

Department of Mathematics

# **Modelling Drug Substance Production Processes**

By

Shivaani Katragadda  
(R00183214)

Supervisor: Dr. David Hawe

Thesis submitted in partial fulfilment of  
**Masters' in Data Science & Analytics**

In association with



Submitted to Cork Institute of Technology  
August 2020

## Declaration of Authorship

I, Shivaani Katragadda, declare that this thesis titled, 'Modelling Drug Substance Production Processes' and the work presented in it are my own. I confirm that:

- ❖ This work was done wholly or mainly while in candidature for the Masters' degree at Cork Institute of Technology.
- ❖ Where any part of this thesis has previously been submitted for a degree or any other qualification at Cork Institute of Technology or any other institution, this has been clearly stated.
- ❖ Where I have consulted the published work of others, this is always clearly attributed.
- ❖ Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work.
- ❖ I have acknowledged all main sources of help.
- ❖ Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.
- ❖ I understand that my project documentation may be stored in the library at CIT and may be referenced by others in the future.

Signed: Shivaani Katragadda

Date: 23<sup>rd</sup> August 2020

## ACKNOWLEDGEMENTS

I would like to thank my supervisor, **Dr. David Hawe** for his valuable inputs and contribution in this project. His guidance and motivation made it possible for me to complete the thesis successfully. I would also like to thank **Patrick T. O'Sullivan, Janssen Pharmaceutical Sciences UC** for giving me an opportunity to work on this project with them.

# List of Abbreviations

Abbreviation	Full Form
<b>IgG</b>	Immunoglobulin G
<b>ML</b>	Machine Learning
<b>MLR</b>	Multiple Linear Regression
<b>SVM</b>	Support Vector Machines
<b>VCD</b>	Viable Cell Density

# Abstract

Biopharmaceutical industries play a prominent role in global health care. There are several steps in the manufacturing of a drug. Pharmaceutical companies concentrate primarily on the production of medicines that help people infected by diseases. The production of medicine costs billions. Hence, biopharmaceutical industries need to predict the domestic production of medicines that can be accomplished using different technologies. Machine Learning is one such approach; it can potentially be an effective solution for predicting the output for the given input. Different machine learning techniques, such as regression, clustering, and classification, can predict the future output. Support Vector Machines (SVMs), Ensemble learning (EL), linear and logistic regression, decision trees, Naive Bayes, and Artificial and Deep Neural Networks (ANNs and DL) are some of the algorithms used to make the prediction. This project's primary goal is to use a machine learning technique for predicting IgG (immunoglobulin G) using regression and tuning the best model by hyper-parameters with Scikit learn GridSearchCV. The algorithms used in this project are Linear Regression, Decision Tree Regressor, and Ensemble Learning - Random Forest Regressor, Support Vector Machine between the four Random Forest Regressor, looks very promising best performance.

**Keywords:** Machine Learning, Biopharmaceutical, Linear Regression, Decision tree, Random Forest, Support Vector Machine, GridSearchCV

# Table of Contents

<b>1</b>	<b>Introduction.....</b>	<b>1—1</b>
1.1	Aim.....	1—1
1.2	Background Work .....	1—1
1.2.1	Biopharmaceutical Production Process.....	1—2
1.3	Document Overview.....	1—4
<b>2</b>	<b>Literature Review .....</b>	<b>2—6</b>
<b>3</b>	<b>Methodology .....</b>	<b>3—9</b>
3.1	Introduction .....	3—9
3.1.1	Data Description.....	3—9
3.1.2	Data Pre-Processing .....	3—12
3.1.3	Split the dataset into train and test set.....	3—16
3.1.4	Applying Machine Learning Technique .....	3—16
3.1.5	Hyperparameter Tuning .....	3—24
<b>4</b>	<b>Results .....</b>	<b>4—26</b>
4.1	Multiple Linear Regression .....	4—26
4.2	Decision Tree .....	4—28
4.3	Random Forest .....	4—30
4.4	Support Vector Machines.....	4—31
4.5	Hyperparameters Tuning.....	4—34
<b>5</b>	<b>Conclusion and Future Scope .....</b>	<b>5—36</b>

5.1 Discussion ..... 5—36

5.2 Conclusion..... 5—36

5.3 Future Scope ..... 5—37

**Bibliography ..... 5—38**

**APPENDIX ..... 5—42**

APPENDIX A ..... 5—42

APPENDIX B..... 5—46

APPENDIX C..... 5—47

APPENDIX D ..... 5—65

# List of Figures

Figure 1: The upstream and downstream bioprocess .....	1—4
Figure 2: The structure of Decision Tree.....	3—18
Figure 3: The structure of Random Forest .....	3—19
Figure 4: Possible Hyperplanes.....	3—20
Figure 5: Support Vectors .....	3—21
Figure 6: Training the SVM .....	3—22
Figure 7: Illustrative Example of Simple SVR.....	3—23
Figure 8: Illustrative Example of SVR with Slack Variables .....	3—23
Figure 9: GridSearchCV and RandonSearchCV.....	3—25
Figure 10: Dependent and independent features .....	4—26
Figure 11: Dividing the data in to training data and testing data.....	4—26
Figure 12: Coefficients of independent variables .....	4—27
Figure 13: Result of the MLR algorithm.....	4—27
Figure 14: Actual vs predicted values of MLR .....	4—28
Figure 15: Result of the Decision Tree algorithm .....	4—29
Figure 16: Actual vs predicted values of Decision Tree.....	4—29
Figure 17: Result of the Random Forest algorithm .....	4—30
Figure 18: Actual vs predicted values of Random Forest .....	4—31
Figure 19: Result of the SVM algorithm .....	4—32



Figure 20: Actual vs predicted values of SVM .....	4—32
Figure 21: Accuracy score of all the algorithms .....	4—33
Figure 22: Performance comparison of all the algorithms.....	4—33
Figure 23: parameter grid for hyperparameter tuning .....	4—34
Figure 24: Result of Random Forest algorithm after hyperparameter tuning .....	4—34
Figure 25: Actual vs predicted values of Random Forest after hyperparameter tuning.....	4—35
Figure 26: Correlation matrix of discrete dataset.....	5—65
Figure 27: Correlation matrix of continuous dataset .....	5—66
Figure 28: outliers .....	5—67
Figure 29: Relation between Biomass and VCD.....	5—68
Figure 30: Relation between VCD and IgG.....	5—68

# List of Tables

Table 1: Description of the data sheets.....3—11

Table 2: Data for MLR .....3—16

Table 3: Discrete data columns .....5—43

Table 4: Continuous data columns.....5—46

# 1 Introduction

This project has been completed in collaboration with Janssen Pharmaceutical Sciences UC, a part of the Johnson & Johnson family of companies, and dataset is provided by them. Since 1981 Janssen Pharmaceutical Sciences UC has been operating in Cork, Little Island. The company manufactures pharmaceutical ingredients to all other Janssen branches and third party companies[1]. These ingredients are processed further into tablets and injectable dosage forms.

## 1.1 Aim

To create a predictive model for a Y variable IgG(immunoglobulin G) concerning the independent variables using different machine learning algorithms.

IgG: Immunoglobulin G (IgG) is an antibody type. It is the most common antibody detected in blood circulation, containing around 75 percent of serum antibodies in humans[2]. IgG molecules are formed and produced by plasma B cells.

## 1.2 Background Work

This project is related to the pharma industry and involves the production processes of the drug substances. Pharmaceutical manufacturing plays an important role in producing cost-effective medicines. This can be achieved only by following a systematic approach to the biopharmaceutical manufacturing process, which supports the supply of medicines. The better manufacturing of medicines is necessary to provide healthcare in Europe and all over the world. The EU Institution for Health[37] has acknowledged in

several documentation and statements that the pharmaceutical industry contributes to citizens by enhancing the quality of medicines, economic growth, and jobs. The Pharmaceutical industry is the high-tech sector in Europe with the most significant value-added for every employed person[38]. Biopharmaceuticals always require high standards of quality, high initial expenditure for authorization, market introduction, and considerable investments in production[3] [4]. A biopharmaceutical is also known as a biological medicinal drug[5]. It is a drug product derived from biological sources, such as proteins, fats, and nucleic acids, living cells, including vaccinations, therapeutic proteins, and live-cell therapy medicines [6]. Biopharmaceutical firms use living organisms such as mammals, bacteria, and yeast to manufacture medicines[7].

### 1.2.1 Biopharmaceutical Production Process

The steps involved in the biopharmaceutical production process is as follows:

- 1. Fermentation:** It includes the cultivation of bacteria, fungi, or biological cells to produce protein drugs.
- 2. Chromatography:** In this step antibodies products are polished and filtered.
- 3. Purification:** In this step, all the antibodies are purified completely.
- 4. Final product:** The final antibodies product is tested in this step[8] [9].

One can express fermentation in two streams[10]. They are bioprocessing upstream and bioprocessing downstream.

The upstream is the process from cell isolation(the process of separating the cells from tissues), cell cultivation(the process of growing the cells under certain controlled conditions) to cell banking(it is a place where cells are

preserved with a specific set of chromosomes for future use in a drug), and culture expansion of the cells until the harvest (terminating the cell culture and collecting the cells). Generally, the cell/seed will be grown in a flask for 4 to 5 days, called the preculture phase. Then the seed will be moved to a bioreactor and processed for 60 days, called the profusion phase. It will then be shifted to a tank to harvest, and then the product will be captured. The upstream bioprocess refers to the first phase in which seed produced. The upstream method includes all steps related to inoculum(the substance which contains bacteria, that can be used for cell culture) development, media production, genetic engineering process improvement of culture media, and optimization of growth kinetics to enhance product quality[10]. The principal objective of the upstream cycle is to turn substrates[11] into the desired metabolic products[12]. After product growth, the next step is to purify and achieve product quality. When the product is achieved, that will be harvested and sent to the downstream process.

The downstream process involves all the required steps to purify the cell collected from the upstream is processed to meet the specifications and eventually get the final purified product. The downstream process consists of three stages[13] [14] [15]. They are as follows

- 1. Initial recovery:** In this phase the separation of the cells of the bioreactor takes place. Generally, in this phase the small molecules, water, and growth supplements are removed[16].
- 2. Purification:** In this phase, contaminants (polluting or poisonous substances), host cell proteins (impurities produced by the cells to produce proteins), harmful viruses are removed[16].

**3. Polishing:** During this process, unwanted biomolecules created during insolation and purification, removal of virus or pyrogens(the substance produced by bacteria) and the pollutants are removed in this phase[17]. In downstream, each step of purification can remove one or more impurities [14].

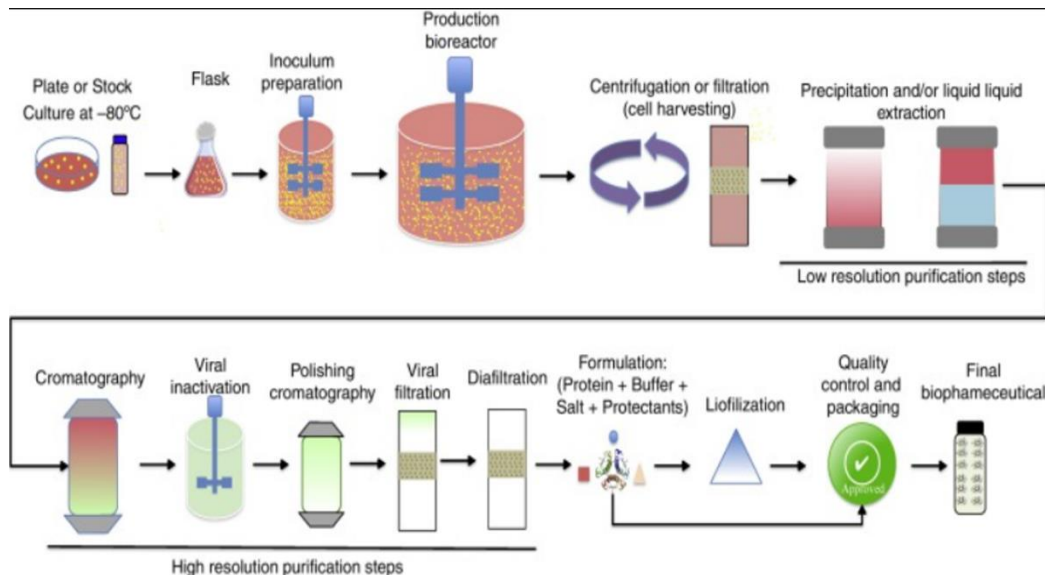


Figure 1: The upstream and the downstream bioprocess[15]

Biopharmaceutical manufacturing represents the upstream and downstream bioprocess, as shown in Figure1. This is the process of biopharmaceutical manufacturing.

### 1.3 Document Overview

Structure of the document is as follows:

**Literature Review (Chapter-2):** This section describes the background information about Machine Learning and presents the existing solutions for the project;

**Methodology (Chapter-3):** This section presents the description of the dataset, how the exploratory data analysis is performed on the dataset, training the models, and tuning the best model;

**Results (Chapter-4):** This section consists of the evaluated results of the models implemented;

**Conclusion and Future Scope (Chapter-5):** This chapter comprises the conclusion and future work and the scope for the improvement of the project.

## 2 Literature Review

This chapter's literature review provides information about the implementation of machine learning in biopharmaceutical and different areas such as housing and agriculture.

Machine learning(ML), defined as a "field of study that gives computers the ability to learn without being explicitly programmed" [18]. ML deals with problems when there is no defined relationship between input and output variables, even where they are difficult to obtain. The term "learning" refers to the automated acquisition of structural descriptions. In contrast to traditional statistical methods, machine learning makes no assumptions about the correct data model structure which describes the data. This method was useful when modelling nonlinear dynamic behaviour. If the application domain information is poorly defined, then ML approaches are helpful[19].

This paper "Drug design by machine learning: support vector machines for pharmaceutical data analysis" [20] shows the SVM classification method that predicts the inhibition of dihydrofolate reductase(it is an important enzyme which produces the factors that are necessary for DNA synthesis) by pyrimidines(it is a chemical compound which will be found in nucleic acids of DNA), using a UCI machine learning repository(contains a collection of datasets which are used by machine learning community) dataset. The authors state that SVM performance is better than artificial neural network algorithms and a C5.0 decision tree as these algorithms requires a lot of time to train the model. It is shown that how efficient the SVM algorithm is predicts the dihydrofolate reductase.



The case study "Using Regression Analysis To Estimate Costs" [21] states that the U.S. Government Accountability Office regional and headquarters staff mostly uses regression analysis to measure the lifecycle cost alternative methods for conducting activities important for health care. The regression analysis is part of the research used to analyze the cost of health care services. It is clearly shown that linear regression is used for the analysis in the pharmacy department. In this project, multiple linear regression algorithm to predict the production of IgG.

The publication, "Applications of Machine Learning Techniques in Agricultural Crop Production" [22], showed that it is an important field of study, and expected to develop in the future. Integrating technologies with agriculture help in crop forecasting, this method also helps in providing information about crops and how to increase the yield rate. The algorithms used in this study are artificial neural networks, decision trees, and regression analysis. In this paper, it is explained clearly how the decision tree algorithm is used to predict the production of the soybean.

