National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Stephanie Whelan
Student ID: 19140649

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name	Stephanie Whelan
Student ID	19140649
Programme	MSc Top-up Data Analytics
Year	2022
Module	MSc Research Project
Lecturer	Dr. Catherine Mulwa
Submission Due Date	15/08/2022
Project Title	Predicting River Water Quality Parameters using Supervised Machine Learning Techniques: UK
Word Count	2884
Page Count	43

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……

Date: ……14/08/2022……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Predicting River Water Quality Parameters using Supervised Machine Learning Techniques: UK

Stephanie Whelan
19140649

1 Hardware Requirements and Technologies Used

Table 1: Device and Operating Specifications

Processor	AMD Ryzen 7 4700U with Radeon Graphics 2.00 GHz
Installed RAM	16.0 GB (15.4 GB usable)
System Type	64-bit operating system, x64-based processor
Windows Edition	Windows 10 Home
Windows Version	21H1
Windows Operating System Build	19043.1826
Windows Experience	Windows Feature Experience Pack 120.2212.4180.0

The main programming languages that were used to complete this research include R Programming Language and Python. Both technologies will need to be downloaded and configured on a local system to replicate the output. A list of the technologies used is shown below:

- R
- R Studio
- Python
- Anaconda and Jupyter Notebook

2 R Programming Language

R Programming language was used for pre-processing of the data including data cleaning and manipulation.

2.1 Downloading R

Search https://cran.r-project.org/bin/windows/base/ on any web browser and click on 'Download R-4.2.1 for Windows' (if you are running this on a windows computer) and the file will begin to download. The steps taken are shown in Figure 1 - Figure 9 below.

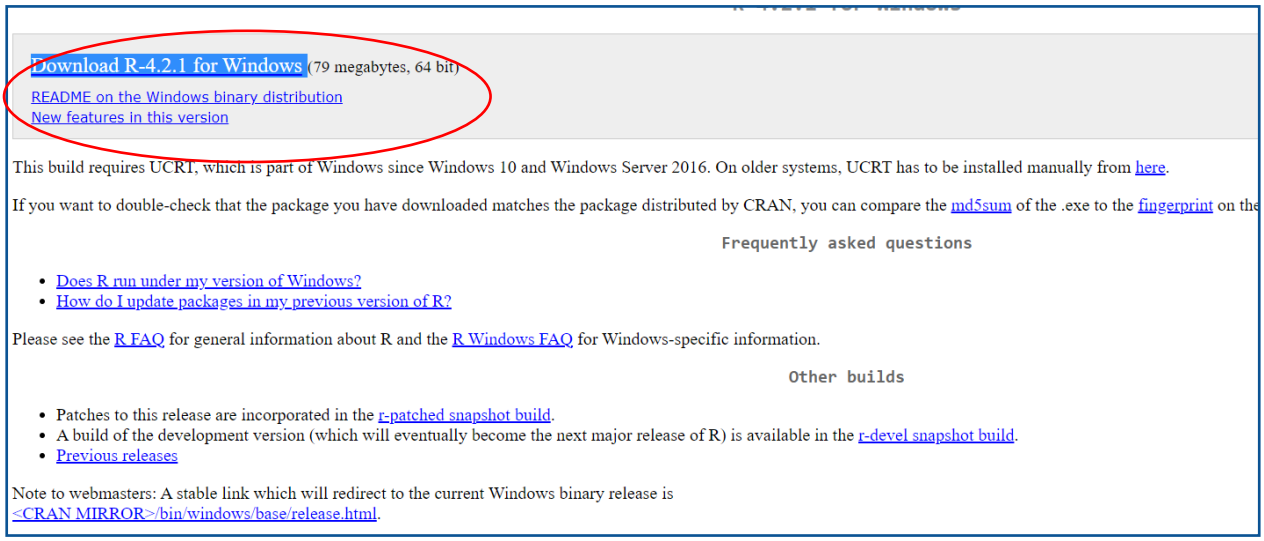


Figure 1

Double click on the downloaded file shown in Figure 2 in the bottom left to begin the installer.

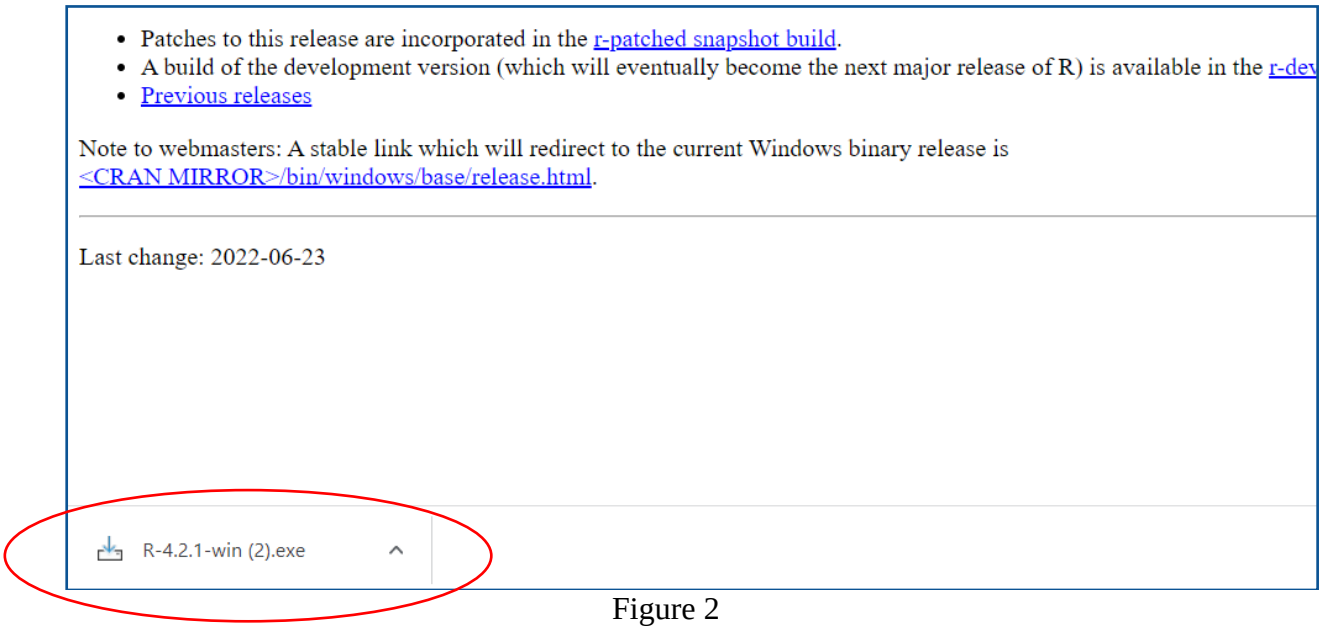


Figure 2

Follow the steps shown in Figure 3 to Figure 9 below to finish installing the software.

