Temperature and Energy Demand

By: Pranav Kutty, Wen Wen, Nishandeep Brar, Yunru Zhu

Background

Intentions

For this project we intended to find the correlation between energy consumption and the weather in Australia. We approached this by having each member look at an individual state so that we each had lighter datasets to look at. This allowed us to create models and compare and contrast said models. Thus, we were able to reach our desired conclusions through the models and analysis that we each did.

Contributions

As mentioned in the “Intentions” section, we approached the coding by separating the workload by state. Pranav worked on the South Australia dataset, Wen worked on the New South Wales dataset, Nishan worked on the Victoria dataset and Yunru worked on the Queensland dataset. We initially had a fifth member, Zihan, who was meant to work on the Tasmania data, but unfortunately could not continue. For the presentation, Pranav worked on the agenda, background and conclusions, Wen worked on the cleaning, Nishan worked on the exploratory data analysis and Yunru did the modelling of demand. For the report, the background and conclusions were done by Pranav, the cleaning was done by Wen, the exploratory analysis was done mainly by Nishan, with some help from Pranav and Yunru, the modelling of demand was done by Yunru, the establishment of the three regression models was completed by Yunru with some help from Pranav, who used his data to make the second model.

Form of Initial Data

To begin our project, we received two folders of data as our datasets. One folder contained temperature and weather data, consisting of 6 text files with 34 variables separated by region, along with two more text files containing notes and information. The

second folder contained energy demand data, which came as 1,136 separate csv files, containing 5 variables.

The energy demand data was gathered from the Australian Energy Market Operator and the temperature and humidity data comes from the Bureau of Meteorology Automatic Weather Station. Both sets feature data for the 20-year period between 2000 and 2019, and both sets have their values recorded in half hour intervals.

Overview

Without looking at the data we already know that there should be relationships between these two datasets. Hotter and colder weather both should lead to an increase in energy demand, as more people use energy to cool and heat their homes. Similarly, seasons like Autumn and Spring should see less demand for energy, due to the milder weather. Energy demand should also have a large impact on the time of day, as humans would tend to use more energy during the day than late at night.

Data Manipulation and Modelling

Cleaning and wrangling the data

Clean ALL_DEMAND DATA

```
# All states data
all_state = files.rename({'SETTLEMENTDATE': 'Date'}, axis=1)
all_state = all_state.drop(["PERIODTYPE"], axis = 1)
all_state['Date']= pd.to_datetime(all_state['Date'])
all_state.set_index('Date', inplace=True)
all_state
```

Date	REGION	TOTALDEMAND	RRP
2000-01-01 00:30:00	NSW1	6763.57000	15.64
2000-01-01 01:00:00	NSW1	6386.10167	14.06
2000-01-01 01:30:00	NSW1	5990.79500	14.30
2000-01-01 02:00:00	NSW1	5655.97667	14.28
2000-01-01 02:30:00	NSW1	5283.83667	14.17
...	...	...	...
2019-12-31 22:00:00	VIC1	4129.96000	52.91
2019-12-31 22:30:00	VIC1	4083.66000	58.66
2019-12-31 23:00:00	VIC1	4104.95000	54.36
2019-12-31 23:30:00	VIC1	4325.88000	66.87
2020-01-01 00:00:00	VIC1	4372.27000	85.23

1658965 rows × 3 columns

- Dropped “PERIOD TYPE” column
- “SETTLEMENT DATE” column set as index
- format of the time changed

  2000/01/01 00:30 -> 2000-01-01 00:30
- Columns containing numeric values converted to float
- Additional columns added

  eg. “year”, “month”, “Season” to aid graphing

Date	REGION	TOTALDEMAND	RRP
2000-01-01 00:30:00	QLD1	3905.56833	39.40
2000-01-01 01:00:00	QLD1	3855.67500	34.18
2000-01-01 01:30:00	QLD1	3814.44667	35.20
2000-01-01 02:00:00	QLD1	3705.36500	25.53
2000-01-01 02:30:00	QLD1	3615.71333	18.55
...	...	...	...

Select state by 'REGION'

My state is Queensland, in data it's annotated as 'QLD1'

Remove outliers

By using .describe() to get 25% and 75% which is Q1,Q3

Calculate IQR, MAX,MIN

IQR = Q3-Q1

MIN = Q1 − 1.5 IQR

MAX = Q3 + 1.5 IQR

```
iqr=q3-q1
max=q3+(1.5*iqr)
min=q1-(1.5*iqr)
print(iqr,max,min)
```

```
32.26 101.28999999999999 -27.75
```

	TOTALDEMAND	RRP
count	350632.000000	350632.000000
mean	5774.627577	48.860914
std	915.209610	185.276468
min	3089.881670	-1000.000000
25%	5112.922497	20.640000
50%	5777.030000	30.450000
75%	6402.680417	52.900000
max	9988.090000	13882.770000

```
QLD_clear=QLD_temp.drop(QLD_temp[(QLD_temp['RRP']<-27.75)|(QLD_temp['RRP']>101.28999999999999)].index)
QLD_clear
```

If the data is lower than minimum -23.95 or higher than maximum 102.16 we should remove them.

Clean Temperature Data

Read file for Queensland: 'TemperatureData\\HM01X_Data_040913_999999999743964.txt'

Delete Useless Columns

1.We only keep one of two different types of time in the data file.

2.All the columns which were to do with quality such as "Quality of mean sea level pressure" contained the letter N and an empty string so they all had to be removed.

```
temp_cleaned = temp_raw.drop( ['hm',
                               'Year Month Day Hour Minutes in YYYY.1', 'MM.1', 'DD.1', 'HH24.1',
                               'MI format in Local standard time',
                               'Quality of precipitation since 9am local time',
                               'Quality of relative humidity',
                               'Wind speed quality',
                               'Wind direction quality',
                               'Quality of speed of maximum windgust in last 10 minutes',
                               'Quality of mean sea level pressure',
                               'Quality of station level pressure',
                               'AWS Flag',
                               '#',
                               'Quality of air temperature',
                               'Quality of Wet bulb temperature',
                               'Quality of dew point temperature'], axis = 1)
```

Rename Columns

Easier to understand the meaning of each column

```
QLD_temp = temp_cleaned.rename({'Year Month Day Hour Minutes in YYYY': 'year',
                                'MM':'month', 'DD':'day', 'HH24':'hour',
                                'MI format in Local time':'minute',
                                'Precipitation since 9am local time in mm':'Percipitation (mm)',
                                'Air Temperature in degrees C': 'Air temperature (C)',
                                'Wet bulb temperature in degrees C': 'Wet Bulb temperature (C)',
                                'Dew point temperature in degrees C': 'Dew Point temperature (C)',
                                'Relative humidity in percentage %': 'Relative humidity (%)',
                                'Wind speed in km/h':'Wind speed (km/h)',
                                'Wind direction in degrees true': 'Wind direction (degrees)',
                                'Speed of maximum windgust in last 10 minutes in  km/h':'Max wind speed (km/h)',
                                'Mean sea level pressure in hPa': 'Sea level pressure (hPa)',
                                'Station level pressure in hPa': 'Station level pressure (hPa)'}, axis=1)

QLD_temp['Date'] = pd.to_datetime(QLD_temp[['year', 'month', 'day', 'hour', 'minute']])
QLD_temp = QLD_temp.set_index('Date')
QLD_temp
```