The paper named, "Machine Learning: Applications in Indian Agriculture" [23] reviews various applications of machine learning in the farming sector and provides insight into the troubles faced by many Indian farmers and how these can be solved using machine learning techniques. The algorithms used are decision trees, artificial neural networks, regression analysis. In this paper, it is shown that how different machine learning algorithms are used in prediction, for crop selection and yield prediction different classification and neural network algorithms are used, for weather forecasting SVM is used, for smart irrigation system any machine learning algorithms can be used, for

crop disease prediction SVM, regression trees, random forest algorithm were used. Here, the author states how ML algorithms can be used in agriculture for different categories such as weather, diseases, crop yield, and selection. , It can be understood that ML algorithms can be used in any sector or in sub-sectors to predict the results.

In this paper[24], SVM, least-square support vector machines (LSSVM), and partial least square (PLS) methods used to forecast house prices. Previous history and experiments show that although there is significant non-linearity in the data collection, the experiment's outcome shows that SVM and LSSVM methods are superior to PLS in solving non-linearity problems. The result of SVM is better compared to LSSVM. And LSSVM is portrayed on SVM basis. It is clearly explained that how the SVM algorithm is implemented and the prediction is very superior when compare to other algorithms.

From the above publications, we can see that ML techniques are used in various fields to predict the output using different algorithms. In this project, the IgG can be predicted by using different ML algorithms. The algorithms used to predict IgG are multiple linear regression, decision trees, random forest, and support vector machines. The papers mentioned above showed how these algorithms can predict the output in biopharmaceutical, agriculture, and housing sectors. All the papers reviewed here are very useful because each paper tells how the algorithm is implemented, predicting the output. These all are useful and related to the algorithm implemented in this project.

## 3 Methodology

This chapter describes the implementation of the project. As mentioned earlier, the main aim of this project is to predict the IgG by using machine learning algorithms.

Python programming language and the jupyter notebook software were primarily used for this project.

### **Why has Python been Chosen?**

Python is an interpreted programming language, famous for its code compatibility and readability. In terms of coding, it is user-friendly and supports several programming paradigms, such as structured, functional, and object-oriented programming. The syntax is straightforward and easy to learn. It is an open-source programming language, and all the packages are available for free. APPENDIX B provides the software packages used in this project.

### **3.1 Introduction**

The process of building the models involves the following steps:

1. Data Description
2. Data Pre-processing
3. Split the dataset into Train and Test set
4. Applying Machine Learning Technique
5. Hyperparameter Tuning

#### **3.1.1 Data Description**

As mentioned earlier, the data was provided by Janssen Pharmaceutical Sciences UC. The data is provided in .xlsx file with four sheets of data. Among

those sheets, three sheets contain continuous data for three batches, and one sheet contains discrete data.

### **Discrete Data**

Janssen uses discrete data to represent In Process Control (IPC: “Checks performed during production to monitor and, if appropriate, to adjust the process to ensure that the intermediate or API (Active Pharmaceutical ingredients, drug substance) conforms to its specifications and/or other defined quality criteria.”[25]) data, representing the performance of the previous day's bioreactor. The results from discrete data represent the performance of IgG as IgG relates to the yield from the bioreactor. Those that are called as flowrate are the only IPCs that do not relate to performance but there are variables that are measured daily but reflect the actual operation and considered to be very critical to perform.

### **Continuous Data**

Data is read continuously through hard sensors; and collected across the 60 days in every 10 minutes. It should be noted that several variables may exhibit collinearity. For example, the active probe(gives the more accurate measures), probe1, and probe2 (probe 1, probe 2 are used as backup when the active probe is broken any one probe among probe1 and probe2 will be used in place of active probe) corresponds to the same data and depends up on the probe which is used as the backup.

Now, the detailed description of each sheet is as follows:

- The First sheet contains discrete data with 2122 observations and 26 features. The details of the features can are in APPENDIX A. The names of the columns are changed/renamed for better understanding as the name of the columns in the given datasheet is very lengthy.

- The details about the sheets two, three and four are explained in the below table. The details of the features are in APPENDIX A.

Sheet Number	Number of observations	Number of features	Information
1	2122	26	Discrete Data
2	81674	90	Continuous data of the first batch
3	18500	60	Continuous data of the second batch. In this sheet, all the columns are the same as of datasheet two, but here the columns Date, UnitID, and 0 to 0.28 were not present, and there is one new column called "Time" that is present in this datasheet.
4	79362	90	continuous data of the third batch The features in sheet two and sheet four are the same.

Table 1 : Description of the data sheets

### 3.1.2 Data Pre-Processing

Initially, each sheet is converted to .csv file to read the files faster, as each sheet of .xlsx file takes a lot of time to load the data.

Exploratory data analysis is performed on a discrete datasheet first. The discrete datasheet has 2122 observations and 26 features.

- Initially, all the missing values concerning the "Date" column dropped, because the "Date" column is the only column that is common between the discrete and continuous datasheets. After preprocessing, both the discrete and continuous datasheets have to merge to make one datasheet for building models. The "Date" column can be imputed with the most repeated value, i.e., by mode(because the date column is in the format "date\_month\_year time\_interval" which cannot be imputed with mean or median), but it does not impact much if the missing value is imputed. So, the missing values in the "Date" column dropped first.
- After that, all the missing values present in other columns are replaced with the mean and checked for duplicate observations. In this sheet, there are no duplicate observations, and only missing values are present. Log transformation ( $\text{np.log1p}$ ) was applied to "Biomass" and "VCD"(Viable Cell Density) to reduce the noise because as the changes occur in VCD will impact Biomass. So, log transformations introduced to compensate for such noises.
- The " cell\_culture " column removed as it is an evolution variable that contains the number of each day starting from 0 to 60 days for each batch, which is not useful in the prediction because the value(0-60) is same and constant for every and it will never change.
- The outliers were checked for each column and replaced the outliers with the median. Initially boxplot is plotted to check whether any

outliers are present and the first quantile(0.25) and the third quantile(0.75) is found to find the inter quantile range(IQR=Q1-Q3). The data greater than  $Q3+1.5*IQR$  and less than  $Q1-1.5*IQR$  are made into nan and then filled the nan values with the median

- To know the relationship between the variables, one needs to develop a correlation matrix. The data must be quantitative (numeric data) to obtain a correlation matrix(pearson correlation coefficient). Among all the columns, the columns 'BatchID' and 'Date' are qualitative data. These two columns are required for the further analysis of the data. It is not possible to find the correlation matrix with these two columns, so these two columns are stored as a list and removed. The 'UnitID' column has removed because it has the same value across the column. For the correlation matrix, all the float values are considered.
- After this, all the constant value (Material\_out, EDTA\_Harvest, ViabilityATF1, ViabilityATF2, ViableCellDensityATF1, ViableCellDensityATF2) columns dropped by using Variance Threshold(useful to remove constant columns) function imported from sklearn.feature\_selection and Quasi-constant features(almost constant features),in this sheet there are no quasi-constant features present. Within this datasheet, correlated features are not deleted as the target variable "IgG" would be removed because it increases day by day.
- The shape of the discrete datasheet is 566 observations and 16 columns. Finally, the two columns Date and BatchID are attached to the data frame by performing some analysis. As mentioned earlier, both are stored as a list,the Date column has 1 row and 566 columns. It is stored as a data frame by using the pd.dataframe, and transpose(it

is applied because when the list is made to a dataframe all rows are stored as column, in order to convert the columns in to rows transpose is used) is applied to the data frame now; then the shape of the data frame is 566 rows and 1columns. The date column is in the format day-month-year hour: minute. Here the time interval is of no use because, for all the dates, the time interval is 00:00, so the time interval is removed and changed the format of day\_month\_year to year\_day\_month. The same approach is followed for the batch column as well, initially converted to the data frame and applied transpose to the data frame and then appended into the main data frame. Finally, the shape of the discrete data sheet is 566 observations and 18 columns.

There are three continuous datasheets, those three sheets combined by using pandas Concat (pd.concat) function . All the sheets are concatenated because all the sheets contains continuous data of three different batches in order to make the analysis easy all the three sheets are combined. Then the same exploratory data analysis approach is followed for continuous datasheet as well.

Some extra steps followed here to clean the data are as follows:

- Initially, a threshold value(0.6) is set up to remove the columns with more percentage of missing values than the threshold because the column which is having more number of missing values will not show good performance if we imputed with mean, median, or mode according to the type of data and these columns are also not useful for prediction. The "Time" column has a large percentage of missing value, so that way, the column is removed.



## Methodology

- There are 35 columns which are having the same constant value i.e., "Tag not found" which are considered unnecessary and not useful in prediction, so all those columns removed.
- There are some values like 'Unit Down'(when the probe is not work unit down will occur), 'Equip Fail'(when any equipment is not working this problem occurs), 'Error'(when any action fails error takes place), these are some of the problems that occur when there is a problem with the probe. All these values are replaced with 0 because the data is quantitative. If these values are removed the entire row will be deleted and there is chance to loose the valuable data in order to avoid this situation it is good to replace the data with zero, moreover zero does not add any worth to data.
- The correlated features removed in this datasheet as they are not useful in prediction.

Apart from these steps, the same exploratory data analysis approach followed for a continuous datasheet.

Every date in the each row of datasheet is considered, and found the mean of each column associated with that date, and replaced each column with the corresponding mean value to have the same frequency when discrete and continuous datasheets are merged. Now both the discrete and continuous datasheets were merged by a date column.

The next step is to find the correlation of every feature concerning 'VCD' and 'IgG' columns and the features with a correlation greater than 0.5 considered and observed the relation between them.

The features concerning 'IgG' were considered as X variables and the target variable i.e. Y variable is 'IgG'.

### 3.1.3 Split the dataset into train and test set

In this step, the records are divided into two groups: training data and test data, the division ratio is 70% or 30%. The training set is the only training set that can train and adjust the model to basically adjust the parameters, while the test data is only used to evaluate the overall performance of the model. Training data can be used for modeling, while test data is sensory data for which predictions must be found. The train-test-split function is imported from sklearn package, by using that function the entire data set is divided into 70% for data training and 30% for data testing

### 3.1.4 Applying Machine Learning Technique

Four machine learning algorithms used for predicting IgG. They are:

#### 1. Multiple Linear Regression

Multiple Linear Regression (MLR) is a statistical method used to predict the outcome of a single dependent variable based on two or more independent variables. The variables we choose to predict are called dependent variables or response variables. The variables used to predict the value of the dependent variable are independent or explanatory variables. The data will be in the form of as shown in Table 2

y	X <sub>1</sub>	X <sub>2</sub>	--	X <sub>k</sub>
y <sub>1</sub>	X <sub>11</sub>	X <sub>12</sub>	--	X <sub>1k</sub>
y <sub>2</sub>	X <sub>21</sub>	X <sub>22</sub>	--	X <sub>2k</sub>
--	--	--	--	--
y <sub>i</sub>	X <sub>i1</sub>	X <sub>i2</sub>	--	X <sub>ik</sub>

Table 2 : Data for MLR

The equation of multiple linear regression is given in equation (1)

$$y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \dots + \beta_k X_{ik} + \epsilon \quad \text{-----}(1) [26]$$

“where, for  $i=n$  observations”

“ $y_i$ =dependent variable”

“ $x_i$ =independent variables”

“ $\beta_0$ =y-intercept”

“ $\beta_k$ =slope coefficients”

“ $\epsilon$ = residuals” [26]

### **2. Decision Tree**

The Decision tree is a supervised machine learning algorithm used for both regression and classification. It is a tree-structured classification method composed of three node types: root node, interior node, and leaf node, these nodes connected by branches. The Root Node is the initial node representing the entire sample and further divided into additional nodes. The Interior Nodes represent the characteristics of the dataset, and the branches represent the rules of decision. The leaf node represents the outcome[27].

This algorithm is very useful in solving decision-related problems. The algorithm starts with a single data point by answering true/false conditions until the leaf node reached. The final prediction for the dependent variable is the mean value for the particular leaf node. The tree will predict the correct value for the data point through multiple iterations[28]. The Decision tree uses a greedy(It is an algorithm that builds a solution step by step and then chooses the next step that provides immediate input) method; for this reason, an attribute/data point chooses at the starting step cannot be used anymore. Also, it overfits the data that gives the poor results, that can be resolved by using Random forest[28].

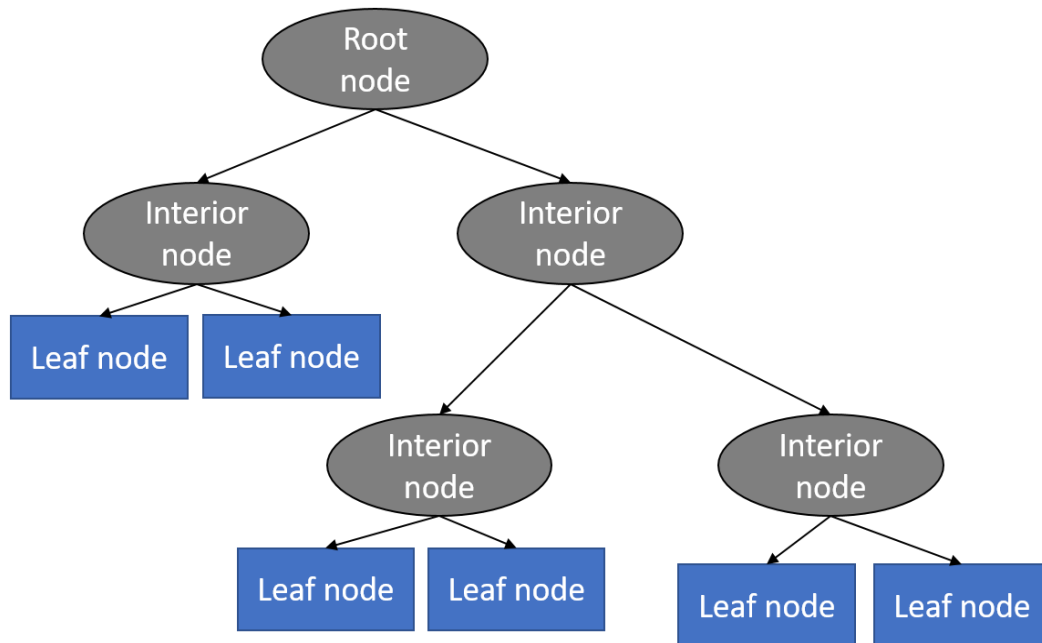


Figure 2: The structure of Decision Tree[27]

### 3. Random Forest

Random Forest is a tree based algorithm that uses multiple decision trees for making decisions. In simple words, the random forest defined as a forest of randomly generated decision trees. The disadvantage of the decision tree is that it creates overfitting. The implementation of the random forest in place of the decision tree will reduce the over fitting problem. The random forest algorithm is stable and efficient when compared to other regression algorithms. The random forest algorithm merges the output of several decision trees to produce the final output[29].

