Classification of Healthcare Data

Miguel Angelo Rosales

1 Summary of Features

1.1 Overall Distribution of Target Variable

When loading the 'health_train.csv' file and producing a countplot of the overall distribution of the target variable, it presents that a majority of the data includes 'Low risk' participants (1235), followed by 'Moderate risk' (219) and 'High risk' (130). This is represented by Figure 1 in A. The distribution can also be visualised in Table 1.1 below:

	Low risk	Moderate risk	High risk
Count	1235	219	130
Percentage (%)	78.0	13.9	8.1

Table 1: Distribution of Target Variable in Count and %

Using pandas in python, we can divide the data into their categorical columns and numerical columns, excluding the target variable. From this, we can perform summary statistics and visualise their distribution based on the target variable.

1.2 Categorical Columns

Using the '.info()' function in pandas, we can determine which columns are considered categorical on the 'health_train.csv', as they're categorised by the type: Object. This determines that columns 'x3' (gender) and 'x14' (blood type) are categorical. Performing a summary statistic analysis on the categorical columns, using the '.describe()' function, allows us to numerically visualise the data. Adding rows which describes the missing and unique values have also been inputted. This is shown below in Table 1.2:

	x3	x14
Count	1584	1584
Unique	2	8
Top	M	O+
Freq	842	410
Missing	0	0
Values	[F,M]	[O+, A+, B+, O-, A-, B-, AB-, AB+]

Table 2: Summary Statistics of the Categorical Columns

From this, we can see there are no missing values from either categories. Additionally, there are more male participants than females in the study and there are more patients with blood type O+ compared to the rest of the values. Using this data, we can then visualise the distribution of the categorical data sets in accordance with the target variable.

A countplot was generated to visualise the distribution of 'x3' with the target variable, which is represented by Figure 2 in A. As previously mentioned there are more Males than Females overall. However, in the Male category in 'x3', there are more 'High risk' participants than 'Moderate risk', deviating from the original trend from Figure 1. Potentially, this data represents that Males are more high risk, compared to their Female counterparts. Nevertheless, the Female category does follow this initial trend.

Looking at the countplot describing the distribution of Blood Type with the target variable shown by Figure 3, there are more participants with A+ that are of ‘High risk’ compared to the rest of the values. Furthermore, the distribution of blood type AB- and AB+ are low, suggesting that this blood type is rare.

It would also be interesting to see the combination of both variables and see their distribution across the target variable. Figure 4 in A represents the grouped columns of ‘x3’ and ‘x14’ and how they vary. From this, it can be deduced that those who are ‘M’ and have ‘O+’ are the most popular participants in the ‘Low risk’ category in the target variable, whereas ‘M’ and ‘O-’ are the least popular. Moreover, there are no combination of (M, AB-) and (M, AB+), suggesting that only ‘F’ participants can only have ‘AB-’ and ‘AB+’. Moreover, when looking at the ‘High risk’ category, the (M, A+) are more popular compared to (F, AB-)

1.3 Numerical Columns

Similar to the categorical columns, we can perform a summary statistic analysis on the numerical columns identified earlier, as well as produce a KDE plot to show the distribution of each value represented by their respective column. Multiple tables have been generated to show the summary statistics of all the numerical columns in the ‘health_train.csv’, shown by Table 12, 13, 14, 15, 16, 17, all in section A.

In addition to the numerical columns summary statistics, each column distribution can be visualised via a KDE plot with respect to the target variable. This is represented by Figures 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24 and 25, in A

1.4 Missing Data

In section 1.2, we have determined that there were no missing data within the categorical columns in the ‘health_train.csv’. Therefore, it is important to determine if there are any in the numerical columns. To do this we can use the ‘isnull().sum()’ function on the health train data to determine the total count of missing values per column within the numerical columns, from this, it can be assigned to a variable called ‘missing_values_count’. Furthermore, using the ‘.shape’ function will retrieve the total cells in the train data, which can be used to determine the percentage of total missing values. This can be assigned to a variable called ‘total_cells’. Afterwards, to retrieve the number of total missing values, we can create a variable called ‘total_missing’ values, which can be achieved by calling the ‘.sum()’ function on the ‘missing_values_count’ variable. Therefore, by dividing the ‘total_missing’ with ‘total_count’, and multiplying this value by 100, will give us the percentage of total missing values.

Column	Missing Values
x5	44
x8	17
x10	27

Table 3: Missing Values per Column

By calling the ‘missing_values_count’ variable, we can visualise the number of missing values per column, which is represented by Table 3 above. Columns with 0 missing values have been excluded from the table, thus only showing columns which have missing values. From this, it can be deduced that columns ‘x5’ has 44 missing values, whereas ‘x8’ has 17 and ‘x10’ has 27. The rest of the columns don’t have any. In total there are 88 missing data, meaning there is a 0.21% of missing data in the whole health train data.

1.5 Outliers in individual fields

To visualise any potential outliers in the whole train data set, it is important to generate a Boxplot. This type of plot can easily detect outliers by pinpointing data points that appear outside of the whiskers. In addition to detecting outliers, it can also represent the data of each column. For example, it can tell us if the data is skewed, symmetrical and how tightly the data is grouped (Agarwal, 2019). Looking at Figure 26 in section A, we can visualise the number of detected outliers in each column. Columns with no outliers have been excluded from the the following visualisation. In addition to the detection of outliers, columns 'x4', 'x5', 'x7', 'x8', 'x9', 'x11' and 'x19', are closely grouped together with almost negligible values, whereas the other columns have a wide range of data points, with outliers appearing above and below the whiskers.