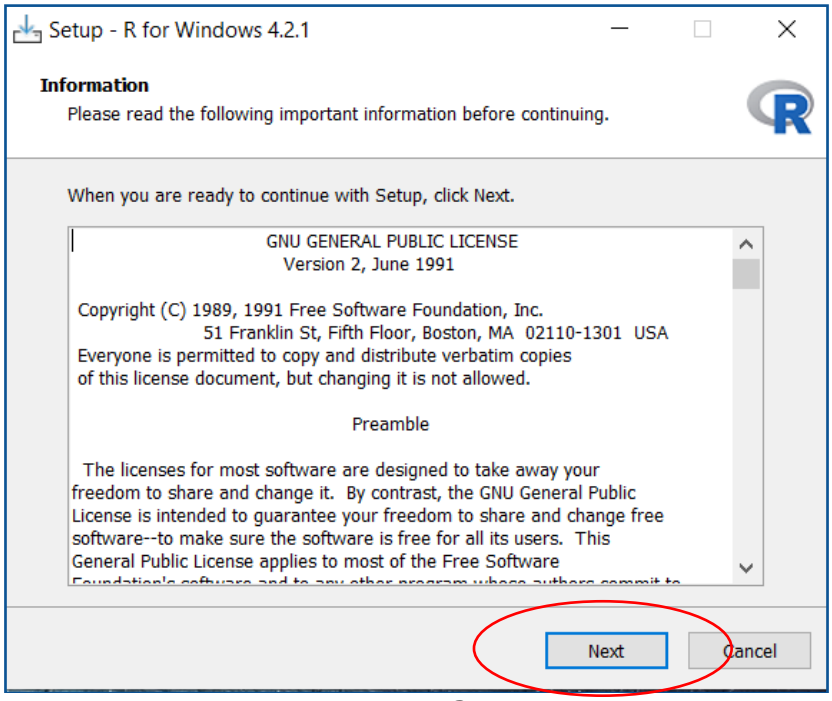


Figure 3

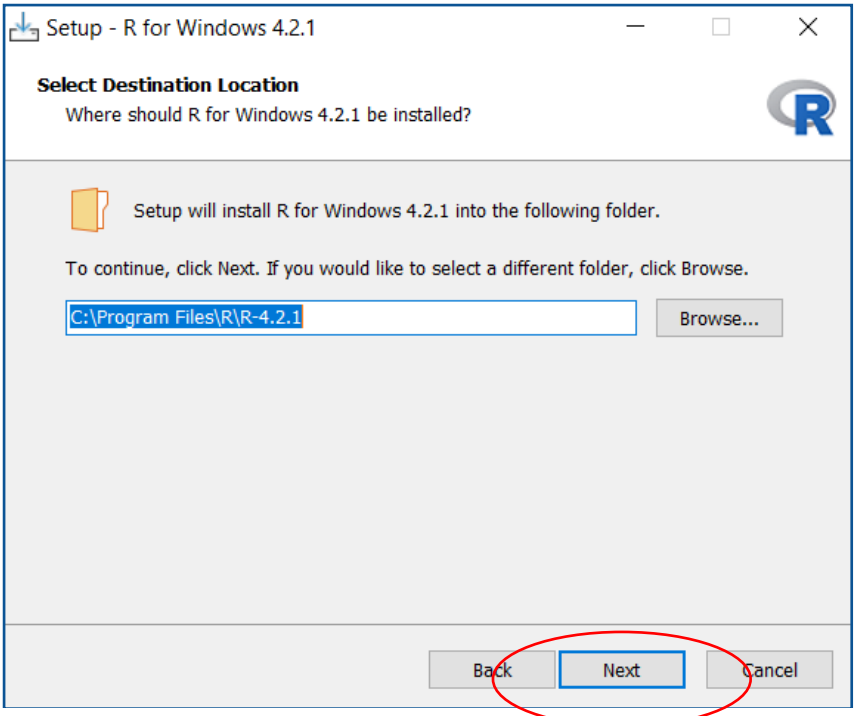


Figure 4

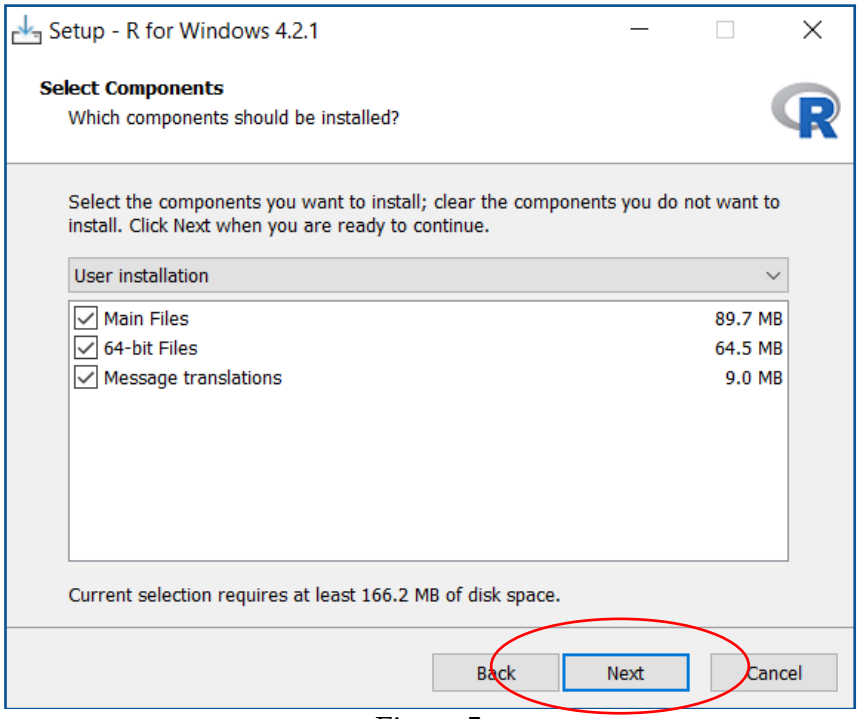


Figure 5

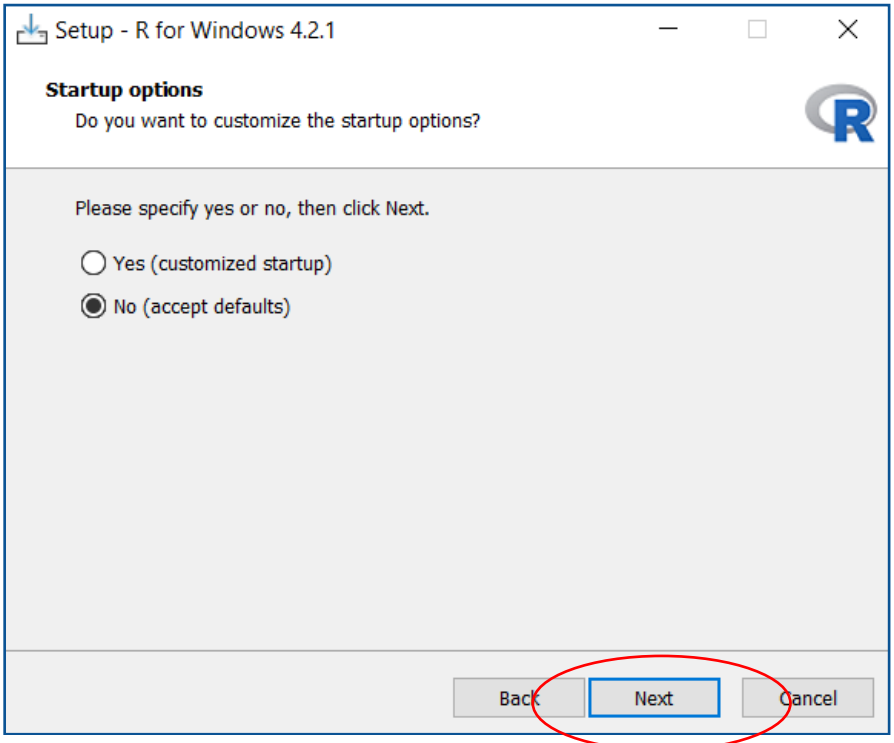


Figure 6

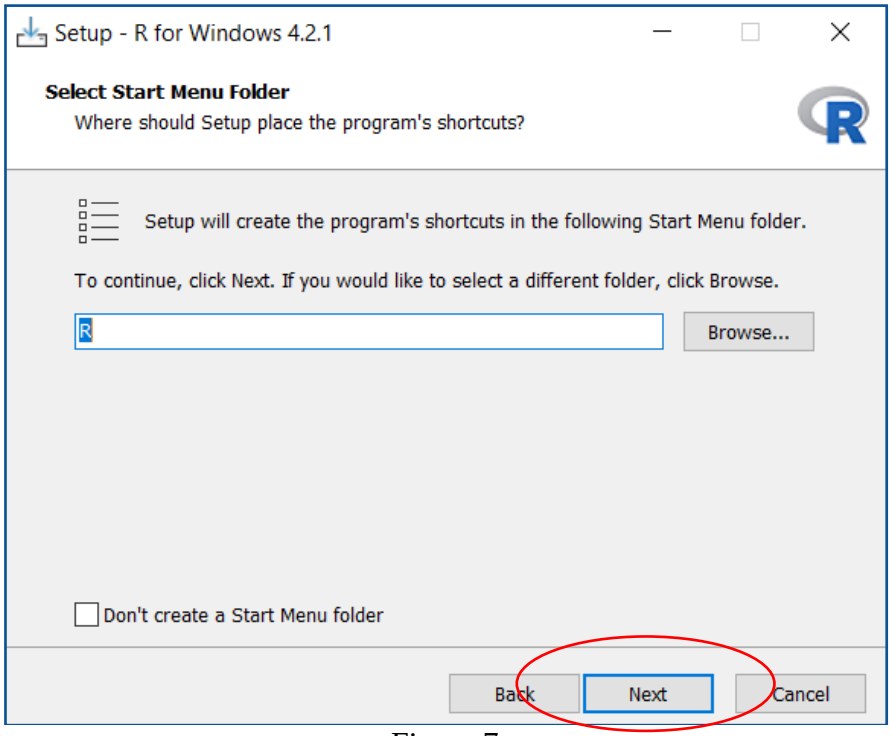


Figure 7

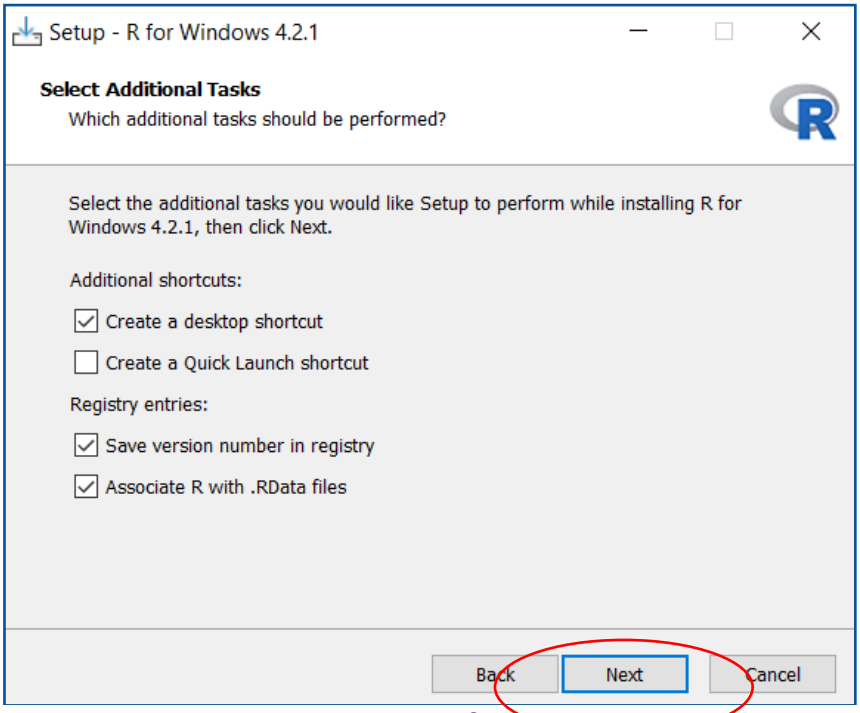


Figure 8

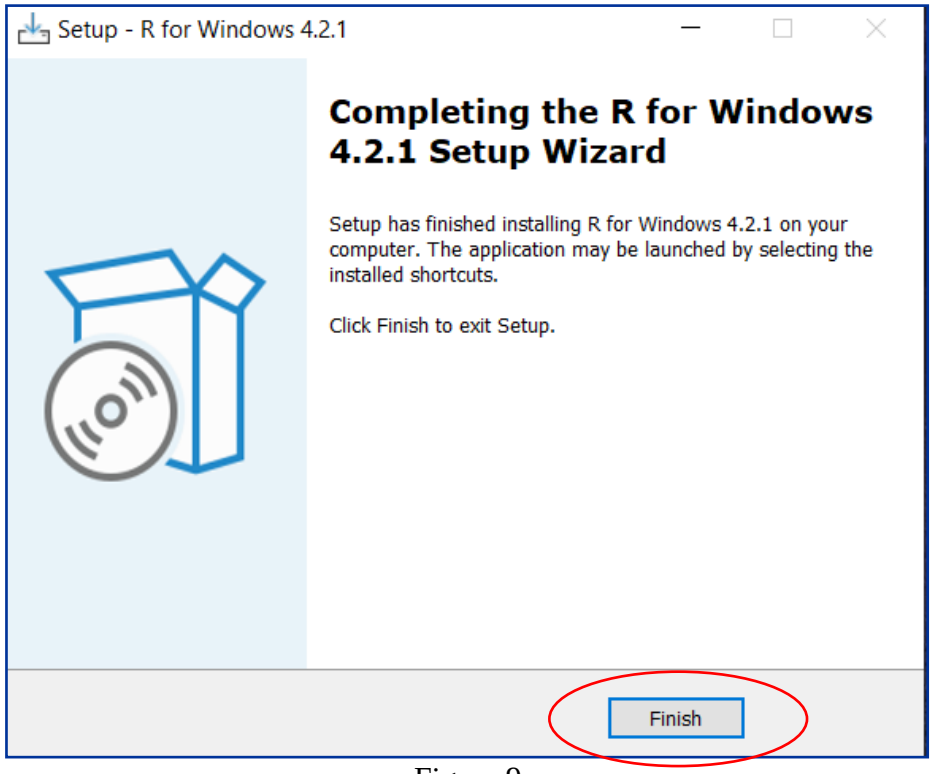


Figure 9

2.2 Downloading R Studio

R Studio is an integrated development environment (IDE) for R. RStudio was downloaded from www.rstudio.com/products/rstudio/download/. The version that was downloaded for the completion of this project was the Free version of RStudio Desktop circled in red on Figure 10 below. The next step was to download the appropriate version of RStudio for the operating system being used. Figure 11 shows the version that was downloaded for the completion of this project circled in red. Figure 12 to Figure 15 shows the steps that were completed to download RStudio.

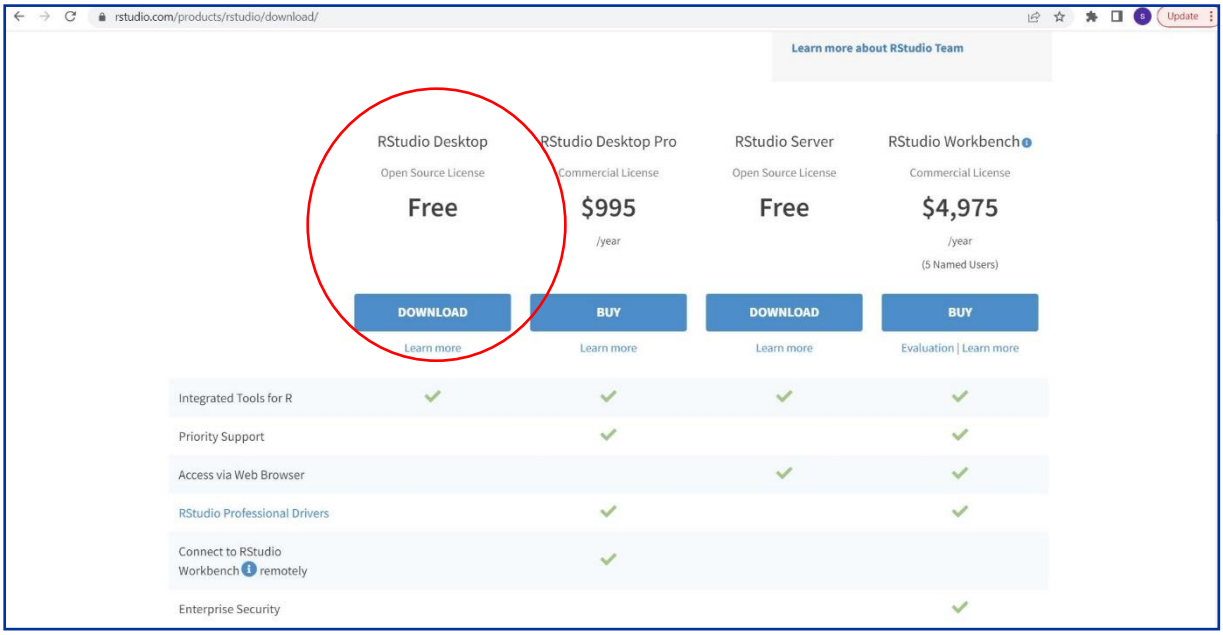


Figure 10: RStudio Desktop Download.

