Machine Learning Assignment 2

Kyriacos Xanthos
CID: 01389741

March 12, 2022

Question 1

In this question we consider a dataset that arises from a single cell analysis of cells in the immune system. This dataset contains 40 biological features which are assosiated with cell proliferation, cell cycle and cell morphology. In total we have 500 observations (cells), with no missing values. We will build a regression model that predicts the log-ratio of the concentration of two proteins of interest (first feature in the dataset y).

For explanatory data analysis it is quite difficult to visualize a dataset with 40 features but we created a correlation plot as seen in figure 1 where we can identify if there are any strong correlations between the variables. We observe that there is a relatively strong correlation between variables $X_8 - X_{15}$ and a negative correlation between variables $X_{21} - X_{25}$ but overall for the size of the dataset we note that these correlations are not enough to make the decision of excluding them or create interactions between them.

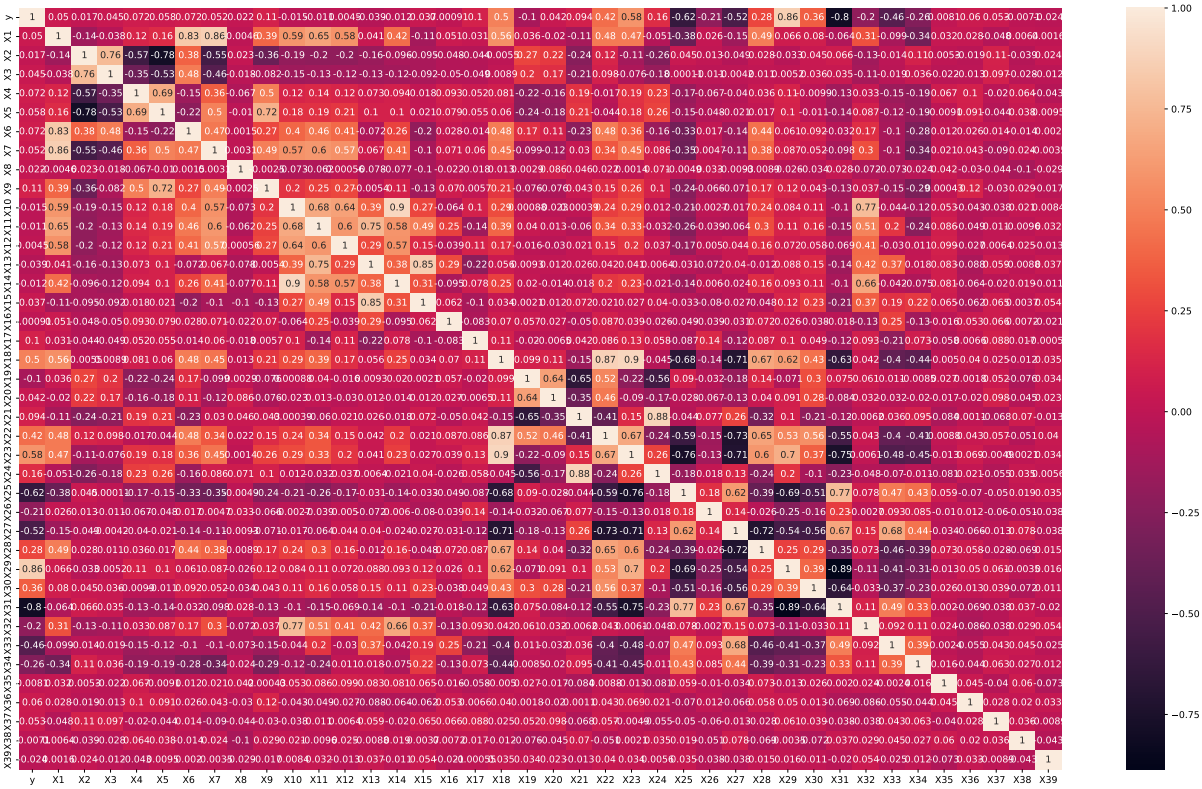


Figure 1: Heatmap of correlation of variables in the dataset. Light colors indicate high correlation and dark colors show anti-correlation (scale 1 to -1).

Exploring the mean and standard deviation of each variable we notice that some variables like X_1, X_6 have disproportionately large mean in comparison to the other variables (order of 10^1) and some have disproportionately low mean (order of 10^{-2}) which suggests we will need to use scaling to capture the effect of each variable. We can continue to clustering methods for the dataset to see if there are any dimensionality reduction methods that would help with the accuracy of the regression models.

Part A

1 Hierarchical Clustering

In this part we focus on the biological features $(X_1, \ldots, X_{39})$ and use Hierarchical Clustering to divide the biological features into an appropriate number of groups. We will use an Agglomerative approach where each observation is first assigned to its own cluster and then the pairs of these clusters are merged *recursively* until all the observations (features in our case) are grouped in one cluster [1].

In order to do this for our dataset we transpose the dataframe so that we cluster the biological features instead of the cells. We then use a `StandardScaler` from `skicit-learn` [2] to make sure all the features have mean zero and standard deviation one.

To perform the clustering we use `AgglomerativeClustering` function and the hyper-parameters we have to choose are the number of clusters and the linkage criterion that determines which distance measure to use between sets of observation. The linkage criterion did not change our results in a significant way so we chose the `ward` method which minimizes the variance of the clusters being merged [3].

In order to choose the optimum number of clusters we use the Silhouette Score which is a graphical tool used to measure how tightly grouped samples in the clusters are [4]. This is calculated by:

$$s_i = \frac{b_i - a_i}{\max\{b_i, a_i\}} \tag{1}$$

where a_i is the avarage distance between sample x_i and all other points in the same cluster, and b_i is the cluster separation from the next closest cluster.

Note that s_i can take values between -1 and 1 with -1 meaning they are assigned in the wrong cluster and 1 indicating instances are inside their own cluster and away from others.

Trying a grid of clusters from 2 to 20 we observe in figure 2 that the best silhouette score is given from 2 clusters with a score of 0.9. We can further visualise the silhouette score in figure 3. This is the usual way of visualizing silhouette scores but the specific figure does not seem very informative, and this is because only one feature seems to be clustered in one of the two clusters.

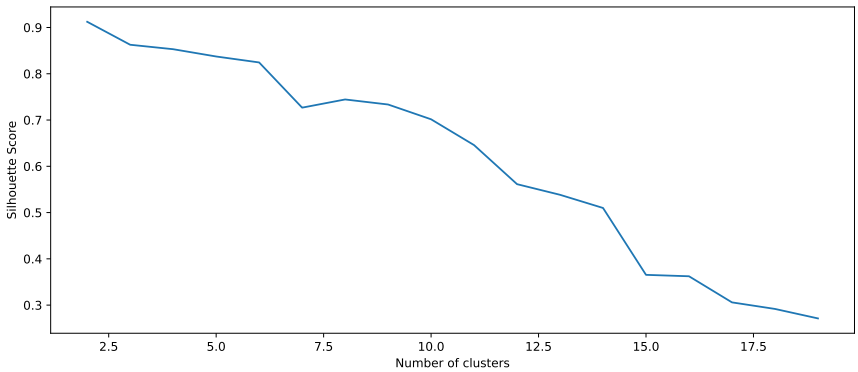


Figure 2: Grid from 2 to 20 clusters and their relative silhouette score according to equation 1.

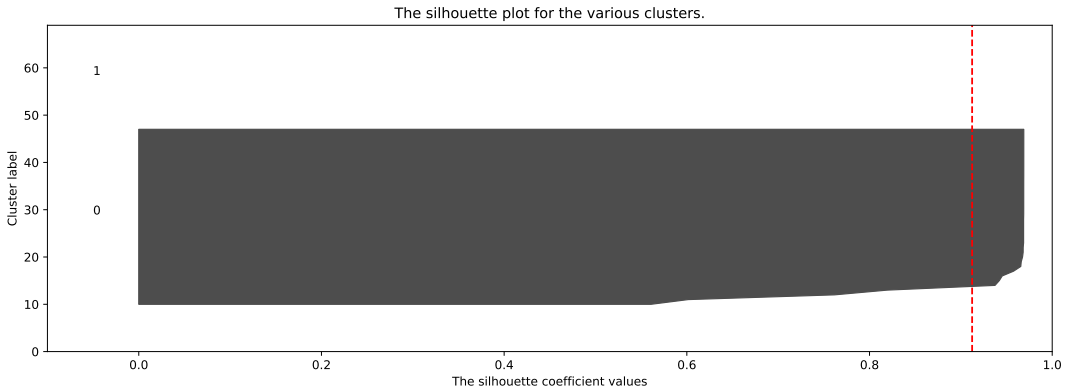


Figure 3: Visualisation of silhouette score for 2 clusters. The red line representing the average silhouette score.

We can now visualise the clustering with the best number of clusters in a dendogram. This is shown in figure 4 and we observe that the only one feature is clustered in one cluster and the rest are in the other cluster. Looking back at the correlation plot 1 we cannot observe any strong positive or negative correlation of feature X_{17} with the rest of the features. Since we do not know what X_{17} feature is, we cannot intuitively understand why it is the only feature clustered in one of the clusters. For this reason, we will not cluster the features for the regression models later on.

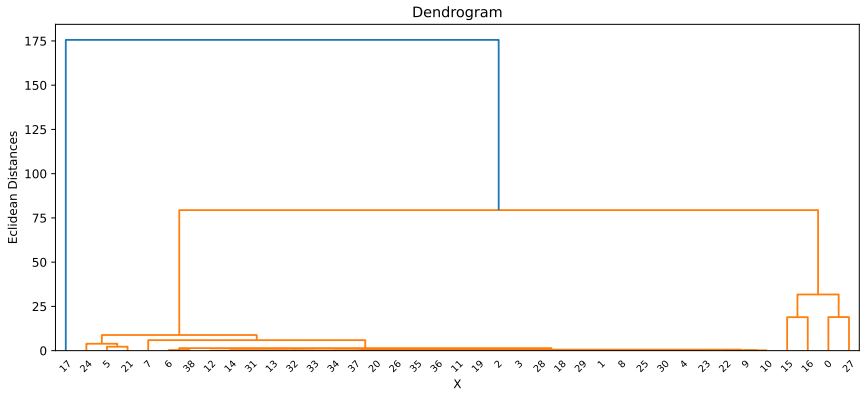


Figure 4: Dendogram for the clustering of features using *ward* as linkage and Euclidean distance for the metric to compute the linkage.

2 K-means

Now we apply the K-means algorithm [1] to cluster the cells into two groups. We again use standard scaling to transform all the observations to have mean zero and variance one. We then use the `KMeans` function to cluster the cells in two clusters. The method used by this function is actually K-means++ which is an algorithm that instead of initializing centroids randomly, it prefers to initialize them first using a specific initialization procedure that clusters the centrers before proceeding with the k-means optimization iterations. It is actually guaranteed that this algorithm will find a solution that is in the order of $\log(k)$ competitive to the best k-means solution [5].

After initializing the centroids the K-Means algorithm is used [6]. We prefer to use this algorithm since it converges much faster than normal K-Means.

To visualise the clustering of this clustering algorithm we perform Principal Component Analysis (PCA) which is again another unsupervised algorithm that performs dimensionality reduction from 39 features to 2, linearly independent *principal components*. The two clusters computed can be seen in figure 5. The silhouette score we get for this clustering is 0.125 which is quite small but note that the range of the silhouette goes from -1 to 1. Therefore, the fact that the score is positive means that most of the points were clustered in a centroid that is closer to the points than to the other centroid.

