Measuring Regression Performance

Evaluation Metrics for performance (RSS, R-Squared, RMSE, MAE etc)

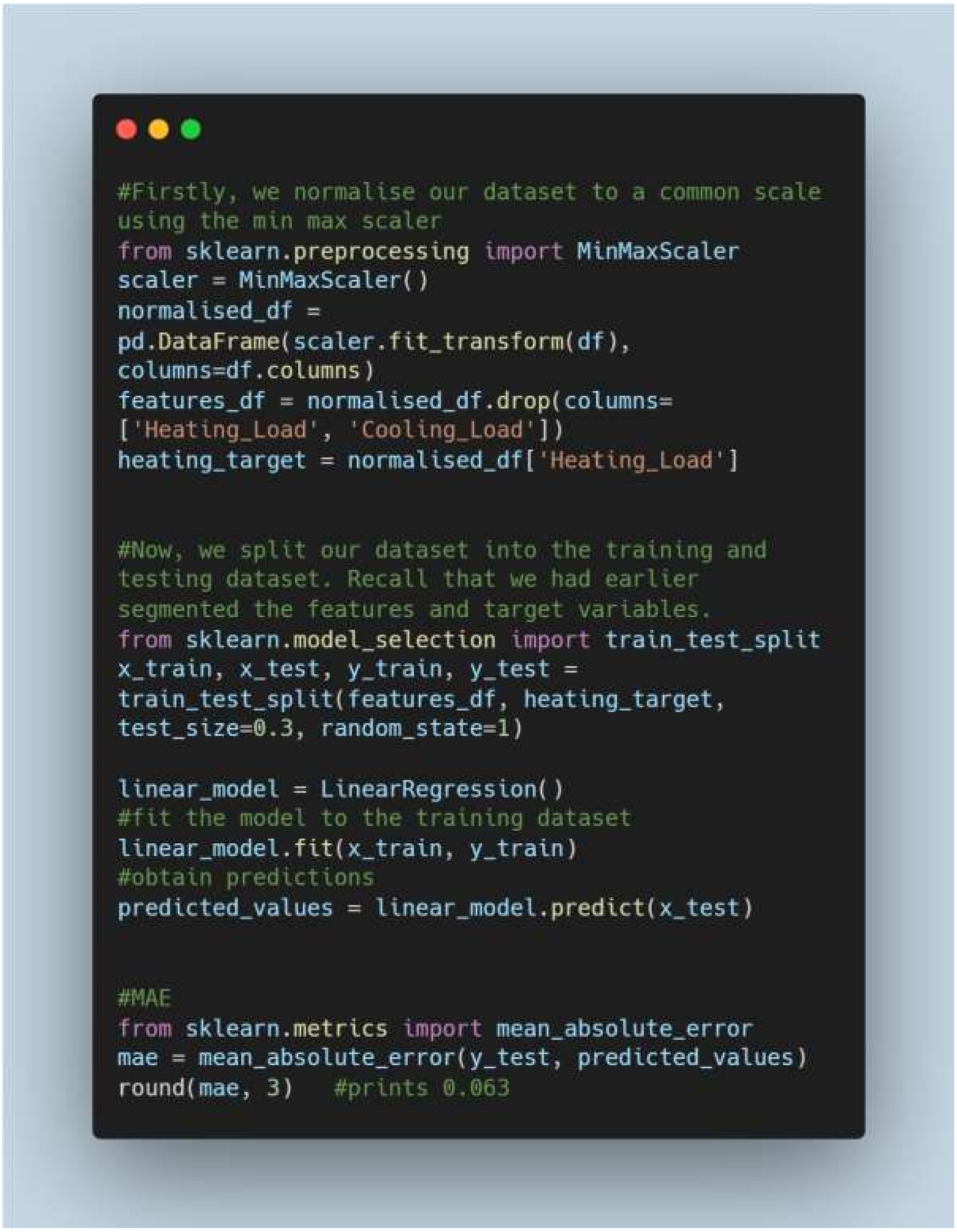
How well a regression model performs can be obtained by how close the predicted value is to the ground truth. It is very important to use the appropriate metric to evaluate the performance. In this section, we discuss some examples of metrics used in evaluating regression models such as RSS, R-Squared, RMSE and MAE

Mean Absolute Error (MAE)

MAE is easy and intuitive such that it calculates the sum of the average of the absolute error between the predicted values and the true values. Since the absolute difference is taken, this metric does not consider direction. However, because the absolute difference is obtained, it is unable to give information about the model overshooting or undershooting. The smaller the MAE is, the better the model. Therefore, if the MAE is 0, the model is perfect and accurately predicts results which is almost impossible. The mean absolute error is more robust to outliers

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

MAE	= mean absolute error
y_i	= prediction
x_i	= true value
n	= total number of data points



