Machine Learning

02524541

April 2024

Question 1

Part 1

(1)

Figure 1 shows the mean absorption over all the samples at each wave length. It can be seen that as the wavelength increases, the mean level of absorption increases from roughly -1 to 1.6. This shows that as the wavelength increases from 800 to 999, the level of absorption increases. Between the wavelengths of 900 and 925, the mean absorption increases and then decreases dramatically.

Feature	Correlation
Temperature	0.164507
Water	0.005536
Ethanol	0.005536
2-Propanol	0.005536

Table 1: Correlations between Various Features and Absorption

In the table above, it is clear that each component has a positive correlation with the mean absorption over the wavelengths. Although the data is not linear, the correlation coefficients can show a general increase in the mean absorption over each feature.

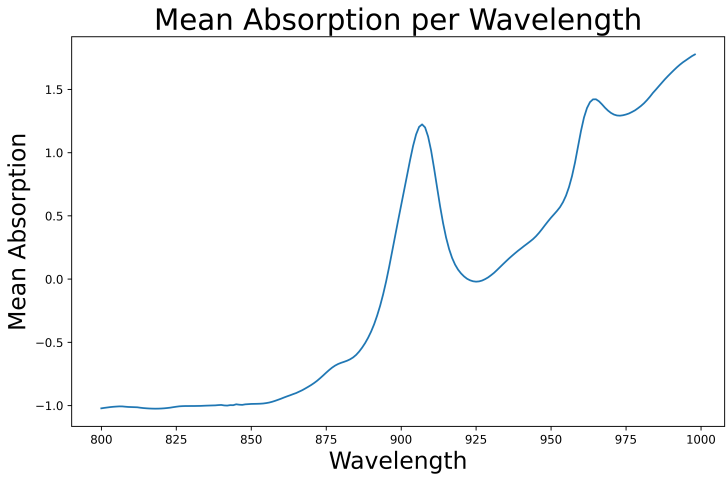


Figure 1: Mean Wavelength Absorption

The temperatures take values between 30 to 70. The concentrations of water, Ethanol and 2-Propanol are take values between 0 and 1.

(2)

Hierarchical clustering was performed,

The 2 metrics used to decide how the hierarchical clustering was performed were "euclidean" and "cityblock". Each of these metrics has different ways to conclude the distance between a new value.

The 3 methods used to decide how the hierarchical clustering is done were "complete", "single" and "average". Each of these methods has different ways to conclude the distance between a new value.

Below is the graphs of each hierarchical cluster using different metrics and methods, this was performed using the `linkage` and `dendrogram` function as part of the `scipy.cluster.hierarchy` library.

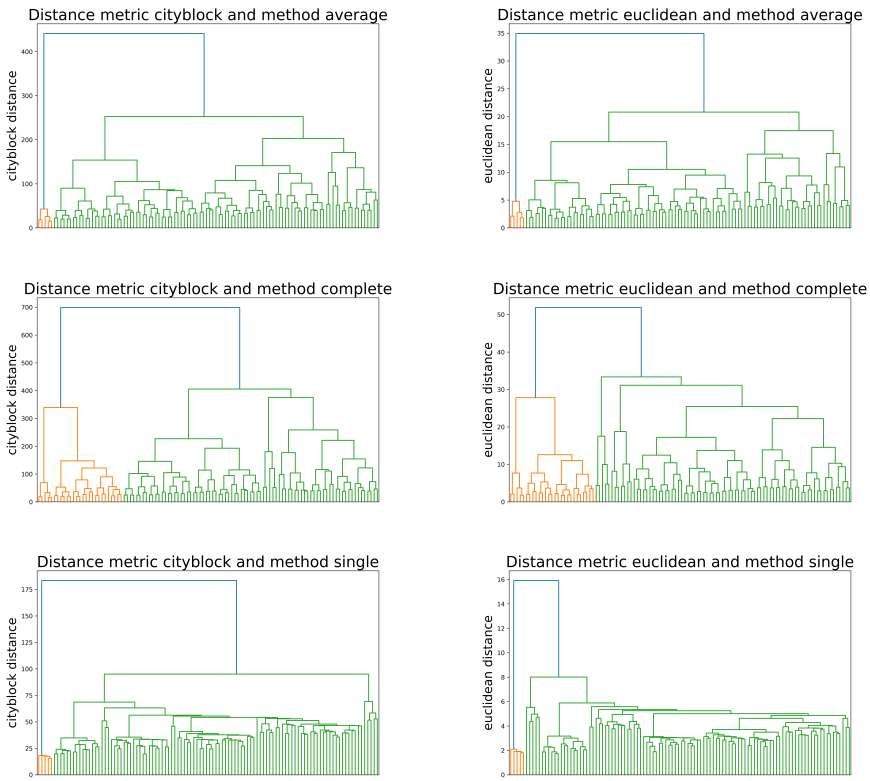


Figure 2: Hierarchical Clustering for Different Metric and Methods

As can be seen in the graphs above, different distance metrics produce different types of data splits. For example, the single and average distance metrics show less clearly distinct clusters or groups compared to the euclidean distance metrics.

Using the `fcluster` function, as part of the `scipy.cluster.hierarchy` library, the data was split into 4 distinct groups using 2 different combinations of methods and metrics.

Group	Water	Ethanol	2-Propanol	Temperature
1	0.000	1.000	0.000	49.966
2	0.231641	0.511095	0.257264	53.063913
3	0.494467	0.186763	0.318770	46.5115
4	0.324083	0.000	0.675917	48.242353

Table 2: Mean Values of Features for each group using "average" linkage method and "cityblock" distance metric

Group	Water	Ethanol	2-Propanol	Temperature
1	0.154217	0.685316	0.160468	54.1
2	0.0	0.0	1.0	50.018
3	0.738271	0.000000	0.261729	35.714286
4	0.385679	0.253295	0.361026	49.428406

Table 3: Mean Values of Features for each Group using "complete" linkage method and "euclidean" distance metric

When separating the data into different clusters, similar components and temperatures tend to cluster together. This is why there are clusters with unnaturally high or unnaturally low mean values for the components and the temperature values.

(3)

The principal component analysis of the spectra was performed on the space using the `PCA` function as part of the `sklearn` library. Plotted below is the cumulative variance for each number of principal components.

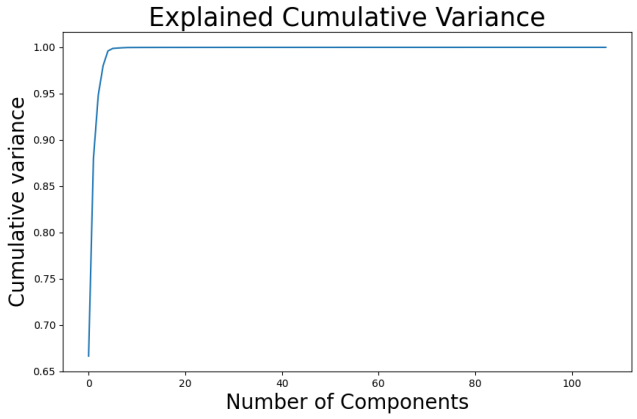
- 

Figure 3: Cumulative Variance for each Number of Principal Components

The smallest amount of principal components to account for 99.99% of the variance is 14.

Dimension reduction was performed by taking the standardised spectra data points that accounted for 99.99% of the variance. The mean of the overall data set and the reduction data sets were charted below to compare the data sets. As can be seen the reduced data set has a smaller mean for temperature and 2-propanol, and a larger mean for ethanol.

Features	14 Principle Features Mean	All Features Mean
Temperature	42.873571	49.734722
Water	0.344916	0.332811
Ethanol	0.392625	0.333156
2-Propanol	0.262459	0.334033

Table 4: Mean Values for 14 Principle Component Features and Overall Data

Hierarchical clustering was performed on the reduced data set, using the same metrics and methods used in the previous part of this question.

