2022

Statistical Regression Analysis using Python

LAKSHAY GOEL

Table of Contents

Introduction:

Problem Statement:

Victoria, the second-most populous state in Australia as of 2020, is home to 6.7 million people. 5 million of them reside or work in Melbourne, the state's capital. Australia was among the first countries to close international borders in 2020, and then interstate borders were shut down. Most of Victoria's population now works from home as a result of some of the harshest limitations on commercial activities that were imposed in response to the pandemic.

Objective:

In this time series project, the electricity demand is aimed to be predicted. The dataset used is the Daily Electricity Price and Demand Data which contains the electricity daily price, demand, and weather data about Victoria, Australia.

Dataset:

The dataset covers 2016 days between 1 January 2015 and 6 October 2020. During some intraday intervals, RRP was negative, so energy producers were paying consumers rather than wise versa. Below is a brief description of the data:

date: DateTime, the date of the recording

demand: float, total daily electricity demand in MWh

RRP: float, a recommended retail price in AUD$ / MWh

demand_pos_RRP: float, a total daily demand at positive RRP in MWh

RRP_positive: float, an averaged positive RRP, weighted by the corresponding intraday demand in AUD$ / MWh

demand_neg_RRP: float, a total daily demand at negative RRP in MWh

RRP_negative: float, an average negative RRP, weighted by the corresponding intraday demand in AUD$ / MWh

frac_at_neg_RRP: float, a fraction of the day when the demand was traded at negative RRP

min_temperature: float, the minimum temperature during the day in Celsius

max_temperature: float, the maximum temperature during the day in Celsius

solar_exposure: float, total daily sunlight energy in MJ/m^2

rainfall: float, daily rainfall in mm

school_day: boolean, if students were at school on that day

holiday: boolean, if the day was a state or national holiday

Target Variable:

demand: float, total daily electricity demand in MWh

Data preparation:

Import dataset

Import the dataset using pandas read_csv and provide the path where the file is stored.

```
# Importing dataset and examining it
dataset = pd.read_csv("Energy_dataset.csv")
pd.set_option('display.max_columns', None) # to make sure you can see a
ll the columns in output window
```

Examining the dataset

Using commands to examine the data frame to make sure the data is imported correctly and if there are any missing values in the dataset.

- **dataset.head()**- Displays the top 5 rows in the data.
- **dataset.shape**- Displays the number of rows and columns in the dataset.
- **dataset.info()**- Displays the data types and missing values in the dataset.
- **dataset.describe()**- Displays the basic statistics like mean, min, max, etc.

```
print(dataset.head())
print(dataset.shape)
print(dataset.info())
print(dataset.describe())
```

Inference:

- Dataset had 3 missing values, so as the algorithm does not accept null values the solution is to remove the rows completely
- Total number of input variables: 12
- Total number of instances: 2103
- The dataset contains non-numerical columns like 'school_day', and 'holiday' which need to be converted into numerical features.
- Looking at the statistics from the dataset.describe, there is clearly a big contrast in the distribution of the columns. For e.g., the 'RRP' column ranges from -6.07 and 4549.64 with a mean of 3.36 whereas the 'demand_pos_RRP' column ranges from 41988.24 and 170653.84 with a mean of 119251.74. Passing the data in its current state will make the algorithm biased towards columns with higher values.

Step 1: Converting non-numerical features to numerical features

The dataset has 3 categories that need to be transformed into numerical columns:

- Holiday
- school_day

Holiday Column contains 2 labels that can be converted into numerical:

- No: 0
- Yes: 1

school_day Column contains 2 labels that can be converted into numerical:

- No: 0
- Yes: 1

```
# Converting Categorical features into Numerical features
dataset['school_day'] = dataset['school_day'].map({'N': 0,'Y': 1})
dataset['holiday'] = dataset['holiday'].map({'N': 0,'Y': 1})
print(dataset.info())
```

Step 2: Plotting a correlation map

Correlation coefficients indicate the strength of the linear relationship between two variables. A coefficient greater than 0 is a positive relationship, less than 0 is a negative relationship, and a value of 0 indicated no relationship. [1]

Positive Correlation:

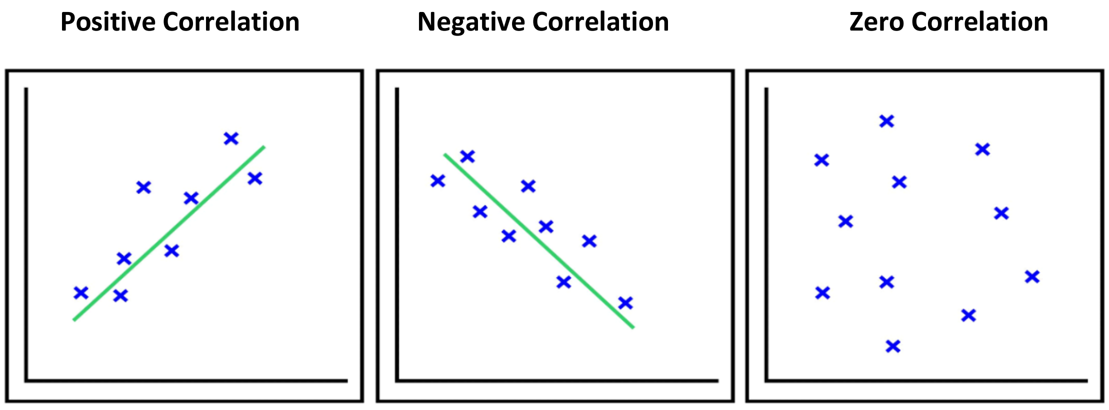
In positive correlation, when one set of information increases the other one also increases. Upon plotting data on the graph, the points lie close to a straight line having a positive gradient i.e. from the lower left to the top right corner.

Negative Correlation:

In Negative Correlation, when one set of information decreases the other one increases. Upon plotting data on the graph, the points lie close to a straight line having a negative gradient i.e. from the top left to the bottom right corner.