Find Missing Values In Data

```
Station Number  has 0  missing values.
Air Temperature in degrees C  has 396  missing values.
Wet bulb temperature in degrees C  has 684  missing values.
Dew point temperature in degrees C  has 418  missing values.
```

Merge ALL DEMAND DATA and TEMPERATURE DATA into one table

```
result = pd.merge(QLD_demand, Finllay_QLD_temp, on='Date')
Result = result.drop(['REGION'],axis =1)
Result
```

Date	TOTALDEMAND	Air Temperature in degrees C	Wet bulb temperature in degrees C	Dew point temperature in degrees C
2000-01-01 00:30:00	3905.56833	22.0	19.2	17.4
2000-01-01 01:00:00	3855.67500	21.7	19.2	17.7
2000-01-01 01:30:00	3814.44667	21.4	19.4	18.2
2000-01-01 02:00:00	3705.36500	21.5	19.6	18.5
2000-01-01 02:30:00	3615.71333	21.3	19.3	18.1
...	...	...	...	...
2019-12-31 22:00:00	6769.25000	24.9	21.3	19.3
2019-12-31 22:30:00	6716.85000	25.0	21.4	19.4
2019-12-31 23:00:00	6614.27000	25.0	21.5	19.6
2019-12-31 23:30:00	6432.74000	24.9	21.4	19.5
2020-01-01 00:00:00	6218.39000	24.6	21.6	19.9

Final Check

Check if there is Nan in the table, if it is dropped

```
Result=Result.apply(pd.to_numeric, errors='coerce')
Result=Result.dropna()
Result.info()
```