2 Data Pre-processing

2.1 Handling Missing Values

From section 1.4, we have determined that the numerical columns 'x5', 'x8' and 'x10' have missing data. Therefore, we can analyse these columns via a summary statistical analysis to determine how to handle the missing values through either dropping/imputation. This data has been tabulated in Table 4 below.

	x5	x8	x10
count	1540.0	1567.0	1557.0
mean	0.009906493506493494	2.5526483726866626e-06	47.09441233140655
std	0.04862691996951116	5.047532309658415e-05	17.269620587495357
min	0.0	0.0	13.0
max	0.477	0.001	86.0
missing	44.0	17.0	27.0
median	0.0	0.0	49.0

Table 4: Summary statistic analysis of columns with missing data

Looking at column 'x5', we can see that a it has a very low standard deviation, suggesting that a majority of the data points are situated around the mean value. Additionally, looking at the KDE plot of 'x5' shown by Figure 8, we can see that a majority of the data points are distributed between -0.1 and 0.1, with large peaks representing 0.0. Therefore, we can impute the missing values by replacing them with the median (0.0). Similarly, with column 'x8', we can see that it has a very low standard deviation meaning that it also has a majority of data points situated around the mean value. This is further supported by Figure 11, where there is a large distribution of data points with a value of 0.0. Therefore, we can also replace all the missing values with 0.0. However, looking at column 'x10' we can see a large distribution of values from the standard deviation, which ranges from values 13-86 (min-max), with a median of 49. Additionally, looking at the KDE plot shown by Figure 15, we can see that there is no clear pattern on how the data is distributed for the 'Low risk' factor in the target variable, however, for the 'Moderate risk' and 'High risk' factors, values are mostly placed within the range of 55-80. Deleting the rows containing the missing values might remove useful information contained in the 'health_train.csv' data, therefore, the best way would be to impute this data. For example, a viable solution to impute the numeric column 'x10' would be to use the KNNImputer model. This is an imputation method that imputes values based on distance. The reason why this method was chosen, is because it attempts to normalise the data and since there is a wide distribution of values (Htoon, 2020), it will handle missing values by filling them with data that is similar.

2.2 Outlier Detection Methods & Masking

During the pre-processing stage, it is also important to detect any outliers and remove the rows which have been identified via outlier detection methods. For the chosen outlier detection method that I have incorporated on the 'health_train.csv' data, is DBSCAN. DBSCAN is an unsupervised method that aids with the identification of clusters (Sander et al., 1998). This method provides decipherable results, as it assigns each data point to a cluster or an outlier (if it appears outside this cluster). Essentially, it separates regions of high density with low density regions, which can be effective in both dense and scattered data (Tan et al., 2013), which the health train data represents within the numerical columns, where the distribution of each column varies in dispersion. To properly apply DBSCAN, the *eps* and *minPts* need to be determined.

Eps relates to the radius of the cluster from a given point and determines how many points are included in the cluster, whereas the *minPts* relates to the minimum number of points within the cluster (Dunham, 2003). Since the data set has a lot of numerical features, it can be assumed that the *minPts* should also be large. To accomodate for this, I have set the value to 15. To determine the *eps* parameter, would be to incorporate a technique that calculates the distance between each value, using the k-nearest neighbour value ($k = minPts$). The calculated distances are then plotted on a graph representing the k-distance in ascending order. The optimal value for *eps* can then be identified through the value of maximum curvature (Rahmah and Sitanggang, 2016). The following graph in Figure 27 in section **??** represents the k-distance graph for *eps* determination. As shown by the graph, the point of maximum curvature is located at the value of 55 for distance.

Now, applying DBSCAN using the defined parameters, it shows that the number of outliers identified is equal to 236. We can then mask these outliers which produces a new data frame with a shape of (1348, 22), which can be used to sample the data which does not introduces any biases, as the data has now been cleaned from missing values and outliers.

2.3 Correlation Analysis

Now that we have filled in the missing values and masked the outliers, it would be beneficial to now visualise the numerical columns and how they relate to each other. One method would be to apply a correlation analysis and pair each column to determine the correlation coefficient. Values that are close to 1.0 are positively correlated, whereas values that are closer to -1.0 are negatively correlated to each other. Values with 0 are not correlated at all. Using the '.corr()' function, we can calculate the correlation coefficient for each respective pair and visualise them using a Heatmap. This can be shown by Figure 28 in section A. As shown by the Heatmap, it can be deduced that the darker hue is representative of a highly correlated pair of variables and those with a lighter hue are negatively correlated.

To accurately determine which columns are highly correlated numerically, we can look at Variance Inflation Factor values (Kaplan, 2022). We can find Multicollinearity in this data set. Columns with high VIF make it difficult to determine their effects on the target variable. By importing the necessary modules to perform VIF calculations, we can tabulate the VIF scores for each column which is represented by 18 in section A. As shown by the table, columns 'x2', 'x20', 'x21' and 'x22' represent extremely high VIF scores compared to the rest of the columns. We can save these columns to a variable that represents the highly correlated variables, which can be used during feature selection in Part 3.

2.4 Encoding

For classification tasks it is necessary to encode the categorical columns so that it can be fitted into the desired model. A few methods have been applied to the data.

2.4.1 One-Hot Encoding

For the 'x3' column, we only have two unique values within the categorical column. Furthermore, there is no ordinal relationship with the values, with no dependency on the target variable. Therefore, One-Hot Encoding has been applied to this column, to quickly identify if the values are either 'M' or 'F', as both are equally as important. Through this encoding technique, each value in the categorical column becomes it's own binary column, where the presence of either value is represented '1' and if it's not, then it is represented by '0'. This views either value as a distinct entity, preventing it from being misinterpreted during machine learning algorithms within classification. Table 5, shows the head of the encoded 'x3' column via One-Hot Encoding.

S_F	S_M
1.0	0.0
0.0	1.0
1.0	0.0
1.0	0.0
1.0	0.0

Table 5: Head of encoded 'x3' column through One-Hot Encoding

2.4.2 Target Encoding

Since there are multiple values in the 'x14' categorical column, the occurrences of these values may depend on the target variable. For example, we have determined from Graph 3 in section A, that the distribution of 'x14' varies for each target variable. This encodes the following column in a way that represents the relationship withe the target. Target encoding decreases the number of variables within this column, while maintaining the relationship with the target to produce a more compact representation. Due to this relationship, it would be useful to see if there is an association through classification tasks. Table 6, represents the head of the encoded 'x14' column via target encoding.

B_tar
1.052910052910050
1.052910052910050
1.052910052910050
1.6814771481521200
1.1976284585022900

Table 6: Head of encoded 'x3' column through One-Hot Encoding

After encoding both categorical column, the original columns would now be dropped from the data set to avoid any issues later on during the classification stage.