Zero Correlation:

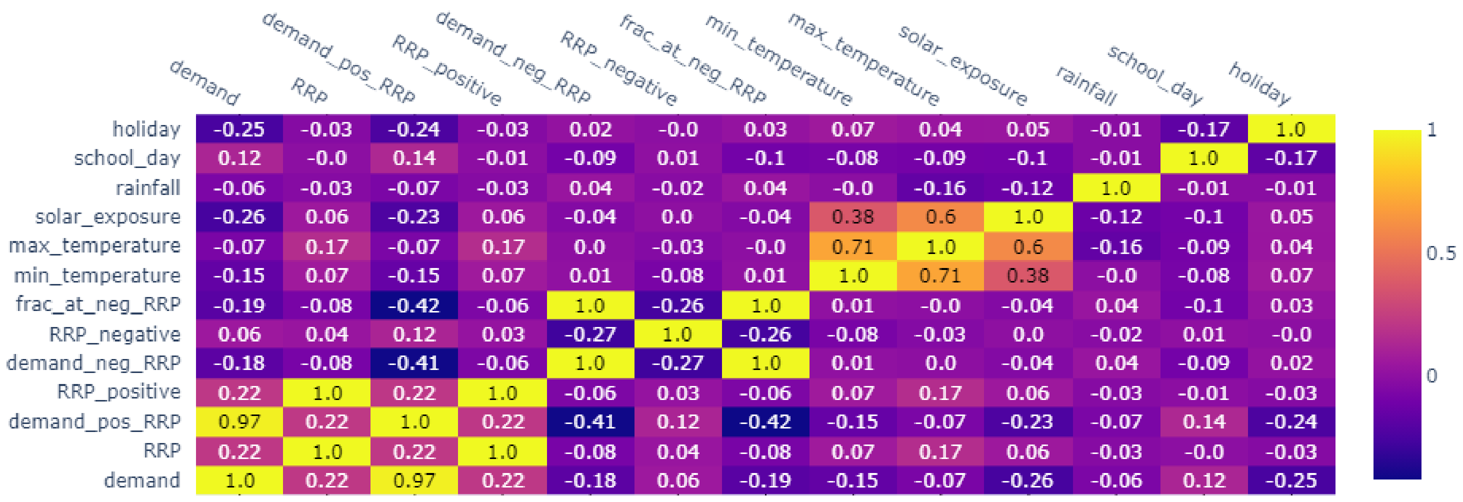
In zero correlation, there is no pattern between all the points. Hence, in plotting the graph there is no connection between the points.[2]



To find out the correlation between the variables in the dataset, use the correlation heatmap:

```
corrs = dataset.corr()
figure = ff.create_annotated_heatmap(
    z=corrs.values,
    x=list(corrs.columns),
    y=list(corrs.index),
    annotation_text=corrs.round(2).values,
    showscale=True)
figure.show()
```

Correlation Heatmap:



	demand	RRP	demand_pos_RRP	RRP_positive	demand_neg_RRP	RRP_negative	frac_at_neg_RRP	min_temperature	max_temperature	solar_exposure	rainfall	school_day	holiday
holiday	-0.25	-0.03	-0.24	-0.03	0.02	-0.0	0.03	0.07	0.04	0.05	-0.01	-0.17	1.0
school_day	0.12	-0.0	0.14	-0.01	-0.09	0.01	-0.1	-0.08	-0.09	-0.1	-0.01	1.0	-0.17
rainfall	-0.06	-0.03	-0.07	-0.03	0.04	-0.02	0.04	-0.0	-0.16	-0.12	1.0	-0.01	-0.01
solar_exposure	-0.26	0.06	-0.23	0.06	-0.04	0.0	-0.04	0.38	0.6	1.0	-0.12	-0.1	0.05
max_temperature	-0.07	0.17	-0.07	0.17	0.0	-0.03	-0.0	0.71	1.0	0.6	-0.16	-0.09	0.04
min_temperature	-0.15	0.07	-0.15	0.07	0.01	-0.08	0.01	1.0	0.71	0.38	-0.0	-0.08	0.07
frac_at_neg_RRP	-0.19	-0.08	-0.42	-0.06	1.0	-0.26	1.0	0.01	-0.0	-0.04	0.04	-0.1	0.03
RRP_negative	0.06	0.04	0.12	0.03	-0.27	1.0	-0.26	-0.08	-0.03	0.0	-0.02	0.01	-0.0
demand_neg_RRP	-0.18	-0.08	-0.41	-0.06	1.0	-0.27	1.0	0.01	0.0	-0.04	0.04	-0.09	0.02
RRP_positive	0.22	1.0	0.22	1.0	-0.06	0.03	-0.06	0.07	0.17	0.06	-0.03	-0.01	-0.03
demand_pos_RRP	0.97	0.22	1.0	0.22	-0.41	0.12	-0.42	-0.15	-0.07	-0.23	-0.07	0.14	-0.24
RRP	0.22	1.0	0.22	1.0	-0.08	0.04	-0.08	0.07	0.17	0.06	-0.03	-0.0	-0.03
demand	1.0	0.22	0.97	0.22	-0.18	0.06	-0.19	-0.15	-0.07	-0.26	-0.06	0.12	-0.25

Correlated variables:

solar_exposure $\longrightarrow$ max_temperature (+0.60)

min_temperature $\longrightarrow$ max_temperature (+0.71)

demand_neg_RRP $\longrightarrow$ frac_at_neg_RRP (+1.00)

RRP $\longrightarrow$ RRP_positive (+0.97)