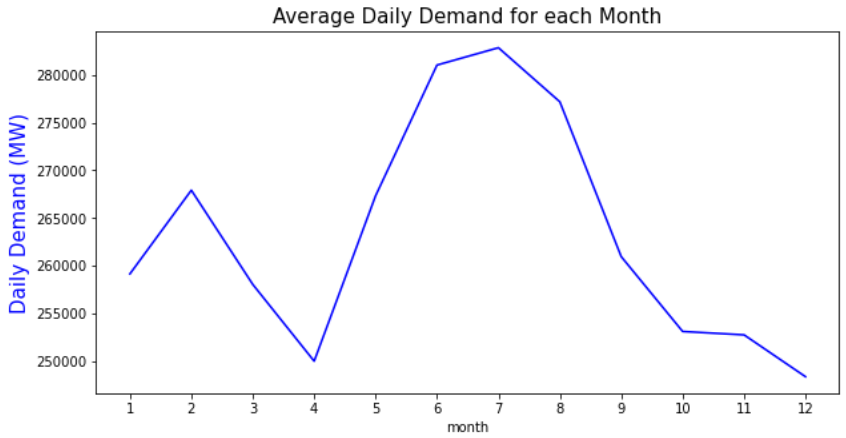
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 332860 entries, 2000-01-01 00:30:00 to 2020-01-01 00:00:00
Data columns (total 4 columns):
 #   Column                              Non-Null Count   Dtype
---  ------                              --------------   -----
 0   TOTALDEMAND                         332860 non-null  float64
 1   Air Temperature in degrees C        332860 non-null  float64
 2   Wet bulb temperature in degrees C   332860 non-null  float64
 3   Dew point temperature in degrees C  332860 non-null  float64
dtypes: float64(4)
memory usage: 12.7 MB
```

Exploratory data analysis

How does daily demand change throughout the year?

This graph shows the average daily demand for each month.

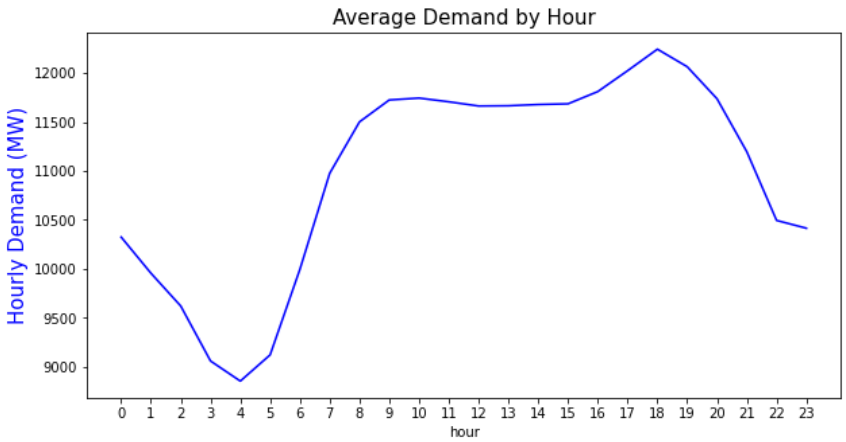
Daily demand is lowest in April and December at approximately 250 GW and highest in winter reaching a peak of over 280 GW in July.



How does demand change in the day?

This graph shows the average hourly demand throughout the day.

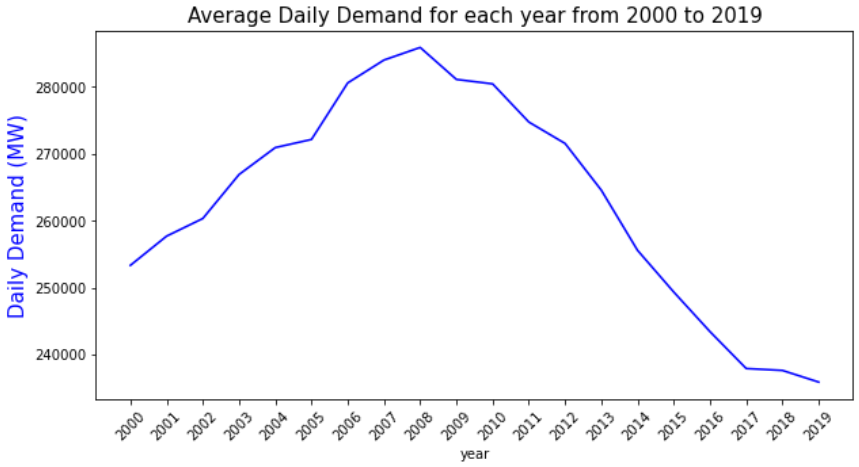
We observe that demand is lowest around 4am and increases rapidly as people start waking up until finally stabilizing at around 11.5 GW by 9am. We see another peak at 6pm which may be because people are coming home from work. As the day ends we see the demand dip as people go to sleep.



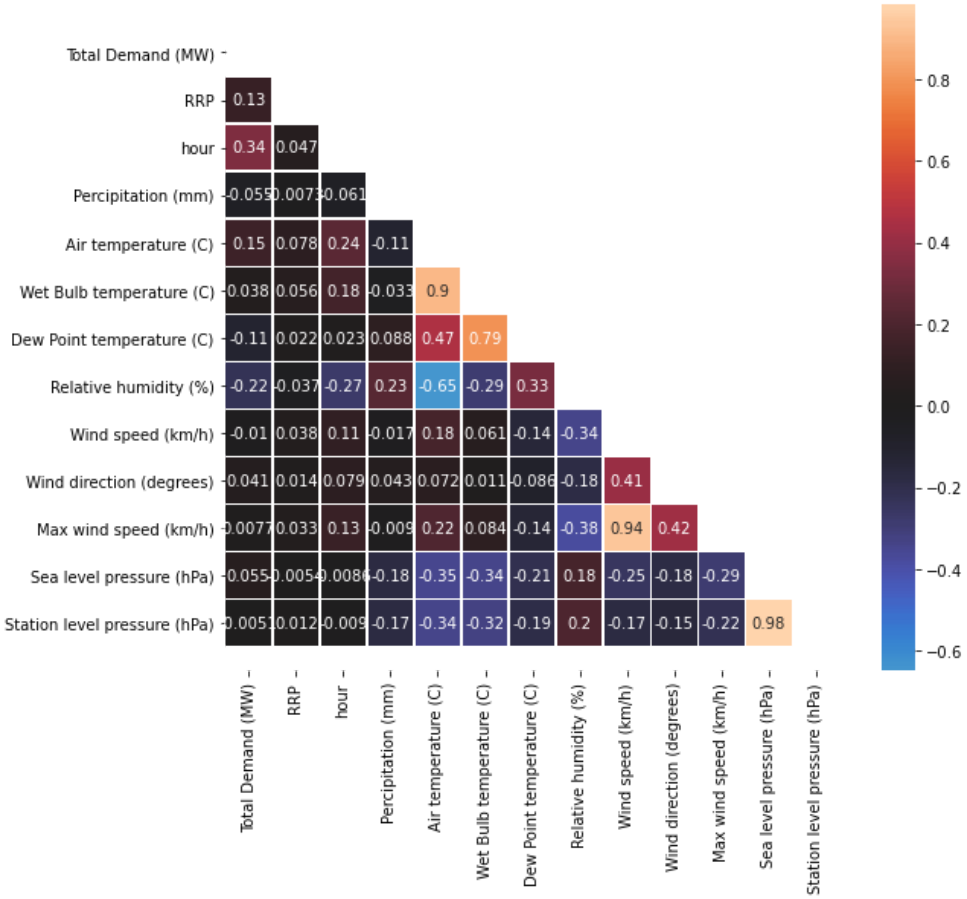
What are the trends in demand from 2000 to 2019?

This graph shows the daily average demand for the years from 2000 to 2019.

Daily average demand peaked in 2009 at 280 GW and has been declining ever since up to 2019 when it was less than 240 GW.



Heatmap



Looking at the first column, we see that the variables RRP, time of day, air temperature, dew point temperature and relative humidity all have an absolute correlation greater than 0.1 with total demand.

We should also take note of the very high correlations (> 0.5) between the following variables as they may result in multicollinearity when the model is created:

- Sea level pressure & station level pressure (0.98)
- Wind speed & max wind speed (0.94)
- Wet bulb temperature & dew point temperature (0.79)
- Air temperature & relative humidity (-0.65)
- Air temperature & wet bulb temperature (0.9)

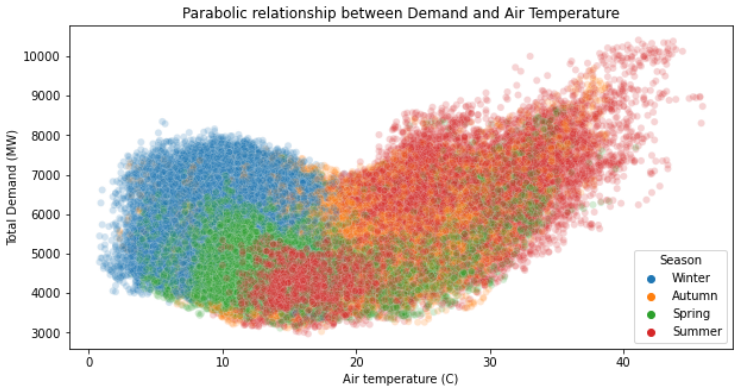
Exploring the relationship demand has with other variables

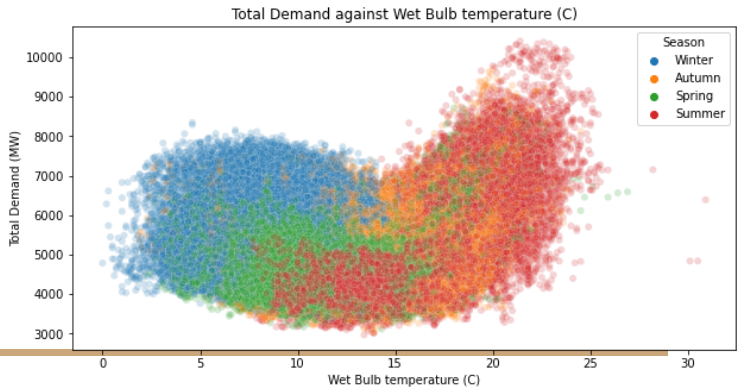
Scatter plots will be used to investigate the relationship between other variables and demand. Each point on the scatter plots represents a 30 minute period.

Relationship of Air Temperature and Wet Bulb temperature with Demand

We can see that both wet bulb temperature and air temperature have a parabolic relationship with demand.

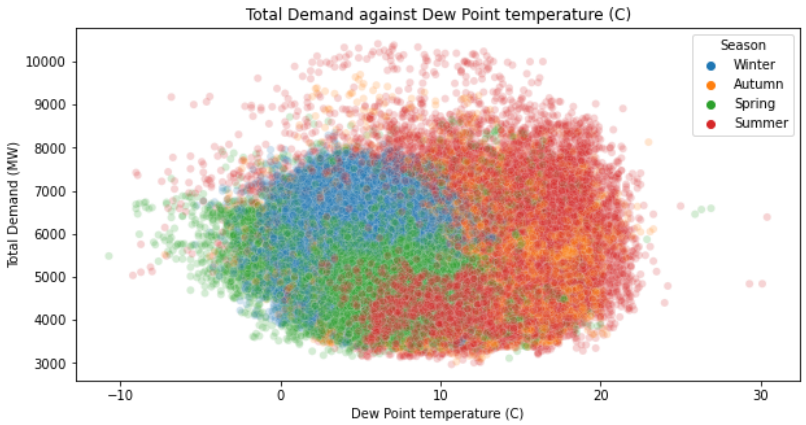
Therefore, it may be useful to include wet bulb temperature squared and air temperature squared in the model.





Relationship of Dew Point temperature with Demand

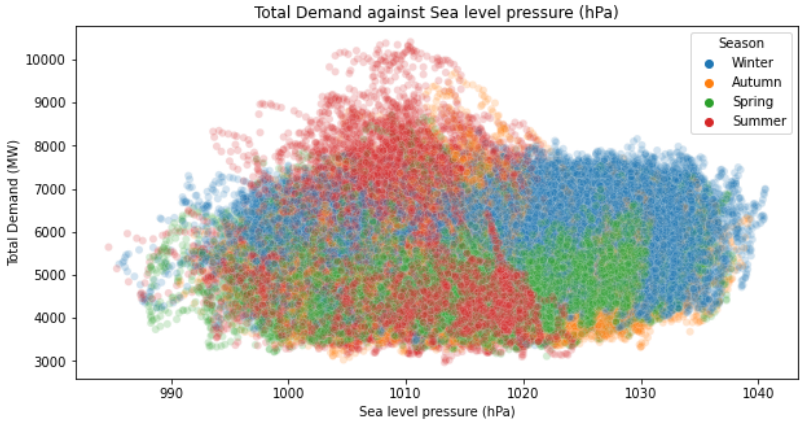
The scatter plot shows us that there is no clear relationship between dew point temperature and total demand.



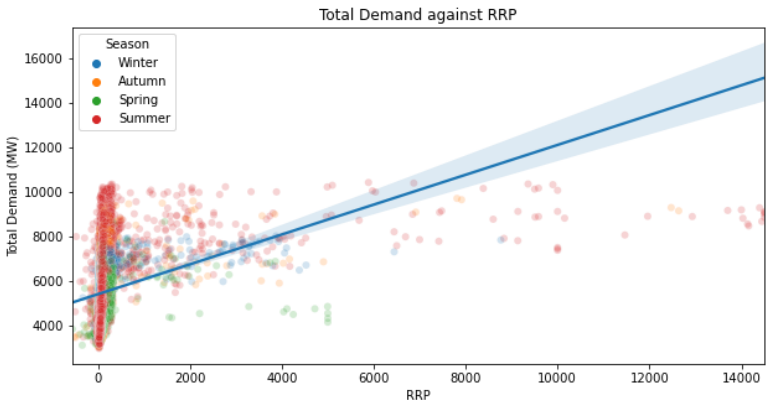
Relationship of Station level pressure with Demand

There also doesn't appear to be a relationship between total demand and sea level pressure since as sea level pressure increases, there is no apparent change in total demand.

Station level pressure and sea level pressure have a correlation of 0.98 so we can conclude that station level pressure also has no relationship with total demand.



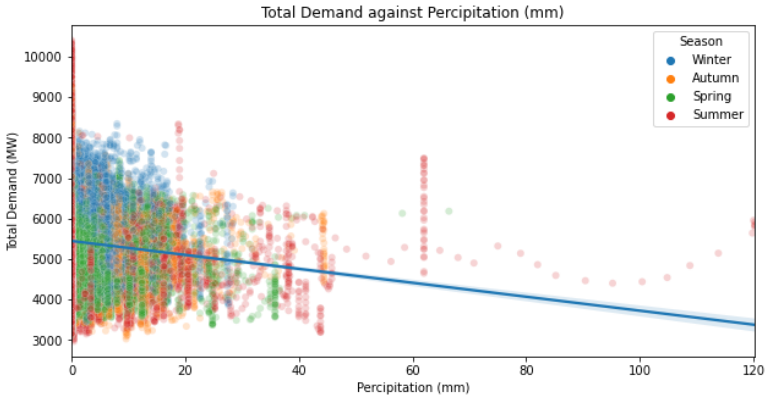
Relationship of RRP with Demand



RRP appears to have a positive linear relationship with total demand.

