TOXON STATISTA

A TOXICOLOGICAL STUDY OF CHEMICAL COMPOUND USING STATISTICS

UNIVERSITY OF MUMBAI

DEPARTMENT OF STATISTICS

Vidyanagari, Mumbai-400098

CERTIFICATE

This is to certify that the following students of M.Sc. Part-II have successfully completed the project **TOXON STATISTA** during the academic year 2019-2020. This work to the best of our knowledge and belief is original.

The team comprises of,

1. **Bhuvad Uday Laxman**
2. **Dalvi Prakash Jaganath**
3. **Devruskar Kunal Ramesh**
4. **Kutekar Mangesh Bhambu**
5. **Khismatrao Niyati Rajendra**
6. **Parkar Suyog Dilip**
7. **Patil Kunal Sanjay**
8. **Tiwari Gourav Surendra**
9. **Vibhute Rohit Tulshiram**

Dr. Mrs. Vijayanti U. Dixit
Head of the Department & Project Mentor

ACKNOWLEDGEMENT

In accomplishment of this project successfully, many people have best owned upon me their blessings and the heart pledged support, this time I am utilizing to thank all people who have been concerned with this project.

Primarily we would thank our guides Dr. Rahul Nabar and Dr. V.U. Dixit for their continuous assistance from the conception of the topic to the final presentation.

We are also thankful to Dr. Santosh Gite, Dr. Alok Dabade, Prof. Ashwini Ghanekar, Prof. Sheetal Chabukswar and Prof. Priyesh Tiwari for extending their help and support and solving all our queries.

Last but not the least, we would like to give our special thanks to all the people for providing us with their vital information, opinions, precious time and support.

INDEX

INTRODUCTION

Our daily lives are exposed to a large number of compounds including the environment, foods, skin care products and medicines. In order to protect human health and the human body from potentially harmful substances, these compounds must pass reliable adverse reaction tests, especially toxicity tests. In addition, the actual chemical experiments could test the toxicity of compounds, but they consume considerable manpower, financial and material resources, and many experiments are very time-consuming and may be involved in ethical issues. *In toxicology, the median lethal dose, LD50 (abbreviation for "lethal dose, 50%") is a measure of the lethal dose of a toxin. The value of LD50 for a substance is the dose required to kill half the members of a tested population after a specified test duration.*

LD50 figures are frequently used as a general indicator of a substance's acute toxicity. A lower LD50 is indicative of increased toxicity. LD50 is usually determined by tests on animals such as laboratory mice. The LD50 is usually expressed as the mass of substance administered per unit mass of test subject, typically as milligrams of substance per kilogram of body mass. Lethal dosage often varies depending on the method of administration; for instance, many substances are less toxic when administered orally than when intravenously administered. For this reason, LD50 figures are often qualified with the mode of administration, e.g., "LD50."

Acute toxicity means the ability of a substance to cause adverse effects within a short period following dosing or exposure, which is usually the first step in the toxicological investigations of unknown substances. The median lethal dose, LD50, is frequently used as a general indicator of a substance's acute toxicity, and there is a high demand on developing non-animal-based prediction of LD50. In screening drugs, determination of LD50 values (the dose which has been proved lethal (causing deaths) to 50% of the population) is usually an initial step in the assessment and evaluation of the toxic characteristics of the substances. At the present time, however, there is increasing opposition to the use of living animals in the research and testing activities from animal rights groups. Therefore, computational methods for estimating the toxicity of chemicals are considered useful. With the rapid development of artificial intelligence, computers are used to accomplish these tasks. Numerous drug toxicity prediction models have been developed using a variety of computational methods. However, the accuracy of the computer-aided method is still a problem which needs to be solved.

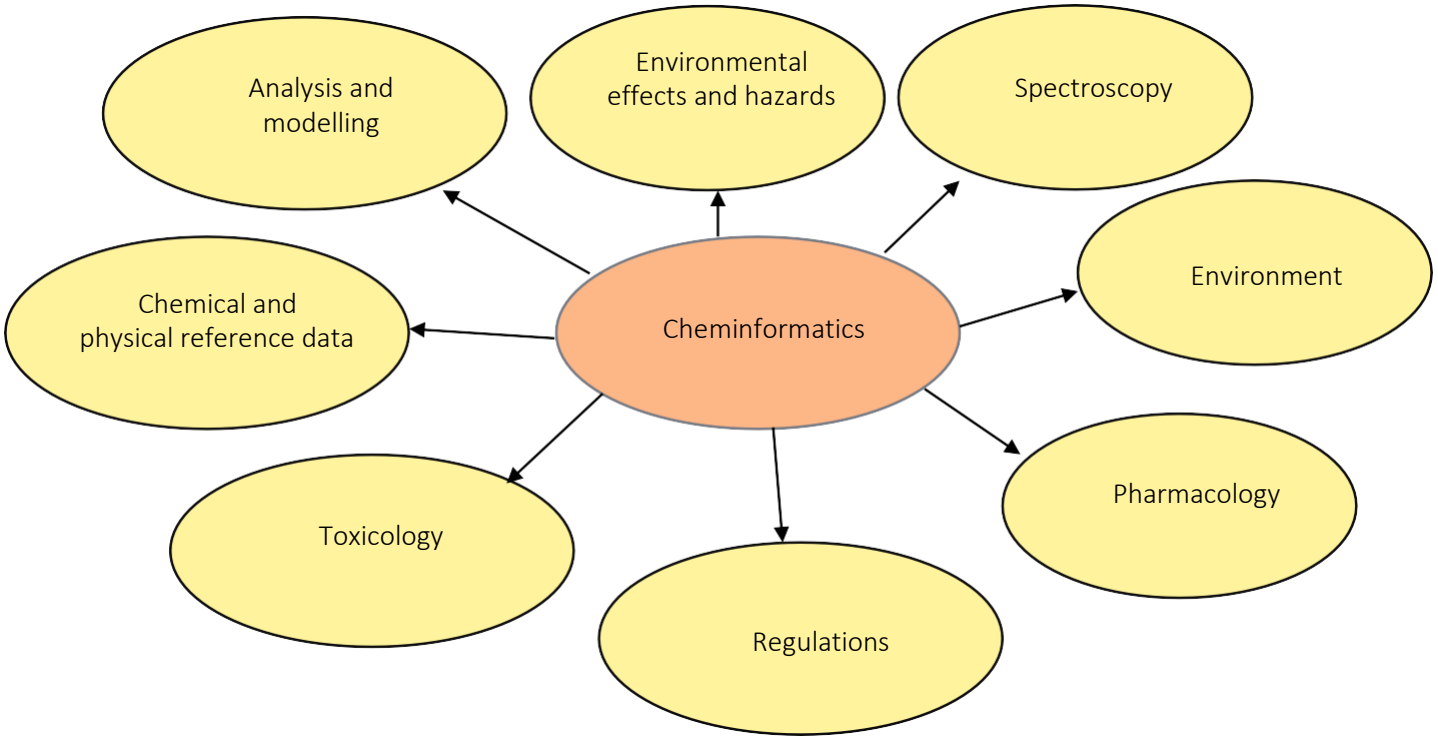
Hence, our project focuses on evaluating the toxicity of the chemical compounds and drugs using statistical methods.

WHAT IS CHEMINFORMATICS?

Saying that computational methods are a very common resource for chemistry researchers would be an understatement. Computers have become so powerful that anyone interested in researching a wide range of chemistry topics will surely resort to them for analysis, calculations, estimations, and predictions.

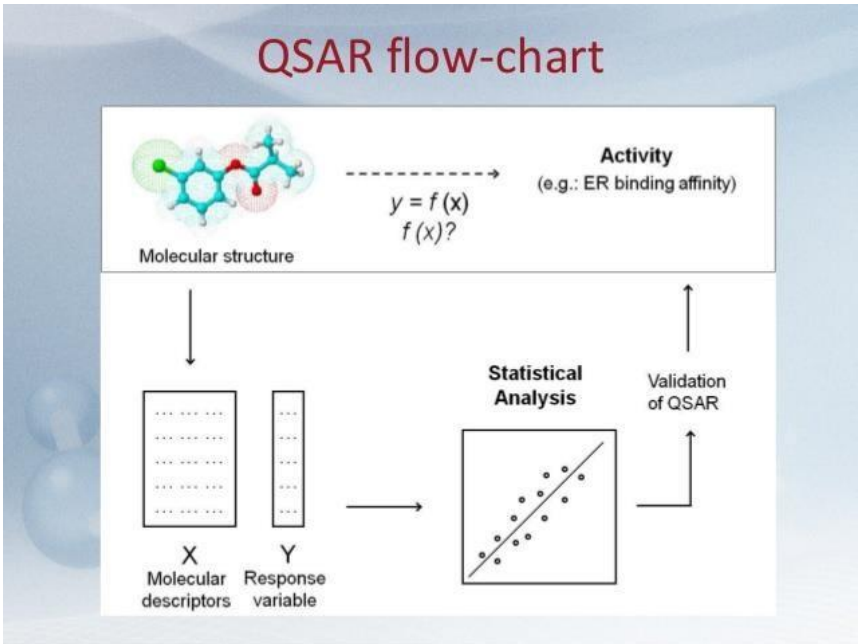
The introduction of new technologies and approaches like machine learning and data science has only deepened this dependence for chemistry researchers. In fact, the field of cheminformatics (that is, solving chemical problems with computers) has only grown since it was defined as such back in 1998. Today, tackling any kind of research within the chemistry field without the intervention of computers seems highly unlikely.

Cheminformatics is a field of information technology that focuses on the collection, storage, analysis, and manipulation of chemical data. The chemical data of interest typically includes information on small molecule formulas, structures, properties, spectra, and activities (biological or industrial). Cheminformatics originally emerged as a vehicle to help the drug discovery and development process; however cheminformatics now plays an increasingly important role in many areas of biology, chemistry, and biochemistry.



QUANTITATIVE STRUCTURE–ACTIVITY RELATIONSHIP

QSAR is a mathematical relationship between a biological activity of a molecular system and its geometric and chemical characteristics. QSAR attempts to identify and qualify the physiochemical properties of compounds and to see whether any of these property has an effect on the compounds biological activity. QSAR study involve statistical methods for studying the correlation between activity and molecular descriptors and also assessing the reliability of the mathematical model. The ultimate goal of QSAR modeling is essentially to use the constructed mathematical model to predict the biological activity of test molecules.



Quantitative structure–activity relationship models (QSAR models) are regression or classification models used in the chemical and biological sciences and engineering. Like other regression models, QSAR regression models relate a set of "predictor" variables (X) to the potency of the response variable (Y), while classification QSAR models relate the predictor variables to a categorical value of the response variable.

In QSAR modeling, the predictors consist of physiochemical properties or theoretical molecular descriptors of chemicals; the QSAR response-variable could be a biological activity of the chemicals. QSAR models first summarize a supposed relationship between chemical structures and biological activity in a data-set of chemicals. Second, QSAR models predict the activities of new chemicals

OBJECTIVES:

1. Use predictive modeling to estimate LD50 value of chemical compounds based on their chemical and physical properties.

2. To find out the important features that are affecting Ld50

3. Dependency of different route of exposure on each other for chemical compounds

DATA PREPARATION

RESPONSE VARIABLE:
LD50_mgkg

PREDICTORS:

The predictors that would be used for our analysis are the descriptors i.e. Physiochemical properties of the chemical compounds. Below table gives the predictors that have been used for the analysis and their description.

PREDICTORS	DESCRIPTION
Molecular Formula	Molecular formula of the chemical compounds
Molecular Weight	The molecular weight is the sum of all atomic weights of the constituent atoms in a compound, measured in g/mol. In the absence of explicit isotope labelling, averaged natural abundance is assumed.
XLogP	Computationally generated octanol-water partition coefficient or distribution coefficient. XLogP is used as a measure of hydrophilicity of the molecule
Exact Mass	The mass of the most likely isotopic composition for a single molecule, corresponding to the most intense ion/molecule peak in a mass spectrum.
Monoisotopic Mass	The mass of a molecule, calculated using the mass of the most abundant isotope of each element.
TPSA	Surface sum over all polar atoms or molecules
Complexity	Complexity rating of a compound is a rough estimate of how complicated the structure is, seen from the point of view of both the elements contained and the displayed structural features including symmetry.
Feature Ring Count 3D	Number of rings of a conformer.
Feature Hydrophobes Count 3D	Number of hydrophobes of a conformer.

Exact Mass	The mass of the most likely isotopic composition for a single molecule, corresponding to the most intense ion/molecule peak in a mass spectrum.
Charge	The total (or net) charge of a molecule.
H-Bond Count	Number of hydrogen-bond donors in the structure.
H-bond Acceptor count	Number of hydrogen-bond acceptors in the structure.
Rotatable bond count	Number of rotatable bonds.
Heavy Atom Count	Number of non-hydrogen atoms.
Isotope Atom Count	Number of atoms with enriched isotope(s)
Atom Stereo Count	Total number of atoms with tetrahedral (sp3) stereo [e.g., (R)- or (S)-configuration]
Defined Atom Stereo Count	Number of atoms with defined tetrahedral (sp3) stereo.
Undefined Atom Stereo Count	Number of atoms with undefined tetrahedral (sp3) stereo.
Bond Stereo Count	Total number of bonds with planar (sp2) stereo [e.g., (E)- or (Z)-configuration
Defined Bond Stereo Count	Number of atoms with defined planar (sp2) stereo.
Undefined Bond Stereo Count	Number of atoms with undefined planar (sp2) stereo.
Covalent Unit Count	Number of covalently bound units.
Conformer Count 3D	The number of conformers in the conformer model for a compound.
Feature Anion Count 3D	Number of anionic centers (at pH 7) of a conformer.
Feature Cation Count 3D	Number of cationic centers (at pH 7) of a conformer.

Volume 3D	Analytic volume of the first diverse conformer (default conformer) for a compound.
X Steric Quadrupole 3D	The x component of the quadrupole moment (Qx) of the first diverse conformer (default conformer) for a compound.
Y Steric Quadrupole 3D	The y component of the quadrupole moment (Qy) of the first diverse conformer (default conformer) for a compound.
Z Steric Quadrupole 3D	The z component of the quadrupole moment (Qz) of the first diverse conformer (default conformer) for a compound.
Feature Count 3D	Total number of 3D features (the sum of FeatureAcceptorCount3D, FeatureDonorCount3D, FeatureAnionCount3D, FeatureCationCount3D, FeatureRingCount3D and FeatureHydrophobeCount3D)
Feature Acceptor Count 3D	Number of hydrogen-bond acceptors of a conformer.
Feature Donor Count 3D	Number of hydrogen-bond donors of a conformer.
Conformer Model RMSD 3D	Conformer sampling RMSD in Å.
Effective Rotor Count 3D	Total number of 3D features (the sum of FeatureAcceptorCount3D, FeatureDonorCount3D, FeatureAnionCount3D, FeatureCationCount3D, FeatureRingCount3D and FeatureHydrophobeCount3D)

DATA EXTRACTION

The data is extracted using Webchem package in R-software.

Webchem*:*

Webchem is useful for extracting the physiochemical properties of the compounds using the CASRN. This package extracts the data from PubChem. It gives values for the properties like molecular weight, XLogP, topological surface area, count of hydrogen bond acceptor and donor, etc. that are mentioned in the above table. So total of 33 numerical predictors will be used for the analysis purpose.

DATA PREPROCESSING

The given data consist of,

1. CASRN: CAS Registry Number is a unique numerical identifier assigned by the Chemical Abstracts Service (CAS) to every chemical substance.
2. DTXSID: DSSTox substance identifier (DTXSID) used in the Environmental Protection Agency CompTox Dashboard
3. Chemical Name: Name of the chemical compounds
4. LD50 in mg/kg: Median lethal dose of the chemical compounds measured in mg/kg
5. Test type: The method of estimating the LD50 values. Three methods are used namely Point estimate, limit test (greater than) and limit test (less than)

The following is the glimpse of the dataset.

```
ï..CASRN        DTXSID           Chemical_Name LD50_mgkg      test_type
100-00-5 DTXSID5020281 1-Chloro-4-nitrobenzene       294 point estimate
100-00-5 DTXSID5020281 1-Chloro-4-nitrobenzene       420 point estimate
100-00-5 DTXSID5020281 1-Chloro-4-nitrobenzene       500 point estimate
100-00-5 DTXSID5020281 1-Chloro-4-nitrobenzene       530 point estimate
100-00-5 DTXSID5020281 1-Chloro-4-nitrobenzene       565 point estimate
100-00-5 DTXSID5020281 1-Chloro-4-nitrobenzene       664 point estimate
```

The structure of the dataset tells us the total number of observations, number of variables, type of the variable and unique number of observations in each column.

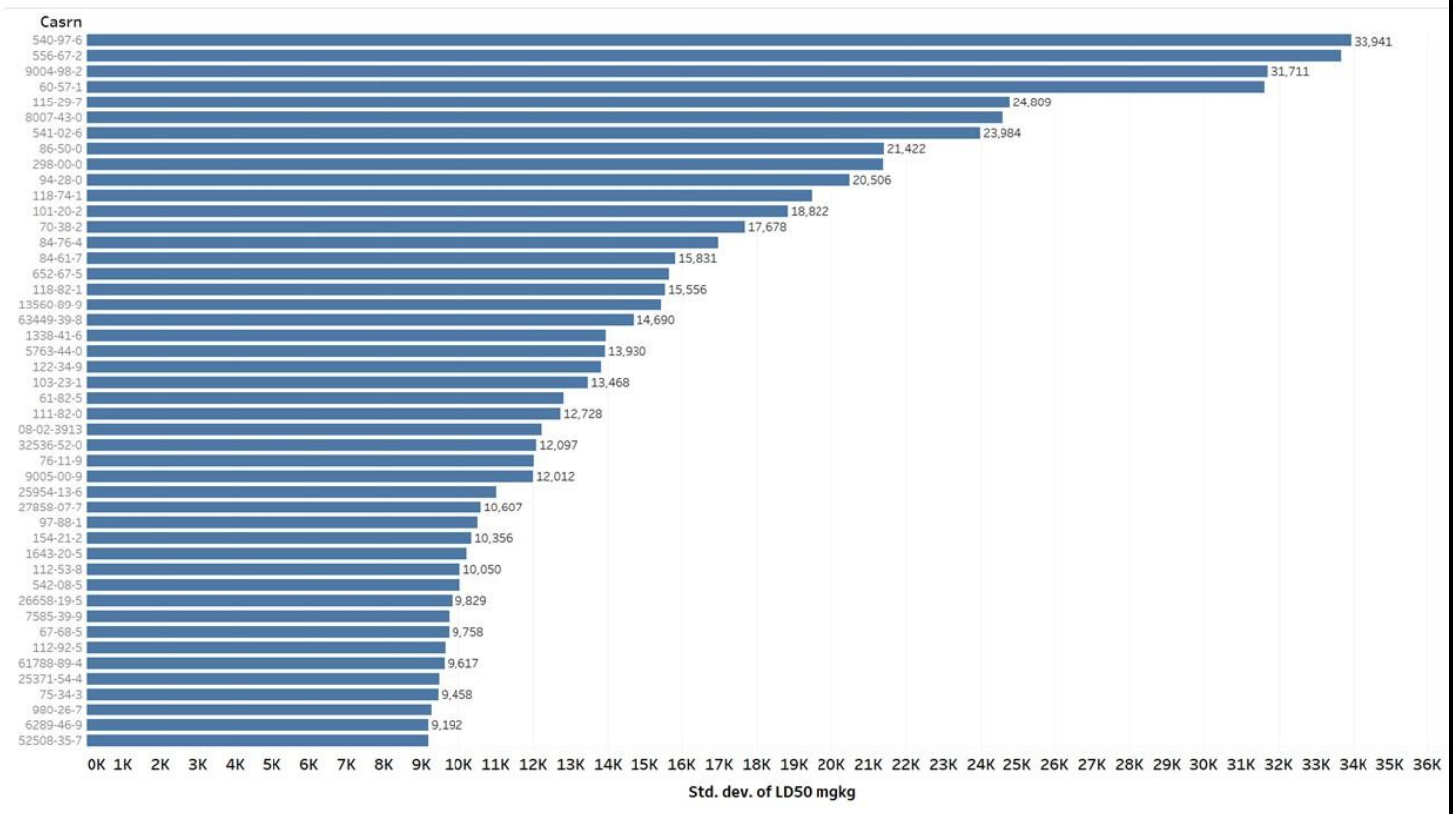
```
'data.frame':   12184 obs. of  5 variables:
 $ ï..CASRN     : Factor w/ 8994 levels "100-00-5","100-01-6",..: 1 1 1 1 1 1 1 2 2 2 ...
 $ DTXSID       : Factor w/ 7076 levels "DTXSID00142939",..: 3624 3624 3624 3624 3624 3624
3624 5756 5756 5756 ...
 $ Chemical_Name: Factor w/ 7076 levels "(-)-alpha,beta-Thujone",..: 177 177 177 177 177 17
7 177 1756 1756 1756 ...
 $ LD50_mgkg    : num  294 420 500 530 565 664 694 750 1410 2000 ...
 $ test_type    : Factor w/ 3 levels "limit test (greater than)",..: 3 3 3 3 3 3 3 3 3 3 ..
```

There are total 5 variables and 12,184 observations in the dataset. As we can see from the head of the dataset the compounds do not have unique LD50 values. So let's first look at the frequency table

Number of LD50 values	Number of compounds
1	7646
2	626
3	334
4	165
5	87
6	41
7	30
8	21
9	16
10	4
11	4
12	4
13	6
14	1
15	2
16	3
17	1
22	1
23	1
57	1

As we can see there are 7646 compounds with single LD50 values whereas the remaining compounds have more than one LD50 values. So we have to include only one LD50 value for single CASRN. To choose the appropriate central tendency we plot a bar plot of standard deviation for each group of the same compounds.

Note: This plot consists of CASRN with high standard deviation values.



Since standard deviation for some groups is too large mean cannot be the appropriate representative. Thus, we aggregate the data by taking the minimum of the values of LD50 which represents the maximum toxicity of the compound.

Also, there are 8994 CASRN and 7076 DTXSIDs. But for each compound CASRN and DTXSID should be unique. This means that there is some discrepancy in the dataset. Hence we first look for the missing values and duplicates.

CASRN	DTXSID	CHEMICAL_NAME	LD50_mgkg	TEST_TYPE	categories
0	1912	1912	0	0	0

Thus there are 1912 missing values in columns DTXSID and Chemical name. Also looking for the duplicates we come to know that there are 6 duplicates DTXSID and Chemical name.

7077	*NA*	1912
7071	DTXSID0027640	2
7072	DTXSID5040176	2
7073	DTXSID6032649	2
7074	DTXSID7023980	2
7075	DTXSID9024194	2
7076	DTXSID9058315	2

Each of these DTXSIDs represents a chemical compound having two different LD50 values. Thus, keeping the one with minimum LD50 value.

Also, CASRN number gives us the CID (Compound IDs). But for 204 compounds the CIDs were not available. Hence removing those compounds.

Since there are no missing values in CASRN, which will be used for extracting the data, and DTXSID and Chemical name will not be used. Hence, we do not treat the missing values in DTXSID and Chemical name as it is.

After removing the duplicates and 204 compounds for which CIDs were not available the data size reduces to 8784. Thus, there are 8784 unique chemical compounds with unique CASRN, DTXSID and Chemical names.