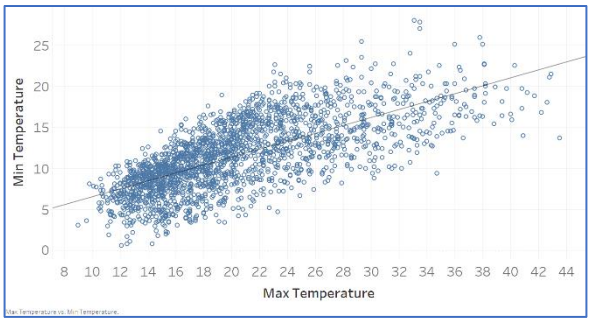


Figure 1 Positive Correlation between Max Temperature and Min Temperature

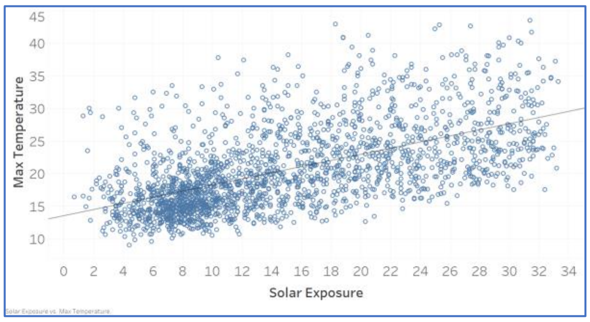


Figure 2 Positive Correlation between Max Temperature and Solar Exposure

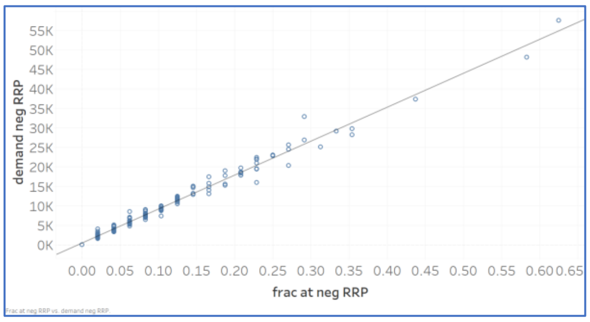


Figure 3 Positive Correlation between demand neg RRP and frac at neg RRP

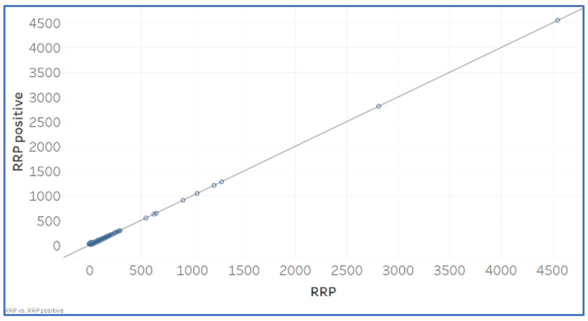


Figure 4 Positive Correlation between RRP and RRP Positive

Columns to be dropped because of correlation:

- Max_temperature
- frac_at_neg_RRP
- RRP_positive

Step 3: Diving Dataset into feature set and labels

The dataset needs to be split up so that the target variable is in 'labels' and the feature set contains only required variables i.e. excluding the target variable and the correlated variables.

Use the 'drop' method to remove the target column and correlated variables from independent variables and make it a separate series as shown below:

```
# Dividing dataset into label and feature sets
X = dataset.drop(['demand','max_temperature','frac_at_neg_RRP','RRP_pos
itive'], axis = 1) # Features
Y = dataset['demand'] # Labels
print(type(X))
print(type(Y))
print(X.shape)
print(Y.shape)
```

Step 4: Normalizing the numerical features

In order to prevent the algorithm from favoring a column with greater values over a column with lower values, the data frame must now be changed so that all columns have a mean of 0 and a variance of 1.

Use the StandardScaler() method from the sklearn package and pass the dataset named 'X' from the last step as a variable:

```
# Normalizing numerical features so that each feature has mean 0 and va
riance 1
feature_scaler = StandardScaler()
X_scaled = feature_scaler.fit_transform(X)
```

Implementation of Linear Regression

Linear regression is a fundamental and widely used kind of predictive analysis. The objective of regression analysis is to investigate two factors:

1. How well can a collection of predictor factors predict an outcome (dependent) variable?
2. Which factors, in particular, are significant predictors of the outcome variable, and how do they influence the outcome variable, as shown by the size and sign of the beta estimates?

These estimations of regression are used to describe the connection between a dependent variable and one or more independent variables. With one dependent and one independent variable, the simplest version of the regression equation is y = c + b*x, where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable.

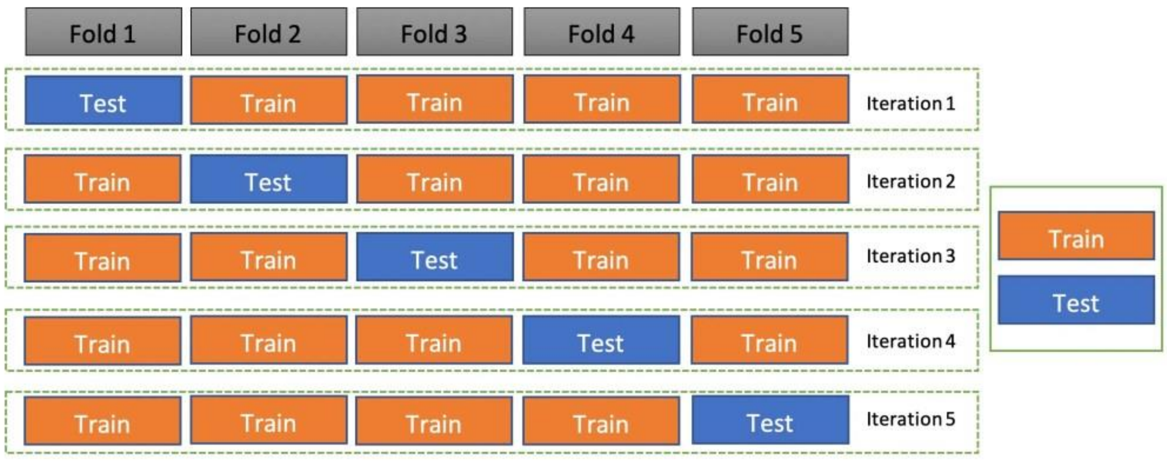
In our assignment, we are taking the dataset of energy demand where the demand for energy needs to be predicted. So, our target variable would be demand, and other variables are remained as in the dependent variable.

Through Linear regression, we will check the most significant Variables and how they are impacting the demand for electricity.