The exact linear relationship, which is represented by the blue line, is calculated by minimizing the sum of square errors. However, may be impractical because this chooses a model which is heavily influenced by the outliers. The vast majority of the data points have an RRP below 1000 so it may be better to use a different type of error function which produces a linear model with a higher slope. The current error function causes us to sacrifice a significant amount of accuracy for the majority of points so that the outlier won't have large errors.

Relationship of Precipitation with Demand

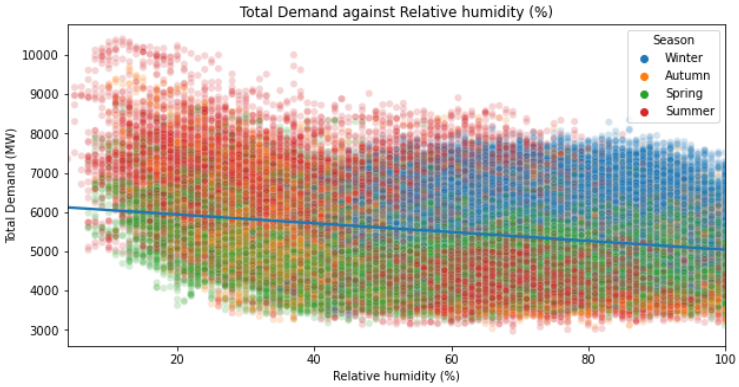


From observation of the plot, the relationship between demand and precipitation appears to be negative and linear. The majority of the data can be seen between 0 and 40mm of precipitation, but the data does appear to be trending downwards and following the trendline, unlike the previous model. A linear model seems like the suitable choice for modelling this relationship, and this is evidenced by the data points following the linear trendline .

Relationship of Humidity with Demand

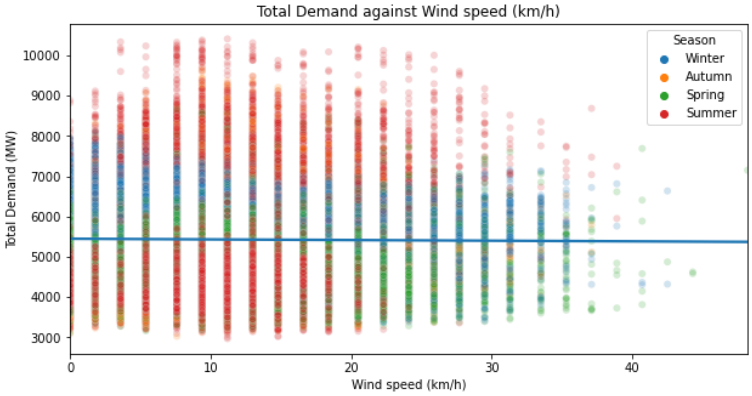
The relationship between total demand and relative humidity is a negative logarithmic relationship.

Therefore it may be better to use -ln(relative humidity) for the model instead.



Relationship between Wind speed and Demand

The scatter plot shows no relationship between total demand and wind speed. This is backed up by the linear regression model which produces a horizontal line indicating that the behaviour of total demand does not change on average as wind speed increases.



Since max wind speed and wind speed have a correlation of 0.94, we can conclude that max wind speed also does not have a relationship with total demand.

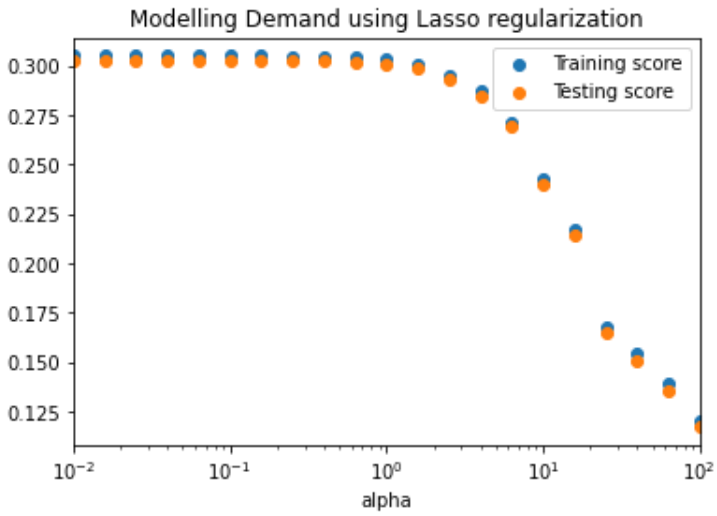
Modelling demand

Lasso Regularization

We use lasso regularization in order to deal with the multicollinearity that occurs due to several variables being highly correlated such as sea level pressure and station level pressure.

Choosing appropriate alpha

We plot both the training score and testing score for several different alpha values. The value of 5 is chosen for the alpha so that there is a significant force to reduce multicollinearity but not too high as to reduce the accuracy of the model.

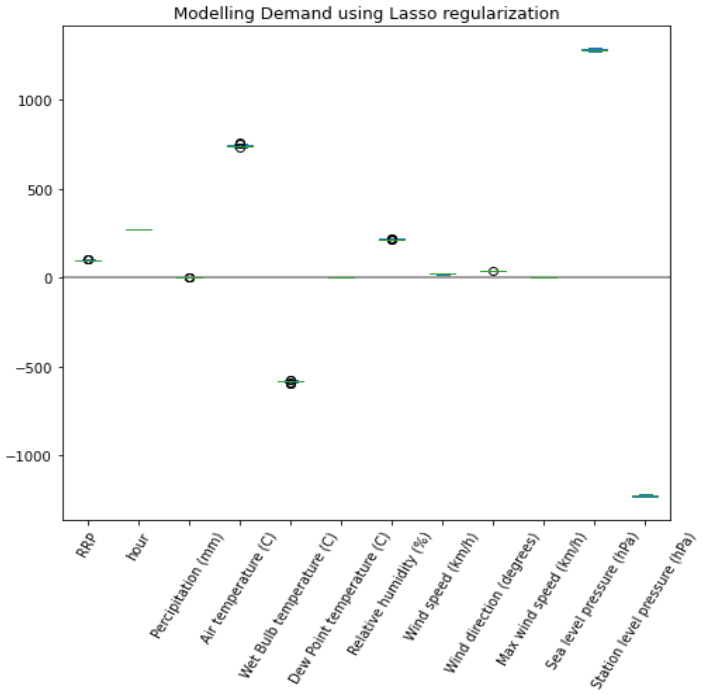


Testing the model's consistency

In order to test how much the coefficients of the model change, we created 10 models each using a different subset of data. The variability of the coefficients from these models are displayed by the box plots.

The coefficients of these models are very similar thus the box plots look almost like horizontal lines.

However, there are clear signs of multicollinearity in the model which means the model does not capture the true relationship between some variables and demand.



This includes the variables air temperature and wet bulb temperature where air temperature has a large positive coefficient and wet bulb temperature has a roughly equally large negative coefficient. Furthermore, sea level pressure and station level pressure are also affected by multicollinearity as sea level pressure's coefficient is large and negative whereas station level pressure's coefficient is large and negative.

Calculating score

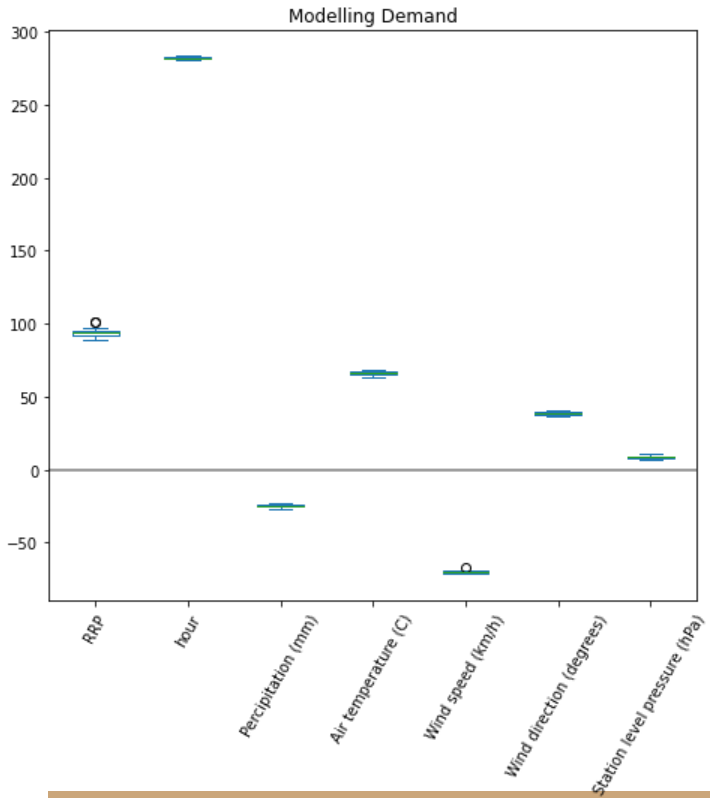
```
Training score is 0.2808788997619297
Testing score is 0.278537140384023
```

The training score and testing score are very similar therefore, the model is not overfitting the data. The testing score tells us that this model can explain 28% of the variation in demand.

Since this model does not represent the true relationship of some variables due to multicollinearity, we will create another model which does not contain dependent variables which have an absolute correlation coefficient greater than 0.5.

Model demand using dependant variables which aren't highly correlated

Testing the model's variability



How well does the model fit the data?