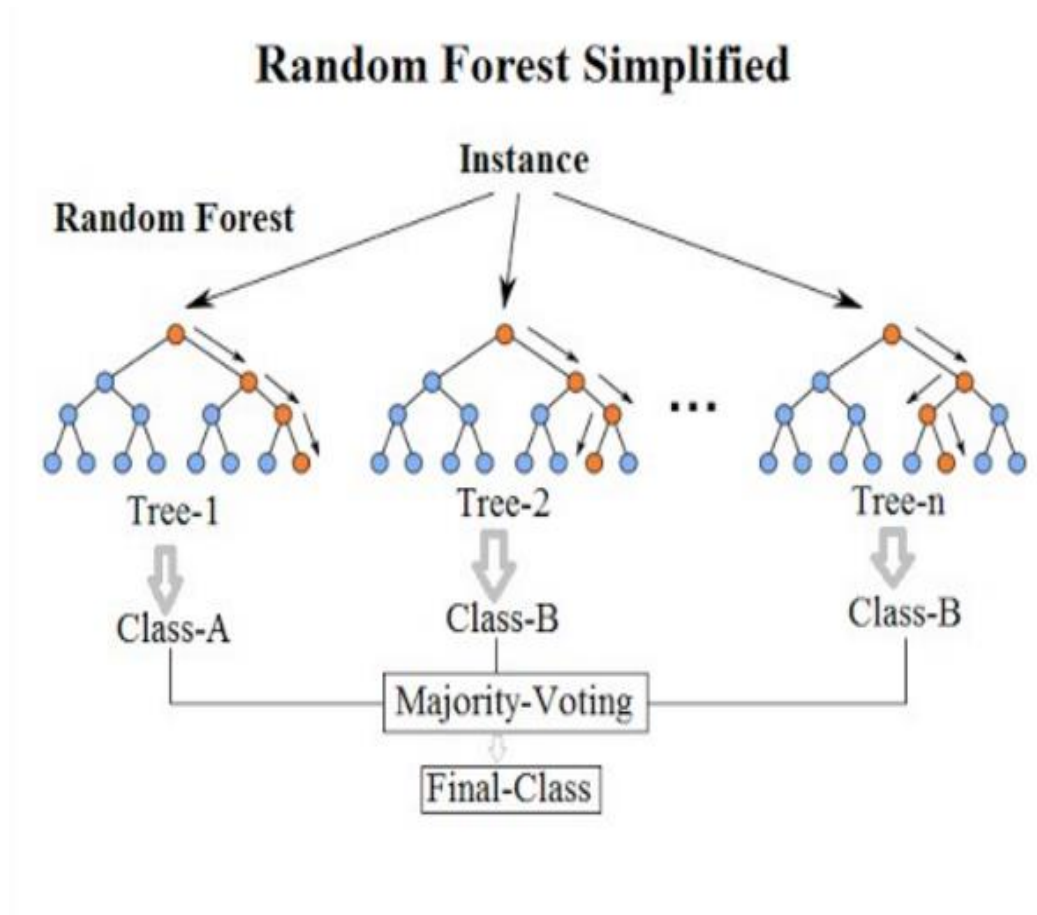


Figure 3: The structure of Random Forest[29]

#### 4. Support vector machine

Support vector machine is a machine learning algorithm used for regression as well as for classification. “The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N- the number of features) that distinctly classifies the data points”[30].

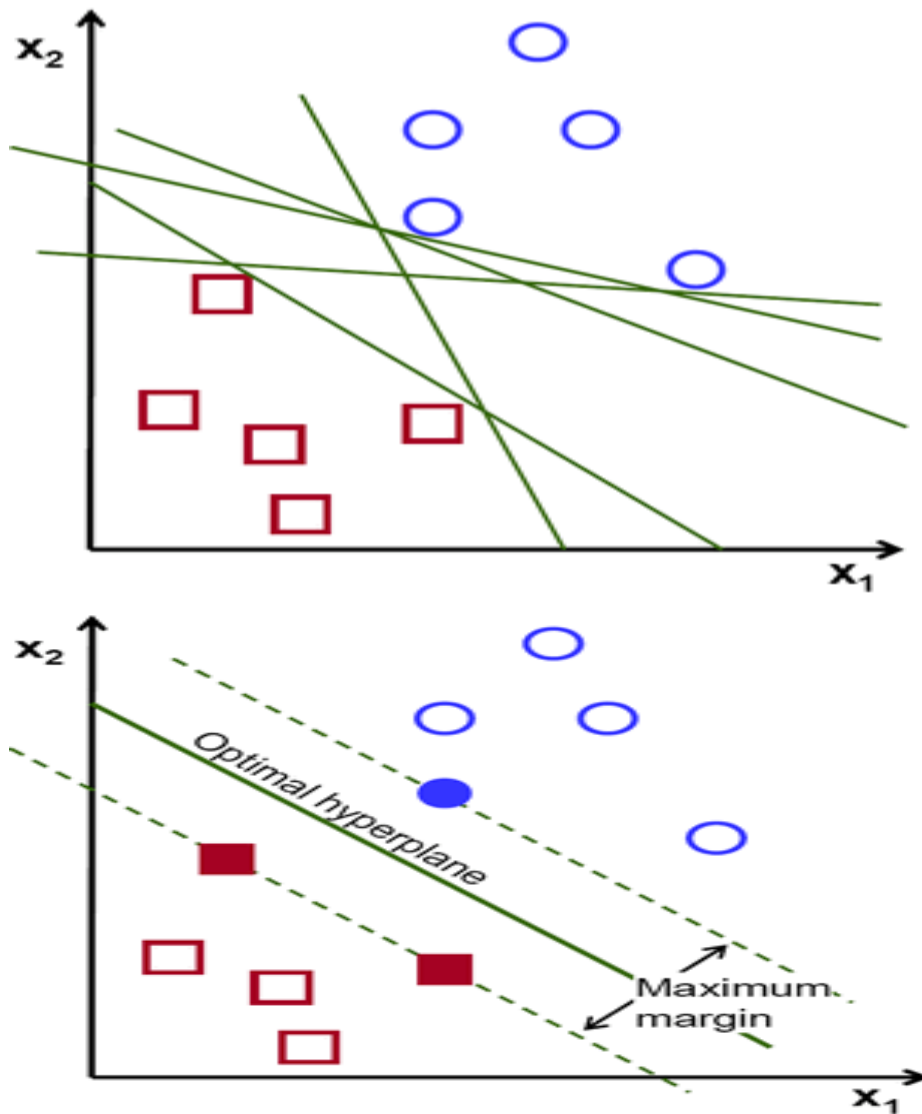


Figure 4: Possible Hyperplanes[30]

There are several possible hyperplanes that could be preferred to separate the data points of two classes. The target is to find a plane with a maximum margin, i.e., the maximum distance between the two classes data points. The margin distance should be maximized to provide some clarification so that data points can be identified with greater confidence[30].

Support vectors are the data points close to the hyperplane. Support Vectors influence the position and orientation of hyperplane. Deleting the support vectors will alter the hyperplane position, and using support vectors will maximize the classifier's margin[30].

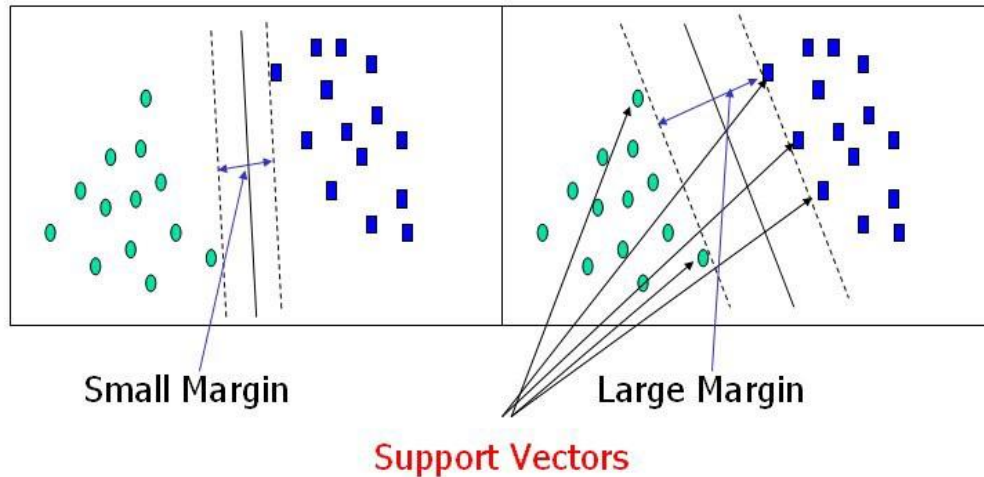
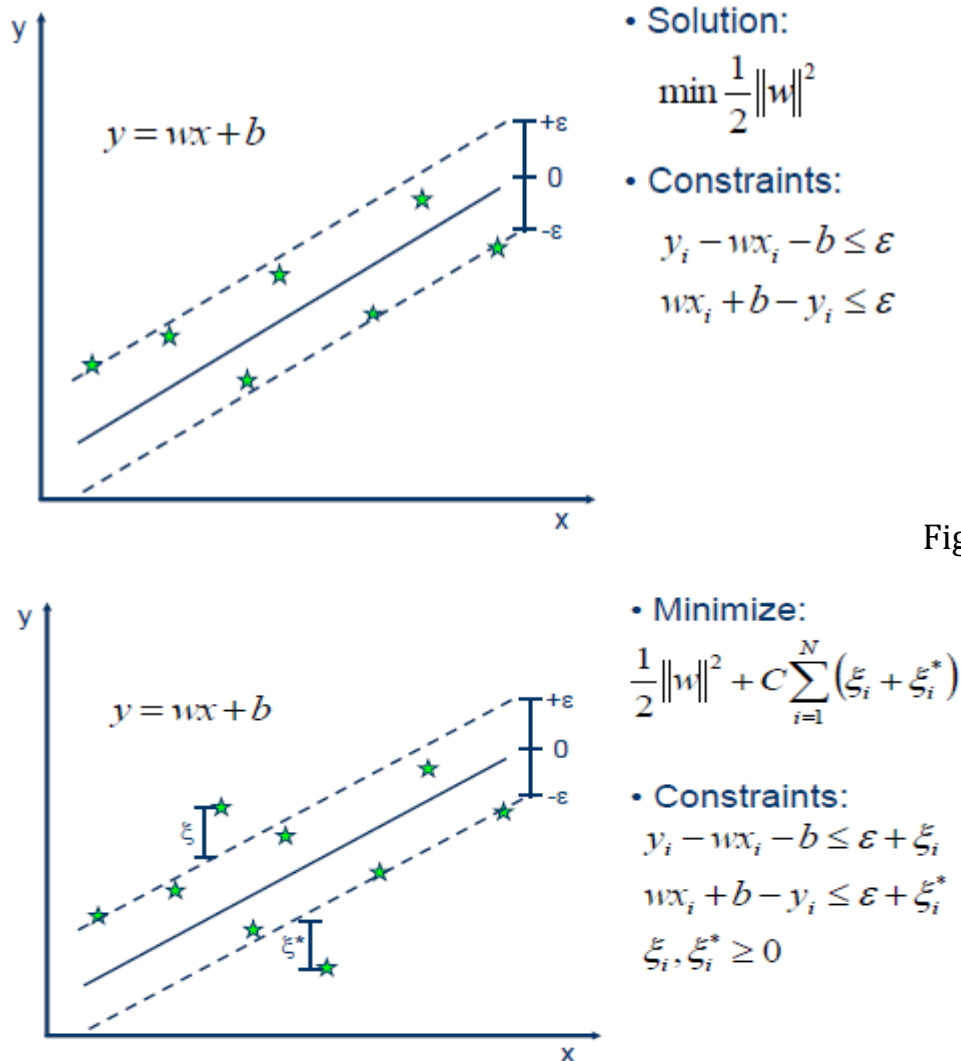


Figure 5: Support Vectors[30]

Support Vector Machine can be used as a regression method, the Support Vector Regression (SVR) concept as the SVM, with some different changes. It is difficult to predict the information with infinite possibilities because the output is a real number.

“In the case of regression, a tolerance margin (epsilon) is set to the SVM that would have already been expected from the problem. The main idea is always the same: to minimize the error, to individualize the hyperplane, which maximizes the margin, taking in to account that some of the error is tolerated”[31].



Figure

Figure 6: Training the SVM[31]

“Where  $x_i$  is a training sample with target value  $y_i$ .

The  $\langle w, x_i \rangle + b$  is the prediction for that sample.

The  $\varepsilon$  is a free parameter that serves as a threshold: all predictions have to be within an  $\varepsilon$  range of the true predictions”[32]

“The concept of slack variables is simple: for any value that falls outside of  $\varepsilon$ , we can denote its deviation from the margin as  $\xi$ .”[33]



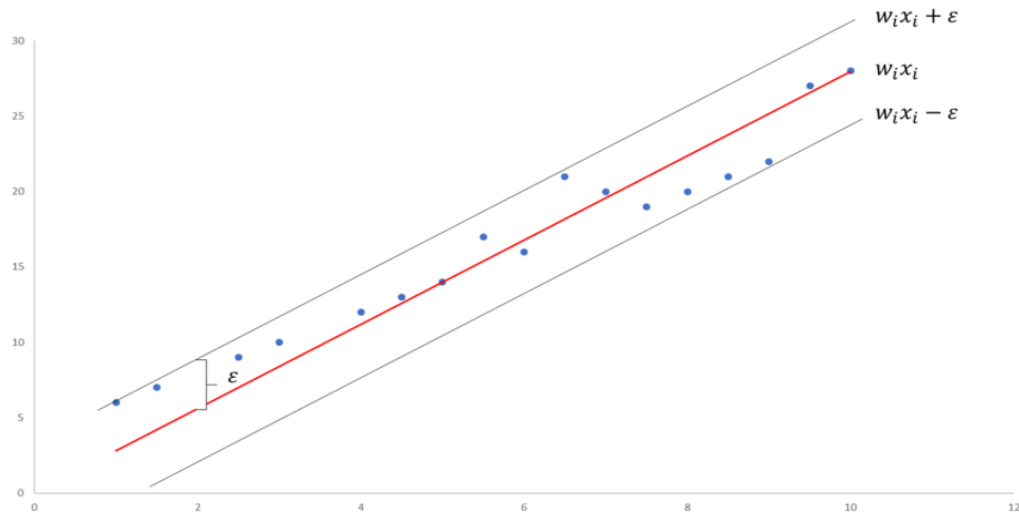


Figure 7: Illustrative Example of Simple SVR[33]

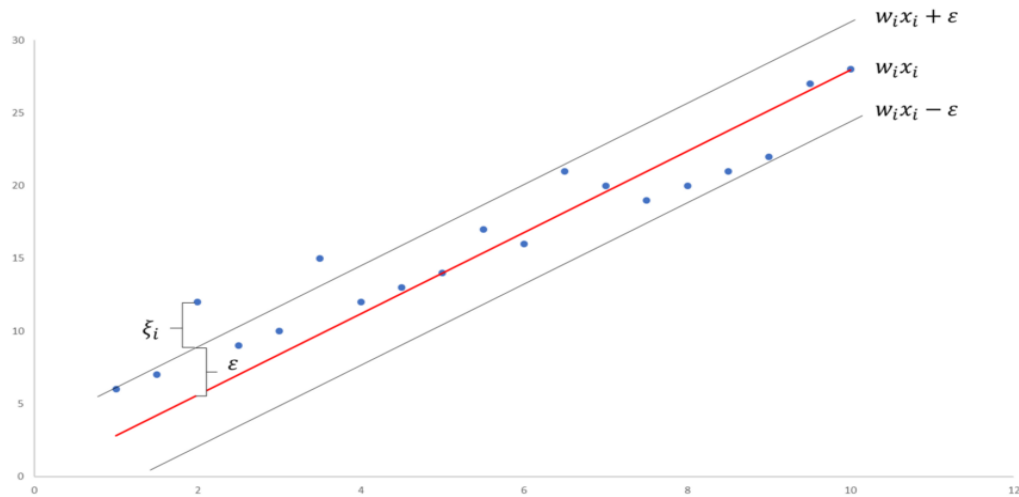


Figure 8: Illustrative Example of SVR with Slack Variables[33]

Among all the four algorithms, random forest performance is very promising when compared to others. Random forest is considered the best model by examining the metric root mean square error because the lower root mean square means the regression line is close to data points and indicates the better fit. Also, random forest limits the overfitting because the performance

of the random forest will not decrease as the number of trees goes on increasing. The SVM algorithm performance is inferior when compared to other algorithms may be due to the presence of so many data points, and SVM takes a long time to process.