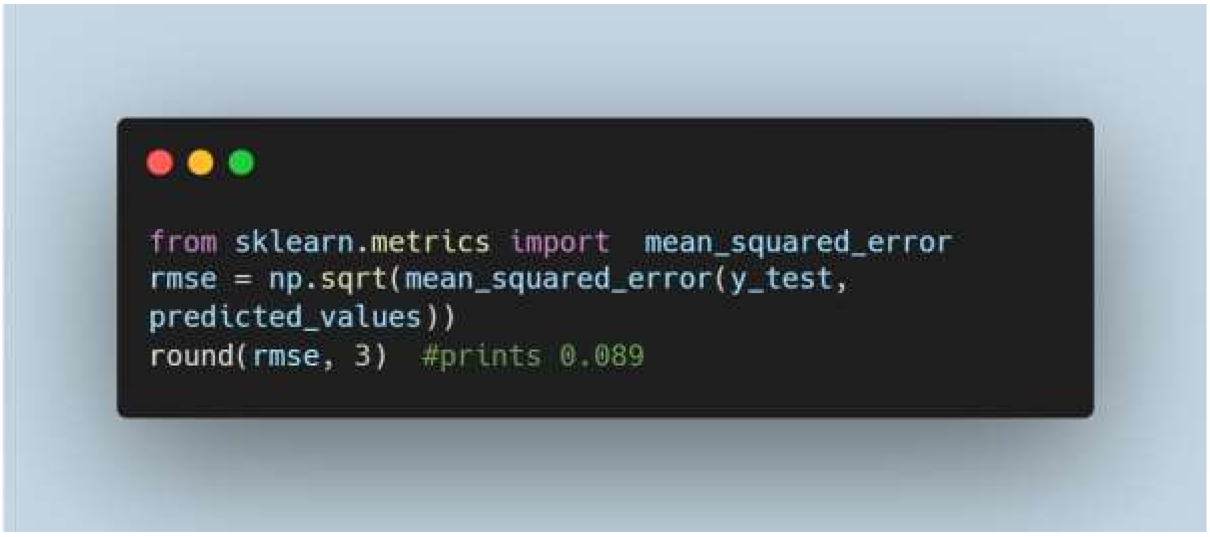
```
#Firstly, we normalise our dataset to a common scale
using the min max scaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
normalised_df =
pd.DataFrame(scaler.fit_transform(df),
columns=df.columns)
features_df = normalised_df.drop(columns=
['Heating_Load', 'Cooling_Load'])
heating_target = normalised_df['Heating_Load']


#Now, we split our dataset into the training and
testing dataset. Recall that we had earlier
segmented the features and target variables.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(features_df, heating_target,
test_size=0.3, random_state=1)

linear_model = LinearRegression()
#fit the model to the training dataset
linear_model.fit(x_train, y_train)
#obtain predictions
predicted_values = linear_model.predict(x_test)


#MAE
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_test, predicted_values)
round(mae, 3)   #prints 0.063
```

Residual Sum of Squares (RSS)

Also known as the sum of squared residuals (SSR), this metric explains the variance in the representation of the dataset by the model; it measures how well the model approximates the data. A residual is the estimated error made by a model. In simpler terms, it is the difference between the nth true value and the nth predicted value by the model. RSS is the sum of the square of errors between the residuals in a model. The lower the RSS, the better the model's estimations and vice versa.

$$RSS = \sum_{i=1}^{n}(y_i - f(x_i))^2$$

RSS	= residual sum of squares
y_i	= i^th value of the variable to be predicted
$f(x_i)$	= predicted value of y_i
n	= upper limit of summation



```
from sklearn.metrics import  mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test,
predicted_values))
round(rmse, 3)  #prints 0.089
```

