ADS1002 | Data Challenges 2

Energy Demand Modelling

How does temperature affect energy demand in Melbourne?

Image obtained from Griffith University

Report completed by:

Emma Kelsall [32592086]
Fiona George [33154430]
Pooja Kampli [33154759]
Sarayu Nousind [33154708]

TABLE OF **CONTENTS**

Introduction —----------03

Background

I. Supplied Data —----------03

II. Variables —----------04

Data Wrangling & Cleaning

I. Data Wrangling & Cleaning ----------05

Data Manipulation

I. Data Visualisation —----------06

II. Exploratory Data Analysis ----------07

Modelling

I. Linear Regression—---------- ----------08

II. Multivariate Polynomial Regression----------09

III. Random Forest regression —----------10

IIII. Neural Prophet Forecasting Model ----------11

Conclusion —----------12

INTRODUCTION

The objective of this assignment is to utilise the data provided to develop an energy demand forecasting model that can predict the energy demand of Victoria based on the average air temperature of the state.

This report investigates trends and patterns in the data, visualises correlations between temperature and energy demand, and measures the predicted values produced by the model against the actual recorded data to determine model accuracy.

Emma Kelsall performed data wrangling and cleaning, linear regression and the Neural Prophet Forecast, Fiona George carried out the data processing and random forest regression. Pooja Kampli prepared the background research and data analysis/visualisation . Sarayu Nousind completed the multivariate polynomial regression, data analysis and conclusion.

BACKGROUND

The National Energy Market states that there is a very clear relationship between the temperature and energy demand. Heaters are used more regularly on colder days whereas air conditioners are used on warmer days. There is a seasonal cycle and there must be enough energy supplied to meet that demand. This requires energy companies to predict what the demand will be like in conjunction with the predicted temperature for the next year.

Supplied Data

For this project, 3 different data sets were provided for a period of 20 years, starting in 2000 to 2019. These data sets were for demand data, weather data and a final data set for solar panel data. The demand data was separated by month and state with only the 5 east coast states (VIC, NSW, SA, QLD, TAS) included. WA was not included as data is obtained from a different energy grid. This data was from the Australian Energy market operator and came in half-hourly timestamps. The weather data was obtained from the Bureau of Meteorology's Automatic Weather Stations. The data was separated by weather station, with one data set for each capital of these eastern states with the exception of Melbourne, which had 2 different data sets, one data set (VIC-regional office) covering 2000-2015 and the other 2013-2020 (VIC-Olympic Park) which was activated to replace the regional office weather station. The weather data was also given in half-hourly timestamps.

We chose not to use the solar panel data (Rooftop PV data) as it was not relevant to our project goal of developing an energy demand forecasting model based upon Melbourne temperature. For the weather data we chose to only focus on the data from Olympic Park from 2014-2019. The Regional Office weather station was not at the same location as Olympic Park, with the Regional Office located at Carlton Gardens and Olympic Park located at Olympic Park Oval. There is 3 km between these 2 sites, with Olympic Park located on the Yarra river and Regional Office located in the inner city. Hence due to the different conditions that would be experienced at these locations we could not combine both data sets into one as they are simply disparate.

Variables

Weather Data Set

The data set HM01X_Data_086338_999999999743964.txt was used and represented data from the Olympic Park Weather Station in Melbourne. This data set contained 25 variables with date and time split into 5 different columns (YYYY, DD, MM, HH, MI) then weather variables based on air temperature, wet bulb temperature, dew point, humidity, precipitation, wind speed and sea level pressure..

The main variables chosen were time, date and temperature. We chose to focus on temperature over other variables as although it had a relatively low correlation with demand. It had the highest correlation out of all the variables. The Australian Energy Market Operator also described temperature to have an effect on demand, hence why we chose to solely focus on this variable.

Demand Data Set

In line with our choice to use temperature data from Melbourne, we chose to only use Victorian demand data. This data contained 5 variables. These included region, settlement date, total demand, rrp and period type. We chose to only use total demand and the settlement date as the other variables were not relevant to our final goal. The Settlement date would allow us to combine our two data sets during our data wrangling process, while the total demand is what we'd be modelling.

DATA WRANGLING & CLEANING

In order to get our data into a state ready for merging we first had to clean our data and rename our columns. The weather data was first converted to excel format to filter and extract the years we needed (2014-2019). There were 2 columns provided for time, 'local time' and 'standard time', 'standard time' was chosen as it accounted for daylight savings which would create duplicate values and missing values if the local time was used. From there, the date and time columns for the temperature dataset were combined into 1 column called 'dateTime'. This data set was then read into Python using Pandas. We converted the 'dateTime' column to the timestamp form and made it the index.

For the energy demand data, we combined the 72 separate files, with each file representing each month for the 6 year period into one big dataset. This was completed using the pandas concat method and joining on the row axis (axis 0). We then renamed the 'SETTLEMENTDATE' column to 'dateTime' to match that of the temperature dataset. This column was then converted to timestamp form to match that of the temperature data set and made the index column for ease in merging the 2 datasets.

After this initial processing unnecessary columns were dropped by subsetting only 'Air Temperature in degrees C' and 'dateTime' from the weather data set, creating the 'tempData' dataframe, then the 'TOTALDEMAND' and 'SETTLEMENTDATE' from the demand data, creating the 'demandData' data frame.

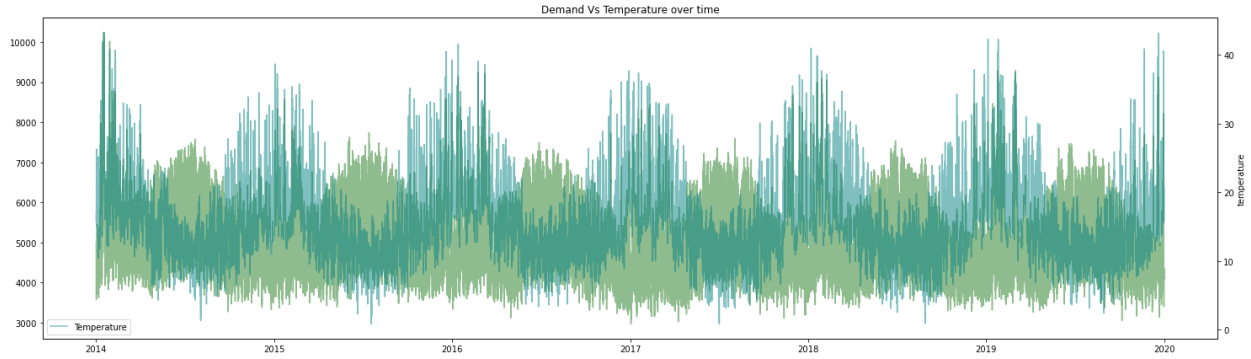
The last step that we undertook in the data cleaning process was sorting through the data and finding anything that would cause issues when merging. We found 2081 columns in the tempData that didn't fit the half-hourly format. For example the timestamp was for 12:34 or 15:56, with an entry still at 12:30 and 16:00. We chose to simply remove these as we did not have a demand value for these timestamps.

Lastly we merged the 2 datasets using pd.merge, merging on the index. This created our dataset 'tempDemand' which was used for all our future data analysis and modelling.

DATA MANIPULATION

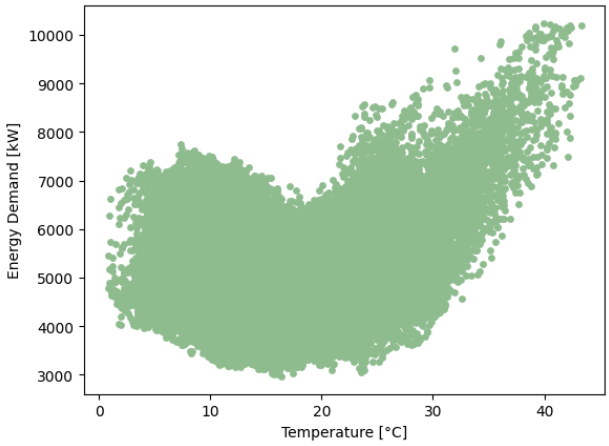
Data Visualisation

Demand vs temperature over time



This graph is a visualisation of the overall dataset we worked with. The dark green line represents the temperature and the light green line represents the energy demand, over the full 6 year period. Following the seasonality trend, we observed that energy demands increased during the winter seasons. The biggest source of this is the usage of heaters. Colder weather can drive up energy usage as heaters work overtime to keep indoor temperatures comfortable.