### 3.1.5 Hyperparameter Tuning

Parameters describing the architecture of the model are referred to as the hyperparameters. This method of looking for the ideal architecture of the model is referred to as hyperparameter tuning[34].

“Hyperparameters are important because they directly control the behavior of the training algorithm and have a significant impact on the performance of the model is being trained.”[35]

“The process of finding the most optimal hyperparameters in machine learning is called hyperparameter optimization.”[35]

The hyperparameter tuning is performed only on the training data, and test data is kept aside for final evaluation. According to the user's parameter space, there will be  $n$  different parameter combinations ( $n$  different models). Each combination of parameters is  $k$ -fold cross-validation, and we obtain the average accuracy for that combination. The parameter combination with the maximum  $k$  times' average accuracy will be selected as the best parameter of the machine learning algorithm. There are many hyper-parameter tuning techniques. The mostly used techniques for tuning are Grid Search CV and Randomized Search CV.

**Grid Search CV** – In this method, each combination of preset values is given in the form of a parameter grid, which is a list of dictionaries. When there are more hyperparameters, the number of evaluations for each additional parameter will increase exponentially.

**Randomized Search CV** – Random combinations of hyperparameters are used to find the optimal parameters. It tries to give a certain range of random values in the form of parameter distribution

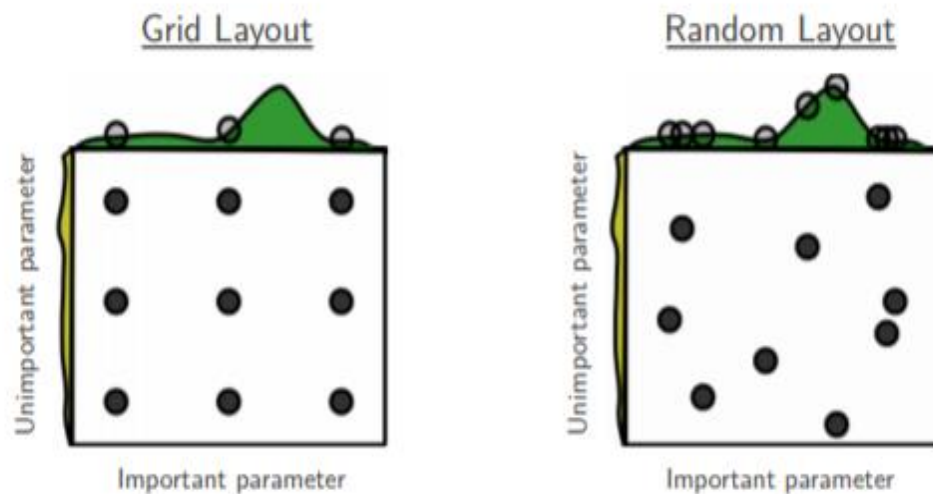


Figure 9: GridSearchCV and RandomSearchCV[36]

The main difference between GridSearchCv and RandomSearchCv is in GridSearchCv a grid of hyperparameter values can be setup and use those parameters for training the models. Where as in RandomSearchCv initially sets up a grid of hyperparameter values and selects a combination randomly to train the model. For this project, GridSearchcv is used for hyperparameter tuning. Hyperparameter tuning is performed in order to increase the performance of the algorithm by considering only certain grid of parameters. The GridSearchCV function is imported from the sklearn package.

## 4 Results

This chapter presents the evaluation of the models carried out.

Firstly, the dependent and the independent features are shown in Figure 10

```
x=comb_df[['Biomass','pCO2offlineBioreactor','Total_Cell_Density','VCD']]
```

```
y=comb_df['IgG']
```

Figure 10 : Dependent and independent features

The data set is divided into 70% for data training and 30% for data testing as shown in Figure 11

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(x,y, test_size = 0.30, random_state = 0)
```

```
print(X_train.shape, y_train.shape)  
print (X_test.shape, y_test.shape)
```

```
(257, 4) (257,)  
(111, 4) (111,)
```

Figure 11 : Dividing the data in to training data and testing data

### 4.1 Multiple Linear Regression

The LinearRegression() function is imported from the Sklearn.  
The coefficients of the independent variables are show in Figure 12

## Results

```
coeff_df = pd.DataFrame(regressor.coef_, x.columns, columns=['Coefficient'])  
coeff_df
```

	Coefficient
<b>Biomass</b>	1589.459268
<b>pCO2OfflineBioreactor</b>	194.417016
<b>Total_Cell_Density</b>	115.082522
<b>VCD</b>	5576.883281

Figure 12 : Coefficients of independent variables

The actual ,predicted values, the metrics such as mean absolute error, mean squared error, root mean squared error and accuracy score of the model is shown in the Figure 13

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)  
Actual    Predicted  
106  16822.0  14687.066217  
260  17218.0  14722.560653  
45    97.0  -1649.669056  
26  11198.0  14991.155082  
78   3338.0  1867.456415  
..    ...    ...  
145  22183.0  14212.720426  
319  16214.5  14975.206467  
235  18657.0  18536.623415  
212   3741.0   3657.236439  
103  10583.0  16761.119032  
  
[111 rows x 2 columns]  
Model Score: 0.7626768404163018  
Mean Absolute Error: 2695.8191417367907  
Mean Squared Error: 11243754.408222185  
Root Mean Squared Error: 3353.1707991425346
```

Figure 13 : Result of the MLR algorithm

The root mean squared error for MLR is 3353.17 and the accuracy score of the model is 76%

The actual values(true values) and the predicted values of MLR is shown in the Figure 14

## Results

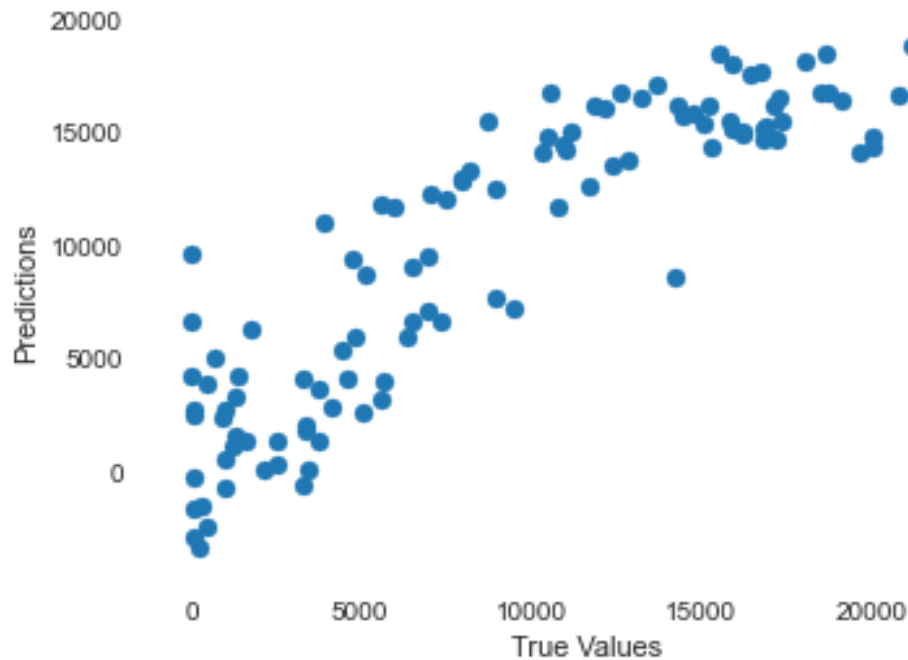


Figure 14 : Actual vs predicted values of MLR

### 4.2 Decision Tree

The `DecisionTreeRegressor()` function is imported from the Sklearn.

The actual ,predicted values, the metrics such as mean absolute error, mean squared error, root mean squared error and accuracy score of the model is shown in the Figure 15

The root mean squared error for decision tree is 3002.86 and the accuracy score of the model is 80%

## Results

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,  
                      max_leaf_nodes=None, min_impurity_decrease=0.0,  
                      min_impurity_split=None, min_samples_leaf=1,  
                      min_samples_split=2, min_weight_fraction_leaf=0.0,  
                      presort=False, random_state=None, splitter='best')
```

	Actual	Predicted
106	16822.0	11747.0
260	17218.0	11934.0
45	97.0	109.0
26	11198.0	11727.0
78	3338.0	2751.0
..	...	...
145	22183.0	22863.0
319	16214.5	16937.5
235	18657.0	15083.0
212	3741.0	3778.0
103	10583.0	11238.0

Model Score: 0.809673570672699  
Mean Absolute Error: 1851.4324324324325  
Mean Squared Error: 9017171.49099099  
Root Mean Squared Error: 3002.8605513728057

Figure 15 : Result of the Decision Tree algorithm  
The actual values(true values) and the predicted values of decision tree is shown in the Figure 16

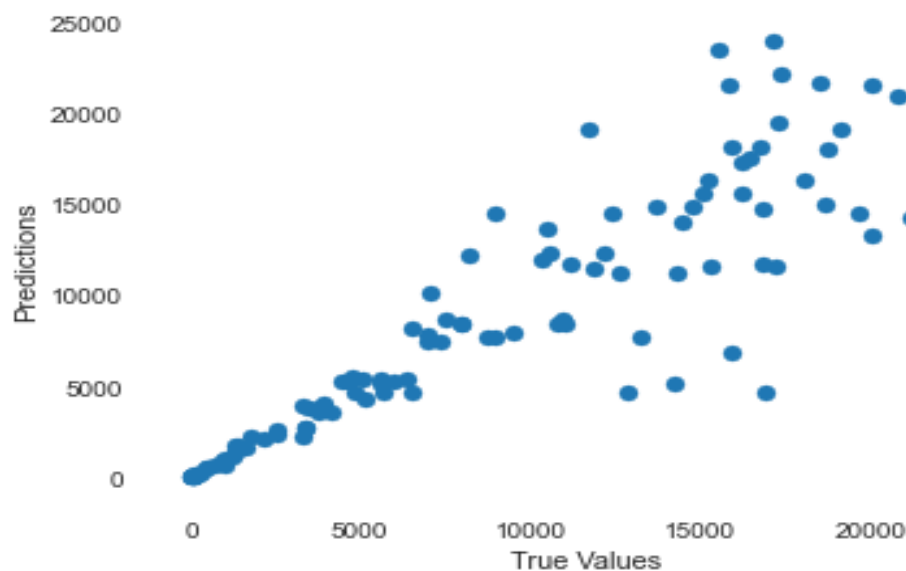


Figure 16 : Actual vs predicted values of Decision Tree

### 4.3 Random Forest

The RandomForestRegressor() function is imported from the Sklearn.

The actual, predicted values, the metrics such as mean absolute error, mean squared error, root mean squared error and accuracy score of the model is shown in the f Figure 17

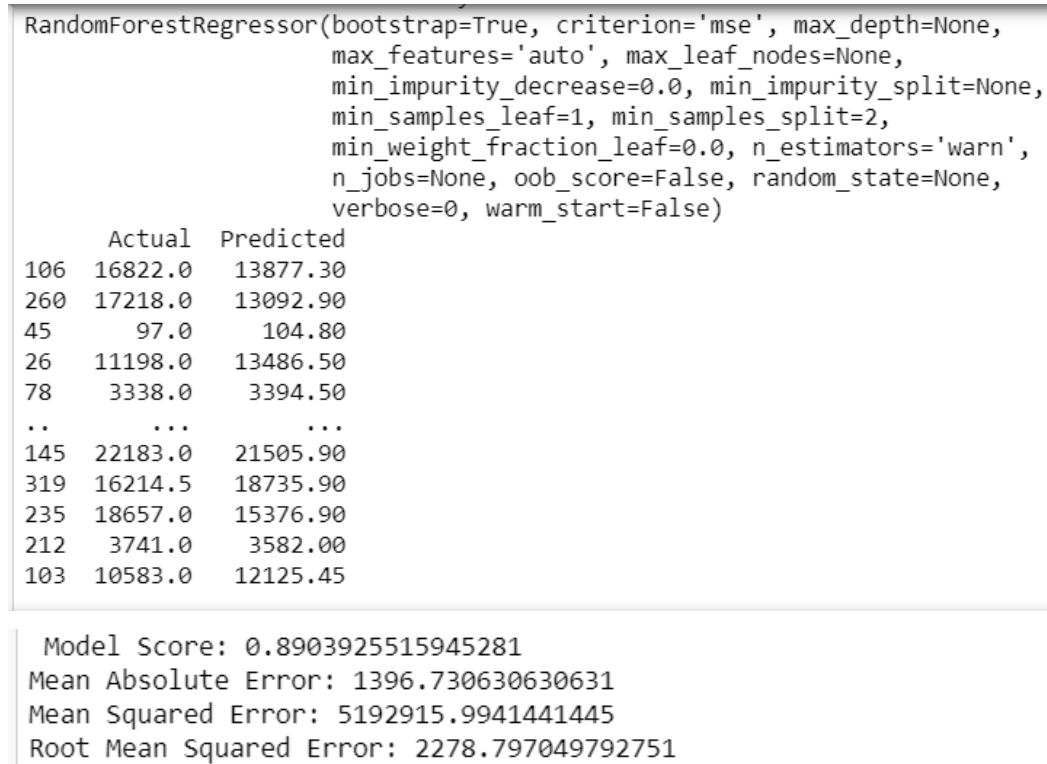


Figure 17 : Result of the Random Forest algorithm

The root mean squared error for decision tree is 2278.79 and the accuracy score of the model is 89%

The actual values(true values) and the predicted values of decision tree is shown in the Figure 18



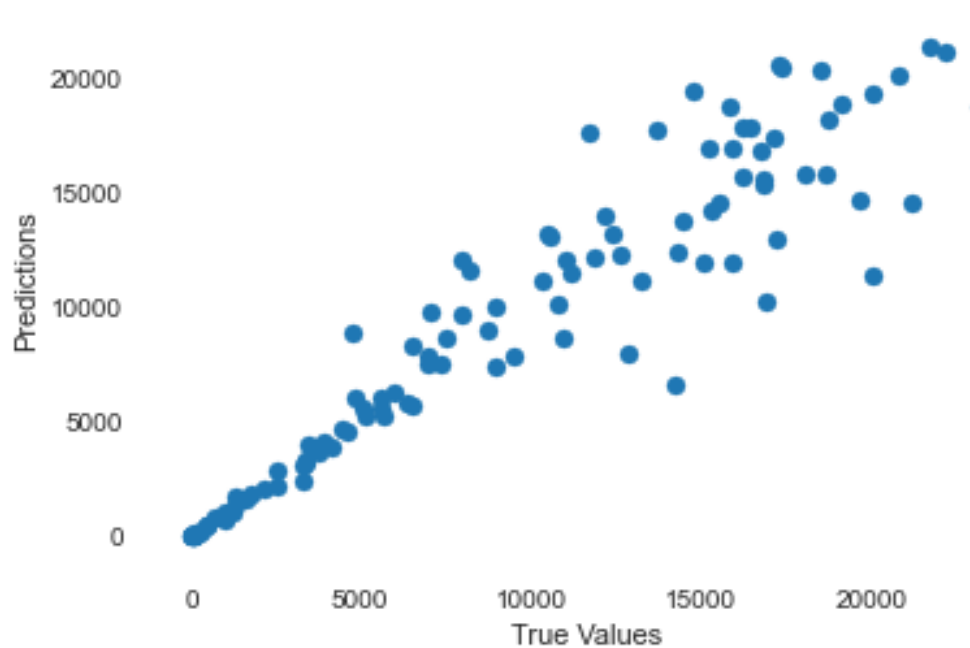


Figure 18 : Actual vs predicted values of Random Forest

### 4.4 Support Vector Machines

The `SVR()` function is imported from the Sklearn.

The actual, predicted values, the metrics such as mean absolute error, mean squared error, root mean squared error and accuracy score of the model is shown in the Figure 19

The root mean squared error for decision tree is 7046.99 and the accuracy score of the model is not good when compared to other algorithms.

## Results

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,  
    gamma='auto_deprecated', kernel='rbf', max_iter=-1, shrinking=True,  
    tol=0.001, verbose=False)  
    Actual    Predicted  
106 16822.0  7810.654983  
260 17218.0  7810.236320  
45   97.0    7808.098968  
26  11198.0  7810.803463  
78  3338.0   7808.316782  
..  
145 22183.0  7811.133419  
319 16214.5  7813.488890  
235 18657.0  7810.046306  
212 3741.0   7807.084426  
103 10583.0  7809.389857  
Model Score: -0.04818344423457721  
Mean Absolute Error: 6061.301178190523  
Mean Squared Error: 49660206.961729616  
Root Mean Squared Error: 7046.999855380275
```