2.5 Feature Selection

After encoding the categorical columns in the 'health_train.csv' file, it would be also important to select important features which are relevant, to improve the model's accuracy and generalising ability.

2.5.1 ANOVA

The ANOVA, univariate feature selection method is a useful approach for continuous columns in the data file. By determining the mean features, this method can differentiate which columns have a significant impact on the target variable. For example, by determining the F-statistic scores, it has the ability to rank which features are most important in correspondence to the target variable. Table 19 in section A shows these scores in order of F-statics values.

Columns with higher the F-statistic values, are regarded as most important, which have an increased likeliness to contribute to the target variable. By setting a threshold value of 100, columns with F-statistic values equal to or above this are considered to be the most important and will be included within the classification model. Therefore, the following columns are to be utilised during classification: 'x4', 'x9', 'x10', 'x12', 'x20', 'x21', 'x22' and 'B_tar'.

2.5.2 Kendall's

Another feature selection method that can be utilised is the Kendall's method. This approach identifies features of importance through correlation of the respective feature with the target variable. It is able to measure the discordance and concordance relationship and ranks them in order. This method does not assume any specific distributions of the data, which is useful in our case as the KDE plots for the numerical columns shown in section 1.3, represents a widely distributed data for each target value for their respective column. Table 20 in section A shows the Kendall's correlation coefficient for each column.

Used in conjunction with the ANOVA method, we can see there are some overlapping columns. By setting a threshold value for greater than 0.25 and lesser than -0.25. We can further select additional columns that can be included during classification. These include the additional column: 'x23'.

Now that we have gathered the most relevant columns, it now time to sample.

2.6 Sampling

Once the data has been cleaned, the next step would be to divide the data into train/validation/test partitions, using stratified. After the first sampling step, we can divide the data into a test partition which would be used to test our model after building the model and adjusting the parameters. Another partition is made from the initial sampling step, which can be sampled again to divide the data into a train and validation partition. The train partition would be used to build our model and the validation partition would be used to adjust the parameters of this model.

2.7 Balancing the Data

The next step after sampling, would be to balance our data set. This is so that it removes any imbalances which have the potential to cause bias model performance during the classification task. There are several methods to balance the data via up-sampling or down-sampling. After these tasks, the data should still be represent the original distribution of the original data set before balancing. Since there are three target categories, it would be useful to see the distribution of the data after sampling. The following data has been formulated and can be visualised by Table 7.

	target
Low risk	701
Moderate risk	109
High risk	52

Table 7: Distribution of Target in Train Partition after Sampling

We can either up-sample the other target categories to the majority class ('Low risk') or vice versa via down-sampling to the minority class ('High risk')

2.7.1 Upsampling

Upsampling aims to mitigate any class imbalances by increasing the minority classes so that it equals the majority class. This was conducted on the data by increasing the following target values: 'Moderate risk' and 'High risk', so that it equals the count of the 'Low risk' majority class.

2.7.2 Downsampling

Alternatively, down sampling reduces the count of the majority classes so that it equals the minorty classes. In this case, both 'Low risk' and 'Moderate risk' were reduced to eqaul the count of the 'High risk' target variable.

2.7.3 Choosing the balanced data

After deliberation, the best method out of the two was to upsample my data. This is because, no information is lost from all the target values. Instead, it offers the classification model to capture strong sense of the minority class in terms of trends and characteristics. This reasoning allows the model to have an improved performance to classify the target variable. Additionally, rather than randomly inputting extra data from the minority classes, instead it replicates existing instances, which does not lead to over-fitting.

Additionally, looking at the distribution of the columns, the selected features seem to be relatively similar. For example graphs 29 and 30 in section A, which represent the distribution of values for 'x21' column before and after upsampling, respectively, represents that the data is still distributed equally. Therefore, we can then apply this to classification stage.

3 Supervised Model Training

3.1 Choosing the best model

The following models were chosen to fit with the training data: KNN, Linear SVC, Linear SGD, NaiveBayes, Decision Tree and Random Forest. By training our models using the train partition, we can determine the accuracy scores for each classifier and choose the best one to test our model.

By compiling the scores together and through direct comparison of the accuracy of each model, the Random Forest Classifier was the most accurate with a score of 94.4%. Therefore, we can then utilise this model to determine the parameters using our validation partition.

3.2 Hyper-parameter Optimisation

The Random Forest Classifier has the following parameters: n_estimators, max_features, max_depth, min_samples_split and min_samples_leaf. Using a cross validation method we can determine the best parameter configuration to apply to the Random Forest Classifier on our test file for predictions.

Using the GridSearchCV() function, we can input a range of values for each parameter listed above. This then tests each combination on the Random Forest Classifier, to optimise it's performance, by testing it on the validation partition. Through this method, the following parameter configuration was determined to be the most optimal:

Parameters	Configuration
n_estimators	50
max_features	sqrt
max_depth	5
min_samples_split	2
min_samples_leaf	3

Table 8: Optimal Parameters for Random Forest Classifier

Additionally, we can also visualise the feature importance scores, which ranks the important features using a bar plot. This is visualised by graph 31 in section A. Through this, we can determine that 'x11' is the most important feature and must be included, while the columns with lower scores are not as important. However, through analyses, it has been determined that all columns must be included during prediction, as removal of columns with lower feature importance scores, resulted in a decreased performance.

3.3 Performance Validation

After optimising the classification model, we can test it's performance on the validation partition to determine the following metrics: Precision, Recall and F1-Score.

The following performance has been tabulated in Table 9 below:

	precision	recall	f1-score	support
High risk	1.00	0.93	0.97	15
Low risk	0.97	1.00	1.00	169
Moderate risk	0.96	0.84	0.90	32

Table 9: Performance of the model on the validation partition

As shown by the table, the precision score which determines the ratio of true positives for all target values are and close to 1.00, indicating that fewer false positives were detected. Additionally, for recall, which correctly determined positive instances show that these values are also close to 1, with the exception of the 'Moderate risk' class being below 0.90. Nevertheless, this shows that there is fewer false negative predictions. Finally, the F1-Score indicates the average value of recall and precision. With all the target values being equal to or above 0.90, this shows that the Random Forest Classifier, with the determined parameters is the optimal classification model to predict classes for the test data.