Missing Value Treatment

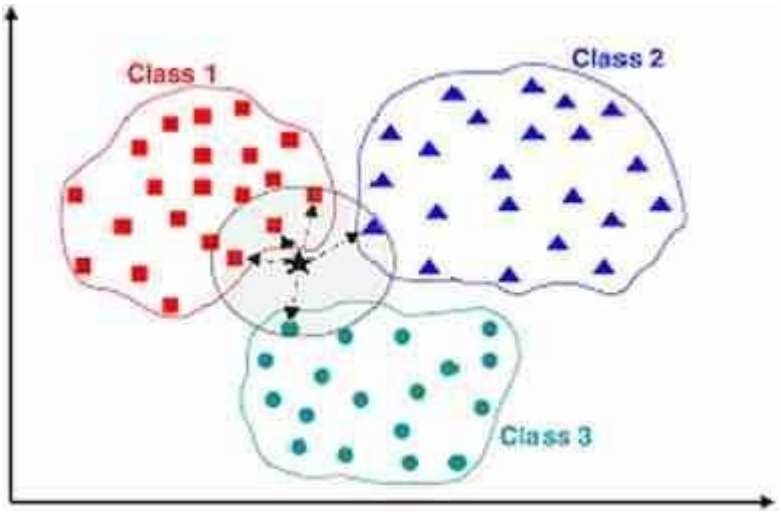
Missing Values in the dataset is one heck of a problem before we could get into Modelling. A lot of machine learning algorithms demand those missing values be imputed before proceeding further.

The data extracted using the package Webchem has missing values in XlogP,

To treat the missing values we use kNN imputation

❖ k-Nearest Neighbor (kNN) based Missing Value Imputation

kNN algorithm is widely used for missing value imputation. In this method, k nearest neighbors are chosen based on some distance measure and their average is used as an imputation estimate.



The method requires the selection of the number of nearest neighbors, and a distance metrics. kNN can predict both discrete(most frequent value among the k nearest neighbors) and continuous attributes (mean among the k nearest neighbors). The metrics varies according to type of data. For continuous data the commonly used distance metrics are Euclidean, Manhattan and cosine. Finally, the number of neighbors (k) has to be carefully selected when using kNN imputation.

To impute the missing values the data is segregated on the basis of categories i.e. extremely toxic, highly toxic, slightly toxic etc. so that the values that are missing for the chemicals that belong to a particular are imputed separately.

DATA VISUALISATION

1. SUMMARY OF EACH CATEGORY

The following is the summary table showing the number of compounds belonging to each group (n), mean, median, minimum and maximum values for the same.

	categories	n	Mean	Median	Min	Max
1	Extremely toxic	50	0.538854	0.55	0.012	1
2	Highly toxic	717	20.452735	17.90	1.020	50
3	Moderately toxic	2109	260.779054	250.00	51.000	500
4	Slightly toxic	5111	2309.162553	2000.00	501.000	5000
5	Practically nontoxic	830	8841.955422	8455.00	5010.000	15000
6	Relatively harmless	177	23713.553672	20000.00	15180.000	70000

2. FREQUENCY DISTRIBUTION OF EACH CATEGORY

We classify each compound based on their toxicity with the help of following scale.

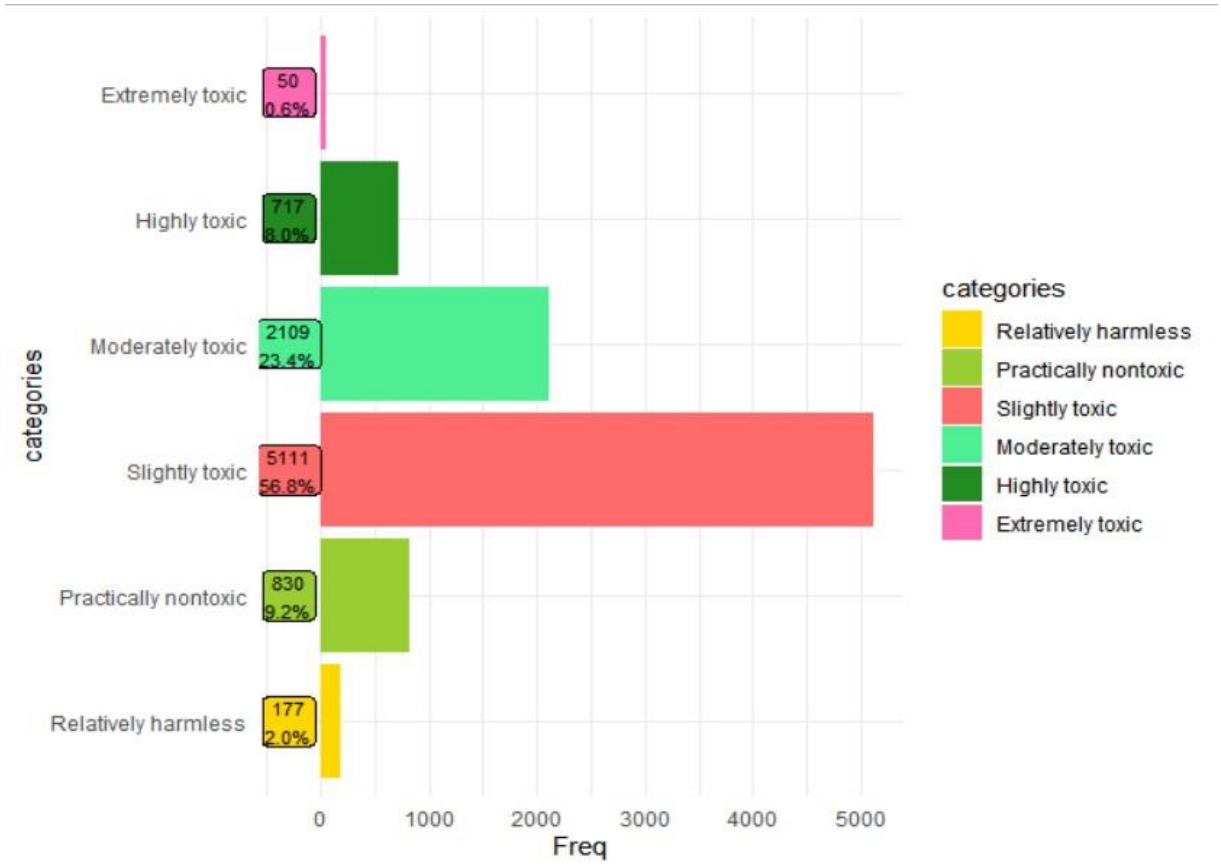
1 – Extremely toxic (1 mg/kg or less)
2 – Highly toxic (1 mg/kg-50 mg/kg)
3 – Moderately toxic (50 mg/kg-500 mg/kg)
4 – Slightly toxic (500 mg/kg -5000 mg/kg)
5 – Practically non-toxic (5000 mg/kg -15000 mg/kg)
6 – Relatively harmless (15000 mg/kg or more).

As per the above scale add a new column in the dataset called Categories which would represent the category of each chemical compound.

The following is the frequency table representing the number of chemical compounds in each group.

Categories	Freq
Extremely toxic	50
Highly toxic	717
Moderately toxic	2108
Practically nontoxic	827
Relatively harmless	177
Slightly toxic	5109

The above table is graphically represented as follows.



From the bar chart we can see that in our data 56.8 % of the compounds belongs to slightly toxic category and only 0.6% of the compounds belongs to extremely toxic category. Thus, our data mostly consist of chemicals compounds belonging to the classes slightly toxic and moderately toxic.

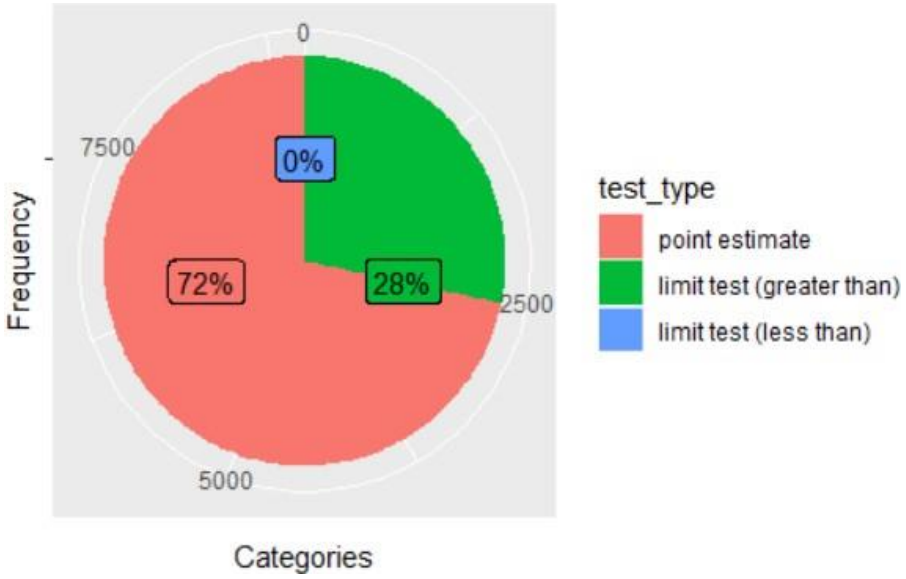
3. FREQUENCY DISTRIBUTION OF TEST_TYPE

The LD50 values for each compound are calculated using three methods namely Point estimate, limit test (greater than) and limit test (less than). We check the number of chemical compounds estimated using each method. Below is the frequency distribution the methods used for estimation of LD50

```
                        Var1 Freq
1 limit test (greater than) 2534
2    limit test (less than)    8
3            point estimate 6452
```

Out of 8994 there are 2534 (28%) chemical compounds whose LD50 values are estimated by limit test (greater than) method, 8 (approximately 0%) chemical compounds whose LD50

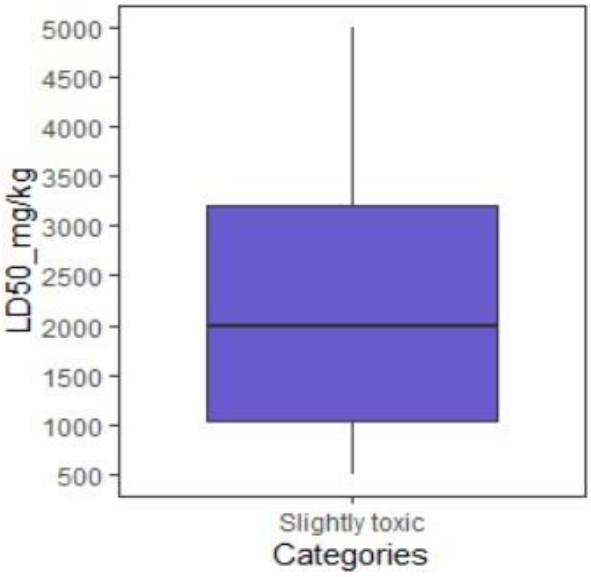
values are calculated with limit test (less than) method and 6452 (72%) chemical compounds whose LD50 values are estimated using point estimate method

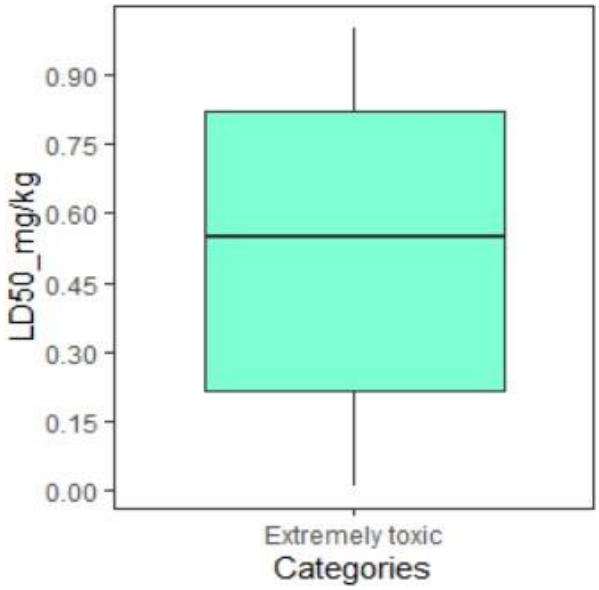


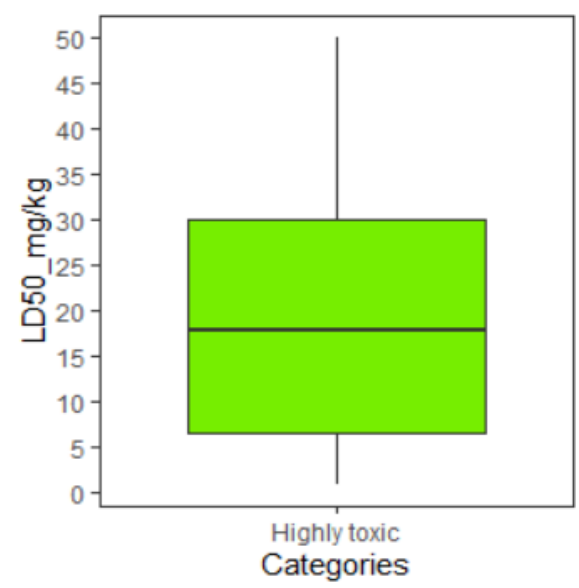
Note: Here the number chemical compounds whose values are estimated using limit test (less than) method is 8 is 0.00089% which is approximated to zero by the software.

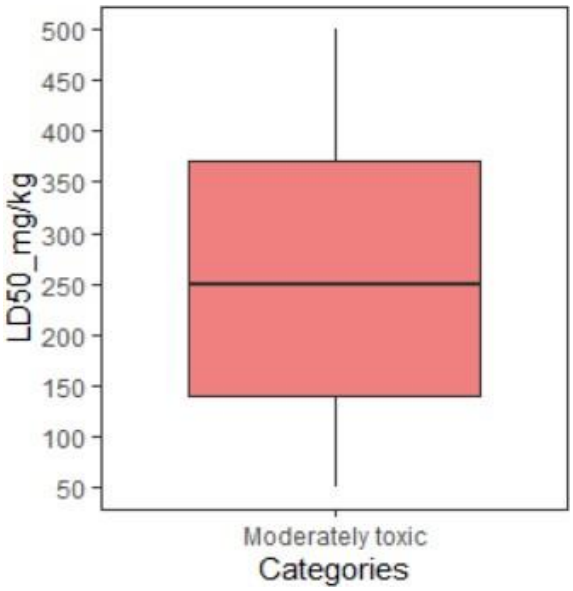
4. DETECTION OF OUTLIERS

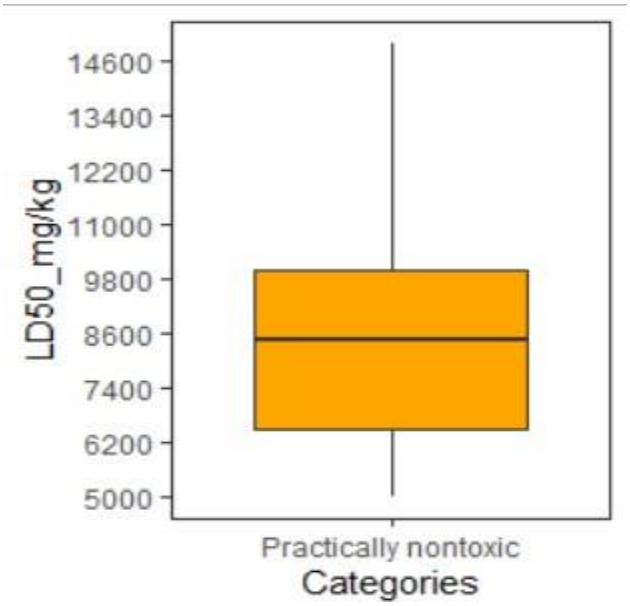
Boxplots are useful in detecting the outliers. We create boxplot for each category to find out which of them has outliers. From the boxplots we can see that the category Relatively harmless has outliers in it whereas the other categories are free from outliers.

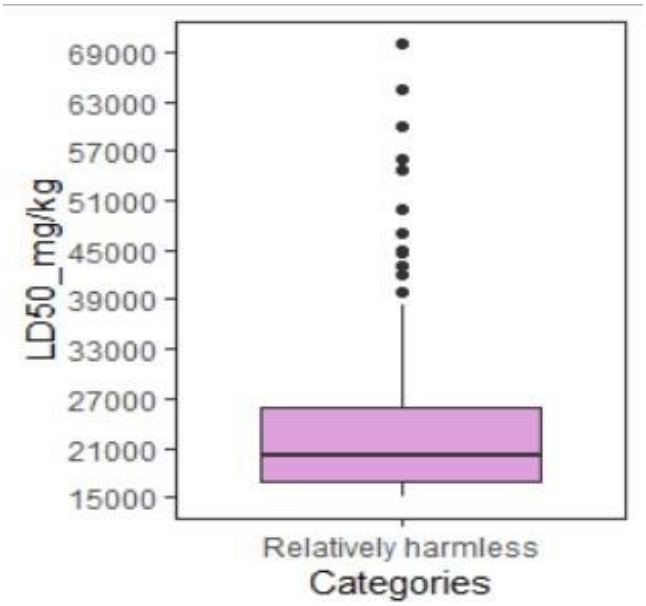






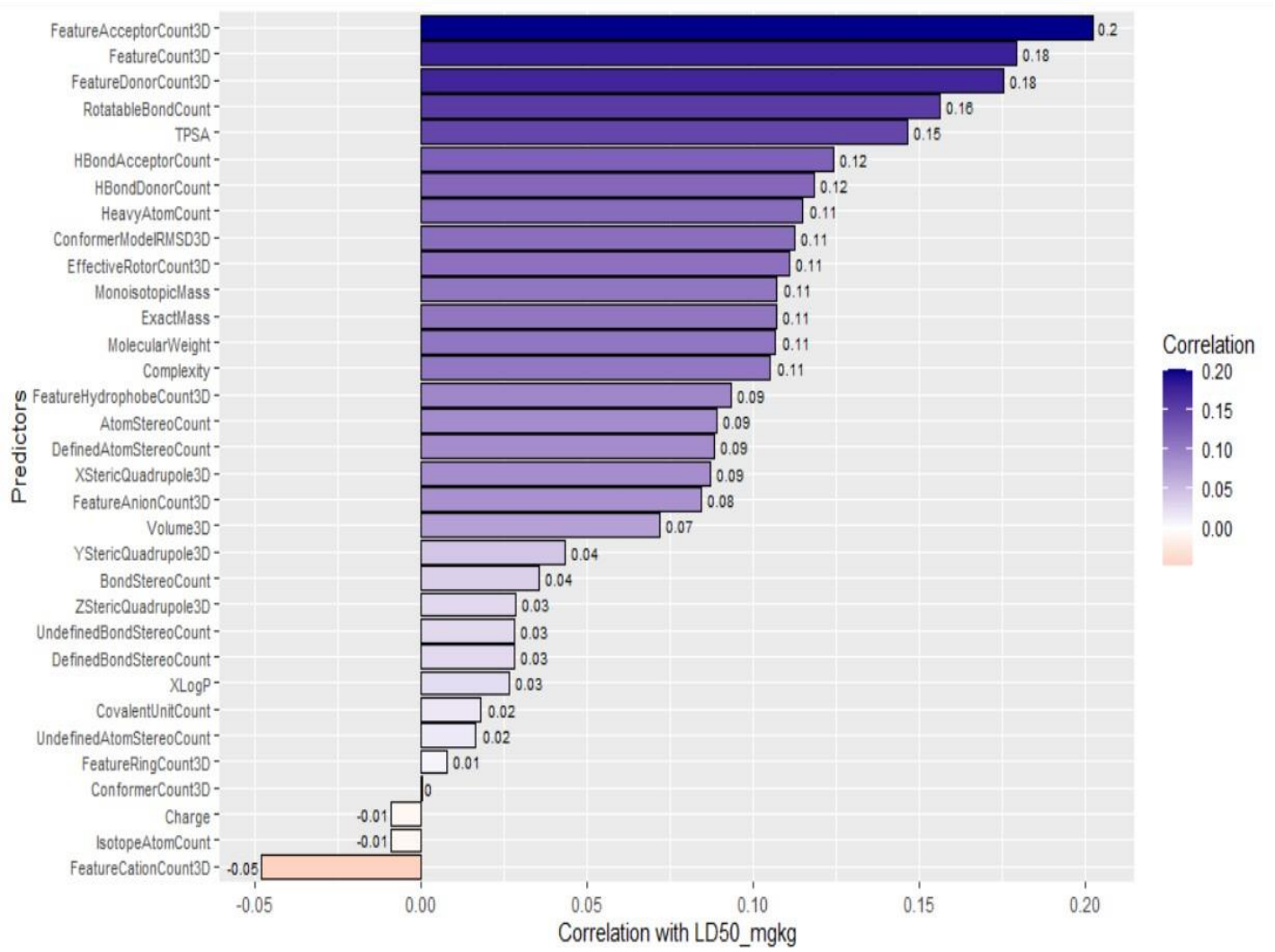






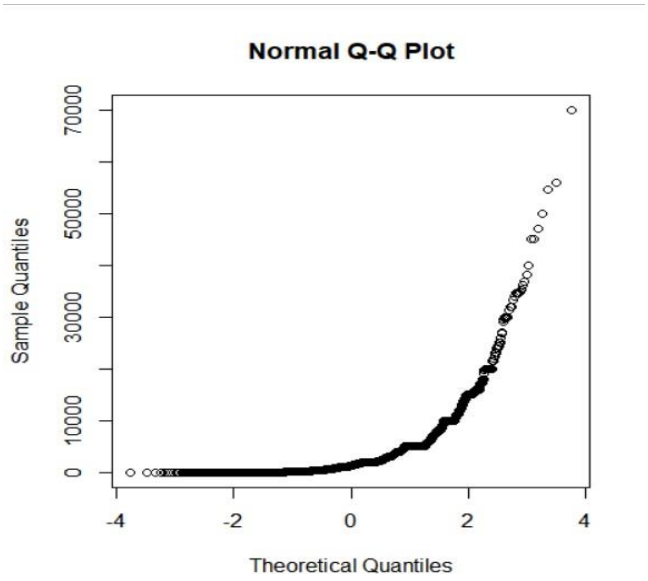
5. CORRELATION BETWEEN RESPONSE VARIABLE (LD50) AND PREDICTORS (MOLECULAR DESCRIPTORS)

We check for the correlation between the our response variable i.e LD50 and the predictors i.e. molecular descriptors of the chemical compounds. From the correlation plot we can see that none of the predictor has high correlation with the response variable, LD50_mgkg.



