1 Question 1

The data presented contains 569 observations with 30 continuous features and one binary categorical response. The binary categorical response pertains to the classification of breast tumours, there are 357 malignant and 212 benign observations in the data. We are tasked with reducing the dimensionality of the 30 continuous features using a method known as Principal Component Analysis (PCA). This principal component analysis results in column vectors, known as principal components, which are orthogonal linear combinations of the observed features. These principal components are constructed using the eigen-vectors of the covariance matrix of the given data-set and represent a projected version of the data. The advantage of PCA is that it maximises the sample variance of the projected data while minimising the average cost. The average cost is defined as the mean squared distance between the original data points and their projection. It can be shown that the eigen-vector which induces the 1-dimensional projection with the largest sample variance corresponds to the largest eigen-value. The eigen-vector which induces the 1-dimensional projection with the next largest sample variance corresponds to the next largest eigen-value. Before proceeding, it should be noted that PCA is not scale invariant and therefore data should be centred and standardised. This will ensure that the variance of each feature is equal to 1, i.e. the entries on the diagonal of the covariance matrix should be equal to 1.

The first k principal components can be constructed on data, $X \in \mathbb{R}^{n \times p}$, using the following method:

1. Standardise the data-set.
2. Construct the covariance matrix of the standardised data. This will be a $p \times p$ matrix.
3. Calculate the eigen-values and eigen-vectors of the covariance matrix using a suitable method. As the covariance matrix is positive semi definite, the eigen-values should be strictly non-negative.
4. Select the k largest eigen-values and their corresponding eigen-vectors.
5. Construct a $p \times k$ matrix, S, from the eigen-vectors, where each column corresponds to an eigen-vector.
6. To construct the projection, $V \in \mathbb{R}^{n \times k}$, set $V = XS$. V will represent a k-dimensional projection of the data as required. The columns of V are the k principal components.

The proportion of variance in X captured by the projection V can be calculated as follows:

$$\text{Proportion of variance} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i} \tag{1}$$

where λ_i denotes the ith largest eigen-value of the covariance matrix.

Using Eq. (1), it can be shown that to capture 90%, 95%, and 99% of the variation requires 7, 10, and 17 principal components respectively.

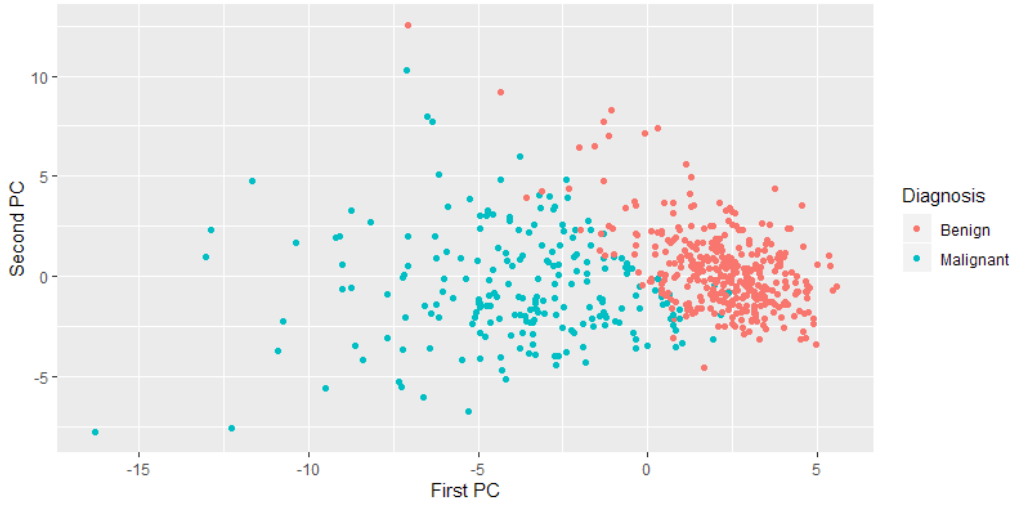


Figure 1: Plot of first two principal components

In Fig. 1, the data has been visualised by projecting its covariance structure to 2-dimensions. The first two principal components capture 63% of the variance in the data. There appears to be a clear relationship between the first principal component and the diagnosis, with higher values indicating benign tumours. We are next asked to complete two unsupervised clustering techniques known as hierarchical and k-means clustering and to evaluate their performance in light of the diagnosis.

Hierarchical clustering relies on a choice of distance metric to construct a similarity matrix, and a linkage function to construct clusters. We shall use an agglomerative clustering algorithm of the following form:

1. Calculate the similarity matrix using the chosen distance metric.
2. Assign each point to a unique cluster.
3. Using the similarity matrix, calculate cluster similarity using the chosen linkage function.
4. Join the two most similar clusters.
5. Repeat step 2-4 until one cluster remains.

There are many options for both the distance metric and linkage functions. Any valid distance metric can be used and a metric is considered valid if it satisfies the following:

1. $d(x, y) \geq 0$ and $d(x, y) = 0 \iff x = y$.
2. $d(x, y) = d(y, x)$
3. $d(x, z) \leq d(x, y) + d(y, z)$

We shall consider two distance metrics and two linkage functions. The distance metrics are the well-known Euclidean, and Manhattan distance metrics denoted as d_E and d_M respectively.

These are defined as:

$$d_E(x,y) = \sqrt{\sum_{i=1}^{p}(x_i - y_i)^2}$$

$$d_M(x,y) = \sum_{i=1}^{p}|x_i - y_i|$$

Two similarity matrices will be constructed using these distance metrics. Then hierarchical clustering will be performed on both using 2 different linkage functions. Linkage functions utilise different criteria to evaluate similarity between sets or, in this case, clusters. The linkage functions which will be used are known as the average, and complete linkage functions. It should be noted that the single linkage function is another well-known linkage function. However, it has a tendency to produce an effect known as chaining. This chaining is caused by the form of the linkage function where clusters are joined based on the minimum distance between them. This chaining effect is usually considered undesirable as the clusters produced are spread out. The 2 linkage functions we shall consider tend to avoid this. Given two clusters, A and B, which for this purpose can be considered two sets of points, are defined as follows:

$$\text{Complete: } d(A,B) = \max_{x\in A, y\in B} d'(x,y)$$

$$\text{Average: } d(A,B) = \frac{1}{|A|\,|B|}\sum_{x\in A}\sum_{y\in B} d'(x,y)$$

Where $d'(x,y)$ is the distance metric of choice, and $|A|$ denotes the cardinality of the set A.

A common method to examine hierarchical clustering solutions is using a dendrogram. A dendrogram is a plot which contains a tree-like structure which shows the order in which clusters were joined. To convert a dendrogram to a clustering solution, select the number of clusters, k, required. Select the height at which a horizontal line will cross exactly k vertical lines in the dendrogram. This will create k disjoint tree structures, with each representing a cluster. However, dendrograms can be a very dense representation of the data, particularly as the tree structure grows. Therefore, the plots can be difficult to decipher and so will contained in the appendix rather than the main body of this report.