```
Training score is 0.1394955833161311
Testing score is 0.13658027845339715
```

From looking at this box plot of these variables we notice that the hour and RRP variables have the highest coefficients, meaning increases in demand should lead to considerable increases in these two variables. The precipitation and wind speed variables, on the other hand, have negative coefficients, meaning as precipitation and wind speed increase, demand is predicted to decrease. The variables of air temperature and wind direction both have moderate positive coefficients, and station level has a low positive coefficient.

When examining the training and testing data, we see their values are very close to each other, meaning that the model is not overfitting the data. A testing score of 0.13658 indicates that this model can explain about 14% of the variation in demand.

MODEL SETTING :

For the part of establishing the regression model, we will only consider the three variables related to temperature(Air temperature in degrees C, Wet bulb temperature in degrees C, Dew point temperature in degrees C) , use them as independent variables, and use total demand as the dependent variable to verify three different types of regression models.

Multiple regression model

1：**Linear regression**

For linear regression, our ideal model would look something like this ：

Y(Total Demand) **= A1 * X1**(Air temperature in degrees C) **+**

A2 * X2(Wet bulb temperature in degrees C) **+**

A3 * X3(Dew point temperature in degrees C) **+ C**(interception)

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
col=[i for i in Result.columns.tolist() if i not in ['TOTALDEMAND'] ]
X_train, X_test, Y_train, Y_test=train_test_split(Result[col],Result['TOTALDEMAND'],
                                                  test_size=0.3,random_state=0)
print(X_train.shape, X_test.shape)
```

```
(233453, 3) (100052, 3)
```

```
clf = LinearRegression().fit(X_train, Y_train)
print(clf.coef_, clf.intercept_)
```

```
[ 381.96088054 -711.21721509  269.39807085] 8649.208655946073
```