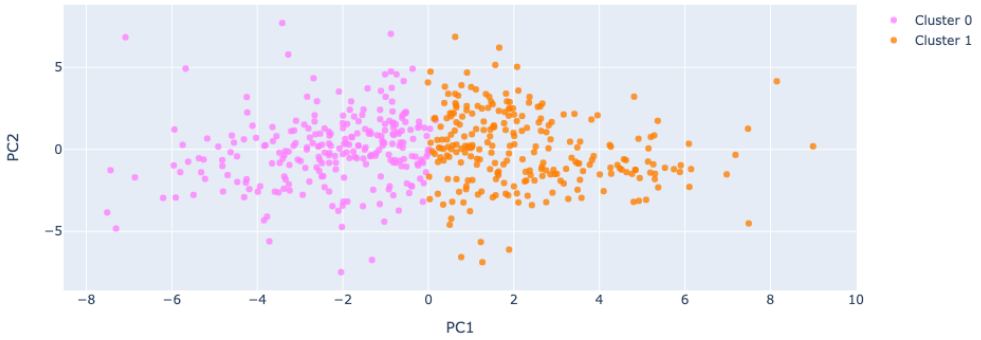


Figure 5: 2-dimensional visualisation of K-Means Clustering of cells using PCA.

We will now use Kernel K-means for clustering. This algorithm works exactly like K-means with the difference that we embed the data into a feature space of our choice and look for centroids there. The details of how the algorithm works can be found in the lecture notes page 272 [1].

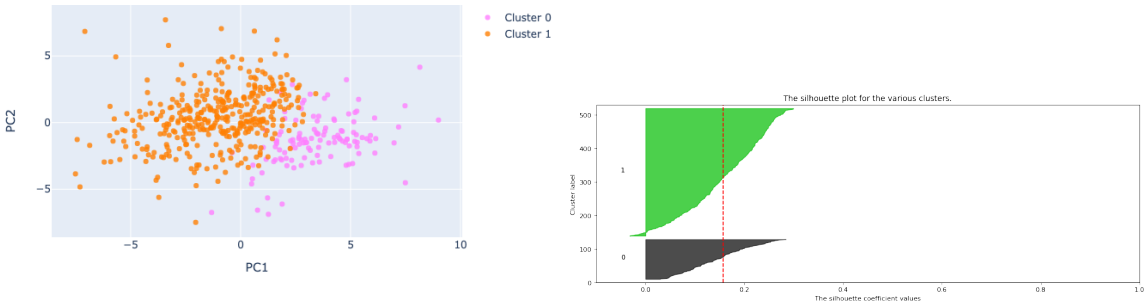
To implement this we used a function created by Mathieu Blondel [7] which performs all of the calculations necessary and uses kernels from the `sklearn.metrics.pairwise` library.

We tried different kernels and assesed their fit by looking at the PCA plots and also looking at their silhouette scores, which are summarised in table 1.

kernel	linear	polynomial	rbf	laplacian	cosine	sigmoid
s_i	0.125	0.157	0.116	0.126	0.125	0.124

Table 1: Silhouette scores as defined in equation 1 for each kernel.

Note that the linear kernel is simply an identity transformation and therefore it is the same as the K-Means we used above. We can see that a polynomial kernel of degree 3 gives the best silhouette score and the clusters are well separated as seen in figure 6. Figure 6b shows the silhouette plot which shows that there is a relatively equal split between the two clusters and the average silhouette measure score is given by the red line with the value 0.157.



(a) Clusters Plot

(b) Silhouette Score Plot, with the red line representing the average silhouette score.

Figure 6: Kernel K-Means using Polynomial Kernel with degree 3.

Comparing the clusters between the Kernel K-Means and K-Means we observe that for the polynomial Kernel, 73.4% of the points are clustered similarly ie. they belong to the same cluster for both algorithms. Note that for the sigmoid kernel 97.1% of the points are

clustered similarly but since we observe a higher silhouette score for the polynomial kernel we believe that it is more appropriate Kernel for our study.

In conclusion, we find that K-Means clustering does not provide an accurate measure of clustering our observations together and therefore we will not use it for the regression model.

Part B

In this part we will build regression models to predict the log-ratio of the concentration of two proteins of interest given all other features. We split the dataset into training and testing subsets of sizes 70% and 30% respectively.

Lasso

Lasso regression is an extension to linear regression by introducing an l_1 penalty on the coefficients to the standard least square problems and attempts to solve the optimization problem:

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1 \tag{2}$$

where in our study the coefficients β correspond to the weights of each variable $\boldsymbol{X} = (X_1, \ldots, X_{39})$, $\boldsymbol{y}$ the response, and λ the penalty which is the hyperparameter we need to tune.

We use the `Lasso` and `GridSearchCV` function from `sklearn` to go through a grid of λ and find the λ that finds the best score for negative RMSE defined as:

$$\text{rmse} = \left(\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right)^{1/2} \tag{3}$$

where $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$.

To tune the hyperparameter λ we use cross validation. The method is that we split the data randomly, into K-folds. We then fit the model K times, and we validate on k^{th} fold. We then train on the remaining $K-1$ folds. The purpose of averaging across multiple folds of the data is that we decrease the variability in the model fit and validation error which is caused by the randomness of the splits. The grid of λ values we iterate over spans from 0.0001 to 0.02 with 80 points. This grid was chosen after multiple tries to find a global minimum of the RMSE function.

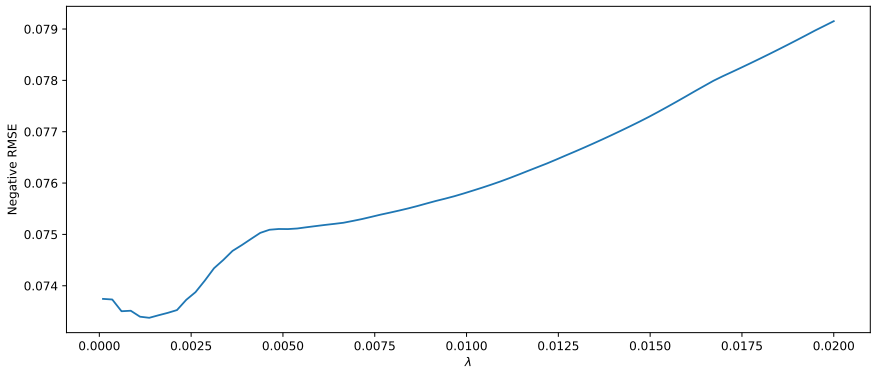


Figure 7: RMSE scores for a grid of hyperparameter λ.

As seen from figure 7 the optimum value of λ is 0.0014 which corresponds to an RMSE of 0.0734. Using this value of α we build a model based on the training subset and compare the predictions to the test set. In figure 8 we can see that the predictions are relatively good and provide an RMSE of 0.0936, that is, on the test set. The residual plot does not exhibit any specific trends which shows that there is no apparent bias that could interfere with our linear model. In fact, to improve the model we introduced interactions between some variables that appeared to have correlations between them. We changed the model matrix X in equation 2 and removed individual features $X_8 - X_{15}$ and $X_{21} - X_{25}$ which

were flagged as the correlated ones in figure 1 and created interaction terms within them. However, the model RMSE did not improve substantially and we therefore kept our model without interactions. We believe it is better to have a model without interactions since we do not know exactly what each of the feature represents.

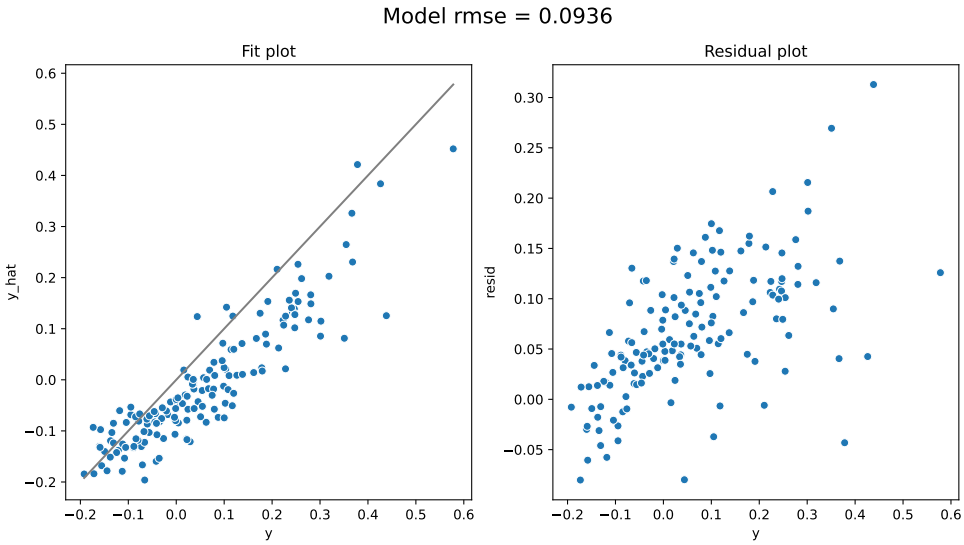


Figure 8: Lasso Predictions with $\lambda = 0.0014$ against true y values (left) and residual plot (right). These were evaluated on the test set which consists of 30% of the dataset.

Another possible improvement to the model was to check more than 80 points for λ to find a more accurate optimum value, but note that this would not significantly improve the RMSE of the model since all of the observations are scaled and the value of 0.0014 is already quite small to have an effect on points that are all scaled to have mean 0. We now move on to a different regression method.

Random Forrest

Random forests build on the idea of regression trees that partition the feature space into sets of rectangles and fit a model in each of the areas. The criterion we use for splitting the tree is the impurity measure of squared error. We can then extend this to ensemble methods that consider multiple trees which brings us to bagging and Random forests [1].

Random forests can be considered as bagged tree classifiers, but they minimize the correlation between the trees by selecting a random subset of features in order to find the optimal split. We then repeat this procedure and aggregate the prediction by each tree to assign the class label by the majority vote. Note that if the number of features we consider is the total number of features, we are essentially performing bagging.

We use the `RandomForestRegressor` function from `sklearn` to perform this analysis. There are a few hyperparameters we need to consider:

- The number of trees, where larger is generally better since we are averaging more trees which will yield a more robust ensemble and reduce overfitting. However, it is computationally expensive so we try to use the least amount of trees that gives stable results.

- The maximum number of features determines how random each tree is, so the smaller the number of features at each split the less overfitting we have; Muller et al [8] suggest that for regression we should use the total number of features in our dataset, however the lecture notes [1] point out that as the value of maximum features approaches the number of features, the individual trees will become more correlated as they consider more similar sets of features. For this reason we will use cross validation to find the optimum number, keeping in mind that the computational cost of the RF algorithm is linear in this parameter. The grid we will use is 15, 30, 39.

- The minimum number of samples required to be at a leaf node. This will have an effect in the smoothing of our regression model. The default value is 1 but we will use cross validation to find the optimum with a grid of 1, 2, 3.

- The maximum amount of allowed leaf nodes in each tree is also considered. If the number of leaf nodes is restricted to a low number, then the leaf nodes will contain a lot of examples and trees that overfit the data. We will use cross validation again in order to tune this with values 50, 80, 100.

The algorithm seemed to be stable with 100 trees so we kept it at that number because we wanted to lower the computational complexity as much as possible. Using grid search Cross validation with 5 folds we found that the best model was the one with 30 max features, 2 minimum number of samples at a leaf node and 80 maximum amount of allowed leaf nodes.

Using this model we can make predictions on the test set and compare them with the true values, just like we did with the lasso model. Figure 9 summarises the fit. We observe that the Random Forrest Regression predicts the true values of $\boldsymbol{y}$ better than the Lasso Regression model with an RMSE of 0.0662 as opposed to 0.0936. Qualitatively the fit also looks better since the points are closer to the $y = \hat{y}$ line.