Figure 19 : Result of the SVM algorithm  
The actual values(true values) and the predicted values of decision tree is shown in the Figure 20

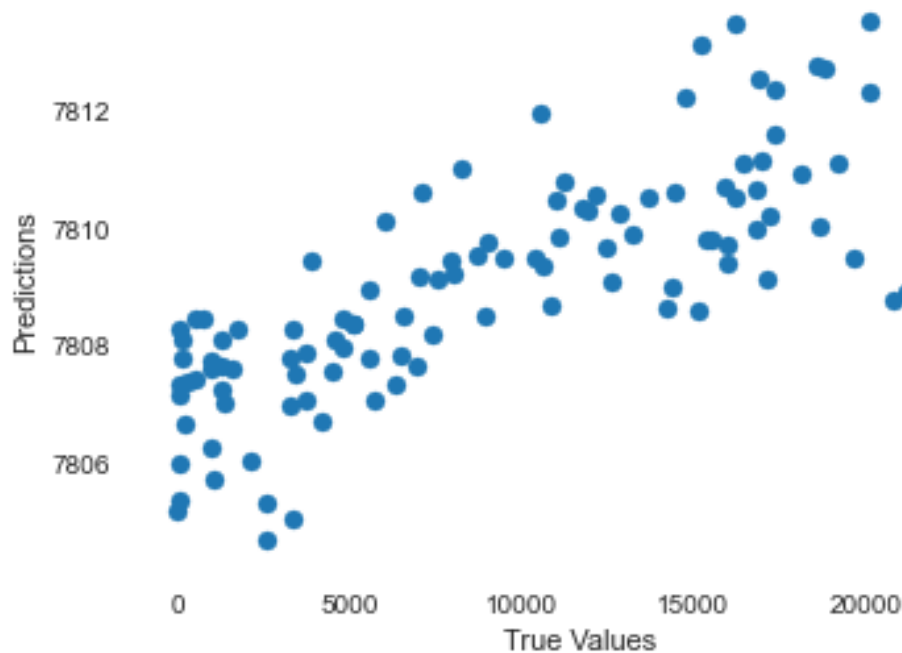


Figure 20 : Actual vs predicted values of SVM

## Results

The Accuracy score of all the algorithms is shown in Figure 21

Test Score	
LinearRegression	0.762677
Decision Tree	0.809674
Random Forset	0.890393
Support Vector Machine	-0.048183

Figure 21 : Accuracy score of all the algorithms

The visualizations of the performance of the algorithms are shown in Figure 22

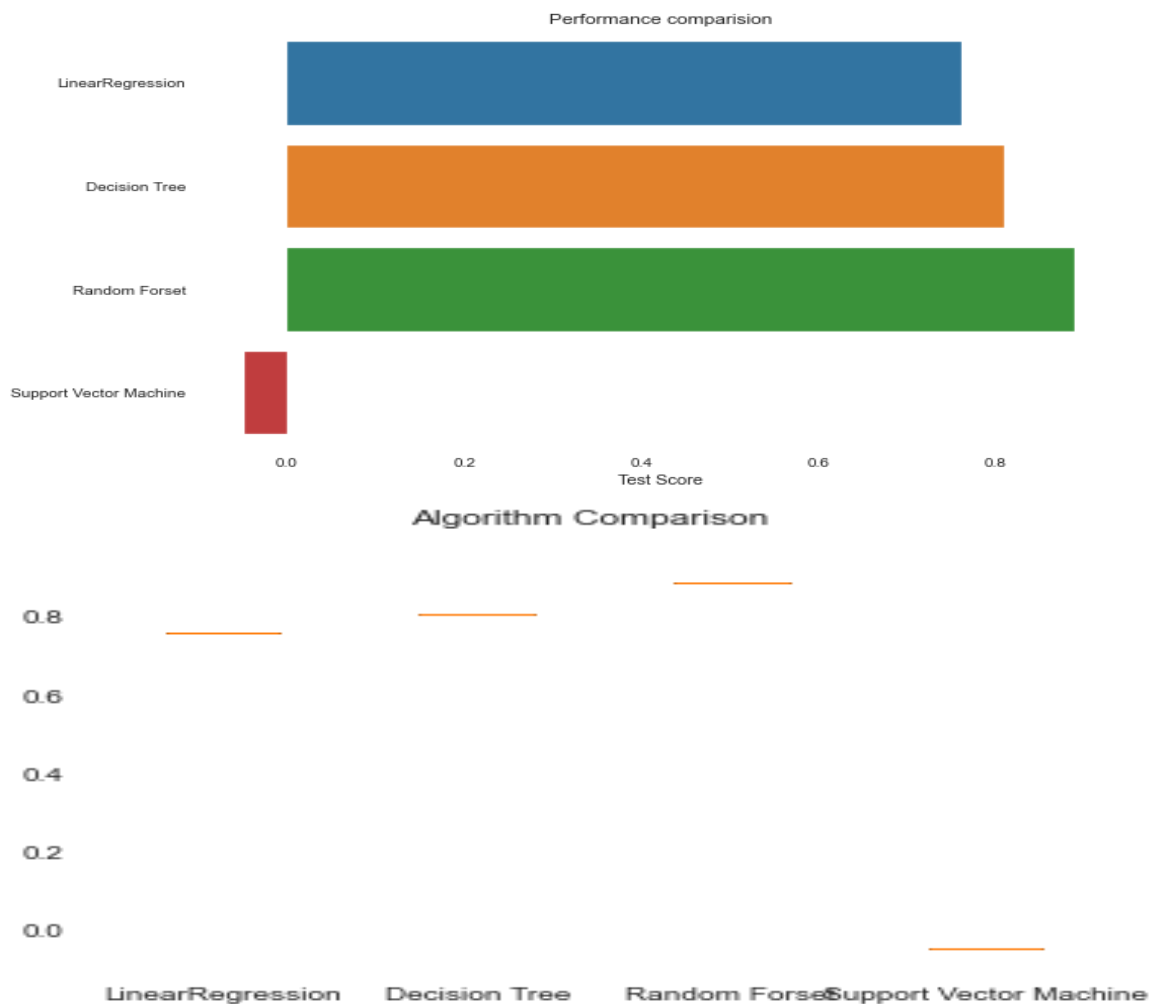


Figure 22 : Performance comparison of all the algorithms

## 4.5 Hyperparameters Tuning

The random forest performance is very good among all the algorithms. So hyperparameter tuning is applied to random forest.

The GridSearchCV() function is imported from SKlearn.

The following are the parameters chosen for tuning the model shown in the Figure 23

```
# Create the parameter grid
rf_param_grid = {
    'min_samples_leaf': [1, 2, 3],
    'min_samples_split': [2, 3, 5, 10],
    'n_estimators': [100, 300, 500]}
```

Figure 23 : parameter grid for hyperparameter tuning

The actual ,predicted values, the metrics such as mean absolute error, mean squared error, root mean squared error and accuracy score of the model is shown in Figure 24

	Actual	Predicted
<b>106</b>	16822.0	13573.679020
<b>260</b>	17218.0	13068.112179
<b>45</b>	97.0	97.922500
<b>26</b>	11198.0	11937.831476
<b>78</b>	3338.0	3391.084071
...	...	...
<b>145</b>	22183.0	20718.819103
<b>319</b>	16214.5	18628.729984
<b>235</b>	18657.0	16236.259111
<b>212</b>	3741.0	3655.337083
<b>103</b>	10583.0	13269.186810

Mean Absolute Error: 1406.1882377377376  
 Mean Squared Error: 5054210.984510865  
 Root Mean Squared Error: 2248.157241945248

Score: 0.8933202134716132

Figure 24 : Result of Random Forest algorithm after hyperparameter tuning

## Results

The actual values(true values) and the predicted values of decision tree is shown in the Figure 25

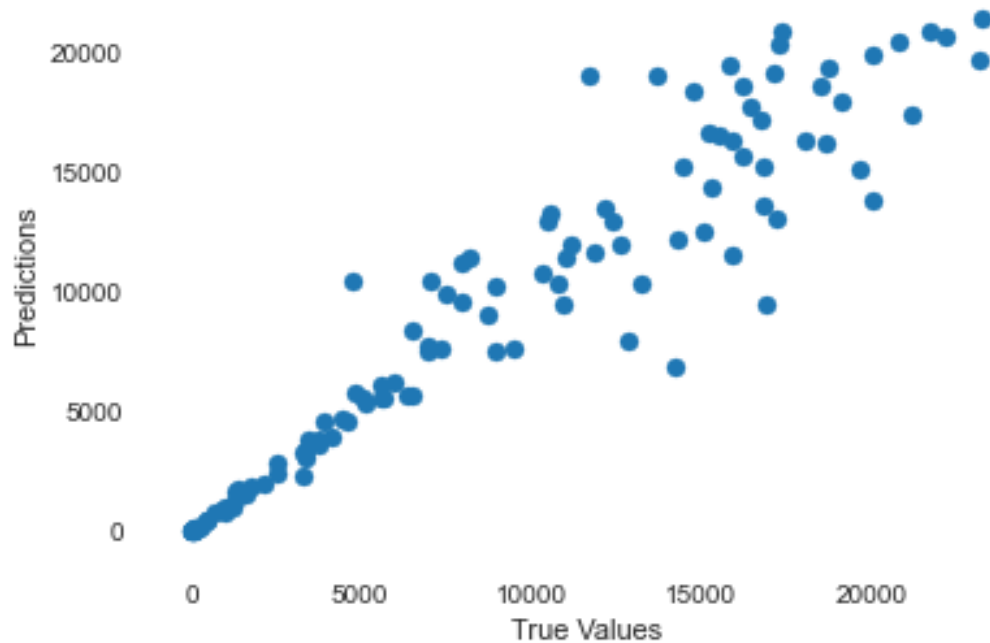


Figure 25 : Actual vs predicted values of Random Forest after hyperparameter tuning

The performance of random forest is increased after hyperparameter tuning.

## 5 Conclusion and Future Scope

This section enumerates the conclusions of the phase of work and includes the following sections:

1. Discussions
2. Conclusion
3. Future work

### 5.1 Discussion

Here are some of the problems encountered during the implementation of the project

1. For every column outliers are detected, performance-wise, there is no significant difference observed when replaced with the median.
2. Among all the algorithms, the SVM model performance is inferior compared to other models; this may occur due to a large volume of datasets because SVM requires much time to process.

### 5.2 Conclusion

Initially, all the data pre-processing steps were applied to the dataset to make it a valid input for various machine learning and optimization algorithms. Out of a range of machine learning algorithms, random forest model performance is the best among linear regression, decision tree, and SVM models. The hyper-parameter optimization using the Grid Search CV method applied to the random forest model showed improved performance after parameter tuning.

### 5.3 Future Scope

Some possible areas of future work could be from either the data preparation phase or hyperparameter tuning of the various models. Different techniques for handling missing values, such as imputation, may be used to test the models' performance. The random search CV technique can be applied instead of the grid search CV method. It can also implement the various combinations of hyper-parameters to improve the performance of machine learning models for predicting the IgG.


## Bibliography

- [1] “Janssen in Ireland,” *Janssen Ireland*. <https://www.janssen.com/ireland/about-us/quick-facts-about-janssen> (accessed Aug. 07, 2020).
- [2] G. Vidarsson, G. Dekkers, and T. Rispen, “IgG Subclasses and Allotypes: From Structure to Effector Functions,” *Front. Immunol.*, vol. 5, Oct. 2014, doi: 10.3389/fimmu.2014.00520.
- [3] “Guidance for Industry PAT - A Framework for Innovative Pharmaceutical Development, manufacturing, and Quality Assurance,” p. 19.
- [4] K. Pramod, M. A. Tahir, N. A. Charoo, S. H. Ansari, and J. Ali, “Pharmaceutical product development: A quality by design approach,” *Int. J. Pharm. Investig.*, vol. 6, no. 3, pp. 129–138, 2016, doi: 10.4103/2230-973X.187350.
- [5] “Biological | Definition of Biological by Oxford Dictionary on Lexico.com also meaning of Biological,” *Lexico Dictionaries | English*. <https://www.lexico.com/en/definition/biological> (accessed Aug. 02, 2020).
- [6] “Biopharmaceutical,” *Wikipedia*. Apr. 28, 2020, Accessed: Jun. 14, 2020. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=Biopharmaceutical&oldid=953634896>.
- [7] PnuVax, “The Difference between Pharmaceuticals and Biopharmaceuticals,” *Medium*, Oct. 12, 2019. <https://medium.com/@pnuvax/the-difference-between-pharmaceuticals-and-biopharmaceuticals-66e8967532ca> (accessed Aug. 02, 2020).
- [8] I. Marison, “Overview of Upstream and Downstream Processing of Biopharmaceuticals,” p. 28.
- [9] “(39) Biopharmaceutical production process - YouTube.” <https://www.youtube.com/watch?v=w3MWkpox-Yo> (accessed Jun. 14, 2020).
- [10] “Bioprocess,” *Wikipedia*. Mar. 24, 2020, Accessed: Jun. 14, 2020. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=Bioprocess&oldid=947143854>.
- [11] “substrates,” *The Free Dictionary*. Accessed: Aug. 23, 2020. [Online]. Available: <https://medical-dictionary.thefreedictionary.com/substrates>.