This can be further represented by the ROC curve plot shown by graph 32 in section A. By interpreting the graph, we can see for each class, the AUC scores for classes 1 and 2, are 0.99 and 1 for class 0. This shows that it can accurately predict classes.

Furthermore, this can be further depicted by the confusion matrix shown by graph (x) in section A. The plot shows that there is a high true positive rate for classes 0 and 1. However, class 2 ('Moderate risk) is less accurate, which correlates with the low measuring metrics in comparison to the other classes, shown by table 9.

Through extensive analyses of the performance of the model, we can then predict the classes of the test data using the Random Forest classifier. The predicted classes can be found by accessing the 'predictedTarget.csv' file and the head of the file, showing the predicted labels are shown by the following table:

id	Predicted Target
PA3001	Moderate risk
PA3002	Low risk
PA3003	Low risk
PA3004	Moderate risk
PA3005	Low risk

Table 10: Head of predicted labels for the test file

4 Unsupervised Learning

4.1 Pre-processing

4.1.1 Encoding

Similar to the supervised learning section, it would be beneficial to encode the categorical columns to ensure that the clustering algorithms are able to interpret the data. However, unlike the supervised section, the data does not include the target variable so encoding the 'x14' column would have to be done through one-hot encoding. This produces the binary columns shown by table 11, which would be concatenated to the main data frame. The same encoding method for the supervised learning section which transforms the 'x3' columns into it's binary columns has also been applied through one-hot encoding.

B_A+	B_A-	B_AB+	B_AB-	B_B+	B_B-	B_O+	B_O-
0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 11: Binary columns of 'x14' through One-Hot Encoding

4.1.2 Correlation Analysis

By determining the correlation coefficient of the data, we can select which features to exclude based on their dependency towards each other. This is so we can avoid multicollinearity, which can lead to unstable modelling. Graph 33 in section A, represents a heatmap which shows which columns are highly correlated. By setting a threshold value of 0.80, we can exclude and features which exceed this value. The following features have then been excluded: 'x2', 'x20', 'x21', 'x22' and 'S_F'.

4.1.3 Dimension Reduction

Since our data is still very large, it would be beneficial to apply dimension reduction techniques such as PCA to ensure that valuable data or trends are still represented but in the form of principal components. This helps extracts the most useful columns within the data.

To determine the n_component parameter for PCA, we can plot a scree plot to determine number of components to include during unsupervised learning. The point at which the graph levels off determines optimal value for the n_component parameter. This is shown by graph 34 in section A. As shown by the graph, the graph levels off when the number of components reaches 5. Therefore, this would be the value for the n_component parameter. The data is then transformed.

4.2 K-Means

4.2.1 Determining Number of Clusters

There are several methods to apply to determine the n_clusters parameter for the K-Means clustering algorithm. For example we can apply the elbow method. This is shown by graph 35 in section A. The point at which the elbow is determined is when the number of clusters is equal to 2.

Another method to validate this approach would be to incorporate the silhouette method. By plotting a graph showing the silhouette scores, against the number of clusters, we can determine the the n_clusters parameter by identifying the highest score. Graph 36 in secation A, shows that the highest score is equal to 0.62, representing the number of clusters which is also equal to 2. Looking at the silhouette score for this model (0.62), indicates that the separation of clusters is moderate, but not the best. Additionally, it also represents a high interia score of 158182077, indicating a dispersed form of clusters for the K-Means algorithm.

4.2.2 Visualising Clusters

Visualising the cluster formation using the K-Means algorithm is shown by graph 37 in section A. As shown by the graph, there is a clear indication of two clusters being formed from the data. The formation shows that there are two classes that can be identified. Those of one class indicated by the colour yellow, shows that they seem to have low 'x23' values but their 'x10' values range from both sides of the spectrum. However, the class indicated by the colour blue, shows that they have a higher 'x23' value but within certain ranges of the 'x10' column. For example, those who belong to this class they seem to only have 'x10' values that are between 20-40 and 50-60.

5 Conclusions

Overall, through data exploration and various pre-processing techniques, my findings have indicated that the best performing model for Supervised Learning is the Random Forest Classifier. This has shown to have really high measuring metrics, during model performance on the validation partition. This has lead to the predictions of different classes within the test the test data

For unsupervised learning, the data has been pre-processed so that it can fit into the clustering algorithms. The K-Means has been chosen due to it's large scalability and identify groups, which have not been labeled within the data. Through determination of the number of clusters via elbow and silhouette method, it was found that the optimal value for the number of clusters is equal to 2. Therefore, by fitting the algorithm to our data, we were able to determine 2 different classes within the data, which can be clearly identified.

References

Agarwal, V. (2019). Outlier detection with boxplots.

Dunham, M. H. (2003). *Data Mining: Introductory and Advanced Topics*. Prentice Hall/Pearson Education, Upper Saddle River, N.J.

Htoon, K. S. (2020). A guide to knn imputation.

Kaplan, D. (2022). Correlation analysis in data mining.

Rahmah, N. and Sitanggang, I. S. (2016). Determination of optimal epsilon (eps) value on dbscan algorithm to clustering data on peatland hotspots in sumatra. In *IOP conference series: earth and environmental science*, volume 31, page 012012. IoP Publishing.

Sander, J., Ester, M., Kriegel, H.-P., and Xu, X. (1998). Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data mining and knowledge discovery*, 2:169–194.

Tan, P.-n., Steinbach, M., and Kumar, V. (2013). *Introduction to Data Mining.* Pearson Education UK.