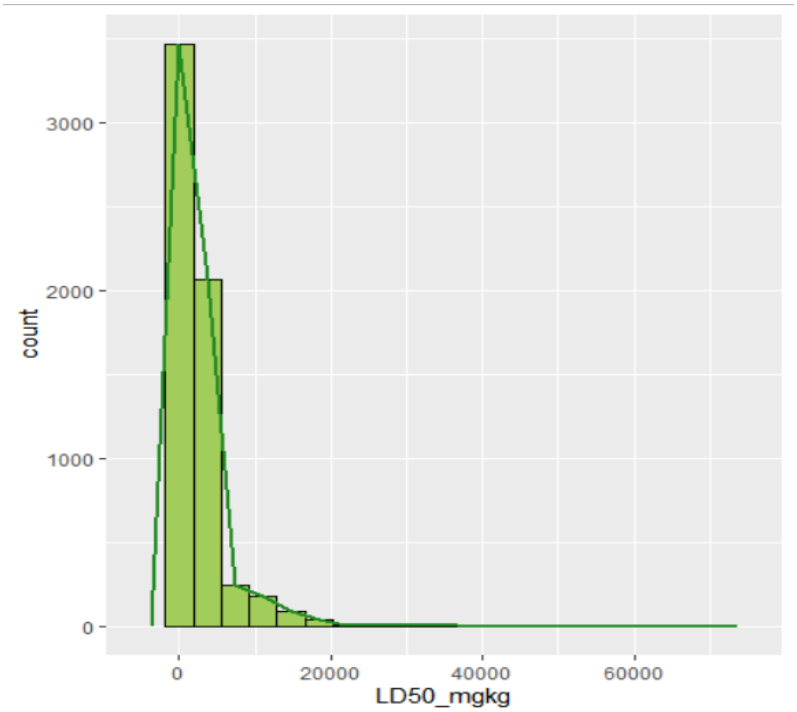
6. DISTRIBUTION OF THE RESPONSE VARIABLE

First we check whether the response variable, LD50_mgkg, follows the normal distribution.We plot QQ plot. As, we can see that the QQ plot is not straight diagonal we say that LD50_mgkg is not normally distributed.

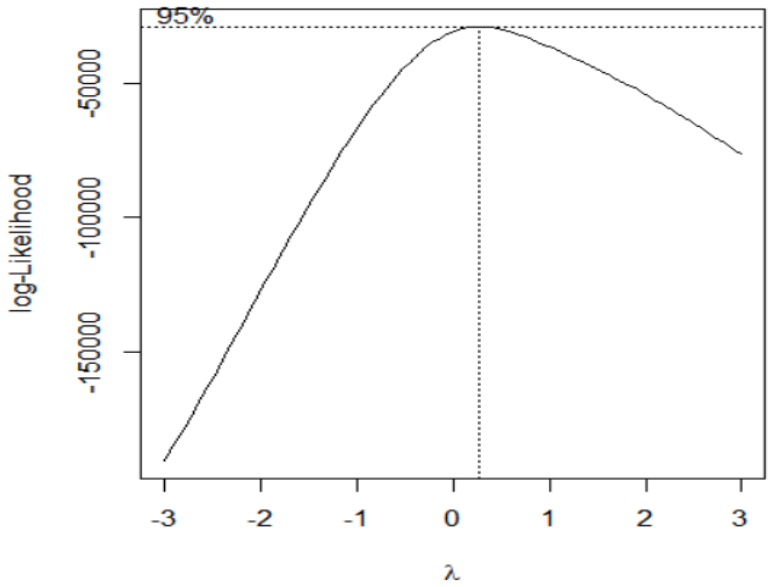


Also we look at the

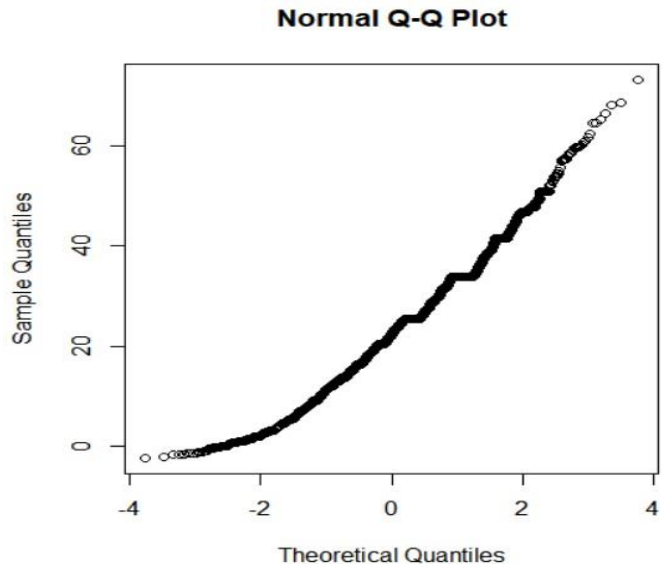
Also we look at the histogram of the LD50_mgkg to check for the skewness.

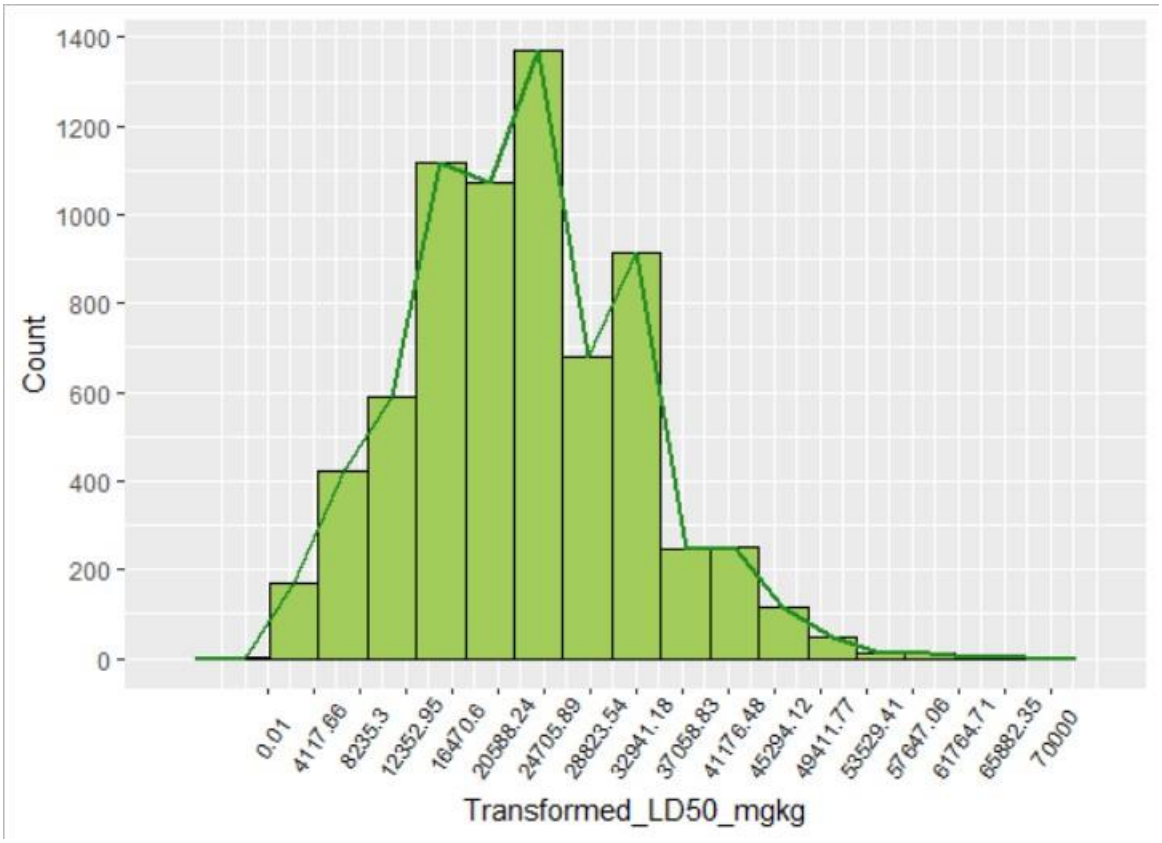


As we can see that the response variable is highly rightly skewed we apply Box Cox transformation. With Box Cox transformation we get lambda, 0.197213.



We transform the response variable and again plot QQplot and histogram of the transformed response variable.





With QQplot and histogram we can see that the LD50_mgkg has become fairly normal. Thus we carry out our further analysis with this transformed variable.

OBJECTIVE 1
TO PREDICTIVE THE LD50 VALUE OF CHEMICAL COMPOUND USING PHYSICAL AND CHEMICAL PROPERTIES.

MULTIPLE LINEAR REGRESSION

Regression models are used to describe relationships between variables by fitting a line to the observed data. Regression allows you to estimate how a dependent variable changes as the independent variable(s) change.
Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable.

The formula for a multiple linear regression is:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \epsilon$$

- **y** = the predicted value of the dependent variable
- **β_0** = the y-intercept (value of y when all other parameters are set to 0)
- **$\beta_1 X_1$**= the regression coefficient (B_1) of the first independent variable (**X_1**)
- **e** = model error (a.k.a. how much variation there is in our estimate of **y**)

To find the best-fit line for each independent variable, multiple linear regression calculates three things:

- The regression coefficients that lead to the smallest overall model error.
- The t-statistic of the overall model.
- The associated p-value (how likely it is that the t-statistic would have occurred by chance if the null hypothesis of no relationship between the independent and dependent variables was true).

It then calculates the t-statistic and p-value for each regression coefficient in the model.

ESTIMATING REGRESSION MODELS USING LEAST SQUARES
Consider a multiple linear regression model with k predictor variables:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \epsilon$$

Let each of the k predictor variables, x_1, x_2... x_k , have n levels. Then x_{ij} represents the i th level of the j th predictor variable x_j. For example, x_{51} represents the fifth level of the first predictor variable x_1, while x_{19} represents the first level of the ninth predictor variable, x_9. Observations, y_1, y_2... y_n, recorded for each of these n levels can be expressed in the following way:

$$
\begin{aligned}
y_1 &= \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + ... + \beta_k x_{1k} + \epsilon_1 \\
y_2 &= \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + ... + \beta_k x_{2k} + \epsilon_2 \\
&.. \\
y_i &= \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_k x_{ik} + \epsilon_i \\
&.. \\
y_n &= \beta_0 + \beta_1 x_{n1} + \beta_2 x_{n2} + ... + \beta_k x_{nk} + \epsilon_n
\end{aligned}
$$

The system of n equations shown previously can be represented in matrix notation as follows:

$$y = X\beta + \epsilon$$

Where,

$$
y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & . & . & . & x_{1n} \\ 1 & x_{21} & x_{22} & . & . & . & x_{2n} \\ . & . & . & & & & . \\ . & . & . & & & & . \\ . & . & . & & & & . \\ 1 & x_{n1} & x_{n2} & . & . & . & x_{nn} \end{bmatrix}
$$

$$
\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ . \\ . \\ . \\ \beta_n \end{bmatrix} \quad \text{and} \quad \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ . \\ . \\ . \\ \epsilon_n \end{bmatrix}
$$

The matrix X is referred to as the *design matrix*. It contains information about the levels of the

predictor variables at which the observations are obtained. The vector β contains all the regression coefficients. To obtain the regression model, β should be known. β is estimated using least square estimates. The following equation is used:

$$\hat{\beta} = (X'X)^{-1}X'y$$

where $'$ represents the transpose of the matrix while $^{-1}$ represents the matrix inverse. Knowing the estimates, $\hat{\beta}$, the multiple linear regression model can now be estimated as:

$$\hat{y} = X\hat{\beta}$$

The estimated regression model is also referred to as the *fitted model*. The observations, y_i, may be different from the fitted values $\hat{y}_i$ obtained from this model. The difference between these two values is the residual, e_i. The vector of residuals, e, is obtained as:

$$e = y - \hat{y}$$

The fitted model can also be written as follows, using $\hat{\beta} = (X'X)^{-1}X'y$:

$$\begin{aligned} \hat{y} &= X\hat{\beta} \\ &= X(X'X)^{-1}X'y \\ &= Hy \end{aligned}$$

where $H = X(X'X)^{-1}X'$. The matrix, H, is referred to as the hat matrix. It transforms the vector of the observed response values, y, to the vector of fitted values, $\hat{y}$.

ASSUMPTIONS OF MULTIPLE LINEAR REGRESSION

Multiple linear regression makes all of the same assumptions as simple linear regression:

1. **No multicollinearity:**

Multicollinearity occurs when independent variables in a regression model are correlated. This correlation is a problem because independent variables should be independent. If the degree of correlation between variables is high enough, it can cause problems when you fit the model and interpret the results.

The idea is that you can change the value of one independent variable and not the others. However, when independent variables are correlated, it indicates that changes in one variable are associated with shifts in another variable. The stronger the correlation, the more difficult it is to change one variable without changing another. It becomes difficult for the model to estimate the relationship between each independent variable and the dependent variable independently because the independent variables tend to change in unison. Hence we assume that the predictors should be independent of each other.

2. Normality of residuals:

The residuals are assumed to be normally distributed with mean zero and constant variance

3. Homogeneity of variance (No heteroscedasticity):

Specifically, heteroscedasticity is a systematic change in the spread of the residuals over the range of measured values. Heteroscedasticity is a problem because ordinary least squares (OLS) regression assumes that all residuals are drawn from a population that has constant variance (homoscedasticity).

4. Linearity:

This assumption addresses the functional form of the model. In statistics, a regression model is linear when all terms in the model are either the constant or a parameter multiplied by an independent variable. You build the model equation only by adding the terms together. These rules constrain the model to one type: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k + \epsilon$

5. No autocorrelation:

No autocorrelation refers to a situation in which no identifiable relationship exists between the values of the error term i.e. the residuals are assumed to be independent of each other.

6. No outliers:

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population. Outliers can have a dramatic impact on linear regression. It can change the model equation completely i.e. bad prediction or estimation. Hence there should be no outliers in the data.

Our first technique to predict the LD50 using the physiochemical properties is Multiple linear regression. To get the appropriate model consisting of significant variables we use Stepwise regression.

Output of stepwise regression,

Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	16	45143	2821.41609	73.39	<.0001
Error	7450	286416	38.44516		
Corrected Total	7466	331559			

It took 20 steps of stepwise regression for selecting 16 significant variables. Following table shows the entrance and removal of the variable from the model

All variables left in the model are significant at the 0.1000 level.

No other variable met the 0.0500 significance level for entry into the model.

TABLE 1

Summary of Stepwise Selection									
Step	Variable Entered	Variable Removed	Label	Number Vars In	Partial R-Square	Model R-Square	C(p)	F Value	Pr > F
1	FeatureAcceptorCount3D		FeatureAcceptorCount3D	1	0.0464	0.0464	765.967	363.48	<.0001
2	FeatureHydrophobeCount3D		FeatureHydrophobeCount3D	2	0.0138	0.0603	648.526	109.93	<.0001
3	FeatureDonorCount3D		FeatureDonorCount3D	3	0.0083	0.0686	578.541	66.84	<.0001
4	HBondAcceptorCount		HBondAcceptorCount	4	0.0084	0.0770	508.355	67.62	<.0001
5	RotatableBondCount		RotatableBondCount	5	0.0112	0.0881	414.008	91.35	<.0001
6	ZStericQuadrupole3D		ZStericQuadrupole3D	6	0.0069	0.0950	356.641	56.71	<.0001
7	XLogP		XLogP	7	0.0049	0.0999	316.187	40.77	<.0001
8	Volume3D		Volume3D	8	0.0110	0.1110	222.934	92.60	<.0001
9	FeatureRingCount3D		FeatureRingCount3D	9	0.0029	0.1139	199.867	24.44	<.0001
10	ConformerModelRMSD3D		ConformerModelRMSD3D	10	0.0033	0.1172	173.105	28.15	<.0001
11		ZStericQuadrupole3D	ZStericQuadrupole3D	9	0.0001	0.1171	172.378	1.25	0.2644

Summary of Stepwise Selection									
Step	Variable Entered	Variable Removed	Label	Number Vars In	Partial R-Square	Model R-Square	C(p)	F Value	Pr > F
12	CovalentUnitCount		CovalentUnitCount	10	0.0017	0.1188	159.862	14.23	0.0002
13	ConformerCount3D		ConformerCount3D	11	0.0016	0.1203	148.381	13.24	0.0003
14	MolecularWeight		MolecularWeight	12	0.0010	0.1213	142.096	8.14	0.0043
15	ExactMass		ExactMass	13	0.0098	0.1310	59.8044	83.78	<.0001
16	HeavyAtomCount		HeavyAtomCount	14	0.0032	0.1342	34.3481	27.39	<.0001
17	DefinedBondStereoCount		DefinedBondStereoCount	15	0.0009	0.1352	28.3180	8.02	0.0046
18	XStericQuadrupole3D		XStericQuadrupole3D	16	0.0006	0.1358	24.7313	5.58	0.0182
19	EffectiveRotorCount3D		EffectiveRotorCount3D	17	0.0005	0.1363	22.3313	4.40	0.0360
20		FeatureRingCount3D	FeatureRingCount3D	16	0.0002	0.1362	21.7010	1.37	0.2420

Output of the regression model fitted using stepwise regression

TABLE 2

Variable	Parameter Estimate	Standard Error	Type II SS	F Value	Pr > F
Intercept	15.65531	0.07175	1830079	47602.3	<.0001
MolecularWeight	-286.22978	34.38502	2663.99101	69.29	<.0001
XLogP	1.46328	0.12590	5193.36770	135.09	<.0001
ExactMass	284.37720	34.47817	2615.42798	68.03	<.0001
HBondAcceptorCount	-0.76041	0.14444	1065.55499	27.72	<.0001
RotatableBondCount	0.69578	0.14139	931.05028	24.22	<.0001
HeavyAtomCount	2.15312	0.33122	1624.57693	42.26	<.0001
DefinedBondStereoCount	-0.27440	0.07481	517.23808	13.45	0.0002
CovalentUnitCount	0.57313	0.09263	1471.68995	38.28	<.0001
Volume3D	-2.32535	0.24962	3336.17890	86.78	<.0001
XStericQuadrupole3D	0.34201	0.11327	350.51476	9.12	0.0025

Variable	Parameter Estimate	Standard Error	Type II SS	F Value	Pr > F
FeatureAcceptorCount3D	2.24084	0.10757	16682	433.91	<.0001
FeatureDonorCount3D	1.03016	0.08540	5594.13797	145.51	<.0001
FeatureHydrophobeCount3D	0.59235	0.08841	1725.88061	44.89	<.0001
ConformerModelRMSD3D	0.63024	0.23321	280.78580	7.30	0.0069
EffectiveRotorCount3D	-0.67856	0.23134	330.76109	8.60	0.0034
ConformerCount3D	0.28246	0.09433	344.69464	8.97	0.0028

GLOBAL TESTING

This section discusses hypothesis tests on the regression coefficients in multiple linear regression. As in the case of simple linear regression, these tests can only be carried out if it can be assumed that the random error terms, ϵ_i, are normally and independently distributed with a mean of zero and variance of σ^2. Three types of hypothesis tests can be carried out for multiple linear regression models:

1. Test for significance of regression: This test checks the significance of the whole regression model.
2. *t* test: This test checks the significance of individual regression coefficients.
3. *F* test: This test can be used to simultaneously check the significance of a number of regression coefficients. It can also be used to test individual coefficients.

TEST FOR SIGNIFICANCE OF REGRESSION

The test for significance of regression in the case of multiple linear regression analysis is carried out using the analysis of variance. The test is used to check if a linear statistical relationship exists between the response variable and at least one of the predictor variables. The statements for the hypotheses are:

$$H_0: \quad \beta_1 = \beta_2 = \ldots = \beta_k = 0$$
$$H_1: \quad \beta_j \neq 0 \text{ for at least one } j$$

The test for Ho is carried out using the following statistic:

$$F_0 = \frac{MS_R}{MS_E}$$

where MS_R is the regression mean square and MS_E is the error mean square. If the null hypothesis, H_0, is true then the statistic F_0 follows the F distribution with k degrees of freedom in the numerator and $n-$ ($k+1$) degrees of freedom in the denominator. The null hypothesis, H_0, is rejected if the calculated statistic, F_0, is such that:

$$F_0 > f_{\alpha,k,n-(k+1)}$$

Analysis of Variance					
Source	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
Model	16	45143	2821.41609	73.39	<.0001
Error	7450	286416	38.44516		
Corrected Total	7466	331559			

Here, P- value=0.0001 < alpha=0.05. Hence, we reject H_0 and conclude that the at least one of the regressor is significant.

TEST ON INDIVIDUAL REGRESSION COEFFICIENTS (*T* TEST)

The t test is used to check the significance of individual regression coefficients in the multiple linear regression model. Adding a significant variable to a regression model makes the model more effective, while adding an unimportant variable may make the model worse. The hypothesis statements to test the significance of a particular regression coefficient, β_j , are:

$H_0: \quad \beta_j = 0$
$H_1: \quad \beta_j \neq 0$

The test statistic for this test is based on the t distribution is

$$T_0 = \frac{\hat{\beta}_j}{se(\hat{\beta}_j)}$$

where the standard error, $se(\hat{\beta}_j)$, is obtained. T_0 follows Normal(0, 1). We would fail to reject the null hypothesis if the test statistic lies in the acceptance region

$$-t_{\alpha/2,n-2} < T_0 < t_{\alpha/2,n-2}$$

Now from table 2 we can see that p-values for all predictors are less than the significance level 0.05 thus we reject Ho and conclude that the predictors are significant.

VALIDATION OF ASSUMPTIONS:

1. **Checking for Multicollinearity**
   VIF > 10 indicates presence of multicollinearity. We have SAS output showing the VIF for each variable. Predictors having VIF > 10 are removed from the model one by one and then recreating the model.

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|	Variance Inflation
Intercept	Intercept	1	15.65531	0.07175	218.18	<.0001	0
MolecularWeight	MolecularWeight	1	-286.22978	34.38502	-8.32	<.0001	229607
XLogP	XLogP	1	1.46328	0.12590	11.62	<.0001	3.07819
ExactMass	ExactMass	1	284.37720	34.47817	8.25	<.0001	230853
HBondAcceptorCount	HBondAcceptorCount	1	-0.76041	0.14444	-5.26	<.0001	4.05144
RotatableBondCount	RotatableBondCount	1	0.69578	0.14139	4.92	<.0001	3.88203
HeavyAtomCount	HeavyAtomCount	1	2.15312	0.33122	6.50	<.0001	21.30522
DefinedBondStereoCount	DefinedBondStereoCount	1	-0.27440	0.07481	-3.67	0.0002	1.08686
CovalentUnitCount	CovalentUnitCount	1	0.57313	0.09263	6.19	<.0001	1.66638
Volume3D	Volume3D	1	-2.32535	0.24962	-9.32	<.0001	12.10083
XStericQuadrupole3D	XStericQuadrupole3D	1	0.34201	0.11327	3.02	0.0025	2.49145

Parameter Estimates							
Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|	Variance Inflation
FeatureAcceptorCount3D	FeatureAcceptorCount3D	1	2.24084	0.10757	20.83	<.0001	2.24731
FeatureDonorCount3D	FeatureDonorCount3D	1	1.03016	0.08540	12.06	<.0001	1.41633
FeatureHydrophobeCount3D	FeatureHydrophobeCount3D	1	0.59235	0.08841	6.70	<.0001	1.51785
ConformerModelRMSD3D	ConformerModelRMSD3D	1	0.63024	0.23321	2.70	0.0069	10.56148
EffectiveRotorCount3D	EffectiveRotorCount3D	1	-0.67856	0.23134	-2.93	0.0034	10.39325
ConformerCount3D	ConformerCount3D	1	0.28246	0.09433	2.99	0.0028	1.72807

Now in the above table the last column is of VIF. We can see that VIF for ExactMass is maximum and exceeding 10. Thus we remove the ExactMass variable from the model and recheck for the VIF. Similarly, we check for VIF again and remove the predictors with maximum VIF exceeding 10. Thus 3 variables are removed from the model namely,

- ExactMass
- HeavyAtomCount
- Volume3D

The final output showing the VIF for each variable is,

Parameter Estimates							
Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|	Variance Inflation
Intercept	Intercept	1	15.65531	0.07269	215.37	<.0001	0
MolecularWeight	MolecularWeight	1	-1.82111	0.16299	-11.17	<.0001	5.02728
XLogP	XLogP	1	0.91248	0.11291	8.08	<.0001	2.41255
HBondAcceptorCount	HBondAcceptorCount	1	-0.27045	0.14196	-1.91	0.0568	3.81341
RotatableBondCount	RotatableBondCount	1	1.67159	0.11710	14.28	<.0001	2.59462

Parameter Estimates							
Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|	Variance Inflation
DefinedBondStereoCount	DefinedBondStereoCount	1	-0.22003	0.07559	-2.91	0.0036	1.08114
CovalentUnitCount	CovalentUnitCount	1	0.52728	0.09294	5.67	<.0001	1.63441
XStericQuadrupole3D	XStericQuadrupole3D	1	0.42885	0.11301	3.79	0.0001	2.41676
FeatureAcceptorCount3D	FeatureAcceptorCount3D	1	2.13183	0.10477	20.35	<.0001	2.07701
FeatureDonorCount3D	FeatureDonorCount3D	1	0.96361	0.08543	11.28	<.0001	1.38094
FeatureHydrophobeCount3D	FeatureHydrophobeCount3D	1	0.73450	0.08489	8.65	<.0001	1.36380
ConformerModelRMSD3D	ConformerModelRMSD3D	1	0.24005	0.22255	1.08	0.2808	9.37257
EffectiveRotorCount3D	EffectiveRotorCount3D	1	-1.45299	0.22498	-6.46	<.0001	9.57769
ConformerCount3D	ConformerCount3D	1	0.26703	0.09539	2.80	0.0051	1.72174

Thus, we can see that VIF for all variables is < 10. We can conclude that there is no multicollinearity present.