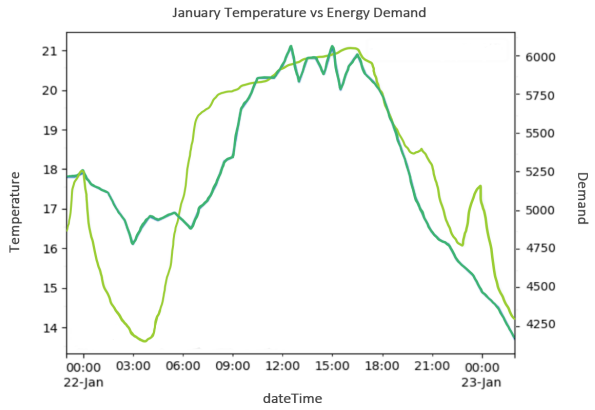
Temperature against energy demand (disregarding time)



This graph shows that when temperature is lower, below 15 degrees, energy demand is slightly higher as more people use heaters in this cold weather. Demand is also very high when the temperature is high, above 30 degrees as many people will use air conditioners for the heat. Whereas when the temperature is moderate, between 15-25 degrees the demand is slightly lower as neither heaters nor air conditions are used regularly.

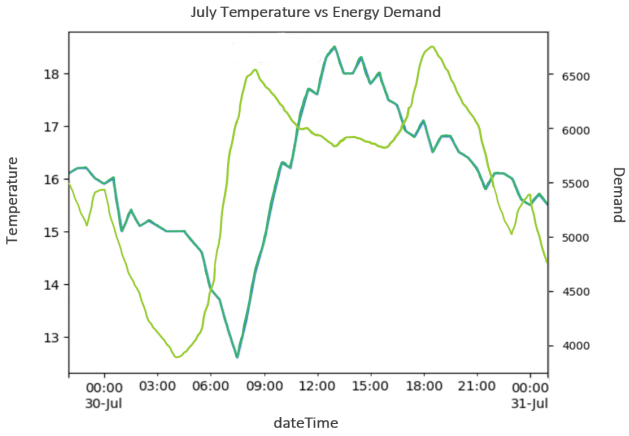
Exploratory Data Analysis

The temperature vs energy demand was closely investigated for one day chosen in summer and one chosen day in winter to compare.



This graph is an example of the energy demand and temperature on a summer day on the 22nd of January, 2014. From the graph, the energy demand mirrors the air temperature. As it is summer, energy is mainly just used for air conditioning when the temperature is higher. Whereas when the temperature drops, air conditioning will no longer be required and hence energy demand also drops. Investigating closely, the demand for energy was relatively low during 12am - 3 am as people are generally asleep at this time and would not need to use air conditioners. The temperature was also relatively low, being around 16-18 degrees, not requiring the use of the air conditioners. However, the energy started to rise quickly between 4am-9am as the temperature was rising quickly at this time as well from 17-20 degrees. People also wake up around this time to turn on their air conditioners. The demand for energy was relatively high and consistent at 9am and 6pm as the temperature was quite high and people would keep their air conditioners on during the day.

Whereas in this graph, we can see how it looked on a winter day, on the 30th of July, 2014. The sudden drop in temperature between 6am and 9am caused a sudden peak in the energy usage in the same time period, as people were likely to use their heaters at that time. There was also a drop in temperature after 3pm, causing an increase in energy demand around that time due to the same reason.



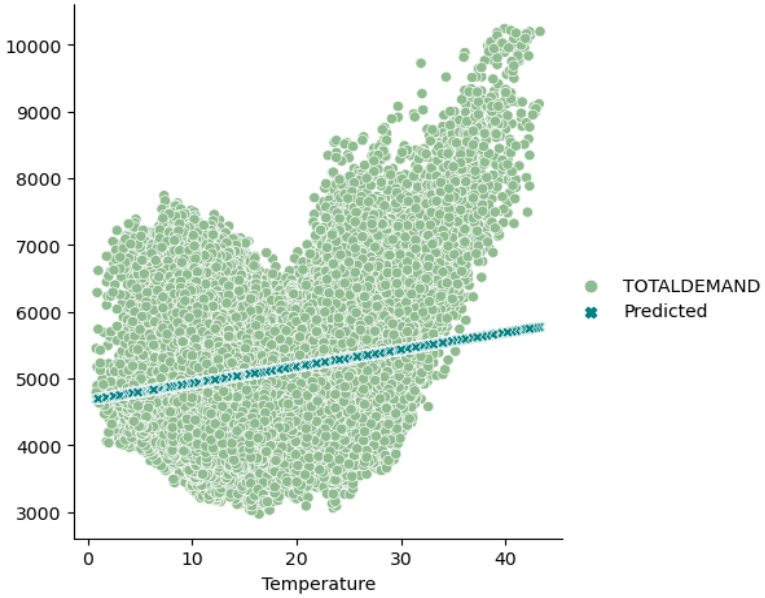
Hence, from the data analysis we can see that generally, the more volatile the weather is, the higher energy demand will be. This includes not just the cold days in winter but also the hot days in summer.

MODELLING

For the modelling aspect, we pursued four different types of analysis: **Linear Regression, Multivariate Polynomial regression**, **Random Forest regression**, and **Neural Prophet time-series forecasting**. From those four different methods, we chose to base our conclusion on the model that had the highest accuracy rate, and lowest error rate.

Linear Regression Model

The first model that we attempted was a linear regression model, for the estimation of energy demand based on air temperature.



Using this model we achieved an r-squared score of 0.0261699. The closer the r-squared value is to 1, the greater the correlation between the two variables. As the r-squared value is extremely low in this model, this indicates that there is very little linear relationship between temperature and demand. This can be seen in the poor fitting of our model, where there is large variation from the regression line. This also resulted in very low testing and training scores of 0.027 and 0.024 respectively. As a result, we decided that this model is not a viable option for us to predict energy demand based on temperature.

Polynomial regression model

The second modelling attempted was a polynomial regression model. It is similar to linear regression where it approximates the relationship as a nth degree polynomial. As shown previously, the relationship between the air temperature and energy demand is that of a non-linear nature.