A Appendix

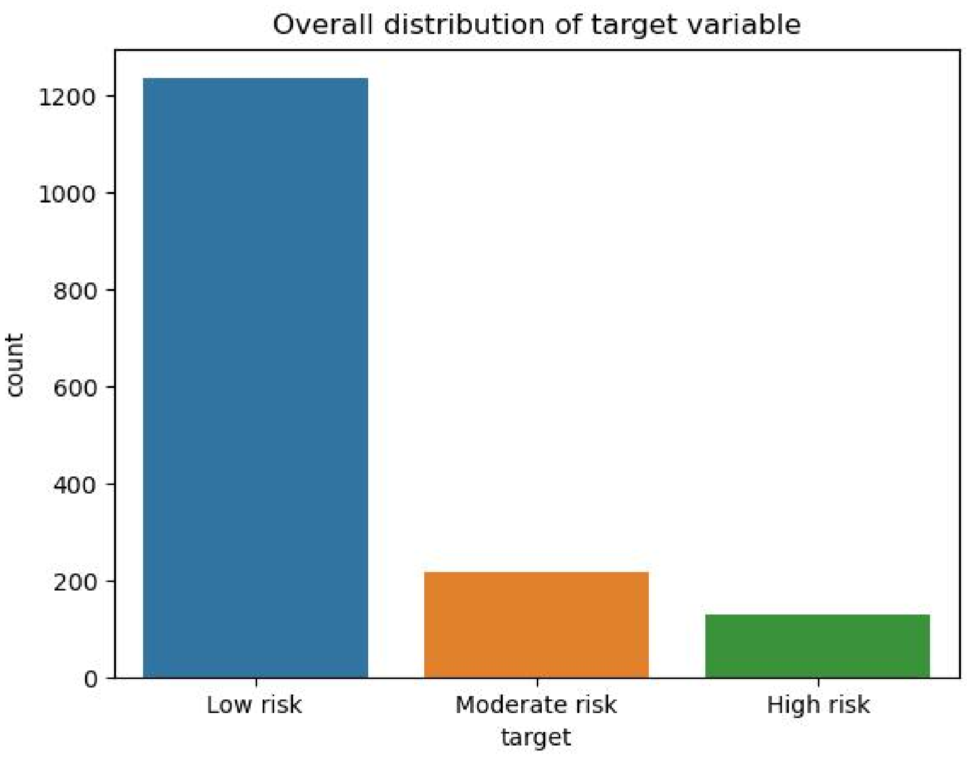

Figure 1: Distribution of Target Variable

	x1	x2	x4	x5
count	1584.0	1584.0	1584.0	1540.0
mean	1053.1881313131300	133.2979797979800	0.003169191919191919000	0.009906493506493490
std	615.9967162874	10.002632424690500	0.0038207347794625200	0.04862691996951120
min	0.0	106.0	0.0	0.0
max	2125.0	160.0	0.019	0.477
median	1049.5	133.0	0.002	0.0
missing	0.0	0.0	0.0	44.0

Table 12: Distribution of Numerical Columns (x1 - x5)

