Imperial College London

Department of Mathematics

# Machine Learning Algorithms

*Author:*
Shane Mullins (CID: )

Date: March 22, 2022

## Exploratory Data Analysis

Initial exploratory data analysis (EDA), yields the following insights from the dataset:

- 500 observations, 39 features, 1 target variable.
- no NA or other missing values.
- All variables are floating numbers.
- Target variable statistics: mean[1] $\approx 0.0563$, standard deviation $\approx 0.1397$, min $\approx -0.4700$ and max $\approx 0.4195$.
- The unnormalized data is not sparse as there are no zero values present.

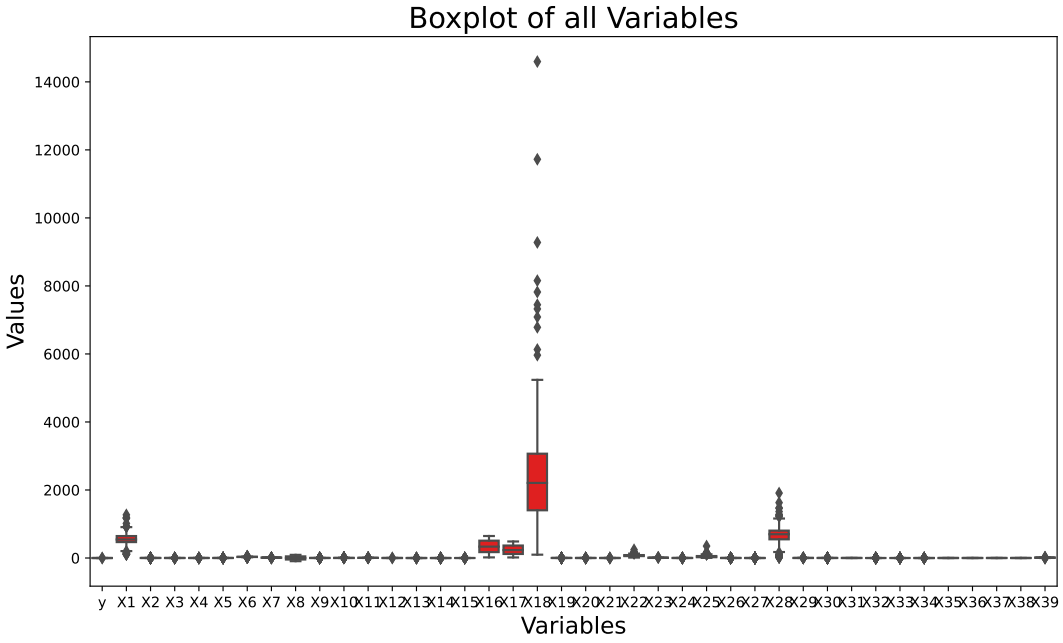


Figure 1: Boxplot of all variables shows one particular feature is on a significantly larger scale than the rest.

[1] All numerical results are given to four significant figures.

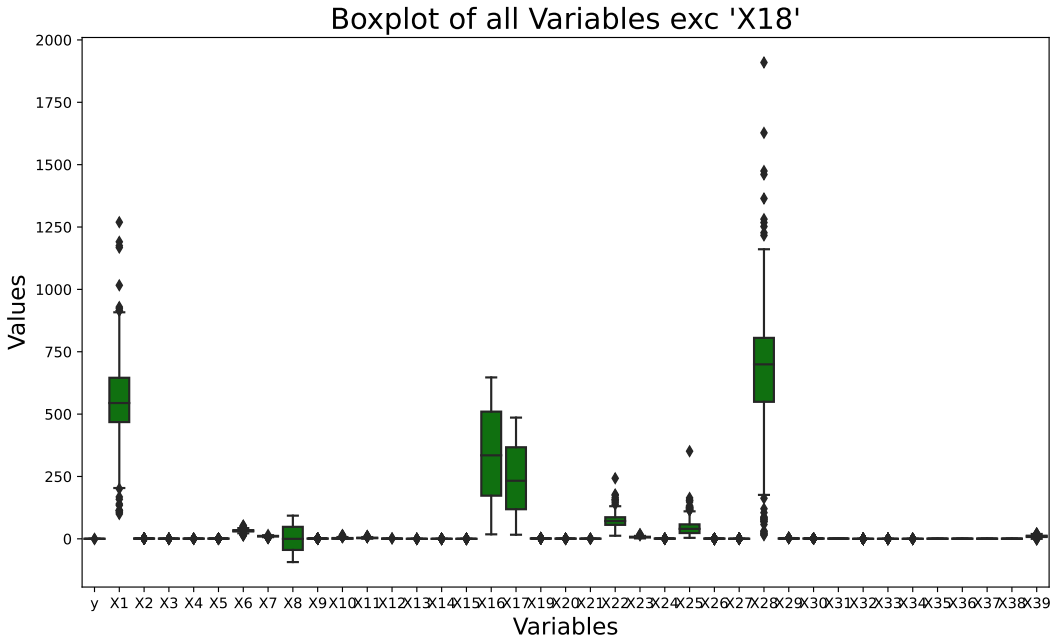


Figure 2: Boxplot of all variables excluding the largest feature shows that the majority of features have a comparatively small range. A few features such as 'X28' and 'X1' have noticeably large scales relative to the others. Many potential outliers could influence the results in the following sections.

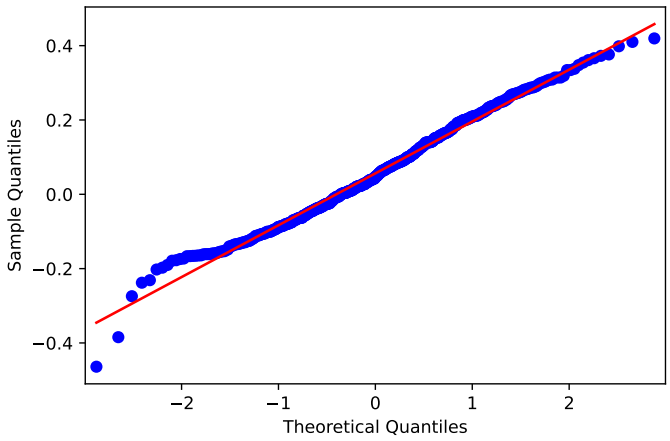


Figure 3: QQ plot of target variable shows reasonable closeness to a normal distribution perhaps due to a previous log transform. There is a slight deviation from the norm at the tails.

Figures 1 and 2 visually highlight some important feature characteristics. Investigating the relationship between variables, having calculated a correlation matrix (code in Appendix B), we see that **there are 14 cases where the correlation between two features is greater than 0.7**. The pairplot in Figure 4 visually highlights this point. This will be important to keep in mind when creating regression models due to the assumption of feature independence. The results so far strongly suggest the existence of multicollinearity in the dataset. This can lead to imprecise parameter estimation (Lau, 2020).