Hierarchical clustering is usually employed as an exploratory data analysis technique and there is no predefined method for choosing the optimal number of clusters in a clustering solution. It appears that average linkage fails to capture reasonably sized clusters with either distance metric. This type of clustering solution may be very useful in identifying outliers, however will not produce the clusters required for diagnosis. The complete linkage appears to capture two reasonably sized clusters under both distance metrics. Using the Manhattan distance, the algorithm successfully identifies two distinct clusters with a cut at a height in the range of 100-110. Under the euclidean distance, it appears that a cut at just below 20 on the y-axis would produce 4 clusters. These would consist of two smaller clusters, potentially of outliers and two larger clusters. For the purposes of diagnosis, we shall continue with the combination of Manhattan distance and complete linkage. This is due to its ability to discern two distinct and reasonably sized clusters at a high level. This decision is made on the basis of the binary classification of tumors as either malignant or benign. This decision is highly susceptible to confirmation bias. There may, in fact, be many sub-types of tumours which are being detected by the other combinations. However, without further domain knowledge this decision cannot be made.

The K-means clustering algorithm is a clustering algorithm which aims to minimise the Total

within cluster Sum of Squares (TSS) which is defined as:

$$\text{TSS} = \sum_{i=1}^{N} \sum_{k=1}^{K} z_{ik} \|x_i - \mu_k\|_2^2$$

Where z_{ik} is an indicator function taking value 1 if the ith point is a member of cluster k and 0 otherwise. μ_k is the mean value of members of cluster k, and $\|\cdot\|_2$ indicates the L_2 norm.

The K-means algorithm is as follows:

1. Initialise K random centres and set μ_k.
2. Re-assign data points to clusters such that TSS is minimized.
3. Recalculate μ_k based on the new cluster assignment.
4. Repeat steps 2-3 until convergence.

It should be noted that due to the random initialisation, the algorithm is started multiple times from different locations to prevent convergence to local minima. In this case the algorithm is initialised 50 times. Unlike hierarchical clustering, due to K-means dependence on TSS for cluster assignment, there exists some predefined methods to determine the "optimal" number of clusters. The most common method is called the elbow method. This method relies on a plot of the number of clusters vs TSS and choosing the number of clusters at which a "statistical elbow" is seen in the plot. A statistical elbow is a point in the plot where TSS most rapidly decreases before the number of clusters when compared to the decrease after the number of clusters. In this case, no statistical elbow is present. An alternative method is the use of average silhouette value seen across clusters. Silhouette value is calculated as:

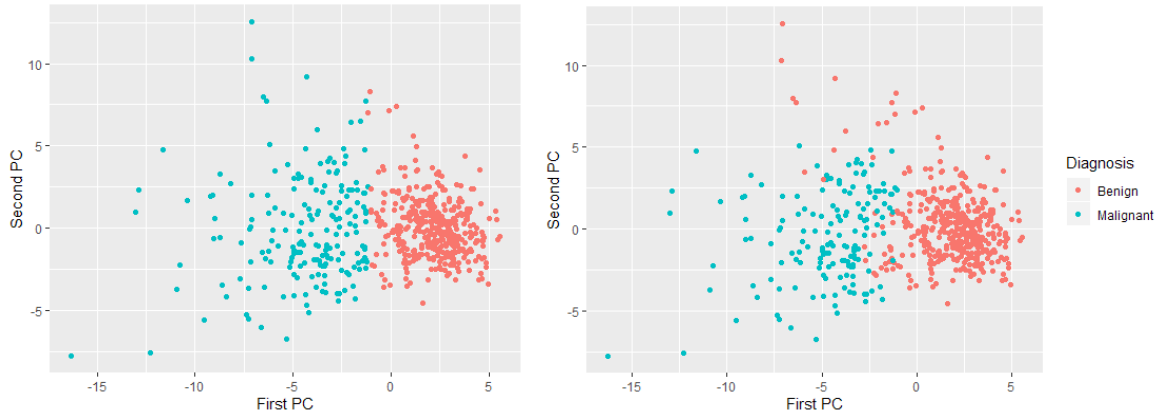
$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

where $a(i)$ is the average distance from observation i to the other members of the same cluster, and $b(i)$ is the minimum average distance from observation i to members of the other clusters. Silhouette values range from -1 to 1, with 1 indicating a point is well clustered and -1 indicating a point would be better clustered in a different cluster. A value of 0 indicates a point is along a boundary and could belong to either cluster. Therefore, and ideal clustering would have an average silhouette value of 1. Calculating the average silhouette value for different values of K.

Number of Clusters	2	3	4	5
Avg. Silhouette value	0.5185	0.4915	0.4330	0.2583

Table 1: Number of Clusters vs Avg. Sil. Value

As can be seen in Table 1, the number of clusters which maximises the average silhouette value is 2. Therefore, we will proceed with a 2 cluster model as expected. It should be noted that silhouette values can also be used with hierarchical clustering. However in this case, a qualitative approach was more appropriate due to the shape of the dendrograms.



(a) K-means Clustering Diagnosis (b) Hierarchical Clustering Diagnosis

Figure 2

Examining Fig. 2, we can see the distinct difference in cluster shape between the clustering methods. The dependence of K-means clustering on the L_2 norm means the clusters are spherical in shape. This leads to the near linear boundary between the benign and malignant tumours. Hierarchical clustering is more successful in clustering the values with a large second principal component. We will now compare both of the clustering solutions to the true diagnosis using confusion matrices.

	K-means		Hierarchical	
Predicted \Actual	Benign	Malignant	Benign	Malignant
Benign	343	37	348	64
Malignant	14	175	9	148
Specificity	0.8255		0.6981	

Table 2: Confusion Matrices

Looking at Table 2, we see that K-means outperforms hierarchical clustering with a specificity of 83% and 70% respectively. Specificity is the chosen metric for comparison as mis-classifying a malignant tumor has severe consequences when compared to mis-classifying a benign tumor. The non linear boundary seen for hierarchical clustering does not outperform the simple, near linear boundary of K-means. However, neither clustering solutions performs to a standard which would make them suitable for diagnosis purposes.

2 Question 2

We have been presented with a subset of the Million Song data-set. This subset contains 90 continuous features on 12,523 songs from 1922 to 2011. We are asked to fit two regression models to the data with the year in which a song was released as the target variable. The two regression models are a random forest and a neural network. We first examine the target variable, Year.

