to the computation cost, it is difficult to apply it to this project.

```
#Prediction function
def xgb_model(df,features,params):
    data = df.dropna()
    assembler = VectorAssembler(inputCols=features, outputCol="features")
    data = assembler.transform(data)
    #split
    train_data = data.filter(year(col("Date")) <= 2022)
    test_data = data.filter(year(col("Date")) > 2022)

    #create model
    xgb_regressor = SparkXGBRegressor(
```

Model XGB3: Time-lag Feature Scenario 2 With Parameter Tuning and Drop Month and Humidity

```
#Do predictions for TP1 to TP48 (Use multiple time-lag features and use best parameters)
#Export each to csv to avoid loosing data in the process as it took sometime to complete all
features_in = ['Lag_1','Lag_2','Day_of_week','Is_holiday','Is_holiday_prev','temp']
with open('dataset/bestparams.json', 'r') as json_file:
    bestparams = json.load(json_file)
for i in range(1,49):
    print(f'Start training and testing for TP{i}')
    df = df_list[i]
    params=bestparams[f'{i}']
    xgb_pred = xgb_model(df,features_in,params)
    df_to_merge = xgb_pred.select('prediction','TP','Date')
    df_to_merge.toPandas().to_csv(f'xgb_pred{i}.csv')
    print(f'Finish training and testing for TP{i}')
```

Figure 43 Modify model function and feature drop before loop.

Applying this improvement to our model results in better performance especially in terms of the mean absolute percentage error. Figure 44 shows sample of predictions and error measurements of the improved model. Table 7 shows that the improved model has now better performance and achieved all metrics target.

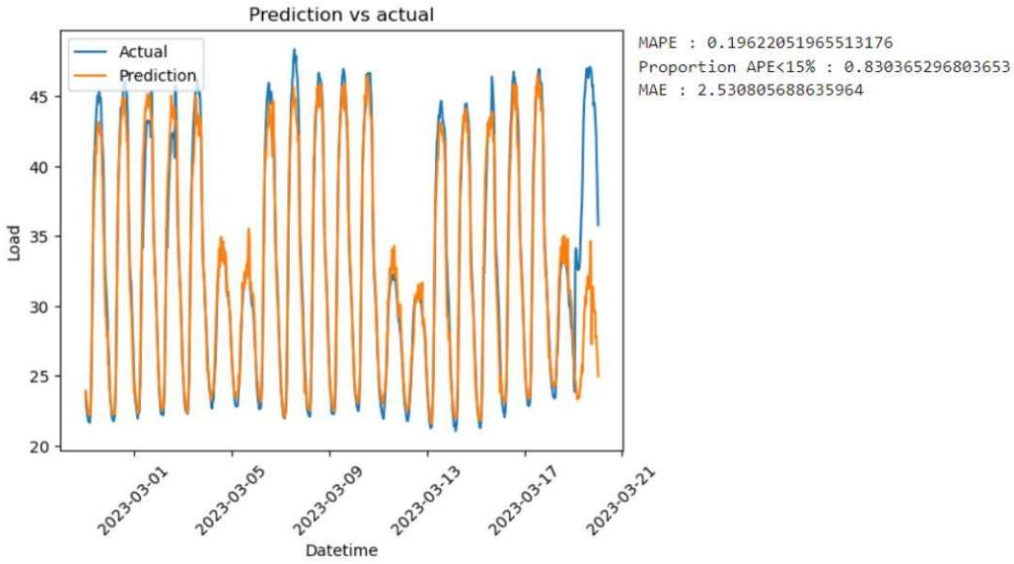


Figure 44 Prediction samples and error measurements in Improved model.

Table 7 Improved Model Evaluation Summary

	Metrics		
	MAE	P(MAPE<15%)	Computation Duration
Target	<5	>80%	Reliable
Performance	2.53	83%	Reliable
Achieved?	Yes	Yes	Yes

8.5.2 Models with Reduced Features

In this section we step further back to previous processes in an attempt to find better prediction performance or find methods/algorithms that can simplify the prediction process. The purpose of simplification is to reduce the risk of our real time prediction workflow from unanticipated events. For example, we can consider dropping the weather features which were taken from a third-party provider. We cannot guarantee that this third party will always provide fast and reliable data that suits our real-time prediction process. Additionally, we know that the month feature basically gives similar

information to our model due to multicollinearity. By not including the weather data, we have to slightly change the way we conduct data preparation to a simpler process.

To further test the robustness of the model we compared it with another type of algorithm that is suitable with time-series problem. As mentioned in chapter 5 and 6, LSTM is a suitable algorithm for this problem. This algorithm has been widely used in time series forecasting. However, since spark offers a very limited time-series model, and doesn't specifically have a library for LSTM model, we look for other algorithm that can be utilized for time-series forecasting. Within the limited option available in spark, there are three main algorithms suitable for tabular regression type problems: Linear Regression, Decision Three Regression, and Random Forest Regression (spark link). We picked Random Forest Regression because it can handle nonlinearity (as our problem has, shown by Month and TP vs Load) and perform better in terms of accuracy compared to Decision Trees, as it aggregates multiple trees. The main step with this algorithm is generally similar to the steps previously done. For report simplification we do not present the whole process, instead we show the summary shown in Figure 45.