These graphs and statistics give some initial insight into the dataset, however far more EDA could have been carried out given the need. This highlights and important point: the analysis in this

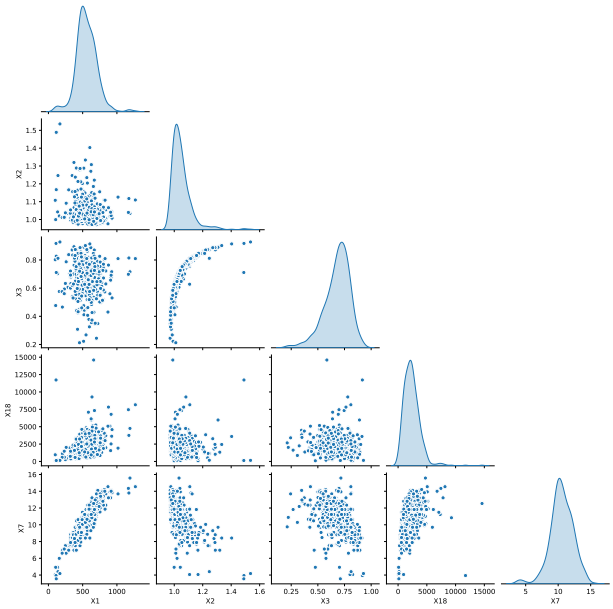


Figure 4: There exists an evident relationship between a chosen subset of features, "X1","X2","X3","X6" and"X7" in this case.

report is driven by illustrating important processes in machine learning rather than achieving the best possible results. As such, there a many alternative and more extensive routes one could take in model selection for example, however the aim here is to implement prevalent machine leaning techniques and to demonstrate a deep understanding of the theory that underpins these powerful tools.

## Hierarchical Clustering

As distinct from k-means, hierarchical clustering does not require a prespecified number of clusters. A **bottom-up (agglomerative) approach**, as opposed to a divisive approach, is taken here. This approach was deemed to be more interpretable and boasts various performance advantages (Murtagh & Contreras, 2012). The main hyperparameters for this method are laid out in Table 1.

| Hyperparameter | Possible Values |
|---|---|
| Distance Measure | ("Euclidean", "Manhattan", "Cosine", ...) |
| Linkage Measure | ("Single", "Complete", "Ward", ...) |
| Dendogram Height Cut | - |

Table 1: Hierarchical clustering algorithm hyperparameters to be tuned.

As this is an unsupervised learning algorithm, there are no ground truth values to evaluate performance. Therefore, we must call upon evaluation metrics such as the Dunn index, Baker-Hubert Gamma index and cophenetic correlation, that look at the nature of the clusters themselves. For this particular report the silhouette index will be used. The effect of each hyperparameter on clustering is discussed below.

- **Distance methods:** The ideal distance measure maximizes the distance between samples in different clusters, and minimizes that within each cluster. As the most commonly used distance measure, the 'Euclidean' distance is chosen as the default. Without further statistical

analysis, it is hard to gain an intuition as to which may be the best. A comprehensive list of possible distances is given in (deza2009encyclopedia).

- **Linkage methods:** These are measures of dissimilarity between groups of observations. Some common examples and their mathematical foundations are given in the lecture notes. Single linkage is fast, but can perform poorly with noisy data. Similarly, average and complete linkage are sensitive to noise and outliers but are more suited to cleanly separated, globular clusters (aggarwal2014data). Ward is a more effective method for noisy data.

- **Dendrogram height cut:** Choosing where to cut a dendrogram in deciding the number of clusters is a case-specific task, however, a general rule of thumb is to cut through the longest vertical line.

Aided by empirically-known advantages (Aggarwal & Reddy, 2014) and given the existence of many potential outliers, as discussed during EDA, ward linkage will initially be used alongside the Euclidean distance measure. Additionally, due to the variation in absolute values of the features, 'X18' in particular, the data is scaled.[2] Using the sklearn.cluster 'AgglomorativeClustering' package, the following dendrogram is produced, grouping the features into clusters[3].

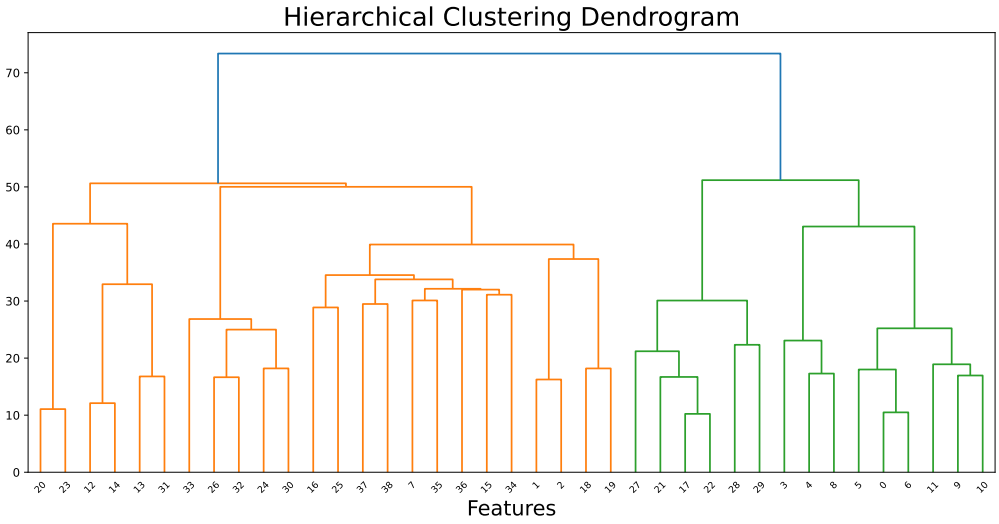


Figure 5: Hierarchical clustering dendrogram using Euclidean distance and ward linkage.

Using these hyperparameters, there seems to be a clear split of the features into two clusters. Visually Euclidean distance coupled with ward linkage performs well. Following our rule of thumb for cutting the dendrogram, we would cut the plot somewhere along the longest vertical line joining 'X20' and 'X21'. Doing so through the middle of the line would result in 20 clusters. As alluded to in Figure refbox2 of the EDA section, 'X1' and 'X18' seem to be related in some manner. This is reflected in the clustering seen hin Figure 5.

The **silhouette method is used to tune the hyperparameters**. This is essentially a measurement based on distances within a cluster and between its nearest cluster. As our data is not very sparse, the 'L1' distance measure is disregarded. **"Euclidean", "Manhattan" and "L2" are chosen** as typical representative choices in clustering analysis. Figure 6 shows the silhouette scores (sklearn.metrics silhouette_score) for each distance and linkage method. This is achieved by

[2]By subtracting the mean and dividing by the standard deviation for each column. The scaled data is used throughout the report due to its aforementioned advantages.

[3]The transpose of the dataset is used when working with features rather than observations.

iterating over all possible hyperparameter combination and concatenating a list of their respective silhouette scores.
It is clear from these graphs, that the ward linkage with Euclidean distance measure and 13 clusters is the optimal choice. This is not too far off the rule of thumb from dendrogram height cut. This results in a silhouette score of $\approx 0.2376$. With an ideal value of 1 indicating highly dense clustering, this method is still far off ideal. One could also have measured performance based on runtime of each combination, however, due to the relatively small size dataset in question this was not included here.