Because of the large amount of data,

we first split the data into training set and

test set, and split the data according to the

proportion of training set and test set

accounting for 70% and 30% of the total

data. Print the shape of the training set and test set by code are (233453, 3), (100052, 3) respectively.

And then we found the correlation coefficients of the independent variables and the interception by fitting the train.

We got the Multiple linear regression model：

Y(Total Demand) **= 381.961 * X1**(Air temperature in degrees C) **-**

711.217 * X2(Wet bulb temperature in degrees C) **+**

269.398 * X3(Dew point temperature in degrees C) **+ 8649.209**(interception)

```
train
train r mean_squared_error 1281.4489592680409
train mean_absolute_error 1046.9565471638637
train r2_score 0.055604205535837115
test
test r mean_squared_error 1281.258199182217
test mean_absolute_error 1044.9519415623909
test r2_score 0.053963369542727735
```

Then, MSE and MAE coefficients were calculated by Python to determine the difference

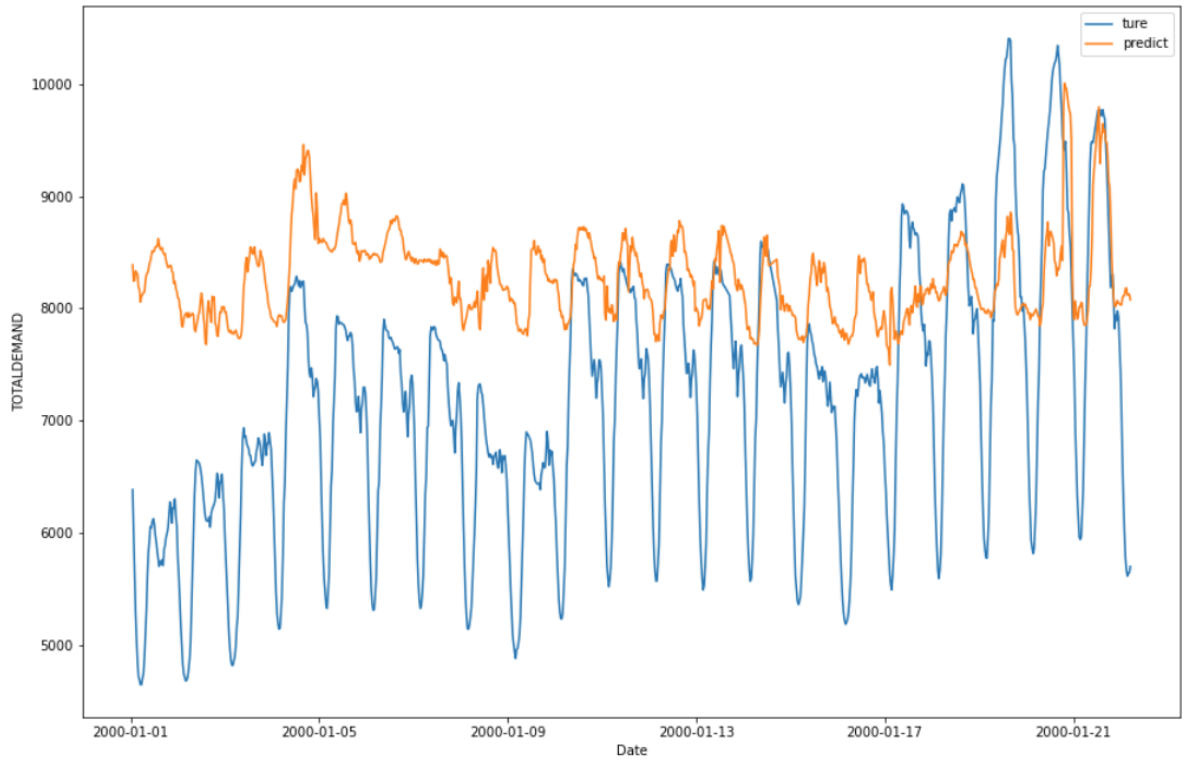
between the predicted value and the actual value, and the fitting degree of the r-square

a coefficient judgment model was calculated.

In order to intuitively show the difference between the expected value and the actual value, we can add a column of predicted values in the table, which can be calculated according to our model.

Date	TOTALDEMAND	Air Temperature in degrees C	Wet bulb temperature in degrees C	Dew point temperature in degrees C	predict
2000-01-01 01:00:00	6386.10167	17.4	13.5	10.0	8387.876282
2000-01-01 01:30:00	5990.79500	16.8	14.1	11.9	8243.825759
2000-01-01 02:00:00	5655.97667	16.5	13.7	11.3	8252.085539
2000-01-01 02:30:00	5283.83667	16.4	13.0	9.9	8334.584202
2000-01-01 03:00:00	5047.90167	16.3	13.2	10.5	8315.783513

Due to the large amount of data, we present the first 1000 groups of data and intuitively compare the difference between the expected value and the actual value through a line plot.



In short, according to analysis of the influence of temperature on total demand in different seasons, we can roughly get that total demand and temperature are not a linear regression relationship. After determining the linear regression model above, by verifying the coefficients such as MAE, MSE, and r-squared, it can also be refined, and the regression model is not a linear model with high curvature.

2：**Quadratic regression**

In order to find a higher goodness of fit and verify our initial conjecture, we try to set a quadratic regression model.

For Quadratic regression, our ideal model would look something like this:

Y(Total Demand) **= A1 * X1^2**(Air temperature in degrees C) **+A2 * X1**

B2 * X2^2(Wet bulb temperature in degrees C) **+ B2 * X2**

C3 * X3^2(Dew point temperature in degrees C) **+ C3 * X3 + T**(interception)

Because we need to calculate more correlation coefficients here, we can't directly use the fit function to get the quadratic coefficients of the independent variables, so we add the square of the variable to the table.

Date	TOTALDEMAND	Air Temperature in degrees C	Wet bulb temperature in degrees C	Dew point temperature in degrees C	Air Temperature in degrees C**2	Wet bulb temperature in degrees C**2	Dew point temperature in degrees C**2
2000-01-01 01:00:00	6386.10167	17.4	13.5	10.0	302.76	182.25	100.0

Then repeat the same steps as above to split the data and find the correlation coefficients.

```
col=[i for i in data.columns.tolist() if i not in ['TOTALDEMAND','predict'] ]
X_train, X_test, Y_train, Y_test=train_test_split(data[col], data['TOTALDEMAND'],
                                                  test_size=0.3, random_state=0)
print(X_train.shape, X_test.shape)
```

```
(233453, 6) (100052, 6)
```

```
clf = LinearRegression().fit(X_train, Y_train)
print(clf.coef_, clf.intercept_)
```

```
[ 1313.36133819 -2449.7022611    705.63225606    -3.28781121
    -4.00282906    19.71239977] 11053.617725676659
```

We got the Quadratic regression model：

Y(Total Demand) **= -3.288 * X1**(Air temperature in degrees C)**^2 + 1313.361 * X1 -**