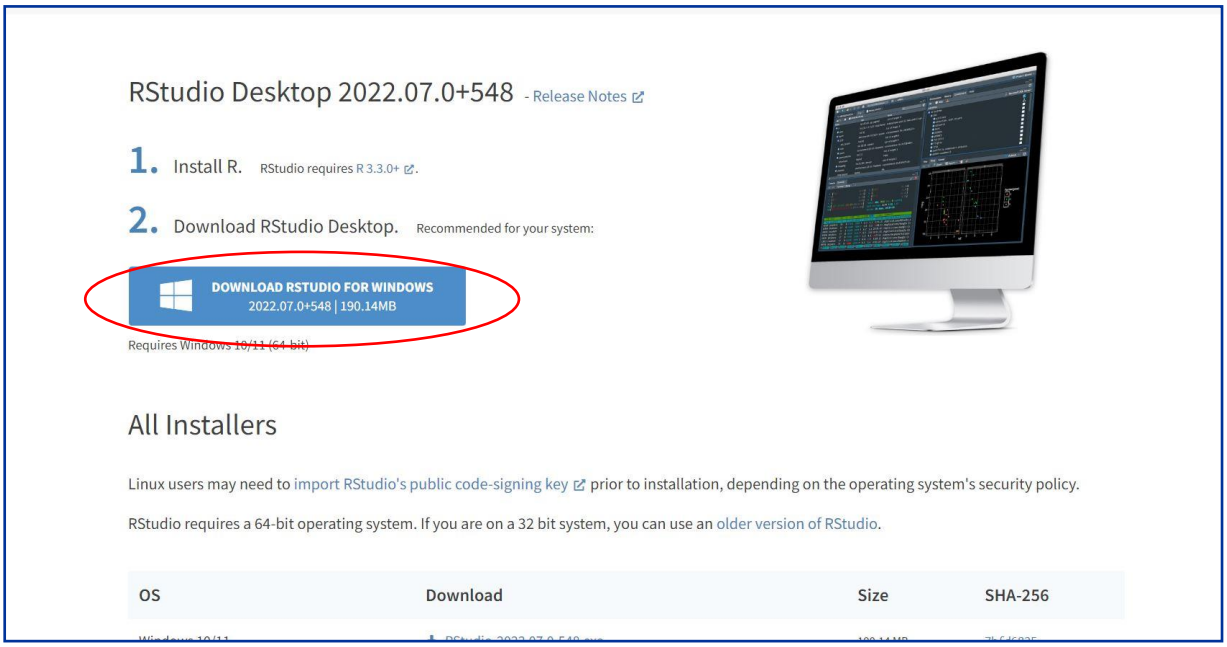


Figure 11: RStudio Desktop Download for Windows

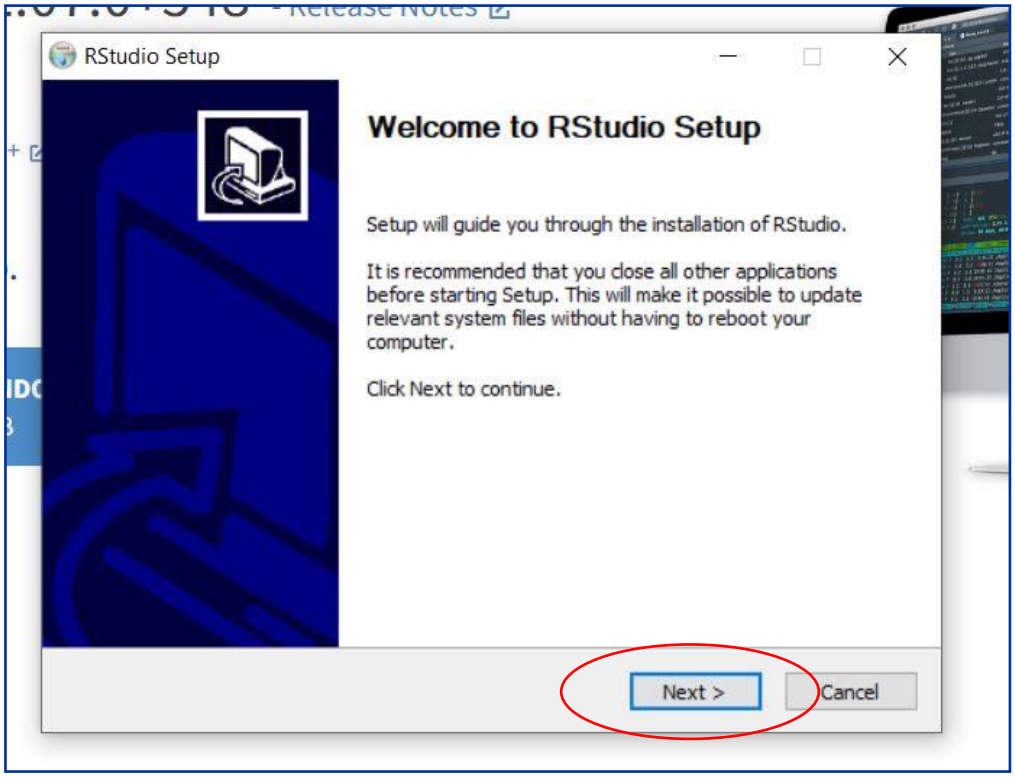


Figure 12

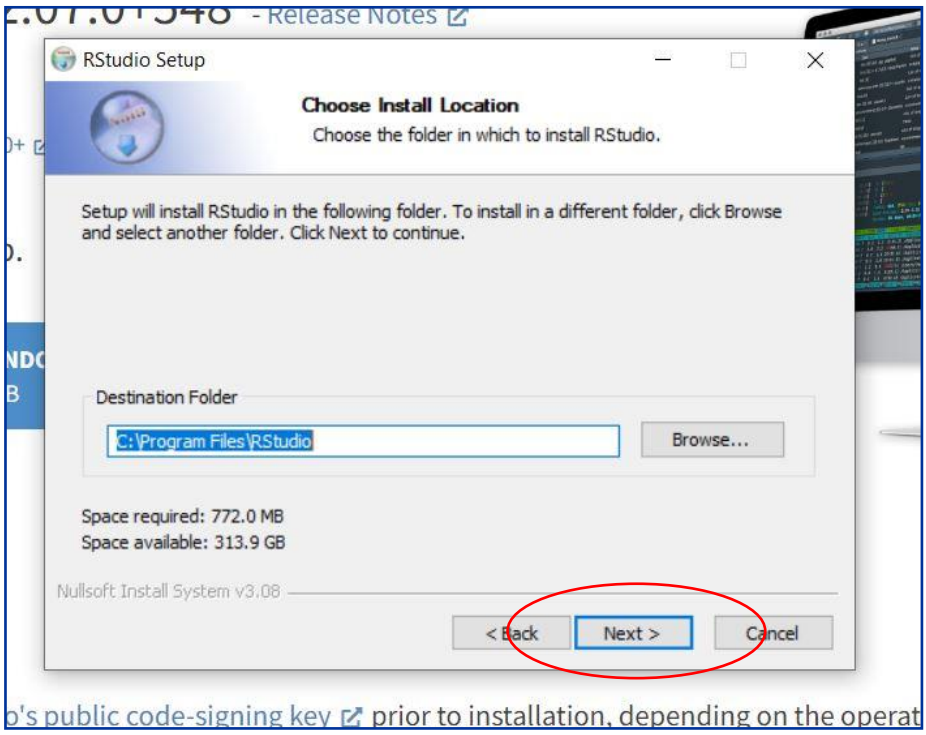


Figure 13

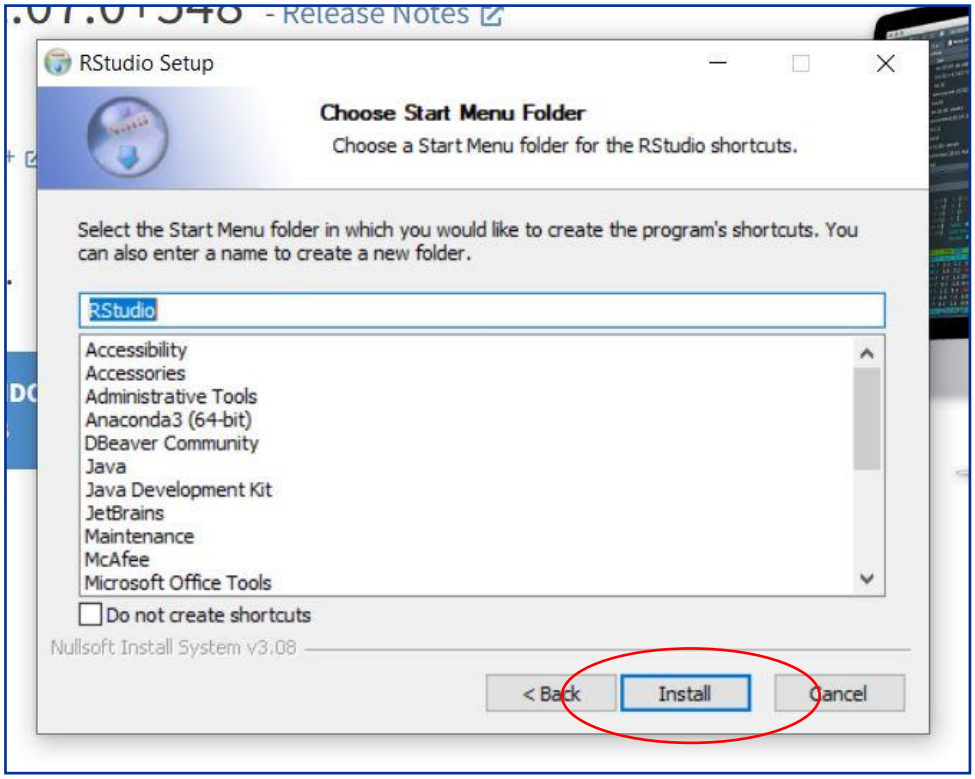


Figure 14

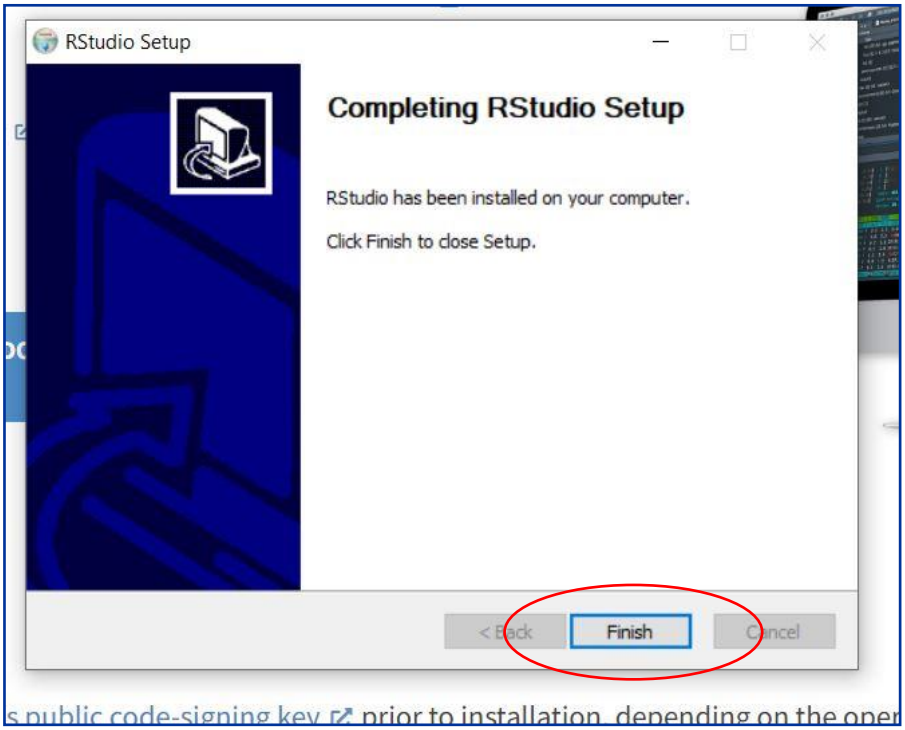


Figure 15

The first file 'Water Quality - Data Clean Up Final' can now be run using RStudio.

2.2.1 Data Pre-processing using RStudio

The following packages shown in Figure 16 were loaded into RStudio to allow for data manipulation, the creation of visualisations and to identify NA values.

```
#loading necessary packages
install.packages("corrplot")
install.packages("dplyr")
install.packages("lubridate")
install.packages("naniar")
install.packages("tidyr")
install.packages("tidyverse")
install.packages("visdat")
library(naniar)
library(ggplot2)
library(tidyr)
library(dplyr)
library(visdat)
library(lubridate)
library(tidyverse)
library(corrplot)
```

Figure 16: Libraries required for data pre-processing

The two datasets that were used for this research include the UK_Lowland_River_Chemistry_data.csv and weather.csv. The datasets were originally opened using excel and they were explored. They were then loaded into RStudio using the read.csv function. These two datasets will need to be placed in the working directory of the machine. The working directory for this project is shown in Figure 17 below.

```
#loading in my 2 datasets
water <-read.csv(file="UK_Lowland_River_Chemistry_data.csv", head=TRUE, sep=",")
weather <- read.csv(file="weather.csv", head=TRUE, sep=",")
```

Figure 17: Loading the datasets into RStudio

Once the datasets were loaded, the data types were viewed, and the NA values were viewed using the *visdat package*. A visualisation showing the amount of missing data in each column and the percentage of missing data in the entire water quality dataset was created. Figures 18 and 19 below show that 46% of the data is missing in the dataset. The sum of NA values in each column was also calculated so that the number of NA values in each column could be viewed.

```
#First the data was opened in excel and viewed, there are quite a lot of Na values
#Check for missing data
#we can see that 46% of the dataset is missing which is a huge amount
visdat::vis_miss(water, warn_large_data = F)

sapply(water, function(x) sum(is.na(x)))
```

Figure 18: Viewing NA values in the Water Quality Dataset.

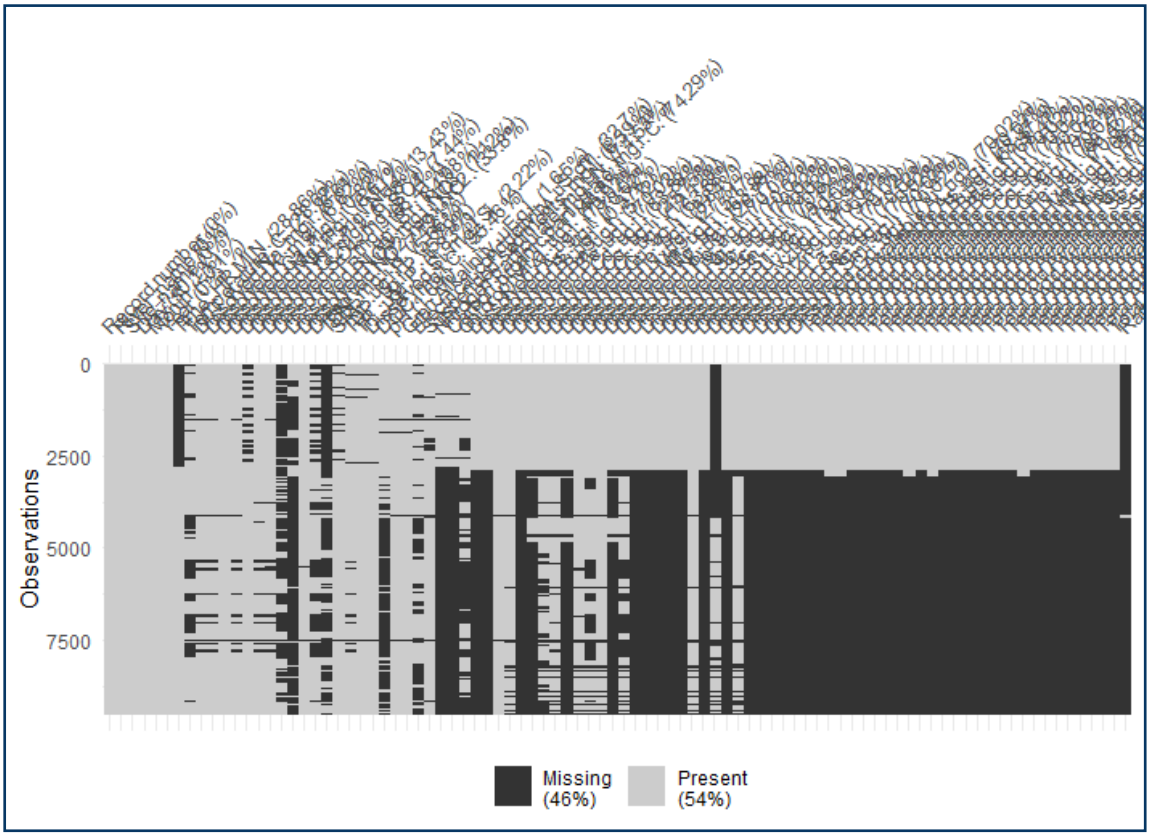


Figure 19: Plotting NA values in the Water Quality Dataset.

As there were 80 variables in the dataset, the decision was made to remove any column with over 6,000 NA values. Once this was done, 8.1% of the dataset contained missing data. The distribution of the columns still containing NA values was calculated and evenly distributed columns had their NA values replaced with the Mean, while skewed columns had their NA values replaced with the Median. The steps taken are shown in Figure 20 below.

```
#replace NA values with the mean on normally distributed columns

water3 <- water
water3$Temperature..C.[is.na(water3$Temperature..C.)] <- mean(water3$Temperature..C., na.rm = TRUE)

#as there is only 1 NA in the day and month columns we will remove that row so that it does not get replaced with the mean

water3 <- water3[!is.na(water3$Month),]
water3 <- water3[!is.na(water3$Day),]

#replace NA values with the median on skewed columns


water3 <- water3 %>%
  mutate_if(is.numeric, function(x) ifelse(is.na(x), median(x, na.rm = T), x))
```

Figure 20: Replacing NA values with the mean and median in selected columns.