The clusters are plotted below,

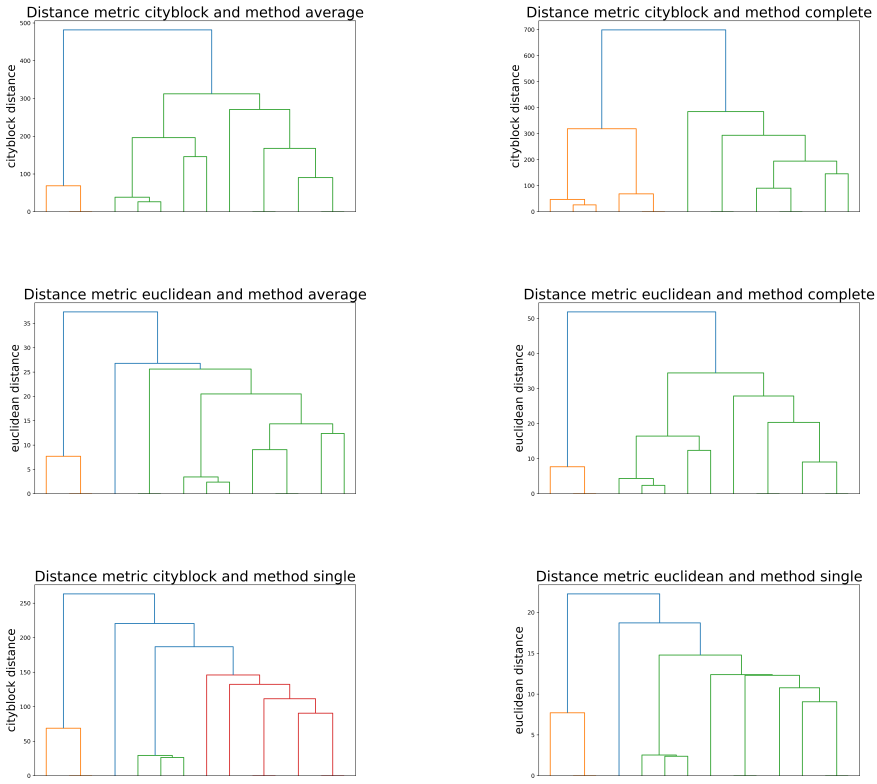


Figure 4: Hierarchical Clustering for Different Metric and Methods with Reduced Data Set

Group	Water	Ethanol	2-Propanol	Temperature
1	0.0	1.0	0.0	43.346667
2	0.232409	0.499349	0.268242	42.01
3	0.733357	0.0	0.266643	46.028
4	0.0	0.0	1.0	30.0

Table 5: Mean Values of Features for each group using "complete" linkage method and "euclidean" distance metric, for the Reduced Data

Group	Water	Ethanol	2-Propanol	Temperature
1	0.0	1.0	0.0	43.346667
2	0.232409	0.499349	0.268242	42.01
3	0.733357	0.0	0.266643	46.028
4	0.0	0.0	1.0	30.0

Table 6: Mean Values of Features for each group using "complete" linkage method and "euclidean" distance metric, for the Reduced Data

Just like the groups from the overall data set, different groups have different means for the component and the temperature values. This is seen in the fact that, the single and average distance metrics show less clearly distinct clusters/groups compared to the euclidean distance metrics.

Part 2

(1)

- A sample of 40 chemical samples was randomly selected from the 108 original chemical samples using the `.sample()` function from a `NumPy` data set,
- Then random noise was added using the parameters specified in the course-work to each element
- 80% of the 40×199 measurements were replaced by `NaN` values

This was all done with the author's student number (2524541) set as the random seed value.

(2)

From the 40 components, the component with the smallest amount of missing wavelength data points was chosen. The reason for this is because with more data points the accuracy of the error is likely to decreases and a more nuanced understanding of the data can be found.

A combination of the following Gaussian processes was tested using k-fold validation with 3 folds and with the `model_selection.KFold` function as part of the `sklearn` package.

The Exponential Sine Squared (ESS) kernel is denoted as $k(x, x')_{\text{ESS}}$, the Radical basis function (RBF) is denoted as $k(x, x')_{\text{RBF}}$, the White Noise kernel is denoted as $k(x, x')_{\text{WN}}$ and finally the Matern kernel is denoted is $k(x, x')_M$. All these kernels were implemented using the `sklearn.model_selection.kernels` library with the default hyperparameters set for each kernel.

As explained in Question 1 Part 1 (a), the absorption does not follow any simple pattern like a linear relationship or a quadratic one. For this reason, these kernels were picked because they can model complex relationships like the one seen between the wavelength and the absorption.

The kernel combinations below were tested using k-fold validation.

$$k(x, x')_M + k(x, x')_{\text{RBF}} + k(x, x')_{\text{WN}}$$

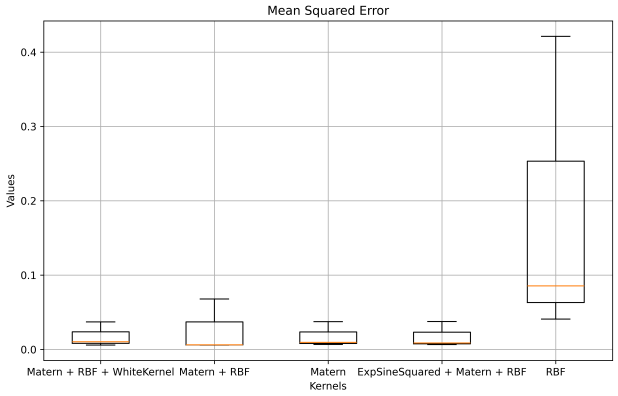
$$k(x, x')_M + k(x, x')_{\text{RBF}}$$

$$k(x, x')_M$$

$$k(x, x')_{\text{ESS}} + k(x, x')_M + k(x, x')_{\text{RBF}}$$

$$k(x, x')_{\text{RBF}}$$

A different combinations of all of these was tested using 3-fold validation to test which combination worked best to predict the absorption at any given the wavelength. A box plot of the results are plotted below with each different combination tested.

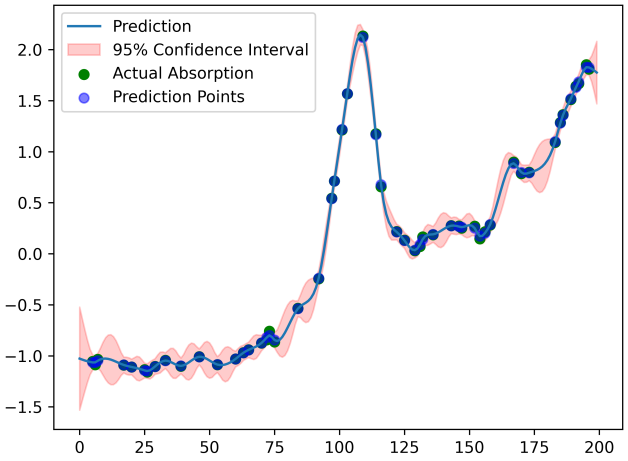


-

Figure 5: Boxplot of MSE for Different Kernels

Using the combinations of the Matern, RBF and white noise kernels, the mean squared error was the smallest at 0.015.

Using this kernel combination, the Gaussian regressor was trained using all the values in sample i^*. The predictions for each wavelength compared to the true absorption was plotted below.



-

Figure 6: Predictions Vs Actual Values

(3)

To perform this Gaussian process considering the data set $\mathcal{D}$ excluding the i^*-th chemical sample, a data set of the covariants and of the response were created.

The data set of the covariants contained each of the 39 chemical and temperature components repeated 199 times with each of the wavelengths. The response data set contained the corresponding absorption at each of the covariants data sets points.

The kernel was chosen by testing how different kernels fitted the data after being put thought the Gaussian process.

This process was very similar to the process found in Part 2 section 2 of this question, were k-fold cross validation below.

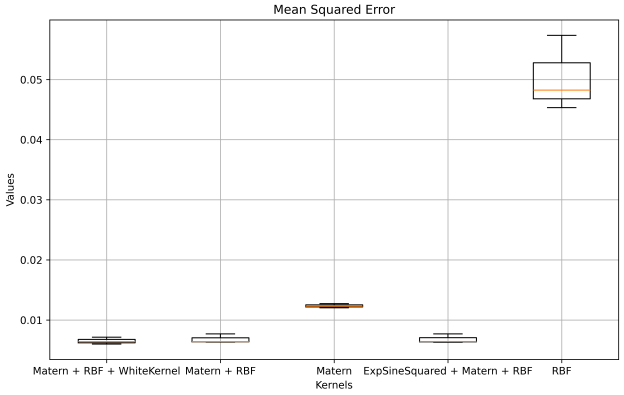


Figure 7: Boxplot of MSE for Different Kernels

Just like the previous section, the combinations of the Matern, RBF and white noise kernel is the smallest.

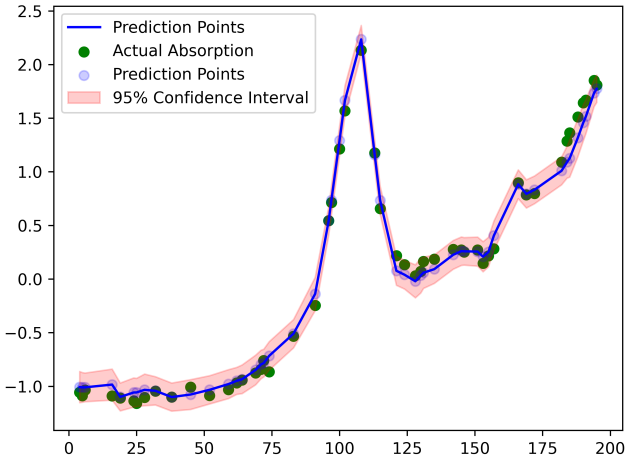


Figure 8: Predictions Vs Actual Values

The mean squared error between the predicted values and the actual values is 0.0079. This is a better predictor than the Gaussian process cross validations in the previous part of this question.