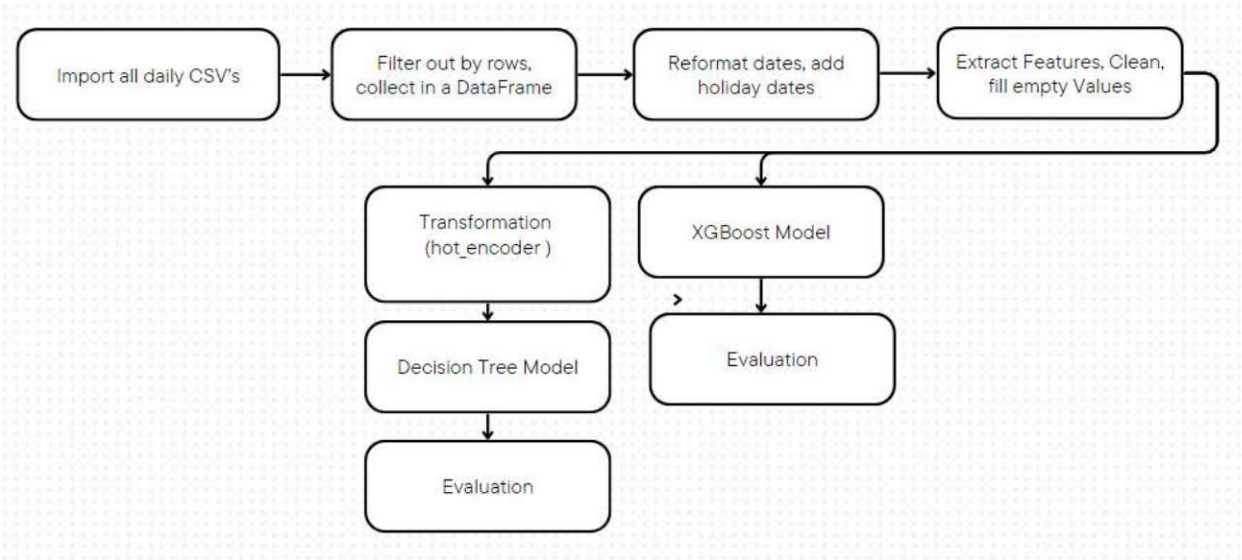
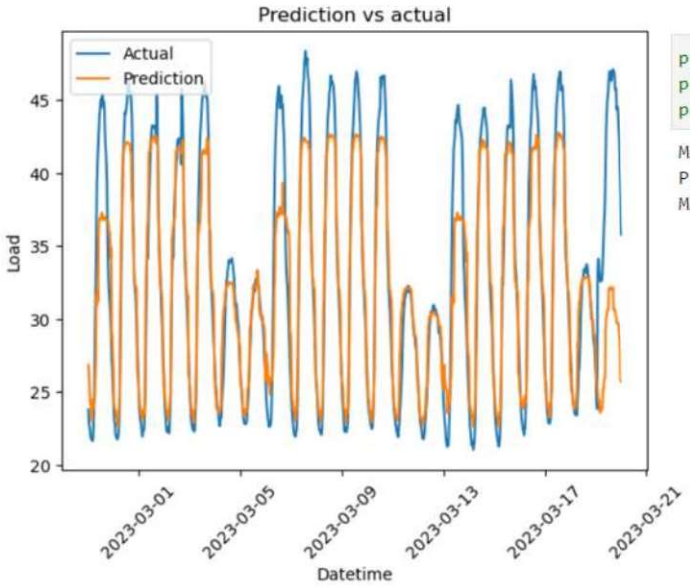


Figure 45 Steps in the reiteration.

8.5.2.1 Random Forest Model

For this algorithm we use both time-lag features as continuous variables and the same categorical features that we used in XGBoost model. We increase the default model parameters (such as number of trees and max depth), using generally higher parameter values as it tends to underfit due to the ensemble nature. The training and predicting process took slightly less computation time than XGBoost. As shown in Figure 46, RF model can recognize daily and weekly patterns. However, it seems to underpredict peak values and overpredict low values. The error measurements are also significantly higher than the previous model.




```
print(f'MAPE : {rf_mape}') #Mean average percentag
print(f'Proportion APE<15% : {rf_mape_15}') #Propo
print(f'MAE : {rf_mae}')#Mean Absolute Error
```

```
MAPE : 0.27610125677149505
Proportion APE<15% : 0.7278538812785388
MAE : 3.798403351505818
```

Figure 46 Sample of prediction result with LSTM and error measurements.

Table 8 Performance Evaluation on LSTM Model

	Metrics		
	MAE	P(MAPE<15%)	Computation Duration
Target	<5	>80%	Reliable
Performance	3.80	73%	Reliable
Achieved?	Yes	No	Yes

8.5.2.2 Result with XGBoost without Weather Data

We can simulate this iteration by using the same code in the primary XGBoost model by dropping temperature and humidity features. As shown in Figure 47, this model can recognize weekly and daily

patterns and just perform slightly worse from our primary XGBoost model (with weather features) on the performance metrics.

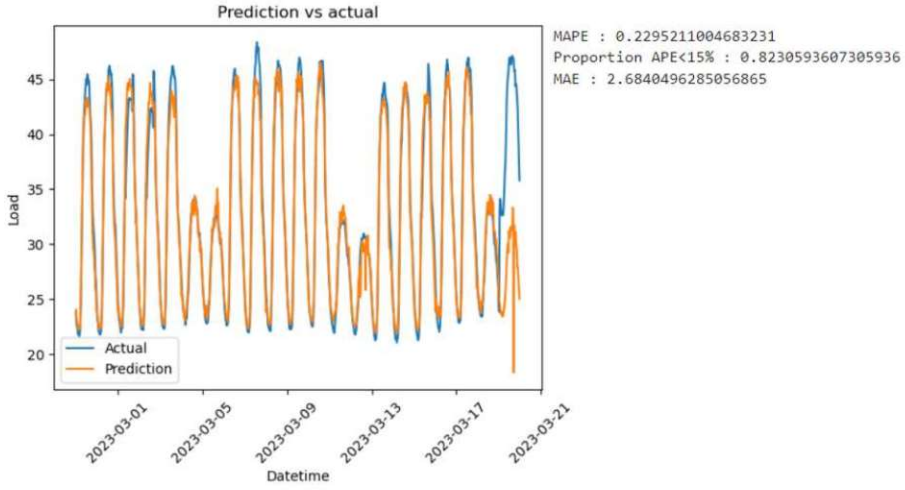


Figure 47 Sample of prediction result using XGBoost without weather data and error measurements

Table 9 Performance Evaluation on XGBoost Model without weather features

	Metrics		
	MAE	P(MAPE<15%)	Computation Duration
Target	<5	>80%	Reliable
Performance	2.68	82%	Reliable
Achieved?	Yes	Yes	Yes

8.5.2.3 Discussion on the Final Model

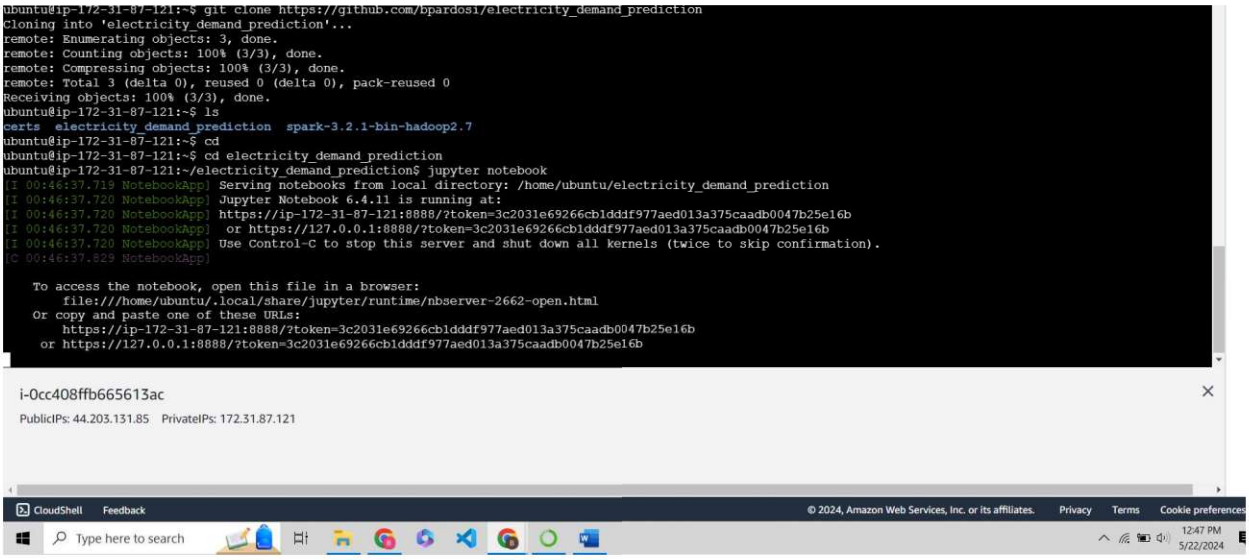
From our analysis in chapter 5 and 6, we have chosen XGBoost as the algorithm for our multistep forecasting problem. Initially this model generates predictions with errors outside our business specification. To overcome this, we use parameter grid search to find the most suitable parameters for each prediction target and remove the month feature to avoid multicollinearity. These adjustments improve our model, which eventually satisfies our business criteria. Then we consider a scenario where weather data is not reliable. We ran this scenario with XGBoost and a new algorithm, Random Forest. Upon evaluating their performance, we can see that XGBoost perform significantly better in our case. It is also observed that without weather data, our model performance decreased slightly but still satisfying business criteria. Based on these findings, we have the option to either include or exclude weather data from our model. Incorporating weather data has the