Final Hyperparameters: Clusters = 13 Distance = Euclidean Linkage = Ward


Figure 6: Silhouette scores for each combination of hyperparameters. Initial choice of Euclidean distance and ward linkage is the best according to this tuning process. Hierarchical clustering dendrogram shows the optimal 13 feature clusters using a dashed line at the appropriate cut height. Clusters can be derived by following the vertical lines towards the leaf nodes and ignoring further splits.

## K-means

The single hyperparameter for k-means (given the desired number of clusters k) is the choice of initialisation process. Typically, k random samples are chosen as the initial centroids of the clusters. Thus, the performance is highly dependent on the particular initialisation. The algorithm is guaranteed to converge given enough time but not necessarily to a global minimum. To circumvent this phenomenon, the **computation is run multiple times with differing initialisations.** The iteration which minimises inertia, given in Equation (1), is chosen.

$$\sum_{i=0}^{n} \min_{\mu_j \in C} \left( \|x_i - \mu_j\|^2 \right) \tag{1}$$

C refers to the particular cluster, x the data points within C and $\mu$ the mean of all $x_i$. Furthermore, the sklearn.cluster package KMeans allows us to smartly chose initial centroids that are, in some way, distant from each other using 'k-means++'. The algorithm is run 20 times to **improve the probability of converging to the global minimum**. Implementing this on the scaled, non-transposed data (observations) yields Figure 7. PCA reduces the 39-feature dataset to two dimensions. With this reduction comes a significant decrease in explained variation. The combined variation of the original dataset that is explained by these two dimensions is $\approx 32\%$.

## Kernel K-means

Kernel k-means allows use to cluster data that are separated in a non-linear way. The regular k-means algorithm can only cluster data in a linear fashion. Kernel k-means embeds the data into a feature space rather than simply working off the Euclidean distances between points. The fact that we can express Lloyd's (k-means) algorithm as a function of inner products allows us to do this. Figure 7 shows the clusters derived from the 'kkmeans' package in R using an RBF kernel (discussion later about this kernel).

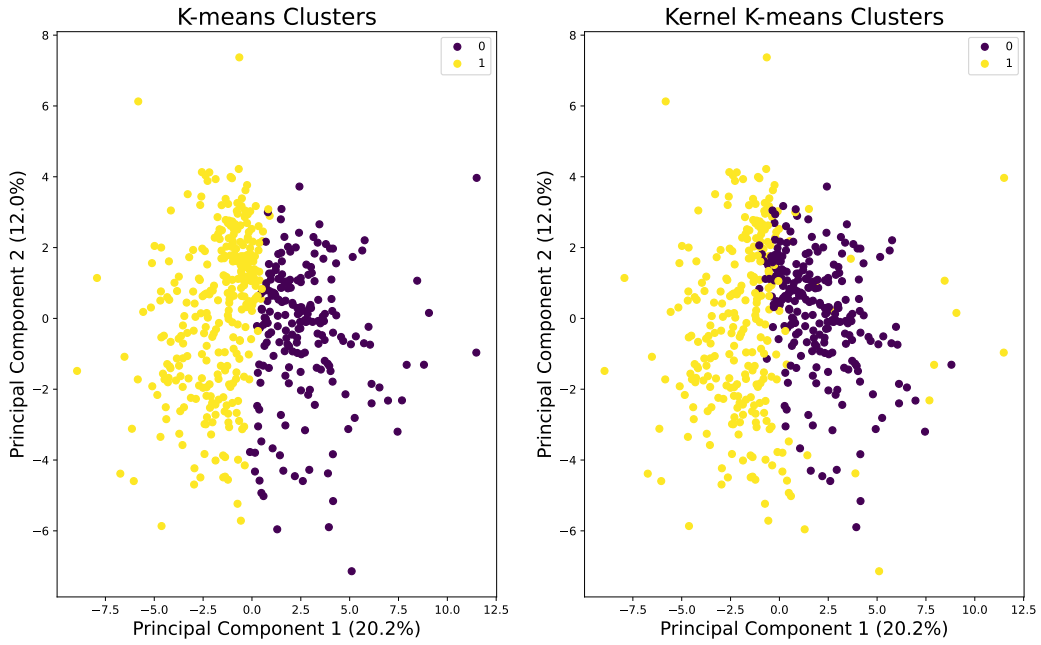


Figure 7: Principal Component Analysis (PCA) used for dimensionality reduction to create 2D plot of k-means clusters. Explained variation of each variable given in brackets. Datapoints are split into two clusters labelled 0 (purple) and 1 (yellow). Both k-means and kernel k-means clusters given above.

**416 out of the 500 observations are clustered similarly by k-means and kernel k-means**. The kernel algorithm clusters extreme values to cluster '1' whereas the regular algorithm seems to split the data linearly down the middle. Some increased cluster overlap from the kernel algorithm illustrates its superiority in dealing with nonlinear separation. This nonlinearity is likely small due to the similarity in clusters between the two graphs. It is worth stressing that the level of explained variation is very low so these graphs must be viewed with caution.

## Lasso Regression

Lasso regression includes an L1 penalty term which tends to remove coefficients corresponding to input variables that have little impact on predicting the target variable. The regularisation term often prevents overfitting of the model. The single hyperparameter in this model is the regularisation parameter $\alpha$ which controls the degree of sparsity of the estimated coefficients. To tune this parameter a train and test split is performed on the data. Given only 500 observations, and the fact that cross validation will inevitably be used to further split the training data into validation sets, a relatively small test size of 20% is used. To ensure the training and test target variables are comparable, their distributions are investigated as seen in Figure 8 below[4]. Due to the randomness of the sklearn.model_selection package 'train_test_split', the splitting process can have a noticeable effect on model accuracy. The effect of this split on the following results will be investigated further in the next section.

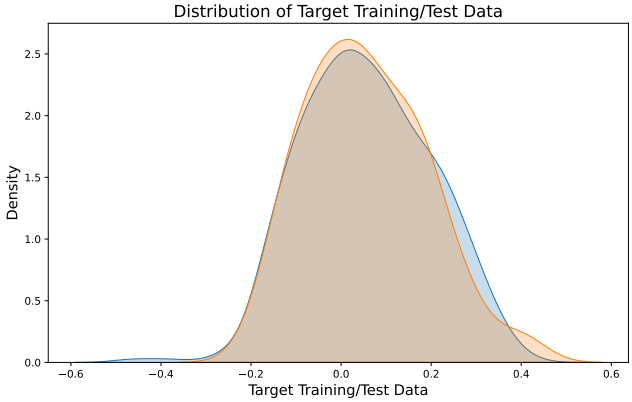


Figure 8: Train/test target variable distributions overlap reasonably well, ensuring that the target variables the models are trained on are similar to those that they will be tested on.