A new column was created called Date, which merged the Day, Month Year columns into one date format Y-M-D. This was done so that it could be used to join the water quality dataset to the weather dataset. The steps taken to do this, are shown in Figure 21 below.

```
#create a new data column from the day, month, year columns
water3$date <- as.Date(with(water3, paste(Year, Month, Day,sep="-")), "%Y-%m-%d")
water3$date
```

Figure 21: Creating a new date column.

As shown in Figure 22 below, the date column in the weather dataset was reformatted using the *Lubridate package* so that it was Y-M-D to match the water quality dataset, this is so that the two separate datasets could be merged using the column.

```
#reformatting the date column to Y-M-D so so that it matches the date column in the weather data set
weather$date = ymd(weather$date)
```

Figure 22: Reformatting the Date column in the Water Quality dataset.

The water quality and weather dataset were merged using an inner_join on the date column. The *dplyr package* from the wider *tidyverse package*. The steps taken to do this are shown in Figure 23.

```
#####merge the weather dataset with the water quality dataset#########

library(tidyverse)

merged <- inner_join(weather1,water3,by=c("date"))

str(merged)
```

Figure 23: Mering the weather and water quality dataset.

Figure 24 shows the final dataset was saved to the working directory using write.csv function.

```
#finally write the final dataset to my directory

write.csv(merged,"C:/Users/35386/Desktop/Msc Data Analytics/Data/Used data/merged-final2.csv", row.names = FALSE)
```

Figure 24: Writing the final dataset to the working directory.

3 Python Programming Language

3.1 Downloading Anaconda and Jupyter Notebook

Anaconda was downloaded from https://www.anaconda.com/ which is shown in Figure 25. The version downloaded is suitable for a Windows machine.

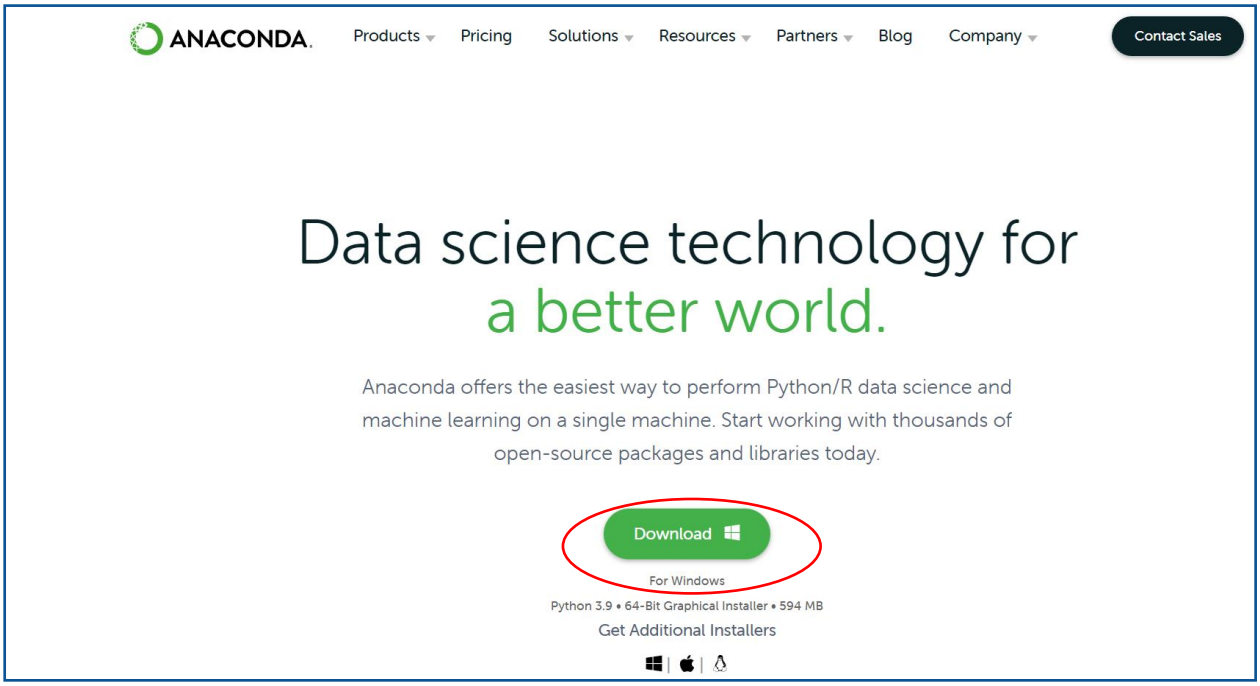


Figure 25

Once downloaded, double click the file in the bottom left corner of the screen to begin the installer shown in Figure 26.

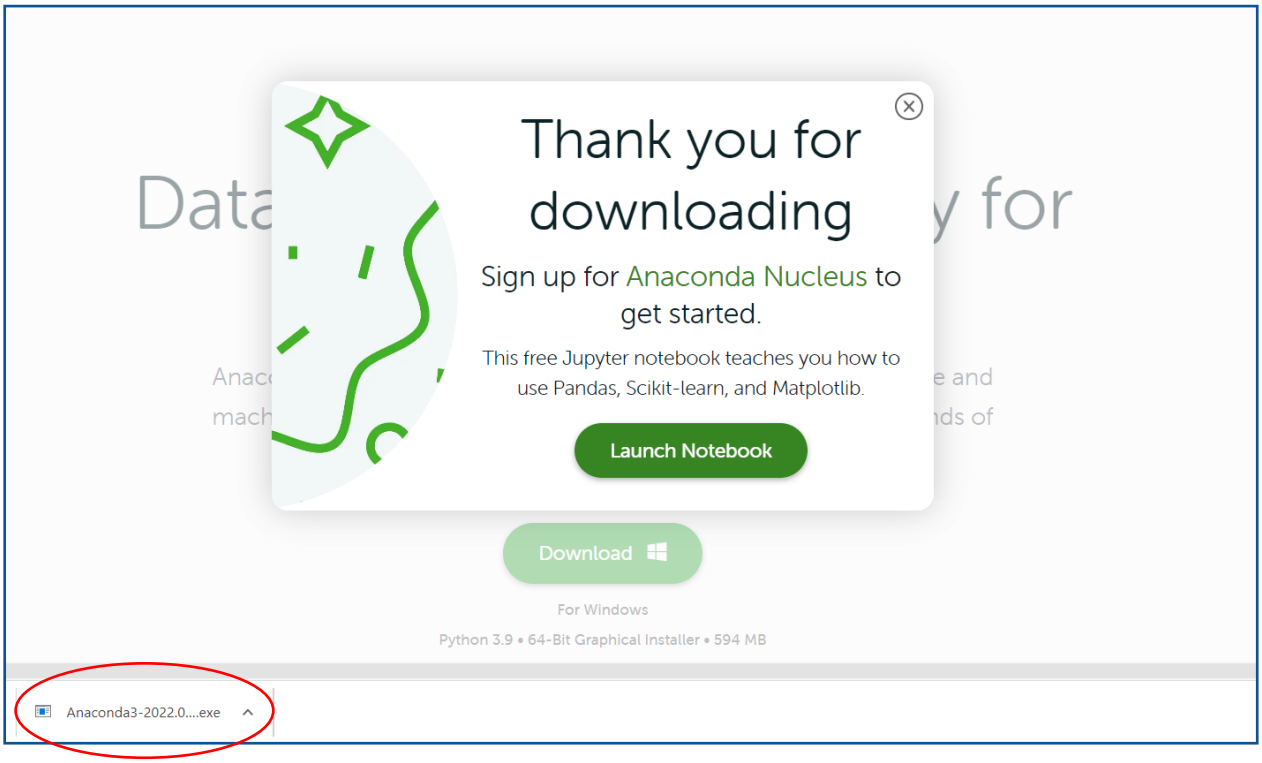


Figure 26

Figures 27 - 33 show the steps taken to download Anaconda.