2. Autocorrelation

To detect the presence of autocorrelation we use Durbin Watson test statistic

SAS output,

Dependent Variable: Trans_LD50_mgkg

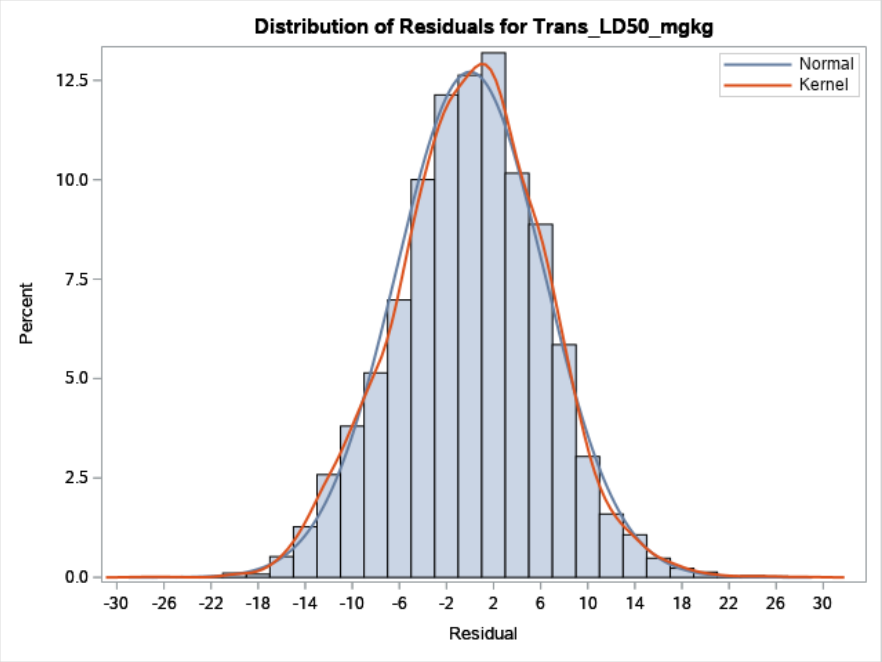
Durbin-Watson D	1.708
Number of Observations	7467
1st Order Autocorrelation	0.146

We can see that value of Durbin Watson statistic < 2, we conclude that there is positive autocorrelation.

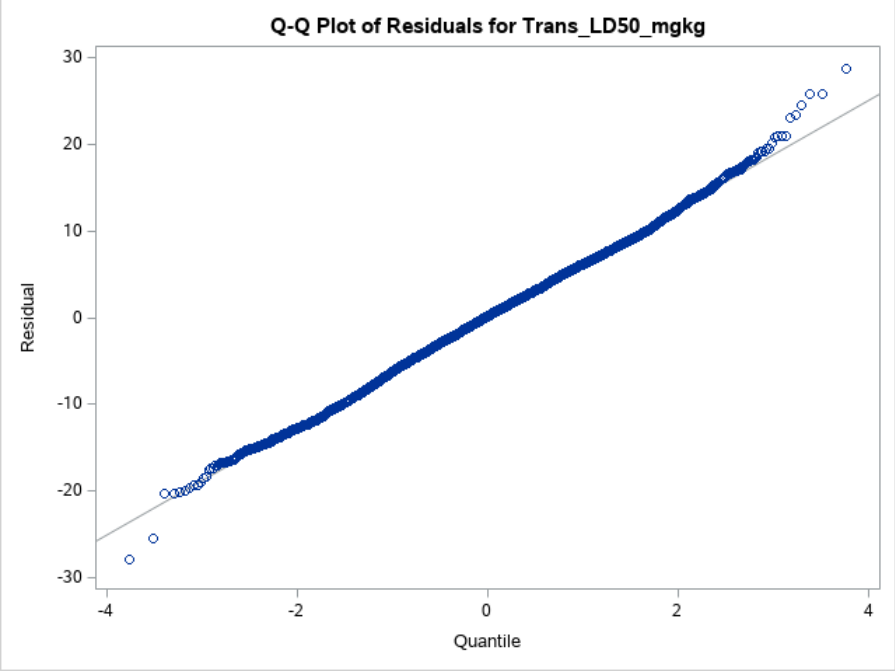
3. Normality of residuals

The normality of residuals is checked using the histogram and QQplot.

Histogram of the residuals



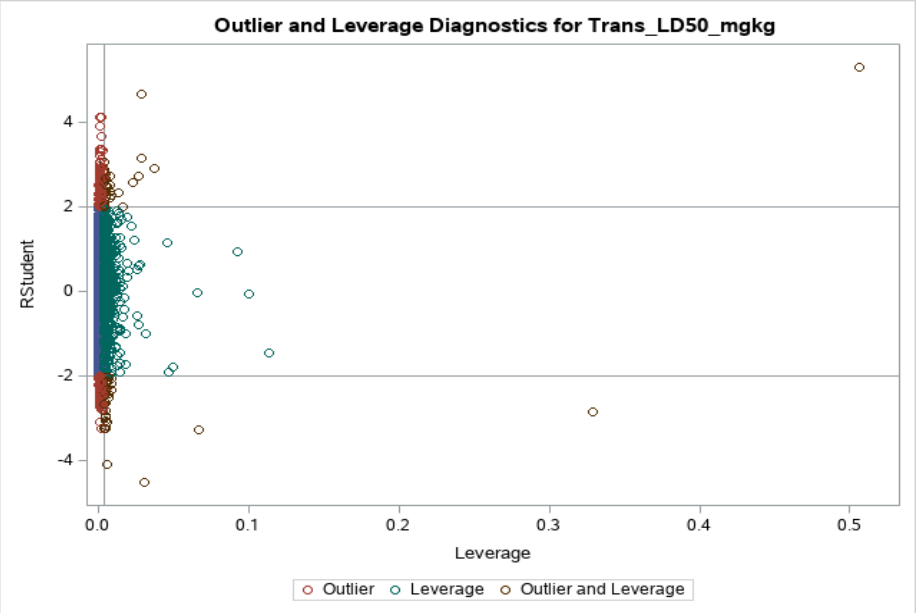
QQplot of the residuals



From histogram and QQplot of the residuals we can say that the residuals are normally distributed.

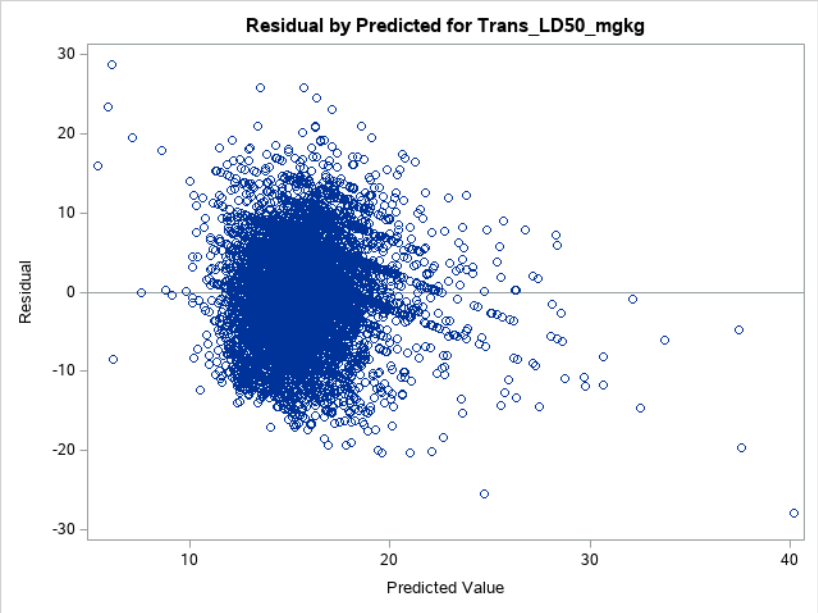
4. Detection of outliers

Following graph clearly shows the presence of the outliers and leverage point in our dataset. Also from the boxplots (create in data visualization part) for LD50 values we have seen the outliers in Relatively harmless categories. Thus the assumption of Absence of outliers is not satisfied.



5. Checking for Homoscedasticity

The constant variance of the residuals is checked with the help of the plot of Residuals v/s Predicted values. If a plot occurs to be of cloud shape then we can say that the residuals have constant variance.



From the above plot we can see the cloud like structure of the residuals. Thus we conclude that the variance of the residuals is constant.

6. Linearity of the model

Linearity of the model can be tested using the residual v/s fitted values plot. A straight horizontal line passing through the points ensures the linearity of the model. These plots are already shown above in detection of homoscedasticity. We can see that horizontal line passing through the points is straight. Thus the model is linear.

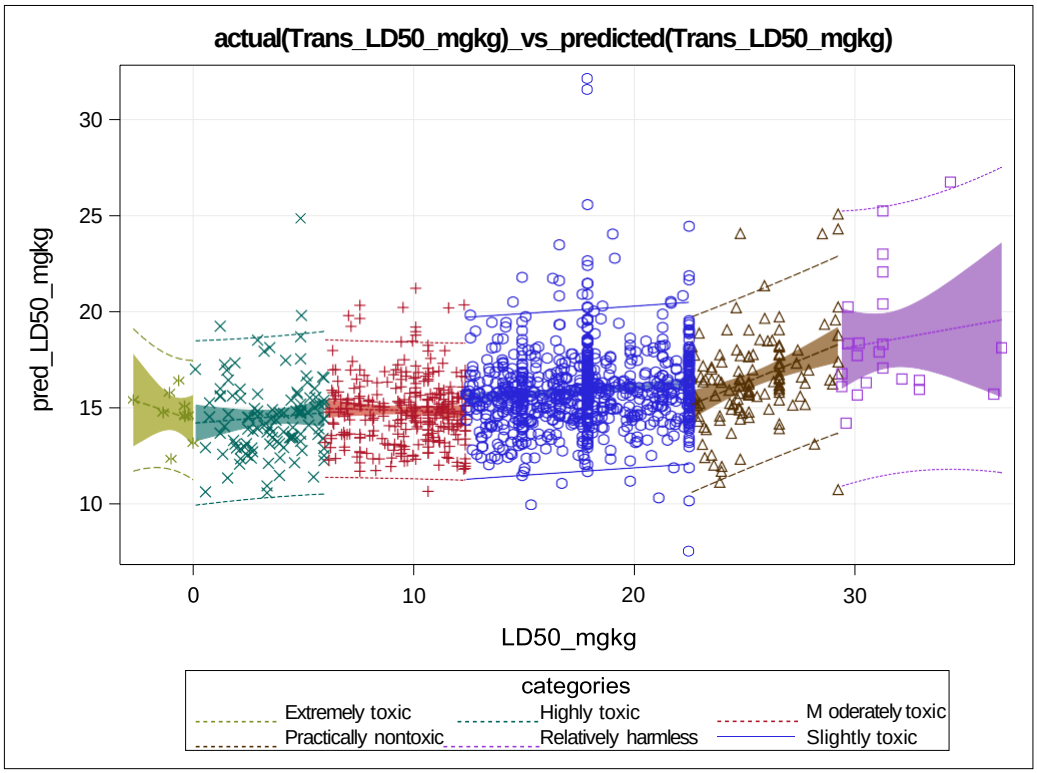
After fitting the model and calculating the predicted values of LD50

Root MSE	9.20042	**R-Square**	0.1362
Dependent Mean	15.65531	**Adj R-Sq**	0.1343
Coeff Var	39.60583		

For fitted model Adj R-square = **0.1343.**

Calculating the MAE from the actual values and predicted values we get,
MAE = 8.31

Scatter plot of Actual LD50 values v/s Predicted LD50 values



From the scatterplot we can see that the LD50 values are lying within the prediction interval but values greater than 15 i.e. for practically toxic and Relatively harmless chemical compounds the LD50 values are not correctly predicted. And thus resulting into higher MAE value. This is because the assumption of no autocorrelation and no outliers are not satisfied. Thus, even after using BoxCox transformation few assumptions are not validated we use non-parametric techniques. We would be using

1. Random Forest Regression
2. Artificial neural Network

RANDOM FOREST REGRESSION

Random forest is a Supervised Learning algorithm which uses ensemble learning method for classification and regression. Random forest is a bagging technique (Bootstrapping + Aggregation).

Bootstrapping: Bootstrapping is a sampling method, where a sample is chosen out of a set, using the replacement method. The learning algorithm is then run on the samples selected.
Aggregation: Model predictions undergo aggregation to combine them for the final prediction to consider all the outcomes possible.

Random Forest Regression operates by constructing a multiple regression trees (type of decision tree) on data (training set) and gets prediction from each of them and it combines the result of multiple regression tree by averaging the predicted value of each regression tree. In Random Forest Regression, the output variable is continuous, and each leaf represents a numerical value. The trees in random forests are run in parallel. There is no interaction between these trees while building the trees. Each tree draws a random sample from the original data set when generating its splits, adding a further element of randomness that prevents over-fitting.

TERMINOLOGY:

- **Node**: A test for the values (data) of a certain attribute.
- **Decision Node**: The other nodes are divided into the further categories.
- **Child Node**: When a node is divided into the other subparts, subparts are called childnodes. And the node which is divided is called the parent node.
- **Leaf**: Terminal node that predicts outcome.
- **Root**: The beginning node that contains the entire dataset.
- **Entropy**: The amount of information disorder or the amount of randomness in the data.
- **Information Gain**: Information collected that can increase the level of certainty in a particular prediction.

WORKING OF RANDOM FOREST REGRESSION

We can understand the working of Random Forest Algorithm with the help of the following step.

- Step 1- Pick at random K data points from the training set.
- Step 2 - Build the regression tree associated with those K data points.
- Step 3 - Choose the number Ntree of trees you want to build and repeat step1 & 2.

- Step 4 - For a new data point, make each one of your Ntree trees predict the value of target for the data point, and assign the new data point the averages across all of the predicted target values.

The Random forest regression has been applied to our data with arbitrarily fixing the values of hyper parameters. And we get the following MAE

Output:
MAE = 4.13

IMPROVING THE MODEL:

There are three general approaches for improving an existing machine learning model:

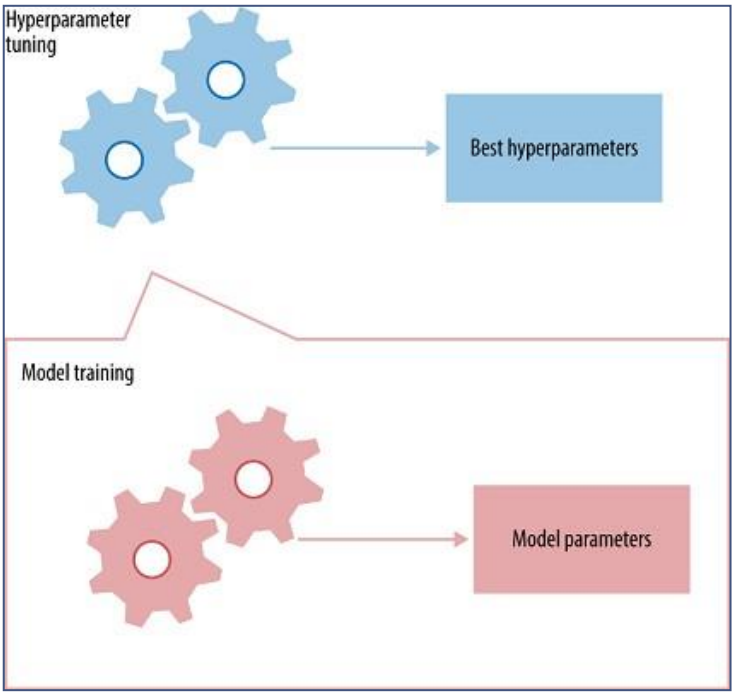
1. Use more (high-quality) data and feature engineering
2. Tune the hyperparameters of the algorithm
3. Try different algorithms.

Here we will use hyperparameter tuning to improve the performance of a trained model.

Hyperparameter Tuning

The best way to think about hyperparameters is like the settings of an algorithm that can be adjusted to optimize performance. In the case of a random forest, hyperparameters include the number of decision trees in the forest and the number of features considered by each tree when splitting a node. (The parameters of a random forest are the variables and thresholds used to split each node learned during training). Hyperparameter tuning relies more on experimental results than theory, and thus the best method to determine the optimal settings is to try many different combinations evaluate the performance of each model. However, evaluating each model only on the training set can lead to one of the most fundamental problems in machine learning: overfitting.

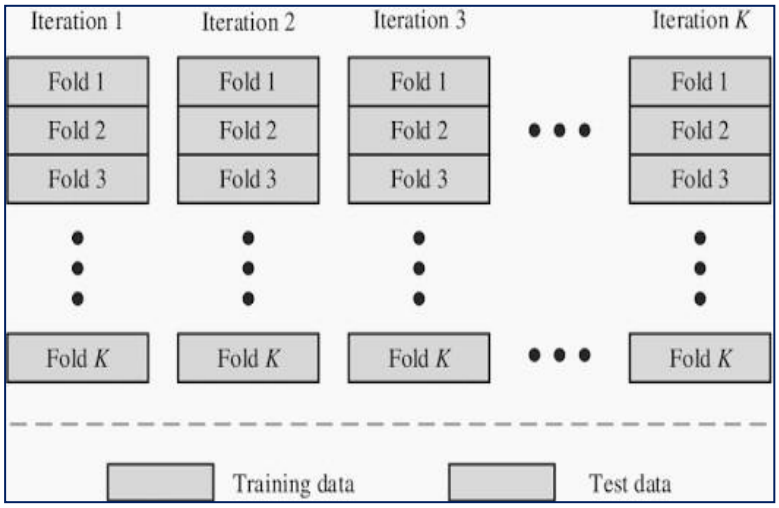
If we optimize the model for the training data, then our model will score very well on the training set, but will not be able to generalize to new data, such as in a test set. When a model performs highly on the training set but poorly on the test set, this is known as overfitting, or essentially creating a model that knows the training set very well but cannot be applied to new problems.



Standard Procedure for hyperparameter Tuning

K-FOLD CROSS VALIDATION:

When we approach a machine learning problem, we make sure to split our data into a training and a testing set. In K-Fold CV, we further split our training set into K number of subsets, called folds. We then iteratively fit the model K times, each time training the data on K-1 of the folds and evaluating on the Kth fold (called the validation data). As an example, consider fitting a model with K = 5. The first iteration we train on the first four folds and evaluate on the fifth. The second time we train on the first, second, third, and fifth fold and evaluate on the fourth. We repeat this procedure 3 more times, each time evaluating on a different fold. At the very end of training, we average the performance on each of the folds to come up with final validation metrics for the model.



K-Fold Cross Validation

After applying parameter tuning to improve the model performance we have the optimal values of the hyperparameters which gives better prediction of the LD50 values.

- n_estimators = number of trees in the forest
- min_sample_split = minimum number of data point spit in a node before the node is split
- min_sample_leaf = minimum number of data points allowed in a leaf node
- max_features = maximum number of features considered for splitting a node
- max_depth = maximum number of levels in each decision tree

Output:

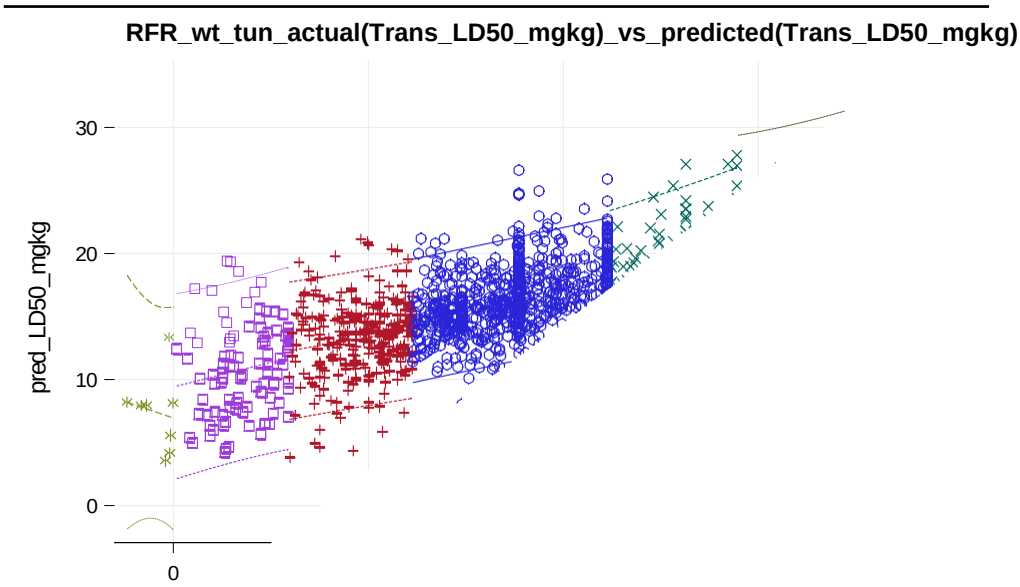
RFR_BEST_PARAMETERS: -

```
{'n_estimators': 800,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 50}
```

We again run the model by giving the appropriate values for the arguments as we got from hyperparameter tuning.

Output:
MAE = 3.86

Thus, it has shown some slight improvement in MAE. After the hyperparameter tuning the MAE has reduced by 0.27 units.
Below is the scatterplot, between the actual values of the LD50 and predicted values of LD50.