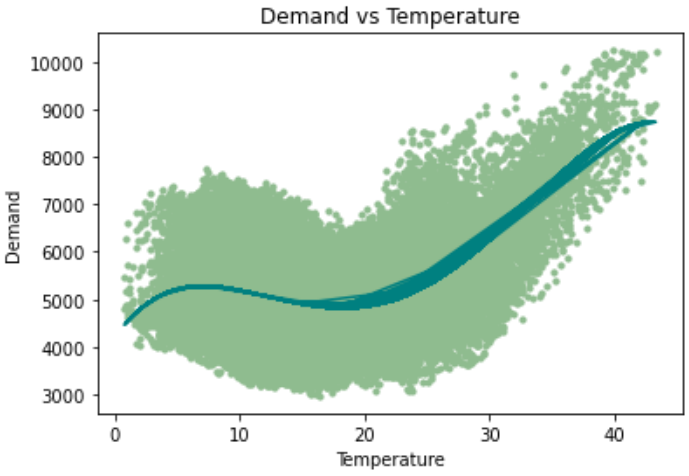


Figure 1: Polynomial Graph of degree 4

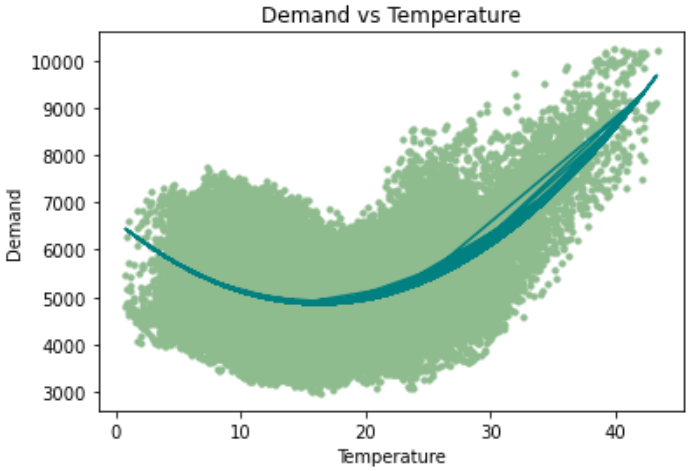


Figure 2: Polynomial Graph of degree 2

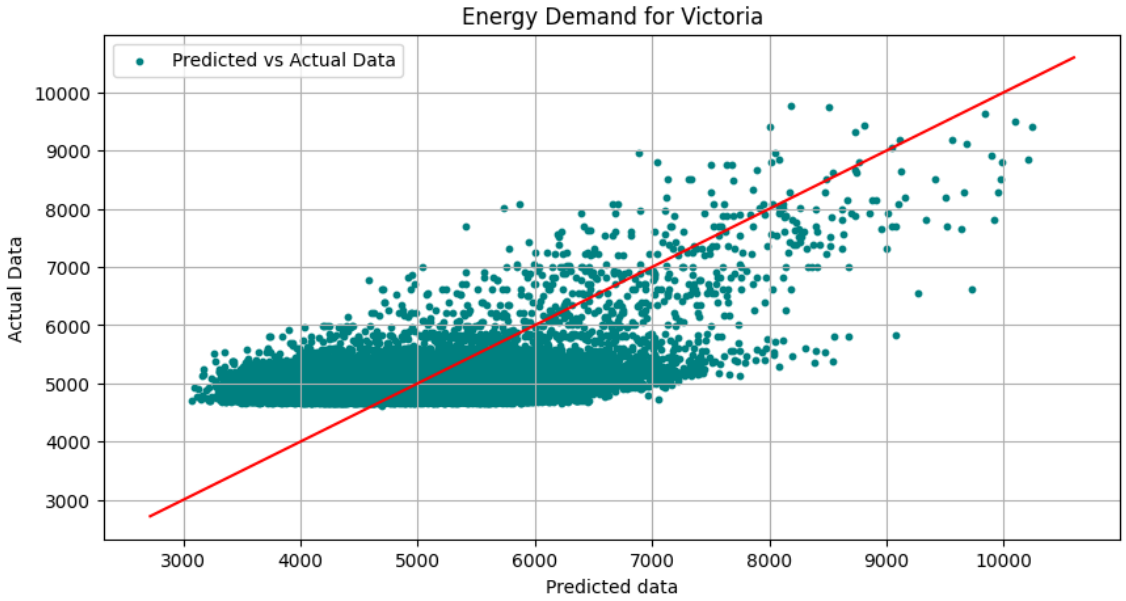
Bias and variance are related to the training and testing scores respectively in the training of the final model. Both need to be minimised to ensure the error of each set is approximately the same. The final model received an RMSE score of 804.86 and a R2 score of 0.183 using the 4th degree. This degree so as not to overfit that data and maintain low variance and bias on the final scores. We later found out that this was too high of a degree, as it should've been 2, meaning that the model was already overfit. The higher degree would've lowered the bias of the testing data, but caused more variance in the training data.

While there was significant improvement in the scores compared to the linear model, we cannot consider this model accurate, as it does not factor time into the temperature changes throughout the day, hence why there is high variation of data points. Despite this, the polynomial model clearly outlines that air temperature does have an impact on the energy demand in Victoria.

Random Forest regression model

The third model we attempted was Random Forest Regression. The biggest advantage of this model is that it relies on the power of "the majority", so the overall biasedness of the algorithm is greatly reduced. Even if a new data point is introduced in the dataset the algorithm is not affected much since it may impact only one decision tree.

To start off, the air temperature *['Temperature']* and energy demand *['TOTALDEMAND']* values were assigned as our x and y variables and the data was split into 'training' and 'testing' sets. N estimators is a parameter that defines the number of trees the model builds before utilising the predictions. The higher the number of estimators, the better the performance of the model, however, it compromises on the speed of the code so 1000 was the value we settled on to train our model.



Visualising the predicted and actual values for energy demand against each other the scatter plot above, we can see a somewhat strong correlation between the two. The RMSE is approximately 797.4 which seems like a large number, but is only around 0.75% of the total energy demand values in the dataset. The R2 value, standing at approximately 0.19, is better than the polynomial regression model but still lacks the accuracy that is needed for a successful model. This may be due to the absence of time as a main factor of the seasonal fluctuations of temperature within a day.

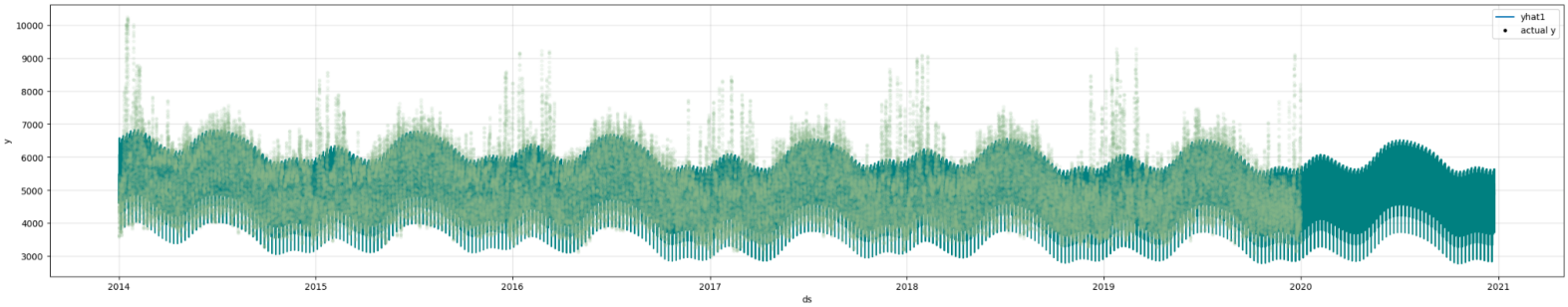
Although the random forest regression model has room for improvement, the scores are well beyond enough to prove the considerable impact that air temperature has on energy consumption in Victoria.

Neural Prophet time-series forecasting model

The final model that we chose to attempt was a Neural Prophet time-series forecasting model. Neural Prophet is an external Python library that was based upon Facebook's 'Prophet' forecast model. It is a neural network based model that predicts future values based upon past values.

We used this model to predict both temperature and energy demand for 2020 to see how the model performed. For the temperature model we achieved an RMSE score of 3.66 and MAE score of 2.76 which indicates that the model performs reasonably well.

For our demand model (below), which is the main variable that we have been attempting to model during the course of this project, we achieved an RMSE of 529 and 383. As we have attempted to model demand in our previous models, we are able to compare how this model performs in comparison to the others. Given the scores that we have achieved, while they are relatively large it is clear that this model performs better than both our random forest model and our multivariate polynomial regression. In the polynomial regression our RMSE was 804.86, while the random forest was 797.4. An RMSE of 529 is significantly better. An indicates that this way of predicting energy demand for a certain period is a more accurate method.



While our model performed better than our previous models, it would be easy to improve its accuracy so that it performs even more reliably. We received 20 years of data for demand spanning 2000-2019. We could simply train our model on this dataset and hence predict 2020 using this. We also used an epoch of 57. An epoch is the number of passes that the algorithm has to complete during training. Ideally we'd like to set this to a higher number however we were unable to run this as it would take too much time to run. An epoch of 1000 would be ideal to make our model more accurate.

CONCLUSION

In conclusion, the 4 models that we came up with all produced a variety of different results, but only 3 were considered viable options. Linear regression was discarded due to the large variation of data around the line, resulting in poor RMSE and R2 scores. The polynomial regression model was an improvement but not by much due to the overfitted data. The random forest model had performed the best of the models in terms of the RMSE and R2 scores. While not a huge improvement from the polynomial model, it helped consolidate the relationship between air temperature and energy demand.