To implement hyperparameter tuning of the Lasso regression model, a pipeline is subsequently created. As discussed previously it is important to scale the data. First, **the training data is scaled** by subtracting the mean and standard deviation of the observations within the training data. Importantly, the test data should be scaled using these values as well to avoid any leakage of information from the test to training set [5]. Secondly, a Lasso regression model is created with a given $\alpha$ value. The sklearn.linear_model package **'LassoCV'** is used for this task. **K-fold cross validation** is used to determine the optimal value as Leave-One-Out cross validation was deemed unnecessarily computationally expensive and is often prone to overfitting. 10 folds are initially chosen. This represented a good trade off between computational expense and potential overfitting for this relatively small dataset. The k-fold cross validation procedure is performed using least angle regression. When evaluating the model, we must keep in mind from the EDA, that there is potential multicollinearity in the dataset. This cross validation procedure is susceptible to this. However, Lasso regression itself can deal well with this due to its estimation of sparse coefficients i.e. it can eliminate variables that are dependent on eachother.

The above pipeline is fit to the scaled training data and iterated over a range of $\alpha$ values. These $\alpha$ **values were concentrated towards 0 (between $10^{-5}$ and $10^{-1}$)** given that these values tend to be small. The mean squared error (MSE) of each fold, adhering to the cross validation algorithm above, is calculated over this range of values and plotted in Figure 9. The average of these values is used to obtain an optimal value of $\alpha \approx 0.0007$. The **MSE associated with this**

[4]This plot only relates to the target variable and other checks between the features themselves could have been used (further discussion in next section).

[5]The appropriate scaling is implicitly used for other models in this report.

**model is $\approx 0.0052$.**

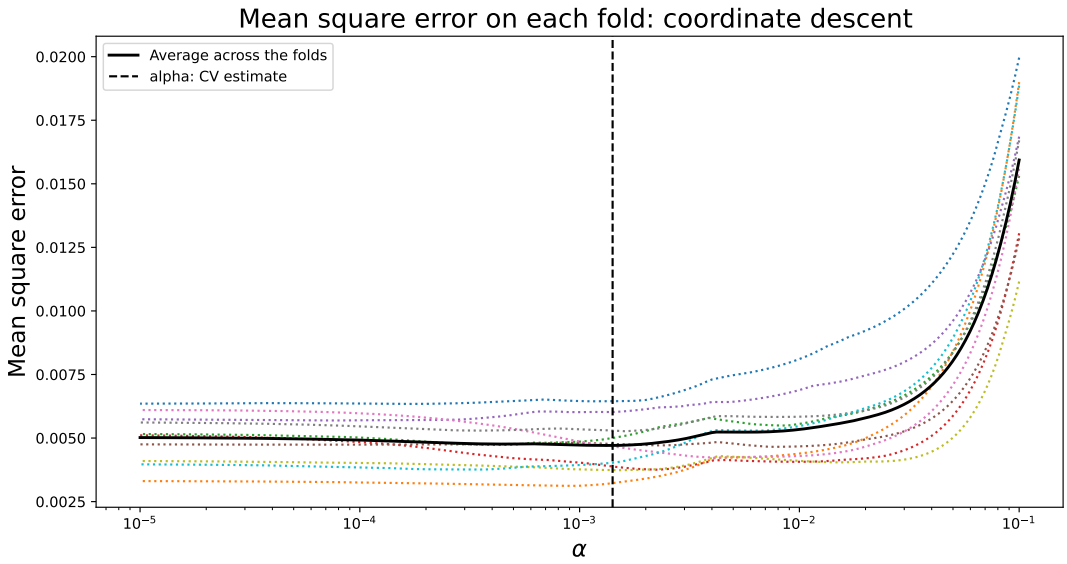


Figure 9: The average MSE over each of the 10 folds is given by the smooth black line. An $\alpha$ estimate of $\approx 0.0007$ minimises this MSE across all folds as given by the dashed black line.

## Random Forest

Random forests is a tree-based ensemble method, highly related to bagging, that operates over a fraction of the existing input variables. In comparison to the algorithms we have looked at previously in this report, there are significantly more parameters in this model. Given the amount of choice, a **specific set of important parameters are tuned** whilst leaving the others constant (in their default state according to the package used). This allows us to illustrate the model selection process whilst also leaving room for further improvements. Tying back to the EDA, random forest ensemble models can deal with multicollinearity through their random choice of the features used at node splits. This reduces the likelihood of dependency on highly-correlated features. The sklearn.ensemble package **'RandomForestRegressor'** is used to implement the random forest algorithm. The three main hyperparameters chosen for tuning are given in Table 2 along with their tuning ranges.

- **max_features**: The size of the (random) subsets of given inputs used when splitting each node. There exists a bias/variance tradeoff for this parameter, with variance reduced (bias increased) for a smaller number of chosen features. More inputted features typically yields higher correlation between individual trees, breaking the assumption of independence associated with bagging methods. When considering computational cost, max_features tends to have a linear relationship with runtime.
- **max_leaf_nodes**: The max number of leaf nodes allowed. A low value can lead to overpopulated nodes and underfitted data. The opposite can result in overfitting.
- **n_estimators**: Total number of trees. Stability is weighed against computational cost for this parameter. One could expect that beyond a critical value, results will remain largely the same.

With these points in mind, the first two parameters are **tuned using out-of-bag errors** to obtain optimal values and, subsequently, the number of trees is increased to the lowest stable point.

Out-of-bag prediction errors are more accurate than mean prediction errors in general. They are calculated based on the inputs a particular tree has not seen during training and, as such, they are closer to true estimates of model performance. They exhibit a higher MSE value and avoid the bias associated with mean predictions i.e. on already-seen data.

The ranges of each hyperparameter are chosen in accordance with their respective tradeoffs as described above. The upper limit of 'max_features' is chosen to be the total number of features, iterating over each value to find the optimum. A mid range of 'max_leaf_nodes' values (30 - 125) relative to the number of observations is chosen mirroring the expected underfitting/overfitting balance. 'n_estimators' extends to the total number of samples and iterates over large values due to its lack of sensitivity to overfitting. Given that this dataset is relatively small, these ranges should not be too computationally expensive to iterate over. Table 2 shows the optimal values achieved, according to out-of-bag error scores, by iterating over each possible combination of 'max_features' and 'max_leaf_nodes'. Given these values, 'n_estimators' is iterated over using the same procedure to obtain its respective optimum.

| Hyperparameter | Tuning Range (Step Size) | Optimal Values |
|---|---|---|
| max_features | 3 - 39 (1) | 35 |
| max_leaf_nodes | 30 - 125 (15) | 120 |
| n_estimators | 100 - 500 (50) | 350 |

Table 2: Random forest hyperparameter tuning.

The optimal random forest model has 350 trees, 35 features at each split and 120 maximum number of leaf nodes. Table 3 shows the gradual improvement of MSE and mean absolute error (MAE) values for the models as the hyperparameters are tuned. Variation in these values is due to the randomness of the random forest algorithm. This outweighs any marginal improvement in this particular example.

| Model | Description | MSE (x100) | MAE (x100) |
|---|---|---|---|
| Default Model | Base random forest model with default parameters as described by 'RandomForestRegressor' package. | 0.3236 | 4.2857 |
| Tuned Model 1 | Model with 'max_features' and 'max_leaf_nodes' tuned . | 0.3187 | 4.3488 |
| Tuned Model 2 | Model with 'max_features', 'max_leaf_nodes' and 'n_estimators' tuned. | 0.3261 | 4.3433 |

Table 3: MSE and MAE values for random forest models as the tuning of hyperparameters progresses.