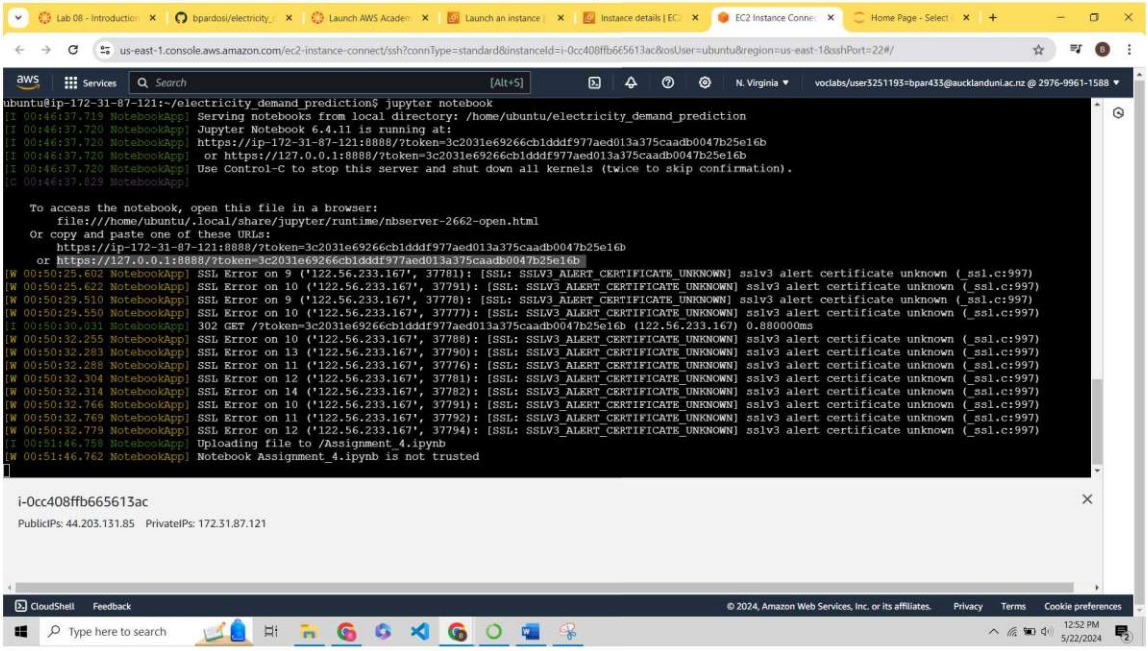
potential to slightly enhance prediction accuracy, provided that the weather data is accurate. However, given that we cannot guarantee the accuracy of the weather data, using the model without it is the best option.

ATTACHMENT – Github AWS EC2 Documentation

This is a documentation of doing my project with Github and Jupyter Notebook in EC2 virtual machine. Please note that the modelling parts take quite a lot of time. So, I continuously save the predictions to csv files during this process for all different prediction targets and different models. So, to check the prediction result and see the visualizations, you don't need to evaluate the modelling cells, you just need to evaluate the cells where it says, "retrieve multiple prediction target, merge to a df". You can evaluate the modelling cell, just make sure the necessary packages are installed.

Cloning Github Reps and add necessary files





```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .ipynb_checkpoints/
        Assignment_4.ipynb

nothing added to commit but untracked files present (use "git add" to track)
ubuntu@ip-172-31-87-121:~/electricity_demand_prediction$ git add Assignment_4.ipynb
ubuntu@ip-172-31-87-121:~/electricity_demand_prediction$ git commit
Aborting commit due to empty commit message.
ubuntu@ip-172-31-87-121:~/electricity_demand_prediction$ git commit -m "added files'
> ^C
ubuntu@ip-172-31-87-121:~/electricity_demand_prediction$ git commit -m "added files"
[main 8db255e] added files
 Committer: Ubuntu <ubuntu@ip-172-31-87-121.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author
```