K-fold Cross-validation:

Also, we are going to use 'K-fold Cross-Validation' with every model to divide our dataset into training and test data where K is the number of groups (or folds) which the data is divided into. The procedure is as follows:

1. Shuffle the dataset randomly.
2. Divide the dataset into 'K' equal number of folds.
3. For each unique fold (or group):
   3.1. Take a group as our test dataset and the remaining groups as the training data.
   3.2. Fit a model on the training dataset and evaluate it on the test data.
   3.3. Retain the evaluation score on the test data set and discard the model.
4. Summarize the skill of the model by taking mean and std. deviation of the test performances.


Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Test Train Train Train Train Iteration 1
Train Test Train Train Train Iteration 2
Train Train Test Train Train Iteration 3
Train Train Train Test Train Iteration 4
Train Train Train Train Test Iteration 5
Train
Test

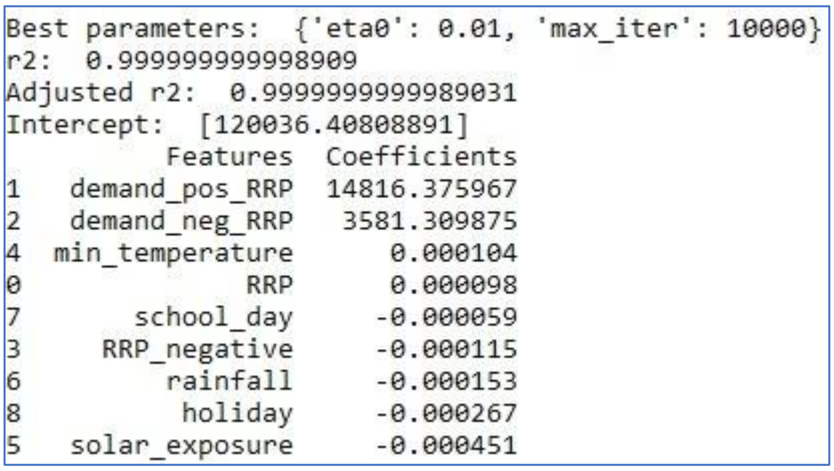
<u>**'K-Fold Cross Validation' is an evaluation technique which avoids overfitting. As we have 2102 instances, 5 folds are probably enough.**</u>

Linear Regression without Regularization:

The first step is to train the model without regularization. Grid search is used to tune the parameters of the model and check which parameters(best_parameters) are significantly impacting the demand for electricity.

Without regularization, we will first tune the hyperparameters as it will show the values of beta's (coefficients) it will not recognize the best parameters.

```
# Linear Regression without Regularization
# Tuning the SGDRegressor parameters 'eta0' (learning rate) and 'max_it
er' using Grid Search
sgdr = SGDRegressor(random_state = 1, penalty = None)
grid_param = {'eta0': [.0001, .001, .01, .1, 1], 'max_iter':[10000, 200
00, 30000, 40000]}

gd_sr = GridSearchCV(estimator=sgdr, param_grid=grid_param, scoring='r2
', cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print("Best parameters: ", best_parameters)

best_result = gd_sr.best_score_ # Mean cross-
validated score of the best_estimator
print("r2: ", best_result)

Adj_r2 = 1-(1-best_result)*(1682-1)/(1682-9-1)
print("Adjusted r2: ", Adj_r2)

'''
Adj_r2 = 1-(1-r2)*(n-1)/(n-p-1)

#where, n = number of observations in training data, p = number of feat
ures
#'''

best_model = gd_sr.best_estimator_
print("Intercept: ", best_model.intercept_)

print(pd.DataFrame(zip(X.columns, best_model.coef_), columns=['Features
','Coefficients']).sort_values(by=['Coefficients'],ascending=False))
```
Result:



```
Best parameters:  {'eta0': 0.01, 'max_iter': 10000}
r2:  0.999999999998909
Adjusted r2:  0.9999999999989031
Intercept:  [120036.40808891]
           Features  Coefficients
1    demand_pos_RRP  14816.375967
2    demand_neg_RRP   3581.309875
4   min_temperature      0.000104
0               RRP      0.000098
7        school_day     -0.000059
3      RRP_negative     -0.000115
6          rainfall     -0.000153
8           holiday     -0.000267
5    solar_exposure     -0.000451
```

In our first scenario i.e., without regularization, the performance of the model is giving the r2 score of 0.99 which signifies the goodness of fit of the model, hence all the independent variables are explaining the dependent variable. But there could be the problem of overfitting and we cannot ignore that.

Linear Regression with Regularization:

To take care of the overfitting, the model will be trained with the regularization method where L1, L2, and elastic net are being used to identify the most important variables for the model as fewer variables help increase the interpretability.

```
# Linear Regression with Regularization
# Tuning the SGDRegressor parameters 'eta0' (learning rate) and 'max_it
er', along with the regularization parameter alpha using Grid Search
sgdr = SGDRegressor(random_state = 1, penalty = 'elasticnet')
grid_param = {'eta0': [.0001, .001, .01, .1, 1], 'max_iter':[1000, 5000
, 10000, 20000, 30000, 40000],'alpha': [.001, .01, .1, 1,10, 100], 'l1_
ratio': [0,0.25,0.5,0.75,1]}

gd_sr = GridSearchCV(estimator=sgdr, param_grid=grid_param, scoring='r2
', cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print("Best parameters: ", best_parameters)

best_result = gd_sr.best_score_ # Mean cross-
validated score of the best_estimator
print("r2: ", best_result)

Adj_r2 = 1-(1-best_result)*(1682-1)/(1682-9-1)
print("Adjusted r2: ", Adj_r2)

'''
Adj_r2 = 1-(1-r2)*(n-1)/(n-p-1)

where, n = number of observations in training data, p = number of featu
res
'''

best_model = gd_sr.best_estimator_
print("Intercept: ", best_model.intercept_)

print(pd.DataFrame(zip(X.columns, best_model.coef_), columns=['Features
','Coefficients']).sort_values(by=['Coefficients'],ascending=False))
```

L1 regularization may solve the multicollinearity issue by restricting the coefficient norm and pinning some coefficient values to 0. On the other hand, **L2 Regularization** may solve the multicollinearity issue by restricting the norm of coefficients and retaining all variables. Estimating a coefficient to be precisely 0 is uncommon. This is not a disadvantage unless a sparse coefficient vector is essential for some reason. Whereas, the elastic net approach combines L1 and L2 regularization in a weighted manner.