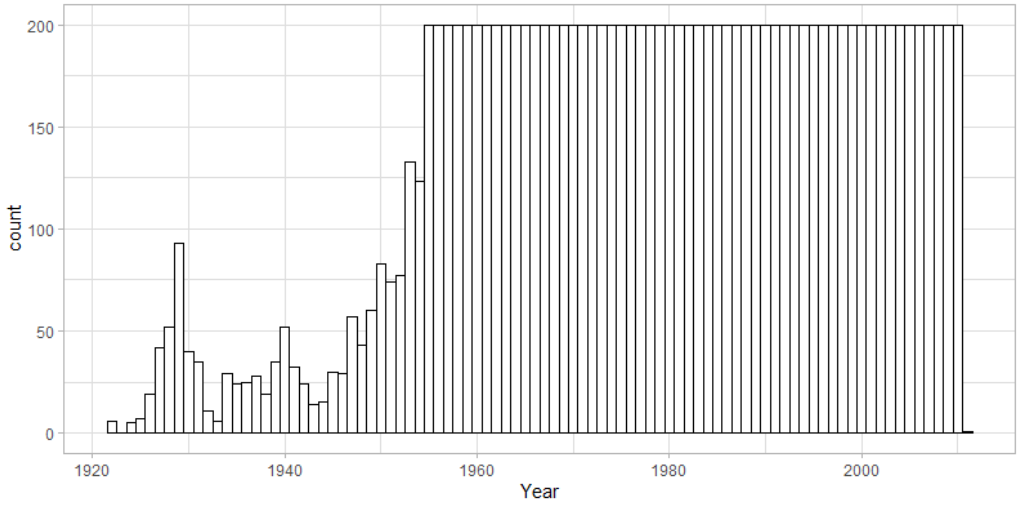


Figure 3: Histogram of song year

Looking at Fig. 3, it is clear that our data-set is unbalanced. There is a clear bias towards songs which were released after the mid 1950's. The number of songs contained in this subset has been limited to 200 per year. We see this limit is enforced for every year after the mid 1950's, with the exception of 2011. There is only one song from 2011 in this subset. To remove this bias from the response would be a difficult task as usual methods employed, such as under sampling, are not appropriate. This is due to some years containing very low numbers of songs and so under sampling would reduce the size of the training set drastically. Therefore, we shall continue with the full data-set in its current form and we can expect any model fit to have a much lower level of accuracy for years which are under represented in the data-set. It should also be noted that the year is recorded as an integer for each song whereas a regression model will return a float as a prediction.

The 90 continuous features, p, represent information about each song on a range of metrics such as timbre, timings, and tempo. These continuous features are of various magnitudes, and will need to be standardised for the neural network as it is not scale invariant. However, as random forest is scale invariant, we shall not standardise the data for that model.

Ensemble methods combine many weak learners to form a more accurate and robust model. The ensemble model's performance is dependent on the independence or diversity of the weak learners in the ensemble. A random forest regression model is an ensemble model comprised of many regression decision trees. Decision trees are a simple classification or regression technique which utilises binary splits on individual features. Decision trees consist of branches and the point at which a split occurs along a branch is called a node. The node at the end of a branch is called a leaf, or terminal node, which consists of every point that terminates in that branch. For regression, the value assigned to a leaf is the average value of the members of the leaf. Each feature is considered at each split and the optimal split will be the split which minimises the cost function [1]. Regression decision trees are grown using the sum of square errors as a cost function. Trees can be pruned using multiple hyper-parameters. Pruning trees ensures they do not over-fit to the data and maintain their diversity. Examples of pruning criteria include the minimum number of data points allowed in a terminal node, or the maximum number of nodes allowed in a branch. We shall be using the former to prune the regression decision trees in the random forest. However, the random forest algorithm uses two more methods to reduce over-fitting and ensure diversity among its trees. The first is bootstrapping, each tree is fit to a different bootstrapped version of the data. This is referred to as bagging and is another well-known ensemble method. The random forest algorithm adds a final step to ensure diversity, in which only a random subset of features are considered at each split.

This leads to the following algorithm:

1. Create a bootstrap sample of the data.
2. Grow a regression decision tree on the bootstrapped sample, considering a random subset of features at each split.
3. Repeat steps 1-2 until the forest is the desired size.

Thus far, we have mentioned 3 hyper-parameters which can be tuned to ensure the diversity of the trees within the random forest. These are; the size of the random subset of features that are considered at each split, the number of trees contained in the forest, and the minimum number of data points allowed in a terminal node. These hyper-parameters will be tuned using a train-validation-test split combined with a grid search over potential parameter values. Ideally, the 3 hyper parameters would be tuned simultaneously. However, a grid in 3 dimensions has a large computational cost. To reduce this computational cost, the size of the random subset of features will be tuned independent of the other hyper-parameters. The number of trees and size of terminal nodes are dependent and so will be tuned together. Some further fine tuning of the size of the terminal nodes will be conducted as a final step.

The data shall be split into 3 sets, with 60%, 20%, and 20% of the data being assigned to the training, validation, and test sets respectively. A random forest will be fit to training set and evaluated on the validation set. This will be repeated for each value of a given hyper parameter. By tuning the hyper-parameters on the unseen validation set, it allows the optimal hyper-parameters for unseen data to be chosen without artificially inflating predictive performance on the test set.

The grid of values for the size of the random subset of features, will consist of 3 values; $\frac{p}{4}$, $\frac{p}{3}$, and $\frac{p}{2}$. The number of trees was held constant at 500, with a minimum node size of 5. The value which achieved the lowest mean square error on the validation set was $\frac{p}{3}$. Next, we tuned the number of trees over 3 possible values; 250, 500, and 1000. The minimum node size was restricted to 25, 50, and 100. A grid of all possible pairs of the parameters was created. The pair which achieved the lowest mean square error on the validation set was when the number of trees was set to 250 and minimum node size was set to 25. It should be noted that there was a marginal difference between the mean square error achieved by the different minimum node sizes when the number of trees was 250. If computational power was at a premium this value could have been selected to be higher. In this case, as the lowest value on the grid was the best performing, we will fine tune this parameter. The grid of values for this fine tuning will consist of 5 values; 5, 10, 15, 20, and 25. The mean square error on the validation set was minimised when terminal node size was limited to 10. This gives us our optimal hyper parameters:

Hyper-parameter	Number of trees	Number of features	Minimum node size
Optimal Value	250	30	10

Table 3: Optimal Hyper parameter values

Next, we shall fit the neural network. A Neural Network is an algorithm designed to loosely replicate the neural structure of a brain. There are many forms that neural networks can take and we shall focus on simple sequential networks, known as a feed-forward network or Multi-Layer Perceptron (MLP). The network consists multiple distinct layer types which have differing properties. The first is called the input layer which contains a single neuron corresponding to each feature in the data. There is only one input layer in a neural network. The second type of layer is a hidden layer, which contains a user specified number of neurons. The number of hidden layers is also user specified and the optimal number is dependent on the structure of the data. The final layer is the output layer, which

for regression will consist of a single neuron as the output. This neuron will be a single continuous real number and will represent the year a song was released.