(4)

(a)

The model found to be the best from testing in Part 2 section 3 was used to train over the entire data set $\mathcal{D}$.

A list of 100 evenly spaced values between the minimum and maximum temperature value in $\mathcal{D}$ were created. Each temperature on this list was used with the test components and the 199 wavelengths to find 199 predictions of the absorption's at each wavelength and the σ (standard deviation) value associated with each prediction.

Then the Negative Log Predictive Density (NLPD) is calculated using the following equations,

$$\text{Wavelenght}_\text{i}\ \text{NLPD} = \left(\log(\sigma_i\sqrt{2\pi}) + \frac{1}{2}\left(\frac{(y_{\text{pred},i} - y_{\text{test},i})^2}{\sigma_i^2}\right)\right)$$

$$\text{Mean NLPD} = \frac{1}{n}\sum_{i=1}^{n}\left(\log(\sigma_i\sqrt{2\pi}) + \frac{1}{2}\left(\frac{(y_{\text{pred},i} - y_{\text{test},i})^2}{\sigma_i^2}\right)\right)$$

This was calculated for every temperature value in the list of possible temperature values and is plotted below.

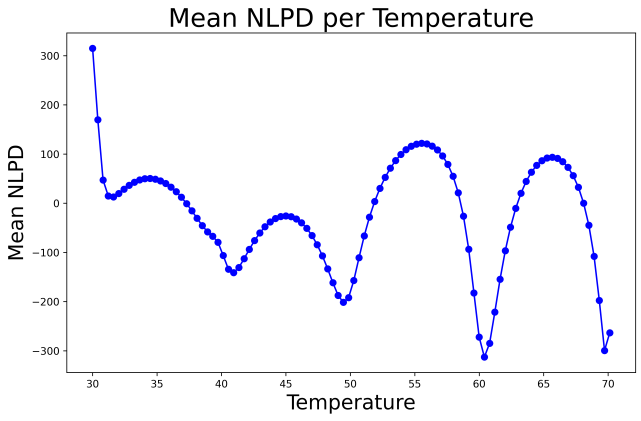
- 

Figure 9: Histogram of correlation coefficients.

The Negative Log Predictive Density (NLPD) has the smallest mean when the temperature is 60.40. From this Gaussian process, which has been shown to be very accurate, one can conclude that the temperature of this chemical sample is 60.40 or a value very close.

(b)

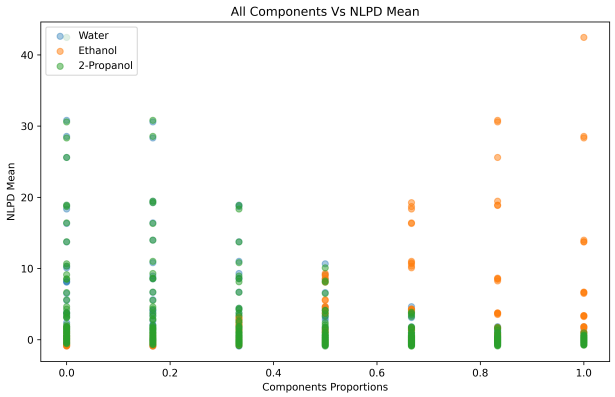
A similar process to the one described above was preformed using the test temperature 69.99 and the second set of observations of the absorption at each of the 199 wavelengths.

The important difference is that there are 3 components in the covariants while only one covariant was used in (a). This means that to create a list of components to test, 3 different factors must be considered

To do this a list of 15 evenly spaced numbers between 0 and 1 was created, then a $15^3 \times 3$ dimensions data frame was created which contains every single possible combination of this original list.

Using the test temperature provided and the 199 wavelengths, predictions and their associated value of sigma was calculated for each combination of 3 components.

Using these prediction values the mean NLPD was calculated for each combination of components.



-

Figure 10: Histogram of correlation coefficients.

The mixture of the 3 components that lead to the smallest NLPD are tabled below, the exact component values are likely not these exact values but something close.

Component	Best Value
Water	0.11111111
Ethanol	0.0
2-Propanol	0.55555556

Table 7: Components for Minimum Value of NLPD mean

Question 2

(1)

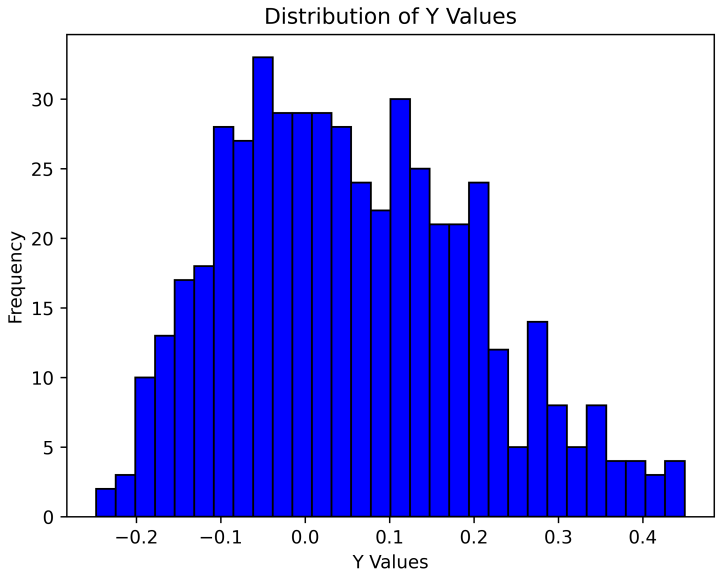


Figure 11: Histogram of 'y' Values

The histogram above shows the minimum value on the x-axis is roughly -0.3 and the maximum is roughly 0.5. This histogram is relatively well rounded between these values, and the mean is 0.056.

There are 49 covariants and they are very diverse; the below graph shows the distribution of the 49 covariants. Most of them have a mean less than one and a few outliers have a mean significantly greater than 1.

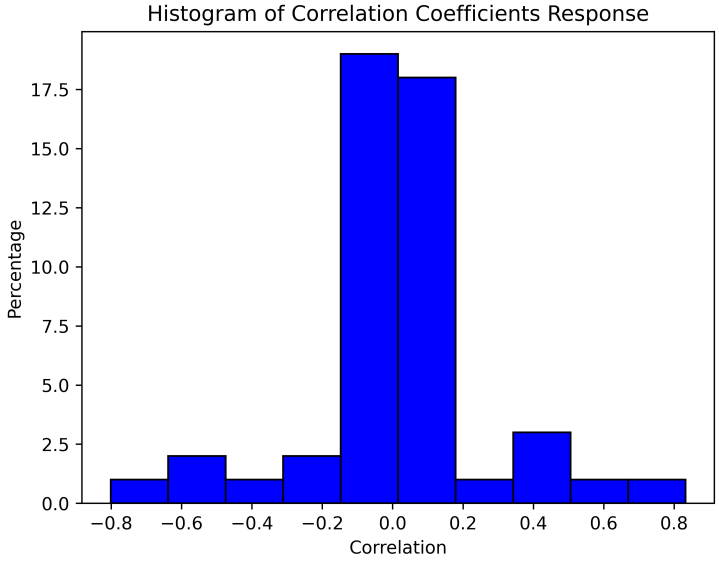


Figure 12: Histogram of Correlations

As can be seen in the graph above, many of the covariants are not strongly correlated with the response variable. Only 10 of the covariants have a absolute correlation value greater than 0.2.

(2)

The 4 machine learning models to predict the output of interest Y given the other biological features were, Decision Trees Regression, Decision Tree Regression, a k-nearest neighbour regression model and a custom Neural network.

The data was split into a training data set and a test data set using a 70%-30% split. All testing of hyperparameters was tested on the training dataset.

Throughout this section, the k-fold cross validation procedure is used to test hyperparameters on the training dataset.

The k-fold cross validation procedure used throughout uses 5 folds. This number of folds is chosen to reduce bias and variance while also not requiring excessive computational power.

The k-fold cross validation used involves splitting the data into 5 subsets. For each iteration 4 of the 5 subsets is used to train the model. Then, the model is tested against the subset not used for training. The step is repeated 5 times.

The loss function used to evaluate the performance of all the machine learning models was the Mean Squared Error (MSE) loss function.

Decision Trees

Since the output of the data is continuous and not discrete, a Regression Decision Tree is used instead of a Classifier Decision Tree. The models were made using the `DecisionTreeRegressor` function as part of the library `sklearn.tree`.

Different Loss criteria were set as hyperparameters in the model and then these hyperparameters were tested using k-fold validation with a MSE loss function. These were the 'absolute error', 'squared error' and 'friedman mse' loss functions.