- [12] P. Gronemeyer, R. Ditz, and J. Strube, "Trends in Upstream and Downstream Process Development for Antibody Manufacturing," *Bioengineering*, vol. 1, no. 4, Art. no. 4, Dec. 2014, doi: 10.3390/bioengineering1040188.
- [13] P. A. J. Rosa, I. F. Ferreira, A. M. Azevedo, and M. R. Aires-Barros, "Aqueous two-phase systems: A viable platform in the manufacturing of biopharmaceuticals," *J. Chromatogr. A*, vol. 1217, no. 16, pp. 2296–2305, Apr. 2010, doi: 10.1016/j.chroma.2009.11.034.
- [14] A. M. Azevedo, P. A. J. Rosa, I. F. Ferreira, and M. R. Aires-Barros, "Chromatography-free recovery of biopharmaceuticals through aqueous two-phase processing," *Trends Biotechnol.*, vol. 27, no. 4, pp. 240–247, Apr. 2009, doi: 10.1016/j.tibtech.2009.01.004.
- [15] A. F. Jozala *et al.*, "Biopharmaceuticals from microorganisms: from production to purification," *Braz. J. Microbiol.*, vol. 47, pp. 51–63, Dec. 2016, doi: 10.1016/j.bjm.2016.10.007.
- [16] "Chapter 11 - Downstream Processing," [http://154.73.92.203:8069/fr\\_FR/slides/embed/17?page=1](http://154.73.92.203:8069/fr_FR/slides/embed/17?page=1) (accessed Aug. 23, 2020).
- [17] "Downstream processing," *Wikipedia*. Jul. 05, 2020, Accessed: Aug. 23, 2020. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Downstream\\_processing&oldid=966180646](https://en.wikipedia.org/w/index.php?title=Downstream_processing&oldid=966180646).
- [18] M. Schuld, I. Sinayskiy, and F. Petruccione, "An introduction to quantum machine learning," *Contemp. Phys.*, vol. 56, no. 2, pp. 172–185, Apr. 2015, doi: 10.1080/00107514.2014.964942.
- [19] G. Magoulas, "Machine Learning In Medical Applications," May 2000.
- [20] R. Burbidge, M. Trotter, B. Buxton, and S. Holden, *Drug*. 2000.
- [21] U. S. G. A. Office, "Using Regression Analysis To Estimate Costs," Aug. 1974, Accessed: Aug. 10, 2020. [Online]. Available: <https://www.gao.gov/products/090875>.
- [22] S. Mishra, D. Mishra, and G. Santra, "Applications of Machine Learning Techniques in Agricultural Crop Production: A Review Paper," *Indian J. Sci. Technol.*, vol. 9, Oct. 2016, doi: 10.17485/ijst/2016/v9i38/95032.
- [23] K. Kaur, "Machine Learning: Applications in Indian Agriculture," vol. 5, no. 4, p. 3.
- [24] "(PDF) Housing Value Forecasting Based on Machine Learning Methods," *ResearchGate*. [https://www.researchgate.net/publication/270627564\\_Housing\\_Value\\_Forecasting\\_Based\\_on\\_Machine\\_Learning\\_Methods](https://www.researchgate.net/publication/270627564_Housing_Value_Forecasting_Based_on_Machine_Learning_Methods) (accessed May 17, 2020).

## Bibliography

- [25] “In-process Control (IPC),” *Contract Pharma*. [https://www.contractpharma.com/contents/view\\_glossary/2013-03-19/in-process-control-ipc/](https://www.contractpharma.com/contents/view_glossary/2013-03-19/in-process-control-ipc/) (accessed Aug. 23, 2020).
- [26] W. Kenton, “How Multiple Linear Regression Works,” *Investopedia*. <https://www.investopedia.com/terms/m/mlr.asp> (accessed Aug. 13, 2020).
- [27] “Machine Learning with Python: Decision Trees in Python.” [https://www.python-course.eu/Decision\\_Trees.php](https://www.python-course.eu/Decision_Trees.php) (accessed Aug. 13, 2020).
- [28] G. M. K, “Machine Learning Basics: Decision Tree Regression,” *Medium*, Jul. 18, 2020. <https://towardsdatascience.com/machine-learning-basics-decision-tree-regression-1d73ea003fda> (accessed Aug. 23, 2020).
- [29] G. M. K, “Machine Learning Basics: Random Forest Regression,” *Medium*, Jul. 18, 2020. <https://towardsdatascience.com/machine-learning-basics-random-forest-regression-be3e1e3bb91a> (accessed Aug. 23, 2020).
- [30] R. Gandhi, “Support Vector Machine — Introduction to Machine Learning Algorithms,” *Medium*, Jul. 05, 2018. <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47> (accessed Aug. 13, 2020).
- [31] “Support Vector Regression.” [https://www.saedsayad.com/support\\_vector\\_machine\\_reg.htm](https://www.saedsayad.com/support_vector_machine_reg.htm) (accessed Aug. 13, 2020).
- [32] “Support vector machine,” *Wikipedia*. Aug. 12, 2020, Accessed: Aug. 13, 2020. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Support\\_vector\\_machine&oldid=972516247](https://en.wikipedia.org/w/index.php?title=Support_vector_machine&oldid=972516247).
- [33] T. Sharp , “An Introduction to Support Vector Regression (SVR),” *Medium*, May 06, 2020. <https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2> (accessed Aug. 13, 2020).
- Jordan, Nov. 02, 2017. <https://www.jeremyjordan.me/hyperparameter-tuning/> (accessed Aug. 14, 2020).
- [34] “Hyperparameter tuning for machine learning models,” *Jeremy Jordan*, Nov. 02, 2017. <https://www.jeremyjordan.me/hyperparameter-tuning/> (accessed Aug. 14, 2020)
- [35] Prabhu, “Understanding Hyperparameters and its Optimisation techniques,” *Medium*, Jul. 03, 2018.

## Bibliography

<https://towardsdatascience.com/understanding-hyperparameters-and-its-optimisation-techniques-f0debba07568> (accessed Aug. 14, 2020).

[36] J. Bergstra and Y. Bengio, “Random Search for Hyper-Parameter Optimization,” p. 25.

[37]

[https://ec.europa.eu/health/sites/health/files/files/pharmacos/pharmpack\\_12\\_2008/communication/citizens\\_summary\\_communication\\_en.pdf](https://ec.europa.eu/health/sites/health/files/files/pharmacos/pharmpack_12_2008/communication/citizens_summary_communication_en.pdf)

[38] <https://www.efpia.eu/media/25055/the-pharmaceutical-industry-in-figures-june-2016.pdf>

# APPENDIX

## APPENDIX A

### Tables:

#### 1. Discrete data sheet columns

Column Number	Column Name	New name of the column
1	\$BatchID	BatchID
2	\$UnitID	UnitID
3	Date_1	Date
4	CELL_CULTURE_DAY_NUM	cell_culture
5	MATERIAL_OUT	Material_out
6	BaseMidnightFlowrate	Base_Midnight
7	BiomassMidnightFlowrate	Biomass
8	EDTAHarvestFlowRatio	EDTA_Harvest
9	EDTAMidnightFlowrate	EDTA_Midnight
10	HarvestMidnightFlowrate	Harvest_Midnight
11	MediaMidnightFlowrate	Media_Midnight
12	NephIgGATF1_EDW_LIMS	NephIgGATF1
13	NephIgGATF2_EDW_LIMS	NephIgGATF2
14	NephIgGBioreactor_EDW_LIMS	NephIgGBioreactor
15	pCO2OfflineBioreactor	The name of the column is not changed
16	RetentionRateATF1_EDW_LIMS	RetentionRateATF1

17	RetentionRateATF2_EDW_LIMS	RetentionRateATF2
18	RetentionRateATF2_MES	The name of the column is not changed
19	TotalCellDensityBioreactor	Total_Cell_Density
20	ViabilityATF1	The name of the column is not changed
21	ViabilityATF2	The name of the column is not changed
22	ViabilityBioreactor	The name of the column is not changed
23	ViableCellDensityATF1	The name of the column is not changed
24	ViableCellDensityATF2	The name of the column is not changed
25	ViableCellDensityBioreactor	VCD
26	CuSum(NephIgGBioreactor_EDW_LIMS)	IgG

Table 3 : Discrete data columns

## 2. Continuous data sheet columns

Column Number	Column Name
1	Date
2	UnitID
3	Batch
4	BR Weight
5	Base Scale Weight
6	Biomass Scale Weight

## APPENDIX

7	EDTA Scale Weight
8	Harvest Filtrate Weight
9	Agitation Speed_SI
10	BR Pressure
11	pH Probe Active
12	pH Probe 1
13	pH Probe 2
14	PA Overlay Flow
15	PA Lower Sparge Flow
16	O2 Sparge Flow
17	N2 Sparge Flow
18	CO2 Sparge Flow
19	CO2 Overlay Flow
20	DO Probe Active
21	DO Probe 1
22	DO Probe 2
23	Perfusion Media Flow
24	Perfusion Flow Control_Tot Int Flow
25	BR Temp Probe Active
26	BR Temp Probe 1
27	BR Temp Probe 2
28	Jacket Temp
29	ATF1 Inlet vs Ret Diff Press
30	ATF1 Inlet vs Filt Diff Press
31	ATF1 Ret vs Pneum Diff Press
32	ATF2 Inlet vs Ret Diff Press

## APPENDIX

33	ATF2 Inlet vs Filt Diff Press
34	ATF2 Ret vs Pneum Diff Press
35	ATF1 Inlet Press
36	ATF1 Filtrate Press
37	ATF1 Retentate Press
38	ATF1 Pneumatic Press
39	ATF2 Inlet Press
40	ATF2 Filtrate Press
41	ATF2 Retentate Press
42	ATF2 Pneumatic Press
43	ATF1 Weight
44	ATF2 Weight
45	ATF1 Differential Weight
46	ATF2 Diff Weight
47	BR Temp Probe Deviation
48	pH Probe Deviation
49	DO Probe Deviation
50	PA Upper Sparge Flow
51	Perfusion Media Pump Speed_PV
52	Perfusion Flow Control_Count
53	Base Pump Speed_PV
54	Biomass Pump Speed_PV
55	Harvest Pump Speed
56	Harvest Flow Rate
57	ATF1 Pump Speed_PV
58	ATF2 Pump Speed_PV

## APPENDIX

59	BR Hall Relative Humidity
60	BR Hall Temperature
61-90	0 – 0.28

Table 4 : Continuous data columns

## APPENDIX B

### Packages Used

**SCIKIT-LEARN:** SCIKIT-LEARN is a library in python mainly used for machine learning and data analysis such as Preprocessing, Classification, Regression, Clustering, Dimensionality reduction, feature extraction, and so on.

**NumPy:** Mostly used for easy processing and manipulation of Multidimensional array objects.

**Pandas:** Pandas is an open-source tool used for analysis and the manipulation of data. Pandas is simple, versatile, efficient, and easy to use.

**Matplotlib:** Matplotlib is a visualization library available in python to design animated and interactive visualizations.

**Seaborn:** Seaborn is a matplotlib based python visualization library. It offers a high-level framework for drawing attractive statistical graphics.



## APPENDIX C

## Code Snippets

```

import pandas as pd
import numpy as np
import pandas_profiling as pp
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.feature_selection import VarianceThreshold
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn import metrics
from sklearn.model_selection import cross_val_score

#reading the discrete data
data_sheet1=pd.read_csv("F:/PROJECT/Discrete_Data.csv",low_memory=False)

#reaming the columns
data_sheet1.rename(columns={'$BatchID': 'BatchID', '$UnitID': 'UnitID','Date_1':'Date','CELL_CULTURE_DAY_NUM':'cell_culture',
                           'MATERIAL_OUT':'Material_out','BaseMidnightFlowrate':'Base_Midnight','BiomassMidnightFlowrate':'Biomass',
                           'EDTAHarvestFlowRatio':'EDTA_Harvest',
                           'EDTAMidnightFlowrate':'EDTA_Midnight','HarvestMidnightFlowrate':'Harvest_Midnight',
                           'MediaMidnightFlowrate':'Media_Midnight',
                           'NephIgGATF1_EDW_LIMS':'NephIgGATF1','NephIgGATF2_EDW_LIMS':'NephIgGATF2',
                           'NephIgGBioreactor_EDW_LIMS':'NephIgGBioreactor',
                           'RetentionRateATF1_EDW_LIMS':'RetentionRateATF1','RetentionRateATF2_EDW_LIMS':'RetentionRateATF2',
                           'TotalCellDensityBioreactor':'Total_Cell_Density','ViableCellDensityBioreactor':'VCD',
                           'CuSum(NephIgGBioreactor_EDW_LIMS)':'IgG'}, inplace=True)

```

## APPENDIX

```
#deleting the date column nan values
data_sheet1=data_sheet1.dropna(axis=0, subset=['Date'])

#considering float columns and imputing with mean
float_cols = list(data_sheet1.select_dtypes('float64').columns)
data_sheet1[float_cols] = data_sheet1[float_cols].fillna( data_sheet1[float_cols].mean())

#Drop duplicates from the dataset
data_sheet1.drop_duplicates(inplace = True)

#NO DUPLICATE ROWS

#applying log transformations to Biomass and VCD
data_sheet1['Biomass']=np.log1p(data_sheet1['Biomass'])
data_sheet1['Biomass']
data_sheet1['VCD']=np.log1p(data_sheet1['VCD'])
data_sheet1['VCD']

#dropping cell_culture column
data_sheet1.drop(["cell_culture"], axis = 1, inplace = True)

#OUTLIERS

#OUTLIERS

import seaborn as sns

sns.boxplot(x=data_sheet1['Base_Midnight'])

#finding IQR and replacing outliers with median
Q1 = data_sheet1['Base_Midnight'].quantile(0.25)
Q3 = data_sheet1['Base_Midnight'].quantile(0.75)
IQR = Q3 - Q1
data_sheet1.loc[data_sheet1['Base_Midnight']>Q3+1.5*IQR,'Base_Midnight'] = np.nan
data_sheet1.loc[data_sheet1['Base_Midnight']< Q1-1.5*IQR,'Base_Midnight'] = np.nan
data_sheet1['Base_Midnight'] = data_sheet1['Base_Midnight'].fillna( data_sheet1['Base_Midnight'].median() )
sns.boxplot(x=data_sheet1['Base_Midnight'])

#outliers are removed
```

## APPENDIX

*#For replacing outliers with median this is the same procedure followed for all the columns.*

*#storing date and BatchID in separate variables*

```
dx=[data_sheet1['Date']]
```

```
dx
```

```
dy=[data_sheet1['BatchID']]
```

```
dy
```

*#dropping 'UnitID','BatchID','Date' values*

```
data_sheet1 = data_sheet1.drop(['UnitID','BatchID','Date'], axis=1)
```

*#correlation matrix*

```
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

```
import seaborn as sns
```

*#correlation matrix*

```
fig, ax = plt.subplots()
```

```
fig.set_size_inches(15, 10)
```

```
sns.heatmap(data_sheet1.corr(),cmap='coolwarm',ax=ax,annot=True,linewidths=2)
```

*#removing the columns which is having same constant vales*

*# <https://stackabuse.com/applying-filter-methods-in-python-for-feature-selection/>*

```
from sklearn.feature_selection import VarianceThreshold
```

```
constant_filter = VarianceThreshold(threshold=0)
```

```
constant_filter.fit(data_sheet1)
```

```
len(data_sheet1.columns[constant_filter.get_support()])
```

```
constant_columns = [column for column in data_sheet1.columns
```

```
if column not in data_sheet1.columns[constant_filter.get_support()]]
```

```
print(len(constant_columns))
```

```
for column in constant_columns:
```

```
    print(column)
```

```
data_sheet1=data_sheet1.drop(constant_columns, axis=1)
```

*#Removing Quasi-Constant features*

## APPENDIX

*#Quasi-constant features, as the name suggests, are the features that are almost constant.*

*#In other words, these features have the same values for a very large subset of the outputs.*

*#Such features are not very useful for making predictions.*