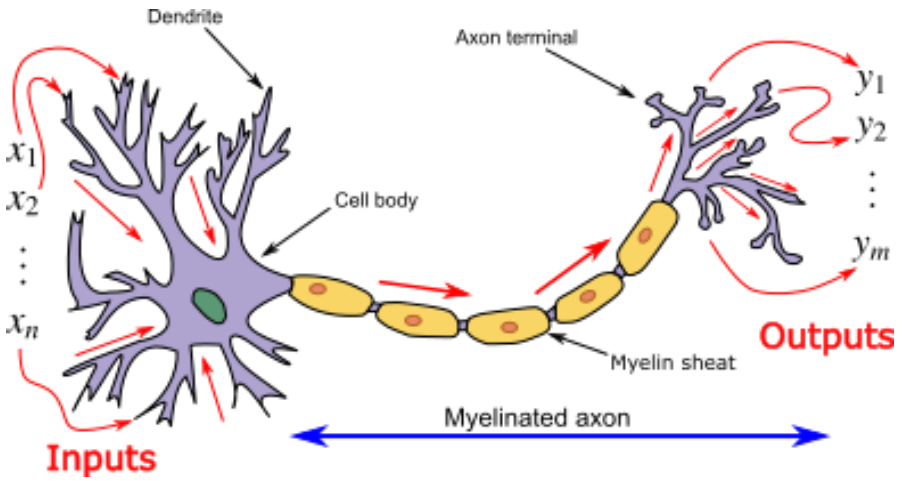
The scatterplot shows how good are the predictions. From the scatterplot we can see that LD50 values of the extremely toxic compounds and relatively harmless are not predicted with good accuracy. The possible reasons for the inaccurate predictions can be the availability of less data for extremely toxic compounds and presence of outliers in relatively harmless category. Also, we can see that most of the predictions are within the prediction interval

ARTIFICIAL NEURAL NETWORK

Artificial Neural Networks or ANN is an information processing paradigm that is inspired by the way the biological nervous system such as brain process information. It is composed of large number of highly interconnected processing elements (neurons) working in unison to solve a specific problem.

❖ Neurons

Biological Neurons (also called nerve cells) or simply neurons are the fundamental units of the brain and nervous system, the cells responsible for receiving sensory input from the external world via dendrites, process it and gives the output through Axons.

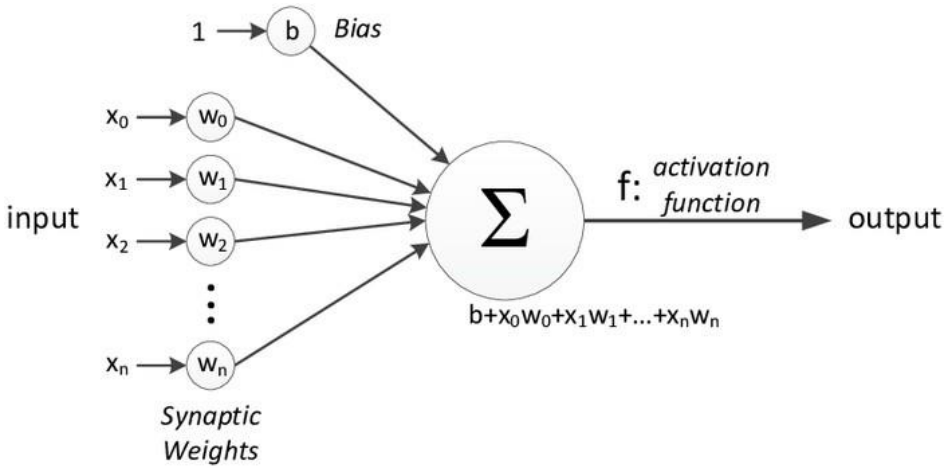


A biological Neuron

- **Cell body (Soma):** The body of the neuron cell contains the nucleus and carries out biochemical transformation necessary to the life of neurons.
- **Dendrites:** Each neuron has fine, hair-like tubular structures (extensions) around it. They branch out into a tree around the cell body. They accept incoming signals.
- **Axon:** It is a long, thin, tubular structure that works like a transmission line.
- **Synapse:** Neurons are connected to one another in a complex spatial arrangement. When axon reaches its final destination it branches again called terminal arborization. At the end of the axon are highly complex and specialized structures called synapses. The connection between two neurons takes place at these synapses.

The following diagram represents the general model of ANN which is inspired by a biological neuron. It is also called Perceptron.

A single layer neural network is called a Perceptron. It gives a single output.



Perceptron

In the represents various inputs(independent variables) to the network. Each of these inputs is multiplied by a connection weight or synapse. The weights are represented as w0, w1, w2, w3….w(n) . Weight shows the strength of a particular node.

b is a bias value. A bias value allows you to shift the activation function up or down.

In the simplest case, these products are summed, fed to a transfer function (activation function) to generate a result, and this result is sent as output.

Mathematically, x1.w1 + x2.w2 + x3.w3 ..... xn.wn = $\sum$ xi.wi

Now activation function is applied $\phi(\sum$ xi.wi)

above figure, for one single observation, x0, x1, x2, x3 ..x(n)

WEIGHT INITIALIZERS:

The neural networks starts with initially assigning some weights randomly to each variable. This initialization is the crucial part of creating the network as inappropriate weights to the variables may result into exploding and vanishing gradient descent. Hence there are some

weight initializers methods for assigning the weights initially. For instance Uniform, he_uniform, glorot_uniform, Xaviers initialization etc.

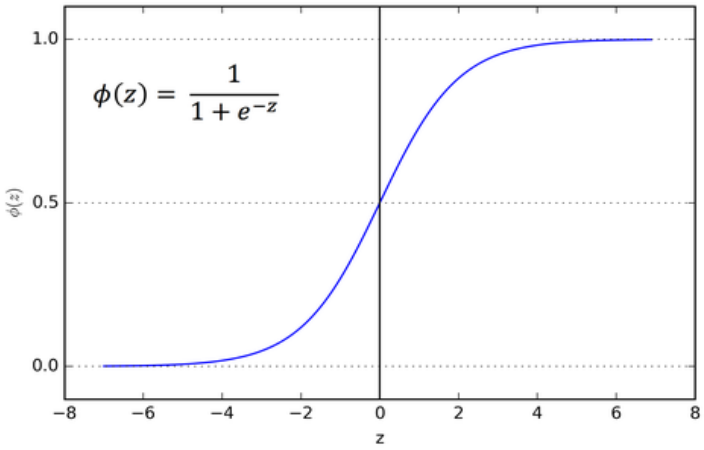
ACTIVATION FUNCTIONS

The Activation function is important for an ANN to learn and make sense of something really complicated. Their main purpose is to convert an input signal of a node in an ANN to an output signal. This output signal is used as input to the next layer in the stack. Activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The motive is to introduce non-linearity into the output of a neuron.If we do not apply activation function then the output signal would be simply linear function (one- degree polynomial). Now, a linear function is easy to solve but they are limited in their complexity, have less power. Without activation function, our model cannot learn and model complicated data such as images, videos, audio, speech, etc.

Types of Activation Functions:

3. Sigmoid Activation Function — (Logistic function)

A Sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve which ranges between 0 and 1, therefore it is used for models where we need to predict the probability as an output.

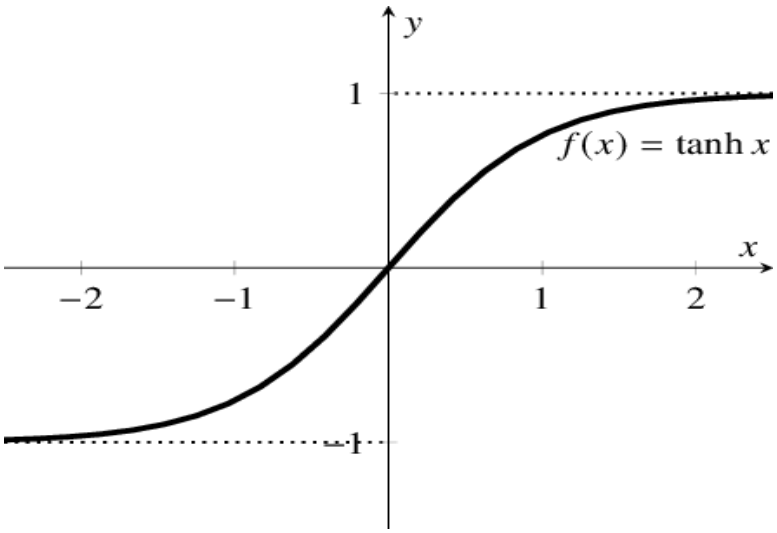


The Sigmoid function is differentiable, means we can find the slope of the curve at any 2 points.

The drawback of the Sigmoid activation function is that it can cause the neural network to get stuck at training time if strong negative input is provided.

4. Hyperbolic Tangent Function —(tanh)

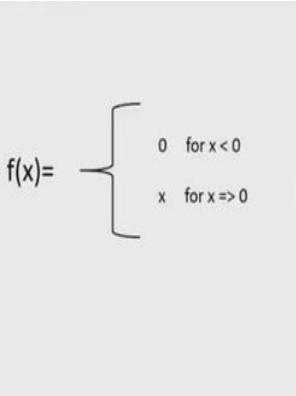
It is similar to Sigmoid but better in performance. It is nonlinear in nature, so great we can stack layers. The function ranges between (-1,1).

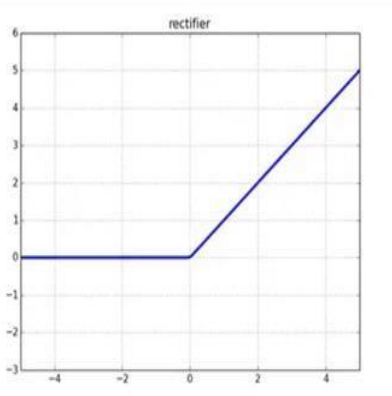


The main advantage of this function is that strong negative inputs will be mapped to negative output and only zero-valued inputs are mapped to near-zero outputs. So, less likely to get stuck during training.

5. Rectified Linear Units — (ReLu)

ReLu is the most used activation function in CNN and ANN which ranges from zero to infinity.[0,∞)





It gives an output ‘x’ if x is positive and 0 otherwise. It looks like having the same problem of linear function as it is linear in the positive axis. Relu is non-linear in nature and a combination of

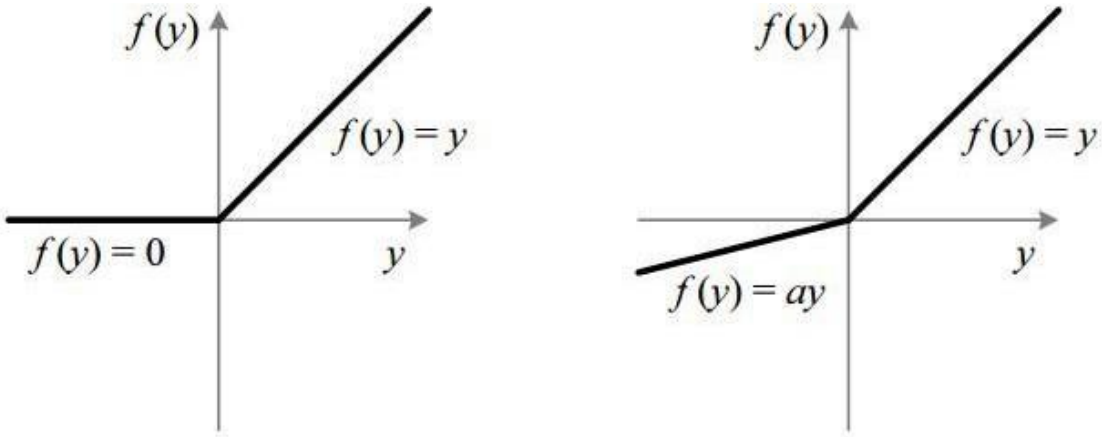
ReLu is also non-linear. In fact, it is a good approximator and any function can be approximated with a combination of Relu.

ReLu is 6 times improved over hyperbolic tangent function.

It should only be applied to hidden layers of a neural network. So, for the output layer use softmax function for classification problem and for regression problem use a Linear function.

Here one problem is some gradients are fragile during training and can die. It causes a weight update which will make it never activate on any data point again. Basically ReLu could result in dead neurons.

To fix the problem of dying neurons, **Leaky ReLu** was introduced. So, Leaky ReLu introduces a small slope to keep the updates alive. Leaky ReLu ranges from -∞ to +∞.



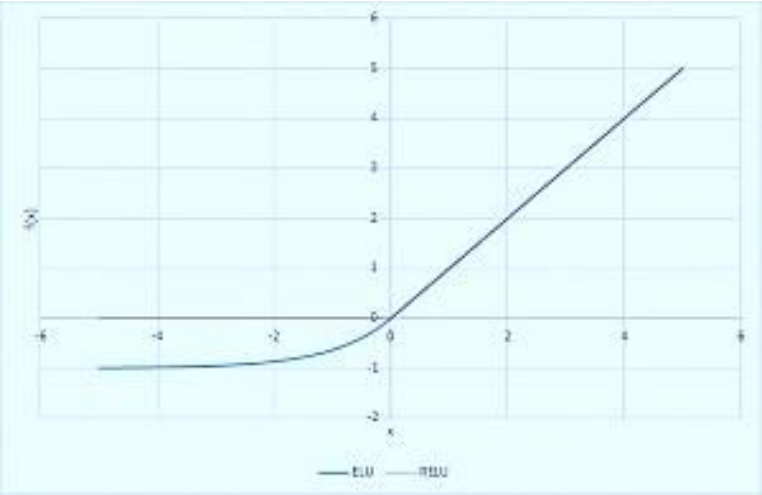
ReLu vs Leaky ReLu

Leak helps to increase the range of the ReLu function. Usually, the value of a = 0.01 or so. When a is not 0.01, then it is called Randomized ReLu.

6. Exponential Linear Unit

Exponential Linear Unit or ELU for short is also a variant of Rectified Linear Unit (ReLu) that modifies the slope of the negative part of the function. Unlike the leaky ReLu and parametric ReLu functions, instead of a straight line, ELU uses a log curve for defining the negative values. It is defined as

f(x) = x, x>=0
= a(e^x-1), x<0



OPTIMIZERS:

There are various optimization techniques for training the neural network Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.

1. Gradient descent:

Gradient descent is a first-order optimization algorithm which is dependent on the first order derivative of a loss function. It calculates that which way the weights should be altered so that the function can reach a minima. Through backpropagation, the loss is transferred from one layer to another and the model's parameters also known as weights are modified depending on the losses so that the loss can be minimized.

2. Stochastic gradient descent:

It's a variant of Gradient Descent. It tries to update the model's parameters more frequently. In this, the model parameters are altered after computation of loss on each training example. So, if the dataset contains 1000 rows SGD will update the model parameters 1000 times in one cycle of dataset instead of onetime as in Gradient Descent.

3. Adagrad:

One of the disadvantages of all the optimizers explained is that the learning rate is constant for all parameters and for each cycle. This optimizer changes the learning rate. It changes the

learning rate 'η' for each parameter and at every time step 't'. It's a type second order optimization algorithm. It works on the derivative of an error function.

4. Adadelta

Adadelta is a more robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients. This way, Adadelta continues learning even when many updates have been done.

5. Adam

Adam (Adaptive Moment Estimation) works with momentums of first and second order. The intuition behind the Adam is that we don't want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for a careful search. In addition to storing an exponentially decaying average of past squared gradients like AdaDelta

6. RMSProp;

RMSprop is a gradient based optimization technique. Gradients of very complex functions like neural networks have a tendency to either vanish or explode as the data propagates through the function. RMSprop deals with the above issue by using a moving average of squared gradients to normalize the gradient. This normalization balances the step size (momentum), decreasing the step for large gradients to avoid exploding, and increasing the step for small gradients to avoid vanishing.

How do Neural networks learn?

Looking at an analogy may be useful in understanding the mechanisms of a neural network. Learning in a neural network is closely related to how we learn in our regular lives and activities — we perform an action and are either accepted or corrected by a trainer or coach to understand how to get better at a certain task. Similarly, neural networks require a trainer in order to describe what should have been produced as a response to the input. Based on the difference between the actual value and the predicted value, an error value also called Cost Function is computed and sent back through the system.

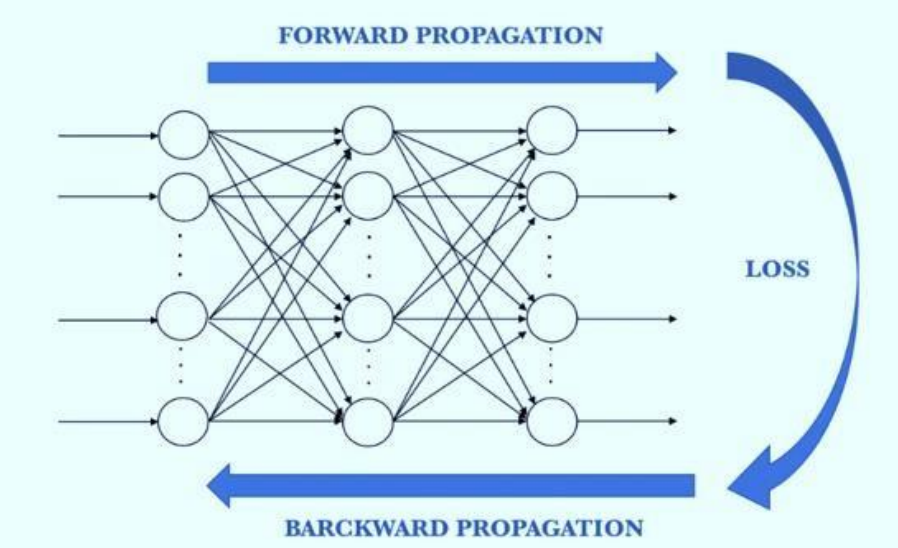
Cost function: mean absolute error

For each layer of the network, the cost function is analyzed and used to adjust the threshold and weights for the next input. Our aim is to minimize the cost function. The lower the cost function, the closer the actual value to the predicted value. In this way, the error keeps becoming marginally lesser in each run as the network learns how to analyze values.

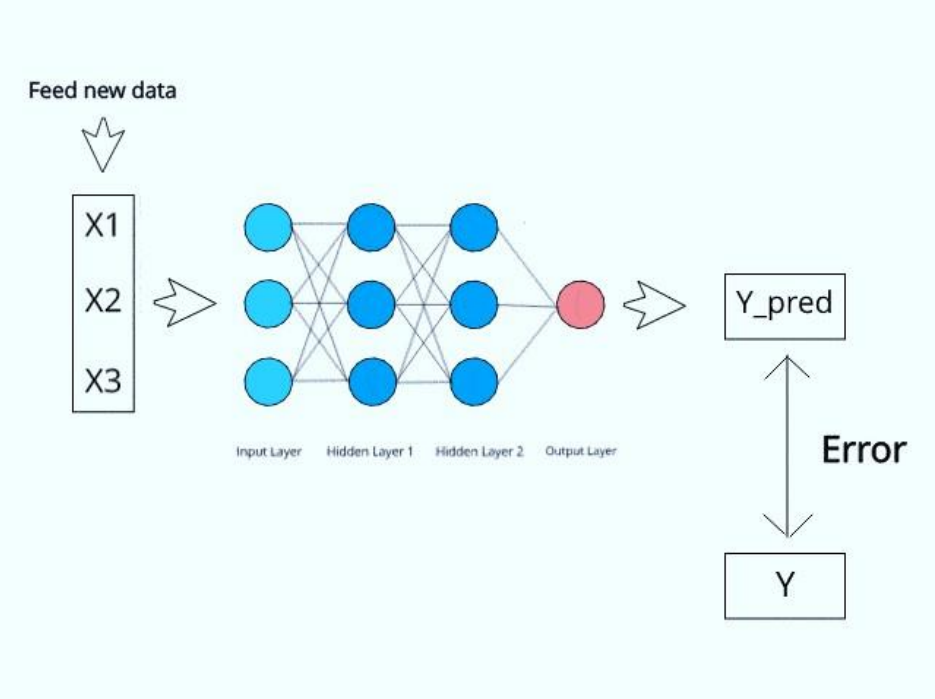
We feed the resulting data back through the entire neural network. The weighted synapses connecting input variables to the neuron are the only thing we have control over.

As long as there exists a disparity between the actual value and the predicted value, we need to adjust those wights. Once we tweak them a little and run the neural network again, A new Cost function will be produced, hopefully, smaller than the last.

We need to repeat this process until we scrub the cost function down to as small as possible.



The procedure described above is known as **Back-propagation** and is applied continuously through a network until the error value is kept at a minimum.



Training ANN

Step-1 → Randomly initialize the weights to small numbers close to 0 but not 0.

Step-2 → Input the first observation of your dataset in the input layer, each feature in one node.

Step-3 → Forward-Propagation: From left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the predicted value.

Step-4 → Compare the predicted result to the actual result and measure the generated error(Cost function).

Step-5 → Back-Propagation: from right to left, the error is backpropagated. Update the weights according to how much they are responsible for the error. The learning rate decides how much we update weights.

Step-6 → Repeat step-1 to 5 and update the weights after each observation(Reinforcement Learning)

Step-7 → When the whole training set passed through the ANN, that makes and epoch. Redo more epochs.

Thus as ANN works very well for non linear data we try to create a network for prediction of the LD50. Here out of all the arguments that needs to be fixed for a good network we would be selecting only 3 of the them namely activation function, weight initializer and optimizers.

Now we try to create a network with 2 layers consisting of 15 nodes each with uniform initialization of the weights and adam optimizers. After giving the predictors to our network we run it 250 times i.e. weights are updated 250 times before convergence. we get the predicted values.

Mean Absolute Error of the train data = 4.14
Mean Absolute Error of the test data = 4.15

Now to choose the appropriate values of the arguments i.e choosing the right activation function, initializers, optimizers and number of layers and nodes in each layer we perform hyperparameter tuning.