The layers are ordered; input layer → hidden layers → output layer. The data is passed sequentially through each layer. Every neuron in a previous layer will be combined using a linear combination, $z^{(l)}$ where l denotes the current layer, and a transformation function, known as the activation function. The linear combination is evaluated and then the output is passed to the activation function, i.e. $a^{(l)} = f(z^{(l)})$. This generates the value for a single neuron in the next layer. This is repeated as many times as there are neurons in the next layer. However, the set of coefficients for each linear combination will differ. The activation function can come in many forms which are usually non-linear. It should be noted the activation function is not applied to the first set of linear combinations which connect the input layer to the first hidden layer. The activation function allows for more complex functions to be replicated by a network. Note: the intercept term in the linear combinations is commonly referred to as the bias.

The coefficients of the linear combinations are known as weights. A neuron which is assigned a weight which is large in magnitude will have a large effect on the neuron it is connected to. A neuron with a near zero weight has a small effect on the neuron to which it is connected. The weights are iteratively adjusted to minimise the loss function. The optimisation algorithm we used to iteratively adjust these weights is known as back propagation. This allows for the network to "learn" the function it is attempting to replicate. Back propagation is a gradient based method which uses the gradient of the loss function with respect to each weight to find the combination of weights which minimises the loss function. Using the assumption that our data is i.i.d, we can sum the individual gradients to get the total gradient:

$$\nabla_\omega J(\boldsymbol{\omega}) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\omega L(x_i, y_i, \omega) \tag{2}$$

This kind of direct computation of all gradients can be computationally expensive, so stochastic gradient descent is used. We select a small subset of the data points of size m, with $m \ll n$, and compute Eq. (2) for this subset. The weights are then updated using the following:

$$\boldsymbol{\omega^{(l,t)}} = \boldsymbol{\omega^{(l,t-1)}} - \alpha \nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega})$$

where α is the learning rate and $\boldsymbol{\omega^{(l,t)}}$ indicates the weights for layer l on iteration t, known as a train epoch. To compute the necessary gradient, the chain rule is utilised recursively:

$$\frac{\partial L}{\partial \omega_{jk}^{(l)}} = \frac{\partial L}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial \omega_{jk}^{(l)}}$$

where $\omega_{jk}^{(l)}$ is the kth weight in the jth neuron in layer l, and $z_j^{(l+1)}$ is the linear combination assigned to the jth neuron in layer l. The recursion comes from the need to calculate $\frac{\partial L}{\partial z_j^{(l+1)}}$ which is done using the following:

$$\frac{\partial L}{\partial z_j^{(l+1)}} = \sum_{r=1}^{q} \frac{\partial L}{\partial z_r^{(l+2)}} \frac{\partial z_r^{(l+2)}}{\partial z_j^{(l+1)}}$$

where q is the number of neurons in the $l+2$ layer. The full derivation can be seen in [2, p. 398].

This leads us to the follow algorithm to train the neural network:

1. Initiate a network with the chosen structure and random initial weights.
2. Compute the output and intermediate values, $z^{(l)}$ and $a^{(l)}$ $\forall\, l$, of each data point in the subset. (forward propagation)

3. Calculate the gradient of the loss function w.r.t each layer and weight recursively.

4. Using stochastic gradient descent, adjust the weights to minimise the loss function. (back propagation)

5. Repeat steps 2-4 until convergence to a local minima of the loss function is achieved.

The loss function we used was the MSE, as it is regularly used for regression models. It also will allow direct comparison with the random forest model fit earlier. The above outlines the complex nature of MLPs and more importantly the many hyper-parameters which must be considered. We approach the tuning of these hyper-parameters utilising the same structure of train-validation-test split combined with a heuristic and trial and error approach to choices. The main hyper-parameters to be tuned were; The number of hidden layers, the number of neurons contained in each layer, the activation function, and the learning rate. Ideally an exhaustive grid search would be employed testing many combinations and fine tuning results to find the optimal result. However, computationally this is expensive and not feasible in this example. Therefore, we initially considered the choice of activation function. We restricted our testing to 3 of the most commonly used activation functions; ReLu, sigmoid, and hyperbolic tangent. They take the following form:

$$\text{ReLU}(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

$$\text{Sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

It should be noted that the sigmoid function output is restricted to [0,1] and tanh is restricted to [-1,1]. Therefore, the response was scaled using a min max scaler to [0,1]. This was in addition to the features being normalised as neural networks are not scale invariant.

To compare these activation functions, we used a base model of 1 hidden layer with 100 neurons. This was a relatively arbitrary choice of model, however some of the heuristics of the choice will be explained later. The learning rate for each activation function was tuned through trial and error, with the resulting values being 0.001, 0.0005, and 0.003 respectively. The learning rate controls the magnitude of the effect of the gradient through each training epoch. The optimal learning rate is as large as possible to allow for quick convergence but small enough to ensure the minima is reached and not "stepped over". More advanced algorithms use an adaptive learning rate. The default value of 0.01 was used as a starting point, and the learning rate was reduced until the MSE on the validation set became stable through late-stage training epochs. The MSE on the validation set of the last training epoch was used as the metric of comparison:

Activation Function	ReLu	Sigmoid	Tanh
MSE of Validation set	0.0841	0.0862	0.0846

Table 4: Comparison of activation functions

As can been seen in table 4, the ReLu activation function outperforms the others. However, in reality due to variation of the MSE between individual epochs, any of these activation function could have been used based on these results.

The next hyper-parameters to be tuned are the number of hidden layers and the number of neurons per hidden layer. There is significant heuristic arguments to guide this decision despite no existence

of an accepted optimal method. For most problems, one hidden layer should suffice, with additional layers allowing for increased complexity to be captured. Whether adding additional layers increase performance is dependent on the problem. There is a risk of over-fitting with additional layers [3]. Therefore, we shall restrict ourselves to one or two hidden layers and increase the number if testing indicates it may be beneficial. The number of neurons per hidden layer is widely heuristic as well. There are many "guidelines" such as the mean of the size of input and output layers, or restricting size to less than twice the number of features. Others suggest limiting 5-100 neurons per layer depending on application [2]. In this case, as the number of features is high and we have restricted the number of layers, we shall test the following models and evaluate the MSE on the validation set of the final epoch:

Structure: (# of neurons)	1 Layer: (50)	1 Layer: (100)	2 Layer: (50,50)	2 Layer: (100,50)	2 Layer: (100,100)
MSE of Validation set	0.0906	0.0857	0.0933	0.0907	0.0893

Table 5: Comparison of structures

Looking at table 5, we see that a single layer of 100 neurons achieves the lowest MSE on the validation set and therefore this will be the structure of our final model. Before proceeding, a few comments about the optimality of the model are necessary. This model is the best performing model found based on the MSE achieved on the validation set. We utilised heuristic arguments in combination with trial and error to examine a small sub set of possible models. As mentioned earlier, due to the large number of hyper-parameters, testing an exhaustive set of models is computationally expensive and not feasible. We have restricted ourselves to the commonly used algorithms such as stochastic gradient descent and back propagation, despite more complex and optimal methods being available. We have also restricted the number of hidden layers and their neurons, and the activation functions which we considered. There is a large probability that a more optimal structure exists and one method which could be considered efficient is the use of regularisation. Regularisation is a method which can shrink the effect of certain neurons making the model less prone to over-fitting. This regularisation works similarly to ridge or lasso regression and makes tuning of layer and neuron combinations easier. This is achieved by fitting a large model and allowing regularisation to "kill" neurons which are unnecessary. This method does require tuning of two other hyper-parameters but could make a grid search more effective.