```
qconstant_filter = VarianceThreshold(threshold=0.01)
qconstant_filter.fit(data_sheet1)
len(data_sheet1.columns)

qconstant_columns = [column for column in data_sheet1.columns
                     if column not in data_sheet1.columns[qconstant_filter.get_support()]]

print(len(qconstant_columns))
for column in qconstant_columns:
    print(column)
#correlation matrix
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

fig, ax = plt.subplots()
fig.set_size_inches(15, 10)
sns.heatmap(data_sheet1.corr(),cmap='coolwarm',ax=ax,annot=True,linewidths=2)
#correlation of VCD with other features
cor= data_sheet1[data_sheet1.columns].corr()['VCD']
#correlation of VCD with other features
cor1= data_sheet1[data_sheet1.columns].corr()['IgG']
#storing Date column as frame
dfOb = pd.DataFrame(dx)
dfOb
#applying transpose
Date =dfOb.transpose()
print(Date)
```

## APPENDIX

```
data_sheet1['Date']=Date
#removing the time interval
data_sheet1['Date'] = pd.to_datetime(data_sheet1['Date'],errors='coerce')
data_sheet1['new_date_column'] = data_sheet1['Date'].dt.date
#https://stackoverflow.com/questions/45858155/removing-the-timestamp-from-a-datetime
-in-pandas-dataframe
data_sheet1.drop(["Date"], axis = 1, inplace = True)
dfObj = pd.DataFrame(dy)
dfObj
BatchID =dfObj.transpose()
print(BatchID)
data_sheet1['BatchID']=BatchID

#now reading 3 sheets of the continuous data and combining them by using pd.concat()
data_sheet2=pd.read_csv("F:/PROJECT/Continuous_data_BR301.csv",low_memory=False)
In [103]:
data_sheet3=pd.read_csv("F:/PROJECT/Continuous_data_BR302.csv",low_memory=False)
data_sheet4=pd.read_csv("F:/PROJECT/Continuous_data_BR303.csv",low_memory=False)
df=pd.concat([data_sheet2,data_sheet3,data_sheet4], ignore_index=True,sort=True)
cols = df.columns[df.isnull().mean()>0.6]
print(cols)
#dropping all columns related to threshold
df=df.drop(cols, axis=1)

#dropping unnecessary columns
def drop_unnecessary_columns(dataframe):
    dataframe_copy = dataframe.copy()
    unique_vals = {}
    columns_to_drop = []
    for column in dataframe_copy.columns:
```

## APPENDIX

```
unique_vals[column] = dataframe_copy[column].unique().tolist()
for key,value in unique_vals.items():
    if value[0] == 'Tag not found':
        columns_to_drop.append(key)
dataframe_copy = dataframe_copy.drop(columns_to_drop, axis=1)
return dataframe_copy, unique_vals

br_COPY, cols_unique = drop_unnecessary_columns(df)
cols_unique['INTERVAL COUNTER DAY']
cols_unique['ATF1 Retentate Press']
#replacing these values with zero
strings_to_replace = ['Unit Down', 'Equip Fail', 'Error', 'Resize to show all values']
for column in br_COPY.columns:
    for string_ in strings_to_replace:
        i = br_COPY[((br_COPY[column] == string_))].index
        br_COPY.loc[i, column] = 0
#Drop duplicates from the dataset

br_COPY.drop_duplicates(inplace = True)

#Here i am resetting the index values
br_COPY =br_COPY.reset_index(drop=True)

x=[br_COPY.Date]
y=[br_COPY.Batch]
br_COPY = br_COPY.drop(['UnitID', 'Batch', 'Date'], axis=1)

br_COPY=br_COPY.astype(float)

float_cols = list(br_COPY.select_dtypes('float64').columns)

br_COPY[float_cols] = br_COPY[float_cols].fillna( br_COPY[float_cols].mean() )
#outliers
```

## APPENDIX

```
sns.boxplot(x=br_COPY['ATF1 Differential Weight'])

Q1 = br_COPY['ATF1 Differential Weight'].quantile(0.25)
Q3 = br_COPY['ATF1 Differential Weight'].quantile(0.75)
IQR = Q3 - Q1
br_COPY.loc[br_COPY['ATF1 Differential Weight']>Q3+1.5*IQR,'ATF1 Differential Weight']
= np.nan
br_COPY.loc[br_COPY['ATF1 Differential Weight']< Q1-1.5*IQR,'ATF1 Differential Weight']
= np.nan

br_COPY.isnull().sum()
br_COPY['ATF1 Differential Weight'] = br_COPY['ATF1 Differential Weight'].fillna( br_COPY
['ATF1 Differential Weight'].median() )
sns.boxplot(x=br_COPY['ATF1 Differential Weight'])

#This is the same procedure followed for replacing all the outliers with median

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

fig, ax = plt.subplots()
fig.set_size_inches(15, 10)
sns.heatmap(br_COPY.corr(),cmap='coolwarm',ax=ax,annot=True,linewidths=2)
#removing the columns which is having same constant vales
# https://stackabuse.com/applying-filter-methods-in-python-for-feature-selection/
from sklearn.feature_selection import VarianceThreshold
constant_filter = VarianceThreshold(threshold=0)
constant_filter.fit(br_COPY)
len(br_COPY.columns[constant_filter.get_support()])

constant_columns = [column for column in br_COPY.columns
if column not in br_COPY.columns[constant_filter.get_support()]]
```

## APPENDIX

```
print(len(constant_columns))

for column in constant_columns:
    print(column)

br_COPY=br_COPY.drop(constant_columns, axis=1)
#Removing Quasi-Constant features
#Removing Constant Features using Variance Threshold
#Before we can remove quasi-constant features, we should first remove the constant features.
Execute the following script to do so
qconstant_filter = VarianceThreshold(threshold=0.01)
qconstant_filter.fit(br_COPY)
len(br_COPY.columns)

qconstant_columns = [column for column in br_COPY.columns
                     if column not in br_COPY.columns[qconstant_filter.get_support()]]

print(len(qconstant_columns))

for column in qconstant_columns:
    print(column)
br_COPY=br_COPY.drop(labels=qconstant_columns, axis=1)
#Removing Correlated Features

#Removing Correlated Features using corr() Method
correlated_features = set()
correlation_matrix = br_COPY.corr()

for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > 0.8:
            colname = correlation_matrix.columns[i]
```



## APPENDIX

```
correlated_features.add(colname)

len(correlated_features)

print(correlated_features)

br_COPY=br_COPY.drop(labels=correlated_features, axis=1)

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

fig, ax = plt.subplots()
fig.set_size_inches(15, 10)
sns.heatmap(br_COPY.corr(),cmap='coolwarm',ax=ax,annot=True,linewidths=2)

dfObj = pd.DataFrame(x)
Date =dfObj.transpose()
br_COPY['Date']=Date

br_COPY['Date'] = br_COPY['Date'].fillna( br_COPY['Date'].mode()[0] )
br_COPY['Date'] = pd.to_datetime(br_COPY['Date'],errors='coerce')
br_COPY['new_date_column'] = br_COPY['Date'].dt.date

br_COPY.drop(["Date"], axis = 1, inplace = True)
br_COPY=br_COPY.groupby(br_COPY['new_date_column'],as_index = False).mean()
comb_df = pd.merge(data_sheet1,br_COPY,on='new_date_column')

cor_matrix=comb_df.corr()

cor1= comb_df[comb_df.columns].corr()['VCD']
```

## APPENDIX

```
#considering features greater than 0.5
relevant_features1 = cor1[cor1>0.5]
relevant_features1

cor2= comb_df[comb_df.columns].corr()['IgG']

relevant_features2 = cor2[cor2>0.5]
relevant_features2

print(relevant_features1)
print(relevant_features2)

x=comb_df[['Biomass','pCO2OfflineBioreactor','Total_Cell_Density','VCD']]
y=comb_df['IgG']

#Split the data into 70% training (X_train & y_train) and 30% testing (X_test & y_test) data sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x,y, test_size = 0.30, random_state = 0)

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
(257, 4) (257,)
(111, 4) (111,)
```

## Linear Regression

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
coeff_df = pd.DataFrame(regressor.coef_, x.columns, columns=['Coefficient'])
coeff_df
```

## APPENDIX

```
y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

from sklearn import 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)))

#Accuracy score

print("Score:", regressor.score(X_test, y_test))
## The line / model
plt.scatter(y_test, y_pred)
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

## Decision Tree

*#AS THERE IS HIGH ERROR, LETS TRAIN OTHER REGRESSION MODEL FOR OUR DATA AND*

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

```
DT = DecisionTreeRegressor()
model1 = DT.fit(X_train, y_train)
predictions1 = DT.predict(X_test)
## The line / model
plt.scatter(y_test, predictions1)
plt.xlabel("True Values")
plt.ylabel("Predictions")

df1 = pd.DataFrame({'Actual': y_test, 'Predicted': predictions1})
#Accuracy score
```

## APPENDIX

```
print("Score:", model1.score(X_test, y_test))

from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predictions1))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predictions1))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predictions1)))

#Accuracy score

#performing cross-validation on training set using k-folds cross validation from sklearn.model_
selection
from sklearn.model_selection import cross_val_score
score = cross_val_score(model1, X_train, y_train, scoring = 'neg_mean_squared_error', cv = 10)
tree_rmse_scores = np.sqrt(-score)

print(score)
print(tree_rmse_scores)

def display_scores(score):
    print('scores:', score)
    print('mean:', score.mean())
    print('standard deviation:', score.std())

display_scores(tree_rmse_scores)
```

## Random Forest

```
#TRAINING RANDOM FOREST REGRESSOR FOR MUCH MORE BETTER MODEL PERFORMANCE

from sklearn.ensemble import RandomForestRegressor
```

## APPENDIX

```
RF = RandomForestRegressor()
model2 = RF.fit(X_train, y_train)
predictions2 = RF.predict(X_test)
df1 = pd.DataFrame({'Actual': y_test, 'Predicted': predictions2})

## The line / model
plt.scatter(y_test, predictions2)
plt.xlabel("True Values")
plt.ylabel("Predictions")
#Accuracy score

print("Score:", model2.score(X_test, y_test))

from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predictions2))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predictions2))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predictions2)))

#performing cross-validation on training set using k-folds cross validation from sklearn.model_
selection
from sklearn.model_selection import cross_val_score
score1 = cross_val_score(model2, X_train, y_train, scoring = 'neg_mean_squared_error', cv =
10)
tree_rmse = np.sqrt(-score1)

print(score1)
print(tree_rmse)

def display_scores(score):
    print('scores:', score)
    print('mean:', score.mean())
    print('standard deviation:', score.std())
```

## APPENDIX

```
display_scores(tree_rmse)
```

### SVM

```
from sklearn.svm import SVR
svr = SVR()
model3 = svr.fit(X_train, y_train)
predictions3 = svr.predict(X_test)
df1 = pd.DataFrame({'Actual': y_test, 'Predicted': predictions3})
## The line / model
plt.scatter(y_test, predictions3)
plt.xlabel("True Values")
plt.ylabel("Predictions")
#Accuracy score

print("Score:", model3.score(X_test, y_test))
Score: -0.04818344423457721

from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predictions3))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predictions3))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predictions3)))

#performing cross-validation on training set using k-folds cross validation from sklearn.model_
selection
from sklearn.model_selection import cross_val_score
score2 = cross_val_score(model3, X_train, y_train, scoring = 'neg_mean_squared_error', cv =
10)
svr_rmse_scores = np.sqrt(-score2)
```

## APPENDIX

```
print(score2)
print(svr_rmse_scores)
def display_scores(score):
    print('scores:', score)
    print('mean:', score.mean())
    print('standard deviation:', score.std())
```

```
display_scores(svr_rmse_scores)
```

## Tuning

```
RF.get_params()
rf_grid = RandomForestRegressor(random_state=2)
rf_param_grid = {
    'min_samples_leaf': [1, 2, 3],
    'min_samples_split': [2, 3, 5, 10],
    'n_estimators': [100, 300, 500]}
from sklearn.model_selection import GridSearchCV
# Instantiate the grid search model (n_jobs = -1 sets to use the max number of processors)
rf_grid_search = GridSearchCV(estimator=rf_grid, param_grid=rf_param_grid, cv=3, scoring
='neg_mean_squared_error', n_jobs=-1, verbose=2)
#training the model
rf_grid_search.fit(X_train, y_train)
#TRAINING RANDOM FOREST REGRESSOR FOR MUCH MORE BETTER MODEL PERFORMANC
E
from sklearn.ensemble import RandomForestRegressor
RF = RandomForestRegressor(min_samples_leaf= 2,
    min_samples_split= 2,
    n_estimators= 100)
mod = RF.fit(X_train, y_train)

predict = RF.predict(X_test)
```

## APPENDIX

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': predict})
## The line / model
plt.scatter(y_test, predict)
plt.xlabel("True Values")
plt.ylabel("Predictions")
#Accuracy score

print("Score:", mod.score(X_test, y_test))

from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predict))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predict))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predict)))

#performing cross-validation on training set using k-folds cross validation from sklearn.model_
selection
from sklearn.model_selection import cross_val_score
score1 = cross_val_score(model2, X_train, y_train, scoring = 'neg_mean_squared_error', cv =
10)
tree_rmse = np.sqrt(-score1)

print(score1)
print(tree_rmse)
def display_scores(score):
    print('scores:', score)
    print('mean:', score.mean())
    print('standard deviation:', score.std())

display_scores(tree_rmse)
#displaying all the algorithms results at one place
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```



## APPENDIX

```
from sklearn.svm import SVR

lr = LinearRegression()
dt = DecisionTreeRegressor()
rf = RandomForestRegressor()
svr = SVR()
models = {'LinearRegression':lr, 'Decision Tree':dt, 'Random Forset':rf,
          'Support Vector Machine':svr,}

from sklearn import metrics
#function for the metrics
def eval_model(model):
    print(model)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    test_accuracy = model.score(X_test, y_test)

    df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
    print(df)
    print(" Model Score:", model.score(X_test, y_test))
    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("\n")
    return test_accuracy

test accuracies = []
#printing the test accuracies
for name, model in models.items():
    test_acc = eval_model(model)
    test accuracies.append(test_acc)
    print(f'{name} ---> Test accuracy - {model.score(X_test, y_test)}')
```

## APPENDIX

*#Building one frame for all test accuracies*

```
results = pd.DataFrame(test_accuracies, index=list(models.keys()), columns=["Test Score"]
)
```

```
import seaborn as sns
```

*#Barplot for all the model accuracies*

```
plt.figure(figsize=(10, 6))
sns.barplot(x='Test Score', y=results.index, data=results)
plt.title('Performance comparision')
plt.show()
```

**#Boxplot**