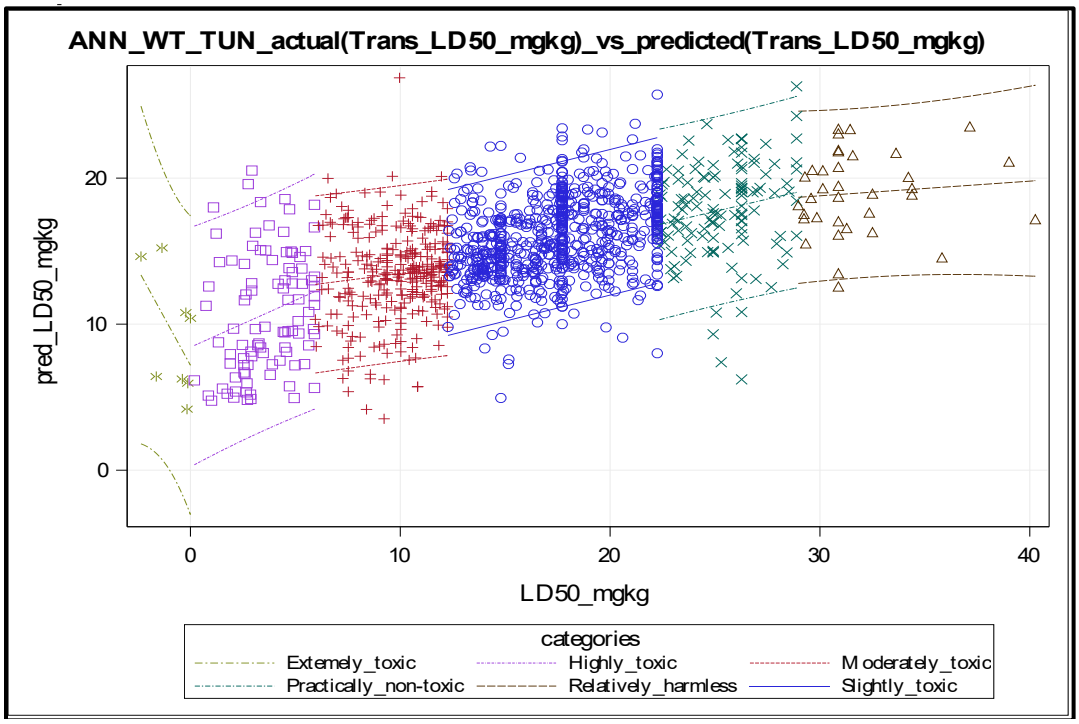
```
In [43]: grid_result.best_params_
Out[43]:
{'activ': 'elu',
 'initial': 'he_uniform',
 'layers': (17, 17, 15),
 'opti': 'RMSprop'}
```

Setting these arguments to their respective values we again create a neural network and run the network. Output,

Mean Absolute Error of the train data = 4.09
Mean Absolute Error of the test data = 4.06

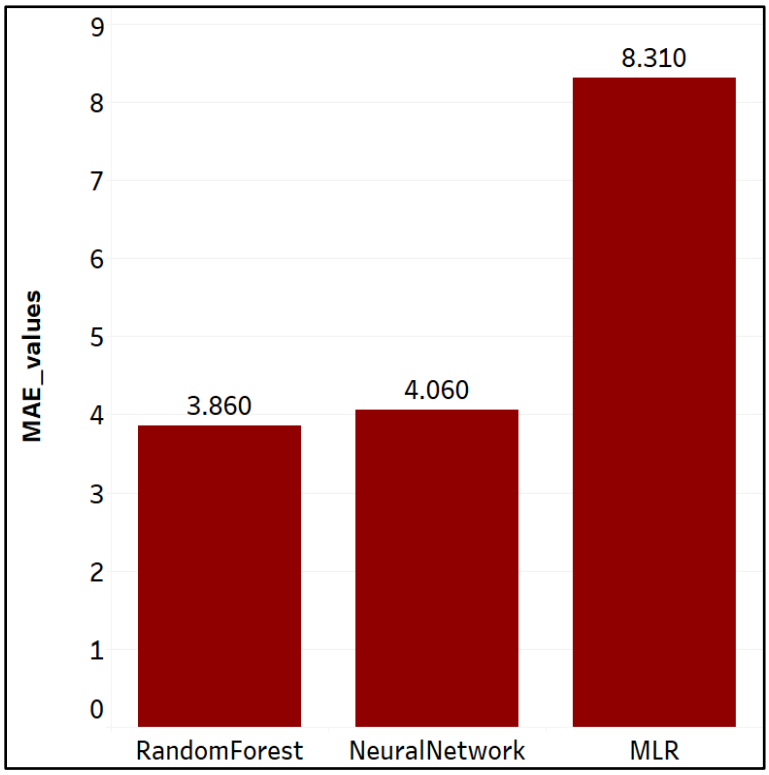
There is not much difference between the MAE on train data and that od test data thus there is no overfitting. Also, after hyperparameter tuning there has been a decrease in MAE of 0.09 units.

Below is the scatterplot, between the actual values of the LD50 and predicted values of LD50.



The scatterplot shows how good are the predictions. From the scatterplot we can see that LD50 values of the extremely toxic compounds and relatively harmless are not predicted with good accuracy. The possible reasons for the inaccurate predictions can be the availability of less data for extremely toxic compounds and presence of outliers in relatively harmless category. Also, we can see that most of the predictions are within the prediction interval.

COMPARISON BETWEEN THE TECHNIQUES USED FOR PREDICTING THE LD50 VALUES:



Thus, we can see that "**Random forest**" gives the lower MAE value thus we can say that "**Random forest**" performs better than **Neural network & MLR.**

OBJECTIVE 2
TO FIND OUT THE IMPORTANT FEATURES THAT ARE

AFFECTING LD50

Out of all 33 predictors there are may be only few predictors that are important for the prediction of the LD50 values. To find the most important features for predicting the LD50 values we will use Permutation Importance technique. This technique will give us the variables that are contributing the most for predicting LD50 values.

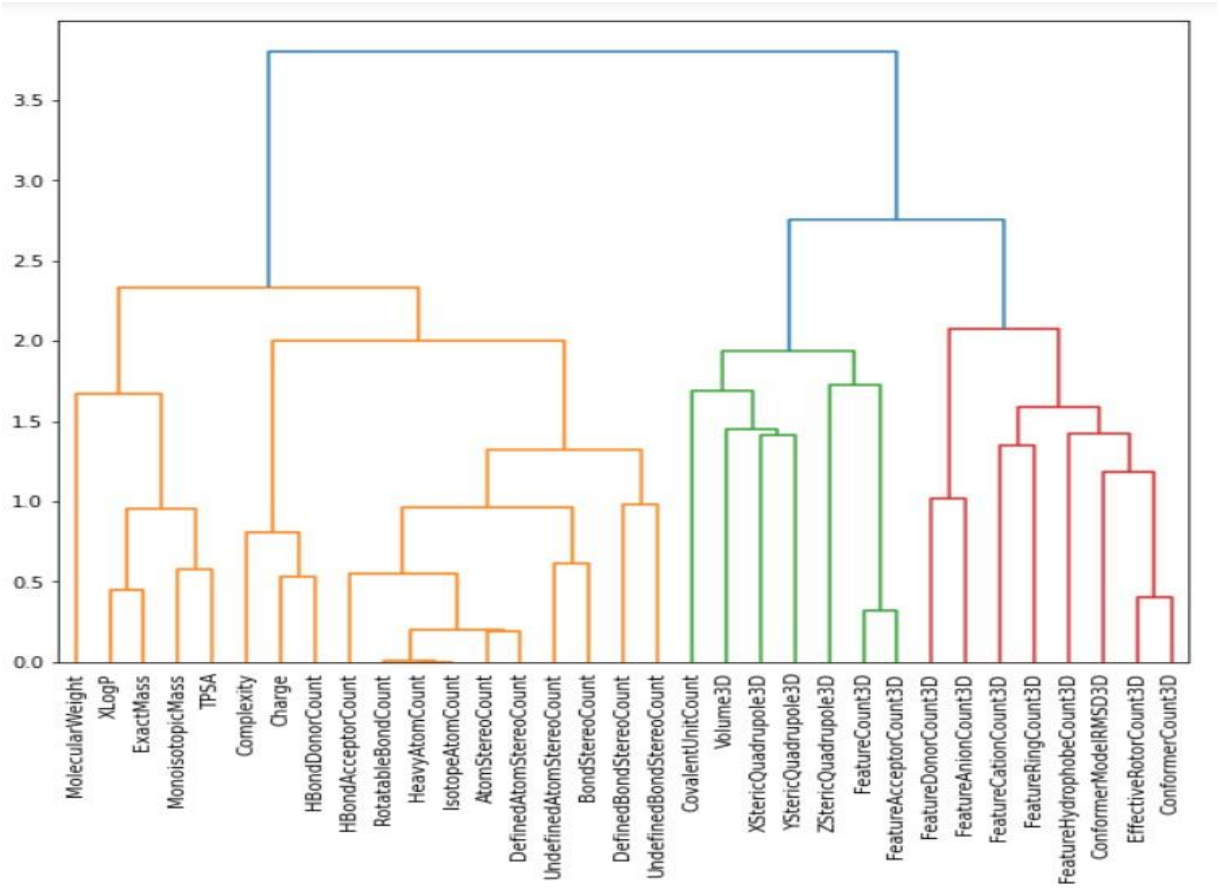
PERMUTATION IMPORTANCE:

Permutation feature importance is a model inspection technique that can be used for any fitted estimator when the data is tabular. This is especially useful for non-linear or opaque estimators. The permutation feature importance is defined to be the decrease in a model score when a single feature value is randomly shuffled. This procedure breaks the relationship between the feature and the target, thus the drop in the model score is indicative of how much the model depends on the feature. This technique benefits from being model agnostic and can be calculated many times with different permutations of the feature.

DRAWBACK:

When two features are correlated and one of the features is permuted, the model will still have access to the feature through its correlated feature. This will result in a lower importance value for both features, where they might ACTUALLY be important.
One way to handle multicollinear features is by performing hierarchical clustering on the Spearman rank-order correlations, picking a threshold, and keeping a single feature from each cluster. First, we plot a dendrogram of the predictors

Clustering the variables on the basis of correlation.



Thus, the predictors are clustered on the basis of correlation between them. The primary motive is to select only one predictor out of highly correlated features

Table showing the clusters of the variables

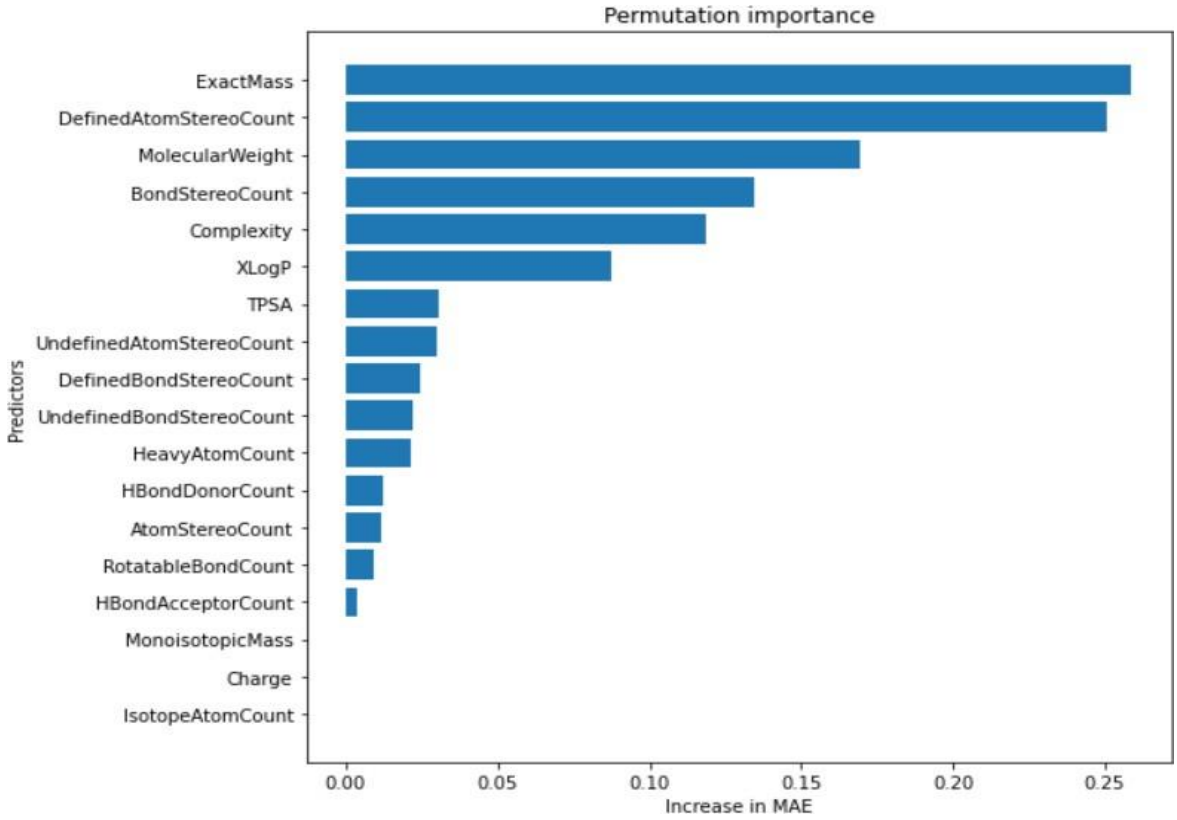
Predictors	Cluster
XStericQuadrupole3D	1
EffectiveRotorCount3D	1
ConformerModelRMSD3D	1
RotatableBondCount	1
XLogP	2
TPSA	3
FeatureAcceptorCount3D	3
HBondAcceptorCount	3
YStericQuadrupole3D	4
ZStericQuadrupole3D	4
MolecularWeight	4
Complexity	4
MonoisotopicMass	4
ExactMass	4
Volume3D	4
HeavyAtomCount	4
FeatureRingCount3D	5
FeatureCount3D	5
Charge	6
IsotopeAtomCount	7
UndefinedBondStereoCount	8
ConformerCount3D	9
BondStereoCount	10
DefinedBondStereoCount	10
FeatureHydrophobeCount3D	11
AtomStereoCount	12
UndefinedAtomStereoCount	13
CovalentUnitCount	14
FeatureAnionCount3D	15
FeatureDonorCount3D	16
HBondDonorCount	16
FeatureCationCount3D	17
DefinedAtomStereoCount	18

Next, we manually pick a threshold by visual inspection of the dendrogram to group our features into clusters and choose a feature from each cluster to keep, select those features from our dataset, and train a new random forest. The test accuracy of the new random forest did not change much compared to the random forest trained on the complete dataset.

Below is the table representing the permutation importance score of the selected features. These scores represent the increase in the MAE if the respective feature is not present while fitting the model.

Features	Score
ExactMass	0.258910
DefinedAtomStereoCount	0.250489
MolecularWeight	0.169391
BondStereoCount	0.134736
Complexity	0.118503
XLogP	0.087534
TPSA	0.030562
UndefinedAtomStereoCount	0.029943
DefinedBondStereoCount	0.024567
UndefinedBondStereoCount	0.022056
HeavyAtomCount	0.021540
HBondDonorCount	0.012050
AtomStereoCount	0.011615
RotatableBondCount	0.008955
HBondAcceptorCount	0.003543
MonoisotopicMass	0.000065
Charge	0.000000
IsotopeAtomCount	-0.000032

Graphically representing the scores,



Now, we run the random forest model with only selected predictors we get,

Accuracy of the model with only selected predictors = 3.99

Accuracy of the model with all predictors = 3.86

Comparing the MAEs of both the model, there is small difference of 0.13 units.

Thus, we can conclude that ExactMass, DefinedAtomStereoCount and MolecularWeight are the top 3 most contributing variables for predicting the LD50.

OBJECTIVE 3
DEPENDENCY OF DIFFERENT ROUTE OF EXPOSURE ON EACH OTHER FOR CHEMICAL COMPOUNDS.

INTRODUCTION:

Toxicity can be defined as the relative ability of a substance to cause adverse effects in living organisms. In addition to dose, other factors may also influence the toxicity of the compound such as the route of entry, duration and frequency of exposure, variations between different species (interspecies) and variations among members of the same species (intraspecies). To apply these principles to hazardous materials response, the routes by which chemicals enter the human body will be considered first.

ROUTES OF EXPOSURE:

There are four routes by which a substance can enter the body: inhalation, skin (or eye) absorption, ingestion, and injection.

1. **Inhalation**: For most chemicals in the form of vapors, gases, mists, or particulates, inhalation is the major route of entry. Once inhaled, chemicals are either exhaled or deposited in the respiratory tract. If deposited, damage can occur through direct contact with tissue or the chemical may diffuse into the blood through the lung-blood interface. Upon contact with tissue in the upper respiratory tract or lungs, chemicals may cause health effects ranging from simple irritation to severe tissue destruction. Substances absorbed into the blood are circulated and distributed to organs that have an affinity for that particular chemical. Health effects can then occur in the organs, which are sensitive to the toxicant.

2. **Skin (or eye) absorption**: Skin (dermal) contact can cause effects that are relatively innocuous such as redness or mild dermatitis; more severe effects include destruction of skin tissue or other debilitating conditions. Many chemicals can also cross the skin barrier and be absorbed into the blood system. Once absorbed, they may produce systemic damage to internal organs. The eyes are particularly sensitive to chemicals. Even a short exposure can cause severe effects to the eyes or the substance can be absorbed through the eyes and be transported to other parts of the body causing harmful effects.

3. **Ingestion**: Chemicals that inadvertently get into the mouth and are swallowed do not generally harm the gastrointestinal tract itself unless they are irritating or corrosive.Chemicals that are insoluble in the fluids of the gastrointestinal tract (stomach, small, and large intestines) are

generally excreted. Others that are soluble are absorbed through the lining of the gastrointestinal tract. They are then transported by the blood to internal organs where they can cause damage.
4. **Injection**: Substances may enter the body if the skin is penetrated or punctured bycontaminated objects. Effects can then occur as the substance is circulated in the blood and deposited in the target organs.
During experiment, most of chemicals are given to target substance from different route of exposure as mentioned earlier. Because of varsity of route of exposure, researcher is able to study multiple aspects (like effect, adverse point of exposure, etc) of effect of chemical on different routes.
Since there is lot of research goes on this topic, UNL Environmental Health and Safety releases **'TOXICOLOGY AND EXPOSURE GUIDELINES'**. According to those guideline, route of exposure and dose response terms are summarized as given in following table.

Category	Exposure Time	Route of Exposure	Toxic Effects	
			Human	Animal
TD_{LO}	Acute or chronic	All except inhalation	Any nonlethal	Reproductive, Tumorigenic
TC_{LO}	Acute or chronic	Inhalation	Any nonlethal	Reproductive, Tumorigenic
LD_{LO}	Acute or chronic	All except inhalation	Death	Death
LD_{50}	Acute	All except inhalation	Not applicable	Death (statistically determined)
LC_{LO}	Acute or chronic	Inhalation	Death	Death
LC_{50}	Acute	Inhalation	Not applicable	Death (statistically determined)

Summary of Dose Response Terms

(Ref: https://ehs.unl.edu/documents/tox_exposure_guidelines.pdf)

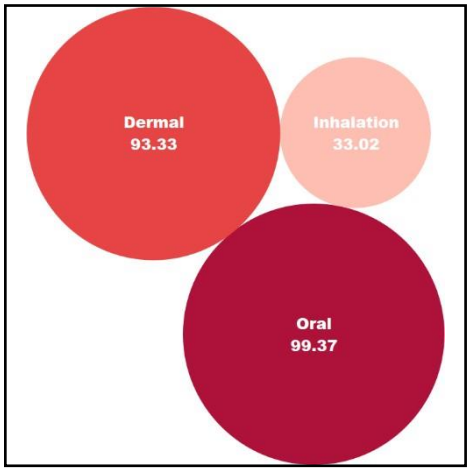
From the above summary table, it is evident that there are LD_{50} values exists for Oral and Dermal routes of exposure. While for Inhalation, LC_{50} values are accounted.

DATA EXTRACTION:

For Objective 3, Data is extracted from ECHA website (https://echa.europa.eu/).
Data consist of 4 columns, 'Chemical Compound', 'LD_{50} value for Oral (mg/kg)', 'LC_{50} value for Inhalation (mg/L)', 'LD_{50} value for Dermal (mg/kg)'.

DATA VISUALIZATION:

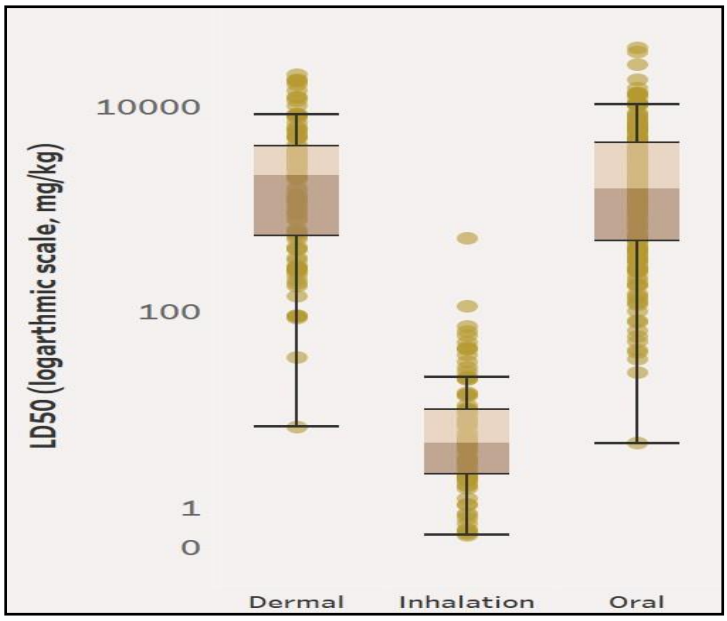
1. Bubble Chart:



Here data has 99.37% of oral observation, 93.33% of Dermal observation & 33.02% of Inhalation observation available for LD_{50} value for 315 chemical compounds.

2. Box-Whisker Plot:

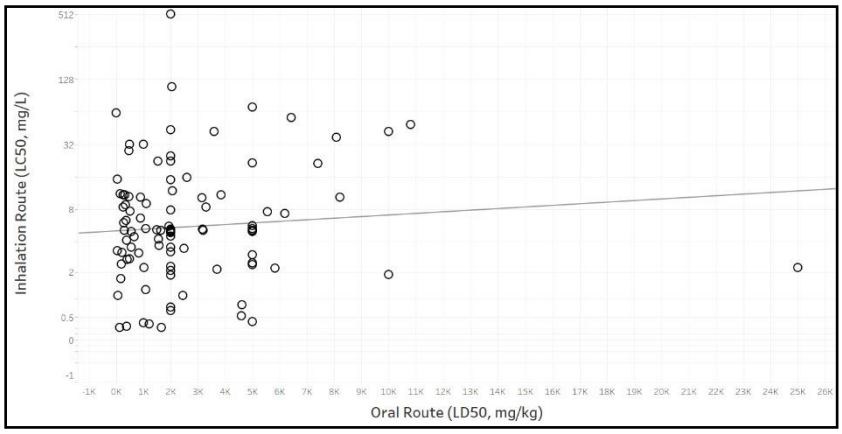
From Box-Whisker plot for log(LD_{50}), we can interpret that there exist outliers in LD_{50} value for each route. Except Inhalation route of exposure, data for other routes of exposure are not symmetric.



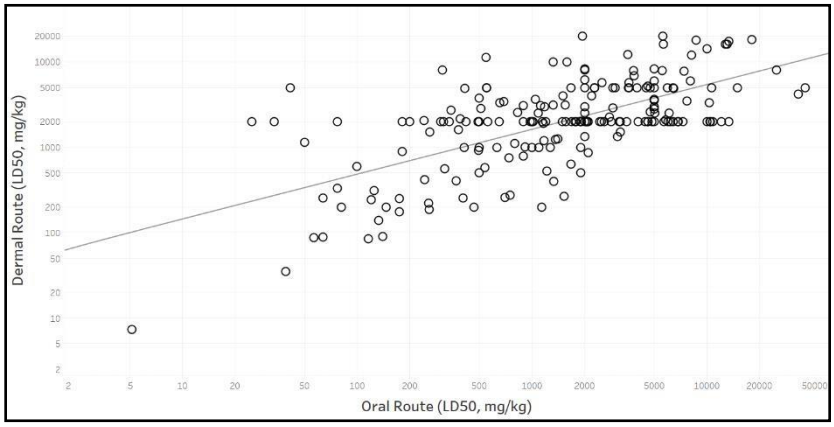
3. Scatter Plot:

A scatter plot is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data.
We can transform variable to log values so that data will look much clearer as compare to regular value.

Scatter plot of Oral LD_{50} (on regular scale) vs Inhalation LC_{50} (on logarithmic scale).