Fitting Loss function	Mean MSE
Squared Error	8.83×10^{-3}
Absolute Error	8.13×10^{-3}
Friedman Mean Squared Error	9.04×10^{-3}

Table 8: Training Loss Function

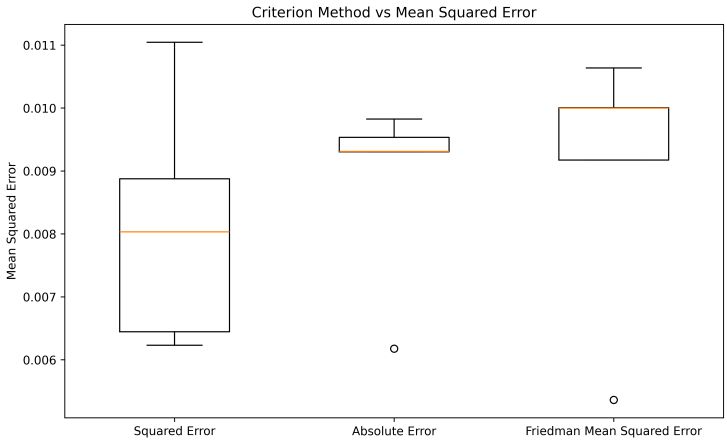


Figure 13: Boxplot of MSE form each Loss Function

The performance of each hyperparameter was tested by using the k-fold cross validation procedure explained above on the training data.

It was tested whether a random training splitter or a best training splitter produced a smaller MSE error. This was tested by performing k-fold validation on the training data.

Splitter	Mean MSE
Best	$8.125722277687000 \times 10^{-3}$
Random	$11.758628541753999 \times 10^{-3}$

Table 9: MSE Comparison for Different Data Splitters

As can be seen in the table above the best training split produces a lower MSE error. The hyperparameters were chosen that minimised the MSE through the cross validations.

Random Forest Regession

As can be seen in the histogram above in question 2 part 1, most of the covariants have a very small correlation with the response variable. In the Decision Trees Regression model it was both tested using the entire data set and using the 10 covariants in which the correlation with the predictor has a absolute value greater than 0.2. The reduced data frame of covariants is referred to as `X_filtered` in the code in the appendix below.

Since the output of the data is continuous and not discrete, a Random Forest Regression is used instead of a Random Forest Classifier. The models were made using the `RandomForestRegressor` function as part of the library `sklearn.ensemble`.

As can be seen in the table below, the reduced dataset has a noticeably lower mean MSE under k-fold cross validation than when training is performed on the entire dataset.

n Estimators	MSE Full	MSE Filtered	Difference
100	4.815×10^{-3}	4.033×10^{-3}	0.786×10^{-3}
350	4.742×10^{-3}	4.018×10^{-3}	0.724×10^{-3}
500	4.714×10^{-3}	4.019×10^{-3}	0.695×10^{-3}
1000	4.706×10^{-3}	4.008×10^{-3}	0.698×10^{-3}

Table 10: MSE with full and reduced data sets against different number of estimators

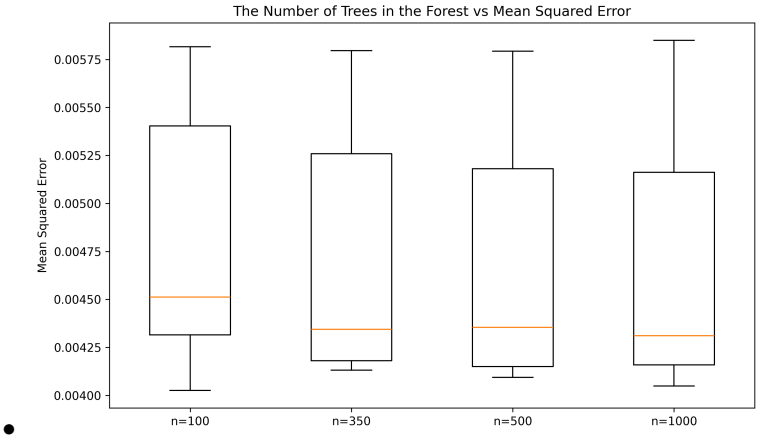


Figure 14: Boxplot of MSE Loss on complete data set against Different Numbers of Estimators

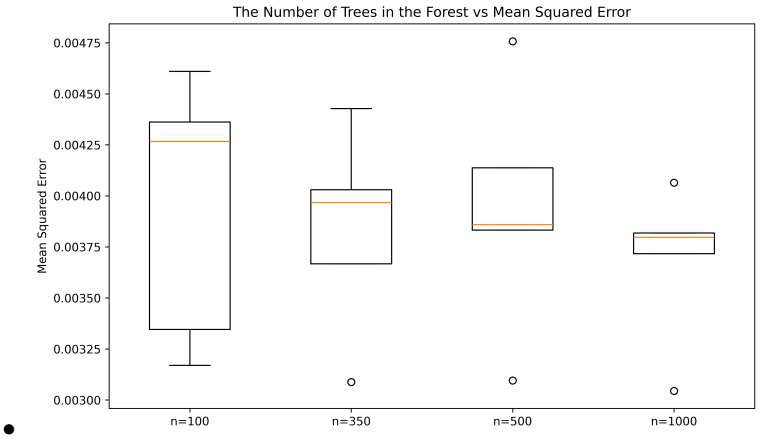


Figure 15: Boxplot of MSE Loss on reduced data set against Different Numbers of Estimators

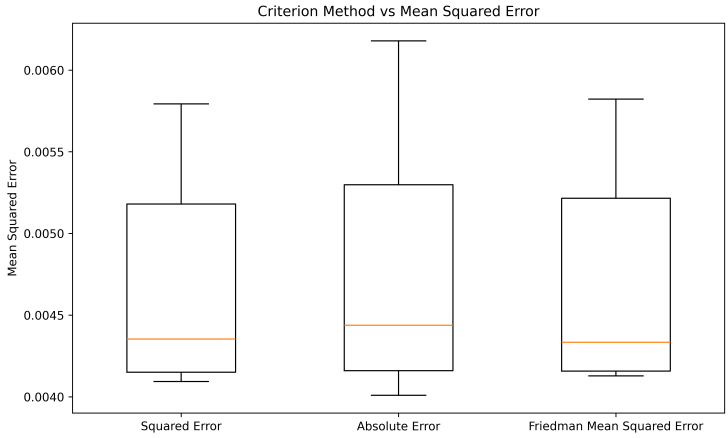


Figure 16: Boxplot of MSE Loss on complete data set against Different Loss Criteria

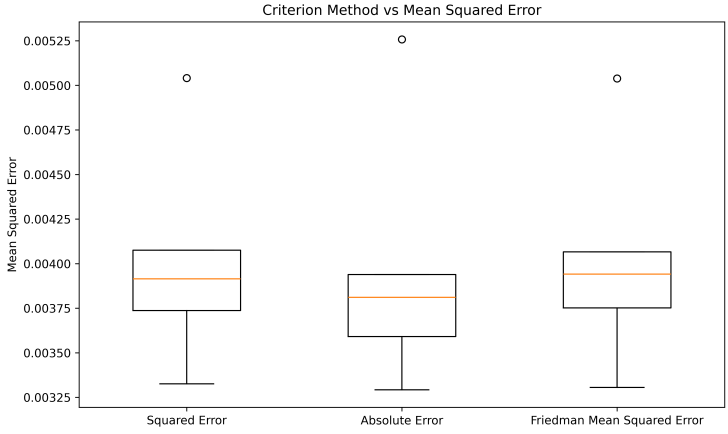


Figure 17: Boxplot of MSE Loss on reduced data set against Different Loss Criteria

As can be seen in boxplots and table above as the number of estimators increases the smaller the mean MSE under k-fold cross validation becomes. However as the number of estimators increases so does the computational costs increase.

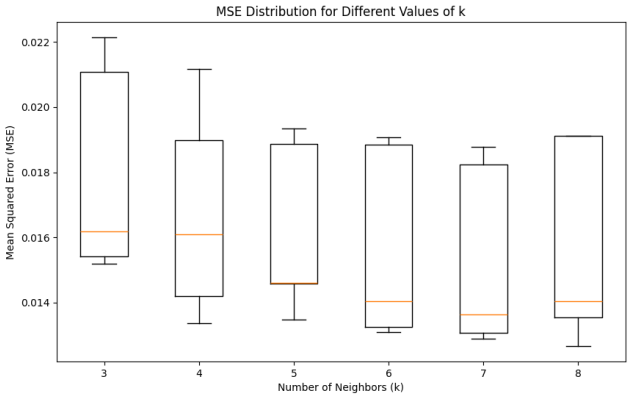
Different Loss criteria were set as hyperparameters for the Random Forest Regression model. Then, the hyperparameters were tested using k-fold validation with a MSE loss function. These were the 'absolute error', 'squared error' and 'friedman mse' loss functions.

From these results, 500 estimators were chosen. The mean squared error function was used as the loss function.