As mentioned previously, lasso regression results in some zero-valued coefficients corresponding to variables of less importance. This can be compared to a measure of variable importance for our random forest model. Many such measures exist, however this choice is not the main focus here. Figure 10 shows the evaluation of variable importance using permutation feature importance (sklearn.inspection.permutation_importance) which does not exhibit bias towards high-cardinality features[6]. Figure 11 shows the variables selected by the Lasso regression model. This model seems to depend on more features than the random forest model, however, there is reasonable agreement between the two in terms of the most important features 'X29' and 'X31'.

[6]For completeness another variable importance technique based on mean decrease in impurity was used to ensure

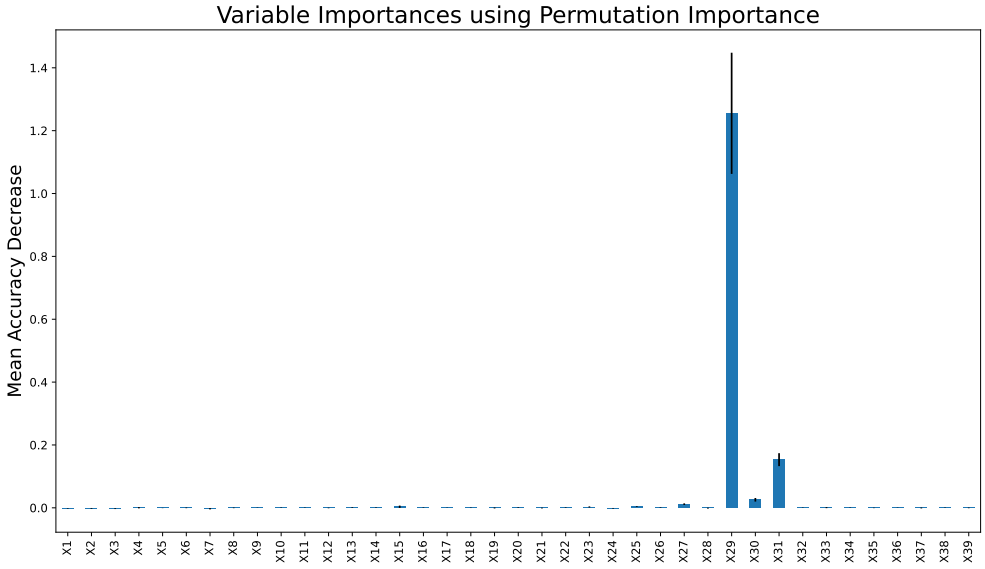


Figure 10: Permutation importance deems features 'X29' and 'X31' significantly more important than all other features for random forest prediction with the tuned model. The standard deviation of the results given by the black vertical lines, leaves us with little doubt of the accuracy of this conclusion.

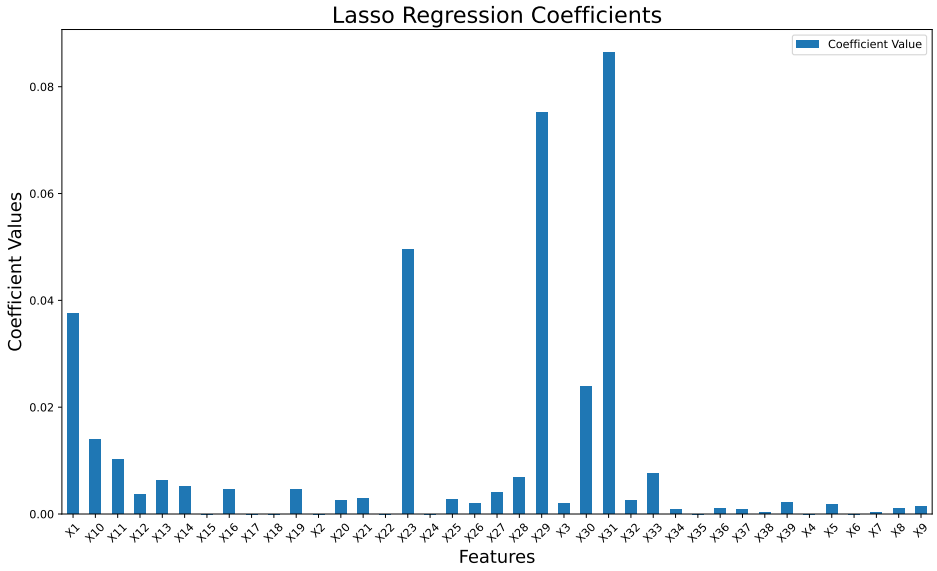


Figure 11: Lasso Regression coefficient values somewhat mirror the result given by the random forest variable importance. Again features 'X29' and 'X31' have significantly higher values than other features. However, more emphasis is placed on other features with this model.

Prediction using the random forest model is significantly better than the Lasso model in terms of both MSE and MAE calculated on the test set. The **Lasso model boasts MSE and MAE values of $\approx 0.0052$ and $\approx 0.0565$ respectively as opposed to the $\approx 0.3261$ and $\approx 0.0434$ values for the random forest model**. The complexity of the model and number of parameters is likely to be responsible for this.

comparable results. A similar result was found.

One method we could use to further improve our models is feature selection/engineering. Up to this point, we have not looked into the specific features themselves. Perhaps some of these features are redundant or simply input noise into the models. To explore this, **all of the features with both low permutation importance for the random forest model and zero coefficient value for the Lasso model are removed** before the training and test split. The reduced dataset, now with 29 features, is subsequently split and scaled as before. The same model tuning procedure is carried out and performance measured. The MSE values were comparable with previous sections however the tuning time was greatly reduced especially in the case of random forests. Depending on the resources available this, could be viewed as an improvement on the models. Further feature engineering/transformations of the variables themselves could also be performed on the dataset to improve results.

Despite a larger tuning runtime, the random forest model performs significantly better than the Lasso model. With the speed improvements mentioned above, this would be my recommended model. One particular concern relating to this choice would be the potential multicollinearity of the features. Random forest models are typically quite good at dealing with this, however, further tests would need to be carried out to ensure correlations between variables are low enough to satisfy the assumptions of the underlying statistical theory. Another concern would be the noticeable effect on results caused by the randomness of the train/test split as discussed in the following section.

### Train/Test Split Evaluation

As mentioned previously, the random nature of the train/test split can have an effect on the results. To account for this, the previous **tuning procedures for both Lasso and random forest models are themselves repeated over multiple train/test splits**[7]. The effect on MSE values and variable importance is explored. For each model, some of the hyperparameters are reduced to reduce computational complexity. Their ranges are kept the same for consistency. 50 seeds are created and the MSEs of the tuned models are graphed in Figure 12 over each iteration. Visually we can see that there is a reasonable amount of variation (standard deviations = $\approx 0.0006$) over different splits. There is a mean value around which the MSEs fluctuate. This mean level could be used as the actual representative MSE figure for the models. It is interesting to see also that the MSE figures for both models follow a similar trend indicating a relationship between the nature of the split and accuracy of the model. This likely refers to the amount of overlap between training and test data.