L1 Regularization adds the absolute value of the magnitude of the coefficient as a penalty term to the loss function.

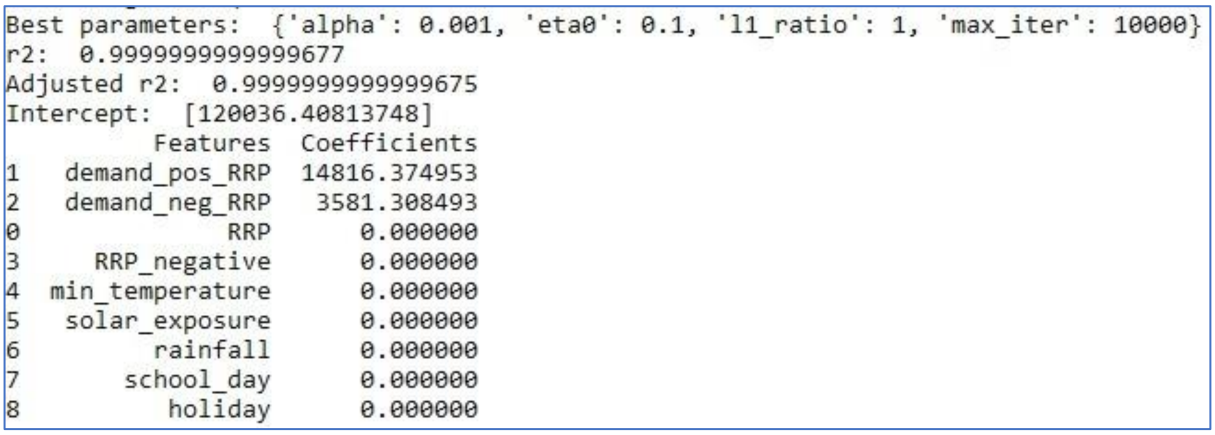
$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

L2 Regularization adds a square magnitude of coefficient as a penalty term to the loss function.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

The main distinction between both strategies is that L1 reduces the coefficient of the less significant characteristic to zero, eliminating certain features entirely.

Result:



```
Best parameters:  {'alpha': 0.001, 'eta0': 0.1, 'l1_ratio': 1, 'max_iter': 10000}
r2:  0.9999999999999677
Adjusted r2:  0.9999999999999675
Intercept:  [120036.40813748]
          Features  Coefficients
1    demand_pos_RRP  14816.374953
2    demand_neg_RRP   3581.308493
0               RRP      0.000000
3      RRP_negative      0.000000
4   min_temperature      0.000000
5    solar_exposure      0.000000
6          rainfall      0.000000
7        school_day      0.000000
8           holiday      0.000000
```

With the Regularization approach, We are using the elastic net model i.e., the combination of L1 and L2 then the performance of the model is showing the same as without regularization but it helps us to identify the most relevant features which are explaining the dependent variable most.

Thus, demand_pos_RRP and demand_neg_RRP are the most influential variables whose coefficients have values, while regularizations render the coefficients of other variables null.

Impact of regularization on linear Regression coefficients, performance, and Interpretability

Linear Regression coefficients:

As we can see in the result screenshot regularizations impact the values of coefficients and help us in identify the most relevant variables in the model. The most relevant variables are demand_pos_RRP and demand_Neg_RRP as their coefficients have good positive values and have a significant impact on the demand for electricity. These two variables are alone able to explain the demand for electricity as the value of coefficients are (demand_pos_RRP =14816.3749, demand_neg_RRP=3581.30).

With Regularization making other variables zero it means they all are irrelevant to the model and not explaining the demand for electricity that much. Some of the variables where the value of the coefficient is zero are RRP, RRP_negative, n_temperature,solar_exposure, rainfall, school_day, and holiday.