K-Nearest neighbour

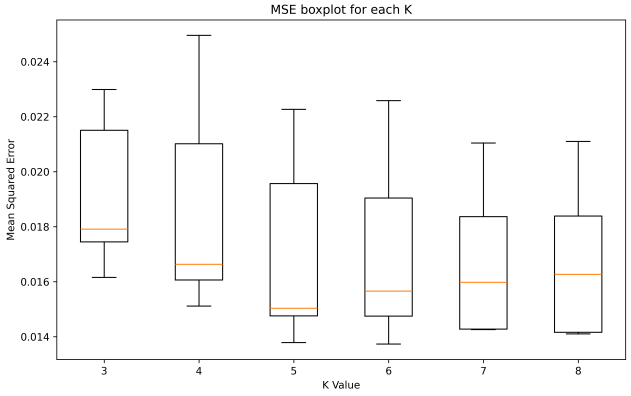
A k-Nearest neighbour algorithms were implemented using the `KNeighborsRegressor` functions as part of the `sklearn.neighbors` library.

To find the best hyperparameters different values for k between 3 and 8 were tested in k-nearest neighbour algorithm. The k-nearest neighbour algorthm was also tested using the 2 different distance metrics 'chebyshev' and 'manhattan'.



-

Figure 18: MSE for different values of k with chebyshev distance metric



-

Figure 19: MSE for different values of k with manhattan distance metric

k Value	Mean MSE (Chebyshev)	Mean MSE (Manhattan)
3	0.018	0.0192
4	0.0168	0.0188
5	0.0162	0.0171
6	0.0157	0.0172
7	0.0153	0.0168
8	0.0157	0.0168

Table 11: Mean MSE values for different k values using Chebyshev and Manhattan Distance

The lowest MSE is found when the distance metric is 'Chebyshev' and with $k = 7$. So this was used on the test data.

Neural network

A neural network was used to fit to the data. The neural network is a combination of the Rectified Linear Unit (ReLU) function and linear functions. This was implemented using functions and optimiser from the `torch` library. The exact architecture of the neural network can be found in the code appendix.

(3)

Method	Mean Squared Error
Random Forest Regression	0.0029
Decision Tree Regression	0.006
Neural Network	0.01922
K-nearest neighbour	0.015

Table 12: Summary of Mean Squared Error (MSE) for each Machine Learning model

(4)

From each of the models, it is clear that the Decision Tree Regression and Random Forest Regression preformed better than the Neural Network and k nearest neighbour with given hyperparameters. The Random Forest Regression had the smallest MSE on the Test data set.

Because of the fact that the Random Forest Regression on the reduced data set consistently preforms better under k-fold cross validation

Code Appendix

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVR
from sklearn.model_selection import cross_val_score, KFold

import seaborn as sns

from scipy.cluster.hierarchy import linkage, dendrogram
from scipy.cluster.hierarchy import cut_tree

seed=2524541

np.random.seed(seed)
```

```
# Import Dataset
components = pd.read_csv('components.csv', header = None, sep=' ')
spectra = pd.read_csv('spectra.csv',header=None, sep=' ')
temperatures = pd.read_csv('temperatures.csv',header=None)
```

Question 1

Part 1

(a)

```
plt.plot(np.mean(spectra, axis=0))

plt.title('Mean Absorption per Wavelength', size = 25)

plt.ylabel('Mean Absorption', size =20)

plt.xlabel('Wavelength', size = 20)

step = 25
original_indices = np.arange(0, 201, step)
mapped_indices = original_indices + 800

plt.xticks(ticks=original_indices, labels=mapped_indices)

plt.gcf().set_size_inches(10, 6)

```

```
plt.savefig('Q1_analysis_1.png', dpi=600, bbox_inches='tight')

plt.show()
```

```
names = ["water", "ethanol", "2-propanol"]
for i in range(3):
    print(f"The Correlation Coefficient between components {names[i]} and absorption is",
          np.corrcoef(np.mean(spectra,axis=1),components[1])[0][1])
print(f"The Correlation Coefficient between temperature {names[i]} and absorption is",
np.corrcoef(np.mean(spectra,axis=1),temperatures[0])[0][1])
```

(b)

```
# Import hierarchy cluster packages need
from sklearn.decomposition import PCA
from scipy.cluster.hierarchy import linkage, fcluster, dendrogram
```

```
# standardized the spectra values
standardized_spectra = (spectra - spectra.mean()) / spectra.std()

Metrics = ["euclidean", "cityblock"]
Methods = ["complete", "single", "average"]

for method in Methods:
    for metric in Metrics:
        fig, axes = plt.subplots(figsize=(10, 6))
        linkage_data = linkage(standardized_spectra, method=method, metric=metric)
        dendrogram(linkage_data, ax=axes)
        # remove x values
        axes.set_xticklabels([])
        axes.set_title(f"Distance metric {metric} and method {method}", size=25)
        axes.set_ylabel(f'{metric} distance', size=20)
        fig.savefig(f"Q1b_{method}_{metric}.png", dpi=600)
```

```
# Linkage data with method='complete' and metric='euclidean'
linkage_data = linkage(standardized_spectra, method='complete', metric='euclidean')
cluster_labels = fcluster(linkage_data, t=4, criterion='maxclust')

for i in range(1,5):
    components_values = components.iloc[cluster_labels==i].mean()
    print(f"Group {i}")
```

```
    print("Water Mean", components_values[0])
    print("Ethanol Mean", components_values[1])
    print("2-propanol Mean", components_values[2])
    print("Temperatures
    ↪  Mean",temperatures.iloc[cluster_labels==i].mean())
    print('------------------------------')
```

```
# Linkage data with method='average' and metric='cityblock'
linkage_data = linkage(standardized_spectra, method='average',
↪  metric='cityblock')
cluster_labels = fcluster(linkage_data, t=4, criterion='maxclust')

# Print Values
for i in range(1,5):
    components_values = components.iloc[cluster_labels==i].mean()
    print(f"Group {i}")
    print("Water Mean", components_values[0])
    print("Ethanol Mean", components_values[1])
    print("2-propanol Mean", components_values[2])
    print("Temperatures
    ↪  Mean",temperatures.iloc[cluster_labels==i].mean())
    print('------------------------------')
```

(c)

```
# Define principal component analysis
pca_calculator = PCA()

pca_calculator.fit(standardized_spectra)

# Get variance explained on every value
var_explained = pca_calculator.explained_variance_ratio_
cumvar_explained = np.cumsum(var_explained)


# Plot Explained Cumulative Variance
plt.figure(figsize=(10,6))

plt.plot(cumvar_explained, linestyle='-')
plt.xlabel('Number of Components', size = 20)
plt.ylabel('Cumulative variance', size = 20)
plt.title('Explained Cumulative Variance', size = 25)
plt.savefig('Q1_c_cumvar.png')

plt.show()


```

```

# get the number of values that explain 99.99\% of the variance
min_cumvar_explained = len(cumvar_explained) - sum((cumvar_explained>=0.9999)*1) + 1

print("Number of components needed to explain 99.99\% is", min_cumvar_explained)
```

```
n=4
linkage_data = linkage(Simplified_data, method='complete', metric='euclidean')
cluster_labels = fcluster(linkage_data, t=n, criterion='maxclust')

for i in range(1,n+1):
    components_values = components.iloc[Simplified_data.index[cluster_labels==i]].mean()
    print(f"Group {i}")
    print("Water Mean", components_values[0])
    print("Ethanol Mean", components_values[1])
    print("2-propanol Mean", components_values[2])
    print("Temperatures Mean",temperatures.iloc[Simplified_data.index[cluster_labels==i]].mean())
    print('-------------------------------')
```

```
# Print mean values for different data sets
print(temperatures.mean(),components.mean())
print(temperatures.loc[Simplified_data.index].mean()
      ,components.loc[Simplified_data.index].mean())
```

```
linkage_data = linkage(Simplified_data, method='average', metric='cityblock')
cluster_labels = fcluster(linkage_data, t=n, criterion='maxclust')

for i in range(1,n+1):
    components_values = components.iloc[Simplified_data.index[cluster_labels==i]].mean()
    print(f"Group {i}")
    print("Water Mean", components_values[0])
    print("Ethanol Mean", components_values[1])
    print("2-propanol Mean", components_values[2])
    print("Temperatures Mean"

    ,temperatures.iloc[Simplified_data.index[cluster_labels==i]].mean())
    print('---------------------------')
```

Part 2

(a)

```
seed_value = 2524541
n=40
components_sampled = components.sample(n=n,
↪  random_state=seed_value)
index = components_sampled.index

spectra_noisy_sampled = spectra.iloc[index]
temperatures_sampled = temperatures.iloc[index]
components_sampled = components.iloc[index]

np.random.seed(seed)

num_rows = 40
num_columns = 199
num_data_points = num_rows * num_columns
random_data = np.random.normal(scale=np.sqrt(0.0025),
↪  size=spectra_noisy_sampled.shape)
spectra_noisy_sampled += random_data

np.random.seed(seed)
NaN_index = np.array([True]*int(num_data_points * .8) +
↪  [False]*int(num_data_points * .2))
np.random.shuffle(NaN_index)
NaN_index = NaN_index.reshape(spectra_noisy_sampled.shape)

spectra_noisy_sampled[NaN_index] = np.nan
```

(b)