Figure 13 below investigates the effect on variable importance. The number of zero-valued coefficients is graphed over iterations for the Lasso model. There seems to be large variation with a range of 10 - 19 zeros depending on the split. The standard deviation of these values is $\approx 2$ which supports this. This indicates that there is a significant effect from the randomness associated with the split. On the other hand, the average permutation importance graph is very comparable to the single graph given in Figure 10. We can conclude that when it comes to Lasso regression in particular, the train/test split has a large effect on results. This should be taken into account in further analysis.

[7] The proportion between train and test sets 20% is kept constant.

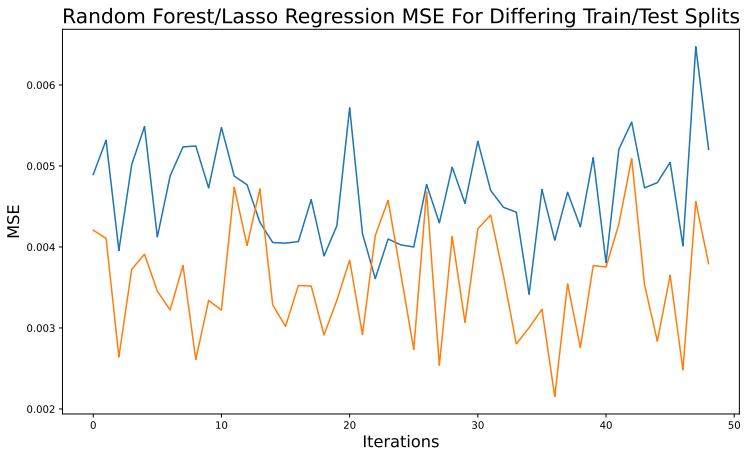


Figure 12: MSE random forest (orange line). MSE of Lasso model (blue line). MSE of both models varies significantly around a mean level over train/test splits. Additionally, the superiority of the random forest model is evident here with an overall lower MSE.

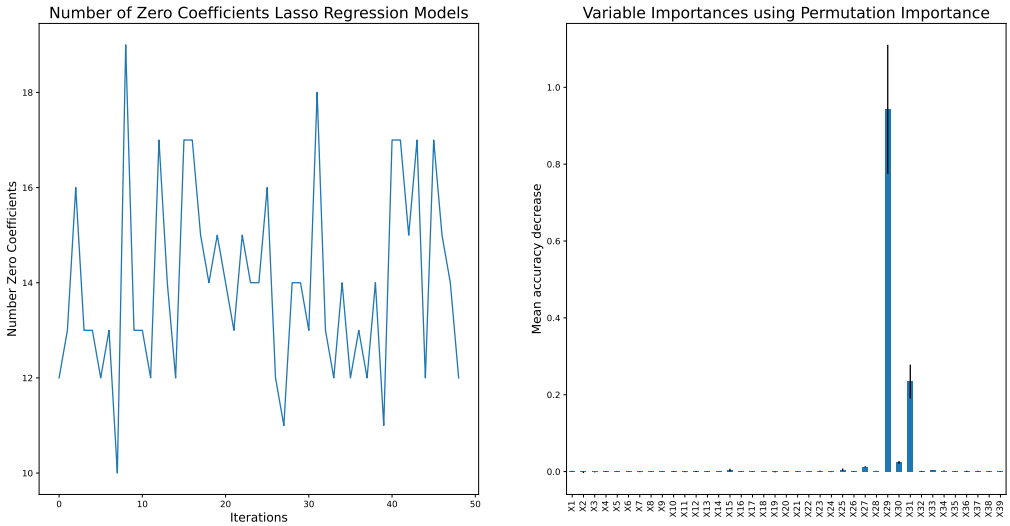


Figure 13: Variable importance comparison shows large effect on Lasso regression model.

## Gaussian Processes

Gaussian processes (GP) require pre-specification of a mean and covariance function which encode structural assumptions about the function we believe describes the data. The covariance function (kernel) hyperparameter is of particular interest in this modelling task. As such, an agnostic mean function in the form of a **zero mean prior** is implemented, eliminating the need to tune any related parameters. To account for this, each target variable is scaled down by the total column's mean value. To inform our choice of kernel, some initial EDA is performed on the data. Figure 14 shows varying water temperatures over time associated with the 39 observations where 'd' = 0. Perhaps it would be useful to include a kernel that encodes some form of periodicity. In the absence of additional significant prior information, it makes sense to choose the **radial-basis function (RBF) kernel** as it is typically the default kernel for GPs (Duvenaud, 2014). A product of the periodic and RBF kernel is also implemented so as to investigate the effect combining kernels has

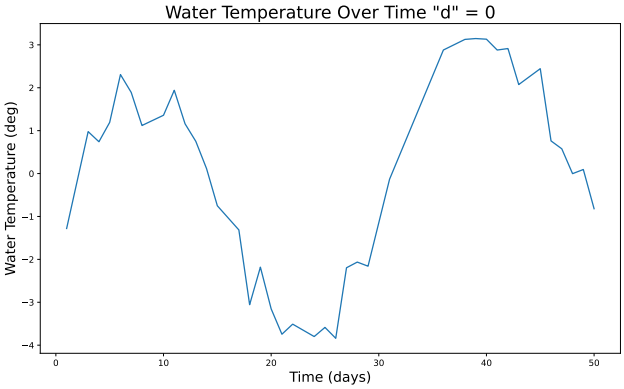


Figure 14: Possible periodic trend in temperature with time.

on modelling accuracy. For further simplicity, isotropic kernels have been chosen. Table 4 outlines the parameters to be tuned for each of the kernels. **Amplitude** dictates the vertical magnitude of the function of interest whereas the **period** directly encodes information about its periodicity. **Lengthscale** controls the function's smoothness through the relationship between the correlation and distance between two points within the function's domain.

In this report, **model selection is performed based on comparing the marginal-likelihoods** (ML) of each model. Parameters are tuned over a predetermined suitable range. The model with the largest ML is chosen as the one that fits the data best. Given the definition of lengthscale, its bounds are given by the min and max spacing between observations. A small value ($10^{-3}$) is chosen in place of the min to avoid a possible zero value. From the EDA a max value of 38 is chosen. With computational expense in mind, a moderate range of 1-10 is chosen for the amplitude. Larger ranges and more precise increments may lead to better results, however the aim of this section is to outline the process of tuning parameters rather than simply striving for the most accurate model. From Figure 14, period values around 30 seem sensible, therefore 10-50 is chosen in increments of 5[8].Beyond the parameters of the kernels themselves, the **likelihood noise variance could also be tuned**. A moderate range of 0.1 - 5 in increments of 0.1 was deemed sufficient to illustrate the combinatory tuning process.