The Neural Prophet model is the best in terms of accuracy. While the other two models were based on the air temperature, this one forecasts energy demand based on past year's trends and seasonality. This naturally includes not just the temperature but also other seasonal factors such as holidays and events which can also influence energy demand.

The limitations of the previous models was that air temperature was only the variable it based the energy demand on, meaning they cannot be considered accurate. By factoring in more variables such as humidity and wind speed, we would've had better RMSE and R2 scores and a model that takes into account more seasonality trends for better accuracy. Improvements we can implement into our models would be looking into the specific temperature changes during the time of days to see its effect on demand, and also energy demand on specific holidays to see if they are higher than expected.

APPENDIX

Data Sets

https://github.com/EmKelsall/tempEnergyDemandModelling

Data Processing and Wrangling

```
pip install neuralprophet
```

```
# load in basic libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

1. Read in CSV Files

```
# Read in weather data
# YYYY MM DD HH MI columns converted into 1 in excel

weather_data = pd.read_csv("tempDataVIC.csv")

weather_data
```

	dateTime object	Temperature float...	TOTALDEMAND f...
0	2014-01-01 00:00:00	16.7	4993.52
1	2014-01-01 00:30:00	16.6	4656.03
2	2014-01-01 01:00:00	16.4	4373.04
3	2014-01-01 01:30:00	16.7	4175.32
4	2014-01-01 02:00:00	16.5	4003.35
5	2014-01-01 02:30:00	16.2	3841.74
6	2014-01-01 03:00:00	15.8	3711.1
7	2014-01-01 03:30:00	15.7	3639.31

8	2014-01-01 04:00:00	15.7	3576.27	
9	2014-01-01 04:30:00	15.8	3591.05	

```
# Read in demand data and concatanate by month

# January
jan2013 = pd.read_csv('DEMAND DATA VIC 2013-2019/JAN/DATA201301_VIC1.csv')
jan2014 = pd.read_csv('DEMAND DATA VIC 2013-2019/JAN/DATA201401_VIC1.csv')
jan2015 = pd.read_csv('DEMAND DATA VIC 2013-2019/JAN/DATA201501_VIC1.csv')
jan2016 = pd.read_csv('DEMAND DATA VIC 2013-2019/JAN/DATA201601_VIC1.csv')
jan2017 = pd.read_csv('DEMAND DATA VIC 2013-2019/JAN/DATA201701_VIC1.csv')
jan2018 = pd.read_csv('DEMAND DATA VIC 2013-2019/JAN/DATA201801_VIC1.csv')
jan2019 = pd.read_csv('DEMAND DATA VIC 2013-2019/JAN/DATA201901_VIC1.csv')

january = pd.concat([jan2013, jan2014, jan2015, jan2016, jan2017, jan2018, jan2019], axis=0)
january.to_csv('all_demand_jan.csv')
```

```
# February
feb2013 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/FEB/DATA201302_VIC1.csv')
feb2014 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/FEB/DATA201402_VIC1.csv')
feb2015 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/FEB/DATA201502_VIC1.csv')
feb2016 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/FEB/DATA201602_VIC1.csv')
feb2017 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/FEB/DATA201702_VIC1.csv')
feb2018 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/FEB/DATA201802_VIC1.csv')
feb2019 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/FEB/DATA201902_VIC1.csv')


february = pd.concat([feb2013, feb2014, feb2015, feb2016, feb2017, feb2018, feb2019], axis=0)
february.to_csv('all_demand_feb.csv')
```

```
# March
mar2013 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAR/DATA201303_VIC1.csv')
mar2014 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAR/DATA201403_VIC1.csv')
mar2015 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAR/DATA201503_VIC1.csv')
mar2016 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAR/DATA201603_VIC1.csv')
mar2017 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAR/DATA201703_VIC1.csv')
mar2018 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAR/DATA201803_VIC1.csv')
mar2019 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAR/DATA201903_VIC1.csv')


march = pd.concat([mar2013, mar2014, mar2015, mar2016, mar2017, mar2018, mar2019], axis=0)
march.to_csv('all_demand_mar.csv')
```

```
# April
apr2013 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/APR/DATA201304_VIC1.csv')
apr2014 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/APR/DATA201404_VIC1.csv')
apr2015 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/APR/DATA201504_VIC1.csv')
apr2016 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/APR/DATA201604_VIC1.csv')
apr2017 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/APR/DATA201704_VIC1.csv')
apr2018 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/APR/DATA201804_VIC1.csv')
apr2019 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/APR/DATA201904_VIC1.csv')


april = pd.concat([apr2013, apr2014, apr2015, apr2016, apr2017, apr2018, apr2019], axis=0)
april.to_csv('all_demand_apr.csv')
```

```
# May
may2013 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAY/DATA201305_VIC1.csv')
may2014 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAY/DATA201405_VIC1.csv')
may2015 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAY/DATA201505_VIC1.csv')
may2016 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAY/DATA201605_VIC1.csv')
may2017 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAY/DATA201705_VIC1.csv')
may2018 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAY/DATA201805_VIC1.csv')
may2019 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/MAY/DATA201905_VIC1.csv')


may = pd.concat([may2013, may2014, may2015, may2016, may2017, may2018, may2019], axis=0)
may.to_csv('all_demand_may.csv')
```

```
# June
jun2013 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUN/DATA201306_VIC1.csv')
jun2014 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUN/DATA201406_VIC1.csv')
jun2015 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUN/DATA201506_VIC1.csv')
jun2016 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUN/DATA201606_VIC1.csv')
jun2017 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUN/DATA201706_VIC1.csv')
jun2018 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUN/DATA201806_VIC1.csv')
jun2019 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUN/DATA201906_VIC1.csv')

june = pd.concat([jun2013, jun2014, jun2015, jun2016, jun2017, jun2018, jun2019], axis=0)
june.to_csv('all_demand_jun.csv')
```

```
# July
jul2013 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUL/DATA201307_VIC1.csv')
jul2014 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUL/DATA201407_VIC1.csv')
jul2015 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUL/DATA201507_VIC1.csv')
jul2016 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUL/DATA201607_VIC1.csv')
jul2017 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUL/DATA201707_VIC1.csv')
jul2018 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUL/DATA201807_VIC1.csv')
jul2019 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/JUL/DATA201907_VIC1.csv')

july = pd.concat([jul2013, jul2014, jul2015, jul2016, jul2017, jul2018, jul2019], axis=0)
july.to_csv('all_demand_jul.csv')
```

```
# August
aug2013 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/AUG/DATA201308_VIC1.csv')
aug2014 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/AUG/DATA201408_VIC1.csv')
aug2015 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/AUG/DATA201508_VIC1.csv')
aug2016 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/AUG/DATA201608_VIC1.csv')
aug2017 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/AUG/DATA201708_VIC1.csv')
aug2018 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/AUG/DATA201808_VIC1.csv')
aug2019 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/AUG/DATA201908_VIC1.csv')

august = pd.concat([aug2013, aug2014, aug2015, aug2016, aug2017, aug2018, aug2019], axis=0)
august.to_csv('all_demand_aug.csv')
```

```
# September
sep2013 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/SEP/DATA201309_VIC1.csv')
sep2014 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/SEP/DATA201409_VIC1.csv')
sep2015 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/SEP/DATA201509_VIC1.csv')
sep2016 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/SEP/DATA201609_VIC1.csv')
sep2017 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/SEP/DATA201709_VIC1.csv')
sep2018 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/SEP/DATA201809_VIC1.csv')
sep2019 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/SEP/DATA201909_VIC1.csv')
```

```
september = pd.concat([sep2013, sep2014, sep2015, sep2016, sep2017, sep2018, sep2019], axis=0)
september.to_csv('all_demand_sep.csv')
```

```
# October
oct2013 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/OCT/DATA201310_VIC1.csv')
oct2014 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/OCT/DATA201410_VIC1.csv')
oct2015 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/OCT/DATA201510_VIC1.csv')
oct2016 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/OCT/DATA201610_VIC1.csv')
oct2017 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/OCT/DATA201710_VIC1.csv')
oct2018 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/OCT/DATA201810_VIC1.csv')
oct2019 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/OCT/DATA201910_VIC1.csv')


october = pd.concat([oct2013, oct2014, oct2015, oct2016, oct2017, oct2018, oct2019], axis=0)
october.to_csv('all_demand_oct.csv')
```