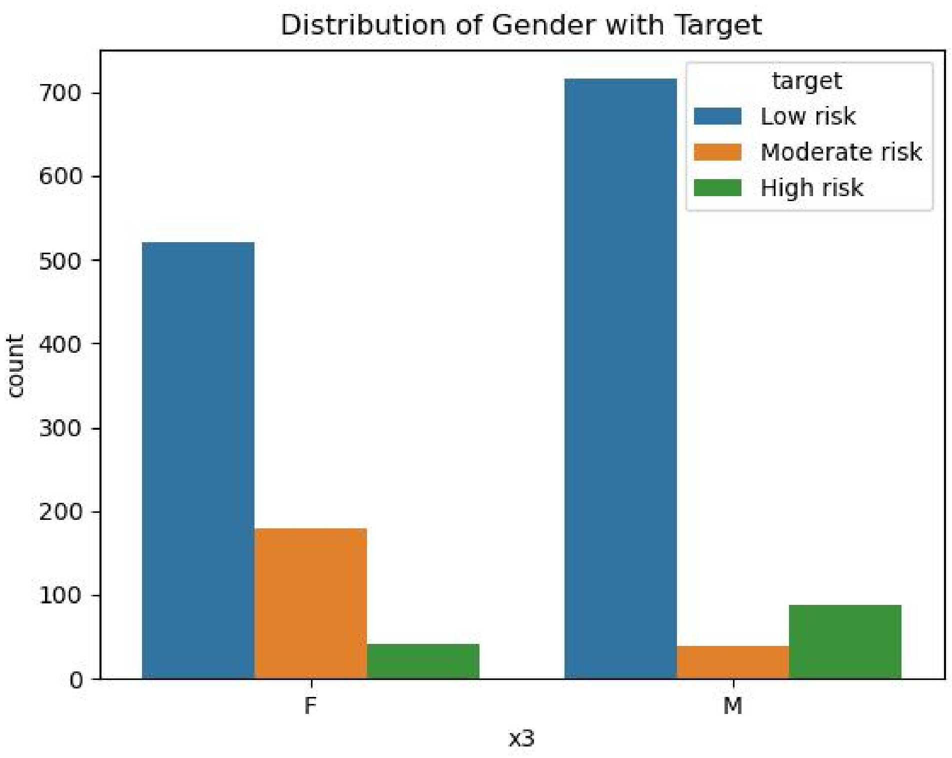

Figure 2: Distribution of Gender with Target

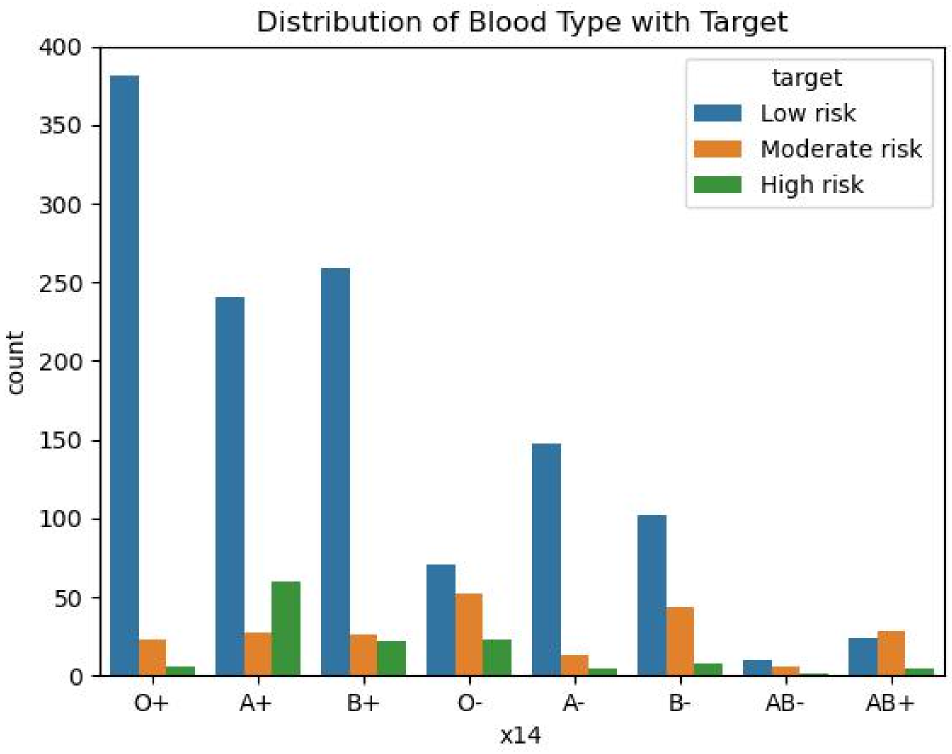

Figure 3: Distribution of Blood Type with Target

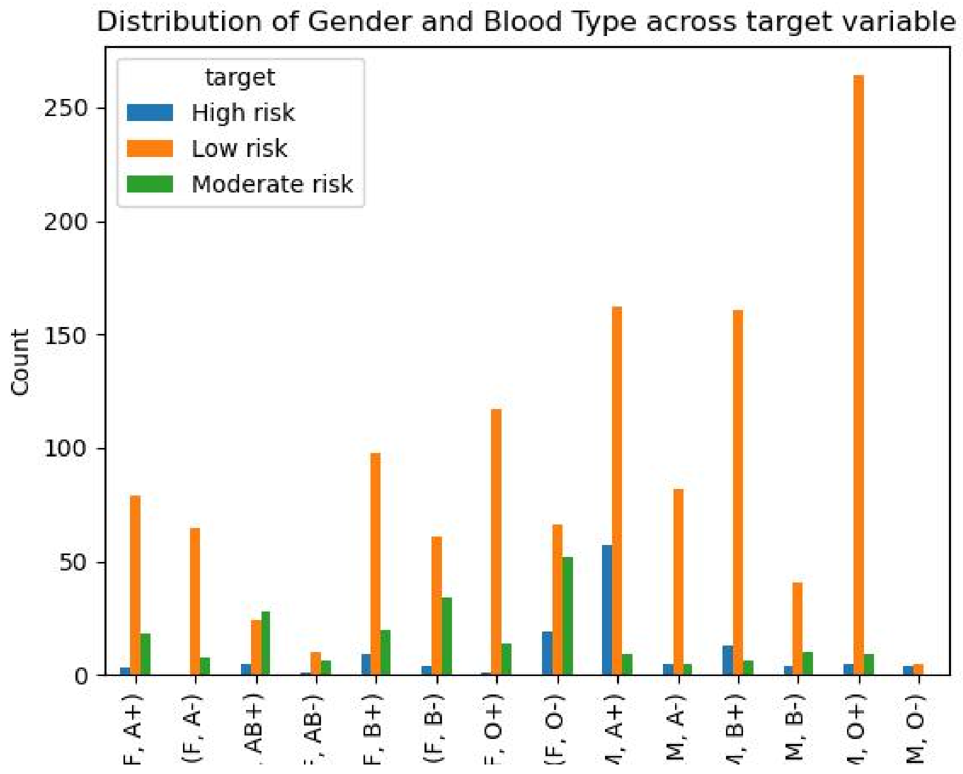

Figure 4: Distribution of Gender and Blood Type with Target

	x6	x7	x8	x9
count	1584.0	1584.0	1567.0	1584.0
mean	0.004347222222222180	0.001853535353535400	2.55264837268666E-06	0.0001571969696969700
std	0.00294774749145699	0.002939699320039860	5.04753230965842E-05	0.0005925724482898130
min	0.0	0.0	0.0	0.0
max	0.014	0.015	0.001	0.005
median	0.004	0.0	0.0	0.0
missing	0.0	0.0	17.0	0.0

Table 13: Distribution of Numerical Columns (x6 - x9)

	x10	x11	x12	x13
count	1557.0	1584.0	1584.0	1584.0
mean	47.09441233140660	1.337815656565660	10.008838383838400	8.25523989898989
std	17.269620587495400	0.8990922783200590	18.520205903223700	5.784578606790800
min	13.0	0.2	0.0	0.0
max	86.0	7.0	91.0	50.7
median	49.0	1.2	0.0	7.4
missing	27.0	0.0	0.0	0.0

Table 14: Distribution of Numerical Columns (x10 - x13)

	x15	x16	x17	x18
count	1584.0	1584.0	1584.0	1584.0
mean	70.4090909090909	93.49684343434340	163.90593434343400	4.063762626262630
std	38.99389223672560	29.593369528259500	17.908749263987300	2.950267927354380
min	3.0	50.0	122.0	0.0
max	176.0	158.0	238.0	18.0
median	68.0	93.0	162.0	3.0
missing	0.0	0.0	0.0	0.0

Table 15: Distribution of Numerical Columns (x15 - x18)

	x19	x20	x21	x22
count	1584.0	1584.0	1584.0	1584.0
mean	0.3244949494949500	137.33396464646500	134.5429292929290	137.93560606060600
std	0.7184991709972200	16.461642712598000	15.729734945976500	14.622679704135700
min	0.0	60.0	73.0	77.0
max	10.0	187.0	182.0	186.0
median	0.0	139.0	136.0	139.0
missing	0.0	0.0	0.0	0.0

Table 16: Distribution of Numerical Columns (x19 - x22)

	x23	x24
count	1584.0	1584.0
mean	18.44823232323230	0.30934343434343400
std	28.375001621585100	0.6158681412330020
min	0.0	-1.0
max	269.0	1.0
median	7.0	0.0
missing	0.0	0.0