The log marginal likelihood from Equation (2) below is used as the performance metric for each model assuming a zero mean prior[9].

$$\log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^\top \boldsymbol{K}_{\boldsymbol{\theta}}^{-1}\boldsymbol{y} - \frac{1}{2}\log|\boldsymbol{K}_{\boldsymbol{\theta}}| + \text{ const}, \quad \boldsymbol{K}_{\boldsymbol{\theta}} := \boldsymbol{K} + \sigma_n^2\boldsymbol{I} \tag{2}$$

Where $\sigma_n$ represents the likelihood the noise, K is the kernel of choice, y is the target variables and X is the design matrix. The user-defined function 'log_marg_likelihood' implements this expression. The user-defined function 'fit_gp_posterior' is used to calculate the posterior predictive distribution mean and variance by Gaussian conditioning as described by Equations (3) and (4) below.

[8]Having initially investigated the fit for the default kernels, the periodic kernel was noted as a particularly poor model of the data. With this in mind, to reduce the computational expense of the tuning process, only a few period parameter values were iterated over. The tuning procedure could have been extended to more accurately account for the period if further model tuning was necessary. This explains why the posterior fails to converge for this kernel in the following analysis.

[9]The constant term is not overly important here as we are looking at ML values relative to eachother.

$$\mathbb{E}\left[f_* \mid X, \boldsymbol{y}, X_*\right] = m_{\text{post}}\ (X_*) = \underbrace{m\left(X_*\right)}_{\text{prior mean}} + \underbrace{k\left(X_*, X\right)\left(K + \sigma_n^2 I\right)^{-1}}_{\text{"Kalman gain"}} \underbrace{\left(\boldsymbol{y} - m(X)\right)}_{\text{error}} \tag{3}$$

$$\mathbb{V}\left[f_* \mid X, \boldsymbol{y}, X_*\right] = k_{\text{post}}\ (X_*, X_*) = \underbrace{k\left(X_*, X_*\right)}_{\text{prior variance}} - k\left(X_*, X\right)\left(K + \sigma_n^2 I\right)^{-1} k\left(X, X_*\right) \tag{4}$$

The $m\left(X_*\right)$ is set to 0 given our chosen mean prior. Table 4 shows the values of each kernels' parameters after tuning as well as their respective maximised log marginal likelihoods.

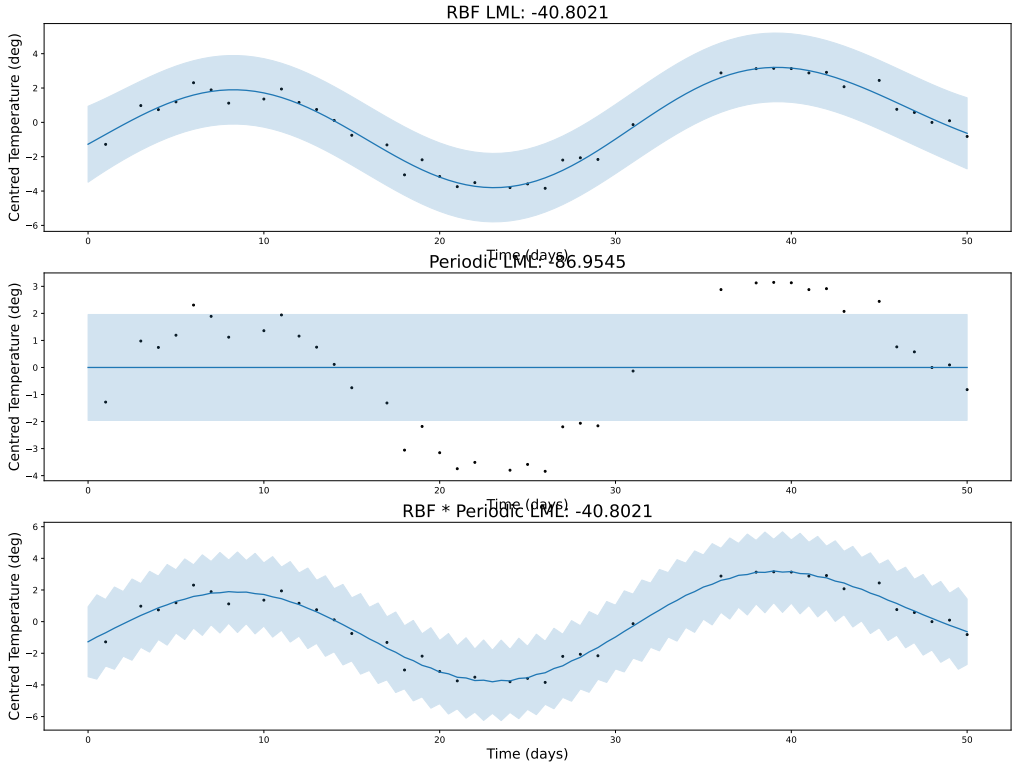


Figure 15: Three GP models fit to water temperature data. The shaded region describes the 95% predictive confidence intervals. Test data over the range of 't' is used to plot the models.

Figure 15 demonstrates the fit of each model to the data. It is no surprise that the effect of the periodic kernel is essentially eliminated for the third model. Additionally, **95% predictive confidence intervals** have been included taking the form:

$$\left(\mu - z_{0.975}\sqrt{\sigma_\star^2},\ \mu + z_{0.975}\sqrt{\sigma_\star^2}\right)$$

where $\mu$ is the predictive posterior mean $\mathbb{E}\left[f_* \mid X, \boldsymbol{y}, X_*\right]$, $z_{0.975}$ is the standard normal quantile value $= 1.96$ and $\sigma_\star$ is the standard deviation of the predictive posterior distribution including likelihood noise $\sqrt{\mathbb{V}\left[f_* \mid X, \boldsymbol{y}, X_*\right] + \sigma_n}$. From Table 4, the **RBF kernel model has the lowest ML**[10] ($\approx -40.8021$) and therefore seems to describe the data the best out of the three. Again, it is important to remember here that other kernels may be more suited to the data but the process

[10]The third model converges to this model.

| Parameter/Metric | Tuning Range | RBF Kernel | Periodic Kernel | RBF Kernel * Periodic Kernel |
|---|---|---|---|---|
| Amplitude | 0.1 - 10 (0.1) | 4 | 0.1 | 2 |
| Lengthscale | 0.1 - 38 (0.1) | 9.8 | 0.1 | 9.8 |
| Period | 1 - 50 (5) | - | 1 | 1 |
| Noise | 0.1 - 5 (0.1) | 0.5 | 2.2 | 0.5 |
| (log) Marginal-Likelihood | - | -40.8021 | -86.9545 | -40.8021 |

Table 4: GP kernel parameter tuning.

of updating parameters based on ML leads is the focus. Th single periodic kernel yields a poor model of the data likely because it fails to accurately represent non-periodic features. This may also explain why combining this kernel with an RBF kernel yields much better results in terms of ML.