```
# November
nov2013 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/NOV/DATA201311_VIC1.csv')
nov2014 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/NOV/DATA201411_VIC1.csv')
nov2015 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/NOV/DATA201511_VIC1.csv')
nov2016 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/NOV/DATA201611_VIC1.csv')
nov2017 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/NOV/DATA201711_VIC1.csv')
nov2018 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/NOV/DATA201811_VIC1.csv')
nov2019 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/NOV/DATA201911_VIC1.csv')

november = pd.concat([nov2013, nov2014, nov2015, nov2016, nov2017, nov2018, nov2019], axis=0)
november.to_csv('all_demand_nov.csv')
```

```
# December
dec2013 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/DEC/DATA201312_VIC1.csv')
dec2014 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/DEC/DATA201412_VIC1.csv')
dec2015 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/DEC/DATA201512_VIC1.csv')
dec2016 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/DEC/DATA201612_VIC1.csv')
dec2017 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/DEC/DATA201712_VIC1.csv')
dec2018 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/DEC/DATA201812_VIC1.csv')
dec2019 = pd.read_csv('/work/DEMAND DATA VIC 2013-2019/DEC/DATA201912_VIC1.csv')


december = pd.concat([dec2013, dec2014, dec2015, dec2016, dec2017, dec2018, dec2019], axis=0)
december.to_csv('all_demand_dec.csv')
```

```
# Concatanate all demand data together and convert to a csv file for
# for greater usability

all_demand_data = pd.concat(
    [january, february, march, april, may, june, july, august, september, october, november, december],

# Remove all unnecessary columns
all_demand_data = all_demand_data[['SETTLEMENTDATE', 'TOTALDEMAND']]
all_demand_data.to_csv('all_demand_data.csv')
```

Clean Data

```
# Remove all columns except dateTime and Temperature

onlyTemp = weather_data[['dateTime', 'Air Temperature in degrees C']]
```

```
tempData = onlyTemp.rename(columns={'Air Temperature in degrees C': 'Temperature'})
```

```
# Renaming
demandData = all_demand_data
demandData = demandData.rename(columns={'SETTLEMENTDATE': 'dateTime'})


# Clean Data -> remove any NaN
demandData = demandData.dropna()
tempData = tempData.dropna()

demandData['dateTime'] = pd.to_datetime(demandData['dateTime'])
tempData['dateTime'] = pd.to_datetime(tempData['dateTime'])
```

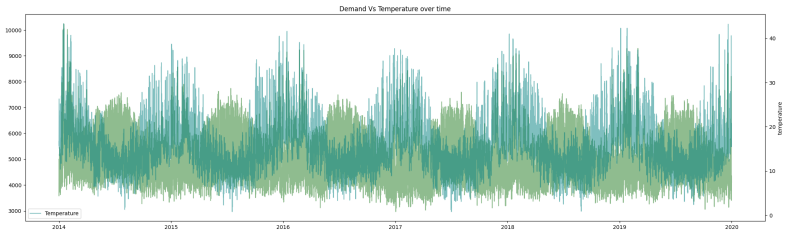
```
# Subset the data so it is only for the timeframe that we want

start = "2014-01-01 00:00:00"
end = "2019-12-31 23:30:00"

demandData = demandData[(demandData['dateTime'] >= start) & (demandData['dateTime'] <= end)]

tempData = tempData[(tempData['dateTime'] >= start) & (tempData['dateTime'] <= end)]
```

```
# Merge temperature and demand data using dateTime column as the join column
tempDemand = pd.merge(demandData, tempData, on='dateTime')

# convert to a csv for usability later
tempDemand.to_csv('tempDemand.csv')
```

Exploratory Data Analysis/ Visualisation

```
data = pd.read_csv('tempDemand.csv')
data["dateTime"] = pd.to_datetime(data["dateTime"])
data = data.dropna()
# Plot a line graph with temperature vs demand data over time


#combine total demand and temperature over time on one graph

fig, ax = plt.subplots(figsize=(25, 7))
ax2 = ax.twinx()

ax.plot(data['dateTime'], data["TOTALDEMAND"], color='darkseagreen', label="Demand")
ax2.plot(data['dateTime'], data["Temperature"], color='Teal', label="Temperature")
plt.gca().get_lines()[0].set_alpha(0.5)
plt.legend()
plt.title("Demand Vs Temperature over time")
plt.xlabel('date')
plt.ylabel('temperature')
```

```
Text(0, 0.5, 'temperature')
```

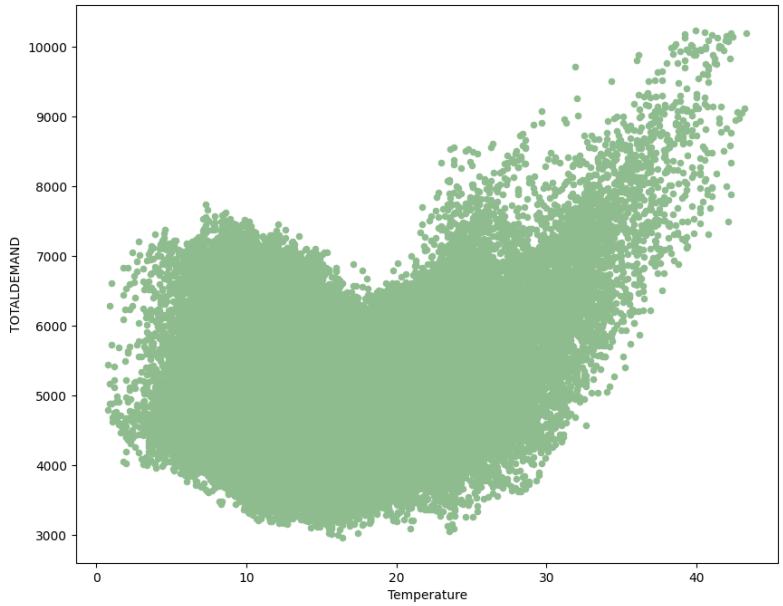



```
from sklearn import preprocessing
import pandas as pd

# Normalise data then create a correlation matrix
scaler = preprocessing.MinMaxScaler()
names = data[['TOTALDEMAND','Temperature']].columns
d = scaler.fit_transform(data[['TOTALDEMAND','Temperature']])
scaled_df = pd.DataFrame(d, columns=names)
scaled_df.head()

scaled_df.corr()
```

	TOTALDEMAND f...	Temperature float...	
TOTALDEMAND	1.0	0.16186148918578067	
Temperature	0.16186148918578067	1.0	

```
#plot temperature over total demand
data.head(50)

data.plot.scatter('Temperature', 'TOTALDEMAND', color='darkseagreen', figsize=(10,8))
```

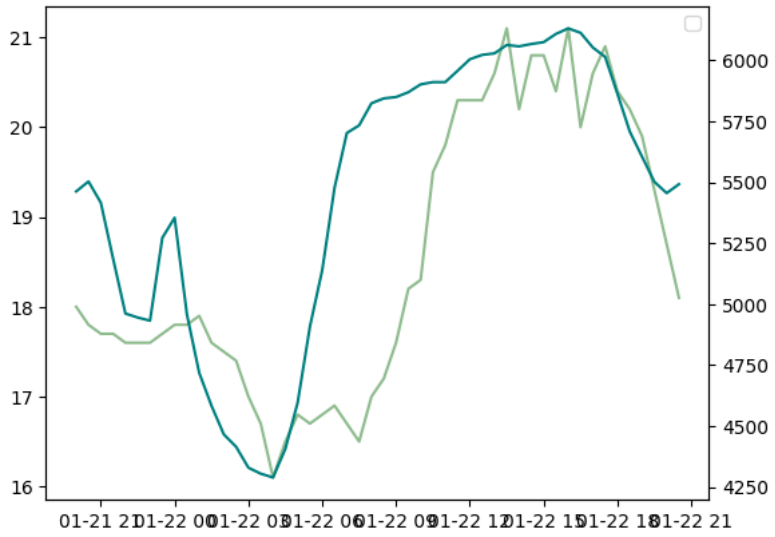
```
<AxesSubplot:xlabel='Temperature', ylabel='TOTALDEMAND'>
```




```
fig, ax = plt.subplots()
ax2 = ax.twinx()

#plot temperature and energy demand on one day
ax.plot(data.iloc[1000:1050]['dateTime'], data.iloc[1000:1050]['Temperature'], color='darkseagreen')

ax2.plot(data.iloc[1000:1050]['dateTime'], data.iloc[1000:1050]['TOTALDEMAND'], color='Teal')
plt.legend()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when
```

```
<matplotlib.legend.Legend at 0x7f5212c493a0>
```



Modelling

```
import pandas as pd
from neuralprophet import NeuralProphet
import matplotlib.pyplot as plt
```

Random Forest Regression Model

```
# importations
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

```
# load dataset
data = pd.read_csv('tempDemand.csv')
data['dateTime'] = pd.to_datetime(data['dateTime'])
# data = data.drop(columns=['dateTime'])
data
```

	dateTime datetim...	Temperature float...	TOTALDEMAND f...