```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(results.index)
plt.show()
```

## APPENDIX D

## Extra material

## 1. The correlation matrix of discrete data

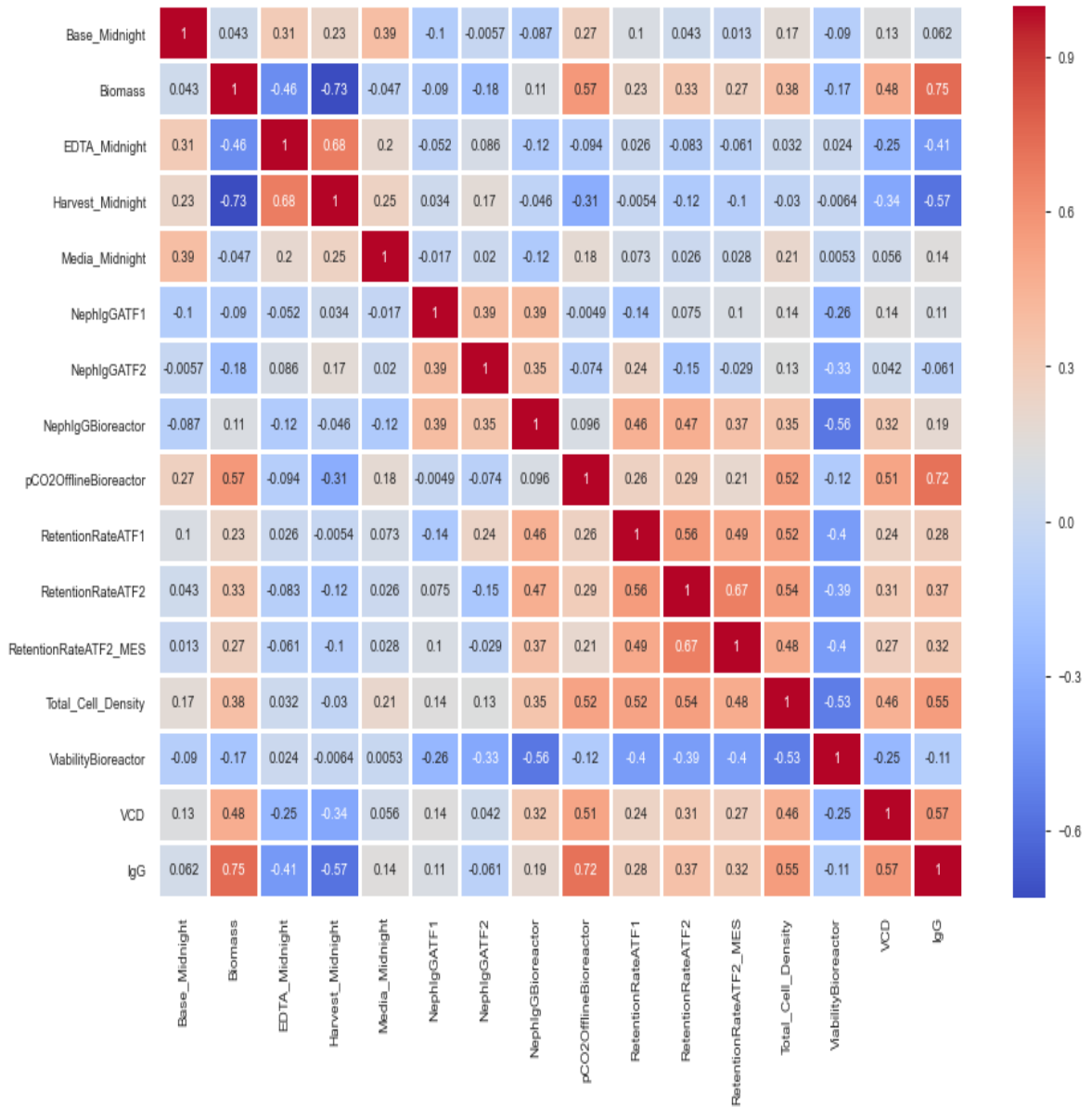


Figure 26 : Correlation matrix of discrete dataset

## 2. The correlation matrix of continuous data

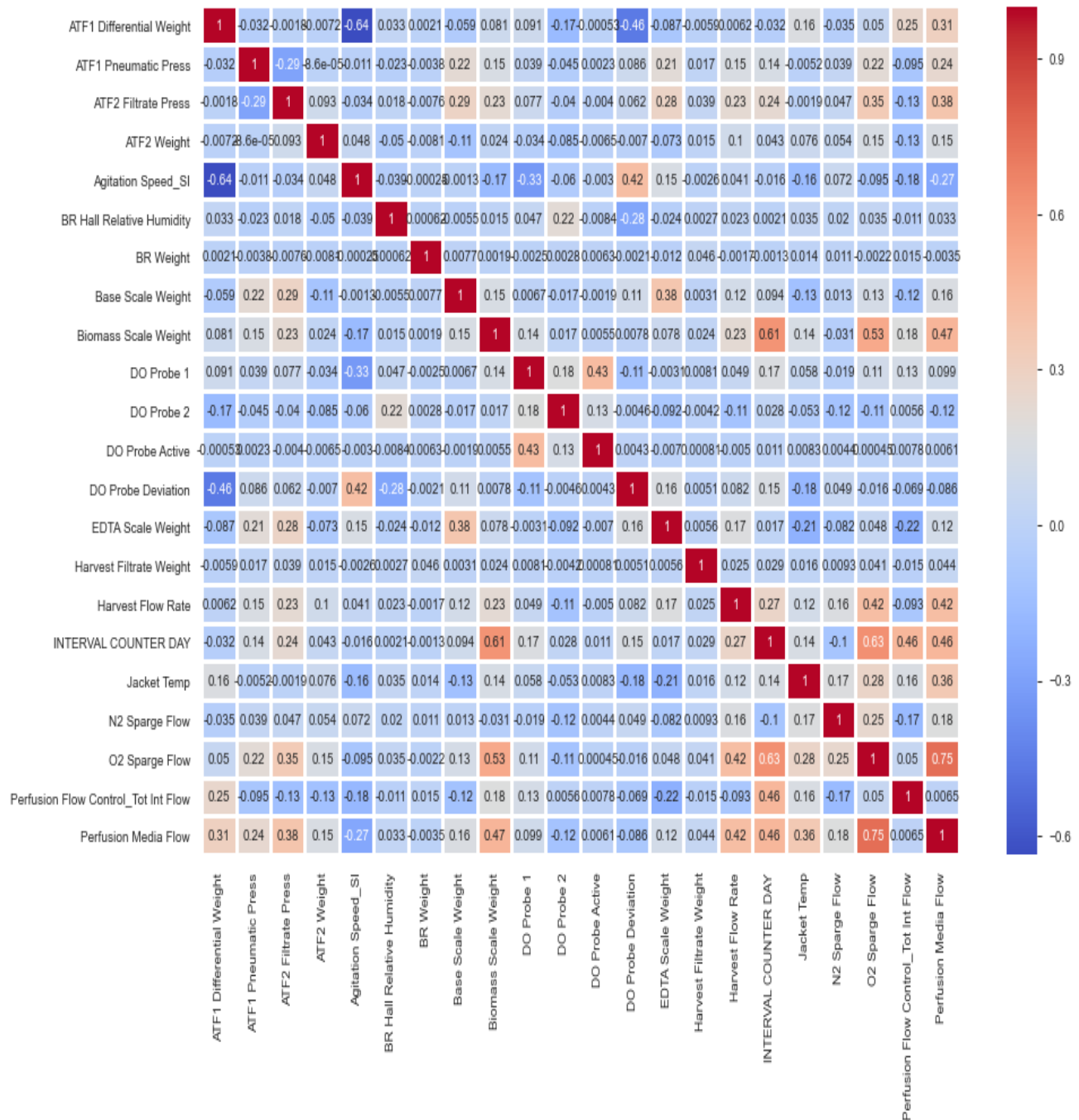


Figure 27 : Correlation matrix of continuous dataset

3. Outliers treatment for Base\_midnight

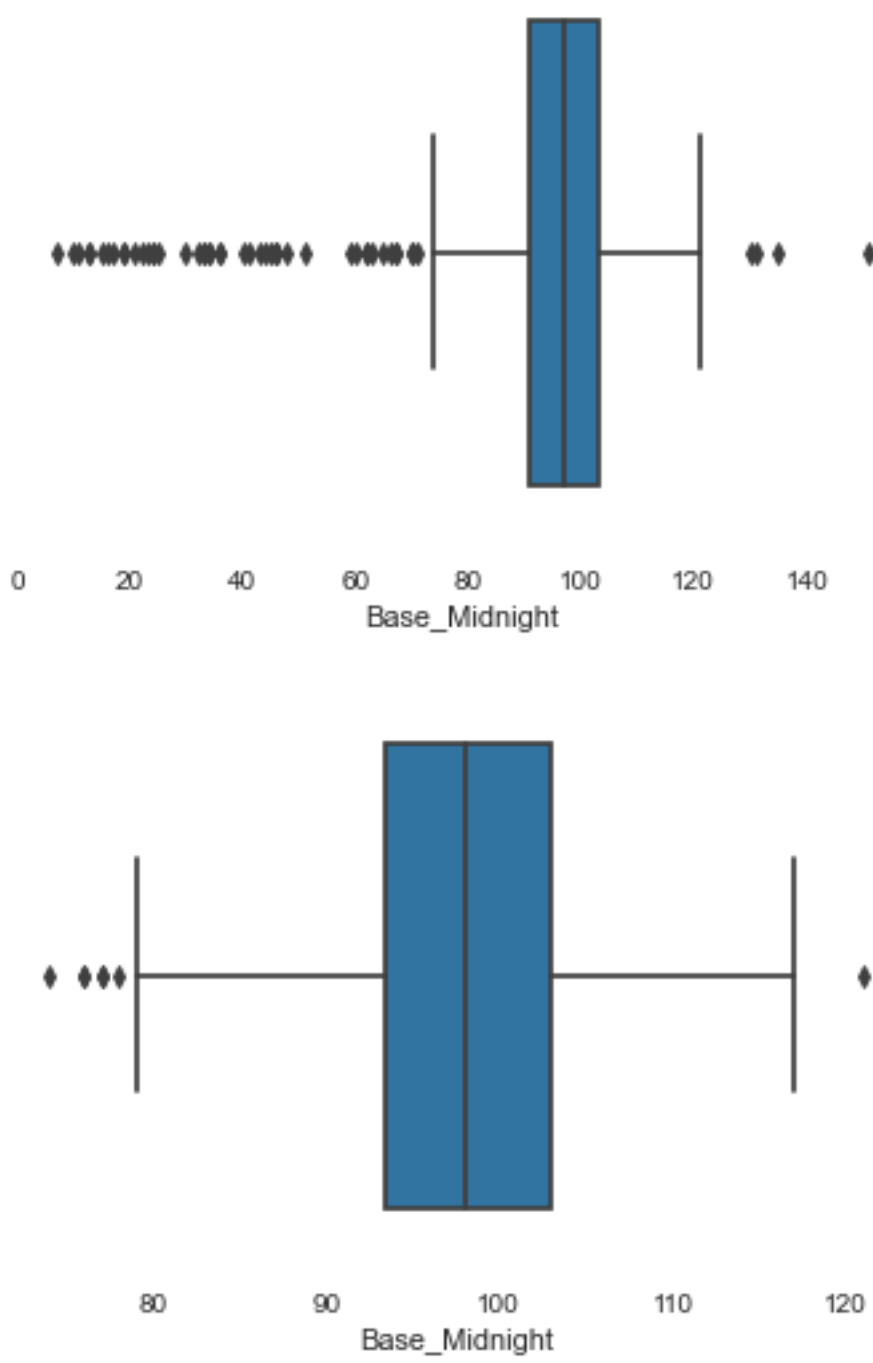


Figure 28 : outliers

4. Relation between Biomass and VCD

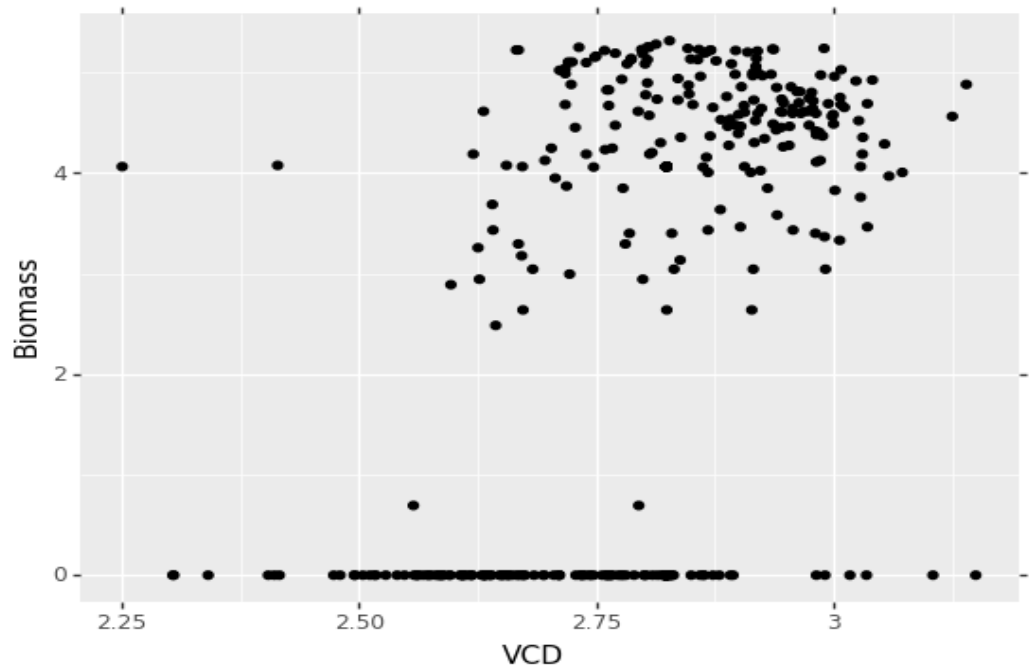


Figure 29 : Relation between Biomass and VCD

5. Relation between VCD and IgG

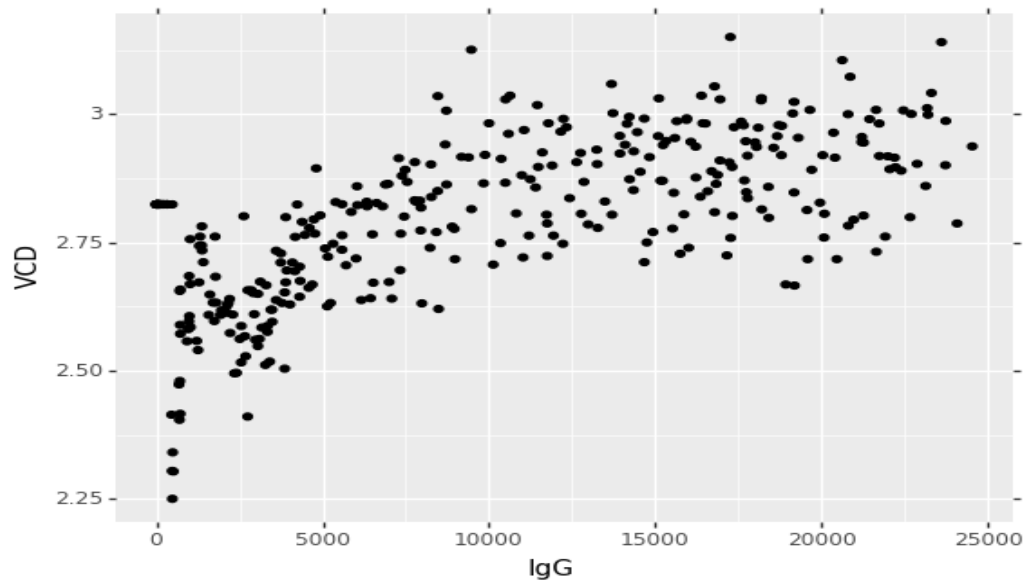


Figure 30 : Relation between VCD and IgG