Using the 'fit_gp_posterior' function with an input of 't' = 35 to the RBF kernel model, yields a **predicted posterior mean of 13.8383 degrees**. This mean value represents the most accurate prediction over the posterior distribution. Using the entire posterior distribution, one can calculate the probability of temperatures equal to, or above, 13 degrees. The 'norm.cdf' method is used to calculate the probability of obtaining such a temperature value given 't' = 35. The input to this method is scaled down by the mean of the target variable column so as to be consistent with the model construction. We see that there is a $\approx 99\%$ chance that the temperature will be 13 degrees or above for this day according to the RBF model. The standard error ($\approx 0.001270$) for this result was very low indicating that it is a trustworthy statistic.

It is common to combine kernels when dealing with multiple inputs especially when they are of different datatypes (Duvenaud, 2014). To investigate this, two separate kernels, with individual input dimensions 't' and 'd', will be summed together to create an overall additive kernel. This configuration does not model the interactions between 't' and 'd' well so other kernel combinations (perhaps using their product) may produce more accurate models. This section is simply a discussion of how one might model multi-dimensional data in the context of kernel choice. Given the results in the previous section, an RBF kernel is used to model the 't' input. To gain more intuition about the nature of 'd' Figure 16 shows the mean water temperature as a function of distance.

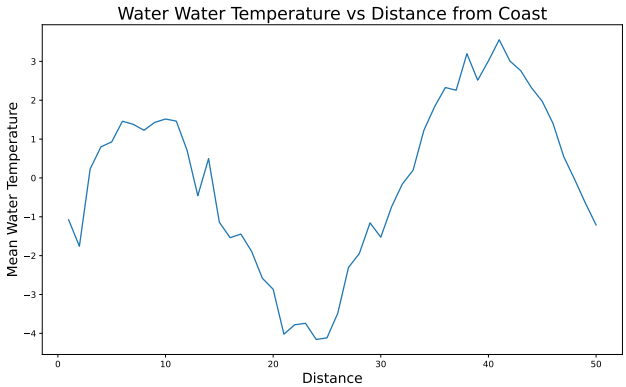


Figure 16: Water temperature vs distance. The mean result for all time values is taken for this plot. There is an evident downward trend.

The downward slope of the curve suggests that water temperature decreases the further away from the coast one measures. It would be wise to choose a kernel that models this behaviour. A linear kernel may be able to do this. Having shown how to manually create and tune GP models in the previous section, the built-in function 'GaussianProcessRegressor' is used for versatility here. The 'DotProduct()' kernel represents a linear kernel in Python. This function contains an in-built optimizer based on ML values similar to the previous section. A resulting ML value of $\approx -740$ indicates a poor fit to the data. Other kernel choices may perform better.[11]
In order to predict and plot water temperature at 't' = 55 as a function of distance, a dataframe is created containing 'd' values from 1-19 and constant 't' values of 55 for the entire column. This encapsulates the range of distance values in the original dataset but only focuses on day 55 (which is outside the original range of 't' values). Using the 'predict' method associated with this Python function on this input dataset, Equations (3) and (4) are implemented. Water temperature as a function of distance according to this model is given in Figure 17 below.

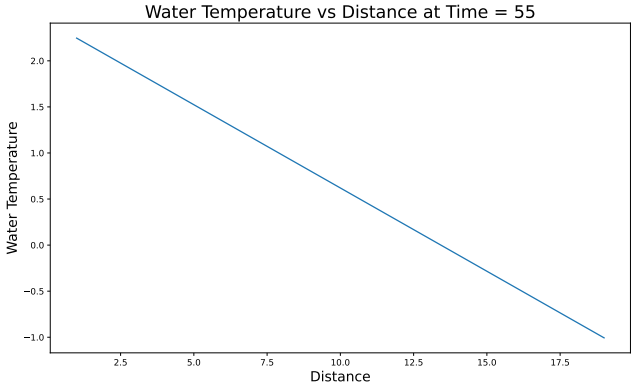


Figure 17: Pedicted water temperature as a function of distance at day 55. The kernel choice captures the downward trend of temperature vs distance.

As expected there is a downward trend as distance increases. We have no data regarding 55 days in the original dataset but have used GPs and our knowledge of kernels to predict the temperature behaviour.

[11]Other choices were implemented out of curiosity, however in the interest of demonstrating thinking about the data, the current method was retained.

# Appendix

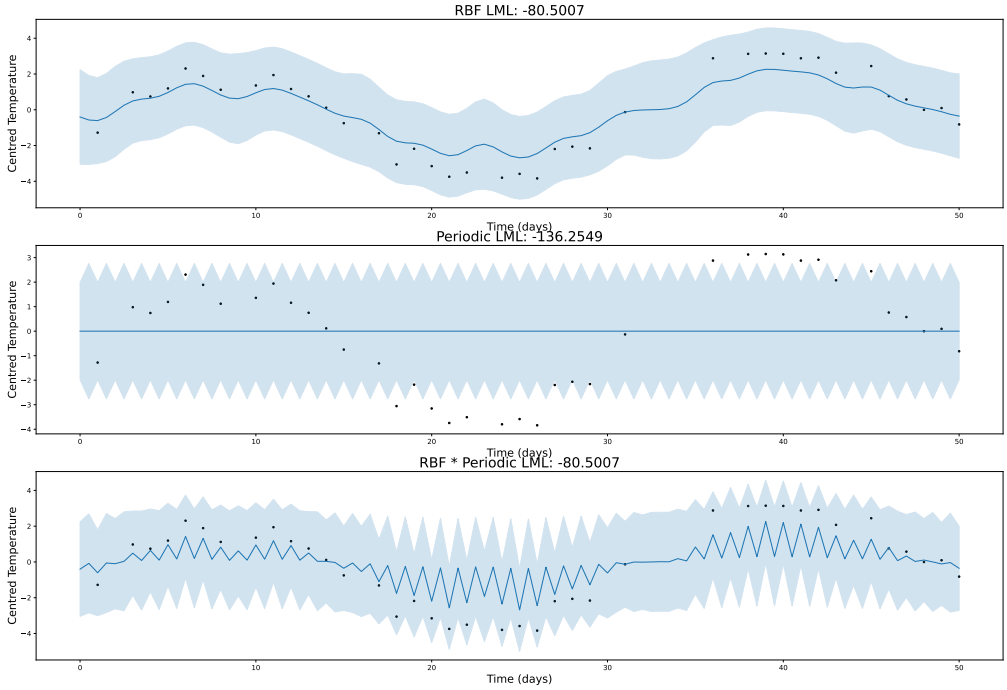


Figure 18: Three default GP models fit to water temperature data. The shaded region describes the 95% predictive confidence intervals. These are results before any tuning has occurred.

# References


Aggarwal, C. C., & Reddy, C. K. (2014). Data clustering. *Algorithms and applications. Chapman&Hall/CRC Data mining and Knowledge Discovery series, Londra*.

Duvenaud, D. (2014). The kernel cookbook: Advice on covariance functions. *URL https://www. cs. toronto. edu/duvenaud/cookbook*.

Lau, D.-H. (2020). *Chapter 2. normal linear models.*

Murtagh, F., & Contreras, P. (2012). Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *2*(1), 86–97.