0	2014-01-01 00:00:00	16.7	4993.52	
1	2014-01-01 00:30:00	16.6	4656.03	
2	2014-01-01 01:00:00	16.4	4373.04	
3	2014-01-01 01:30:00	16.7	4175.32	
4	2014-01-01 02:00:00	16.5	4003.35	
5	2014-01-01 02:30:00	16.2	3841.74	
6	2014-01-01 03:00:00	15.8	3711.1	
7	2014-01-01 03:30:00	15.7	3639.31	
8	2014-01-01 04:00:00	15.7	3576.27	
9	2014-01-01 04:30:00	15.8	3591.05	

```
# view missing values
data.isnull().sum()
```

```
dateTime       1
Temperature    2
TOTALDEMAND    1
dtype: int64
```

```
# remove missing values
data.dropna(inplace = True)
data.tail()
```

	dateTime datetim...	Temperature float...	TOTALDEMAND f...	
105163	2019-12-31 21:30:00	17.6	4172.46	
105164	2019-12-31 22:00:00	17.1	4129.96	
105165	2019-12-31 22:30:00	16.6	4083.66	
105166	2019-12-31 23:00:00	16.3	4104.95	
105167	2019-12-31 23:30:00	16.0	4325.88	

```
# assigning our x and y variables
xVars = data[['Temperature']]
yVars = data[['TOTALDEMAND']]

# splitting training and testing sets
xTrain, xValid, yTrain, yValid = train_test_split(xVars, yVars, test_size = 0.2, random_state = 42)

regressor = RandomForestRegressor(n_estimators = 1000, random_state = 42)
```

```
# fitting the model
regressor.fit(xTrain, yTrain)
```

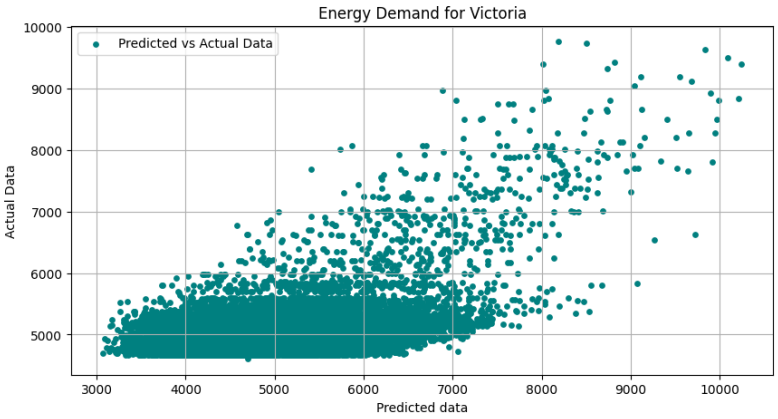
```
WARNING - (py.warnings._showwarnmsg) - /tmp/ipykernel_809/451802834.py:11: DataConversionWarning: A column-vector y was
  regressor.fit(xTrain, yTrain)
```

```
RandomForestRegressor
RandomForestRegressor(n_estimators=1000, random_state=42)
```

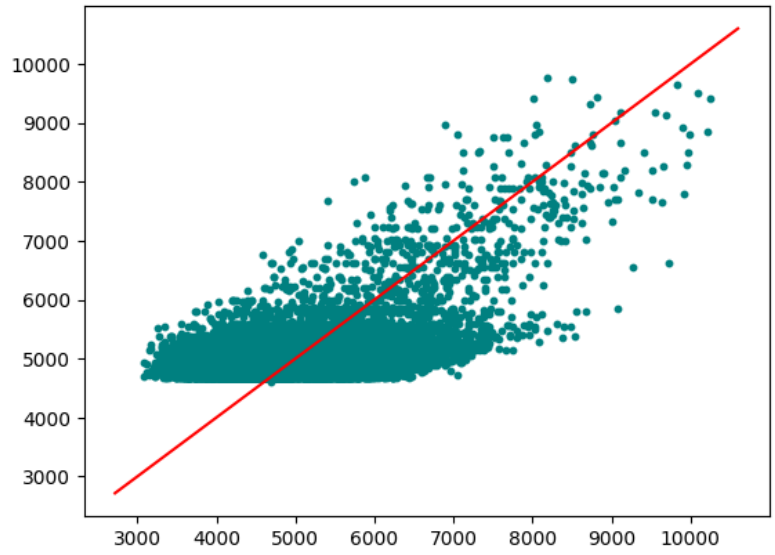
```
# predicted values
y_pred = regressor.predict(xValid)
y_pred = pd.DataFrame(y_pred, columns = ['yPredict'])
y_pred
```

	yPredict float64 4605.942170398...
0	4704.872879868348
1	4663.917299706529
2	5179.134563030922
3	4853.563066599323
4	4700.819439617434
5	4754.66094577971
6	5193.170712810532
7	5070.857562709582
8	4741.240755401196
9	5145.346653897517

```
# visualising the predicted data against the actual values
plt.figure(figsize = (10,5))
plt.scatter(yValid, y_pred, label = 'Predicted vs Actual Data', s = 15, color = 'teal')
plt.legend()
plt.grid()
plt.title('Energy Demand for Victoria')
plt.xlabel('Predicted data')
plt.ylabel('Actual Data')
plt.show()
```




```
# visualising the predicted data against the actual values
fig, ax = plt.subplots()
plt.scatter(yValid, y_pred, s = 10, color = 'teal')
x = np.linspace(*ax.get_xlim())
ax.plot(x, x, color = 'red')
plt.show()
```




```
# view model scores
mse = mean_squared_error(yValid, y_pred)
rmse = np.sqrt(mean_squared_error(yValid, y_pred))
r2 = r2_score(yValid, y_pred)

print("MSE =",mse)
```

```
print("RMSE =",rmse)
print("R2 =",r2)
```

```
MSE = 635873.1820699641
RMSE = 797.4165674664429
R2 = 0.19091718720056794
```