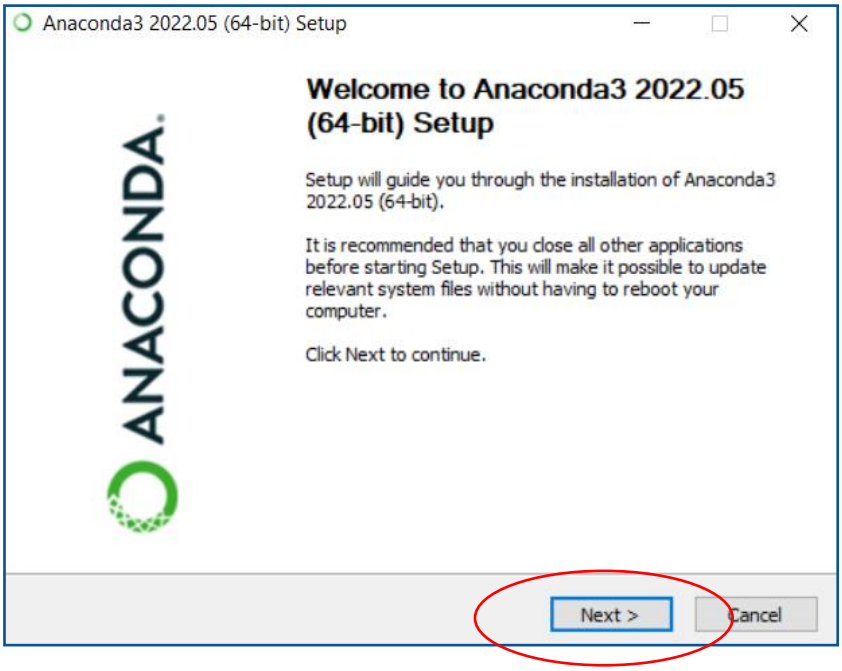


Figure 27

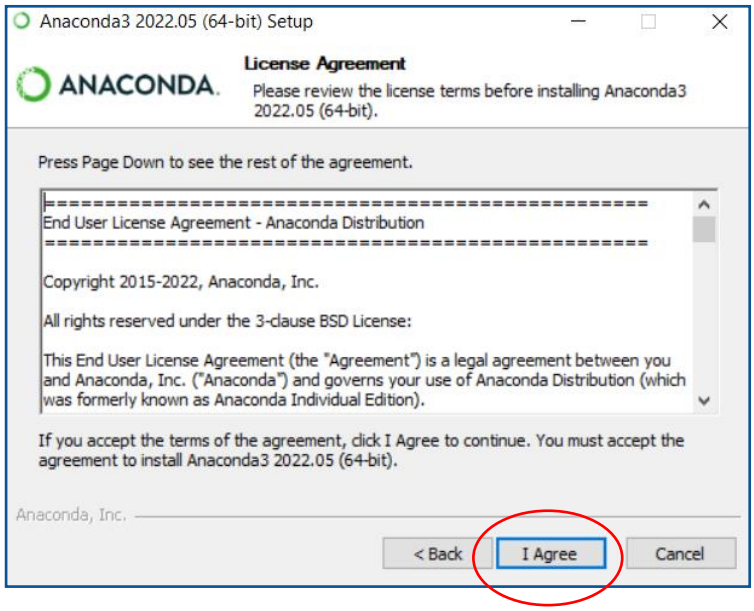


Figure 28

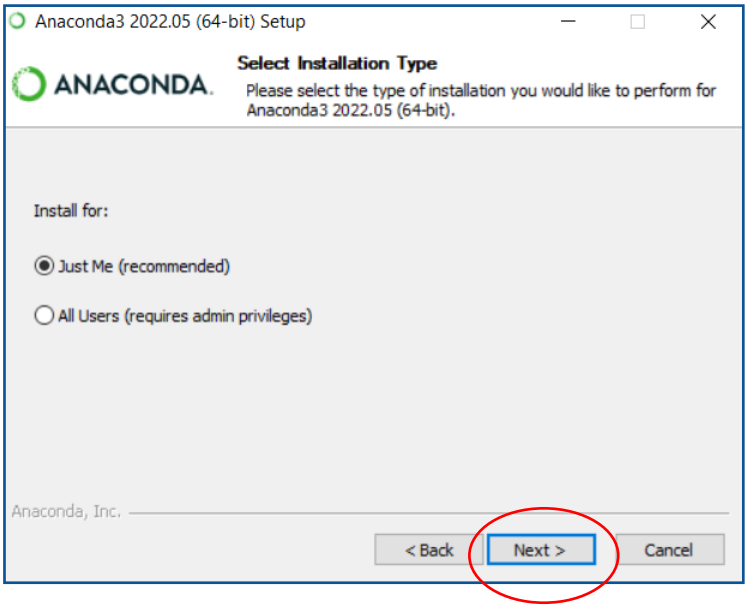


Figure 29

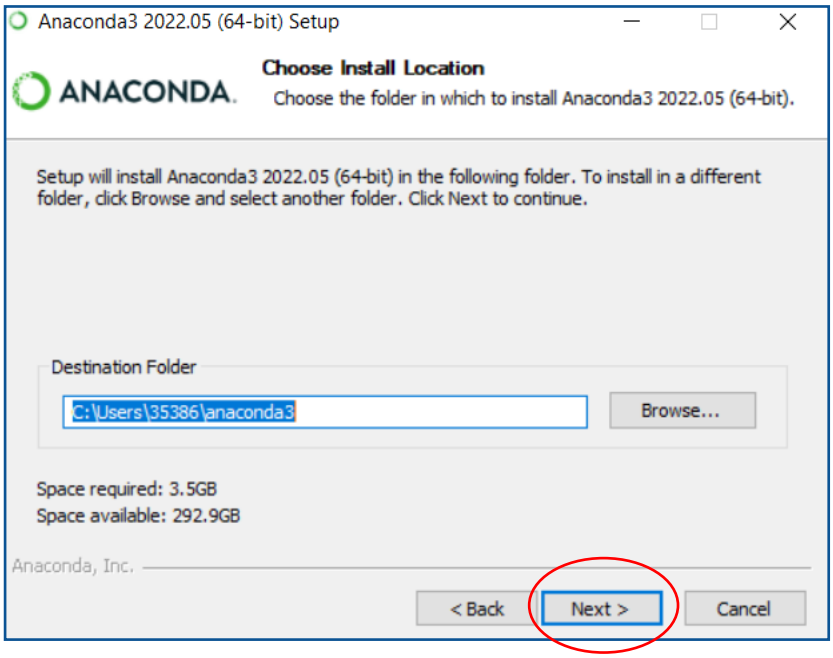


Figure 30

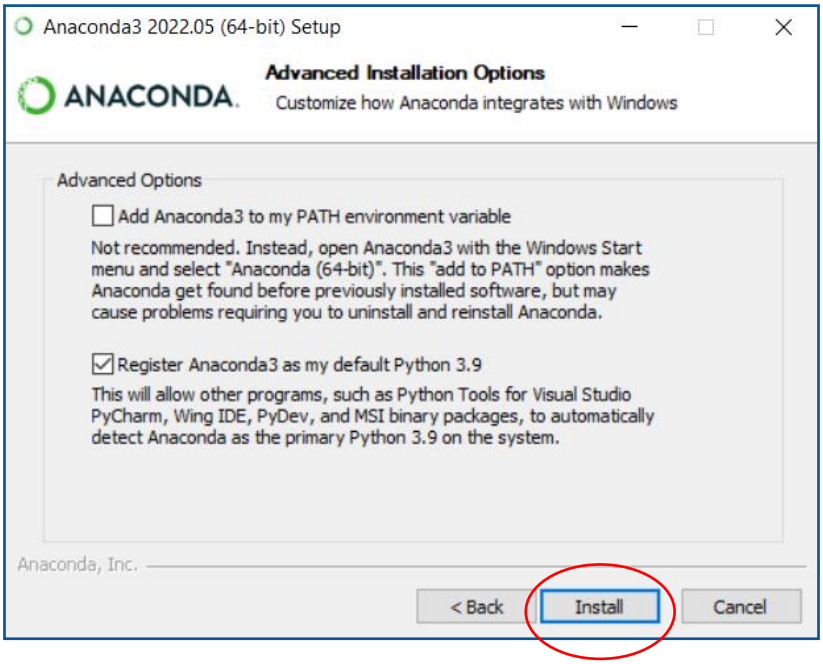


Figure 31

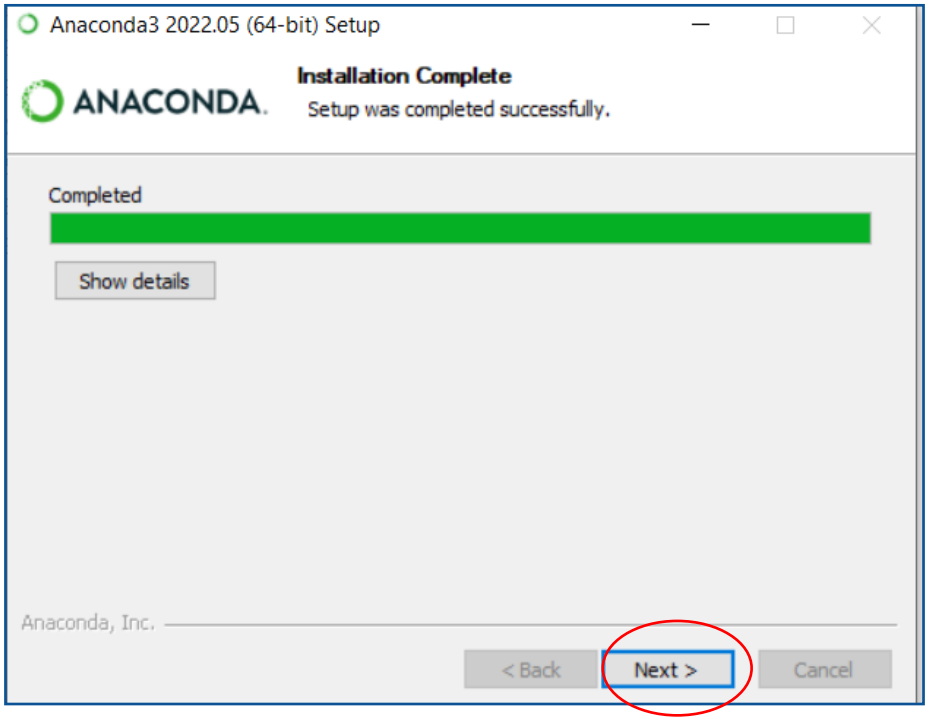


Figure 32

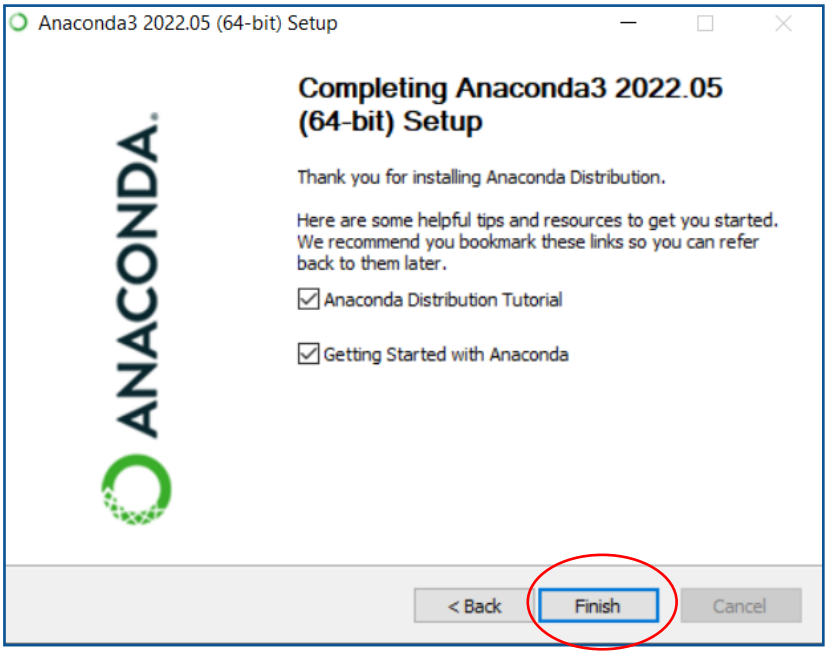


Figure 33

Jupyter Notebook can be accessed from the Anaconda Navigator shown in Figure 34 below.

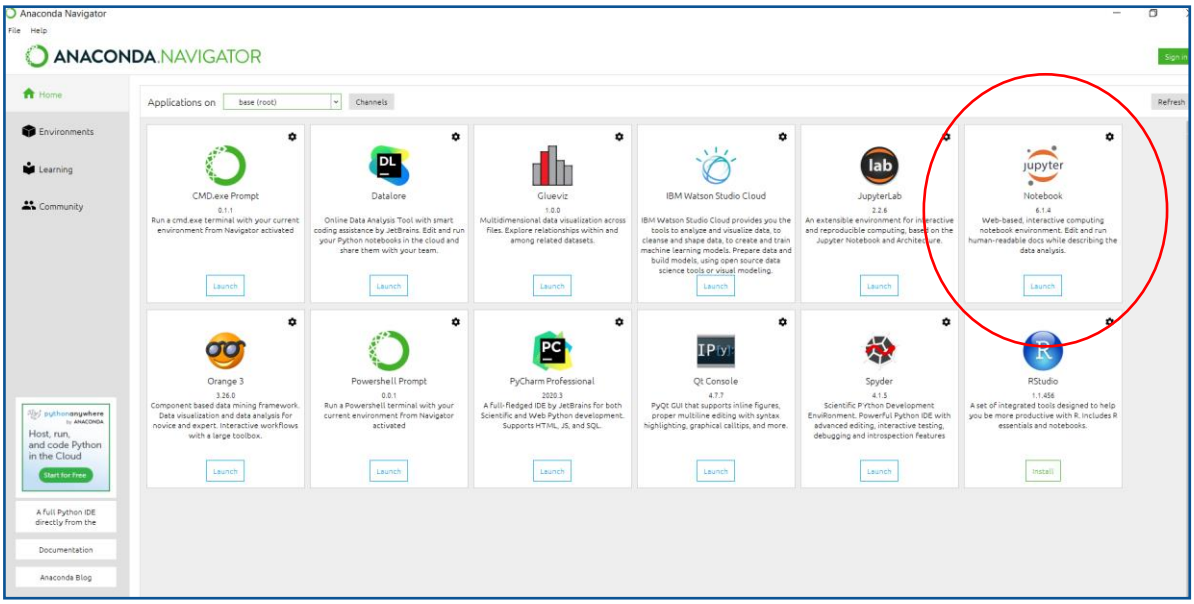


Figure 34

3.2 Modelling using Jupyter Notebook

Once the RStudio code above is run, a new dataset named **merged-final2.csv** will be saved into the relevant working directory. This is the dataset that will be used for further

exploratory data analysis and modelling. There are five Jupyter Notebook files for this research, each one has the code to run each of the five machine learning models. Each model was used to predict four water quality parameters. To replicate this analysis, you will need to replace the parameter within each file - this is explained in further detail below.

The five files that need to be uploaded to Jupyter Notebook and ran are listed below:

1. WATER QUALITY - EDA & DT FINAL.ipynb
2. WATER QUALITY - Random Forest FINAL.ipynb
3. WATER QUALITY - XGBOOST FINAL.ipynb
4. WATER QUALITY - SVM FINAL.ipynb
5. WATER QUALITY - MLR FINAL.ipynb

3.2.1 Modelling Decision Trees

The required libraries for building a Decision Tree model are listed below in Figure 35. The scikit-learn package was used for building the model, evaluating the model, scaling the data and performing Gridsearch.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot
```

Figure 35: Libraries required for implementing the model.

Figure 36 shows the working directory used for this research. Figure 37 shows the data being loaded into Python.

```
import os

os.getcwd()

os.chdir('C:/Users/35386/Desktop/Msc Data Analytics/Data/Used data')

os.getcwd()
```

Figure 36: Set the correct working directory.

```
# Load the data into the notebook

WQ = pd.read_csv("merged-final2.csv")
print(WQ)
```

Figure 37: Load the dataset from the working directory using pandas.

The data was first split into input and output columns using the parameter Dissolved Sodium. The Data was then scaled as the dataset contains many different measurements. The data was then split into 80% training and 20% testing data. Finally, the model was fit using DecisionTreeRegressor from scikit-learn. The steps taken are shown in Figure 38.

```
# split data into input and output columns
X = WQ.drop('Dissolved.Na..mg.l.', axis=1)
y = WQ.loc[:,['Dissolved.Na..mg.l.']]

#scale the data
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)

#Split the data into 80% training data and 20% test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)

#Build the first model - Decision Tree Regression
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X_train, y_train)
```

Figure 38: Splitting, Scaling and building the decision tree model.

The model was then used to predict on the test data and the model was evaluated using the Mean Absolute Error, Mean Squared Error, Root Mean Square Error and R-Squared. The steps taken are shown in Figure 39.

```
#Predict using the test data
y_pred = regressor.predict(X_test)

#evaluate the model
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R Squared:',metrics.r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 0.06418450414621879
Mean Squared Error: 0.02781091559528927
Root Mean Squared Error: 0.16676605048776946
R Squared: 0.9735582890786211
```

Figure 39: Evaluating the decision tree model.

Hyper parameter tuning using GridSearchCV was applied to the model to identify the best performing parameters. As shown in Figure 40 the best performing max depth is 10 and the minimum sample split is 30. Figure 41 shows the new model being built with the best

performing parameters and finally Figure 42 shows the evaluation of the new model. The feature importance of the new model was calculated, and the results are shown in Figure 43. A plot of the actual Vs predicted values for the new model is shown in Figure 44.

```
#Apply Hyper parameter tuning
from sklearn.model_selection import GridSearchCV

model = DecisionTreeRegressor()

gs = GridSearchCV(model,
                  param_grid = {'max_depth': range(1, 11),
                                'min_samples_split': range(10, 60, 10)},
                  cv=5,
                  n_jobs=1,
                  scoring='neg_mean_squared_error')

gs.fit(X_train, y_train)

print(gs.best_params_)
print(-gs.best_score_)
```

```
{'max_depth': 10, 'min_samples_split': 30}
0.029788003659817902
```

Figure 40: Applying Hyper parameter tuning using GridsearchCV

```
#Apply the results to the new model
new_model = DecisionTreeRegressor(max_depth=10,
                                  min_samples_split=30)

new_model.fit(X_train, y_train)
```

```
DecisionTreeRegressor(max_depth=10, min_samples_split=30)
```

Figure 41: Applying the results of hyper parameter tuning to a new model.

```
#Predict using the new model
y_pred = new_model.predict(X_test)
```

```
#evaluate the new model
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R Squared:',metrics.r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 0.0716420975079898
Mean Squared Error: 0.029080893083810802
Root Mean Squared Error: 0.17053120853325002
R Squared: 0.9723508359290443
```

Figure 42: Evaluating the new model.