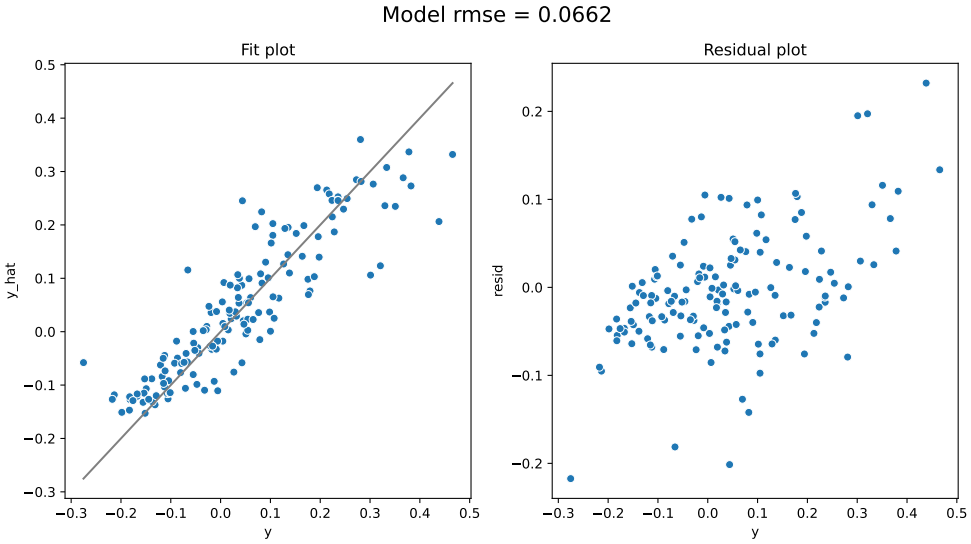


Figure 9: Random Forrest Predictions $\hat{y}$ against true y values (left) and residual plot (right). These were evaluated on the test set which consists of 30% of the dataset.

To interpret RF models we use the variable importance measures. Variable importance is an impurity based feature importance measure, and the importance of a feature is computed as the total reduction criterion brought by that feature. This is also known as Gini Importance [9]. This is a method available from the `RandomForestClassifier` function.

We calculate the variable importances in this way for the RF model and visualise them in figure 10. In this figure we have also plotted the absolute values of the Lasso coefficients from our Lasso Regression model. We can see that both models agree on the most important feature, X_{29} and the second most important feature X_{31}. The rest of the features are very close to zero for both RF and Lasso. Some coefficients of lasso are not exactly zero, like X_1, X_{11}, X_{23} which might be one of the reasons the model performs worse than RF. We will explore this further later on.

We could improve the RF model by including interactions between some correlated variables which might show some more important variables. We could also remove the variable that was clustered on its own in the Hierarchical Clustering and perform the same analysis and observe if there was an increase in the RMSE. We tried both methods but they did not improve the RMSE, instead they increased it by 0.01 in the test set.

To recommend a model for the prediction task we look at the RMSEs of both models aswell as the variable importance plots. Since RF produces the lowest RMSE and has more node impurities that are exactly zero which would favour sparse solutions, we believe RF is the optimal model. However, we are mindful of the fact that only two variables out of the

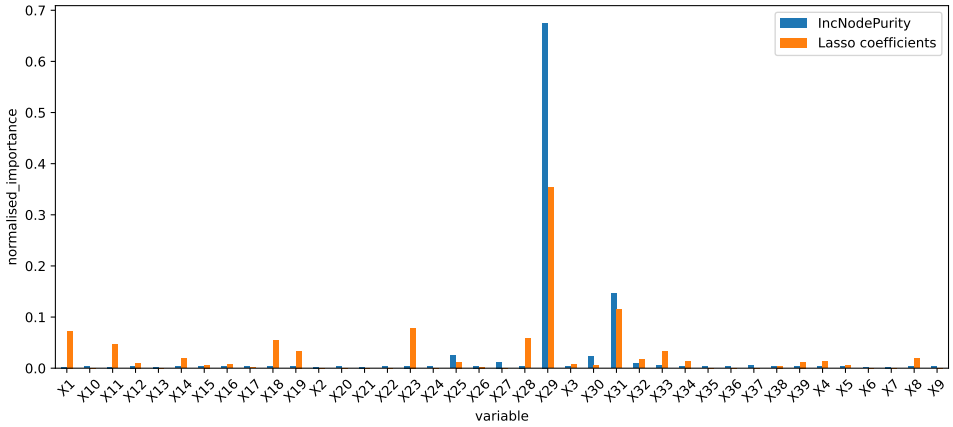


Figure 10: Absolute value of coeffisients from Lasso Regression (orange) and Variable importances of Random Forrest Models using the Gini importance (blue).

39 seem to play an important role in the regression, something that needs to be looked into. At this point we cannot explore further since we do not know what these variables are, and why they do not affect the log-ratio of the concentration of two proteins of interest.

Training/Test Split Variability

Now we will evaluate how the conclusions from our two regression models vary depending on the training/test split of the dataset. This is important and should be done in any model that we use in practice because it measures how much we have overfitted our data in our original model, and how reliable it is. Sometimes the split of the training dataset might have some patterns that the real-world data does not exhibit, and this is a method to check that, by randomizing the training/test split and trying multiple different splits.

We will need to use a nested cross validation procedure for Lasso Regression which is able to estimate the generalization error of the underlying model and its hyperparameter search. Inside the inner loop which will be executed by `GridSearchCV`, the score will be maximized by fitting a model to each training set and then it will be directly maximized in selecting the hyperparameters over the validation set. We then have the outer loop where the generalization error will be estimated by averaging test set scores over several dataset splits.

This was implemented in python adapted from [10]. We created smaller grid of values in the hyperparameter search to lower the computational cost. We have tried 20 different trials (with $K = 5$ for K-Fold CV) but this can easily be extended to a larger number of trials. Since there was not a lot of variation between those 20 trials we did not try more.

We included a plot which compares the differences between the RMSE scores between the nested method for CV and the non-nested one in figure 11. We can see that nested Cross-Validation produces lower RMSE in general than the non-nested one, something that is expected since in the non-nested CV "information may leak into the model and overfit the data" [10].

We can now evaluate how the conclusions from the previous question vary depending on the splits. In figure 12 the left plot shows the hyperparameter value chosen by the grid search in the CV. We chose to include this in the report because it gives us confidence that the hyperparameter value chosen by our initial analysis was not training split dependent. This is indeed true since for 18 out of the 20 trials the optimum λ was very close to the one chosen by our model. 2 trials seem to choose a value of λ slightly larger, but not by a significant amount meaning no further exploration would be required at this stage.

The middle plot of 12 shows that the distribution of RMSE scores achieved by the Lasso model are always very close to the 0.09 RMSE value we obtained from our single run before. This shows that the prediction performance on the test set was not split dependent.

The plot on the right summarises the frequency of the largest absolute values of Lasso coefficients (with values more than 0.1). This plot shows us how many times out of the 20 splits the coefficients appeared to be important.

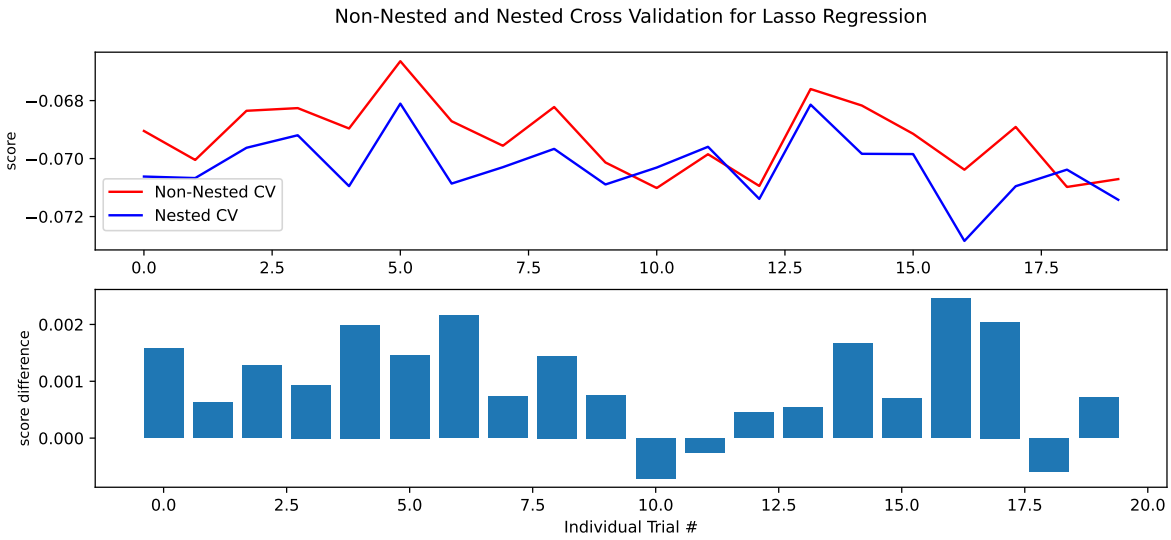


Figure 11: Comparison between Lasso Non-Nested and Nested CV. The comparison is made on the RMSE of each trial.

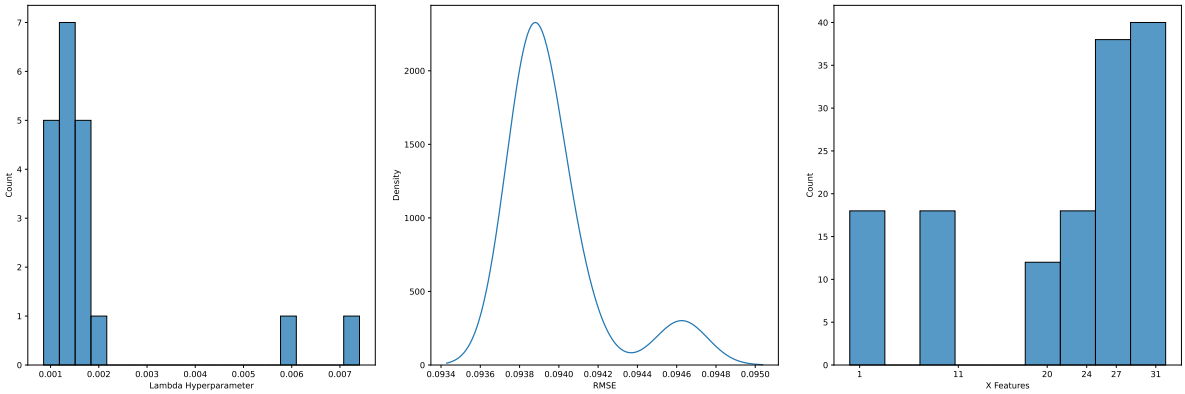


Figure 12: Frequency of hyperparameter λ values (left), RMSE score density (middle) and frequency of highest Lasso coefficients (absolute value more than 0.1) for the 20 different trials of nested CV for Lasso regression.

As we can see that X_{29} feature is one of the most frequent features with some candidates like X_1, X_{11}, X_{31} getting a lot of counts aswell. Note that if we increased the threshold to 0.5 only the X_{29} feature would be visible. This proves that variable importance also does not vary significantly for the Lasso regression depending on the training/test split.

To assess the variability of the Random Forrest Model depending on the splits we used the same procedure of the nested cross validation. Note that here we could also use the Out-Of-Bag prediction which takes advantage of the bootstrap sample and uses the unused candidates for validation. This decreases the computational time dramatically. However, for the 20 trials we wanted to test, the computational expense was not significantly large so we used the same nested CV procedure as before.

Figure 13 summarises the results of the RF Nested Cross validation. The plot on the right shows that again we have a sharp density of values for the RMSE score, and the distribution is centered around 0.06 which agrees with the RMSE predictive score we found in our initial analysis. The plot on the left assesses the variable importance in the same way as above, any features found to have a gini importance more than 0.1 are included in the histogram. The feature X_{29} is always the most important feature and we see some occurrences of variables X_{25}, X_{30}, X_{31} which were also found in the initial analysis in figure 10. Note that if we increase the threshold to 0.5 only X_{29} appears in the plot.