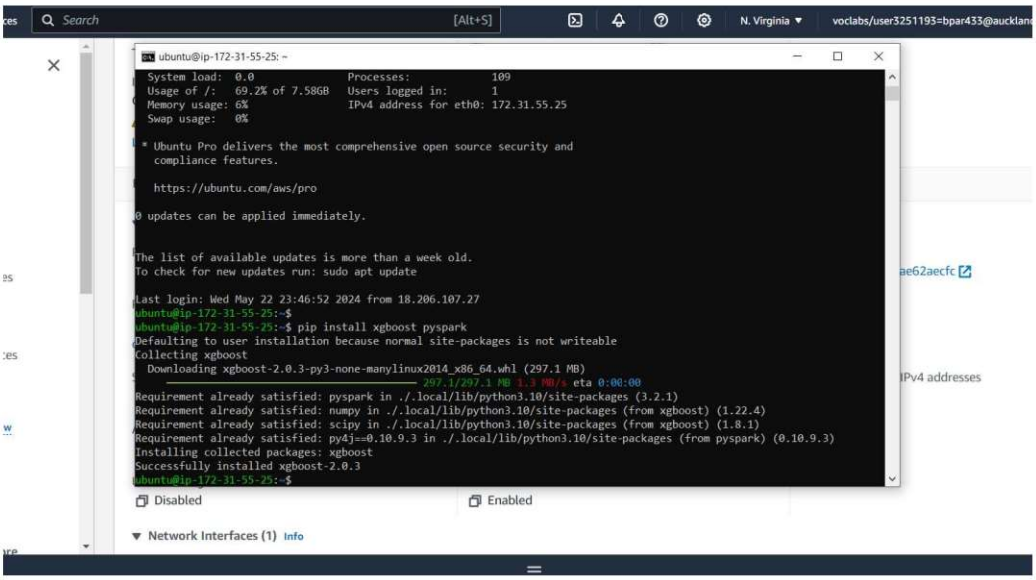
```
ubuntu@ip-172-31-87-121:~/electricity_demand_prediction$ git push
Username for 'https://github.com': bpardosi
Password for 'https://bpardosi@github.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1.62 MiB | 6.41 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/bpardosi/electricity_demand_prediction
   fff6a75..8db255e  main -> main
ubuntu@ip-172-31-87-121:~/electricity_demand_prediction$
```

```
W 01:45:35.889 NotebookApp] SSL Error on 14 ('122.56.233.167', 40170): [SSL: SSLV3_ALERT_CERTIFICATE_UNKNOWN
I 01:45:48.561 NotebookApp] Creating new directory in
I 01:46:04.249 NotebookApp] Creating new directory in
I 01:46:37.308 NotebookApp] Uploading file to /dataset/bestparams.json
I 01:46:38.201 NotebookApp] Uploading file to /dataset/raw_load_df_2023.csv
I 01:46:38.820 NotebookApp] Uploading file to /dataset/raw_load_df_2022.csv
I 01:46:41.464 NotebookApp] Uploading file to /dataset/raw_load_df_2021.csv
I 01:46:43.212 NotebookApp] Uploading file to /dataset/auckland_weather_DP_output1.csv
I 01:47:01.230 NotebookApp] Uploading file to /predictions/predictions.zip
I 01:47:13.482 NotebookApp] Shutting down on /api/shutdown request.
I 01:47:13.483 NotebookApp] Shutting down 0 kernels
I 01:47:13.483 NotebookApp] Shutting down 0 terminals
buntu@ip-172-31-62-132:~/electricity_demand_prediction$ git commit
n branch main
our branch is up to date with 'origin/main'.

ntracked files:
 (use "git add <file>..." to include in what will be committed)
       dataset/
       predictions/

othing added to commit but untracked files present (use "git add" to track)
buntu@ip-172-31-62-132:~/electricity_demand_prediction$ git add dataset/auckland_weather_DP_output1.csv
buntu@ip-172-31-62-132:~/electricity_demand_prediction$ git add dataset/bestparams.json
buntu@ip-172-31-62-132:~/electricity_demand_prediction$ git add dataset/raw_load_df_2021.csv
buntu@ip-172-31-62-132:~/electricity_demand_prediction$ git add dataset/raw_load_df_2022.csv
buntu@ip-172-31-62-132:~/electricity_demand_prediction$ git add dataset/raw_load_df_2023.csv
buntu@ip-172-31-62-132:~/electricity_demand_prediction$ git add predictions/predictions.zip
```

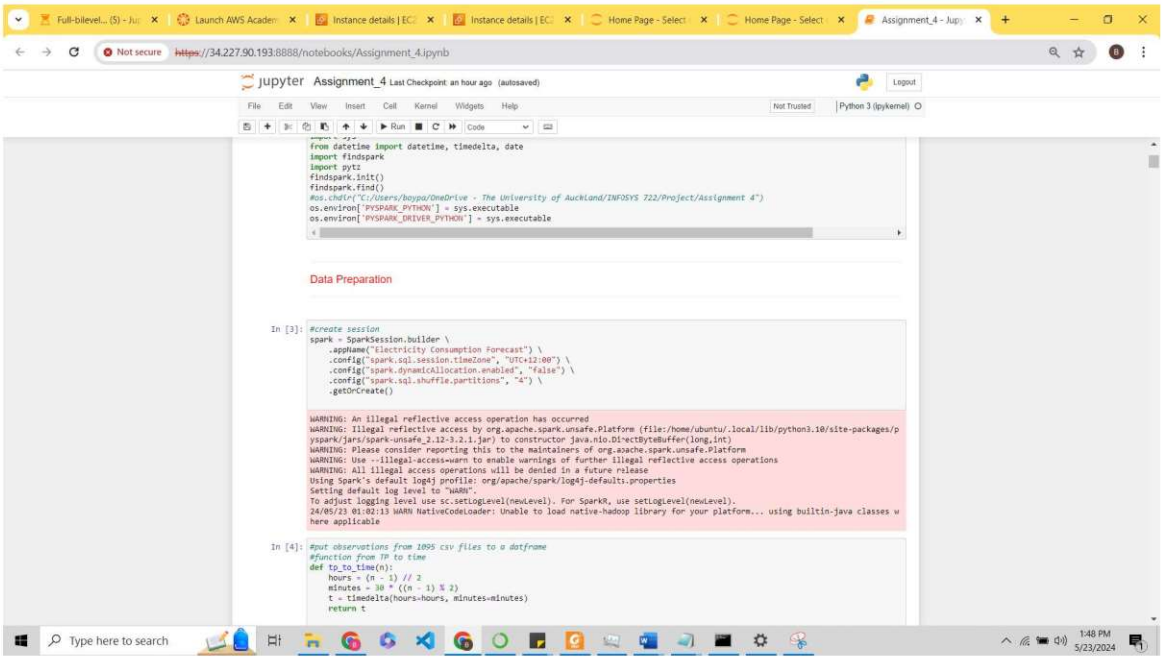
```
ubuntu@ip-172-31-62-132:~/electricity_demand_prediction$ git add predictions/predictions.zip
ubuntu@ip-172-31-62-132:~/electricity_demand_prediction$ git commit -m "added files"
[main 5b447df] added files
 Committer: Ubuntu <ubuntu@ip-172-31-62-132.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 6 files changed, 78845 insertions(+)
 create mode 100644 dataset/auckland_weather_DP_output1.csv
 create mode 100644 dataset/bestparams.json
 create mode 100644 dataset/raw_load_df_2021.csv
 create mode 100644 dataset/raw_load_df_2022.csv
 create mode 100644 dataset/raw_load_df_2023.csv
 create mode 100644 predictions/predictions.zip
ubuntu@ip-172-31-62-132:~/electricity_demand_prediction$ git push
Username for 'https://github.com': bpardosi
Password for 'https://bpardosi@github.com':
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 2 threads
```