```
# import kernel packages
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, WhiteKernel,
↪  Matern, ExpSineSquared
```

```
# Get the row with the least amount of na values
nan_per_row = spectra_noisy_sampled.isna().mean(axis=1)
nan_min_row = nan_per_row.idxmin()
min_na_value_sample = spectra_noisy_sampled.loc[nan_min_row]
X = np.arange(1,200)[min_na_value_sample.notna()].reshape(-1, 1)
y = min_na_value_sample.to_numpy()[min_na_value_sample.notna()]
```

```
# Define the k-fold cross validation
def k_fold_cross_validation(kernel, X=X, y=y, k=3, seed=seed):
```

```
    gp = GaussianProcessRegressor(kernel=kernel,
    ↪  n_restarts_optimizer=10)
    K_Fold = KFold(n_splits=k, shuffle=True, random_state=seed)
    mse = cross_val_score(gp, X, y, cv=K_Fold,
    ↪  scoring='neg_mean_squared_error')
    return -mse
```

```
# Use k-fold validations to see which kernel creates the best
↪  scores
mses = []

kernel = Matern() + RBF() + WhiteKernel()
mses.append(k_fold_cross_validation(kernel=kernel, X=X, y=y, k = 3,
↪  seed = seed))

kernel = Matern() + RBF()
mses.append(k_fold_cross_validation(kernel=kernel, X=X, y=y, k = 3,
↪  seed = seed))

kernel = Matern()
mses.append(k_fold_cross_validation(kernel=kernel, X=X, y=y, k = 3,
↪  seed = seed))

kernel = ExpSineSquared() +  Matern() + RBF()
mses.append(k_fold_cross_validation(kernel=kernel, X=X, y=y, k = 3,
↪  seed = seed))

kernel = RBF()
mses.append(k_fold_cross_validation(kernel=kernel, X=X, y=y, k = 3,
↪  seed = seed))
```

```
# Name of each combinations
knames = ["Matern + RBF + WhiteKernel","Matern + RBF",
    "Matern","ExpSineSquared + Matern + RBF","RBF"]
# Plot boxplots of the mses
plt.figure(figsize=(10, 6))
plt.boxplot(mses, labels=knames)
plt.title("Mean Squared Error")
plt.xlabel("Kernels")
plt.ylabel("Values")
plt.savefig("Question_1_part_2a_boxplot.png", dpi=600)
plt.show()
```

```
# Find the smallest MSE and the kernel combinations
mse = [np.mean(mse_group) for mse_group in mses]

```

```
print("The smallest MSE is with kernel combination",knames[np.argmin(mse)])

print("The smallest MSE is",np.min(mse))

```

```
# All wave lengths
X_long = np.linspace(0,199,200).reshape(-1,1)
# Kernel with lowest k-fold cross validation
kernel = Matern() + RBF() + WhiteKernel()
# fit GaussianProcess
gp = GaussianProcessRegressor(kernel=kernel, alpha=5e-10)
gp.fit(X, y)
#Calculate the prediction for i*
y_pred_train = gp.predict(X, return_std=False)
#Calculate the prediction for every wavelength
y_pred_long, sigma = gp.predict(X_long, return_std=True)
```

```
# Plot the results with  the confidence intervals
plt.plot(X_long, y_pred_long, label='Prediction')
plt.fill_between(X_long.squeeze(),y_pred_long - sigma * 1.96,y_pred_long + sigma * 1.96,color='red',
                 alpha=0.2,label='95% Confidence Interval')
plt.scatter(X, y, color='green', label='Actual Absorption')
plt.scatter(X, y_pred_train, color='blue', label='Prediction Points',alpha = 0.5)
plt.legend()
# Adding labels and legend
plt.savefig("Question_1_part_2a_predictions.png", dpi=600)
plt.show()
```

(c)

```
# Consider entire index except the
filtered_spectra = spectra_noisy_sampled.drop(index=nan_min_row)
filtered_components = components_sampled.drop(index=nan_min_row)
filtered_temperatures = temperatures_sampled.drop(index=nan_min_row)

spectra_star = spectra_noisy_sampled.loc[nan_min_row]
components_star = components_sampled.iloc[nan_min_row].to_frame().T
temperatures_star = temperatures_sampled.loc[nan_min_row]

# Create data frame fit for gaussian regression
spectra_longer = pd.DataFrame({'Column': np.repeat(np.arange(0, 199), 39)})
```

```
components_longer = pd.concat([filtered_components] * 199,
↪  ignore_index=True)
temperatures_longer = pd.concat([filtered_temperatures] * 199,
↪  ignore_index=True)
y = filtered_spectra.values.flatten(order='F')
X =
↪  pd.concat([temperatures_longer,components_longer,spectra_longer],axis=1)

# Create data frame fit for gaussian regression

spectra_longer = pd.DataFrame({'1': np.repeat(np.arange(0, 199),
↪  1)})
components_star = pd.concat([components_star] * 199,
↪  ignore_index=True)
temperatures_star = pd.concat([temperatures_star] * 199,
↪  ignore_index=True)
y_star = spectra_star.values.flatten(order='F')
X_star =
↪  pd.concat([temperatures_star,components_star,spectra_longer],axis=1)

# remove na values
indices = np.where(~np.isnan(y))[0]
y = y[indices]
X = X.iloc[indices,:].to_numpy()

# remove na values
indices = np.where(~np.isnan(y_star))[0]
y_star = y_star[indices]
X_star = X_star.iloc[indices,:].to_numpy()
```

```
mses = []

kernel = Matern() + RBF() + WhiteKernel()
mses.append(k_fold_cross_validation(kernel=kernel, X=X, y=y, k = 3,
↪  seed = seed))

kernel = Matern() + RBF()
mses.append(k_fold_cross_validation(kernel=kernel, X=X, y=y, k = 3,
↪  seed = seed))

kernel = Matern()
mses.append(k_fold_cross_validation(kernel=kernel, X=X, y=y, k = 3,
↪  seed = seed))

kernel = ExpSineSquared() +  Matern() + RBF()
mses.append(k_fold_cross_validation(kernel=kernel, X=X, y=y, k = 3,
↪  seed = seed))

```

```
kernel = RBF()
mses.append(k_fold_cross_validation(kernel=kernel, X=X, y=y, k = 3,
↪  seed = seed))
```

```
# Find the smallest MSE and the kernel combinations
mse = [np.mean(mse) for mse in mses]

print("The smallest MSE is with kernel
↪  combination",knames[np.argmin(mse)])

print("The smallest MSE is",np.min(mse))
```

```
knames = ["Matern + RBF + WhiteKernel","Matern + RBF",
    "Matern","ExpSineSquared + Matern + RBF","RBF"]
plt.figure(figsize=(10, 6))
plt.boxplot(mses, labels=knames)
plt.title("Mean Squared Error")
plt.xlabel("Kernels")
plt.ylabel("Values")
plt.grid(True)
plt.savefig("Question_1_part_2b_boxplot.png", dpi=600)
plt.show()
```

```
# Kernel
kernel = Matern() + RBF() + WhiteKernel()
# fit GP
gp_regressor = GaussianProcessRegressor(kernel=kernel, alpha=5e-10)

gp_regressor.fit(X, y)

y_pred_star, sigma = gp_regressor.predict(X_star, return_std=True)

```

```
plt.plot(X_star[:,4], y_pred_star, color='blue', label='Prediction
↪  Points')
plt.scatter(X_star[:,4], y_star, color='green', label='Actual
↪  Absorption')
plt.scatter(X_star[:,4], y_pred_star, color='blue',
↪  label='Prediction Points',alpha = 0.2)
plt.fill_between(X_star[:, 4], y_pred_star - sigma * 1.96,
↪  y_pred_star + sigma * 1.96, color='red',
                alpha=0.2,
                label='95\% Confidence Interval')
plt.legend()
plt.savefig("Question_1_part_2b_predictions.png", dpi=600)
plt.show()
```

```
mse = mean_squared_error(y_star, y_pred_star)
print("The mean squared error is",mse)
```

4

(a)

```
# Create data frame fit for gaussian regression
spectra_longer = pd.DataFrame({'Column': np.repeat(np.arange(0,
↪  199), 40)})
components_longer = pd.concat([components_sampled] * 199,
↪  ignore_index=True)
temperatures_longer = pd.concat([temperatures_sampled] * 199,
↪  ignore_index=True)
y = spectra_noisy_sampled.values.flatten(order='F')
X =
↪  pd.concat([temperatures_longer,components_longer,spectra_longer]
              ,axis=1)

# remove na values
indices = np.where(~np.isnan(y))[0]
y = y[indices]
X = X.iloc[indices,:].to_numpy()
```

```
kernel = RBF() +  WhiteKernel() + Matern()
gp_regressor = GaussianProcessRegressor(kernel=kernel)
gp_regressor.fit(X, y)
```