Table 17: Distribution of Numerical Columns (x23 - x24)

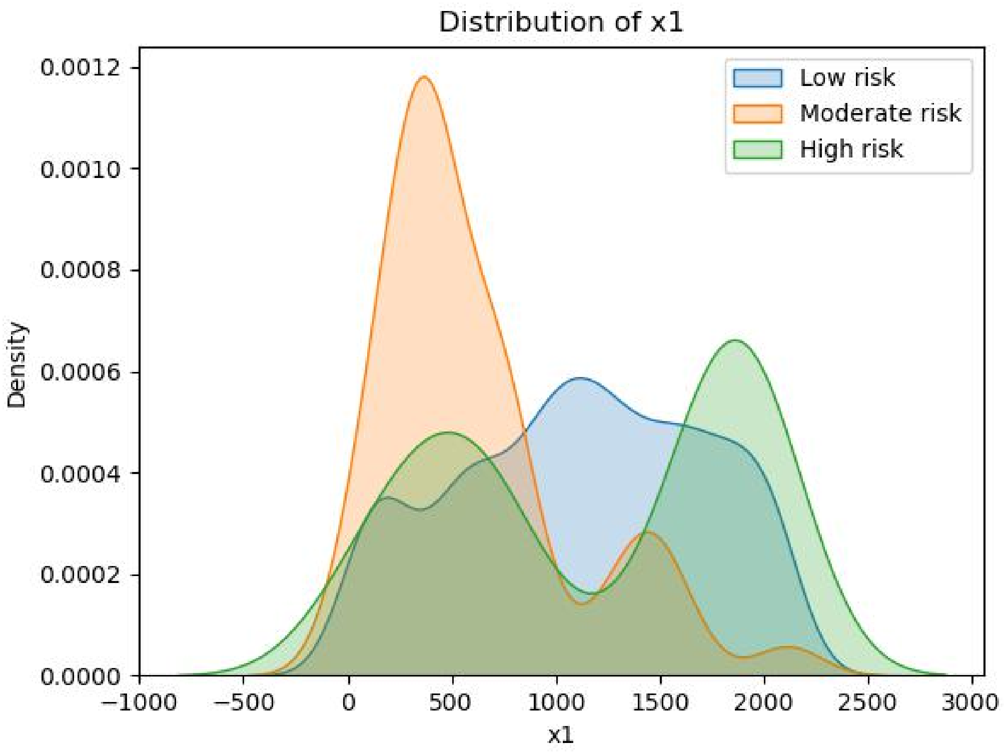

Figure 5: Distribution of x1 with Target

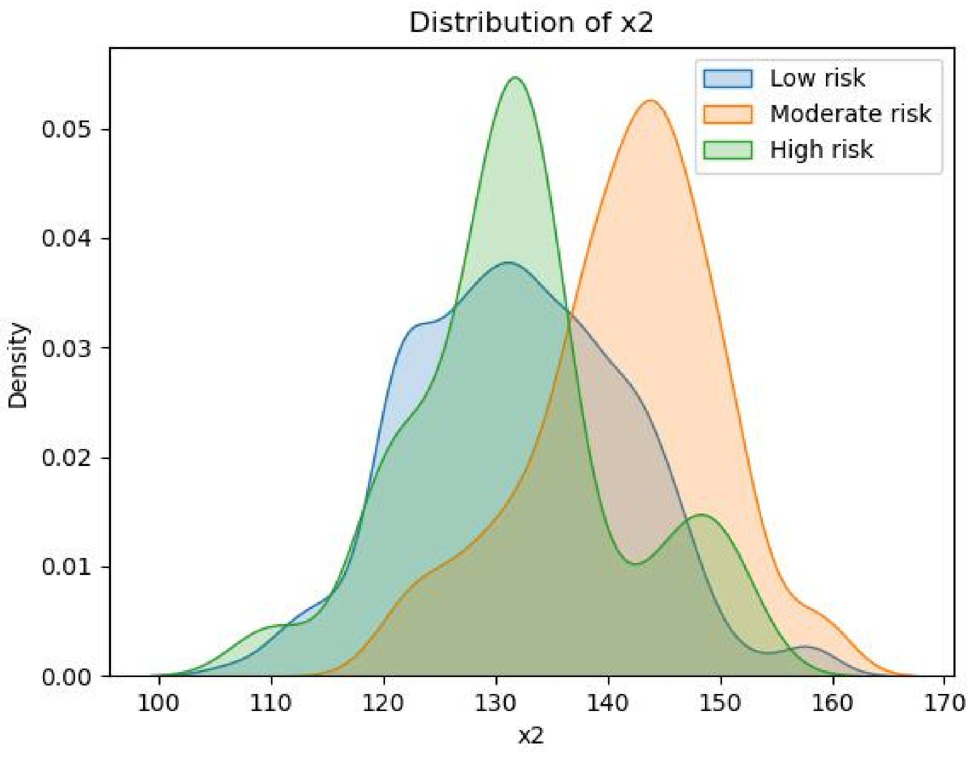

Figure 6: Distribution of x2 with Target