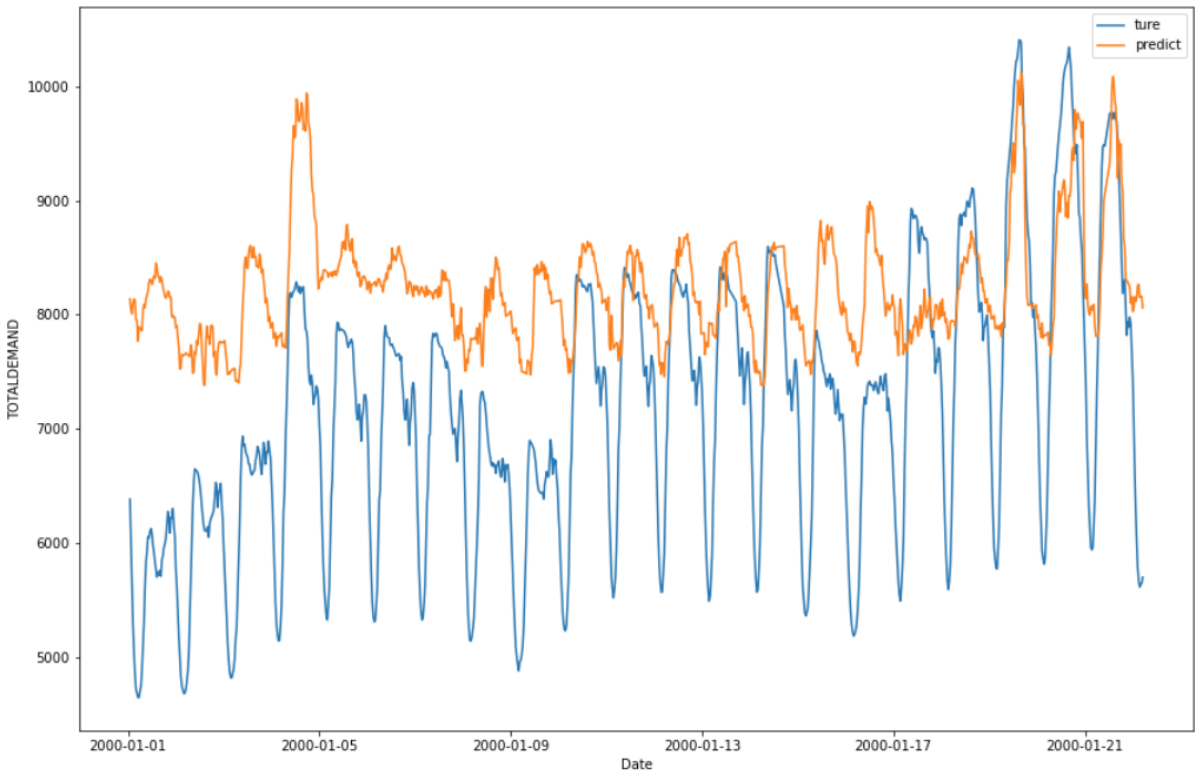
4.003 * X2(Wet bulb temperature in degrees C)**^2 - 2449.702 * X2 +**

19.712 * X3(Dew point temperature in degrees C)**^2 + 705.632 * X3 +**
11053.618(interception)

```
train
train r mean_squared_error 1246.0283451958378
train mean_absolute_error 1016.894531025273
train r2_score 0.1070908714748473
test
test r mean_squared_error 1247.8064811313227
test mean_absolute_error 1017.0237726352533
test r2_score 0.1027176780935849
```

After calculating the MAE, MSE, and R-square coefficients for the test set and training set, we can find that the quadratic regression model has a higher goodness of fit than the linear regression model by comparing these coefficients.

Date	TOTALDEMAND	Air Temperature in degrees C	Wet bulb temperature in degrees C	Dew point temperature in degrees C	Air Temperature in degrees C**2	Wet bulb temperature in degrees C**2	Dew point temperature in degrees C**2	predict
2000-01-01 01:00:00	6386.10167	17.4	13.5	10.0	302.76	182.25	100.00	8137.753703
2000-01-01 01:30:00	5990.79500	16.8	14.1	11.9	282.24	198.81	141.61	8042.028821
2000-01-01 02:00:00	5655.97667	16.5	13.7	11.3	272.25	187.69	127.69	8007.482059
2000-01-01 02:30:00	5283.83667	16.4	13.0	9.9	268.96	169.00	98.01	8103.618099
2000-01-01 03:00:00	5047.90167	16.3	13.2	10.5	265.69	174.24	110.25	8136.776958



This is the comparison between the actual value and the expected value of the quadratic regression model in a line plot

Random forest (regression model)

Although the quadratic regression is better than the linear regression model, by consulting the data, we know that the correlation coefficients of the regression model by random forest could be better for our data.

```
from sklearn.ensemble import RandomForestRegressor
col=[i for i in Result.columns.tolist() if i not in ['TOTALDEMAND'] ]
X_train, X_test, Y_train, Y_test=train_test_split(Result[col], Result['TOTALDEMAND'],
                                                  test_size=0.3, random_state=0)
print(X_train.shape, X_test.shape)
```

```
(233453, 3) (100052, 3)
```

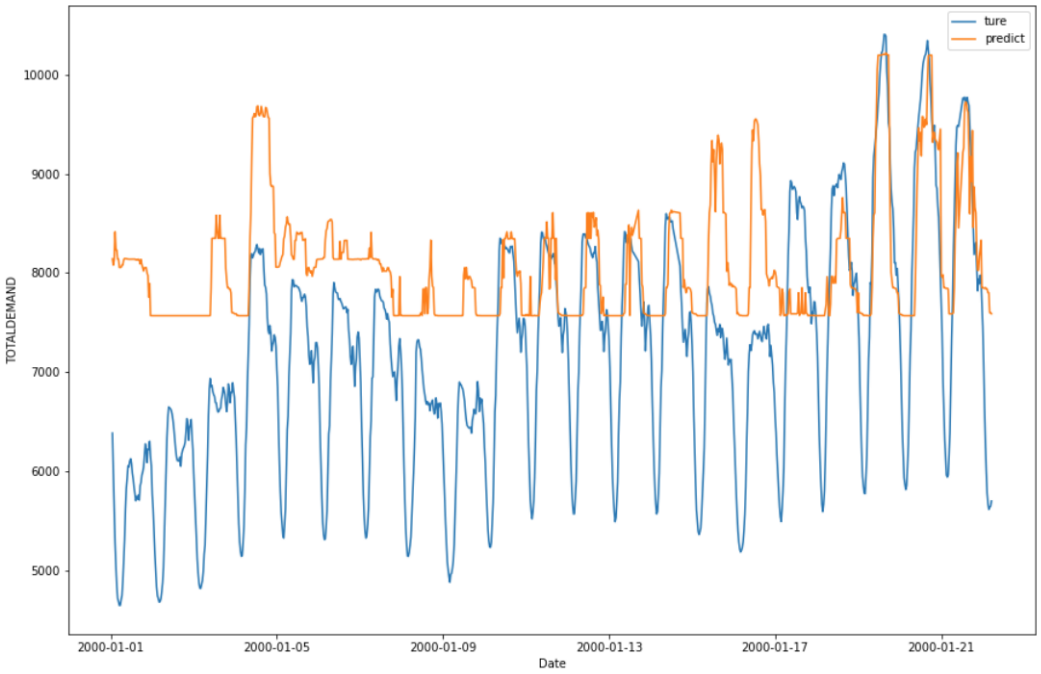
```
clf=RandomForestRegressor(max_depth=5, n_estimators=300, random_state=0)
clf.fit(X_train, Y_train)
```

```
RandomForestRegressor(max_depth=5, n_estimators=300, random_state=0)
```

```
train
train r mean_squared_error 1203.2490315887917
train mean_absolute_error 981.0882038530037
train r2_score 0.1673500493372191
test
test r mean_squared_error 1206.559503462365
test mean_absolute_error 982.4422578028343
test r2_score 0.1610576334292284
```

Comparing the R-squared coefficient of the regression model, this regression model has a higher r-squared coefficient than the above two models, which means that the model has a higher goodness of fit.

Date	TOTALDEMAND	Air Temperature in degrees C	Wet bulb temperature in degrees C	Dew point temperature in degrees C	predict
2000-01-01 01:00:00	6386.10167	17.4	13.5	10.0	8142.028855
2000-01-01 01:30:00	5990.79500	16.8	14.1	11.9	8080.127028
2000-01-01 02:00:00	5655.97667	16.5	13.7	11.3	8132.608897
2000-01-01 02:30:00	5283.83667	16.4	13.0	9.9	8414.560443
2000-01-01 03:00:00	5047.90167	16.3	13.2	10.5	8232.434575



This is a comparison graph of the actual value and expected value of this model.