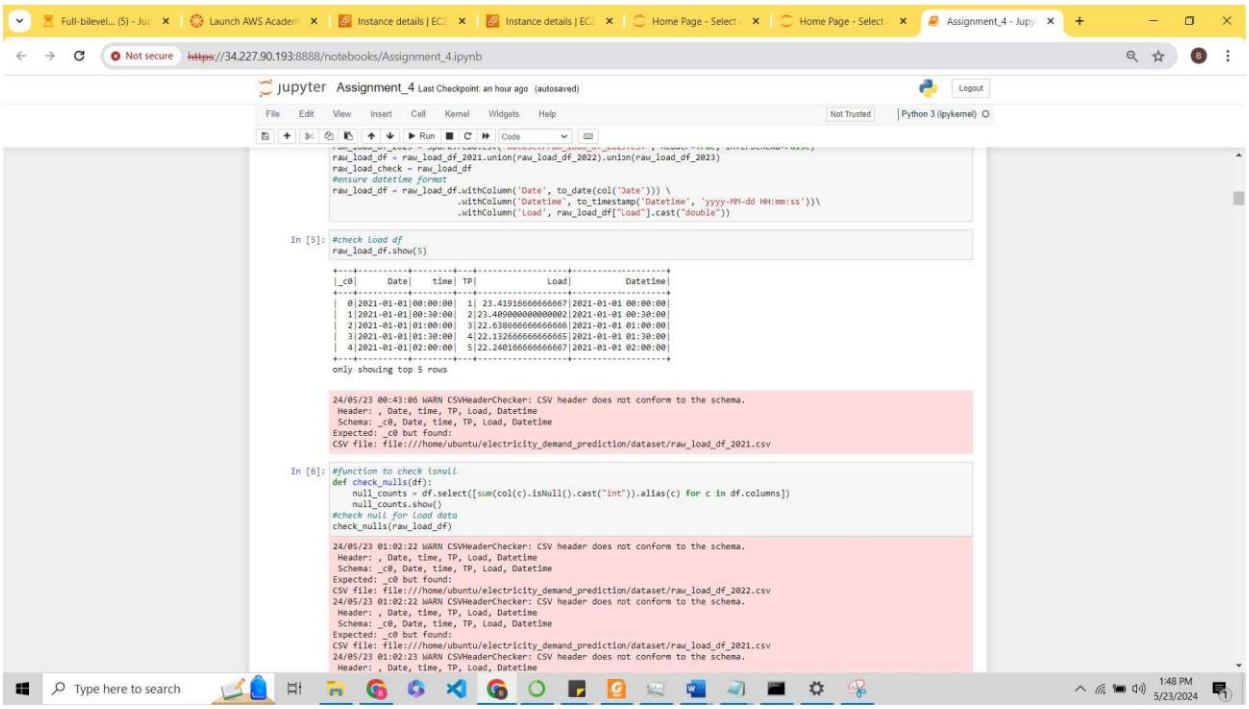
```
ubuntu@ip-172-31-62-132:~/electricity_demand_prediction$ git push
Username for 'https://github.com': bpardosi
Password for 'https://bpardosi@github.com':
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 2 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (10/10), 2.31 MiB | 6.25 MiB/s, done.
Total 10 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/bpardosi/electricity_demand_prediction
   8db255e..5b447df  main -> main
ubuntu@ip-172-31-62-132:~/electricity_demand_prediction$
```

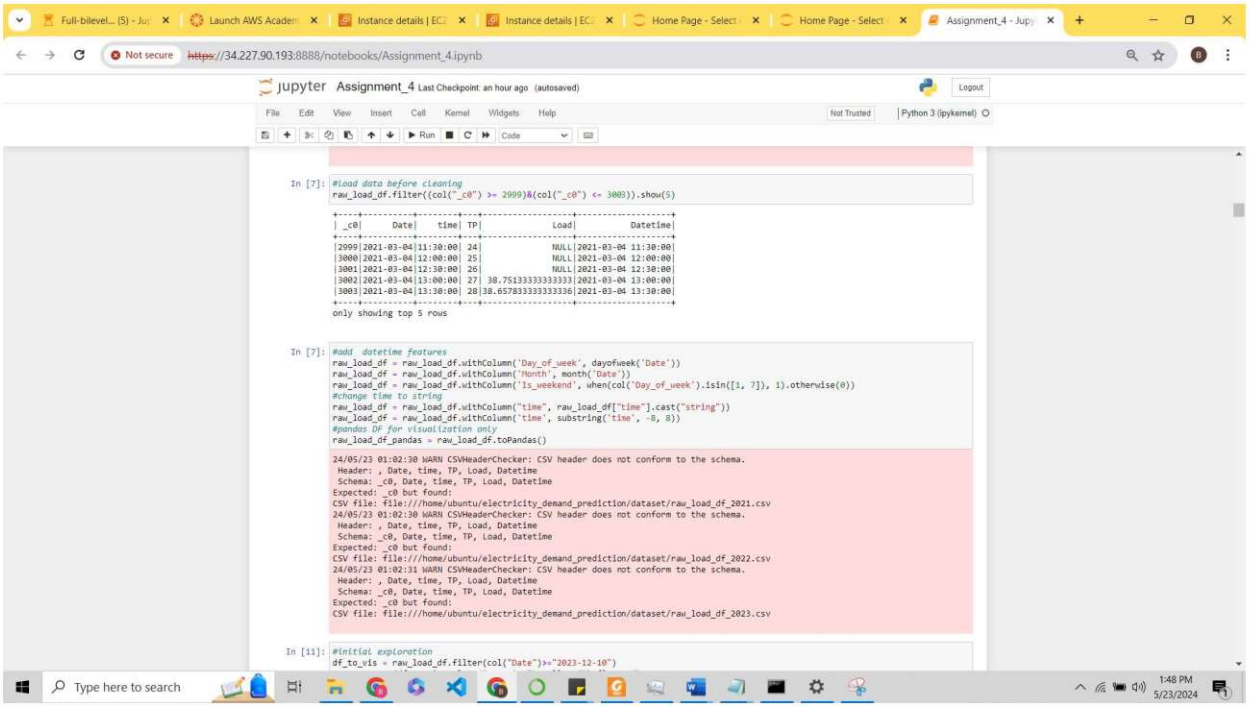
Install Necessary Packages for ML Modelling to EC2 Machine