This concludes the evaluation of the training/test split importance. For both models the prediction performances were very close to the original analysis and their density was sharply peaked. This gives us confidence that the model we have built is not biased based on the samples it trains on and we can have confidence in its predictive performance. The variable

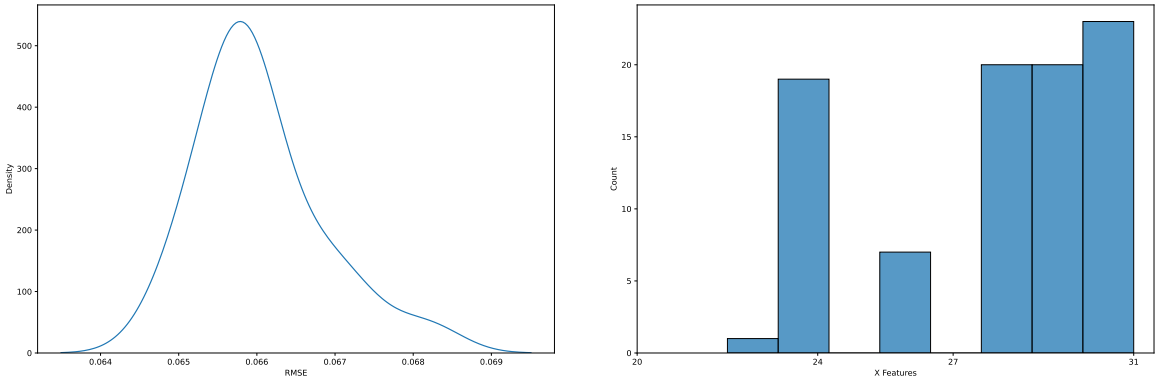


Figure 13: RMSE score density (left) and frequency of most important features using Gini Importance (right) for the 20 different trials of nested CV for Lasso regression.

importance exploration showed that some variables appear quite frequently to have high gini importance and absolute value of the lasso coefficients which suggests that we may consider to include more features in the model than just X_{29} and X_{31}. This is because irrespective of the training splits, some features had consistently values more than 0.1. This might suggest that the Lasso model might be able to generalize better since it finds more features that are important than RFs. Moreover, this evaluation eluded even more confidence that X_{29} is the most important feature of the dataset since it always had a variable importance of more than 0.5. We are confident that there was no overfitting in our analysis and the results we presented can be generalised.

Question 2

In this question we consider a simulated dataset containing the water temperature measured at different times and at various distances from the coast. An illustration of the data can be seen in figure 14. Note that we have centered the temperature so that it has mean 0 because in this analysis we will use a zero mean prior.

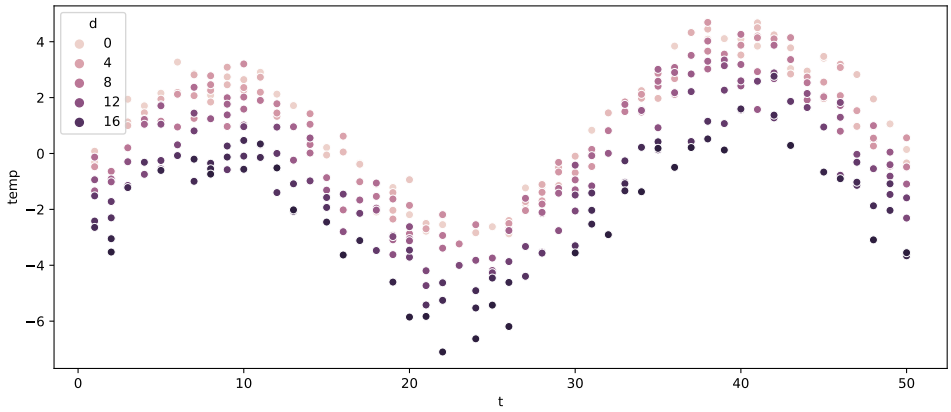


Figure 14: Centered Water Temperature (degrees Celsius) against time (days) for different distances (kilometers) away from the coast.

Water Temperature near the coast

We use a Gaussian process to construct predictive models of the water temperature as a function of time. A Gaussian process $\mathcal{GP}(m(\cdot), k(\cdot,\cdot))$ is fully determined by its mean function $m(\cdot)$ and positive semi-definite kernel function $k(\cdot,\cdot)$. In this analysis we will use an agnostic mean function, ie. $m(\cdot) = 0$, and we will consider different kernel functions. The objective is for a set of observations

$$y_i = f(\boldsymbol{x}_i) + \epsilon, \quad \epsilon \sim \mathcal{N}\left(0, \sigma_n^2\right) \tag{4}$$

to find a posterior distribution over functions $p(f|X, \boldsymbol{y})$ that explains the data. In our analysis y_i is the temperature and $\boldsymbol{x_i}$ is the time, and σ_n^2 the noise variance. Using Bayes Theorem and the property of conjugate priors we obtain the posterior of the function [1]:

$$f(\cdot) \mid X, \boldsymbol{y} \sim GP\left(m_{\text{post}}\left(\cdot\right), k_{\text{post}}\right) \tag{5}$$

where

$$\begin{aligned} m_{\text{post}}\left(\cdot\right) &= m(\cdot) + k(\cdot, X)\left(K + \sigma_n^2 I\right)^{-1}\left(\boldsymbol{y} - m(X)\right) \\ k_{\text{post}}\left(\cdot, \cdot\right) &= k(\cdot, \cdot) + k(\cdot, X)\left(K + \sigma_n^2 I\right)^{-1} K(X, \cdot) \end{aligned} \tag{6}$$

with $K = k(X, X)$. Then the predictive mean and variance of the Gaussian process for unknown inputs $\boldsymbol{x_*}$ will be:

$$\begin{aligned} \mathbb{E}\left[f\left(\boldsymbol{x}_*\right) \mid \boldsymbol{x}_*, X, \boldsymbol{y}\right] &= m\left(\boldsymbol{x}_*\right) = k\left(X, \boldsymbol{x}_*\right)^T\left(K + \sigma_n^2 I\right)^{-1}\boldsymbol{y} \\ \mathbb{V}\left[f\left(\boldsymbol{x}_*\right) \mid \boldsymbol{x}_*, X, \boldsymbol{y}\right] &= \sigma^2\left(\boldsymbol{x}_*\right) = k\left(\boldsymbol{x}_*, \boldsymbol{x}_*\right) - k\left(X, \boldsymbol{x}_*\right)^T\left(K + \sigma_n^2 I\right)^{-1} k\left(X, \boldsymbol{x}_*\right) \end{aligned} \tag{7}$$

Note that the covariance function encodes high-level structural assumptions about the latent function f which means we should try different kernels and decide on which one produces the best prediction based on Marginal Likelihood Maximization.

We will consider three different kernels for our analysis. The first one will be the Radial Basis Function (RBF) defined as:

$$k_{\text{RBF}}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \sigma_f^2 \exp\left(-\left(\boldsymbol{x}_i - \boldsymbol{x}_j\right)^\top\left(\boldsymbol{x}_i - \boldsymbol{x}_j\right)/\ell^2\right) \tag{8}$$

where σ_f is the amplitude of the latent function and ℓ is the length scale. We will also consider a Periodic Kernel function also known as Exponential Sine Squared Kernel defined as:

$$k_{\text{Periodic}}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \sigma_f^2 \exp\left(-\frac{2\sin^2\left(\pi d\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)/p\right)}{\ell^2}\right) \tag{9}$$

where p is the periodicity of the kernel, $d(\cdot, \cdot)$ is the Euclidean distance and σ_f, ℓ are as defined above. The third kernel we will consider will be a combination of the above, ie:

$$k_{\text{Mixed}}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = k_{\text{RBF}}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) * k_{\text{Periodic}}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) \tag{10}$$

All of the parameters of the covariance functions σ_f, ℓ, p are hyperparameters we need to tune. We also need to tune the likelihood parameter σ_n^2. We find good hyperparameters by maximizing the log marginal likelihood function (LML):

$$\log p\left(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}\right) = \mathcal{N}\left(\boldsymbol{y} \mid \boldsymbol{0}, \boldsymbol{K} + \sigma_n^2 \boldsymbol{I}\right) \tag{11}$$

where $\boldsymbol{\theta}$ are the hyperparameters we are tuning, $\boldsymbol{K}$ our kernel function and $\boldsymbol{I}$ the identity matrix. We learn the Gaussian Process hyperparameters by the optimization problem:

$$\boldsymbol{\theta}^* \in \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) \tag{12}$$

We implement this in Python by defining our Kernels as functions that accept the covariates $\boldsymbol{X}$ and their parameters, and then we calculate the log marginal likelihood based on a set of parameters for all 3 kernels. The grids of all hyperparameter values checked and the optimum value found by maximizing the log marginal likelihood are summarised in table 2. In calculating these we tried different values of σ_n^2 and $\sigma_n^2 = 1$ gave us the most stable results. Note that we have also used the value 30 for the period and did not perform a grid search to lower computational cost. The value 30 was chosen by qualitatively looking at figure 14 and noticing that the two maximum points of the wave (which is defined as the period of a wave) were about 30 time points apart.

As we can see from table 2 the highest LML is given by the RBF Kernel, followed marginally by the multiplication of RBF with Periodic and the Periodic gives the lowest

	RBF		Periodic		RBF $*$ Periodic	
σ_f^2	$(1, 5, 0.1)$	**3**	$(1, 10, 0.5)$	**4.5**	$(0, 3, 0.5)$	**1**
ℓ	$(1, 15, 1)$	**9**	$(1, 20, 1)$	**3**	$(3, 20, 1)$	**8**
σ_{f2}^2	-	-	-	-	$(0.1, 5, 0.5)$	**2.6**
ℓ_2	-	-	-	-	$(0, 10, 1)$	**9**
p	-	-	-	**30**	-	**30**
$\log p\left(\boldsymbol{y}\middle\vert\boldsymbol{X}, \boldsymbol{\theta}\right)$	-53.4		-61.7		-53.5	

Table 2: Summary of hyperparameter search for RBF, Periodic and RBF $*$ Periodic kernels. The values in the brackets indicate (`initial_value`, `final_value`, `step`) and the bold value next to them is the best value chosen by the highest log marginal likelihood. The last row represents the best log marginal likelihoods achieved for the Gaussian Process by the three kernels.

score. This is expected because the Periodic Kernel on its own is not able to capture the in-between interactions of points at the scale that RBF is able to capture.

We create plots of the data points and add on top the gaussian distributions with the predicted mean and variance using the best hyperparameter values from our grid search. These are summarised in figure 15.