Neural Prophet Forecasting Model

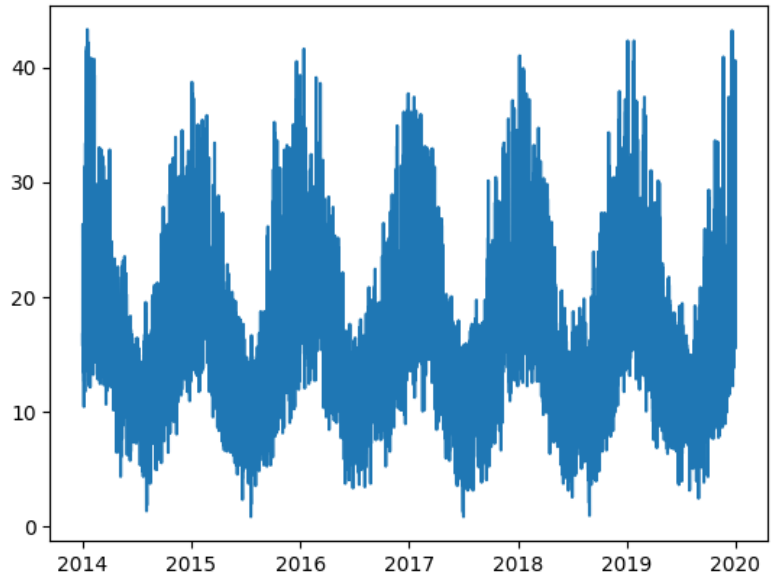
Forecasting Temperature for 2020

```
import pandas as pd
from neuralprophet import NeuralProphet
import matplotlib.pyplot as plt

data = pd.read_csv("tempDemand.csv")
data = data.dropna()
data.tail()

data["dateTime"] = pd.to_datetime(data["dateTime"])
data
```

	dateTime datetim...	Temperature float...	TOTALDEMAND f...
0	2014-01-01 00:00:00	16.7	4993.52
1	2014-01-01 00:30:00	16.6	4656.03
2	2014-01-01 01:00:00	16.4	4373.04
3	2014-01-01 01:30:00	16.7	4175.32
4	2014-01-01 02:00:00	16.5	4003.35
5	2014-01-01 02:30:00	16.2	3841.74
6	2014-01-01 03:00:00	15.8	3711.1
7	2014-01-01 03:30:00	15.7	3639.31
8	2014-01-01 04:00:00	15.7	3576.27
9	2014-01-01 04:30:00	15.8	3591.05

```
plt.plot(data['dateTime'], data["Temperature"])
```

```
[<matplotlib.lines.Line2D at 0x7f9445152ee0>]
```




```
toModel = data[['dateTime', 'Temperature']]
toModel.columns = ['ds', 'y']
toModel.tail()
```

	ds datetime64[ns]	y float64
105163	2019-12-31 21:30:00	17.6
105164	2019-12-31 22:00:00	17.1
105165	2019-12-31 22:30:00	16.6
105166	2019-12-31 23:00:00	16.3
105167	2019-12-31 23:30:00	16.0

```
# Fit model to forecast temperature

n = NeuralProphet()

metrics = n.fit(toModel)
```

```
INFO - (NP.df_utils._infer_frequency) - Major frequency 30T corresponds to 99.998% of the data.
INFO - (NP.df_utils._infer_frequency) - Dataframe freq automatically defined as 30T
INFO - (NP.config.init_data_params) - Setting normalization to global as only one dataframe provided for training.
INFO - (NP.config.set_auto_batch_epoch) - Auto-set batch_size to 128
INFO - (NP.config.set_auto_batch_epoch) - Auto-set epochs to 57
```

```
IPyWidgets are not supported when hardware is not ready.
```

```
INFO - (NP.utils_torch.lr_range_test) - lr-range-test results: steep: 7.88E-02, min: 1.74E+00
```

```
IPyWidgets are not supported when hardware is not ready.
```

```
INFO - (NP.utils_torch.lr_range_test) - lr-range-test results: steep: 7.00E-02, min: 1.37E+00
INFO - (NP.forecaster._init_train_loader) - lr-range-test selected learning rate: 8.20E-02
Epoch[57/57]: 100%|██████████| 57/57 [04:12<00:00,  4.43s/it, SmoothL1Loss=0.0102, MAE=2.77, RMSE=3.65, Loss=0.00773, Re
```

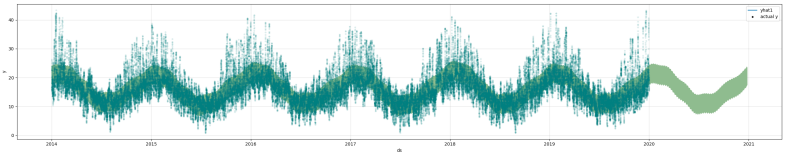
```
future = n.make_future_dataframe(toModel, periods=17260, n_historic_predictions=True)
forecast = n.predict(future)
forecast.tail()
```

```
INFO - (NP.df_utils._infer_frequency) - Major frequency 30T corresponds to 99.998% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - 30T
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
INFO - (NP.df_utils._infer_frequency) - Major frequency 30T corresponds to 99.998% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - 30T
INFO - (NP.df_utils._infer_frequency) - Major frequency 30T corresponds to 99.998% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - 30T
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
```

	ds datetime64[ns]	y float64	residual1 float64	yhat1 float64	trend float64	season_yearly flo...
122422	2020-12-25 11:30:00	nan	nan	22.253786087036133	15.646880149841309	4.541290760040283
122423	2020-12-25 12:00:00	nan	nan	22.610521316528320	15.646877288818360	4.543212890625
122424	2020-12-25 12:30:00	nan	nan	22.917373657226562	15.646873474121094	4.545134067535
122425	2020-12-25 13:00:00	nan	nan	23.183132171630860	15.646872520446777	4.547054767608643
122426	2020-12-25 13:30:00	nan	nan	23.403600692749023	15.646869659423828	4.548973083496094

```
plt.figure(figsize=(5,25))

plot = n.plot(forecast, figsize=(25,5))

plt.gca().get_lines()[0].set_color("darkseagreen")
plt.gca().get_lines()[1].set_color("Teal")
plt.gca().get_lines()[1].set_alpha(0.1)
```

```
<Figure size 500x2500 with 0 Axes>
```



Forecasting Demand for 2020

```
toModel = data[['dateTime', 'TOTALDEMAND']]
toModel.columns = ['ds', 'y']
toModel.tail()
```

	ds datetime64[ns]	y float64
105163	2019-12-31 21:30:00	4172.46
105164	2019-12-31 22:00:00	4129.96
105165	2019-12-31 22:30:00	4083.66
105166	2019-12-31 23:00:00	4104.95
105167	2019-12-31 23:30:00	4325.88

```
n = NeuralProphet()
model = n.fit(toModel)
```

```
INFO - (NP.df_utils._infer_frequency) - Major frequency 30T corresponds to 99.998% of the data.
INFO - (NP.df_utils._infer_frequency) - Dataframe freq automatically defined as 30T
INFO - (NP.config.init_data_params) - Setting normalization to global as only one dataframe provided for training.
INFO - (NP.config.set_auto_batch_epoch) - Auto-set batch_size to 128
INFO - (NP.config.set_auto_batch_epoch) - Auto-set epochs to 57
```

```
IPyWidgets are not supported when hardware is not ready.
```

```
INFO - (NP.utils_torch.lr_range_test) - lr-range-test results: steep: 4.70E-01, min: 1.96E+00
```

```
IPyWidgets are not supported when hardware is not ready.
```

```
INFO - (NP.utils_torch.lr_range_test) - lr-range-test results: steep: 4.70E-01, min: 1.96E+00
INFO - (NP.forecaster._init_train_loader) - lr-range-test selected learning rate: 2.81E-01
Epoch[57/57]: 100%|██████████| 57/57 [04:34<00:00,  4.82s/it, SmoothL1Loss=0.0108, MAE=383, RMSE=528, Loss=0.00811, RegL
```

```
future = n.make_future_dataframe(toModel, periods=17260, n_historic_predictions=True)
forecast = n.predict(future)
forecast.tail()
```

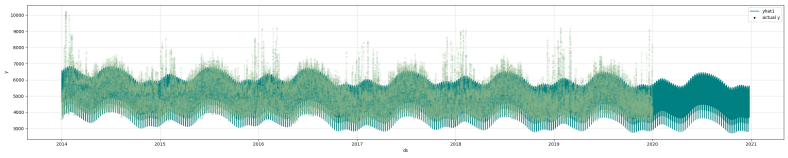
```
INFO - (NP.df_utils._infer_frequency) - Major frequency 30T corresponds to 99.998% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - 30T
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
INFO - (NP.df_utils._infer_frequency) - Major frequency 30T corresponds to 99.998% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - 30T
INFO - (NP.df_utils._infer_frequency) - Major frequency 30T corresponds to 99.998% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - 30T
INFO - (NP.df_utils.return_df_in_original_format) - Returning df with no ID column
```

	ds datetime64[ns]	y float64	residual1 float64	yhat1 float64	trend float64	season_yearly flo...
122422	2020-12-25 11:30:00	nan	nan	4911.341796875	4842.017578125	-282.0095825195 3125
122423	2020-12-25 12:00:00	nan	nan	4896.077636718 75	4842.015625	-281.8897705078 125