```
reg.feature_importances_
```

```
array([0.00041914, 0.0007986 , 0.00046427, 0.00070981, 0.00042346,
       0.00619081, 0.00107899, 0.23710613, 0.00620573, 0.00965549,
       0.35744473, 0.14684786, 0.00220003, 0.04130292, 0.00075374,
       0.00366541, 0.05697229, 0.00061634, 0.09096335, 0.00279883,
       0.00241785, 0.00126937, 0.00653843, 0.01254657, 0.0052825 ,
       0.00482802, 0.00049932])
```

```
# Get numerical feature importances
WQ_list = list(WQ.columns)
importances = list(new_model.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 3)) for feature, importance in zip(WQ_list, importances)]
# Sort the feature importances
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable: Dissolved.Mg..mg.l.  Importance: 0.953
Variable: Dissolved.Na..mg.l.  Importance: 0.014
Variable: Gran.Alkalinity..uEq.l. Importance: 0.006
Variable: Dissolved.Ca..mg.l.  Importance: 0.005
Variable: Dissolved.Ba..ug.l.  Importance: 0.004
Variable: Suspended.sediments..mg.l. Importance: 0.003
Variable: Dissolved.B..ug.l.   Importance: 0.003
Variable: Dissolved.Fe..ug.l.  Importance: 0.003
Variable: Dissolved.Ni..ug.l.  Importance: 0.003
Variable: Dissolved.Mn..ug.l.  Importance: 0.002
Variable: year                 Importance: 0.001
Variable: Dissolved.Cl..mg.l.  Importance: 0.001
Variable: Dissolved.Li..ug.l.  Importance: 0.001
Variable: sunshine             Importance: 0.0
Variable: mean_temp            Importance: 0.0
Variable: precipitation        Importance: 0.0
Variable: month                Importance: 0.0
Variable: day                  Importance: 0.0
Variable: Temperature..C.      Importance: 0.0
Variable: Dissolved.K..mg.l.   Importance: 0.0
Variable: Dissolved.SO4..mg.l.SO4. Importance: 0.0
Variable: Dissolved.NO3..mg.l.NO3. Importance: 0.0
Variable: TDP..ug.l.P.         Importance: 0.0
Variable: pH                   Importance: 0.0
Variable: Electrical.conductivity..uS.cm. Importance: 0.0
Variable: Dissolved.Cr..ug.l.  Importance: 0.0
Variable: Dissolved.Sr..ug.l.  Importance: 0.0
```

Figure 43: Calculating the feature importance for the new model.

```
## plot actual Vs predicted - line chart
plt.figure(figsize=(15,8))
pyplot.plot(y_test, label='Expected')
pyplot.plot(y_pred, label='Predicted')
pyplot.legend()
pyplot.show()
```

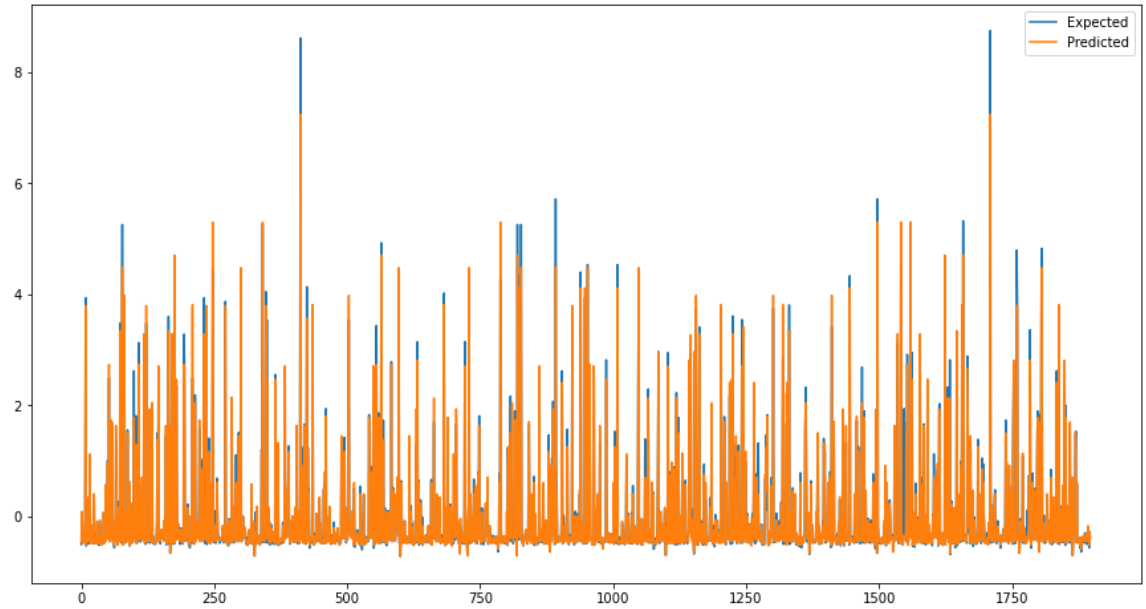


Figure 44: Plotting the actual Vs Predicted values for Decision Tree Model.

Figure 38 to 44 details the decision tree model predicting the parameter 'Dissolved Sodium'. To find the results for the three remaining parameters they will need to replace 'Dissolved Sodium' in the file and the file run again.

```
# split data into input and output columns
X = WQ.drop('Dissolved.Na..mg.l.', axis=1)
y = WQ.loc[:,['Dissolved.Na..mg.l.']]
```

Figure 45

The location where the variable name will need to be replaced is shown above in Figure 45.

3.2.1.1 Testing Parameters

The parameters to replace Dissolved Sodium in the file are Dissolved Nitrate, Gran Alkalinity and Electrical Conductivity. Their variable names in the dataset are listed below in Table 1 and this is the name that should be used in the code:

Table 1: Variables that should be used to replace Dissolved Sodium in the code.

Dissolved.NO3..mg.l.NO3.

Gran.Alkalinity..uEq.l.
Electrical.conductivity..uS.cm.

3.2.2 Modelling Random Forest

Random Forest was applied to the model using the scikit-learn library. The data was split into training and testing datasets, scaled and evaluated also using the scikit-learn library. Hyper parameter tuning was carried out using RandomizedSearchCV from the scikit-learn library. The steps taken are shown in Figure 47 - Figure 51.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
import sklearn.metrics as metrics
```

Figure 46: Libraries required for implementing the model.

Run the code to load the data into Jupyter Notebook using pandas. Split the data into output and input columns, scale the data and then split the data into 80% training and 20% testing.

```
import os

os.getcwd()

os.chdir('C:/Users/35386/Desktop/Msc Data Analytics/Data/Used data')

os.getcwd()
```

```
# Load the Pandas libraries with alias 'pd'

WQ = pd.read_csv("merged-final2.csv")

print(WQ)
```

```
# split data into input and output columns
#split the dataset into 75% training and 25% testing
X = WQ.drop('Dissolved.Na..mg.l.', axis=1)
y = WQ.loc[:,['Dissolved.Na..mg.l.']]
```

```
#scale the data
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)
```

```
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size = 0.20, random_state = 42)
```

Figure 47: Splitting and Scaling the data

Hyper parameter tuning was carried out on the Random Forest model to achieve a higher performance. The best parameters for Max Features, Max Depth, Minimum Sample Split, Minimum Sample Leaf and Bootstrap were calculated and are shown in Figure 48. The best performing parameters were then used when fitting the model as shown in Figure 49. The model was then evaluated as shown in Figure 50 – Figure 51.

```
#hyperparameter tuning using gridsearch

n_estimators = [5,20,50,100] # number of trees in the forest
max_features = ['auto', 'sqrt'] # number of features in consideration at every split
max_depth = [int(x) for x in np.linspace(10, 120, num = 12)] # maximum number of levels allowed in each decision tree
min_samples_split = [2, 6, 10] # minimum sample number to split a node
min_samples_leaf = [1, 3, 4] # minimum sample number that can be stored in a leaf node
bootstrap = [True, False] # method used to sample data points

random_grid = {'n_estimators': n_estimators,

'max_features': max_features,

'max_depth': max_depth,

'min_samples_split': min_samples_split,

'min_samples_leaf': min_samples_leaf,

'bootstrap': bootstrap}
```

```
## Importing Random Forest regressor from the sklearn.ensemble
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
```

```
from sklearn.model_selection import RandomizedSearchCV
rf_random = RandomizedSearchCV(estimator = rf,param_distributions = random_grid,
               n_iter = 100, cv = 5, verbose=2, random_state=35, n_jobs = -1)
```

```
rf_random.fit(train_X, train_y)
```

Figure 48: Hyper Parameter Tuning using gridsearch.

```
#print the best performing parameters
print ('Random grid: ', random_grid, '\n')
# print the best parameters
print ('Best Parameters: ', rf_random.best_params_, ' \n')
```

```
Random grid:  {'n_estimators': [5, 20, 50, 100], 'max_features': ['auto', 'sqrt'], 'max_depth': [10, 20, 30, 40, 50, 60, 70,
80, 90, 100, 110, 120], 'min_samples_split': [2, 6, 10], 'min_samples_leaf': [1, 3, 4], 'bootstrap': [True, False]}

Best Parameters:  {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth':
90, 'bootstrap': True}
```

```
#model with the best parameters used
randmf = RandomForestRegressor(n_estimators = 100, min_samples_split = 2, min_samples_leaf= 1, max_features = 'auto', max_dep
randmf.fit(train_X, train_y.ravel())
```

```
RandomForestRegressor(max_depth=120)
```

```
predictions = randmf.predict(test_X)
```

Figure 49: Using the model with the best performing parameters.

```
#evaluate the model
import sklearn.metrics as metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(test_y, predictions))
print('Mean Squared Error:', metrics.mean_squared_error(test_y, predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(test_y, predictions)))
print('R Squared:',metrics.r2_score(test_y, predictions))
```

```
Mean Absolute Error: 0.04428426443552525
Mean Squared Error: 0.015410766706340158
Root Mean Squared Error: 0.12414010917644691
R Squared: 0.9850853230519386
```

```
# Get numerical feature importances
importances = list(randmf.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 3)) for feature, importance in zip(WQ_list, importances)]
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# Print out the feature and importances
[print('Variable: {:30} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable: Dissolved.Mg..mg.l.            Importance: 0.928
Variable: Dissolved.Na..mg.l.            Importance: 0.026
Variable: pH                             Importance: 0.014
Variable: Dissolved.Cl..mg.l.            Importance: 0.005
Variable: Electrical.conductivity..uS.cm. Importance: 0.003
Variable: Dissolved.Cr..ug.l.            Importance: 0.003
Variable: Dissolved.K..mg.l.             Importance: 0.002
Variable: Dissolved.Ca..mg.l.            Importance: 0.002
Variable: Dissolved.NO3..mg.l.NO3.       Importance: 0.002
Variable: Suspended.sediments..mg.l.     Importance: 0.002
Variable: Dissolved.B..ug.l.             Importance: 0.002
Variable: Dissolved.Mn..ug.l.            Importance: 0.002
Variable: Temperature..C.                Importance: 0.001
Variable: Dissolved.SO4..mg.l.SO4.       Importance: 0.001
Variable: TDP..ug.l.P.                   Importance: 0.001
Variable: TP..ug.l.P.                    Importance: 0.001
Variable: Gran.Alkalinity..uEq.l.        Importance: 0.001
Variable: Dissolved.Ba..ug.l.            Importance: 0.001
Variable: Dissolved.Fe..ug.l.            Importance: 0.001
Variable: Dissolved.Li..ug.l.            Importance: 0.001
Variable: sunshine                       Importance: 0.0
Variable: mean_temp                      Importance: 0.0
Variable: precipitation                  Importance: 0.0
Variable: month                          Importance: 0.0
Variable: day                            Importance: 0.0
Variable: year                           Importance: 0.0
Variable: Dissolved.Ni..ug.l.            Importance: 0.0
```

Figure 50: Evaluating the model and finding feature importance.

```
#plot actual Vs predicted - line chart

plt.figure(figsize=(15,8))
pyplot.plot(test_y, label='Expected')
pyplot.plot(predictions, label='Predicted')
pyplot.legend()
pyplot.show()
```

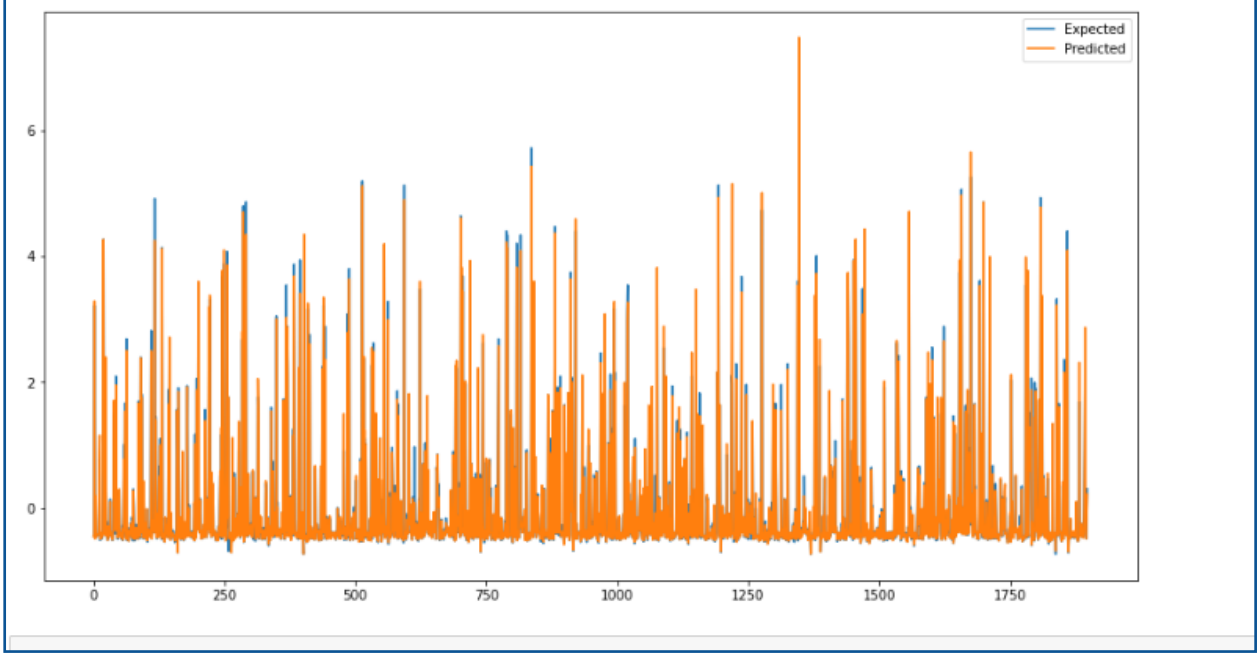


Figure 51: Plotting the actual VS predicted values in a line plot.