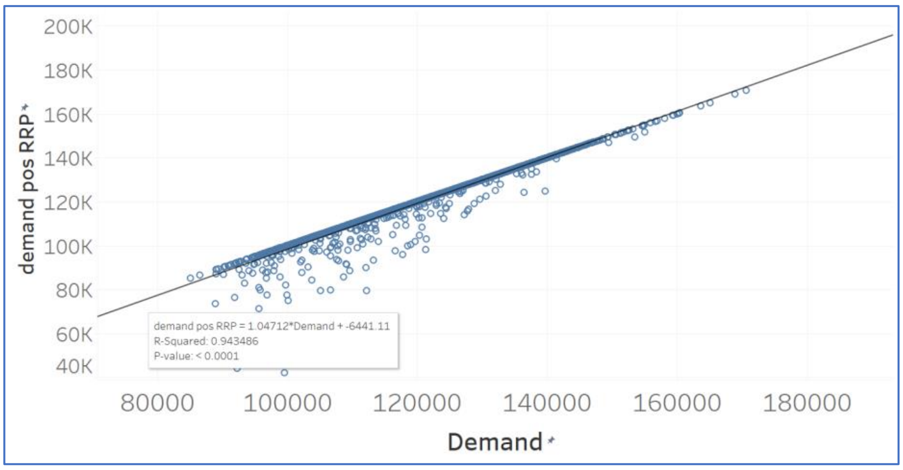


Figure 5 Impact of demand pos RRP on Demand

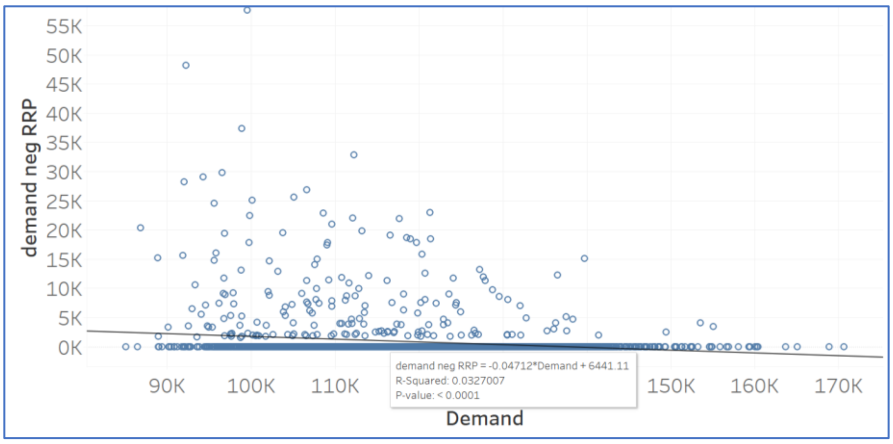


Figure 6 Impact of demand neg RRP on Demand

Performance:

As far as performance is concerned there is a minute increase in performance after the implementation of the regularization which can be negligible. R2 score shows the goodness of fit of the model so from the performance we can conclude that there is no difference in the performance of the model before and after the regularization but after regularization, the problem of overfitting could be minimized.

Interpretability:

Without regularization, we cannot identify the most relevant features in the model it will only tell the r2 score or goodness fit which will only interpret whether independent variables are explaining the Demand for electricity in Victoria, Australia.

We can easily interpret from the regularization model that Fewer variables such as demand_pos_RRP and demand_neg_RRP are the variables that need to be more considered than other variables.

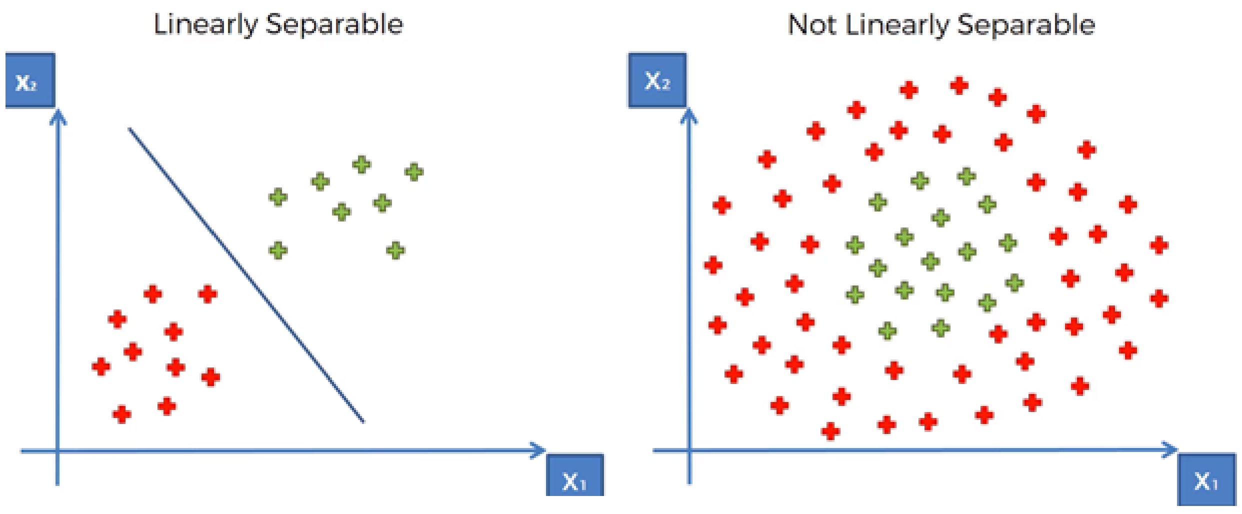
We can say that a single unit increase in the demand_pos_RRP will directly increase the total demand for electricity by 14816.37 units. On the other hand, the single unit increase in the demand_neg_RRp will directly increase the demand for electricity by 3581.30 units.

Implementation of Support Vector Regression

In Support Vector Regression, the needed straight line to fix the data is known as the hyperplane. The purpose of a support vector machine method is to classify data points using a hyperplane in an n-dimensional space. Support Vectors refer to the data points on each side of the hyperplane that is closest to the hyperplane. These variables affect the location and direction of the hyperplane and contribute to the formation of the SVM.

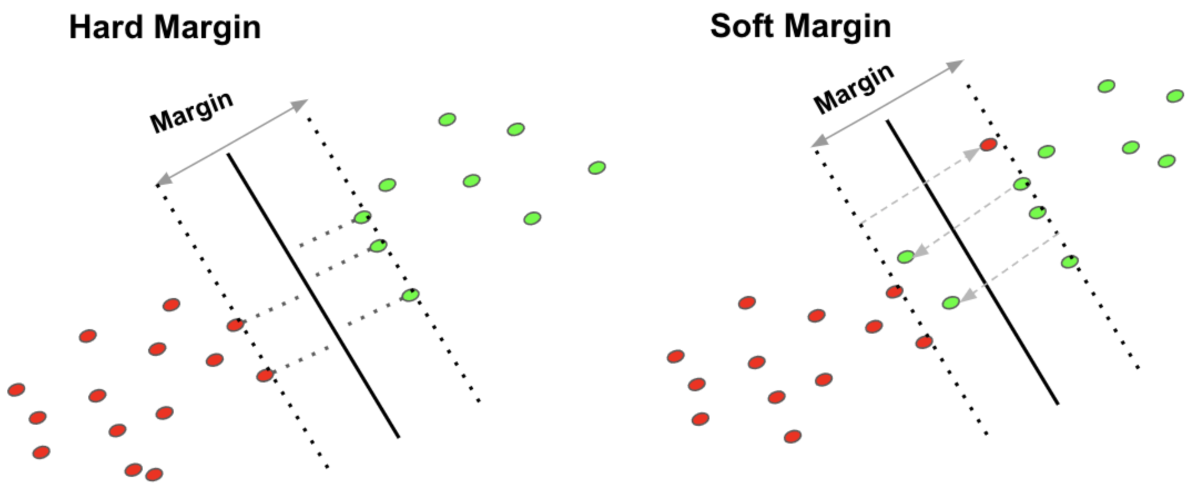
Kernel:

In cases where the two classes are non-linearly separated in consideration of a two-dimensional space, a kernel can be used to transform a two-dimensional space into a higher dimension by creating as many additional variables as needed in order to achieve a position where the groups can now be linearly separable.