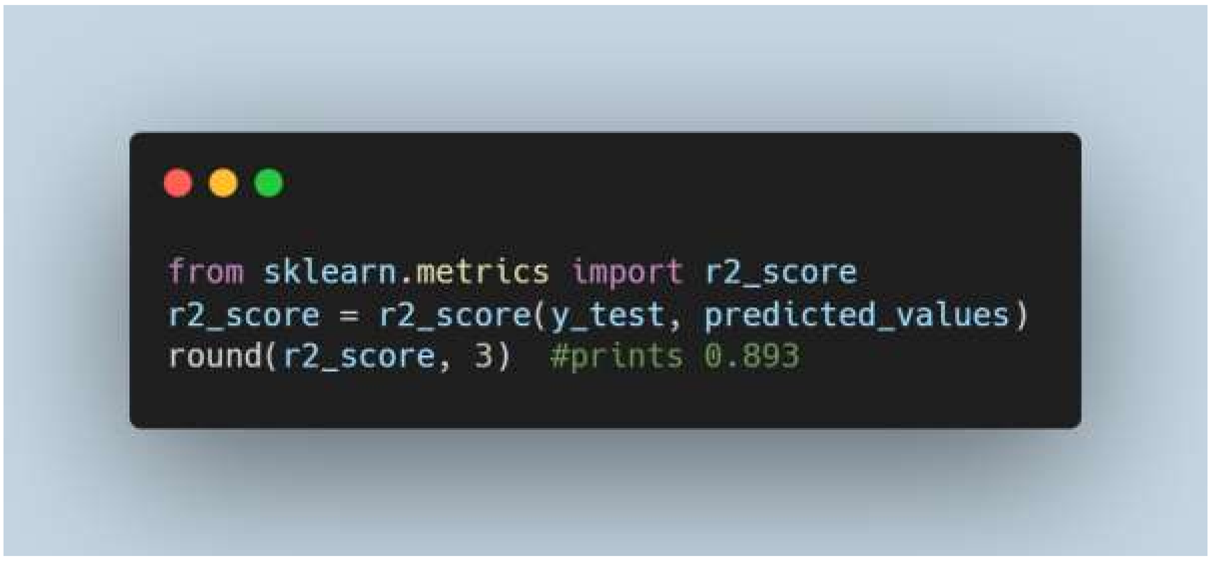
R-Squared

Also known as the coefficient of determination, r-squared is a metric used in regression to determine the goodness of fit of the model. With values ranging from 0 to 1, It gives information on the percentage of the response variable explained by the model. Mostly, the higher the value, the better the model however,

$$R^2 = 1 - \frac{RSS}{TSS}$$

R^2	= coefficient of determination
RSS	= sum of squares of residuals
TSS	= total sum of squares

this is not necessarily always true.



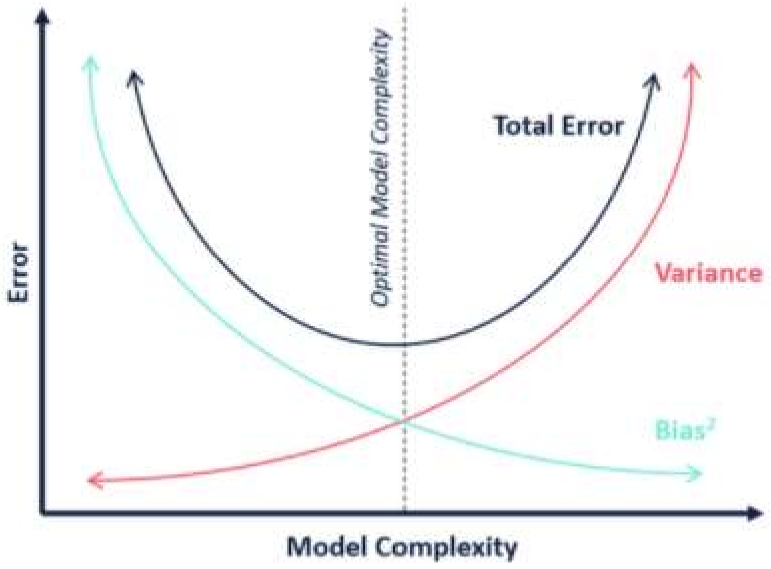
```python
from sklearn.metrics import r2_score
r2_score = r2_score(y_test, predicted_values)
round(r2_score, 3)  #prints 0.893
```

Model complexity, Underfitting and Overfitting

Model complexity refers to the number of input features used to train a model and the algorithmic learning complexity. An overly complex model can be difficult to interpret, prone to overfitting and also require more computing. When creating models, it is imperative for the model to generalise well enough to make reasonable predictions on new and unseen data. An overfit model will perform well on the training data and poorly on unseen data. While a model is required to learn the actual relationship of the variables in the training set, an overfit model memorises the training set, fits the noise, outliers and irrelevant information, then makes predictions based on this noise which is incorrect. On the other hand, when a model is too simple, it can be as a result of having very few features not sufficient enough to learn details and relationships in the data. In a later section, we will discuss methods that can be used to achieve optimal and acceptable model complexities while avoiding overfitting and underfitting.

The Bias-Variance tradeoff

Bias and variance are common occurrences in machine learning and there is a constant struggle to achieve low bias and variance. Bias is a measure of correctness of a model i.e. how far off is a model from being correct? While high bias results in an increase in the error by making assumptions which prevent the model from capturing relevant relationships between the predictors and response variable, low bias gives lower error and also prevents underfitting by capturing important relationships. On the other hand, variance tells how much the values estimated by a model will vary across different training data. When the variance is low, it means that there is only a small change in the estimate of the model with new training data. A high variance causes overfitting such that the changes in estimates obtained with new training data is large because the model is so complex that it has now learnt patterns from one training data such that it cannot generalise to other training sets. While it is essential to obtain low bias and low variance, it is almost impossible to achieve this simultaneously which is where the 'bias-variance tradeoff' occurs.


Optimal Model Complexity
Total Error
Variance
Bias^2
Error
Model Complexity