```
components_test = pd.read_csv('components_test.csv',header = None,
↪  sep = ' ')
spectra_test = pd.read_csv('spectra_test.csv',header=None, sep = '
↪  ')
temperatures_test =
↪  pd.read_csv('temperatures_test.csv',header=None)

n = 100
min_temperatures = np.min(temperatures)
max_temperatures = np.max(temperatures)
temperature_grid = np.linspace(min_temperatures, max_temperatures,
↪  n)
```

```
spectra_longer_test = pd.DataFrame({'Column': np.arange(0, 199)})
components_longer_test =
↪  pd.concat([components_test.loc[0].to_frame().transpose()] * 199
                                    , ignore_index=True)

one_d_array = spectra_test.iloc[0].values.flatten(order='F')
```

```
y_test_1 = pd.DataFrame({'y': one_d_array}).to_numpy().squeeze()
X_test_1 = pd.concat([components_longer_test,spectra_longer_test],axis=1).to_numpy()
```

```
nlpd_mean = []
for i in range(n):

    constant_column = np.full((X_test_1.shape[0], 1), temperature_grid[i])

    X_test_np = np.hstack((constant_column, X_test_1))

    y_pred, sigma = gp_regressor.predict(X_test_np, return_std=True)
    nlpd = -np.log(sigma * np.sqrt(2 * np.pi)) - 0.5 * (((y_pred - y_test_1) ** 2) / (sigma**2))
    nlpd_mean.append(-nlpd.mean())

```

```
plt.figure(figsize=(10, 6))
plt.plot(temperature_grid, nlpd_mean, marker='o', linestyle='-', color='blue')
plt.title('Mean NLPD per Temperature', fontsize=25)
plt.xlabel('Temperature', fontsize=20)
plt.ylabel('Mean NLPD', fontsize=20)
plt.savefig("Question_1_part_2c_tempatures.png", dpi=600)
plt.grid(True)

plt.show()

print("Minimum value for nlpd mean is when the temperature is", temperature_grid[np.argmin(nlpd_mean)])
```

(b)

```
import itertools

n = 10
numbers = np.linspace(0, 1, n)

components_combinations = itertools.product(numbers, repeat=3)

components_combinations = np.array(list(components_combinations))
```

```
spectra_longer_test = pd.DataFrame({'Column': np.arange(0, 199)}).to_numpy()
```

```
temperatures_longer_test =
↪  np.repeat([temperatures_test[0][1]],199).reshape(199,1)
one_d_array = spectra_test.iloc[1].values.flatten(order='F')
y_test_1 = pd.DataFrame({'y': one_d_array}).to_numpy().squeeze()
X_test_1 =
↪  np.hstack([temperatures_longer_test,spectra_longer_test])
```

```
nlpd_mean = []
for i in range(components_combinations.shape[0]):
    constant_column = np.full((X_test_1.shape[0], 3),
    ↪  components_combinations[i])
    X_test_np = np.hstack((X_test_1[:,[0]],constant_column,
    ↪  X_test_1[:,[1]]))
    y_pred, sigma = gp_regressor.predict(X_test_np,
    ↪  return_std=True)
    nlpd=np.log(sigma * np.sqrt(2 * np.pi)) + 0.5 * ((y_pred -
    ↪  y_test_1) ** 2) / sigma**2
    nlpd_mean.append(np.mean(nlpd))
```

```
plt.figure(figsize=(10, 6))
plt.scatter(components_combinations[:, 0], nlpd_mean,
↪  label='Water', alpha=.4)
plt.scatter(components_combinations[:, 1], nlpd_mean,
↪  label='Ethanol', alpha=0.5)
plt.scatter(components_combinations[:, 2], nlpd_mean,
↪  label='2-Propanol', alpha=.5)

plt.title('All Components Vs NLPD Mean')
plt.xlabel('Components Proportions')
plt.ylabel('NLPD Mean')
plt.legend(loc = 'upper left')
plt.savefig("Question_1_part_2_4b_NLPD.png", dpi=600)
plt.show()

print("Minimum value for nlpd mean is when the componts are",
↪  components_combinations[np.argmin(nlpd_mean)])
```

Question 2

1

```
data = pd.read_csv('02524541.csv')
y = data.y.to_numpy()
X = data.iloc[:, 1:].to_numpy()
```

```
plt.hist(y, bins=30, color='blue', edgecolor='black')
plt.title("Distribution of Y Values")
plt.xlabel("Y Values")
plt.ylabel("Frequency")


plt.savefig('Q2_y_hist.png', dpi=600, format='png',
↪  bbox_inches='tight')

plt.show()
```

```
low_corr_columns = correlations[abs(correlations) < 0.2].index
filtered_data = data.drop(columns=low_corr_columns)
X_filtered = filtered_data.iloc[:, 1:].to_numpy()
```

2

```
X_train, X_test, y_train, y_test= train_test_split(X,y,
↪  random_state=seed, test_size=0.3,
                                                   shuffle=True)

correlations = data.corr()['y'].drop('y')
low_corr_columns = correlations[abs(correlations) < 0.2].index
filtered_data = data.drop(columns=low_corr_columns)
X_filtered = filtered_data.iloc[:, 1:].to_numpy()
X_train_filtered, X_test_filtered, y_train_filtered,
↪  y_test_filtered= train_test_split(X_filtered,y,
                                     random_state=seed,
                                     test_size=0.3,
                                     shuffle=True)
kf = KFold(n_splits=5, shuffle=True, random_state=seed)
```

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
```

```
##Random Forest output 1
model_comp = []
model = RandomForestRegressor(n_estimators=100, random_state=seed)
cross_val_scores = cross_val_score(model, X_train, y_train, cv=kf,
↪  scoring='neg_mean_squared_error')
model_comp.append(-cross_val_scores)
print("n=100",np.mean(-cross_val_scores))


model = RandomForestRegressor(n_estimators=350, random_state=seed)
```

```
cross_val_scores = cross_val_score(model, X_train, y_train, cv=kf,
↪  scoring='neg_mean_squared_error')
print("n=350",np.mean(-cross_val_scores))
model_comp.append(-cross_val_scores)


model = RandomForestRegressor(n_estimators=500, random_state=seed)
cross_val_scores = cross_val_score(model, X_train, y_train, cv=kf,
↪  scoring='neg_mean_squared_error')
print("n=500",np.mean(-cross_val_scores))
model_comp.append(-cross_val_scores)


model = RandomForestRegressor(n_estimators=1000, random_state=seed)
cross_val_scores = cross_val_score(model, X_train, y_train, cv=kf,
↪  scoring='neg_mean_squared_error')
print("n=1000",np.mean(-cross_val_scores))
model_comp.append(-cross_val_scores)
```

```
##Random Forest output 1
model_comp_filtered = []
model = RandomForestRegressor(n_estimators=100, random_state=seed)
cross_val_scores = cross_val_score(model, X_train_filtered,
↪  y_train, cv=kf, scoring='neg_mean_squared_error')
model_comp_filtered.append(-cross_val_scores)
print("n=100",np.mean(-cross_val_scores))


model = RandomForestRegressor(n_estimators=350, random_state=seed)
cross_val_scores = cross_val_score(model, X_train_filtered,
↪  y_train, cv=kf, scoring='neg_mean_squared_error')
print("n=350",np.mean(-cross_val_scores))
model_comp_filtered.append(-cross_val_scores)


model = RandomForestRegressor(n_estimators=500, random_state=seed)
cross_val_scores = cross_val_score(model, X_train_filtered,
↪  y_train, cv=kf, scoring='neg_mean_squared_error')
print("n=500",np.mean(-cross_val_scores))
model_comp_filtered.append(-cross_val_scores)


model = RandomForestRegressor(n_estimators=1000, random_state=seed)
cross_val_scores = cross_val_score(model, X_train_filtered,
↪  y_train, cv=kf, scoring='neg_mean_squared_error')
print("n=1000",np.mean(-cross_val_scores))
model_comp_filtered.append(-cross_val_scores)
```

```
labels = ['n=100', 'n=350', 'n=500', 'n=1000']

plt.boxplot(model_comp, labels = labels, widths=0.5)

plt.ylabel('Mean Squared Error')

plt.title('The Number of Trees in the Forest vs Mean Squared
↪  Error')

plt.savefig('Random_Forest_output_1.png', dpi=600)

plt.show()
```

```
labels = ['n=100', 'n=350', 'n=500', 'n=1000']

plt.boxplot(model_comp_filtered, labels = labels, widths=0.5)

plt.ylabel('Mean Squared Error')

plt.title('The Number of Trees in the Forest vs Mean Squared
↪  Error')


plt.savefig('Random_Forest_filtered_output_1.png', dpi=600)

plt.show()