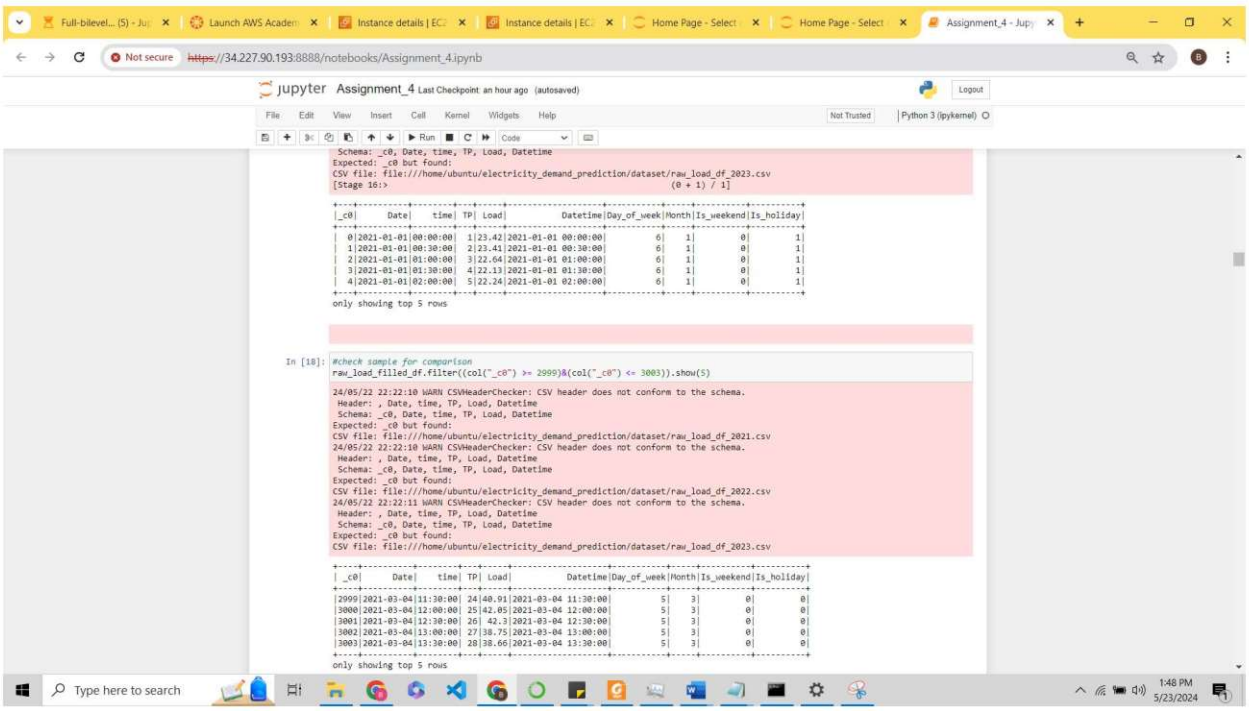


Running The Codes in EC2 with Jupyter Notebook









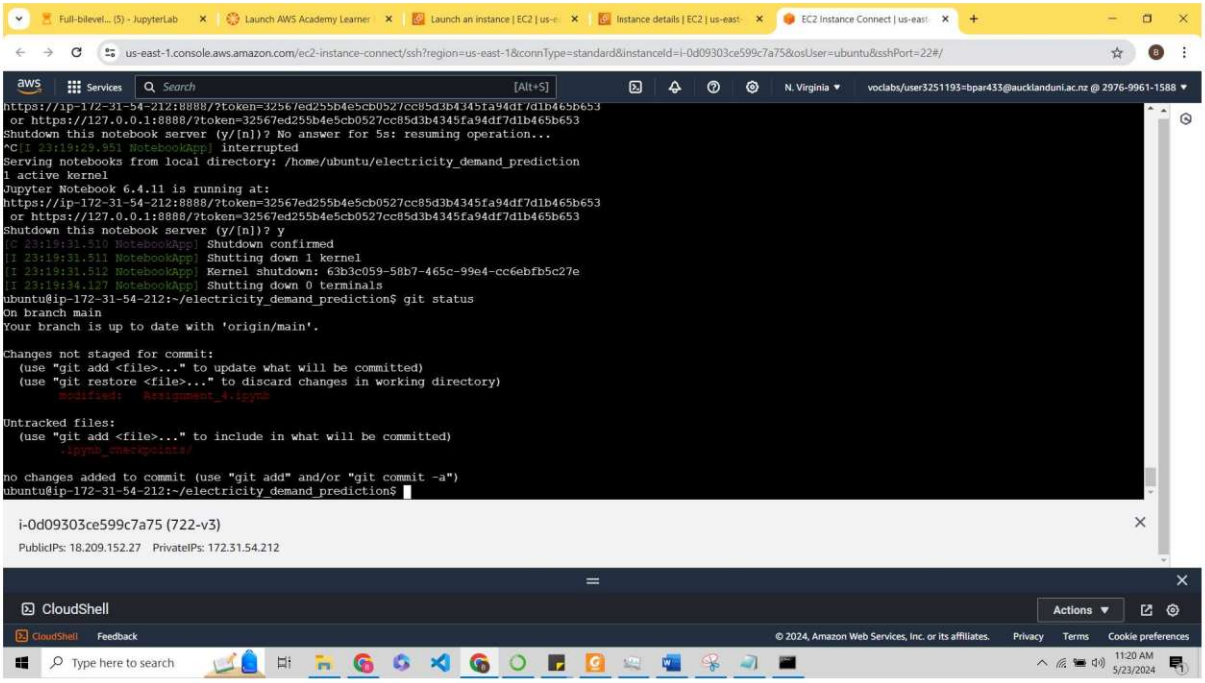
Running the Model in EC2 with Jupyter Notebook

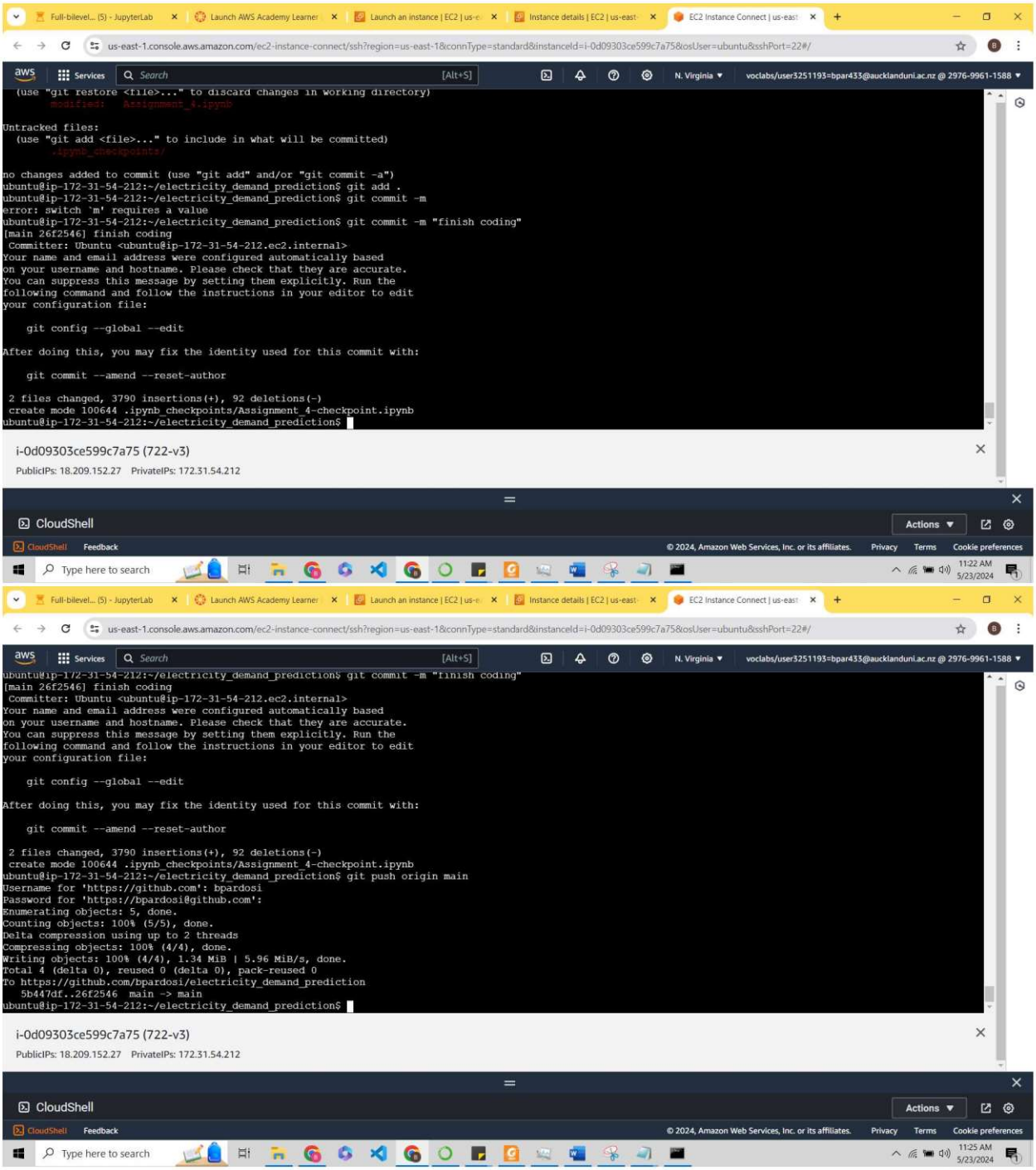
Model XGB1: Time-lag Feature Scenario 1 and Without Parameter Tuning