L2 Regularization Parameter ‘C’ (to avoid Overfitting):

Regularization is a technique used to calibrate our machine-learning models in order to avoid underfitting and overfitting. In the case of SVC, we use ‘C’ as a regularization parameter whose value can be controlled in order to allow some data points to end up within or on the other side of the margin which is generally called ‘Slack Data Points’.



Higher the value of ‘C’, the smaller the no. of allowed slack data points (Hard Margin SVM) and on the other hand the smaller the value of ‘C’, the higher number of slack data points are allowed (Soft Margin SVM). Hence, the value of ‘C’ will control the slack data points and it’s a hyperparameter that can be controlled.

Impact of L2 regularization with Support vector regressor

Regularization is accomplished in support vector regression using L2 (alpha C) and the epsilon sensitivity.

- When epsilon has a small value, the model is resistant to outliers.
- When the value of epsilon is large, outliers will be considered.
- When the epsilon is sufficiently large, the penalization term is applied to minimize the norm of the coefficients.

We have utilized Regularization in SVR to modify our model, and the model's performance is fairly satisfactory, with an R2 score of 0.9999 and alpha(C=10000) and epsilon values of 100.

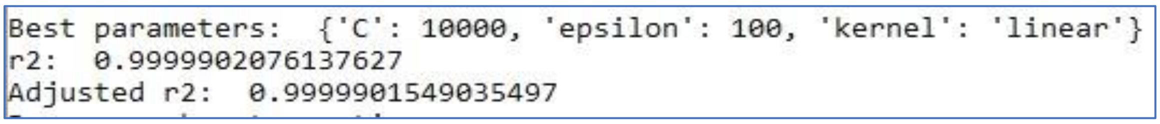
Thus, in our example, both values are relatively high, which increases the number of outliers and maximizes the coefficient norm.

SVR can be used to display the effect of alpha (or C, which is 1/alpha). When C is small, regularization is robust; hence, the slope will be modest. However, when the value of C is relatively high, weak regularization results and the model is unable to make accurate predictions.

Through performance and interpretability, we can determine the effect of Support vector regression with regularization. How much the model is interpretable and how well it performs.

```
# Implementing Support Vector Regression
# Tuning the SVR parameters 'kernel', 'C', 'epsilon' and implementing c
ross-validation using Grid Search
svr = SVR()
grid_param = {'kernel': ['linear', 'poly', 'rbf', 'sigmoid'], 'C': [100
,1000,10000,100000], 'epsilon': [100,1000,10000]}

gd_sr = GridSearchCV(estimator=svr, param_grid=grid_param, scoring='r2'
, cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print("Best parameters: ", best_parameters)

best_result = gd_sr.best_score_ # Mean cross-
validated score of the best_estimator
print("r2: ", best_result)

Adj_r2 = 1-(1-best_result)*(1682-1)/(1682-9-1)
print("Adjusted r2: ", Adj_r2)
```

Result:



```
Best parameters:  {'C': 10000, 'epsilon': 100, 'kernel': 'linear'}
r2:  0.9999902076137627
Adjusted r2:  0.9999901549035497
```

Performance:

With an r2 score of 0.99, the model's performance is excellent; nevertheless, C and epsilon values are rather high, hence we cannot advocate using this model for prediction. As SVR is more resistant to outliers, yet when the value of epsilon is large, the model will adjust for outliers, just as our model does. As a result, we penalize the model with C in order to minimize the norm of coefficients, but the value of C is rather high, indicating that it has a very narrow margin of error. Therefore, regardless of the model's performance, this model cannot be used for prediction. The second argument for not choosing the SVR model in this instance is that the epsilon-insensitive tube allows us to disregard data points with tiny mistakes. Less data equals less information, therefore from the perspective of model optimization, it is not particularly valuable. However, from a computational standpoint, it can accelerate model training. In the end, only a few data points are utilized to define the model, and these are referred to as support vectors.

Interpretability:

In the case of interpretability, we are unable to interpret the most critical or relevant characteristics of the model that explain our demand for electricity. We are unable to do so, therefore its interpretability is quite low; but, if we are able to reduce the values of c and epsilon, the model will be able to anticipate the y (electricity demand) in a very sustainable manner.

Implementation of Random Forest Regression

In a random forest model, the algorithm splits data on the variable that results in the maximum reduction in entropy (results in the purest child nodes). In a decision tree regressor, the algorithm splits data on the variable that results in the maximum reduction of the fitting error of a given node.

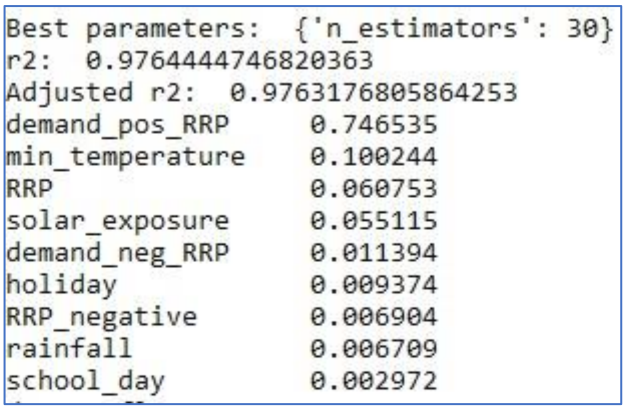
We are tuning the **Hyparameters** (n_estimators) which will help us to increase the capability of the model to take the correct decision as more decision trees leads to more tuned results.