```

```
##Random Forest output 2
model_comp_filtered = []
n=500
model = RandomForestRegressor(n_estimators=n, random_state=seed,
↪  criterion='squared_error')
cross_val_scores = cross_val_score(model, X_train_filtered,
↪  y_train, cv=kf, scoring='neg_mean_squared_error')
model_comp_filtered.append(-cross_val_scores)
print("criterion='squared_error'",np.mean(-cross_val_scores))
model = RandomForestRegressor(n_estimators=n, random_state=seed,
↪  criterion='absolute_error')
cross_val_scores = cross_val_score(model, X_train_filtered,
↪  y_train, cv=kf, scoring='neg_mean_squared_error')
print("criterion='absolute_error'",np.mean(-cross_val_scores))
model_comp_filtered.append(-cross_val_scores)
model = RandomForestRegressor(n_estimators=n, random_state=seed,
↪  criterion='friedman_mse')
cross_val_scores = cross_val_score(model, X_train_filtered,
↪  y_train, cv=kf, scoring='neg_mean_squared_error')
print("criterion='friedman_mse'",np.mean(-cross_val_scores))
```

```
model_comp_filtered.append(-cross_val_scores)
```

```
##Random Forest output 2
model_comp = []
n=500
model = RandomForestRegressor(n_estimators=n, random_state=seed, criterion='squared_error')
cross_val_scores = cross_val_score(model, X_train, y_train, cv=kf, scoring='neg_mean_squared_error')
model_comp.append(-cross_val_scores)
print("criterion='squared_error'",np.mean(-cross_val_scores))
model = RandomForestRegressor(n_estimators=n, random_state=seed, criterion='absolute_error')
cross_val_scores = cross_val_score(model, X_train, y_train, cv=kf, scoring='neg_mean_squared_error')
print("criterion='absolute_error'",np.mean(-cross_val_scores))
model_comp.append(-cross_val_scores)
model = RandomForestRegressor(n_estimators=n, random_state=seed, criterion='friedman_mse')
cross_val_scores = cross_val_score(model, X_train, y_train, cv=kf, scoring='neg_mean_squared_error')
print("criterion='friedman_mse'",np.mean(-cross_val_scores))
model_comp.append(-cross_val_scores)
```

```
labels = ['Squared Error', 'Absolute Error', 'Friedman Mean Squared Error']

plt.boxplot(model_comp_filtered, labels = labels, widths=0.5)

plt.ylabel('Mean Squared Error')

plt.title('Criterion Method vs Mean Squared Error for Filtered Data')

plt.savefig('Random_Forest_filtered_output_2.png', dpi=600)

plt.show()
```

```
plt.boxplot(model_comp, widths=0.5, labels = labels)

plt.ylabel('Mean Squared Error')

plt.title('Criterion Method vs Mean Squared Error')

plt.savefig('Random_Forest_output_2.png', dpi=600)

plt.show()
```

K Nearest Neighbor Regressor

```
from sklearn.neighbors import KNeighborsRegressor

k_values = [3, 4, 5, 6, 7, 8]

mses = []
for k in k_values:
    KNN = KNeighborsRegressor(n_neighbors=k, metric='manhattan')
    mse = -cross_val_score(KNN, X_train, y_train, cv=kf,
    ↪  scoring='neg_mean_squared_error')
    mses.append(mse)
mses2 = [np.mean(mse) for mse in mses]

for i in range(len(k_values)):
    print(k_values[i], mses2[i])

print("Min K value", k_values[np.argmin(mses2)])


```

```
plt.figure(figsize=(10, 6))
plt.boxplot(mses, labels = k_values)
plt.xlabel('K values')
plt.ylabel('Mean Squared Error')
plt.title('MSE Distribution for Different Values of k')
plt.savefig('Knn_output1.png')
plt.show()

```

```
k_values = [3, 4, 5, 6, 7, 8]

mses = []
for k in k_values:
    KNN = KNeighborsRegressor(n_neighbors=k, metric='chebyshev')
    mse = -cross_val_score(KNN, X_train, y_train, cv=kf,
    ↪  scoring='neg_mean_squared_error')
    mses.append(np.mean(mse))
mses2 = [np.mean(mse) for mse in mses]

for i in range(len(k_values)):
    print(k_values[i], mses2[i])

print("Min K value", k_values[np.argmin(mses2)])

```

```
# Plot the box plots for MSEs
plt.figure(figsize=(10, 6))
plt.boxplot([mse_results[k] for k in k_values], labels=k_values)
plt.xlabel('K values')
plt.ylabel('Mean Squared Error')
plt.title('MSE Distribution for Different Values of k')
plt.savefig('Knn_output2.png', dpi = 600)
plt.show()

```

Decision Trees

```
from sklearn.tree import DecisionTreeRegressor
```

```
##Decision Tree output 1
mse_comp = []
model = DecisionTreeRegressor(random_state=seed,criterion='absolute_error'
                              ,splitter='best')
cross_val_scores = cross_val_score(model, X_train
                        , y_train, cv=kf,
                        scoring='neg_mean_squared_error')
print("absolute_error",np.mean(-cross_val_scores))
mse_comp.append(-cross_val_scores)
model = DecisionTreeRegressor(random_state=seed,criterion='squared_error'
                              ,splitter='best')
cross_val_scores = cross_val_score(model, X_train
                        , y_train, cv=kf,
                        scoring='neg_mean_squared_error')
print("squared_error",np.mean(-cross_val_scores))
mse_comp.append(-cross_val_scores)
model = DecisionTreeRegressor(random_state=seed,criterion='friedman_mse'
                              ,splitter='best')
cross_val_scores = cross_val_score(model, X_train
                                   , y_train, cv=kf,
                                   scoring='neg_mean_squared_error')
print("friedman_mse",np.mean(-cross_val_scores))
mse_comp.append(-cross_val_scores)
```

```
labels = ['Squared Error', 'Absolute Error', 'Friedman Mean Squared
Error']

plt.boxplot(mse_comp, labels = labels)

plt.ylabel('Mean Squared Error')
```

```

plt.title('Criterion Method vs Mean Squared Error')

plt.savefig('Decision_Tree_output_1.png', dpi=600,
↪  bbox_inches='tight')

plt.show()
```

```
##Decision Tree output 2
mse_comp = []
model =
↪  DecisionTreeRegressor(random_state=seed,criterion='absolute_error'
,splitter='best')
cross_val_scores = cross_val_score(model, X_train, y_train, cv=kf,
↪  scoring='neg_mean_squared_error')
print("best",np.mean(-cross_val_scores))
mse_comp.append(-cross_val_scores)
model =
↪  DecisionTreeRegressor(random_state=seed,criterion='absolute_error'
,splitter='random')
cross_val_scores = cross_val_score(model, X_train, y_train, cv=kf,
↪  scoring='neg_mean_squared_error')
print("random",np.mean(-cross_val_scores))
mse_comp.append(-cross_val_scores)
```

Neural Network

```
X_train_nn = torch.tensor(X_train, dtype=torch.float32)
y_train_nn = torch.tensor(y_train, dtype=torch.float32)
train_data = TensorDataset(X_train_nn, y_train_nn)
train_data = DataLoader(dataset=train_data, batch_size=10,
↪  shuffle=True)

class Neural_Network(nn.Module):
    def __init__(self, dim):
        nn.Module.__init__(self)
        self.layer1 = nn.Linear(dim, 128)
        self.layer2 = nn.Linear(128, 64)
        self.layer3 = nn.Linear(64, 1)

    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))
        x = F.relu(self.layer3(x))
        return x

model = Neural_Network(49)
```

```

criterion = nn.MSELoss()

Adam = optim.Adam(model.parameters(), lr=0.001)

epochs = 1000

for epoch in range(epochs):
    for X_vals, y_vals in train_data:
        Adam.zero_grad()
        y_pred = model(X_vals)
        loss = criterion(y_pred, y_vals)
        loss.backward()
        Adam.step()
    print(loss)
```

3

```
rf = RandomForestRegressor(n_estimators=350, random_state=seed)

rf.fit(X_train_filtered, y_train_filtered)

y_pred = rf.predict(X_test_filtered)

mse = mean_squared_error(y_test_filtered, y_pred)

print("Mean Squared Error is",mse)
```

```
dt = DecisionTreeRegressor(random_state=seed)

dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

print("Mean Squared Error is",mse)
```

```
knn = KNeighborsRegressor(n_neighbors=7, metric='manhattan')
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error is", mse)
```

```
X_test_nn = torch.tensor(X_test, dtype=torch.float32)
y_test_nn = torch.tensor(y_test, dtype=torch.float32)
test_data = TensorDataset(X_test_nn, y_test_nn)
test_data = DataLoader(dataset=test_data, batch_size=10,
↪  shuffle=False)
loss_count = 0
count = 0
with torch.no_grad():
    for X_vals, y_vals in test_data:
        y_pred = model(X_vals)
        loss = criterion(y_pred, y_vals)
        loss_count += loss.item()
        count +=1

mse = loss_count / count
print("Mean Squared Error is",mse)
```