Finally, to choose and compare the random forest and MLP, we fit both models to the training and validation sets combined and evaluate their performance on the test set. The predictions produced from the neural network are scaled due to the minmax scaler placed on y. Therefore, these shall need to be reverted using the minimum and maximum of the training responses. This results in the model being able to extrapolate on values greater than one, but due to the use of ReLu, it cannot extrapolate beyond the minimum of the training set. In this case, both 1922 and 2011 appear in the training set but this is seed dependent. Potentially this minmax scaling should not have been applied for the ReLu function, however this would've made model comparison over activation functions more difficult.

Model	Random Forest	Neural Network
MSE of test set	180.7176	196.2425

Table 6: Comparison of final model performance

Looking at table 6, we see that the random forest outperforms the neural network on the test set. This leads us to deduce that the random forest has modelled the data better and use it for variable importance.

To estimate the variable importance, we shall use a permutation based method:

1. Randomly permutate a single feature in the out of bag data.
2. Calculate the increase in out of bag error.
3. Average this increase over all trees.
4. Repeat steps 1-3 for each feature.

Using this method, the feature with the largest average increase in out of bag error is deemed the most important. The out of bag error is the MSE of the predictions on the out of bag data-set. This leads to the following plot:

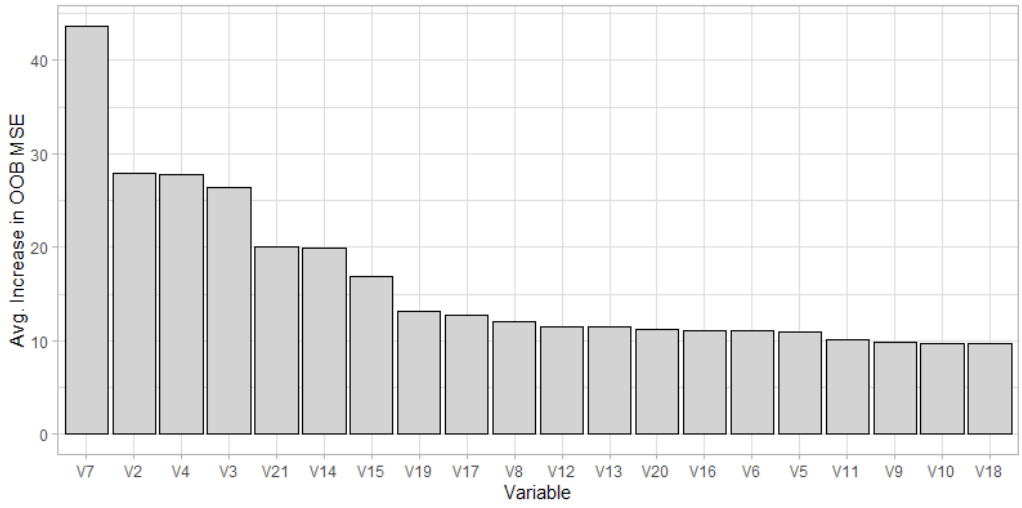


Figure 4: Top 20 most Important variables by Average increase in MSE on Out of Bag data

As can be seen in Fig. 4, variable 7 is the most important variable by far with a mean increase of 45. Variables 2, 4, and 3, all have similar levels of importance, taking the next 3 spots.

3 Question 3

We consider a Bayesian linear regression model on the data presented. The data contains 600 observations of average temperature on a monthly basis from 1960 to 2010. Setting temperature as the dependent variable, $\boldsymbol{Y} = (y_1, \ldots, y_n)$, and months passed since January 1960 as the independent variable, $\boldsymbol{X} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_n})$. We consider a set of basis functions, $\{\Phi_1, \ldots, \Phi_m\}$ s.t. $\boldsymbol{x_i} \mapsto \Phi_j(\boldsymbol{x_i}), \quad j = 1, \ldots, m$. This leads to the regression:

$$y_i = \sum_{j=1}^{m} \theta_j \Phi_j(\boldsymbol{x_i}) + \epsilon_i, \quad i = 1, \ldots, n$$

where $\epsilon_i \overset{i.i.d}{\sim} N(0, \sigma^2)$, which leads to the assumption that: $y_i \overset{i.i.d}{\sim} N(\sum_{j=1}^{m} \theta_j \Phi_j(\boldsymbol{x_i}), \sigma^2)$.

Note: That as simplification of notation, we shall use $\boldsymbol{X}$ to indicate the version of the data which has had the basis functions $\boldsymbol{\Phi}$ applied.

We are interested in the estimation of the coefficients, $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$, under a Bayesian framework. This can be done by examining the form of $\pi(\boldsymbol{\theta}, \sigma | \boldsymbol{Y}, \boldsymbol{X})$. This requires a specification of a prior for σ. As the form of this prior is unknown, we shall instead make a simplifying assumption that σ is known. We use Bayes theorem for the following:

$$\pi(\boldsymbol{\theta}|\sigma, \boldsymbol{Y}, \boldsymbol{X}) = \frac{g(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta}, \sigma)\pi(\boldsymbol{\theta})}{\int g(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta}, \sigma)\pi(\boldsymbol{\theta}) d\boldsymbol{\theta}} \tag{3}$$

where $\pi(\boldsymbol{\theta}|\sigma, \boldsymbol{Y}, \boldsymbol{X})$ is the posterior distribution of $\boldsymbol{\theta}$,
$g(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta}, \sigma)$ is the likelihood function,
$\pi(\boldsymbol{\theta})$ is the prior distribution of $\boldsymbol{\theta}$.
Using $y_i \overset{i.i.d}{\sim} N(\sum_{j=1}^{m} \theta_j \Phi_j(\boldsymbol{x_i}), \sigma^2)$, we get that:

$$\begin{aligned} g(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta}, \sigma) &= \prod_{i=1}^{n} g(y_i|\boldsymbol{x_i}, \boldsymbol{\theta}, \sigma) \\ &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{y_i - \boldsymbol{x_i}\boldsymbol{\theta}}{2\sigma^2} \right) \\ &= \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left(- \frac{1}{2\sigma^2} (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\theta})^T (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\theta}) \right) \end{aligned}$$