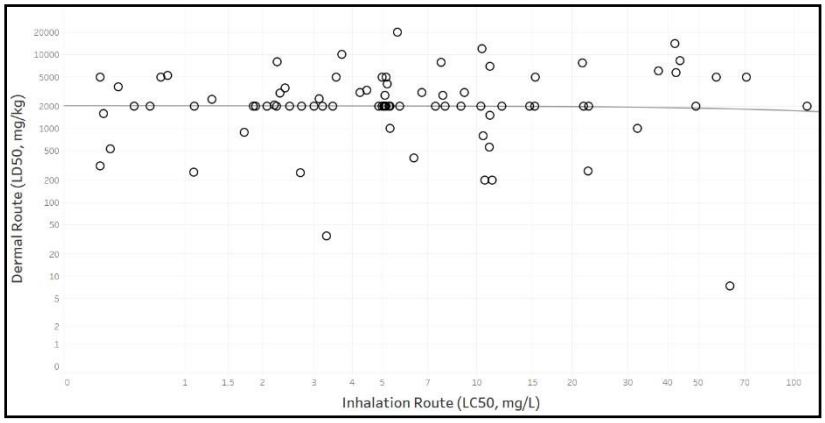


Scatter plot of Oral LD_{50} (on logarithmic scale) vs Dermal LD_{50} (on logarithmic scale)



Trend line for scatter plot shows that there is moderate correlation between LD_{50} values of Oral and Dermal routes of exposure. But there is slight correlation between LD_{50} values of Oral and Inhalation routes of exposure.

Scatter plot of Inhalation LD_{50} vs Dermal LD_{50}



Similarly, for LD_{50} value of Dermal and Inhalation route of exposures, Scatter plot conclude that slight correlation exists.

DATA ANALYSIS:

Data in imported in python using pandas module.

Overlook of data:

```
In [2]: df = pd.read_csv(r'C:\Users\DELL\Desktop\project\Analysis\Objective 3\final_data.csv')
        df.head()
```

Out[2]:

	Chemical Compound	Oral	Inhalation	Dermal
0	3-Chloroaniline	256.0	NaN	223.0
1	Tetradonium bromide	390.0	NaN	2150.0
2	N,N-dimethylpyridin-4-amine	140.0	NaN	90.0
3	Urea, N-(hexahydro-6-methyl-2-oxo-4-pyrimidinyl)-	2000.0	NaN	2000.0
4	1,4-Dichlorobenzene	2000.0	5.07	2000.0

```
In [3]: df.describe()
```

Out[3]:

	Oral	Inhalation	Dermal
count	313.000000	104.000000	294.000000
mean	3153.789032	16.368577	3117.555068
std	4006.930336	52.198244	3252.395000
min	5.090000	0.270000	7.350000
25%	977.000000	2.722500	2000.000000
50%	2000.000000	5.125000	2000.000000
75%	5000.000000	11.000000	3475.000000
max	36640.000000	515.000000	20000.000000

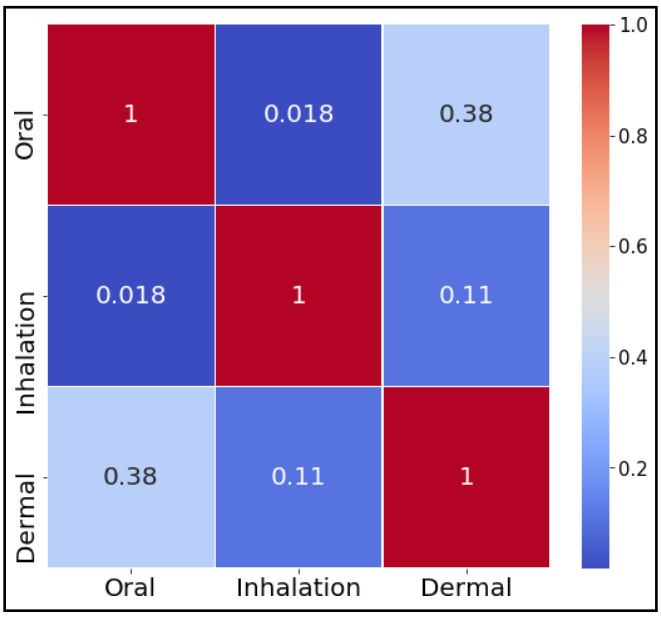
1. Correlation Analysis:

A correlation coefficient is a numerical measure of some type of correlation, meaning a statistical relationship between two variables. We have data in 2 different unit format. Hence it is necessary to check correlation between different routes of exposure.

Hence correlation matrix is

	Oral Route	Inhalation Route	Dermal Route
Oral Route	1.0000	0.0185	0.3847
Inhalation Route	0.0185	1.0000	0.1078
Dermal Route	0.3847	0.1078	1.0000

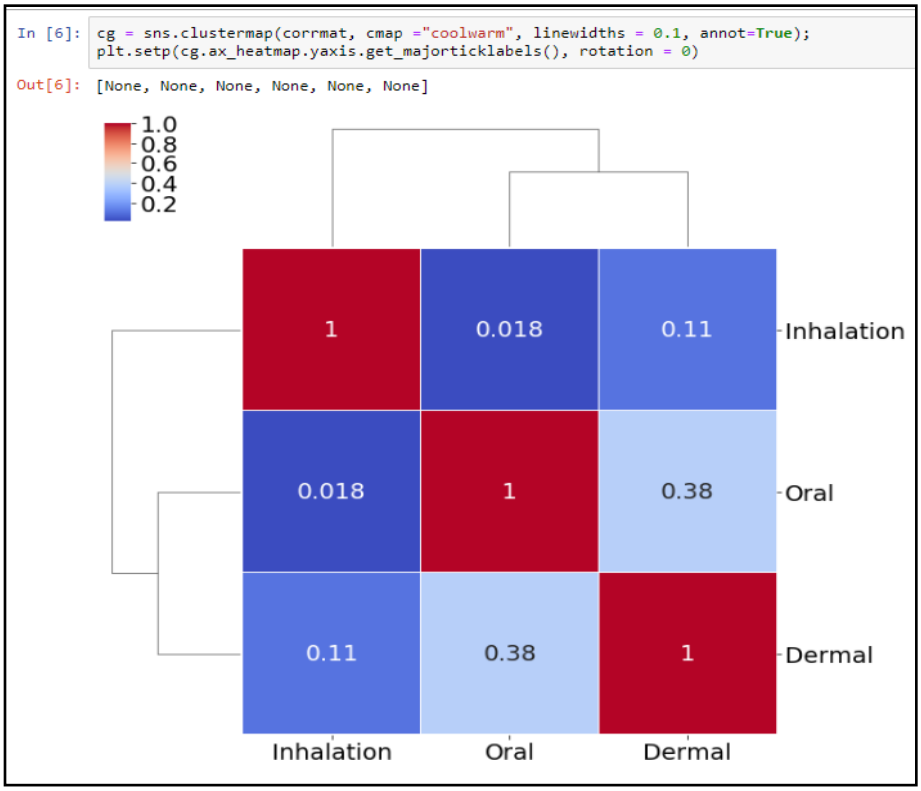
And Heatmap for given correlation matrix is as follow,



From heatmap, we can conclude that, there is moderate positive correlation exists between Oral and Dermal Route of exposure. There is weak correlation between Inhalation and Dermal route of exposure. And there is slightly positive correlation between Oral and Inhalation route of exposure.

Even Grid Heat map explains relationship of routes in better way. as below,

```
In [6]: cg = sns.clustermap(corrmat, cmap ="coolwarm", linewidths = 0.1, annot=True);
        plt.setp(cg.ax_heatmap.yaxis.get_majorticklabels(), rotation = 0)
Out[6]: [None, None, None, None, None, None]
```



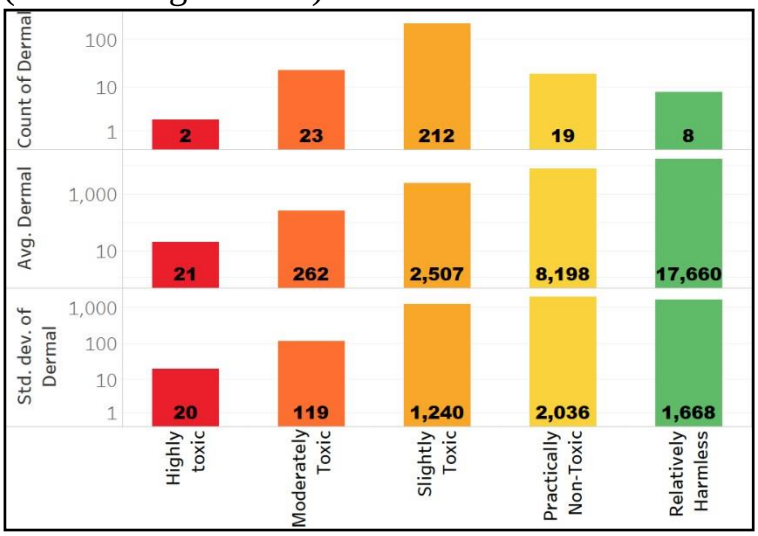
Conclusion of Correlation analysis:

Change (increment or decrement) in LD_{50} value of 'Oral Route' of exposure made changes on LD_{50} value of 'Dermal Route' of exposure. Hence it is possible to fit regression model such that we can predict LD_{50} value of Dermal Route using LD_{50} value of Oral Route.

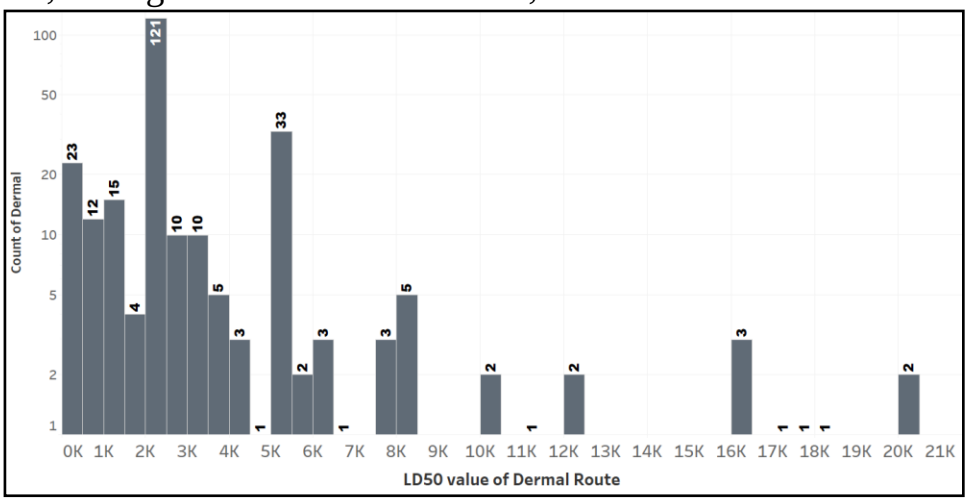
Predicting LD_{50} Values of Dermal Route of Exposure with and without using LD_{50} Values of Oral Route of Exposure:

It is cost consuming, when it came to testing toxicity for each of route of exposure. Hence, we can predict LD_{50} value of Dermal route of exposure using available data (i.e. LD_{50} value for 'Oral Route').

Following graph represent columns for count, average, standard deviation for each categories (Scale is logarithmic).



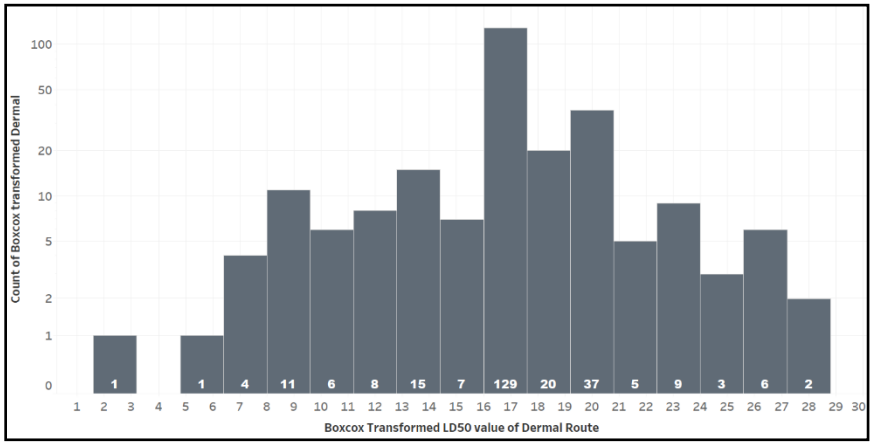
First choice for prediction is linear regression. Assumption for linear regression is normality. Hence, Histogram for data is as follow,



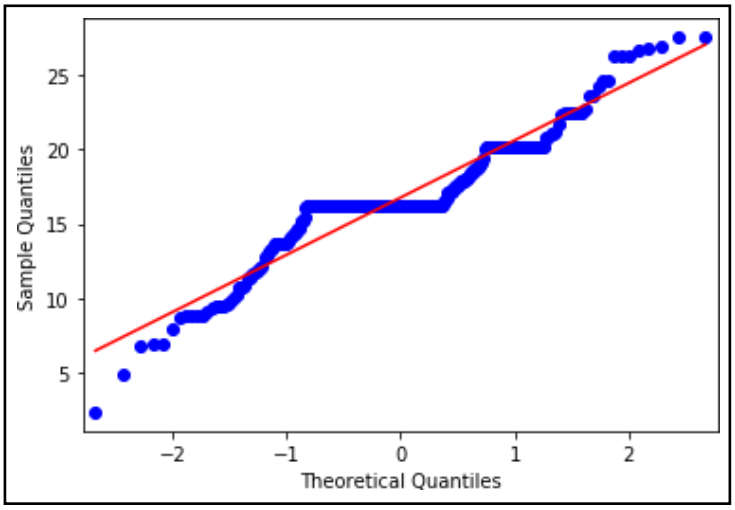
Since the data seems to be non-normal, We can transform data using Box Cox transformation.

```
In [7]: from scipy.stats import boxcox

In [8]: df['Dermal']

Out[8]: 0          7.35
        1       2000.00
        2       2000.00
        3         35.00
        4       5000.00
                  ...
        274     5000.00
        275    18080.00
        276     8000.00
        277     4173.00
        278     5000.00
        Name: Dermal, Length: 264, dtype: float64

In [9]: trn = boxcox(df['Dermal'], lmbda=0.18)

In [10]: df['boxcox_dermal'] = trn.transpose()

In [11]: df.describe()
```

Hence, Histogram for Box-Cox transformed LD_{50} value for dermal route of exposure



Then probability plot for testing normality of data,



And also using Shapiro-Wilks test,

```
In [16]: from scipy.stats import shapiro
         stat, p = shapiro(df.boxcox_dermal)
         print('Statistics=%.3f, p=%.3f' % (stat, p))

         Statistics=0.911, p=0.000
```

It concludes that the data is non-normal (p-value <0.005).

Normality assumption is not satisfied. Therefore, linear regression is not useful for this dataset. In such situations, Non-Linear regression techniques will be most suitable one.
And even if we assumed normality for data, linear regression can predict poorly (with R^2 value is 0.1155) in this example. Hence It is not useful to fit linear regression model.
Random Forest Regression is considered to be powerful and accurate ML technique, good performance on many problems, including non-linear. Hence it is more preferable choice in Non- Linear regression model.

We will try to compare two models. One will be with the top 3 predictors found in objective 2 and another model will be with LD40 values for oral LD50 values and with 3 most important predictors. Following are the 3 predictors that we will be using.

- Molecular Weight,
- DefinedAtomStereoCount,
- Exact Mass.

Boosting techniques can help to precision of model. Hence model like XGBoost, AdaBoost, CatBoost are being used. Here we are fitting AdaBoost technique with base_estimator RandomForestRegressor(max_depth = 50, max_features=5).
Hyperparameter tuning is applied for AdaBoostRegressor. So that we get model with best parameter combination. Therefore, post performing hyperparameter tuning, best parameters are as follows;

- learning_rate = 0.1
- loss = ‘exponential’
- base_estimator = ‘RandomForestRegressor(max_depth = 50, max_feature=5)’

<u>WITHOUT ORAL LD50</u>

Fitting AdaBoostRegressor model with hyperparameter,

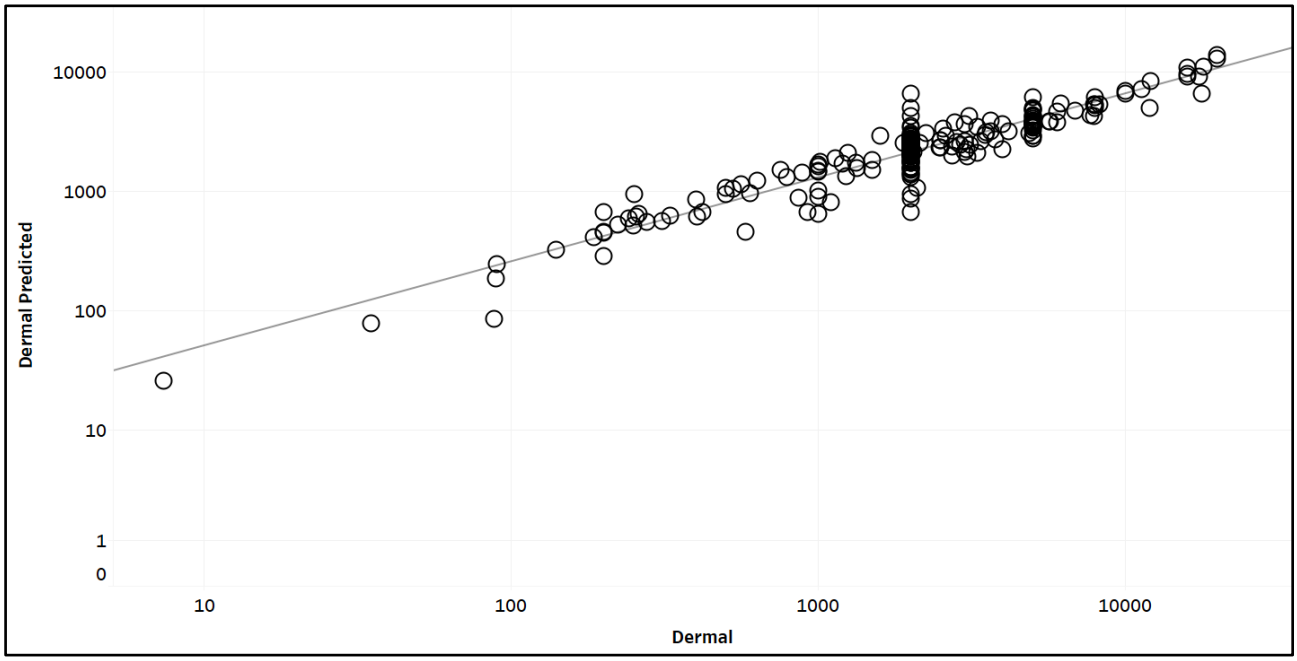
1. RandomForestRegressor(max_depth = 50, max_features=3).
2. learning_rate = 0.1
3. loss = ‘exponential’

Features used:
1. Molecular Weight

2. Defined Stereo Atom Count
3. Exact Mass

We can get MAE value **1.2459**.

Scatter plot for fitted model is as follow,



Now we will be fitting the same model with one additional parameter i.e LD50 values for oral route of exposure.

Fitting AdaBoostRegressor model with hyperparameter,

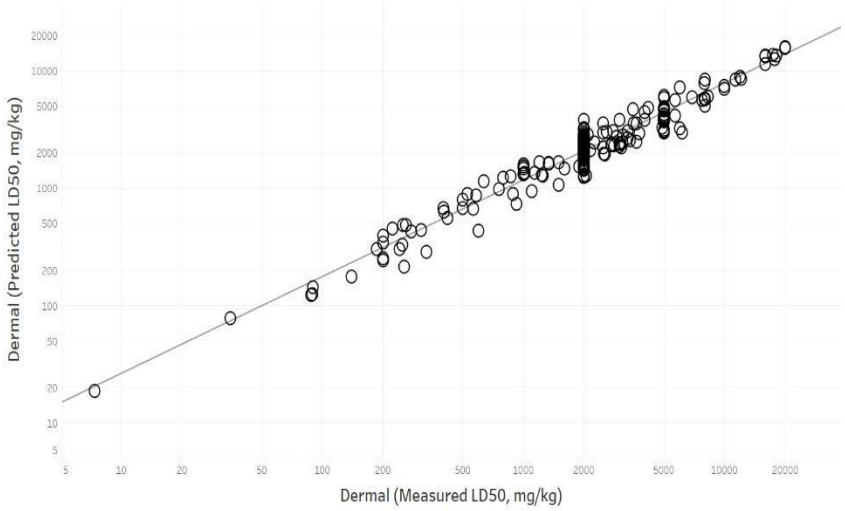
1. RandomForestRegressor(max_depth = 50, max_features=3).
2. learning_rate = 0.1
3. loss = ‘exponential’

Features used:

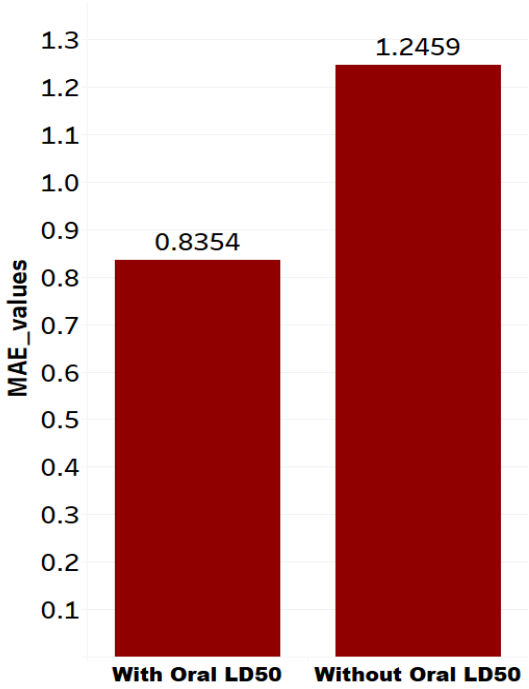
1. LD50 for oral route of exposure
2. Molecular Weight
3. Defined Stereo Atom Count
4. Exact Mass

We can get MAE value **0.8354**

Scatter plot for fitted model is as follow,

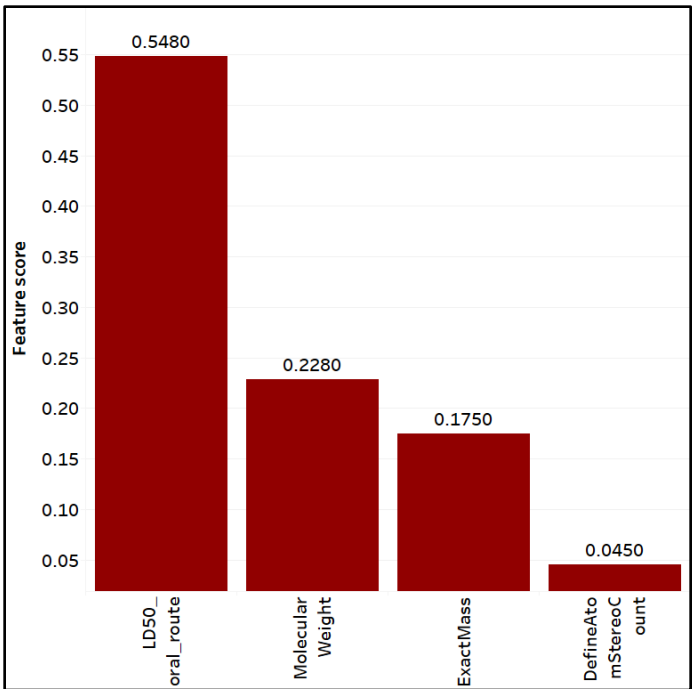


Thus, we can see that the model with oral LD50 values included in it gives lower MAE as co-mpare to the model without oral LD50 values. The comparison between the models is shown b y the bar plot.