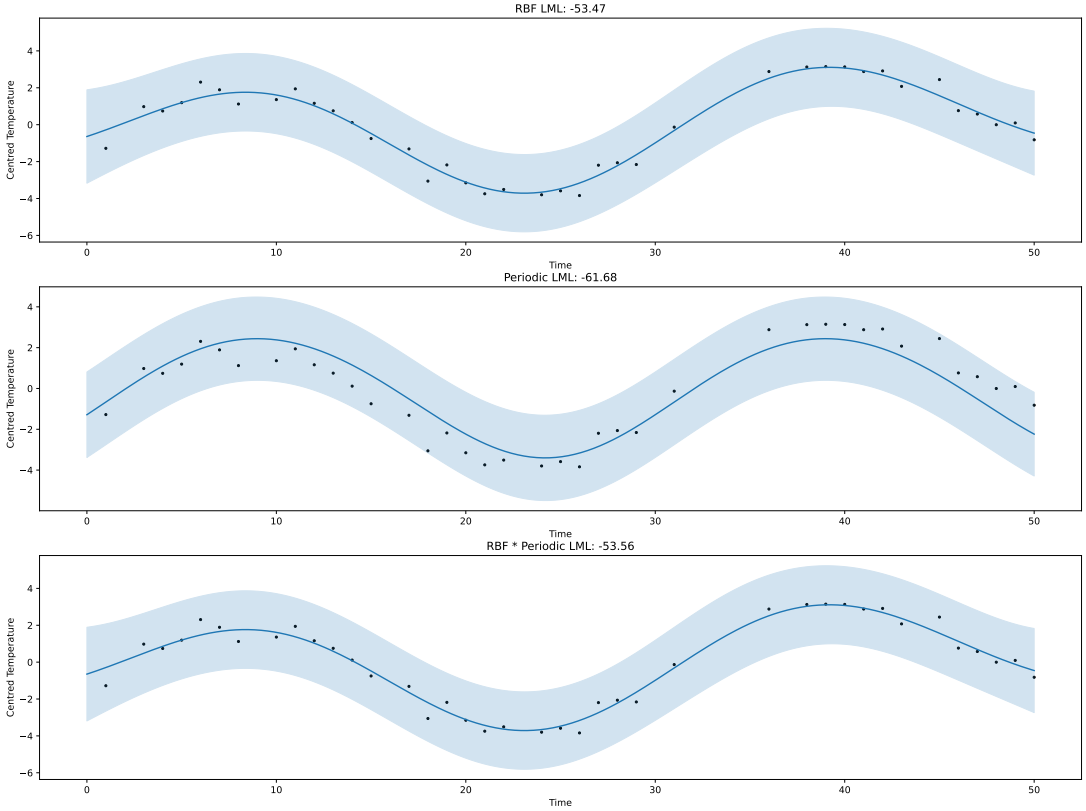


Figure 15: Centered Water Temperature (degrees Celsius) against time (days) for distance away from the coast equal to zero, with a fitted line of a Gaussian Process model for three different kernels: RBF, Periodic and RBF $*$ Periodic. The shaded area represents the 95% credible interval of the line predicting the variability of temperature against time.

We can observe that all three kernels are able to capture the variability of temperature and the 95 % confidence interval regions contain all of the points in the dataset. It is clear however, that although the Periodic Kernel captures the correct period, it misses most of the points and the line is above most points in the first half, and below most points in the second half. The other two kernels capture the variability very well. These results are reflected in the LMLs of the models, and therefore we choose the RBF kernel for our prediction task since it gives the lowest LML and it is simpler than the multiplication of RBF with Periodic.

For the prediction task we will predict the water temperature near the coast at $t = 35$ days. To do that we use equation 7 with $x_* = 35$. This gives an updated mean of 2.09 and

variance 1.57 for our gaussian distribution. Of course we need to add the original mean of the temperature to find the predicted temperature on the 35^{th} day of 13.21. The advantage of Bayesian inference methods over frequentist approaches is that we can assign a credible interval for our prediction. We compute the 95% credible interval using the above mean and variances we can plot our prediction with its credible interval in figure 16.

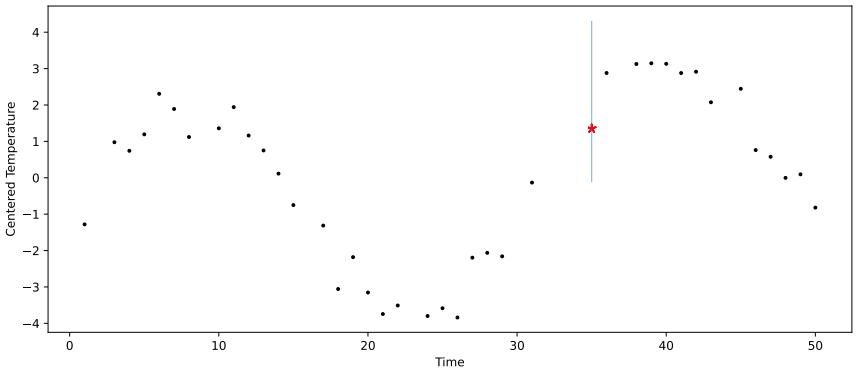


Figure 16: Centered Water Temperature (degrees Celsius) against time (days) for distance away from the coast equal to zero with the prediction of the temperature on the 35^{th} day (red star). The faded line represents the 95% confidence interval of the prediction.

We can also compute how likely it is to measure a temperature of 13 degrees or higher that day by using the cumulative distribution function of our model with mean 2.09 and variance 1.57 at the value $13 - 11.65$ where 11.65 is the mean of all temperatures. Subtracting the value of the CDF from 1 we get that there is a 91.8 % chance we measure a temperature higher than 13 degrees on the 35^{th} day.

Water Temperature at any distance

In this part we create a Gaussian Process model that is able to predict the water temperature as a function of both time and distance from the coast. For this, we are going to use `sklearn`'s function `GaussianProcessRegressor`. We use this function because it is more flexible with the inputs and the hyperparameter search is done within the function using an `fmin_l_bfgs_b` optimization method proposed by Byrd et al [11]. This optimisation method is fast and ideal for our analysis.

We try different kernels again to find the one that gives the largest LML. The mathematics behind the the GP model are the same as the last section, with X now being a 2-dimensional matrix, one dimension corresponding to distance and the other to time. We explore a grid of σ_n^2 values from 0.1 to 1 and find that the lowest LML value (accessible through the `log_marginal_likelihood_value_` attribute of the function). The optimum noise variance value is given by $\sigma_n^2 = 0.25$.

After trying various kernels including RBF, Periodic, Constant, Matern [12] we concluded that the RBF kernel multiplied by a constant kernel with value 1 gives the lowest observed LML for our model, with a value of -345. The multiplication of the constant kernel encapsulates the amplitude of the latent function which is not included by default in `sklearn`'s `RBF` function. The hyperparameters chosen by the optimization method were 1 for the amplitude (σ_f^2) and 1 for the lengthscale. These were chosen after a range of starting values so we believe the optimization function has reached a global minimum.

As before, we expected a kernel of the RBF form since our data have the same variation as before, even by including the distance to the coast. We can see a 3-dimensional scatterplot of the predicted values in figure 17 where we can qualitatively observe that the predicted values from the Gaussian Process follow the same pattern as the true values. This plot confirms that our model works well in predicting the true values of the temperature.

Using this model we can now make predictions for days in the future and get different temperatures for different distances from the coast. We expect that on day 55 the same periodic pattern will follow as the last 50 days, since the temperature cycle we have access to is just for 30 days. We also expect a decreasing temperature with respect to distance from the coast as it is easily visible from figure 14.

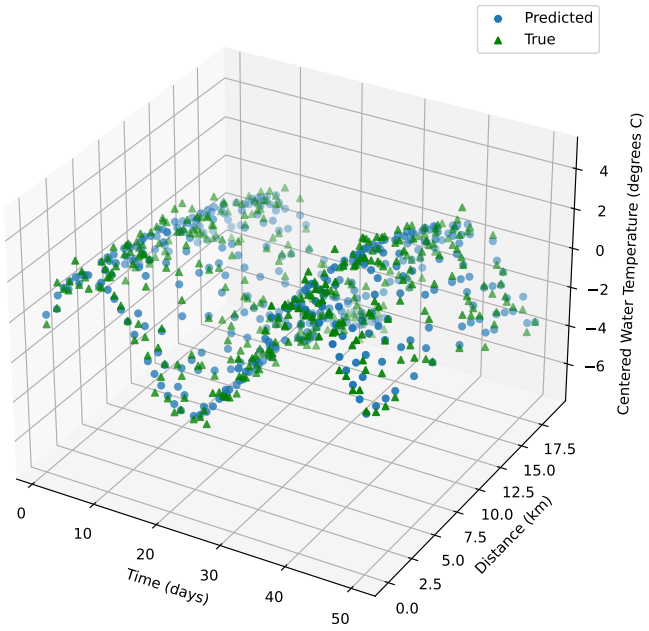


Figure 17: Centered Water Temperature (degrees Celsius) against time (days) against distance (km). Blue circles represent the predicted values from the GP model and green triangles represent the True values.

Using the predictive posterior distribution as defined in 7 for $\boldsymbol{x}_* = 55$ we create a function that calculates the temperature and takes as argument the distance from the coast. The resulting plot can be seen in figure 18.

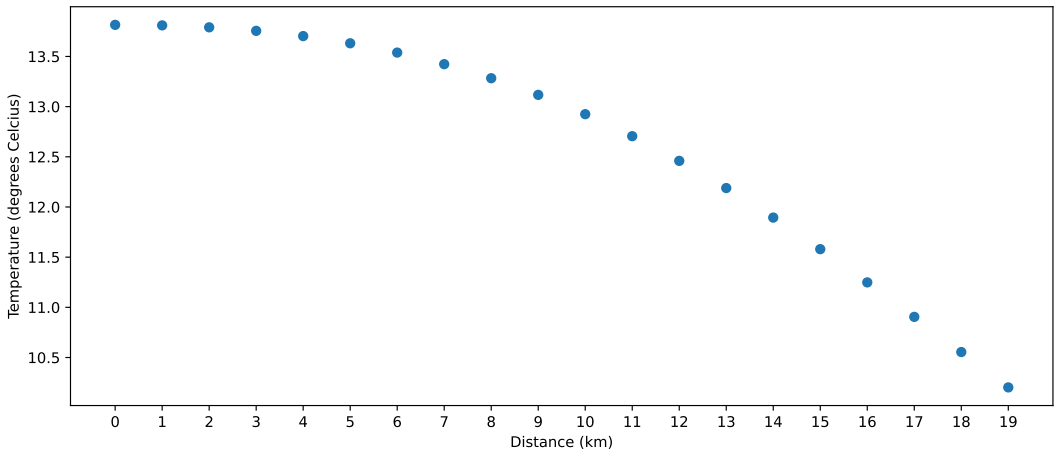


Figure 18: Water Temperature (degrees Celsius) against distance (km) at t=55.

As expected, the model is able to predict the decreasing trend of temperature with distance and it predicts a reasonable range of temperatures ($13.5 - 10.5$ degrees Celsius) at a time $t = 55$ that the model did not have access to originally. This plays well with the theory of gaussian processes, and Bayesian inference in general, where the update of the model is done when it has access to more information. This also gives a more interpretable way of understanding the underlying assumptions of the model and how the model becomes better the more information it has access to.

References

[1] Dr Sarah Filippi. Machine learning lecture notes. *Imperial College London*, pages 30–40, October 2021.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[3] Joe H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963. doi: 10.1080/01621459.1963.10500845. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500845.

[4] Sebastian Raschka. *Python Machine Learning*. Packt Publishing - ebooks Account, 2015. ISBN 1783555130. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1783555130.

[5] Wikipedia. K-means — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=K-means&oldid=282639335, 2022. [Online; accessed 11-March-2022].

[6] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 9780898716245.

[7] Mathieu Blondel. Kernel kmeans. https://gist.github.com/mblondel/6230787#file-kernel_kmeans-py, 2014.

[8] Andreas Müller and Sarah Guido. Introduction to machine learning with python: A guide for data scientists. 2016.

[9] Soren Havelund Welling. How to interpret mean decrease in accuracy and mean decrease gini in random forest models. Cross Validated. URL https://stats.stackexchange.com/q/197904. URL:https://stats.stackexchange.com/q/197904 (version: 2019-03-03).

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Nested versus non-nested cross-validation. https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html, 2022. [Online; accessed 06-March-2022].

[11] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal of Scientific Computing*, 16:1190–1208, September 1995. ISSN 1064-8275. doi: 10.1137/0916069.

[12] Wikipedia. Matérn covariance function — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Mat%C3%A9rn%20covariance%20function&oldid=1030171734, 2022. [Online; accessed 06-March-2022].