3.2.2.1 Testing Parameters

Similar to the Decision Tree model the parameter 'Dissolved Sodium' will need to be replaced and the file ran again for each parameter. The location in the code that the new parameter will need to be used is shown in Figure 52 below.

```
# split data into input and output columns
X = WQ.drop('Dissolved.Na..mg.l.', axis=1)
y = WQ.loc[:,['Dissolved.Na..mg.l.']]
```

Figure 52

The location where the variable name will need to be replaced is shown above in Figure 52. The parameters to replace Dissolved Sodium in the file are Dissolved Nitrate, Gran Alkalinity and Electrical Conductivity as shown in Table 2. Their variable names in the dataset are listed below and this is the name that should be used in the code:

Table 2: Variables that should be used to replace Dissolved Sodium in the code.

Dissolved.NO3..mg.l.NO3.
Gran.Alkalinity..uEq.l.
Electrical.conductivity..uS.cm.

3.2.3 Modelling Extreme Gradient Boosting

Extreme gradient boosting model was applied to the dataset using the xgboost library, this was installed using *!pip install xgboost.* The scikit-learn library was used to split the dataset into training and testing, scale the data and evaluate the final model. The packages used to apply the model are shown in Figure 53 below. The steps taken are shown in Figure 54 - Figure 58.

```
import numpy as np
import pandas as pd
!pip install xgboost
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot
```

Figure 53: Libraries required for implementing the model.

```
import os

os.getcwd()

os.chdir('C:/Users/35386/Desktop/Msc Data Analytics/Data/Used data')

os.getcwd()
```

```
'C:\\Users\\35386\\Desktop\\Msc Data Analytics\\Data\\Used data'
```

```
# Load the Pandas libraries with alias 'pd'

WQ = pd.read_csv("merged-final2.csv")

print(WQ)
```

Figure 54: Setting the working directory and loading the dataset.

```
# split data into input and output columns
#split the dataset into 80% training and 20% testing
X = WQ.drop('Dissolved.Na..mg.l.', axis=1)
y = WQ.loc[:,['Dissolved.Na..mg.l.']]
```

```
#scale the data
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)
```

```
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
#Split the data into training and testing sets

# split data into train and test sets
seed = 7
test_size = 0.20
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=seed)
```

```
###############################################################
###### Xgboost Regression in Python #######
from xgboost import XGBRegressor

# define model
model = XGBRegressor()
#fit the model
model.fit(X_train, y_train)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, ...)
```

```
# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
```

```
#evaluate the model
import sklearn.metrics as metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predictions))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
print('R Squared:',metrics.r2_score(y_test, predictions))
```

```
Mean Absolute Error: 0.35897462855077533
Mean Squared Error: 0.1551735659606748
Root Mean Squared Error: 0.3939207610175869
R Squared: 0.8453469720816984
```

Figure 55: Preparing the dataset, applying, and evaluating the model.

```
WQ_list = list(WQ.columns)
# Get numerical feature importances
importances = list(model.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(WQ_list, importances)]
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# Print out the feature and importances
[print('Variable: {:10} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
Variable: Dissolved.Mg..mg.l. Importance: 0.8999999761581421
Variable: Dissolved.Na..mg.l. Importance: 0.019999999552965164
Variable: Gran.Alkalinity..uEq.l. Importance: 0.019999999552965164
Variable: Dissolved.Ca..mg.l. Importance: 0.009999999776482582
Variable: Dissolved.Cl..mg.l. Importance: 0.009999999776482582
Variable: Dissolved.Ba..ug.l. Importance: 0.009999999776482582
Variable: Dissolved.Ni..ug.l. Importance: 0.009999999776482582
Variable: sunshine   Importance: 0.0
Variable: mean_temp  Importance: 0.0
Variable: precipitation Importance: 0.0
Variable: month      Importance: 0.0
Variable: day        Importance: 0.0
Variable: year       Importance: 0.0
Variable: Temperature..C. Importance: 0.0
Variable: Dissolved.K..mg.l. Importance: 0.0
Variable: Dissolved.SO4..mg.l.SO4. Importance: 0.0
Variable: Dissolved.NO3..mg.l.NO3. Importance: 0.0
Variable: TDP..ug.l.P. Importance: 0.0
Variable: pH         Importance: 0.0
Variable: Electrical.conductivity..uS.cm. Importance: 0.0
Variable: Suspended.sediments..mg.l. Importance: 0.0
Variable: Dissolved.B..ug.l. Importance: 0.0
Variable: Dissolved.Cr..ug.l. Importance: 0.0
Variable: Dissolved.Fe..ug.l. Importance: 0.0
Variable: Dissolved.Li..ug.l. Importance: 0.0
Variable: Dissolved.Mn..ug.l. Importance: 0.0
Variable: Dissolved.Sr..ug.l. Importance: 0.0
```

Figure 56: Calculating the feature importance.

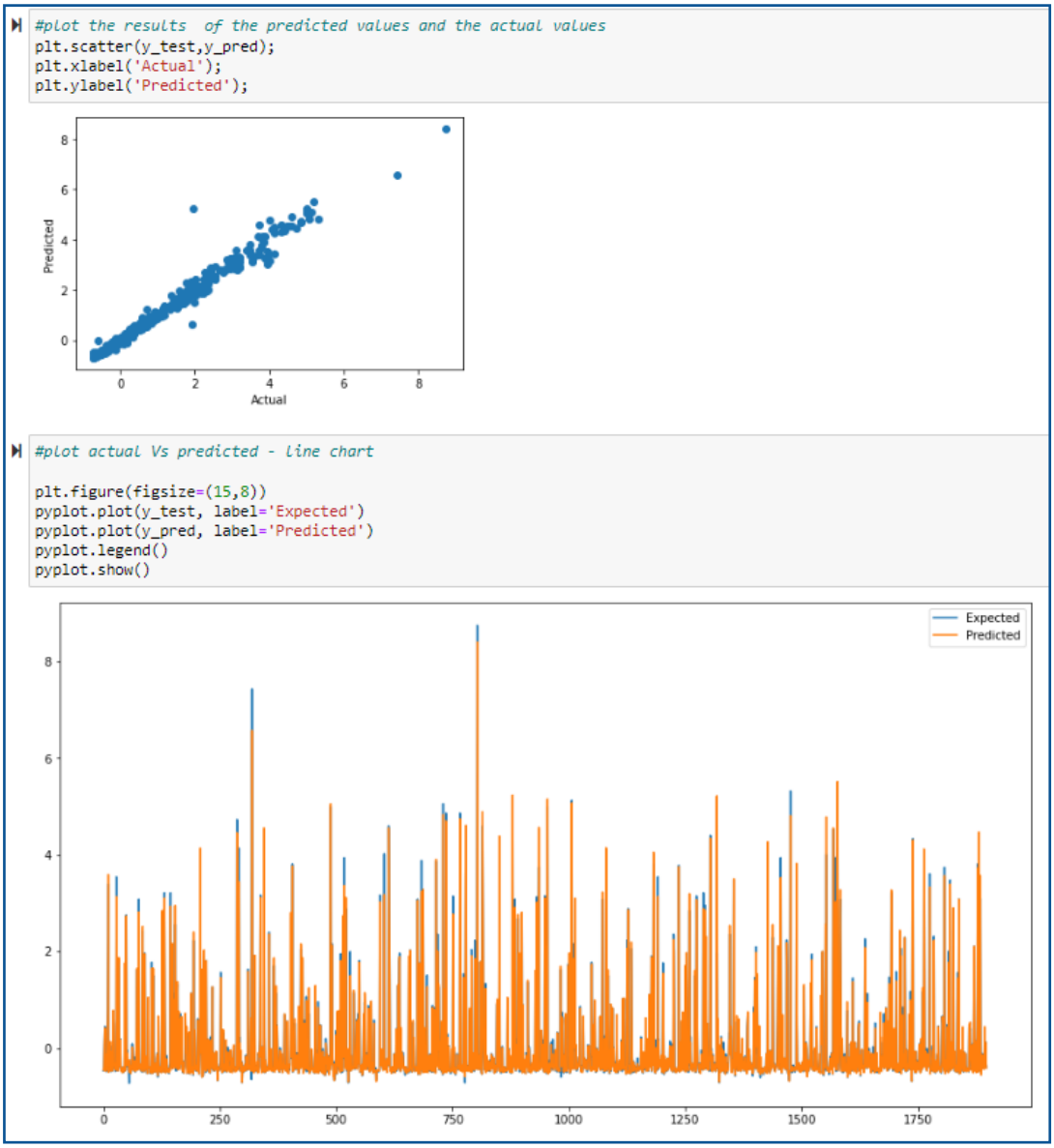


Figure 57: Plotting the results of the predicted Vs actual values.

3.2.3.1 Testing Parameters

Similar to previous models the parameter ‘Dissolved Sodium’ will need to be replaced and the file ran again for each parameter.

```
# split data into input and output columns
X = WQ.drop('Dissolved.Na..mg.l.', axis=1)
y = WQ.loc[:,['Dissolved.Na..mg.l.']]
```

Figure 58

The location where the variable name will need to be replaced is shown above in Figure 58.

The parameters to replace Dissolved Sodium in the file are Dissolved Nitrate, Gran Alkalinity and Electrical Conductivity and are shown in Table 3. Their variable names in the dataset are listed below and this is the name that should be used in the code:

Table 3: Variables that should be used to replace Dissolved Sodium in the code.

Dissolved.NO3..mg.l.NO3.
Gran.Alkalinity..uEq.l.
Electrical.conductivity..uS.cm.

3.2.4 Modelling Support Vector Machine

Support Vector Machine model was applied to the data using the scikit-learn library. The data was split into training and testing, scaled and evaluated also using the scikit-learn library. The packages used to apply the model are shown in Figure 59 below. The steps taken are shown in Figure 60 - Figure 64.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn.metrics as metrics
from matplotlib import pyplot
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
```

Figure 59: Libraries required for implementing the model.

```
import os

os.getcwd()

os.chdir('C:/Users/35386/Desktop/Msc Data Analytics/Data/Used data')

os.getcwd()
```

```
'C:\\Users\\35386\\Desktop\\Msc Data Analytics\\Data\\Used data'
```

```
# Load the Pandas libraries with alias 'pd'

WQ = pd.read_csv("merged-final2.csv")

print(WQ)
```

Figure 60: Setting the working directory and loading the dataset.

```
# split data into input and output columns
X = WQ.drop('Dissolved.Na..mg.l.', axis=1)
y = WQ.loc[:,['Dissolved.Na..mg.l.']]

#scale the data

sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)

# Splitting to training and testing data

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.20, random_state = 4)

#Fitting SVR on the dataset
from sklearn.svm import SVR
svr = SVR(kernel = 'rbf')
svr.fit(X_train, y_train)
```

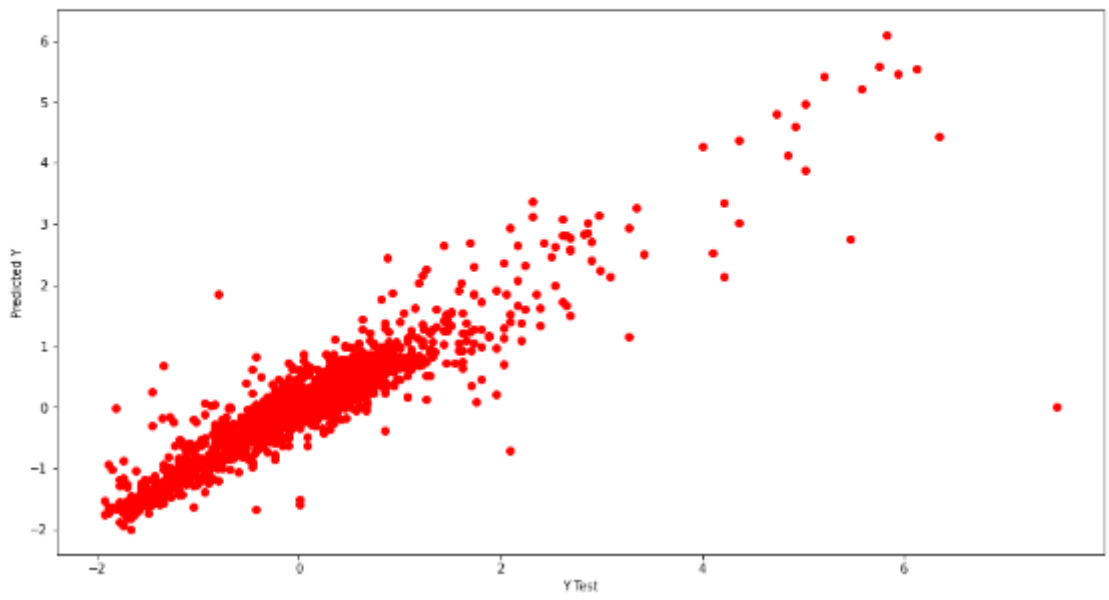
Figure 61: Splitting, scaling and fitting the Support Vector Machine model.

```
svr_pred = svr.predict(X_test)
svr_pred= svr_pred.reshape(-1,1)

print('MAE:', metrics.mean_absolute_error(y_test, svr_pred))
print('MSE:', metrics.mean_squared_error(y_test, svr_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, svr_pred)))
print('R Squared:',metrics.r2_score(y_test, svr_pred))
```

```
MAE: 0.0779404198462084
MSE: 0.03845162335760743
RMSE: 0.19609085485459904
R Squared: 0.9620916312665934
```

Figure 62: Predicting and Evaluating the model.