Noting that $\boldsymbol{\theta} \sim N(\boldsymbol{0}, \alpha \boldsymbol{I})$ is given, we drop the denominator in Eq. (3) as it is a constant relative to $\boldsymbol{\theta}$ to get:

$$g(\boldsymbol{\theta}|\sigma, \boldsymbol{Y}, \boldsymbol{X}) \propto \exp\left(- \frac{1}{2\sigma^2} (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\theta})^T (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\theta}) \right) \exp\Big(- \frac{1}{2}\, \boldsymbol{\theta}^T \alpha^{-1}\, \boldsymbol{\theta} \Big)$$

As given by Rasmussen in [4] this leads to:

$$g(\boldsymbol{\theta}|\sigma, \boldsymbol{Y}, \boldsymbol{X}) \propto \exp\left(- \frac{1}{2} \Big(\boldsymbol{\theta} - \sigma^{-2} A^{-1} \boldsymbol{X}^T \boldsymbol{y} \Big)^T A \Big(\boldsymbol{\theta} - \sigma^{-2} A^{-1} \boldsymbol{X}^T \boldsymbol{y} \Big) \right)$$

where $A = \sigma^{-2} \boldsymbol{X}^T \boldsymbol{X} + \alpha^{-1} \boldsymbol{I}$
Noting that the form of the posterior distribution is Gaussian, we get that:

$$g(\boldsymbol{\theta}|\sigma, \boldsymbol{Y}, \boldsymbol{X}) = N(\sigma^{-2} A^{-1} \boldsymbol{X}^T \boldsymbol{y}, A^{-1}) \tag{4}$$

This then leads us to the predictive posterior, which is also given by Rasmussen in [4]:

$$\begin{aligned} g(y_{\text{new}}|\boldsymbol{x}_{\text{new}}, \boldsymbol{X}, \boldsymbol{Y}, \sigma) &= \int g(y_{\text{new}}|\boldsymbol{x}_{\text{new}}, \boldsymbol{\theta}, \sigma)\, g(\boldsymbol{\theta}|\sigma, \boldsymbol{Y}, \boldsymbol{X})\, d\boldsymbol{\theta} \\ &= N(\sigma^{-2} \boldsymbol{x}_{\text{new}} A^{-1} \boldsymbol{X}^T \boldsymbol{y}, \boldsymbol{x}_{\text{new}} A^{-1} \boldsymbol{x}_{\text{new}}^T) \end{aligned}$$

As the mean and median of a normal distribution are equal, we note that the estimate of $\boldsymbol{\theta}$ from Eq. (4), is in-fact the maximum a posteriori estimate. If we consider a reformulation:

$$\begin{aligned} \hat{\theta}_{\text{MAP}} &= \sigma^{-2} A^{-1} \boldsymbol{X}^T \boldsymbol{y} \\ &= \sigma^{-2} (\sigma^{-2} \boldsymbol{X}^T \boldsymbol{X} + \alpha^{-1} \boldsymbol{I})^{-1} \boldsymbol{X}^T \boldsymbol{y} \\ &= (\boldsymbol{X}^T \boldsymbol{X} + \sigma^2 \alpha^{-1} \boldsymbol{I})^{-1} \boldsymbol{X}^T \boldsymbol{y} \\ &= (\boldsymbol{X}^T \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^T \boldsymbol{y} \end{aligned}$$

Where $\lambda = \sigma^2\alpha^{-1}$. This gives that the linear system is solvable for any $\lambda > 0$. A further reformulation using the push-through-identity [5]: $(\boldsymbol{U}\boldsymbol{V} + \boldsymbol{I})^{-1}\boldsymbol{U} = \boldsymbol{U}(\boldsymbol{V}\boldsymbol{U} + \boldsymbol{I})^{-1}$

$$(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}_m)^{-1}\boldsymbol{X}^T\boldsymbol{y} = \boldsymbol{X}^T B^{-1}\boldsymbol{y}$$

where $B = (\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I}_n)$.

Next we see that:

$$\begin{aligned} A^{-1} &= (\sigma^{-2}\boldsymbol{X}^T\boldsymbol{X} + \alpha^{-1}\boldsymbol{I})^{-1} \\ &= \boldsymbol{I} - (\sigma^{-2}\boldsymbol{X}^T\boldsymbol{X} + \alpha^{-1}\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{X} \\ &= \boldsymbol{I} - \boldsymbol{X}^T(\sigma^{-2}\boldsymbol{X}\boldsymbol{X}^T + \alpha^{-1}\boldsymbol{I})^{-1}\boldsymbol{X} \\ &= \boldsymbol{I} - \boldsymbol{X}^T\sigma^2(\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{X} \\ &= \boldsymbol{I} - \boldsymbol{X}^T\sigma^2 B^{-1}\boldsymbol{X} \end{aligned}$$

From this we complete our reformulation of Eq. (4) to give:

$$\begin{aligned} g(\boldsymbol{\theta}|\sigma, \boldsymbol{Y}, \boldsymbol{X}) &= N(\boldsymbol{X}^T B^{-1}\boldsymbol{y}, \boldsymbol{I} - \boldsymbol{X}^T\sigma^2 B^{-1}\boldsymbol{X}) \\ g(y_{\text{new}}|\boldsymbol{x}_{\text{new}}, \boldsymbol{X}, \boldsymbol{Y}, \sigma) &= N(\boldsymbol{x}_{\text{new}}\boldsymbol{X}^T B^{-1}\boldsymbol{y}, \boldsymbol{x}_{\text{new}}(\boldsymbol{I} - \boldsymbol{X}^T\sigma^2 B^{-1}\boldsymbol{X})\boldsymbol{x}_{\text{new}}^T) \end{aligned} \tag{5}$$

Recalling our simplification of notation from earlier, we note that $\boldsymbol{x}_{\text{new}} = \Phi(\boldsymbol{x}_{\text{new}})$ and $\boldsymbol{X}\boldsymbol{X}^T = \Phi(\boldsymbol{X})\Phi(\boldsymbol{X})^T$. If we examine the entries of the latter:

$$(\Phi(\boldsymbol{X})\Phi(\boldsymbol{X})^T)_{i,j} = \langle\Phi(\boldsymbol{x_i}), \Phi(\boldsymbol{x_j})\rangle$$

Therefore, it is clear we can write $\Phi(\boldsymbol{X})\Phi(\boldsymbol{X})$ only as a matrix of inner products. Similarly working with the covariance from Eq. (5), we see that the features only enter in some combination of $\boldsymbol{X}$ or $\boldsymbol{x}_{\text{new}}$, i.e. an inner product. This means the covariance can also be written as a matrix of inner products.