A Code for Question 1

```
# %%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import sklearn.manifold
import sklearn.preprocessing
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import Lasso
from sklearn import ensemble
from clusters_2dplot import plot2dclust
from sklearn.model_selection import cross_val_score
seed = 222
np.random.seed(seed)

plt.rcParams['figure.figsize'] = (12,5)
plt.rcParams['figure.dpi'] = 80

# %%
d = pd.read_csv('dataQ1.csv')
d.describe(include='all')

# %%
fig, ax = plt.subplots(figsize=(25,15))
sns.heatmap(d.corr(), annot=True, ax=ax)
plt.savefig('plots/corrplot.pdf')
plt.show()

# %%
X = d.drop(['y'], axis=1)

#we are going to explore features so we need to transpose.
X = X.T
sc = StandardScaler()
X = sc.fit_transform(X)

# %%
from clusters_2dplot import plot2dclust

all_cls = range(2,20,1)
plt2d = plot2dclust(X, tp = 'km')
all_siluets = plt2d.clustering(aggclust=True, range_n_clusters=all_cls)

# %%
plt.plot(all_cls, all_siluets)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.savefig('plots/silhouette.pdf')

# %%
from scipy.spatial.distance import pdist
from sklearn.cluster import AgglomerativeClustering
from sklearn.neighbors import NearestCentroid

import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'),
```

```
    show_leaf_counts=False, no_labels=False)
plt.title("Dendrogram")
plt.xlabel("X")
plt.ylabel("Eclidean Distances")
plt.savefig('plots/dendogram.pdf')
plt.show()

clusterer = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage=
y_predict = clusterer.fit_predict(X)
cluster_labels = clusterer.labels_

clf = NearestCentroid(metric='euclidean')
clf.fit(X, y_predict)

y = pdist(d)
print(y)

# %% [markdown]
# ### Part A 2.

# %%
# https://www.kaggle.com/minc33/visualizing-high-dimensional-clusters

d = pd.read_csv('dataQ1.csv')
X = d.drop(['y'], axis=1)
sc = StandardScaler()
X = pd.DataFrame(sc.fit_transform(X), columns=X.columns)

kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
clusters_linear = kmeans.predict(X)


# %%
plt2d = plot2dclust(X, clusters = clusters_linear, tp = 'km')
plt2d.make_plot(ker ='No Kernel')

# %%
plt2d.clustering(range_n_clusters=[2])

# %% [markdown]
# #### Now Kernel Kmeans

# %%
# https://gist.github.com/mblondel/6230787

from kernel_kmeans import KernelKMeans

for ker in ['poly', 'linear', 'polynomial', 'rbf' , 'laplacian', 'cosine', 'sign

    kkm = KernelKMeans(n_clusters=2, max_iter=100, kernel=ker, random_state=0, v

    d = pd.read_csv('dataQ1.csv')
    X = d.drop(['y'], axis=1)
    sc = StandardScaler()
    X = pd.DataFrame(sc.fit_transform(X), columns=X.columns)


    clusters = kkm.fit_predict(X)


# %%
#chose the best kernel: polynomial
kkm = KernelKMeans(n_clusters=2, max_iter=100, kernel='polynomial', random_state
```

```
d = pd.read_csv('dataQ1.csv')
X = d.drop(['y'], axis=1)
sc = StandardScaler()
X = pd.DataFrame(sc.fit_transform(X), columns=X.columns)
clusters_poly = kkm.fit_predict(X)

# %%
#percentage of times the points are in the same cluster
ll =[]
for n, i in enumerate(clusters_linear):
    if i != clusters_poly[n]:
        ll.append(True)
    else:
        ll.append(False)
print(sum(ll) / len(ll))

# %% [markdown]
# ### Part B 1.

# %%
def model_fit(m, X, y, plot = False):

    y_hat = m.predict(X)
    rmse = mean_squared_error(y, y_hat, squared=False)
    res = pd.DataFrame(
            data = {'y': y, 'y_hat': y_hat, 'resid': y - y_hat}
    )
    if plot:
            plt.figure(figsize=(12, 6))
            plt.subplot(121)
            sns.lineplot(x='y', y='y_hat', color="grey",
            data = pd.DataFrame(data={'y': [min(y),max(y)], 'y_hat': [min(y),ma
            sns.scatterplot(x='y', y='y_hat', data=res).set_title("Fit plot")
            plt.subplot(122)
            sns.scatterplot(x='y', y='resid', data=res).set_title("Residual plo
            plt.suptitle("Model rmse = " + str(round(rmse, 4)), fontsize=16)
            plt.savefig('plots/lasso.pdf')
            plt.show()
    return(rmse)

# %%
d = pd.read_csv('dataQ1.csv')
y = d['y']
X = d.drop(['y'], axis=1)

TEST_SIZE = 0.3

# train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=TEST_SIZE)

sc = StandardScaler()

# %%
alphas = np.linspace(0.0001, 0.02, num=80)
l_gs = GridSearchCV(
    make_pipeline(
        StandardScaler(),
        Lasso()
        ),
        param_grid={'lasso__alpha': alphas},
        cv=KFold(5, shuffle=True, random_state=1234),
        scoring="neg_root_mean_squared_error"
).fit(X_train, y_train)

```

```
# %%
print( "best alpha:", l_gs.best_params_['lasso__alpha'])
best_alpha = l_gs.best_params_['lasso__alpha']
print( "best rmse :", l_gs.best_score_ * -1)
print( "validation rmse:", model_fit(l_gs.best_estimator_, X_test, y_test, plot=

# %%
cv_res = pd.DataFrame(
    data = l_gs.cv_results_
).filter(
    regex = '(split[0-9]+|mean)_test_score'
).assign(
# Add the alphas as a column
    alpha = alphas
)
cv_res.update(
# Convert negative rmses to positive
-1 * cv_res.filter(regex = '_test_score')
)
sns.lineplot(x='alpha', y='mean_test_score', data=cv_res)
plt.xlabel('$\lambda$')
plt.ylabel('Negative RMSE')
# plt.savefig('plots/single_l.pdf')
plt.show()

# %%
m = make_pipeline(
    StandardScaler(),
    Lasso(alpha=best_alpha, fit_intercept = False)
).fit(X_train, y_train)

# %%
model_fit(m, X_test, y_test, plot=True)

# %%
all_coef_lasso = m.named_steps['lasso'].coef_

# %% [markdown]
# #### Random Forrest

# %%
d = pd.read_csv('dataQ1.csv')
y = d['y']
X = d.drop(['y'], axis=1)

TEST_SIZE = 0.3

# train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=TEST_SIZE)

# %%
# model selecion using cross-validation

tuned_parameters = {'n_estimators': [100],
                    'max_features': [15, 30, 39],
                    'min_samples_leaf': [1,2,3],
                    'max_leaf_nodes': [50, 80, 100]
                    }

clf = GridSearchCV(ensemble.RandomForestRegressor(oob_score=True), tuned_paramet
                   n_jobs=-1, verbose=1).fit(X_train, y_train)
clf.best_estimator_

# %%
```

```
model = clf.best_estimator_
model.fit(X_train, y_train)

# %%
yhat_oob = model.oob_prediction_
yhat = model.predict(X_train)

plt.scatter(y_train, yhat, color='red', marker='.', label='yhat')
plt.scatter(y_train, yhat_oob, color='teal', marker='.', label='yhat_oob')
plt.axline((0, 0), slope=1, color="black")
plt.xlabel('y')
plt.ylabel('prediction')
plt.legend()

# %%

def train_pred_mse(model, train_X, train_y, test_X, test_y):
    model.fit(train_X, train_y)
    pred_y = model.predict(test_X)
    return mean_squared_error(pred_y, test_y)


default_model = ensemble.RandomForestRegressor(random_state=1234)
default_mse = train_pred_mse(default_model, X_train, y_train, X_test, y_test)

tuned_mse = train_pred_mse(model, X_train, y_train, X_test, y_test)

print(f'Tuned model MSE: {round(tuned_mse, 5)}')
print(f'Default model MSE: {round(default_mse, 5)}')

# %%
model_fit(model, X_test, y_test, plot=True)

# %% [markdown]
# #### 2. Variable Importance - Random Forrests

# %%
# Feature importance based on mean decrease in impurity
from sklearn.inspection import permutation_importance
from scipy.stats import spearmanr

impurity_importances = model.feature_importances_

mse_importances = permutation_importance(
    model, X_test, y_test, n_repeats=10, random_state=42, n_jobs=2
).importances_mean

# %%
# Normalise
mse_importances = mse_importances / mse_importances.sum()
impurity_importances = impurity_importances / impurity_importances.sum()
lasso_importances = np.abs(all_coef_lasso)


correl = round(spearmanr(impurity_importances, lasso_importances).correlation, 4

df = pd.DataFrame.from_dict({
    'variable': d.columns[1:],
    'IncNodePurity': impurity_importances,
    'Lasso coefficients': np.abs(all_coef_lasso / np.abs(all_coef_lasso).sum())
})
df = df.set_index('variable')
df = df.sort_index()

```

```
df.plot.bar()
plt.xlabel('variable')
plt.ylabel('normalised_importance')
plt.xticks(rotation=45)
plt.savefig('plots/importances.pdf')
plt.show()

# %% [markdown]
# ### 2. train/test split importance

# %%
d = pd.read_csv('dataQ1.csv')
y = d['y'].values
X = d.drop(['y'], axis=1).values
TEST_SIZE = 0.3

# %%
#lasso

NUM_TRIALS = 20

# Arrays to store scores
non_nested_scores = np.zeros(NUM_TRIALS)
nested_scores = np.zeros(NUM_TRIALS)
all_best_alpha1 = []
all_rmse1 = []
all_coefs1 = []

# Loop for each trial
for i in range(NUM_TRIALS):

    # Choose cross-validation techniques for the inner and outer loops,
    # independently of the dataset.
    # E.g "GroupKFold", "LeaveOneOut", "LeaveOneGroupOut", etc.
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)

    # Non_nested parameter search and scoring
    clf = GridSearchCV(
            make_pipeline(
                StandardScaler(),
                Lasso()
                ),
                param_grid={'lasso__alpha': alphas},
                cv=inner_cv,
                scoring="neg_root_mean_squared_error"
        ).fit(X_train, y_train)
    non_nested_scores[i] = clf.best_score_

    # Nested CV with parameter optimization
    clf = GridSearchCV(
            make_pipeline(
                StandardScaler(),
                Lasso()
                ),
                param_grid={'lasso__alpha': alphas},
                cv=inner_cv,
                scoring="neg_root_mean_squared_error"
        ).fit(X_train, y_train)
    nested_score = cross_val_score(clf, X=X_train, y=y_train, cv=outer_cv)
    nested_scores[i] = nested_score.mean()

    best_alpha = clf.best_params_['lasso__alpha']
    m = make_pipeline(
```

```
    StandardScaler(),
    Lasso(alpha=best_alpha, fit_intercept = False)
    ).fit(X_train, y_train)
    rmse = model_fit(m, X_test, y_test, plot=False)
    all_coef_lasso = m.named_steps['lasso'].coef_
    all_best_alpha1.append(best_alpha)
    all_rmse1.append(rmse)
    all_coefs1.append(all_coef_lasso)

score_difference = non_nested_scores - nested_scores


# %%
print(
    "Average difference of {:6f} with std. dev. of {:6f}.".format(
        score_difference.mean(), score_difference.std()
    )
)

# Plot scores on each trial for nested and non-nested CV
plt.figure()
plt.tight_layout()
plt.subplot(211)
(non_nested_scores_line,) = plt.plot(non_nested_scores, color="r")
(nested_line,) = plt.plot(nested_scores, color="b")
plt.ylabel("score", fontsize="9")
plt.legend(
    [non_nested_scores_line, nested_line],
    ["Non-Nested CV", "Nested CV"],
    bbox_to_anchor=(0, 0.4, 0.5, 0),
)
plt.title(
    "Non-Nested and Nested Cross Validation for Lasso Regression",
    x=0.5,
    y=1.1,
    fontsize="12",
)

# Plot bar chart of the difference.
plt.subplot(212)
difference_plot = plt.bar(range(NUM_TRIALS), score_difference)
plt.xlabel("Individual Trial #")
plt.ylabel("score difference", fontsize="9")

plt.savefig('plots/nested.pdf')
plt.show()

# %%
list_of_important_features = []
another1 = []

for arr in all_coefs1:
    op = np.squeeze(np.where(abs(arr) > 0.01))
    list_of_important_features.append(op)
    if op.shape:
        another1 += list(op)

# %%
fig, axs = plt.subplots(1,3, figsize=(25,8))
plt.subplots_adjust(wspace=0.16)
g = sns.histplot(data=all_best_alpha1, ax=axs[0], stat='count', bins=NUM_TRIALS)
g.set_xlabel('Lambda Hyperparameter')
q = sns.kdeplot(all_rmse1, ax=axs[1])
q.set_xlabel('RMSE')
```

```
g = sns.histplot(another1, ax=axs[2])
g.set_xticks([1, 11, 20, 24, 27, 31])
g.set_xlabel('X Features')
fig.savefig('plots/lasso_all.pdf')

# %%
#rf

NUM_TRIALS = 20
# Arrays to store scores
non_nested_scores = np.zeros(NUM_TRIALS)
nested_scores = np.zeros(NUM_TRIALS)
all_rmse_RF1 = []
all_impurities1 = []

# Loop for each trial
for i in range(NUM_TRIALS):
    print(i)

    # Choose cross-validation techniques for the inner and outer loops,
    # independently of the dataset.
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)

    # Non_nested parameter search and scoring
    clf = GridSearchCV(
            make_pipeline(
                StandardScaler(),
                Lasso()
                ),
                param_grid={'lasso__alpha': alphas},
                cv=inner_cv,
                scoring="neg_root_mean_squared_error"
        ).fit(X_train, y_train)
    non_nested_scores[i] = clf.best_score_



    tuned_parameters = {'n_estimators': [100],
                'max_features': [30],
                'min_samples_leaf': [1,2,3],
                'max_leaf_nodes': [50, 80, 100]
                }

    # Nested CV with parameter optimization
    clf = GridSearchCV(ensemble.RandomForestRegressor(oob_score=True), tuned_par
                    n_jobs=-1, verbose=0).fit(X_train, y_train)

    nested_score = cross_val_score(clf, X=X_train, y=y_train, cv=outer_cv)
    nested_scores[i] = nested_score.mean()


    model = clf.best_estimator_
    model.fit(X_train, y_train)


    rmse = train_pred_mse(model, X_train, y_train, X_test, y_test)
    all_rmse_RF1.append(rmse)
    impurity_importances = model.feature_importances_
    all_impurities1.append(impurity_importances)

score_difference = non_nested_scores - nested_scores