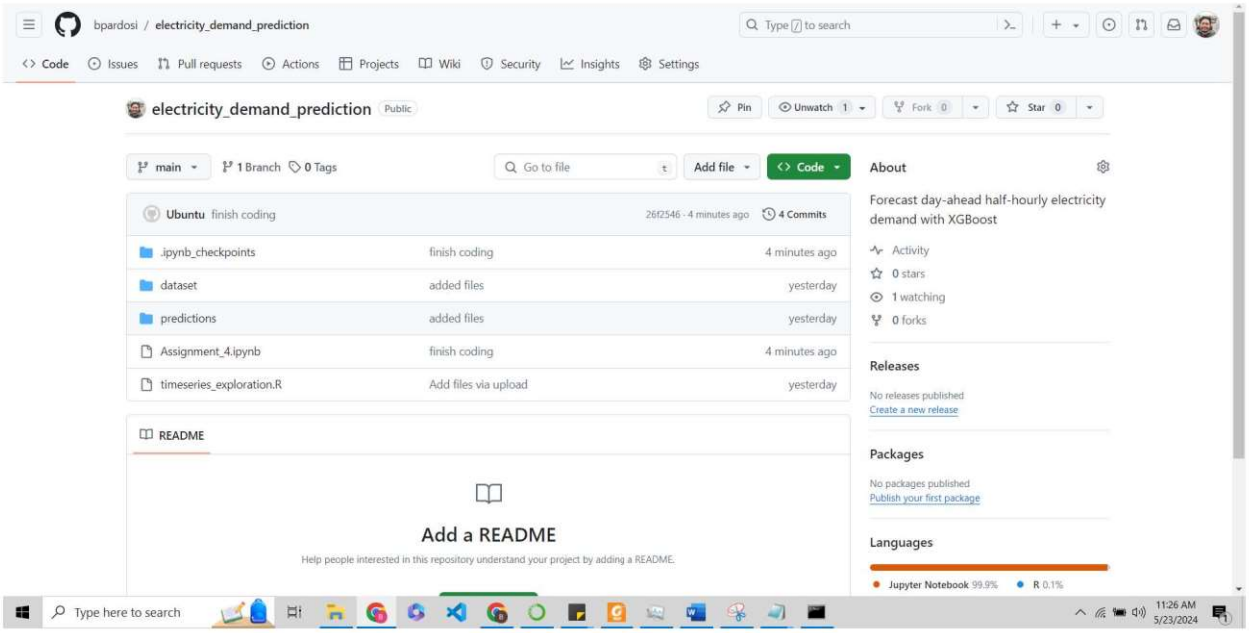
```
In [*]: #Do predictions  for TP1 to TP48 -- First attempt
        #Export each to csv to avoid loosing data in the process as it took sometime to complete all prediction target
        features_in = ['Lag_1','Day_of_week','Month','Is_holiday','Is_holiday_prev','temp','humidity']
        for i in range(1,49):
            print(f'Start training and testing for TP{i}')
            df = df_list[i]
            params={'eta': 0.3, 'lambda': 0, 'max_depth': 6}
            xgb_pred_0 = xgb_model(df,features_in,params)
            df_to_merge_0 = xgb_pred_0.select('prediction','TP','Date')
            df_to_merge_0.toPandas().to_csv(f'predictions/0_xgb_pred{i}.csv')
            print(f'Finish training and testing for TP{i}')
```

```
Start training and testing for TP1

2024-05-23 01:07:29,659 WARNING SparkXGBRegressor: _validate_gpu_params You have enabled GPU in spark local mode. Please make
sure your local node has at least 1 GPUs
24/05/23 01:07:30 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can
cause serious performance degradation.
24/05/23 01:07:30 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can
cause serious performance degradation.
24/05/23 01:07:30 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can
cause serious performance degradation.
24/05/23 01:07:30 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can
cause serious performance degradation.
24/05/23 01:07:30 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can
cause serious performance degradation.
24/05/23 01:07:30 WARN CSVHeaderChecker: CSV header does not conform to the schema.
 Header: , Date, time, TP, Load, Datetime
 Schema: _c0, Date, time, TP, Load, Datetime
Expected: _c0 but found:
CSV file: file:///home/ubuntu/electricity_demand_prediction/dataset/raw_load_df_2021.csv
24/05/23 01:07:30 WARN CSVHeaderChecker: CSV header does not conform to the schema
```