Random Forest Model without Significant Features

Implement the GridSearch method with 5 folds and n_estimators. It will create multiple models with a different number of decision trees and Cross-Validate each one to give us the best score and the optimal number of decision trees.

```
# Implementing Random Forest Regression
# Tuning the random forest parameter 'n_estimators' and implementing cr
oss-validation using Grid Search
rfr = RandomForestRegressor(criterion='squared_error', max_features='sq
rt', random_state=1)
grid_param = {'n_estimators': [10,20,30,40,50,100]}

gd_sr = GridSearchCV(estimator=rfr, param_grid=grid_param, scoring='r2'
, cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print("Best parameters: ", best_parameters)

best_result = gd_sr.best_score_ # Mean cross-
validated score of the best_estimator
print("r2: ", best_result)

Adj_r2 = 1-(1-best_result)*(1682-1)/(1682-9-1)
print("Adjusted r2: ", Adj_r2)

'''
Adj_r2 = 1-(1-r2)*(n-1)/(n-p-1)

where, n = number of observations in training data, p = number of featu
res
'''

featimp = pd.Series(gd_sr.best_estimator_.feature_importances_, index=l
ist(X)).sort_values(ascending=False) # Getting feature importances list
 for the best model
print(featimp)
```

Result:



```
Best parameters:  {'n_estimators': 30}
r2:  0.9764444746820363
Adjusted r2:  0.9763176805864253
demand_pos_RRP      0.746535
min_temperature     0.100244
RRP                 0.060753
solar_exposure      0.055115
demand_neg_RRP      0.011394
holiday             0.009374
RRP_negative        0.006904
rainfall            0.006709
school_day          0.002972
```

Significant features:

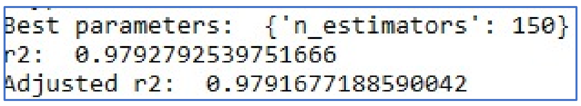
Selecting the significant features and putting them into a new variable "X_scaled_"

```
# Selecting features with higher sifnificance and redefining feature se
t
X_ = dataset[['demand_pos_RRP','RRP','solar_exposure','min_temperature'
,'demand_neg_RRP']]

feature_scaler = StandardScaler()
X_scaled_ = feature_scaler.fit_transform(X_)
```

Random Forest Model with Significant Features

```
# Tuning the random forest parameter 'n_estimators' and implementing cr
oss-validation using Grid Search
rfr = RandomForestRegressor(criterion='squared_error', max_features='sq
rt', random_state=1)
grid_param = {'n_estimators': [50,100,150,200,250]}

gd_sr = GridSearchCV(estimator=rfr, param_grid=grid_param, scoring='r2'
, cv=5)

gd_sr.fit(X_scaled_, Y)

best_parameters = gd_sr.best_params_
print("Best parameters: ", best_parameters)

best_result = gd_sr.best_score_ # Mean cross-
validated score of the best_estimator
print("r2: ", best_result)

Adj_r2 = 1-(1-best_result)*(1682-1)/(1682-9-1)
print("Adjusted r2: ", Adj_r2)
```

Result:


```
Best parameters:  {'n_estimators': 150}
r2:  0.9792792539751666
Adjusted r2:  0.9791677188590042
```

Performance of the RFR as compared to LR with Regularization and SVR with Regularization:

The comparative performance of the three models on the dataset of energy demand is quite good (R2 score: LR=99.999, SVR=99.99, RFR=97.92). But overall, the performance of the linear regression model is quite good as two variables are alone able to explain the dependent variable. Whereas in the case of SVR and RFR we need to increase the performance of the model. Although, RFR is also giving a good performance of 97.92 with feature importance and without importance feature R2 score decrease by .26(97.64).

Interpretability:

As far as interpretability is concerned, interpretation using LR with regularization is easier as compared to other models as LR with regularization helps to identify the most relevant variables whose coefficients are more effective than other variables. On the other hand, SVR and RFR are not able to deduce the variables with the same technique as RFR in we will able to identify the most significant features but not clearly defines the values of coefficients as LR do. LR will help in reducing the values of coefficients irrelevant to the model to zero and improves the values of relevant features.

We only have two variables in the case of LR with regularization we can easily explain the value of y i.e. Demand of energy on the other hand through SVR and RFR interpretability is a bit difficult as we cannot reduce the important features as Lr with regularization can do.

So, we can say that LR with regularization had more capability to interpret the model as compared to the other two models.

Conclusion:

From the above discussion of three different models, namely linear regression, random forest, and support vector regressor, we can infer that all of the models perform well, and that regularization has a favorable effect on their performance and makes them easier to read. In the actual world, however, we would prefer to perform Linear regression with regularization because it would help us find the most important model characteristics so that we may use them in subsequent analyses. Feature selection aids in effectively resolving the business issue. On the other hand, the other two models, Random Forest and SVR have good performance, but they are not able to perform as LR with regularization is doing. Although Random Forest helps in removing the problem of overfitting, our data set is quite similar to the multiple regression problem, so we should choose LR with regularization in this case. As far as SVR with regularization is concerned, C and epsilon have high values, so implementing SVR with regularization would be problematic in predicting the demand for electricity because performance is high with a hard margin.

References

Nagpal, A. (2017, Oct 13). *L1 and L2 Regularization Methods*. Retrieved from Towards Data Science: https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

Nickolas, S. (2021, May 31). *What Do Correlation Coefficients Positive, Negative, and Zero Mean?* Retrieved from Investopedia: https://www.investopedia.com/ask/answers/032515/what-does-it-mean-if-correlation-coefficient-positive-negative-or-zero.asp

Shi, A. (2022, Dec 1). *Understanding SVR and Epsilon Insensitive Loss with Scikit-learn*. Retrieved from Towards Data Science: https://towardsdatascience.com/understanding-svr-and-epsilon-insensitive-loss-with-scikit-learn-28ec03a3d0d9

Team, I. E. (2022, August 25). *Positive and Negative Correlations (Definitions and Examples)*. Retrieved from Indeed: https://www.indeed.com/career-advice/career-development/positive-and-negative-correlations#:~:text=A%20positive%20correlation%20exists%20when,variable%20rises%2C%20the%20other%20falls.