Conclusions

Did we end up meeting our initial aim of finding the correlation between energy usage and weather in Australia? From our analysis and modelling, we have found that demand has a low negative linear correlation with air temperature, however there is a clear parabolic relationship. What else did we find? We've found that the highest demand occurs in winter, whereas the 30-minute periods with the highest demand occur in Summer, that demand is lowest at around 4am and highest around 6pm, and how much impact exactly demand has on our other variables. We've also found that the accuracy of the model is not as high as you may expect it to be despite all these correlations. To conclude, from our work on this project we've found that although we can see some correlations, like demand increasing as temperature peaks and troughs and the relationship between demand and time of day, we can't build a highly accurate model from just looking at the impact of temperature, as demand is impacted by a variety of different other variables as well.

Appendix

Function to create line plots:

```
def graph(x_axis, rotate = False):
    fig, ax1 = plt.subplots(figsize = (10, 5))

    # We have to sort this because when x_axis == 'hour', the array produced is not in order but instead it starts at 9
    # which messes up the graphs as the line starts from 9 instead of 0.
    x = np.sort(demand_vic[x_axis].unique())

    if x_axis == "year":
        y1 = demand_vic.groupby(["year", "month", "day"]).sum().groupby("year").mean()['Total Demand (MW)']
    elif x_axis == "month":
        y1 = demand_vic.groupby(["year", "month", "day"]).sum().groupby("month").mean()["Total Demand (MW)"]
    elif x_axis == "hour":
        y1 = demand_vic.groupby(["year", "month", "day", "hour"]).sum().groupby("hour").mean()["Total Demand (MW)"]

    ax1.plot(x, y1, 'b')
    plt.xticks(demand_vic[x_axis].unique())

    if rotate:
        ax1.tick_params(axis='x', labelrotation = 45)

    ax1.set_xlabel(x_axis)
    if x_axis != "hour":
        ax1.set_ylabel('Daily Demand (MW)', color = 'b', fontsize = 15)
    else:
        ax1.set_ylabel('Hourly Demand (MW)', color = 'b', fontsize = 15);
```

Function to create scatterplots:

```
def scatter_plot_against_demand(column):
    fig = plt.figure(figsize = (10, 5))

    sns.scatterplot(data = vic_all,
                    x = column,
                    y = 'Total Demand (MW)',
                    alpha = 0.2, hue = 'Season',
                    hue_order = ["Winter", "Autumn", "Spring", "Summer"])

    if "temperature" not in column and "pressure" not in column:
        sns.regplot(data = vic_all, x = column, y = 'Total Demand (MW)',scatter = False)
    plt.title("Total Demand against " + column)
```

```
for c in corrs.columns[1:]:
    if c != 'hour':
        scatter_plot_against_demand(c)
```

Heatmap:

```
corrs = vic_all.corr()
mask = np.triu(np.ones_like(corrs, dtype=bool))
f, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(corrs, mask=mask, center=0, annot=True, square=True, linewidths=.5)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5);
```

Lasso regularization Model:

Note: The code for the model without highly correlated dependent variables uses the same approach but without the regularisation and fewer variables.

Finding optimal alpha parameter

```
# Construct normalised features
X = vic_all.drop(['Total Demand (MW)','Season'],axis=1)
Y = vic_all['Total Demand (MW)']
nX =(X-X.mean())/X.std()
feature_names = X.columns
```

```
# Split data into training and testing sets.
X_train, X_test, Y_train, Y_test = train_test_split(nX,Y,test_size=0.8, random_state=1)
linear = LinearRegression()
```

```
# create an array of 21 alpha values logarithmically distributed between 10**(-2) and 10**2
alfas = np.logspace(-2, 2, num=21)

# create two arrays for storage of the same size as alfas, but filled with zeros
lasso_training_score = np.zeros_like(alfas)
lasso_rsquared = np.zeros_like(alfas)


# loop over the values in the alfas array, at each loop the current value is alfa
# and index (idx) is incremented by 1, starting at 0
for idx, alfa in enumerate(alfas):
    lasso = Lasso(alpha=alfa) # instantatiate Lasso regularization with the current alfa
    lasso.fit(X_train,Y_train) # train the model to our data set
# calculate the training score and store in the array lasso_training_score
    lasso_training_score[idx] = lasso.score(X_train,Y_train)
    preds_linear = lasso.predict(X_test) # calculate the model prediction for the test data
# calculate the correlation between the predicted and actual test data and store in the  array lasso_rsquared
    lasso_rsquared[idx] = r2_score(Y_test,preds_linear)


plt.scatter(alfas,lasso_training_score,label='Training score') # plot the training score against alpha
plt.scatter(alfas,lasso_rsquared,label='Testing score') # plot the testing score against alpha
plt.xscale('log') # make the x-axis a logarithmic scale
plt.gca().set_xlim(left=.01, right=100); # fix the x-axis limits
plt.title('Modelling Demand using Lasso regularization')
plt.xlabel('alpha')
plt.legend(loc='best');
```

Plotting coefficients

```
scores = cross_validate(Lasso(alpha=5), nX, Y, cv=RepeatedKFold(n_splits=10, n_repeats=10,random_state=23),
                        return_estimator=True)


coefs = pd.DataFrame([est.coef_ for est in scores['estimator']],columns=feature_names)

ax = coefs.plot(kind='box',figsize=(10,7))
plt.title('Modelling Demand using Lasso regularization')
plt.axhline(y=0, color='.5')
plt.subplots_adjust(left=.3)

# Rotate x-axis labels
plt.xticks(rotation=60);
```

Testing and training scores

```
lasso = Lasso(alpha=5)

lasso.fit(X_train,Y_train)
training_score = lasso.score(X_train,Y_train)
preds_linear = lasso.predict(X_test)
testing_score = r2_score(Y_test,preds_linear)

print("Training score is", training_score)
print("Testing score is", testing_score)
```

Find the correlation coefficient

```
from sklearn.metrics import mean_squared_error #MSE
from sklearn.metrics import mean_absolute_error #MAE
from sklearn.metrics import r2_score#R square
y_predict_train=clf.predict(X_train)
print("train")
print("train r mean_squared_error %s"%mean_squared_error(Y_train,y_predict_train)**0.5)
print("train mean_absolute_error %s"%mean_absolute_error(Y_train,y_predict_train))
print("train r2_score %s"%r2_score(Y_train,y_predict_train))
print("test")
y_predict_test=clf.predict(X_test)
print("test r mean_squared_error %s"%mean_squared_error(Y_test,y_predict_test)**0.5)
print("test mean_absolute_error %s"%mean_absolute_error(Y_test,y_predict_test))
print("test r2_score %s"%r2_score(Y_test,y_predict_test))
```

```
train
train r mean_squared_error 1281.4489592680409
train mean_absolute_error 1046.9565471638637
train r2_score 0.055604205535837115
test
test r mean_squared_error 1281.258199182217
test mean_absolute_error 1044.9519415623909
test r2_score 0.053963369542727735
```

Plot line plots for expected and actual values

```
import seaborn as sns
fig = plt.gcf()
fig.set_size_inches(15,10)
n=1000
sns.lineplot(x=Result.index[0:n],y=Result.TOTALDEMAND.head(n),label='ture')
sns.lineplot(x=Result.index[0:n],y=Result.predict.head(n),label='predict')
```