122424	2020-12-25 12:30:00	nan	nan	4882.048339843 75	4842.013183593 75	-281.769714355 6875
122425	2020-12-25 13:00:00	nan	nan	4868.531738281 25	4842.010742187 5	-281.649200439 531
122426	2020-12-25 13:30:00	nan	nan	4861.131835937 5	4842.008789062 5	-281.528320312

```
plt.figure(figsize=(5,25))

plot = n.plot(forecast, figsize=(25,5))

plt.gca().get_lines()[1].set_color("darkseagreen")
plt.gca().get_lines()[0].set_color("Teal")
plt.gca().get_lines()[1].set_alpha(0.1)
```

```
<Figure size 500x2500 with 0 Axes>
```



```
# Neural prophet: https://neuralprophet.com/html/index.html

@misc{triebe2021neuralprophet,
      title={NeuralProphet: Explainable Forecasting at Scale},
      author={Oskar Triebe and Hansika Hewamalage and Polina Pilyugina and Nikolay Laptev and Christoph
      year={2021},
      eprint={2111.15397},
      archivePrefix={arXiv},
      primaryClass={cs.LG}
}
```

Linear Regression

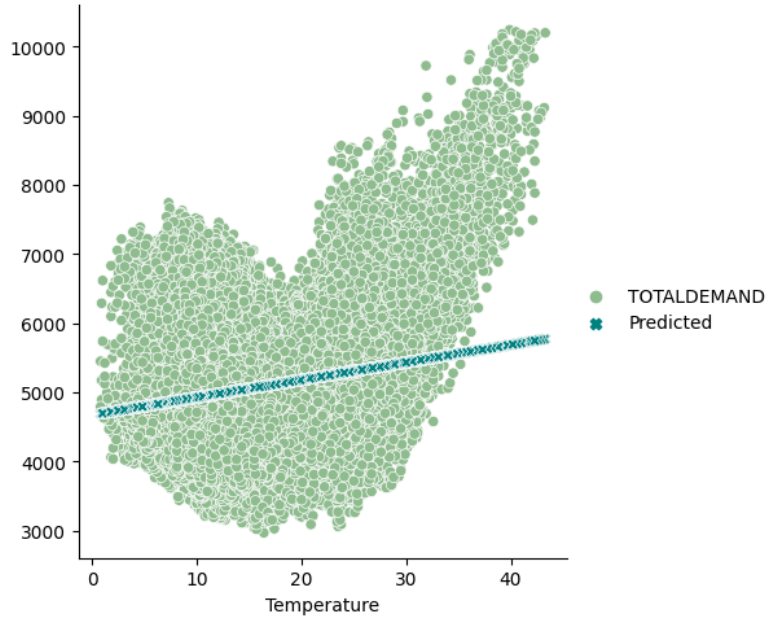
```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


from sklearn.linear_model import LinearRegression # linear regression model
from sklearn.model_selection import train_test_split # for splitting the data into training and testing
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
# Linear Regression based on Ordinary Least Squares method:

# Set up the model type. This indicates we are fitting a linear regression model
# and we want to include an intercept parameter.
model = LinearRegression(fit_intercept=True)
```

```
# drop na's
data = pd.read_csv('tempDemand.csv')
data = data.dropna()


df = data[["Temperature", "TOTALDEMAND"]]


# Fit the model.
model.fit(X=df[["Temperature"]], y=df["TOTALDEMAND"])

# Set up the data matrices X and y (note we add a column of ones to X).
X = np.matrix(data[['Temperature']].assign(constant=1).values)
y = data["TOTALDEMAND"].values

# Compute the model parameters through matrix multiplication.
beta = np.linalg.inv(X.transpose() @ X) @ X.transpose() @ y
optimal_gradient, optimal_intercept = beta.tolist()[0]

# Plot the linear Regression
plot_predictions(optimal_gradient, optimal_intercept)

# Also display the prediction error.
print("Model is y = {:.2f}x + {:.2f}".format(optimal_gradient, optimal_intercept))
print("RMSE = {:.2f}".format(prediction_root_mean_squared_error(optimal_gradient, optimal_intercept)))
print("MAE = {:.2f}".format(prediction_mean_absolute_error(optimal_gradient, optimal_intercept)))
print("R2 = ", r_squared(optimal_gradient, optimal_intercept))
```

```
Model is y = 25.38x + 4669.37
RMSE = 878.82
MAE = 714.97
R2 =  0.026199141681438776
```

```
<Figure size 2000x500 with 0 Axes>
```



Polynomial Regression

```
# import libraries and load in data
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

DF = pd.read_csv('tempDemand.csv')
```

```
# drop any NA values and check the types of each column
DF = DF.dropna()
DF.dtypes
```

```
dateTime        object
Temperature    float64
TOTALDEMAND    float64
dtype: object
```

```
# select the columns need for variables
X = DF.iloc[:, 1:2].values
Y = DF.iloc[:, 2].values
```

```
# load linear regression
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression() # create model object
lin_reg.fit(X, Y)
```

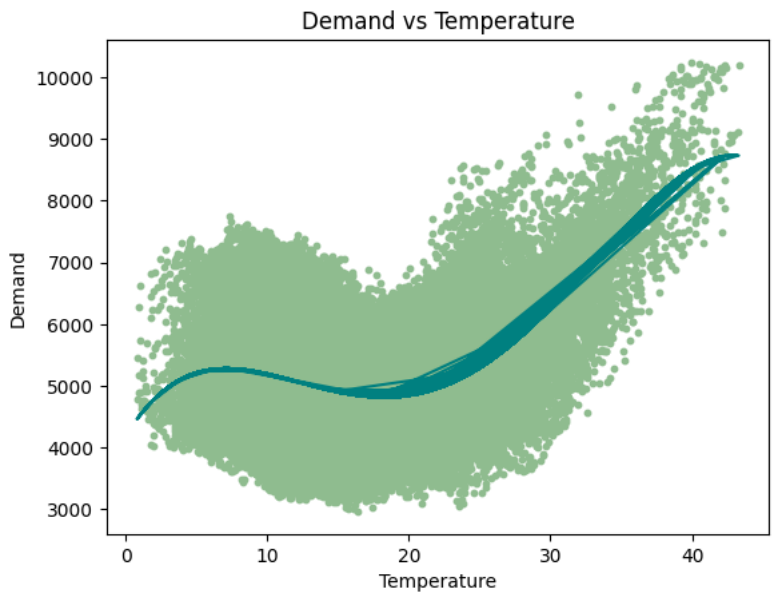
```
▾ LinearRegression
LinearRegression()
```

```
# load polynomial regression
from sklearn.preprocessing import PolynomialFeatures
poly_regr = PolynomialFeatures(degree = 4) # switched this for degree 2 graph in report
X_poly = poly_regr.fit_transform(X) # transforms the features to the polynomial form
lin_reg_2 = LinearRegression() # creates a linear regression object
lin_reg_2.fit(X_poly, Y)
```

```
▾ LinearRegression
LinearRegression()
```

```
# graph of the polynomial regression
plt.scatter(X, Y, color = 'darkseagreen', s=10)
plt.plot(X, lin_reg_2.predict(poly_regr.fit_transform(X)), color = 'teal')
plt.title('Demand vs Temperature')
plt.xlabel('Temperature')
plt.ylabel('Demand')
```

```
Text(0, 0.5, 'Demand')
```




```
# reshape columns for scores calculation
X = DF['Temperature'].values.reshape(-1,1)
y = DF['TOTALDEMAND'].values.reshape(-1,1)
```

```
#calculate the rmse and r2 scores
```

```
poly = PolynomialFeatures(degree = 4)

X_poly = poly.fit_transform(X)
regressor = LinearRegression()
regressor.fit(X_poly,y)
y_poly_pred=regressor.predict(X_poly)

rmse = np.sqrt(mean_squared_error(y,y_poly_pred))
r2 = r2_score(y,y_poly_pred)
print('RMSE score is '+ str(rmse))
print('R Squared is '+str(r2))
```

```
RMSE score is 804.8587071462243
R Squared is 0.1832161614616783
```