Push Updated Code to Git




Full-bilevel... (5) - JupyterLab
Launch AWS Academy Learner
Launch an instance | EC2 | us-e
Instance details | EC2 | us-east
EC2 Instance Connect | us-east
us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=us-east-1&connType=standard&instanceId=i-0d09303ce599c7a75&osUser=ubuntu&sshPort=22#/
Services
Search
[Alt+S]
N. Virginia
voclabs/user3251193=bpar433@aucklanduni.ac.nz @ 2976-9961-1588
(use "git restore <file>..." to discard changes in working directory)
modified: Assignment_4.ipynb
Untracked files:
(use "git add <file>..." to include in what will be committed)
.ipynb_checkpoints/
no changes added to commit (use "git add" and/or "git commit -a")
ubuntu@ip-172-31-54-212:~/electricity_demand_prediction$ git add .
ubuntu@ip-172-31-54-212:~/electricity_demand_prediction$ git commit -m
error: switch `m' requires a value
ubuntu@ip-172-31-54-212:~/electricity_demand_prediction$ git commit -m "finish coding"
[main 26f2546] finish coding
Committer: Ubuntu <ubuntu@ip-172-31-54-212.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:
git config --global --edit
After doing this, you may fix the identity used for this commit with:
git commit --amend --reset-author
2 files changed, 3790 insertions(+), 92 deletions(-)
create mode 100644 .ipynb_checkpoints/Assignment_4-checkpoint.ipynb
ubuntu@ip-172-31-54-212:~/electricity_demand_prediction$
i-0d09303ce599c7a75 (722-v3)
PublicIPs: 18.209.152.27 PrivateIPs: 172.31.54.212
CloudShell
Actions
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
11:22 AM
5/23/2024
Full-bilevel... (5) - JupyterLab
Launch AWS Academy Learner
Launch an instance | EC2 | us-e
Instance details | EC2 | us-east
EC2 Instance Connect | us-east
us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=us-east-1&connType=standard&instanceId=i-0d09303ce599c7a75&osUser=ubuntu&sshPort=22#/
Services
Search
[Alt+S]
N. Virginia
voclabs/user3251193=bpar433@aucklanduni.ac.nz @ 2976-9961-1588
ubuntu@ip-172-31-54-212:~/electricity_demand_prediction$ git commit -m "finish coding"
[main 26f2546] finish coding
Committer: Ubuntu <ubuntu@ip-172-31-54-212.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:
git config --global --edit
After doing this, you may fix the identity used for this commit with:
git commit --amend --reset-author
2 files changed, 3790 insertions(+), 92 deletions(-)
create mode 100644 .ipynb_checkpoints/Assignment_4-checkpoint.ipynb
ubuntu@ip-172-31-54-212:~/electricity_demand_prediction$ git push origin main
Username for 'https://github.com': bpardosi
Password for 'https://bpardosi@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.34 MiB | 5.96 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/bpardosi/electricity_demand_prediction
5b447df..26f2546 main -> main
ubuntu@ip-172-31-54-212:~/electricity_demand_prediction$
i-0d09303ce599c7a75 (722-v3)
PublicIPs: 18.209.152.27 PrivateIPs: 172.31.54.212
CloudShell
Actions
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
11:25 AM
5/23/2024


bpardosi / electricity_demand_prediction
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
electricity_demand_prediction
Public
Pin
Unwatch 1
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
Add file
Code
Ubuntu finish coding
26f2546 · 4 minutes ago
4 Commits
.ipynb_checkpoints
finish coding
4 minutes ago
dataset
added files
yesterday
predictions
added files
yesterday
Assignment_4.ipynb
finish coding
4 minutes ago
timeseries_exploration.R
Add files via upload
yesterday
README
Add a README
Help people interested in this repository understand your project by adding a README.
About
Forecast day-ahead half-hourly electricity demand with XGBoost
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Languages
Jupyter Notebook 99.9%
R 0.1%
Type here to search
11:26 AM
5/23/2024

References

Behmiri, N. B., Fezzi, C., & Ravazzolo, F. (2023). Incorporating air temperature into mid-term electricity load forecasting models using time-series regressions and neural networks. *Energy*.

Fayyad, U., Gregory, P.-S., & Padhraic, S. (1996, Fall). From Data Mining to Knowledge Discovery in Databases. *AI Magazine*.

Feng, Y., & Ryan, S. M. (2016). Day-ahead hourly electricity load modeling by functional regression. *Applied Energy, 170*, 455-465.

Gao, T., Niu, D., Ji, Z., & Sun, L. (2022). Mid-term electricity demand forecasting using improved variational mode decomposition and extreme learning machine optimized by sparrow search algorithm. *Energy, 261-Part B (125328)*.

Ghiassi, M., Zimbra, D. K., & Saidane, H. (2006). Medium term system load forecasting with a dynamic artificial neural network model. *Electric Power Systems Research, 76*(5), 302-316.

Grinsztajn.Léo, Oyallon, E., & Gaël, V. (2022). *Why Do Tree-based Models Still Outperform Deep Learning on Tabular Data*. arXiv:2207.08815v1.

Hastie, T., Tibshirani, R., & Friedman, J. (2013). *Elements of Statistical Learning.* New York: Springer.

Hyndman, R. J., & Athanasopoulos, G. (2021). *FORECASTING: PRINCIPLES AND PRACTICE.* OTexts.

Iftikhar et. al, H. (2023). Day-Ahead Electricity Demand Forecasting Using a Novel Decomposition Combination Method. *Energies*.

Li, L., Song, X., Li, J., Li, K., & Jiao, J. (2023). The impacts of temperature on residential electricity consumption in Anhui, China: does the electricity price matter? *Climatic Change, 176*(26).

Liu et. al., C. (2023). A day-ahead prediction method for high-resolution electricity consumption in residential units. *Energy*.

OpenAI. (2024, 03 22). ChatGPT (4) [Large language model].

Panapakidis, I. (2020). Short-Term, Medium-Term and Long-Term Load Forecasting: Methods and Applications (Special Issue). *Forecasting, 2*.

Pavićević, M., & Popović, T. (2022). Forecasting Day-Ahead Electricity Metrics with Artificial Neural Networks. *Sensors*.

Sak, H., Senior, A., & Beaufays, F. (2014). *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling*.

Semmelmann, L., Henni, S., & Weinhardt, C. (2022). Load forecasting for energy communities: a novel LSTM-XGBoost hybrid model based on smart meter data. *Energy Informatics*.

Shwartz-Ziv, R., & Armon, A. (2021). *Tabular Data: Deep Learning is Not All You Need*. arXiv:2106.03253v2.

Treasury, S. S. (2004). Crown Entities Act 2004. Wellington: State Services Commision and Treasury.

Wang, W., Shi, Y., Lyu, G., & Deng, W. (2017). Electricity Consumption Prediction Using XGBoost Based on Discrete Wavelet Transform. *DEStech Transactions on Computer Science and Engineering*.

Wu, X., Dou, C., & Yue, D. (2020). Electricity load forecast considering search engine indices. *Electric Power Systems Research, 199(107398)*.

Yao, J. (2021). *Electricity Consumption and Temperature: Evidence from Satellite Data*. Washington, DC: International Monetary Fund.

Zhe, W., Tianzhen, H., Han, L., & Piette, M. A. (2021). Predicting city-scale daily electricity consumption using data-driven models. *Advances in Applied Energy, 2(100025)*.