```

```

# %%
another3 = []

for arr in all_impurities1_save:
    op = np.squeeze(np.where(abs(arr) > 0.01))
    if op.shape:
        another3 += list(op)


# %%
#created saved arrays so that I don't have to run it everytime,
#if not accessible just remove the '_save'
fig, axs = plt.subplots(1,2, figsize=(25,8))
plt.subplots_adjust(wspace=0.16)
q = sns.kdeplot(all_rmse_RF1_save, ax=axs[0])
q.set_xlabel('RMSE')
g = sns.histplot(another3, ax=axs[1])
g.set_xticks([20, 24, 27, 31])
g.set_xlabel('X Features')
fig.savefig('plots/RF_all.pdf')

# %%
```

```

##!!!## code is adapted from:  https://www.kaggle.com/minc33/visualizing-high-di


#Instructions for building the 2-D plot
import plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.neighbors import NearestCentroid
import matplotlib.cm as cm
import matplotlib.pyplot as plt




class plot2dclust():
    def __init__(self, X,tp, clusters = None):
        self.clusters = clusters
        self.tp = tp
        self.X = X

    def make_plot(self, ker):
        X = self.X
        clusters = self.clusters
        nameofcol = 'cluster_' + self.tp
        X[nameofcol] = clusters
        #Try PCA for visulization
        plotX = X
        plotX.columns = X.columns

        #PCA with two principal components
        pca_2d = PCA(n_components=2)

        #This DataFrame contains the two principal components that will be used
        #for the 2-D visualization mentioned above
```

```
        PCs_2d = pd.DataFrame(pca_2d.fit_transform(plotX.drop([nameofcol], axis=
        PCs_2d.columns = ["PC1_2d", "PC2_2d"]
        plotX = pd.concat([plotX,PCs_2d],axis=1, join='inner')
        cluster0 = plotX[plotX[nameofcol] == 0]
        cluster1 = plotX[plotX[nameofcol] == 1]
        #trace1 is for 'Cluster 0'
        trace1 = go.Scatter(
                            x = cluster0["PC1_2d"],
                            y = cluster0["PC2_2d"],
                            mode = "markers",
                            name = "Cluster 0",
                            marker = dict(color = 'rgba(255, 128, 255, 0.8)'),
                            text = None)
        #trace2 is for 'Cluster 1'
        trace2 = go.Scatter(
                            x = cluster1["PC1_2d"],
                            y = cluster1["PC2_2d"],
                            mode = "markers",
                            name = "Cluster 1",
                            marker = dict(color = 'rgba(255, 128, 2, 0.8)'),
                            text = None)
        data = [trace1, trace2]
        title = ""
        layout = dict(title = title,
                      xaxis= dict(title= 'PC1',ticklen= 5,zeroline= False),
                      yaxis= dict(title= 'PC2',ticklen= 5,zeroline= False)
                      )
        fig = dict(data = data, layout = layout)
        iplot(fig)
        # fig.write_image("plots/kmeans.pdf")
    def clustering(self, range_n_clusters = [2,3,5,8,12,15], aggclust = False):
        X = self.X
        siluets = []
        for n_clusters in range_n_clusters:
            # Create a subplot with 1 row and 2 columns
            if aggclust:
                # X = X.T
                clusterer = AgglomerativeClustering(n_clusters=n_clusters, linka
                y_predict = clusterer.fit_predict(X)
                cluster_labels = clusterer.labels_
            else:
                y_predict = self.clusters
                cluster_labels = self.clusters
            clf = NearestCentroid()
            clf.fit(X, y_predict)
            silhouette_avg = silhouette_score(X, cluster_labels)
            siluets.append(silhouette_avg)
            if (n_clusters == 2 or n_clusters == 3 or n_clusters == 5):
                print("For n_clusters =", n_clusters,
                      "The average silhouette_score is :", silhouette_avg)
                fig, (ax1) = plt.subplots(1, 1)
```

```
                fig.set_size_inches(15, 5)

                ax1.set_xlim([-0.1, 1])
                ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

                sample_silhouette_values = silhouette_samples(X, cluster_labels)

                y_lower = 10
                for i in range(n_clusters):
                    ith_cluster_silhouette_values = \
                        sample_silhouette_values[cluster_labels == i]

                    ith_cluster_silhouette_values.sort()

                    size_cluster_i = ith_cluster_silhouette_values.shape[0]
                    y_upper = y_lower + size_cluster_i

                    color = cm.nipy_spectral(float(i) / n_clusters)
                    ax1.fill_betweenx(np.arange(y_lower, y_upper),
                                      0, ith_cluster_silhouette_values,
                                      facecolor=color, edgecolor=color, alpha=0.7)
                    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
                    y_lower = y_upper + 10  # 10 for the 0 samples

                ax1.set_title("The silhouette plot for the various clusters.")
                ax1.set_xlabel("The silhouette coefficient values")
                ax1.set_ylabel("Cluster label")
                ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

        plt.savefig('plots/single_siluette.pdf')
        plt.show()
        return(siluets)
```