```
#plot actual Vs predicted - scatter plot

plt.figure(figsize=(15,8))
plt.scatter(y_test,svr_pred, c='red')
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
plt.show()
```




```
#plot actual Vs predicted - line chart

plt.figure(figsize=(15,8))
pyplot.plot(y_test, label='Expected')
pyplot.plot(svr_pred, label='Predicted')
pyplot.legend()
pyplot.show()
```

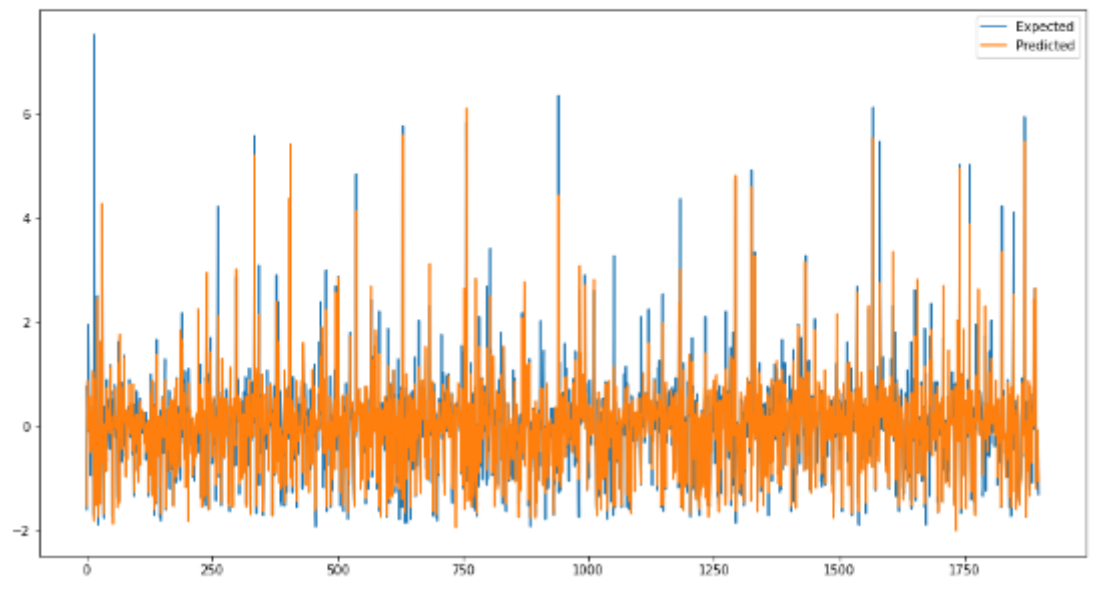


Figure 63: Plotting the actual Vs predicted values.

3.2.4.1 Testing Parameters

Similar to previous models the parameter ‘Dissolved Sodium’ will need to be replaced and the file ran again for each parameter.

```
# split data into input and output columns
X = WQ.drop('Dissolved.Na..mg.l.', axis=1)
y = WQ.loc[:,['Dissolved.Na..mg.l.']]
```

Figure 64

The location where the variable name will need to be replaced is shown above in Figure 64. The parameters to replace Dissolved Sodium in the file are Dissolved Nitrate, Gran Alkalinity and Electrical Conductivity and are shown in Table 4. Their variable names in the dataset are listed below and this is the name that should be used in the code:

Table 4: Variables that should be used to replace Dissolved Sodium in the code.

Dissolved.NO3..mg.l.NO3.
Gran.Alkalinity..uEq.l.
Electrical.conductivity..uS.cm.

3.2.5 Modelling Multiple Linear Regression

The Multiple linear Regression model was applied to the dataset using the Scikit-learn library, the data was also split into training and testing, scaled and evaluated using the same library. The packages used to apply the model are shown in Figure 65. A diagnostic was applied to the model using the statsmodels.stats library to view the Durbin-Watson statistic. The steps taken to apply the model and to test the assumptions are shown in Figure 66 to Figure 76 below.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import pyplot
from sklearn.preprocessing import StandardScaler
import sklearn.metrics as metrics
from statsmodels.stats import diagnostic
```

Figure 65: Libraries required for implementing the model.

```
import os

os.getcwd()

os.chdir('C:/Users/35386/Desktop/Msc Data Analytics/Data/Used data')

os.getcwd()
```

```
'C:\\Users\\35386\\Desktop\\Msc Data Analytics\\Data\\Used data'
```

```
# Load the Pandas libraries with alias 'pd'

WQ = pd.read_csv("merged-final2.csv")

print(WQ)
```

Figure 66: Setting the working directory and loading the dataset.

```
# split data into input and output columns
#split the dataset into 80% training and 20% testing
X = WQ.drop('Dissolved.Na..mg.l.', axis=1)
y = WQ.loc[:,['Dissolved.Na..mg.l.']]

#scale the data
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)

# Using Skicit-learn to split data into training and testing sets
seed = 7
test_size = 0.20
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=seed)

#fit the model to the data

linreg=LinearRegression()
linreg.fit(X_train,y_train)
```

```
LinearRegression()
```

```
#predict the results

y_pred=linreg.predict(X_test)
y_pred
```

Figure 67: Preparing the data to fit and predict with the Multiple Linear Regression Model.

```
#evaluate the model
import sklearn.metrics as metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R Squared:',metrics.r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 0.0949853921663505
Mean Squared Error: 0.027942205300303852
Root Mean Squared Error: 0.16715922140373785
R Squared: 0.972151528324728
```

Figure 68: Evaluating the Multiple Linear Regression Model.

It is important to note that Multiple Linear Regression has several assumptions that must be met, these assumptions are shown below, and the code used is shown in Figure 69 to 76.

Assumption 1: There is a linear relationship between the dependent and independent variables.

```
#Assumption 1: There is a linear relationship between the dependent and independent variables.
# visualize the relationship between the features and the response using scatterplots
p = sns.pairplot(WQ, x_vars=['sunshine', 'mean_temp', 'precipitation', 'month', 'day', 'year', 'Temperature..C.', 'Dissolved.
```

Figure 69: Visualisation of the response variable and independent variable using scatterplots.

Assumption 2: The data should not show multicollinearity which is when the independent variable is correlated with another independent variable.

```
#find the residuals
residual = y_test - y_pred
```

```
X_train = pd.DataFrame(X_train)
```

```
#Assumption 2: The data should not show multicollinearity which is when the independent variable is correlated with another i
#VIF > 10 - high VIF indicates high multicollinearity

from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
pd.DataFrame({'vif':vif[0:]}, index = X_train.columns).T
```

	0	1	2	3	4	5	6	7	8	9	...	17	18	19	20	
vif	1.386647	2.600474	1.085228	1.187578	1.013872	2.359422	2.515918	7.748729	11.920543	6.435131	...	1.134776	9.940352	2.320277	1.12239	1.2502

Figure 70: VIF Score for each variable.

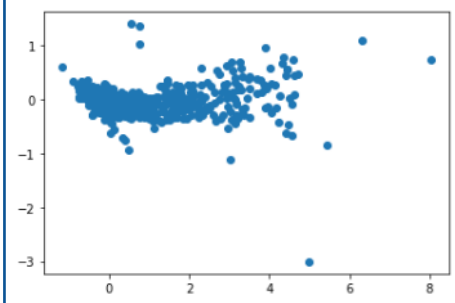
Table 5: VIF values above 10 in the dataset

Variable	VIF
Dissolved Sodium	41.204560338553364
Dissolved Chlorine	23.879478473810476
Dissolved Sulphate	11.140613350667332
Total Dissolved Potassium	53.853347933113184

Assumption 3: Homoscedasticity which means that the residuals have a constant variance.

```
# Assumption 3 - Homoscedasticity which means that the residuals have a constant variance.
#residuals vs predicted plot
#residuals seem to cluster which shows that Homoscedasticity may exist
plt.scatter(y_pred, residual)
```

```
<matplotlib.collections.PathCollection at 0x14ac4cc4d60>
```




```
#Assumption 3 Continued -autocorrelation - all points after the first point should be within the blue shaded area,
#if any points are outside, autocorrelation is present
#One point is outside which suggests autocorrelation here

import statsmodels.api as sm
import statsmodels.tsa.api as smt

acf = smt.graphics.plot_acf(residual, lags=40 , alpha=0.05)
```

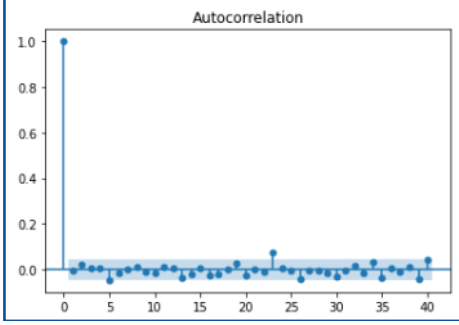


Figure 71: Homoscedasticity and autocorrelation of the residuals.

Assumption 4: Multivariate Normality which occurs when the distribution of the residuals are normal.

```
#Assumption 4: Multivariate Normality which occurs when the distribution of the residuals are normal.
%matplotlib inline
import seaborn as sns
sns.distplot(residual)
```

```
C:\Users\35386\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` i
and will be removed in a future version. Please adapt your code to use either `displot` (a figure-lev
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
<AxesSubplot:ylabel='Density'>
```

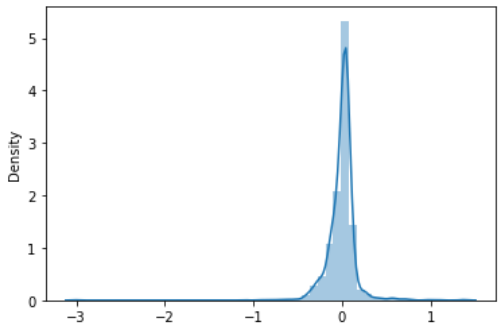


Figure 72: Multivariate Normality of the residuals.

Assumption 5: Observations should be independent of each other.

```
#Assumption 5 - that the observations should be independent of each other using the Durbin-Watson test
from statsmodels.stats import diagnostic

import statsmodels.api as sm
model = sm.OLS(y, X).fit()
predictions = linreg.predict(X_test)

print_model = model.summary()
print(print_model)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.572
Model:                            OLS   Adj. R-squared (uncentered):              0.572
Method:                 Least Squares   F-statistic:                              1154.
Date:                Sun, 14 Aug 2022   Prob (F-statistic):                        0.00
Time:                        10:54:16   Log-Likelihood:                         -9433.7
No. Observations:                9490   AIC:                                  1.889e+04
Df Residuals:                    9479   BIC:                                  1.897e+04
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.5841      0.012     49.177      0.000       0.561       0.607
x2             0.9107      0.018     50.428      0.000       0.875       0.946
x3            -0.0449      0.012     -3.635      0.000      -0.069      -0.021
x4            -0.3156      0.017    -18.643      0.000      -0.349      -0.282
x5             0.0415      0.013      3.181      0.001       0.016       0.067
x6            -0.0203      0.007     -2.936      0.003      -0.034      -0.007
x7            -0.1746      0.015    -11.489      0.000      -0.204      -0.145
x8            -0.0563      0.009     -5.979      0.000      -0.075      -0.038
x9            -0.1019      0.010     -9.753      0.000      -0.122      -0.081
x10            0.1407      0.010     13.763      0.000       0.121       0.161
x11           -0.2312      0.010    -23.668      0.000      -0.250      -0.212
==============================================================================
Omnibus:                     5190.688   Durbin-Watson:                   1.488
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           201360.311
Skew:                           1.980   Prob(JB):                         0.00
Kurtosis:                      25.216   Cond. No.                         7.46
==============================================================================
```

Figure 73: Durbin-Watson test for the Multiple Linear Regression model.

```
#Now apply MLR again dropping the unwanted columns
WQ = WQ.drop(['Dissolved.Na..mg.l.', 'Dissolved.Cl..mg.l.', 'TDP..ug.l.P.', 'Dissolved.SO4..mg.l.SO4.','sunshine', 'mean_temp
```

Figure 74: Dropping columns from the dataset.

```
# split data into input and output columns
#split the dataset into 80% training and 20% testing
X = WQ.drop('Dissolved.NO3..mg.l.NO3.', axis=1)
y = WQ.loc[:,['Dissolved.NO3..mg.l.NO3.']]
```

```
#scale the data
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)
```

```
# Using Skicit-learn to split data into training and testing sets

seed = 7
test_size = 0.20
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=seed)
```

```
#fit the model to the data

linreg=LinearRegression()
linreg.fit(X_train,y_train)
```

```
LinearRegression()
```

```
#predict the results

y_pred=linreg.predict(X_test)
y_pred
```

Figure 75: Splitting, Scaling and fitting the new model on the new dataset.

```
#evaluate the model
import sklearn.metrics as metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R Squared:',metrics.r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 0.4270127003916817
Mean Squared Error: 0.39787580594518185
Root Mean Squared Error: 0.6307739737379641
R Squared: 0.5505046937857332
```

Figure 70: Evaluating the new model.

3.2.5.1 Testing Parameters

Similar to the previous models the parameter ‘Dissolved Sodium’ will need to be replaced and the file ran again for each parameter.

```
# split data into input and output columns
X = WQ.drop('Dissolved.Na..mg.l.', axis=1)
y = WQ.loc[:,['Dissolved.Na..mg.l.']]
```

Figure 76

The location where the variable name will need to be replaced is shown in Figure 76 above.

The parameters to replace Dissolved Sodium in the file are Dissolved Nitrate, Gran Alkalinity and Electrical Conductivity and are shown in Table 6. Their variable names in the dataset are listed below and this is the name that should be used in the code:

Table 6: Variables that should be used to replace Dissolved Sodium in the code.

Dissolved.NO3..mg.l.NO3.
Gran.Alkalinity..uEq.l.
Electrical.conductivity..uS.cm.

References

Cran.r-project.org. 2022. Download R-4.2.1 for Windows. The R-project for statistical computing.. [online] Available at: <https://cran.r-project.org/bin/windows/base/> [Accessed 01 August 2022].

Rstudio.com. 2022. Download the RStudio IDE. [online] Available at: <https://www.rstudio.com/products/rstudio/download/> [Accessed 1 August 2022].

Anaconda. 2022. Anaconda | The World's Most Popular Data Science Platform. [online] Available at: <https://www.anaconda.com/> [Accessed 1 August 2022].