A matrix of this form is more commonly known as a kernel matrix as defined by the following definition: A kernel function $K : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is a symmetric, positive definite function. A matrix given by applying this function to any set of points from $\mathcal{X}$ is positive semi-definite and known as a Kernel matrix.

Extending this, we have that Eq. (5) in fact satisfies the following definition of a Gaussian process given by Ramussen [4]: A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution. They defined the Gaussian process to have a mean function for its mean and a kernel function for its covariance. We have noted that the covariance matrix given by Eq. (5) is a kernel matrix. This shows that a Bayesian linear regression with some set of basis functions, Φ, gives a valid Gaussian process with some combination of kernel function $k(\cdot\,,\cdot) = \langle\Phi(\cdot), \Phi(\cdot)\rangle$ as its covariance function.

We are asked to consider fitting 3 Gaussian processes to the data presented. We shall consider a Gaussian process with the same assumptions as outlined above, i.e. $y_i \overset{i.i.d}{\sim} N(\sum_{j=1}^m \theta_j\Phi_j(\boldsymbol{x_i}), \sigma^2)$. However, we do not want a posterior for $\boldsymbol{\theta}$ as we are not fitting a linear model. In fact, we want a posterior of function values, $\boldsymbol{f} = (f_1, \dots)$, which represent a distribution of possible functions which could fit the data. The leads to a change of Eq. (3):

$$\pi(\boldsymbol{f}|\boldsymbol{Y}, \boldsymbol{X}) = \frac{g(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{f})\pi(\boldsymbol{f}|\boldsymbol{X})}{\int g(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{f})\pi(\boldsymbol{f}|\boldsymbol{X})d\boldsymbol{f}}$$

We assume that the prior is a Gaussian process with a mean function, $m(\cdot) = 0$ and covariance structure $k(\cdot\,,\cdot) = \langle\Phi(\cdot), \Phi(\cdot)\rangle$. We re-write Eq. (5), substituting the mean and kernel functions to give a predictive posterior of:

$$y_{\text{new}}|\boldsymbol{x}_{\text{new}}, \boldsymbol{X}, \boldsymbol{Y} \sim N(m_{\text{pred}}, k_{\text{pred}}) \tag{6}$$

where $m_{\text{pred}} = k(\boldsymbol{x}_{\text{new}}, \boldsymbol{X})(k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I})^{-1}\boldsymbol{Y}$
and $k_{\text{pred}} = k(\boldsymbol{x}_{\text{new}}, \boldsymbol{x}_{\text{new}}) - k(\boldsymbol{x}_{\text{new}}, \boldsymbol{X})(k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I})^{-1}k(\boldsymbol{X}, \boldsymbol{x}_{\text{new}})$

To fit 3 Gaussian models we shall use 3 different kernel functions, known as the Gaussian, the matérn, and periodic, which are as follows:

$$k_{\text{Gauss}}(\boldsymbol{x}, \boldsymbol{y}) = \sigma_f^2 \exp\Big(- \frac{(\boldsymbol{x} - \boldsymbol{y})^T(\boldsymbol{x} - \boldsymbol{y})}{l^2}\Big)$$

$$k_{\text{Mat}}(\boldsymbol{x}, \boldsymbol{y}) = \sigma_f^2 \exp\Big(- \frac{\sqrt{3}\|\boldsymbol{x} - \boldsymbol{y}\|_2}{l}\Big)\Big(1 + \frac{\sqrt{3}\|\boldsymbol{x} - \boldsymbol{y}\|_2}{l}\Big)$$

$$k_{\text{Periodic}}(\boldsymbol{x}, \boldsymbol{y}) = k_{\text{Gauss}}\left(\begin{bmatrix}\cos \omega \boldsymbol{x} \\ \sin \omega \boldsymbol{x}\end{bmatrix}, \begin{bmatrix}\cos \omega \boldsymbol{y} \\ \sin \omega \boldsymbol{y}\end{bmatrix} \right)$$

Where σ_f^2 is the signal variance, which controls the variance of the expectation over long stretches,
l is the length scale, which controls the number of turning points,
and ω is a periodicity parameter.

As can be seen above, there are 2 or 3 hyper-parameters to be trained depending on choice of kernel as well as the noise variance, σ^2, from the likelihood. Therefore, we shall fit models and tune the hyper-parameters using the following methodology. First, we shall create a training set of the data from January 1960 - December 2005. Then the period from January 2006 - June 2007 will be used as a validation set. Finally the period from July 2007 - December 2009 as a test set. The reason for this choice was we are asked to predict the temperature in March 2011, which is 15 months after the data ends. Therefore, a validation and test set size of 18 months is appropriate.

Using these sets, we shall fit the Gaussian processes to the training set and tune the hyper-parameters for the given kernel by minimising the negative of the log marginal likelihood (below) on the validation set. Once, the optimal parameters have been found, we shall fit the model to the training and validation set combined and calculate the log marginal likelihood on the test set. The model which achieves the lowest score shall be our chosen model. We shall then fit the process to the entire data-set and predict the temperature in March 2011. The negative log marginal likelihood is given as follows:

$$\log\left(\int g(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{f})\pi(\boldsymbol{f}|\boldsymbol{X})d\boldsymbol{f} \right) \propto \frac{1}{2}\boldsymbol{Y}^T k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I})^{-1}\boldsymbol{Y} + \frac{1}{2}\log\Big(|k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}|\Big)$$

where $|\cdot| = \det(\cdot)$

It should be noted that a global minimum may not exist and local minimum will be found. This will be dependent on the starting point of the optimisation algorithm. In this case, every parameter starts at a value of 1. This is chosen arbitrarily with an acceptance that the optimiser may have found a local minimum.

Kernel	Gaussian	Matérn	Periodic
σ^2	23.8628	24.0510	23.8192
σ_f^2	808.0482	646.2665	598.0437
l	356.4.4029	1409.9611	344.6086
ω	-	-	0.5293

Table 7: Optimal Hyper-parameters for the Gaussian Processes

Looking at the table 7, we see that the noise variance, σ^2, is similar for each of the models with a value near 24. We see that the 3 models have a large value of function variance and a large length-scale. This should lead to long, and smooth expectations with the ability to vary greatly over time.

As mentioned, there is chance these hyper-parameters have found local minima, but these fears may be abated due to the similarity of the values seen. Finally, we compare these models on the test set using the negative of the log marginal likelihood.

Kernel	Gaussian	Matérn	Periodic
Negative Log Marginal Likelihood	53.2781	53.4894	53.4744

Table 8: Comparison of kernel performance on test set

Looking at table 8, each Gaussian process performs similarly with the Gaussian kernel having a slightly lower score. Therefore, we shall refit the Gaussian process using the Gaussian kernel and its tuned hyper parameters.