B Code for Question 2

```
# %%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
import sklearn.manifold
import sklearn.preprocessing
from sklearn.gaussian_process import GaussianProcessRegressor
from typing import List, Tuple
from scipy.stats import norm
from sklearn.gaussian_process.kernels import ConstantKernel

seed = 222
np.random.seed(seed)

plt.rcParams['figure.figsize'] = (12,5)
plt.rcParams['figure.dpi'] = 80

# %%
d = pd.read_csv('dataQ2.csv')
d

# %%
df = d[d['d'] == 0]
df.T

# %%
sns.scatterplot(df['t'], df['temp'])
```

```
30
31  # %%
32  # code adapted from tutorial 5
33
34  class Kernel:
35      """Base class for kernels"""
36
37      def __add__(self, kernel2):
38          return SumKernel([self, kernel2])
39
40      def __mul__(self, kernel2):
41          return ProductKernel([self, kernel2])
42
43
44  class SumKernel:
45      """Kernel to enable summation of kernels"""
46
47      def __init__(self, kernels: List[Kernel]) -> None:
48          self.kernels = kernels
49
50      def __call__(self, X: np.ndarray, X2: np.ndarray = None) -> np.ndarray:
51          return np.sum([k(X, X2) for k in self.kernels], axis=0)
52
53
54  class ProductKernel:
55      """Kernel to enable product of kernels"""
56
57      def __init__(self, kernels: List[Kernel]) -> None:
58          self.kernels = kernels
59
60      def __call__(self, X: np.ndarray, X2: np.ndarray = None) -> np.ndarray:
61          return np.prod([k(X, X2) for k in self.kernels], axis=0)
62
63
64  class RBF(Kernel):
65      def __init__(
66          self, sigma_f: np.float64 = 1.0, lengthscale: np.float64 = 1.0
67      ) -> None:
68          self.sigma_f = sigma_f
69          self.lengthscale = lengthscale
70
71      def __call__(self, X: np.ndarray, X2: np.ndarray = None) -> np.ndarray:
72          """
73          Calculate a kernel matrix using the RBF kernel
74
75          Args:
76          - X: matrix with shape n1 x 1
77          - X2: matrix with shape n2 x 1 or None (default), in which case X2=X
78
79          Returns: n1 x n2 kernel matrix
80          """
81
82          if X2 is None:
83              X2 = X
84
85          # we make use of broadcasting to compute the pairwise difference
86          # between each element of X and X2
87          diff = (X[:, None, 0] - X2[None, :, 0]) ** 2
88          K = self.sigma_f ** 2 * np.exp(-diff / (2 * self.lengthscale ** 2))
89          return K
90
91
92  class PeriodicKernel(Kernel):
93      def __init__(
```

```
        self,
        sigma_f: np.float64 = 1.0,
        lengthscale: np.float64 = 1.0,
        period: np.float64 = 1.0,
    ) -> None:
        self.sigma_f = sigma_f
        self.lengthscale = lengthscale
        self.period = period

    def __call__(self, X: np.ndarray, X2: np.ndarray = None) -> np.ndarray:
        """
        Calculate a kernel matrix using the periodic kernel function

        Args:
        - X: matrix with shape n1 x 1
        - X2: matrix with shape n2 x 1 or None (default), in which case X2=X

        Returns: n1 x n2 kernel matrix
        """

        if X2 is None:
            X2 = X

        # we make use of broadcasting to compute the pairwise difference
        # between each element of X and X2
        diff = np.abs(X[:, None, 0] - X2[None, :, 0])
        K = -2 * np.sin(np.pi * diff / self.period) ** 2
        K = self.sigma_f ** 2 * np.exp(K / self.lengthscale ** 2)
        return K

# %%
# format the data into matrices of the appropriate sizes
X = df["t"].values[:, None]
y = df["temp"].values[:, None]
mean_y = np.mean(y)
print(mean_y)
y -= mean_y

plt.scatter(X, y)

# %%
def fit_gp_posterior(
    X: np.ndarray, y: np.ndarray, X_star: np.ndarray, kernel: Kernel, sigma_n: f
) -> Tuple[np.ndarray]:
    """Returns the mean and variance of the GP posterior
    for data (X,y) and kernel function k_fn at test points X*
    sigma_n is the noise std
    """
    Kff = kernel(X)  # K(X,X)
    Kffs = kernel(X, X_star)  # K(X,X*)
    Kfsfs = kernel(X_star)  # K(X*,X*)

    # calculate posterior
    sigma_n_sq = sigma_n ** 2
    f_mean = Kffs.T @ np.linalg.solve(Kff + sigma_n_sq * np.eye(Kff.shape[0]), y
    f_cov = Kfsfs - Kffs.T @ np.linalg.solve(
        Kff + sigma_n_sq * np.eye(Kff.shape[0]), Kffs
    )

    return f_mean, f_cov


def log_marg_likelihood(
    X: np.ndarray, y: np.ndarray, kernel: Kernel, sigma_n: float
```

```
) -> float:
    Kff = kernel(X)  # K(X,X)
    A = Kff + sigma_n ** 2 * np.eye(Kff.shape[0])
    lml = -0.5 * (
        y.T @ np.linalg.solve(A, y)
        + np.log(np.linalg.det(A) + 1e-9)
        + X.shape[0] * np.log(2 * np.pi)
    )
    return lml[0, 0]

# %%
#RBF KERNEL
SIGMA_N = 1
#maybe u should go up to 8
sigmas = np.arange(1., 5., 0.1)
wavelengths = np.arange(1., 15.,1)
all_lml = []


for s in sigmas:
    for l in wavelengths:
        ker = RBF(sigma_f=s, lengthscale=l)
        lml = log_marg_likelihood(X, y, ker, SIGMA_N)
        all_lml.append((s, l, lml))

all_lml = pd.DataFrame(np.array(all_lml), columns=['s', 'l', 'lml'])

plt.plot(all_lml['lml'], '.')


# %%
best_s_RBF = all_lml[all_lml['lml'] ==np.max(all_lml['lml'])].values[0][0]
best_l_RBF = all_lml[all_lml['lml'] ==np.max(all_lml['lml'])].values[0][1]

# %%
#PERIODIC KERNEL
SIGMA_N = 1
sigma_ns = np.arange(0.5, 1, 0.05)
sigmas = np.arange(1., 10., 0.5)
wavelengths = np.arange(1., 20.,1)
all_lml = []


for s in sigmas:
    for l in wavelengths:
        ker = PeriodicKernel(sigma_f=s, lengthscale=l, period=30)
        lml = log_marg_likelihood(X, y, ker, SIGMA_N)
        all_lml.append((s, l, lml))

all_lml = pd.DataFrame(np.array(all_lml), columns=['s', 'l', 'lml'])

plt.plot(all_lml['lml'], '.')


# %%
best_s_PERIODIC = all_lml[all_lml['lml'] ==np.max(all_lml['lml'])].values[0][0]
best_l_PERIODIC = all_lml[all_lml['lml'] ==np.max(all_lml['lml'])].values[0][1]

# %%
#MIXED KERNEL
SIGMA_N = 1
sigmas1 = np.arange(0., 3., 0.5)
wavelengths1 = np.arange(3., 20.,1)
sigmas2 = np.arange(0.1, 5., 0.5)
```

```
wavelengths2 = np.arange(0., 10.,1)
all_lml = []


for s in sigmas1:
    for l in wavelengths1:
        for s2 in sigmas2:
            for l2 in wavelengths2:
                ker = RBF(sigma_f=s,
                    lengthscale=l) * PeriodicKernel(sigma_f=s2,
                    lengthscale=l2, period=30)

                lml = log_marg_likelihood(X, y, ker, SIGMA_N)
                all_lml.append((s, l, s2, l2, lml))

all_lml = pd.DataFrame(np.array(all_lml),
    columns=['s1', 'l1', 's2', 'l2' ,'lml'])

plt.plot(all_lml['lml'], '.')

# %%
best_s1_MIXED = all_lml[all_lml['lml'] ==np.max(all_lml['lml'])].values[0][0]
best_l1_MIXED = all_lml[all_lml['lml'] ==np.max(all_lml['lml'])].values[0][1]
best_s2_MIXED = all_lml[all_lml['lml'] ==np.max(all_lml['lml'])].values[0][2]
best_l2_MIXED = all_lml[all_lml['lml'] ==np.max(all_lml['lml'])].values[0][3]

# %%
print('RBF: ', best_s_RBF, best_l_RBF )
print('Periodic: ', best_s_PERIODIC, best_l_PERIODIC)
print('Mixed: ', best_s1_MIXED, best_l1_MIXED, best_s2_MIXED, best_l2_MIXED)

# %%
kernel_functions = [
    # RBF kernel
    RBF(sigma_f=best_s_RBF, lengthscale=best_l_RBF),
    # periodic kernel
    PeriodicKernel(sigma_f=best_s_PERIODIC, lengthscale=best_l_PERIODIC, period=
    # RBF x Periodic
    RBF(sigma_f=best_s1_MIXED, lengthscale=best_l1_MIXED)
    * PeriodicKernel(sigma_f=best_s2_MIXED, lengthscale=best_l2_MIXED, period=30
]

# %%
gp_posteriors = [
    fit_gp_posterior(X, y, np.linspace(0, 50, 300)[:, None], kernel, SIGMA_N)
    for kernel in kernel_functions
]

# %%
all_lmls = [log_marg_likelihood(X, y, kernel, SIGMA_N)
    for kernel in kernel_functions]
all_lmls

# %%
titles = ["RBF", "Periodic", "RBF * Periodic"]
x_plot = np.linspace(0, 50, 300)[:, None]
fig, axs = plt.subplots(3, 1, figsize=(20, 15))
for i, (f_mean, f_cov) in enumerate(gp_posteriors):
    lml = all_lmls[i]
    f_var = np.diag(f_cov)[:, None]
    axs[i].plot(x_plot, f_mean, color="C0")
    axs[i].scatter(X, y, color="black", s=5)
    axs[i].set_xlabel("Time")
    axs[i].set_ylabel("Centred Temperature")
```

```
    axs[i].set_title(titles[i] + f" LML: {np.round(lml, 2)}")

    # plot the 95% predictive distribution (f_var + likelihood_variance)
    axs[i].fill_between(
        x_plot[:, 0],
        (f_mean - 1.96 * np.sqrt(f_var + SIGMA_N ** 2))[:, 0],
        (f_mean + 1.96 * np.sqrt(f_var + SIGMA_N ** 2))[:, 0],
        color="C0",
        alpha=0.2,
    )
fig.savefig('plots/all_post.pdf')

# %%
Xs = np.array([[35]])
y_13 = 13 - mean_y
gp_posteriors_with_pred = [
    fit_gp_posterior(X, y, Xs, kernel, SIGMA_N) for kernel in kernel_functions
]

# %%
def compute_tail(mean, var):
    return 1 - norm(loc=mean, scale=np.sqrt(var)).cdf(y_13)[0][0]

# %%
probs = [compute_tail(preds[0], preds[1]) for preds
            in gp_posteriors_with_pred]
probs


# %%
fig, axs = plt.subplots(3, 1, figsize=(20, 15))
for i, (f_mean, f_cov) in enumerate(gp_posteriors_with_pred):
    lml = all_lmls[i]
    f_var = np.diag(f_cov)[:, None]
    # print(f_cov)
    axs[i].plot(Xs, f_mean, color="C0")
    axs[i].scatter(X, y, color="black", s=5)
    axs[i].scatter(35, y_13, color="red", s=50, marker=(5, 2))
    axs[i].set_xlabel("Months after January 1960")
    axs[i].set_ylabel("Centred Temperature")
    axs[i].set_title(titles[i])

    # plot the 95% predictive distribution (f_var + likelihood_variance)
    axs[i].fill_between(
        Xs[:, 0],
        (f_mean - 1.96 * np.sqrt(f_var + SIGMA_N ** 2))[:, 0],
        (f_mean + 1.96 * np.sqrt(f_var + SIGMA_N ** 2))[:, 0],
        color="C0",
        alpha=0.2,
    )
    # axs[i].set_xlim(578, 620)

# %%
my_mean = gp_posteriors_with_pred[0][0]
my_cov = gp_posteriors_with_pred[0][1]
f_var = np.diag(my_cov)[:, None]
plt.plot(Xs, my_mean, color="C0")
plt.scatter(X, y, color="black", s=5)
plt.scatter(35, y_13, color="red", s=50, marker=(5, 2))
plt.fill_between(
        Xs[:, 0],
        (my_mean - 1.96 * np.sqrt(my_cov + SIGMA_N ** 2))[:, 0],
        (my_mean + 1.96 * np.sqrt(my_cov + SIGMA_N ** 2))[:, 0],
        color="C0",
```

```
        alpha=0.5,
    )
plt.xlabel('Time')
plt.ylabel('Centered Temperature')
plt.savefig('plots/y13.pdf')
plt.show()

# %% [markdown]
# ### Part 2

# %%
sns.scatterplot(d['t'], d['temp'], hue=d['d'])
plt.savefig('plots/tempsctr.pdf')
plt.show()

# %%
d = pd.read_csv('dataQ2.csv')
X = d[['t', 'd']].to_numpy()
y = d['temp'].to_numpy()[:, None]
mean_y = np.mean(y)
print(mean_y)
y -= mean_y
print(X.shape)
print(y.shape)

# %%
kern = ConstantKernel(constant_value=1) * sklearn.gaussian_process.kernels.RBF()

alphas = [0.1, 0.2, 0.25, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 , 1.]
for a in alphas:
    gpr = GaussianProcessRegressor(kernel=kern, random_state=0,
        alpha=a, normalize_y=False).fit(X,y)
    print('for a' , a, 'LML: ', gpr.log_marginal_likelihood_value_)


# %%
# kern = sklearn.gaussian_process.kernels.RBF()
kern = ConstantKernel(constant_value=1) * sklearn.gaussian_process.kernels.RBF()
gpr = GaussianProcessRegressor(kernel=kern, random_state=0,
 alpha=0.25, normalize_y=False).fit(X,y)
gpr.get_params

# %%
y_pred, std = gpr.predict(X, return_std=True)

# %%
#code for 3d plot

# Fixing random state for reproducibility

fig = plt.figure(figsize= (18,8))
ax = fig.add_subplot(projection='3d')

n = 100

# For each set of style and range settings, plot n random points in the box
# defined by x in [23, 32], y in [0, 100], z in [zlow, zhigh].
# for m, zlow, zhigh in [('o', -50, -25), ('^', -30, -5)]:
xs = X[:,0]
ys = X[:,1]
zs = y_pred
ax.scatter(xs, ys, zs, marker= 'o', label= 'Predicted')
ax.scatter(xs, ys, y, marker='^', color = 'green', label= 'True')
# ax.plot_surface(xs, ys, zs)
```

```
414 |
415 |
416 | ax.set_xlabel('Time (days)')
417 | ax.set_ylabel('Distance (km)')
418 | ax.set_zlabel('Centered Water Temperature (degrees C)')
419 | ax.legend()
420 |
421 | plt.savefig('plots/3d.pdf')
422 | plt.show()
423 |
424 |
425 | # %%
426 | def calc_post(model, d_val):
427 |     X_star = np.array([55 - mean_y, d_val])[:, None].T
428 |     y_star, std2 = model.predict(X_star, return_std=True)
429 |     # print(y_star[0][0])
430 |     return y_star[0][0], std2
431 |
432 |
433 | # %%
434 | ds = np.arange(0,20,1)
435 | y_all_stars =[]
436 |
437 | for dval in ds:
438 |     y_stars, stds = calc_post(gpr, dval)
439 |     y_all_stars.append(y_stars)
440 |
441 | plt.scatter(ds, np.array(y_all_stars) + mean_y)
442 | plt.xticks(ds)
443 | plt.xlabel('Distance (km)')
444 | plt.ylabel('Temperature (degrees Celcius)')
445 | plt.savefig('plots/last.pdf')
446 | plt.show()
447 |
448 | # %%
```