Thus, having prior knowledge about the oral LD50 values can help in predicting the LD50 val u-es for dermal more precisely.

Random Forest technique allow us to find out important features for prediction.
[0.548, 0.228, 0.175, 0.045]



Using feature score, most important features is Oral LD50.

CONCLUSIONS

1. From the correlation plot of molecular descriptors and LD50 values we can say that LD50 values are not highly correlated with any of chosen predictors.
2. The Random forest regression model gives us the better prediction results as compared to Neural network and multiple linear regression.
3. The Extremely toxic and Relatively harmless compound are not predicted with good accuracy because of the less data for Extremely toxic compounds and presence of outliers in Relatively harmless compounds
4. Predictors namely ExactMass, DefinedAtomStereoCount and MolecularWeight are the most contributing predictors for the LD50 prediction.
5. LD50 values for Dermal route of exposure is moderately positively correlated with LD50 values of Oral route of exposure.
6. The prior knowledge of the LD50 values for oral route of exposure can improve the predictions of dermal LD50 values.

Source of the dataset:
https://ntp.niehs.nih.gov/iccvam/methods/acutetox/model/trainingset_full_ld50.txt

CODES

Codes for Data Preprocessing and EDA

```
library(xlsx)
library(plyr)
library(dplyr)
library(scales)
library(ggplot
2)
library(MASS)
library(forcats
)
library(webch
em)

##Code to export the data
write.table(data, file =  "for_analysis.csv",  sep  = ",")

setwd("C:\\Users\\soura\\OneDrive\\Desktop\\Toxon_Statis
ta") data = read.csv("Eda.csv", check.names = FALSE)
View(data)

head(data)
str(data)
colSums(is.na(data))

#xl = read.xlsx("g_data_cid_NA_removed.xlsx", 1)
```

```
#data = read.csv("full_final_data.csv", check.names = F)

#View(data)



c = table(data$CASRN) ##Creating Table

df = data.frame(table(c))

View(df)



data1 = aggregate(data$LD50_mgkg, list(data$CASRN), min)

View(data1)



split(data, data$CASRN)

e=lapply(split(data, data$CASRN), function(chunk) chunk[which.min(chunk$LD50_mgkg),])

a = do.call(rbind, lapply(split(data, data$CASRN), function(chunk)
chunk[which.min(chunk$LD50_mgkg),]))

View(a)

colSums(is.na(a))




write.table(a, file = "data1.csv", sep = ",")



categories = cut(a$LD50_mgkg, breaks = c(0,1,50,500,5000,15000,Inf), labels = c("Extremely
toxic", "Highly toxic",                                                    "Moderately toxic",
"Slightly toxic",

                                                                  "Practically nontoxic", "Relatively
harmless"))

categories
```

```
data2 = cbind(a, categories)

View(data2)



#t2=fct_unique(data2$DTXSID)

t3=fct_count(data2$DTXSID) ##Gives the duplicate DTXSID

View(t3)



data$categories =NULL

data2 = read.csv("data_exp.csv", check.names = F)

View(data2)



##############################number of drugs in each class####################



r = data.frame(table(data$categories))

names(r)[names(r) == "Var1"] = "categories" ##Changing the name of a column in r from Var1 to Categories

r



df1 = mutate(data, categories = ordered(categories, levels = c("Relatively harmless", "Practically nontoxic", "Slightly toxic",

                                                               "Moderately toxic", "Highly toxic", "Extremely toxic")),

                     cumulative = cumsum(Freq),

                      label = (percent(Freq/sum(Freq))))

df1



ggplot(data = df1, aes(x =categories, y = Freq)) +
```

```
  geom_bar(aes(fill = categories),stat = "identity" , show.legend = TRUE) +
  coord_flip() +
  geom_label(aes(label = paste(df1$Freq, percent(df1$Freq/sum(df1$Freq)), sep = "\n"),
                 y = -300, fill =categories),
             show.legend = FALSE,
             size = 3, label.padding = unit(0.1, "lines"))

############################BOXPLOT#############################################

#ggplot(data2, aes(x = categories, y = LD50_mgkg, fill = categories)) +
# geom_boxplot(fill = c("green","blue","yellow","forestgreen","purple","pink")) +
# scale_y_continuous(name = "LD50", breaks = seq(0, max(data2$LD50_mgkg), by=5000))

new_data1 = data.frame(categories = data2$categories[data2$categories == "Extremely toxic"],
                       LD50 = data2$LD50_mgkg[data2$categories == "Extremely toxic"])

ggplot(new_data1, aes(x = categories, y = LD50, fill = categories)) +
  geom_boxplot(fill = "aquamarine") +
  scale_y_continuous(breaks = seq(0, 1, by=0.15))+
  theme_bw()+
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())+
  xlab("Categories") + ylab("LD50_mg/kg")

new_data2 = data.frame(categories =data2$categories[data2$categories =="Highly toxic"],
                       LD50=data2$LD50_mgkg[data2$categories == "Highly toxic"])
```

```
ggplot(new_data2, aes(x = categories, y = LD50, fill = categories)) +
  geom_boxplot(fill = "chartreuse2") +
  scale_y_continuous(breaks = seq(0, 50, by=5))+
  theme_bw()+
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())+
    xlab("Categories") + ylab("LD50_mg/kg")



new_data3 = data.frame(categories =data2$categories[data2$categories == "Moderately toxic"],
                       LD50=data2$LD50_mgkg[data2$categories =="Moderately toxic"])


ggplot(new_data3, aes(x = categories, y = LD50, fill = categories)) +
  geom_boxplot(fill = "lightcoral") +
  scale_y_continuous(breaks = seq(50, 500, by = 50))+
  theme_bw()+
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())+
  xlab("Categories") + ylab("LD50_mg/kg")


new_data4 = data.frame(categories = data2$categories[data2$categories =="Slightly toxic"],
                       LD50=data2$LD50_mgkg[data2$categories == "Slightly toxic"])


ggplot(new_data4, aes(x = categories, y= LD50)) +
  geom_boxplot(fill = "slateblue") +
  scale_y_continuous(breaks = seq(500, 5000, by = 500))+
```

```
  theme_bw()+
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())+
  xlab("Categories") + ylab("LD50_mg/kg")

new_data5 = data.frame(categories = data2$categories[data2$categories =="Practically
nontoxic"],
                       LD50=data2$LD50_mgkg[data2$categories == "Practically nontoxic"])

ggplot(new_data5, aes(x = categories, y= LD50)) +
  geom_boxplot(fill = "orange1") +
  scale_y_continuous(breaks = seq(5000, 15000, by = 1200))+
  theme_bw()+
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())+
  xlab("Categories") + ylab("LD50_mg/kg")

new_data6 = data.frame(categories = data2$categories[data2$categories =="Relatively
harmless"],
                       LD50=data2$LD50_mgkg[data2$categories == "Relatively harmless"])

ggplot(new_data6, aes(x = categories, y= LD50)) +
  geom_boxplot(fill = "plum") +
  scale_y_continuous(breaks = seq(15000, max(new_data6$LD50), by = 6000))+
  theme_bw()+
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())+
  xlab("Categories") + ylab("LD50_mg/kg")
```

```
##########################Number of estimates#########################


df = data.frame(table(data2$TEST_TYPE)); df


df2 = mutate(df, test_type = factor(Var1,

levels = c("point estimate", "limit test (greater than)", "limit test (less than)")),

cummulative = cumsum(Freq), label =percent(Freq/sum(Freq)))

df2


ggplot(df2, aes(x= "", y = Freq, fill = test_type)) +

  geom_bar(stat = "identity", width = 1, position = "stack") +

  coord_polar(theta = "y") +

  geom_label(aes(label = label), show.legend = FALSE) + xlab("Frequency") + ylab("Categories")


########################FITTING OF DISTRIBUTION############################


qqnorm(data$LD50_mgkg)


data3 = mutate(data2, scale_LD50_mgkg = (data2$LD50_mgkg/sd(data2$LD50_mgkg)))
##Adds another column of scaled LD50

                                                    ##values

View(data3)


ggplot(data, aes(x = LD50_mgkg)) +
```

```
  geom_histogram(bins = 20, color = "darkblue", fill = "lightblue") +
  geom_freqpoly(bins = 20, color = "red")

ggplot(data2, aes(x = log(LD50_mgkg))) +
  geom_histogram(bins = 20, color = "darkblue", fill = "lightblue") +
  geom_freqpoly(bins = 20, color = "red")

plot(density(log(data3$scale_LD50_mgkg)))

ggplot(data3, aes(log(scale_LD50_mgkg))) + geom_density()
shapiro.test(data2$LD50_mgkg)

install.packages("nortest")
library(nortest)
ad.test(l)

l = log(data2$LD50_mgkg)
max(l); min(l)

fw <-  fitdistr(data3$scale_LD50_mgkg,  "weibull")
fg <- fitdistr(data3$scale_LD50_mgkg, "gamma") fe
<- fitdistr(data3$scale_LD50_mgkg,  "exp")

ggplot(data = data3, aes(x=data3$scale_LD50_mgkg)) + stat_ecdf(geom = "step")
```

```
exp=rexp(8994, rate = 1.5998)

we=rweibull(8994, shape =0.6865038, scale = 0.4842841 )


#ggplot(data = data3) +

  #stat_ecdf(aes(x=data3$scale_LD50_mgkg), geom = "step", color="green",size=1.5) +

  #stat_ecdf(aes(x=exp), geom = "step", color="blue", size=1.4) +

  #stat_ecdf(aes(x=we), geom = "step", color="red", size=1.4)


ks.test(log(data2$LD50_mgkg),"pweibull", scale=0.4842841, shape=0.6865038 )


ks.test(data3$scale_LD50_mgkg,"pweibull", scale=fw$estimate[2], shape=fw$estimate[1] )


####################VARIANCE############


s= data.frame(table(data$CASRN));s ##Obtains the number of compounds in each group

lenth_group = s$Freq

data4 = cbind(data2, lenth_group)

data4


grp = group_by(data, CASRN)

w = summarise(grp, variance = var(LD50_mgkg)) ## w is dataset that contains variance of
each group


spread = c() ##A vetor that will contain the variance of each group

            ##and 0 for the group that has length 1
```

```
for (i in  1:length(data4$lenth_group)){
  if (data4$lenth_group[i] != 1){
    spread = append(spread, w$variance[i])
  }
  else {spread = append(spread, 0)}
}
```

```
data4$SD = sqrt(spread) ##Spread id the column of variance of ld50 and here we are
calculating standard deviation
```

```
data_mean_ld50 = aggregate(data$LD50_mgkg, list(data$CASRN), mean)
names(data_mean_ld50)<-c("CASRN", "mean_ld50") ##to change the columns names
```

Codes for modelling

Importing required libraries for analysis

```
import pandas as pd # Required for data handling
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
! pip install -U scikit-learn
fromsklearn.impute import KNNImputer
fromsklearn.metrics import accuracy_score
fromsklearn.metrics import mean_squared_error
fromsklearn.model_selection import train_test_split # Required for splitting data into training and test
set
```

```
fromsklearn.preprocessing import StandardScaler #TO standardize data

fromsklearn.ensemble import RandomForestRegressor # Required to build random forest

fromsklearn.model_selection import RandomizedSearchCV

import keras
```

Import data

```
df=pd.read_csv(r'C:\\Users\\dell\\Desktop\\Toxon Statista\\full_final_data.csv')


df=pd.read_csv(r'C:\\Users\\dell\\Desktop\\Toxon Statista\\Imputed_data.csv') ##This is imputed data (KNN imputer)


df =df.drop(columns = ['CASRN', 'DTXSID', 'CHEMICAL_NAME', 'TEST_TYPE', 'categories', 'MolecularFormula', 'CID', 'CanonicalSMILES', 'IsomericSMILES', 'InChI', 'InChIKey', 'IUPACName', 'Fingerprint2D'])
```

Imputation of missing values using kNN imputer

```
nan = np.nan

imputer = KNNImputer(n_neighbors=5, weights="uniform")

df = pd.DataFrame(imputer.fit_transform(df))
```

Creating separate data of dependent and independent variable

```
 Create_Separate _dataframe_consisting_of_only_dependent_variable

y = pd.DataFrame(df.iloc[:, 0])


Create_Separate_dataframe_consisting_of_only_independent_variable

x = df.drop(columns = ["LD50_mgkg"], inplace = False, axis = 1)
```

Splitting the data into training set and test set

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 1)
```

Standardizaton of vaiables

```
standardizing_y_variable

#sc_x = StandardScaler()

sc_y = StandardScaler()


y_train = sc_y.fit_transform(y_train)

y_test= sc_y.fit_transform(y_test)


y_test = pd.DataFrame(y_test)

y_test = y_test.rename(columns = {0 :"Actual_Values"})


#x_train = sc_x.fit_transform(x_train)

#x_test = sc_x.fit_transform(x_test)
```

Fitting the regression model to the data

```
regressor = RandomForestRegressor(n_estimators = 700,max_depth=50, min_samples_leaf = 2,
min_samples_split= 2, max_features="sqrt", random_state =  42)

regressor.fit(x_train, y_train)
```

Predicting a new result with regression

```
y_pred = pd.DataFrame(regressor.predict(x_test))

y_pred = y_pred.rename(columns={0:"Predicted_values"})
```

mean_squre_error

```
mse=mean_squared_error(y_test, y_pred)

rmse=np.sqrt(mse); rmse
```

Feature_Importance_Graph

```
feat_importances = pd.Series(regressor.feature_importances_, index=df.columns[1:34])

feat_importances.nlargest(10).plot(kind = 'barh', color ={"coral", "yellowgreen",
"lightsteelblue","mediumpurple","crimson", "gold", "forestgreen", "turquoise","orchid", "darkorange"})
```

Plot of Predicted vs Actual

```
df1 = pd.DataFrame({'Actual': y_test["Actual_Values"], 'Predicted': y_pred["Predicted_values"]})

df1 = df1.head(25)


df1.plot(kind='bar',figsize=(10,8))

plt.grid(which='major',  linestyle='-',  linewidth='0.5',  color='green')

plt.grid(which='minor',  linestyle=':',  linewidth='0.5',  color='black')

plt.show()
```

Hypertuning and cross validation

```
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1000, num = 10)]

max_features = ['auto', 'sqrt']

max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]

max_depth.append(None)

min_samples_split = [2, 5, 10]

min_samples_leaf = [1, 2, 4]
```

```
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}


rf = RandomForestRegressor()
rf_random = RandomizedSearchCV(estimator = rf, param_distributions =
    random_grid, n_iter = 100, cv = 5, verbose=2, random_state=42)
```

Neural Network codes

```
    import statistics
    import pandas as pd
    import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from scipy import stats
    from sklearn.preprocessing import StandardScaler
    from sklearn.ensemble import RandomForestRegressor
    import theano
    import keras
    from keras.models import Sequential
    from keras.layers import Dense, Dropout, Activation, LeakyReLU
    from keras.activations import relu, sigmoid
```

```
    from keras.optimizers import RMSprop
    from keras.wrappers.scikit_learn import KerasRegressor
    from sklearn.model_selection import cross_val_score, GridSearchCV
    from sklearn.metrics import mean_squared_error
    from sklearn.metrics import mean_absolute_error
    from collections import defaultdict
    from sklearn.inspection import permutation_importance
    from scipy.cluster import hierarchy
from collections import defaultdict
##NEURAL NETWORK
    def model():
        regressor = Sequential()
        regressor.add(Dense(output_dim = 15, init = "uniform", activation= "elu", input_dim =
    33))
        regressor.add(Dropout(0.1))
        regressor.add(Dense(output_dim = 15, init = "uniform", activation = "elu"))
        regressor.add(Dropout(0.2))
        regressor.add(Dense(output_dim = 1, init = "he_uniform", activation = "linear"))
        regressor.compile(optimizer = "adam", loss = "mean_absolute_error")
        return regressor


    reg = KerasRegressor(build_fn = model, epochs = 250, batch_size = 10)
    reg.fit(x_train, y_train)
    y_pred = reg.predict(x_test)
```

```
mean_absolute_error(y_test_transformed, y_pred)

#train_mae = 4.1484 test_mae = 4.155212161098474



Neural_network_ka_data_without_tuning = pd.concat([y_test_transformed,
pd.DataFrame(y_pred)], axis =1)

Neural_network_ka_data_without_tuning.columns = ["y_test_transformed", "y_pred"]

Neural_network_ka_data_without_tuning.to_csv("Neural_network_without_tuning.csv")




def model():

    regressor = Sequential()

    regressor.add(Dense(output_dim = 17, init = "he_uniform", activation= "elu", input_dim
= 33))

    regressor.add(Dropout(0.1))

    regressor.add(Dense(output_dim = 17, init = "he_uniform", activation = "elu"))

    regressor.add(Dropout(0.2))

    regressor.add(Dense(output_dim = 15, init = "he_uniform", activation = "elu"))

    regressor.add(Dropout(0.1))

    regressor.add(Dense(output_dim = 1, init = "he_uniform", activation = "linear"))

    regressor.compile(optimizer = "RMSprop", loss = "mean_absolute_error")

    return regressor


reg = KerasRegressor(build_fn = model, epochs = 250, batch_size = 10)

reg.fit(x_train, y_train)
```

```
    y_pred = reg.predict(x_test)

    mean_absolute_error(y_test_transformed, y_pred)

    ##train_mae = 4.0952, test_mae = 4.061301823173346


    Neural_network_ka_data_with_tuning = pd.concat([y_test_transformed,
    pd.DataFrame(y_pred)], axis =1)

    Neural_network_ka_data_with_tuning.columns = ["y_test_transformed", "y_pred"]

Neural_network_ka_data_with_tuning.to_csv("Neural_network_with_tuning.csv")
```

Hyper parameter tuning

```
    def create_model(layers, activation, optimizer, initializer):

        model = Sequential()

        for i, nodes in enumerate(layers):

            if i==0:

                model.add(Dense(nodes, input_dim = 33))

                model.add(Activation(activation))

                model.add(Dropout(0.2))

            else:

                model.add(Dense(nodes))

                model.add(Activation(activation))

                model.add(Dropout(0.2))

        model.add(Dense(units = 1, kernel_initializer= 'glorot_uniform', activation= "linear"))

        model.compile(loss = mean_absolute_error, optimizer = optimizer)

        return model
```

```
        layers = [(17, 17), (17, 17, 17), (17, 17, 15)]
        opti = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam']
        activ = ['relu', 'tanh', 'elu', 'linear']
        initial = ['uniform', 'glorot_normal', 'glorot_uniform', 'he_normal', 'he_uniform']

        param_grid = dict(layers = layers, activation=activations, optimizer = optimizers,
                initializer = initializers)
        grid = GridSearchCV(estimator = model, param_grid = param_grid, cv = 5)
            grid_result = grid.fit(x_train, y_train)
 #######algorithm_comparison_code###########
data = {'Random forest':3.86, 'Neural network': 4.06, 'MLR':8.31}
Algorithms = list(data.keys())
MAE_values = list(data.values())

fig = plt.figure(figsize = (10, 5))
plt.bar(Algorithms, MAE_values, color ='maroon',
        width = 0.4)
plt.xlabel("Algorithms_used")
plt.ylabel("MAE_values")
plt.title("Comparison between the algorithms")
plt.show()

#objective 3#
```

```
df =  pd.read_csv(r'C:\Users\DELL\Desktop\project\Analysis\Objective  3\obj_3_reg.c')df.hea
d()

#simple correlation matrix

corrmat = df1.corr()

  print(corrmat)

  f, ax = plt.subplots(figsize =(9, 8))

  plt.rcParams['font.size']=15

  sns.heatmap(corrmat, ax = ax, cmap ="coolwarm", linewidths = 0.1, annot=True, annot_kw
  s={'size':20})

      cg = sns.clustermap(corrmat, cmap ="coolwarm", linewidths = 0.1, annot=True);

      plt.setp(cg.ax_heatmap.yaxis.get_majorticklabels(), rotation = 0)

from scipy.stats import  boxcox

trn = boxcox(df['Dermal'], lmbda=0.18)

df['boxcox_dermal'] = trn.transpose()

  from scipy.stats import shapiro

  stat, p = shapiro(df.boxcox_dermal)

  print('Statistics=%.3f, p=%.3f' % (stat, p))

  Out: Statistics=0.911, p=0.000

      from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor

      model = RandomForestRegressor()

      reg = RandomForestRegressor(max_depth=50, max_features=5)

      reg.fit(X_train, y_train)

    reg.feature_importances_
```

Objective 2

```
regressor = RandomForestRegressor(n_estimators= 800,min_samples_split=
2,min_samples_leaf= 1,max_features= 'auto',max_depth= 50)

regressor.fit(x_train, y_train)

y_pred = regressor.predict(x_test)

mean_absolute_error(y_test_transformed, y_pred)

fig, ax1 = plt.subplots(figsize=(10, 8))

corr = stats.spearmanr(x).correlation

corr_linkage = hierarchy.linkage(corr, method= 'complete')

dendro = hierarchy.dendrogram(corr_linkage, labels= list(x.columns))

dendro_idx = np.arange(0, len(dendro['ivl']))

ax1.set_xticklabels(list(x.columns), rotation='vertical', fontsize=10)

cluster_output = pd.DataFrame({'Predictors':x.columns.tolist() , 'Cluster':cluster_ids})

cluster_output.nsmallest(33, "Cluster")

cluster_ids = hierarchy.fcluster(corr_linkage, 1, criterion='distance')

cluster_id_to_feature_ids = defaultdict(list)

for idx, cluster_id in enumerate(cluster_ids):

    cluster_id_to_feature_ids[cluster_id].append(idx)

selected_features = [v[0] for v in cluster_id_to_feature_ids.values()]

selected_features

x_train_sel = x_train.iloc[:, selected_features]

x_test_sel = x_test.iloc[:, selected_features]

regressor = RandomForestRegressor(n_estimators= 800,min_samples_split=
2,min_samples_leaf= 1,max_features= 'auto',max_depth= 50)
```

```
regressor.fit(x_train_sel, y_train)

y_pred = regressor.predict(x_test_sel)

mean_absolute_error(y_test_transformed, y_pred)

print("Accuracy on test data: {:.2f}".format(mean_absolute_error(y_test_transformed,
y_pred)))

result = permutation_importance(regressor, x_test_sel, y_test_transformed, n_repeats=5,
random_state=42)

x_columns = pd.DataFrame(x.columns)

features_importance = pd.concat([x_columns, pd.DataFrame(result.importances_mean)],
axis = 1)

features_importance.columns = ["Features", "Score"]

features_importance = features_importance.nlargest(33, "Score")

features_importance

features_importance = features_importance.nsmallest(33, "Score")



sorted_idx =result.importances_mean.argsort()

y_ticks = np.arange(0, len(x_train_sel.columns))

fig, ax = plt.subplots(figsize=(9,7))

ax.barh(features_importance.Features, features_importance.Score)

#ax.set_yticklabels(x_test_sel.columns[sorted_idx])

#ax.set_yticks(y_ticks)

ax.set_title("Permutation importance")

ax.set_xlabel("Increase in MAE")

ax.set_ylabel("Predictors")
```

```
fig.tight_layout()

plt.show()
```