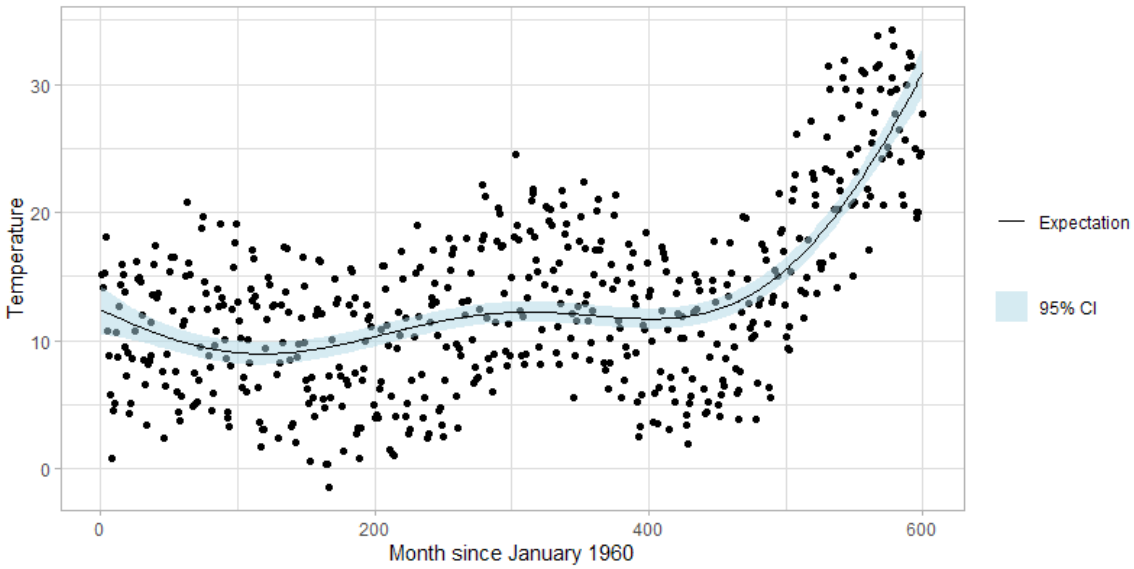
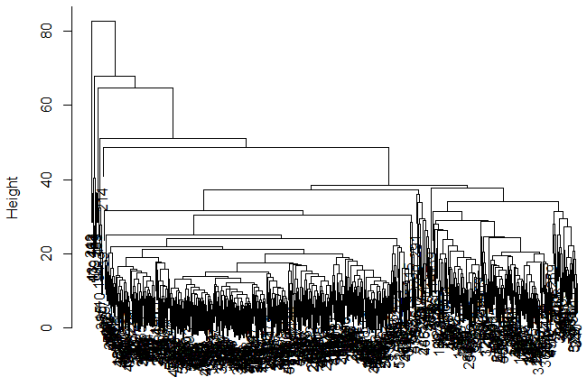


Figure 5

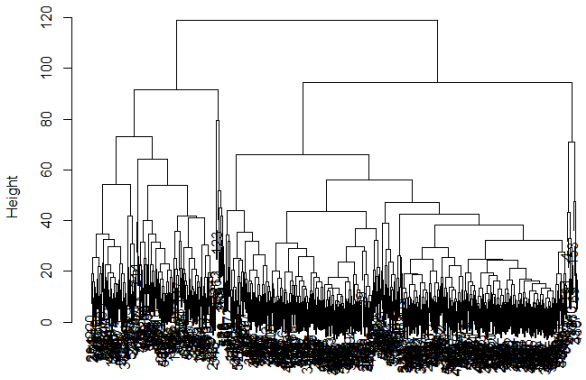
Examining Fig. 5, we see the effect of the tuned parameters; a low number of turning points due to a large length scale, a large function signal variance allowing for the large increase between 400 - 600 months, and a small confidence interval around the expectation.

Using this model, we predict the temperature in March 2011 to be 33.9598 with a standard deviation of 1.3074. This gives a 95% asymptotic confidence interval of $(31.3973, 36.5223)$, meaning the recorded value of 36 falls within our confidence interval. As a final note about this model and the inference drawn, the data appears to have steep increase over the last 100 observations. The current of assumption of normality appears reasonable but should this increase continue, the assumption of normality will be violated as observations to the right of the mean will occur more frequently. Therefore, this model should only be used for short term predictions, as if this trend continues the model assumptions will be invalid.

A Plots

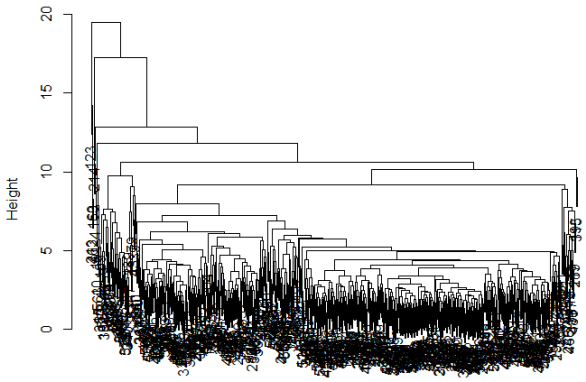


(a) Manhattan Distance with average linkage

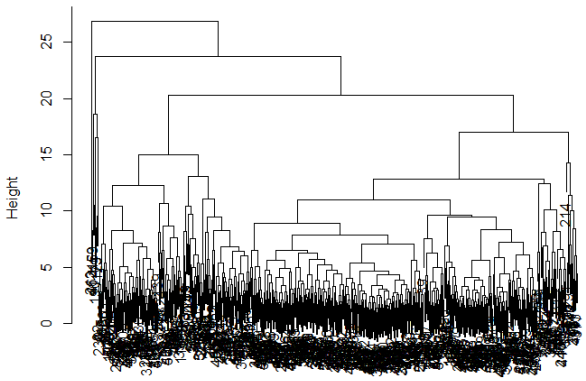


(b) Manhattan Distance with complete linkage

Figure A.1: Manhattan based Hierarchical Clustering Dendrograms



(a) Euclidean Distance with average linkage



(b) Euclidean Distance with complete linkage

Figure A.2: Euclidean based Hierarchical Clustering Dendrograms

References

[1] L. Breiman, *Classification and regression trees.* Wadsworth International Group, 1984.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning.* Springer Series in Statistics, Springer New York Inc., 2001.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org.

[4] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning.* Adaptive Computation and Machine Learning, MIT Press, 2006.

[5] H. V. Henderson and S. R. Searle, "On deriving the inverse of a sum of matrices," *Siam Review*, vol. 23, no. 1, pp. 53–60, 1981.