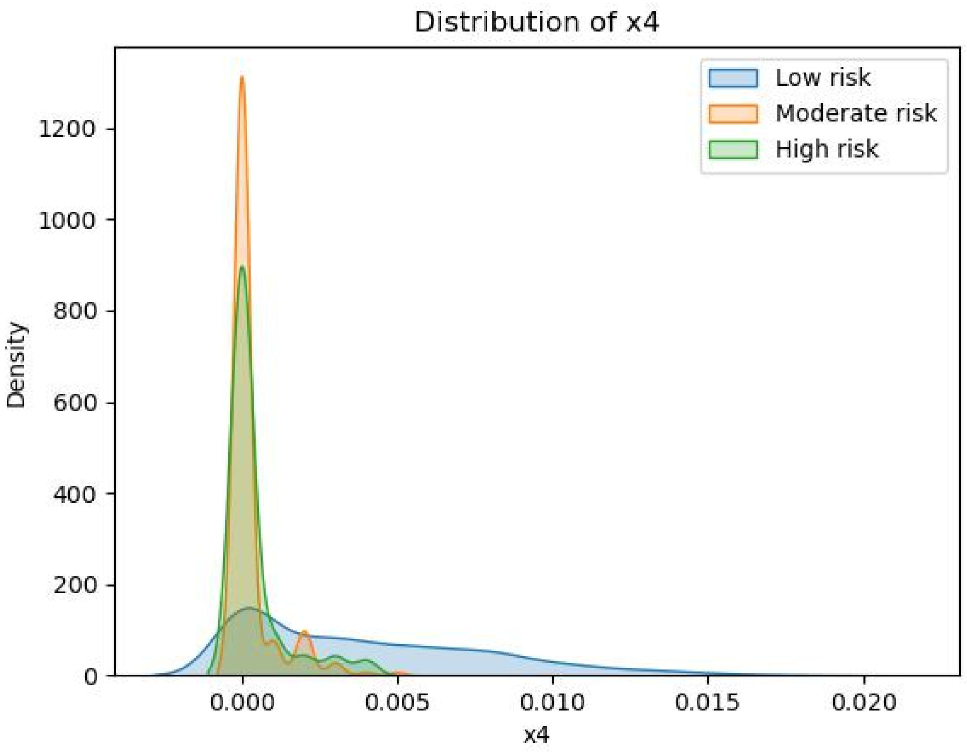

Figure 7: Distribution of x4 with Target

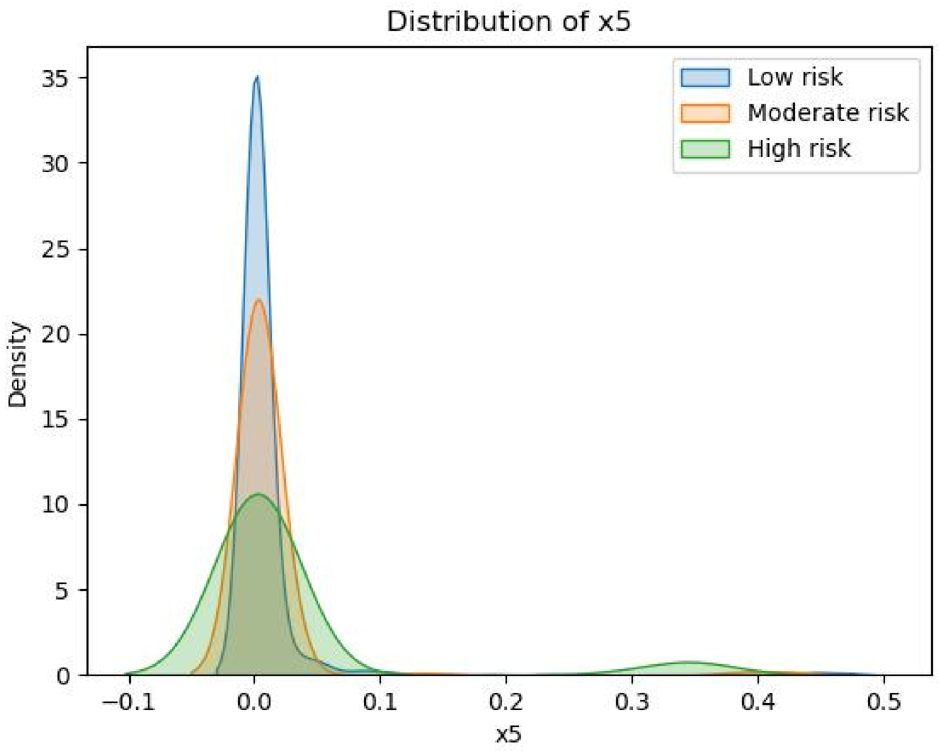

Figure 8: Distribution of x5 with Target

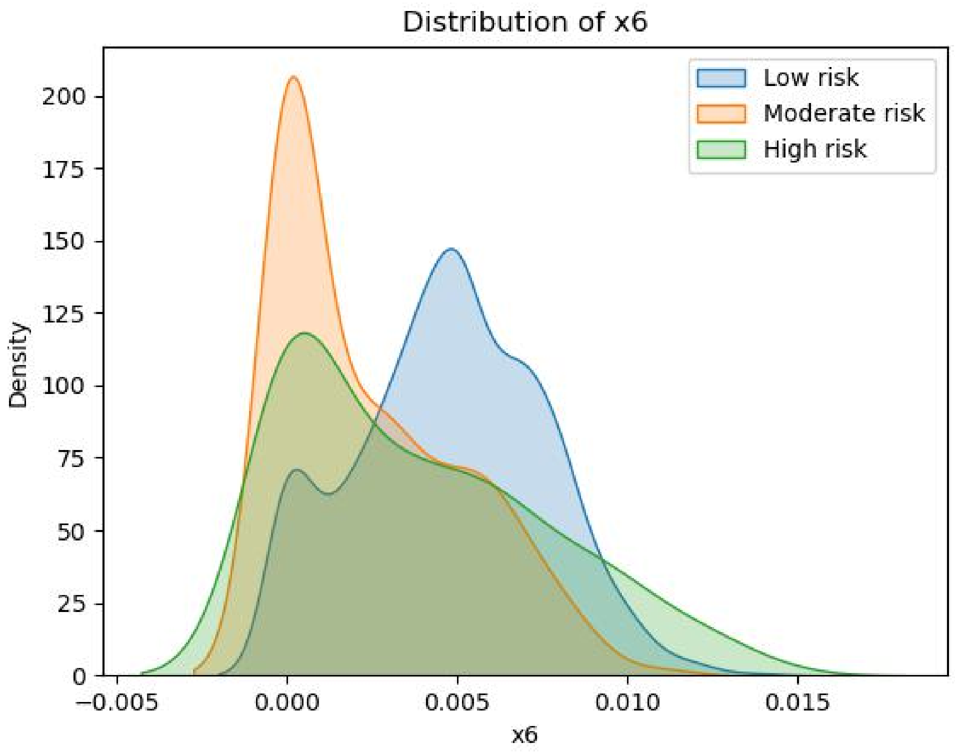

Figure 9: Distribution of x6 with Target

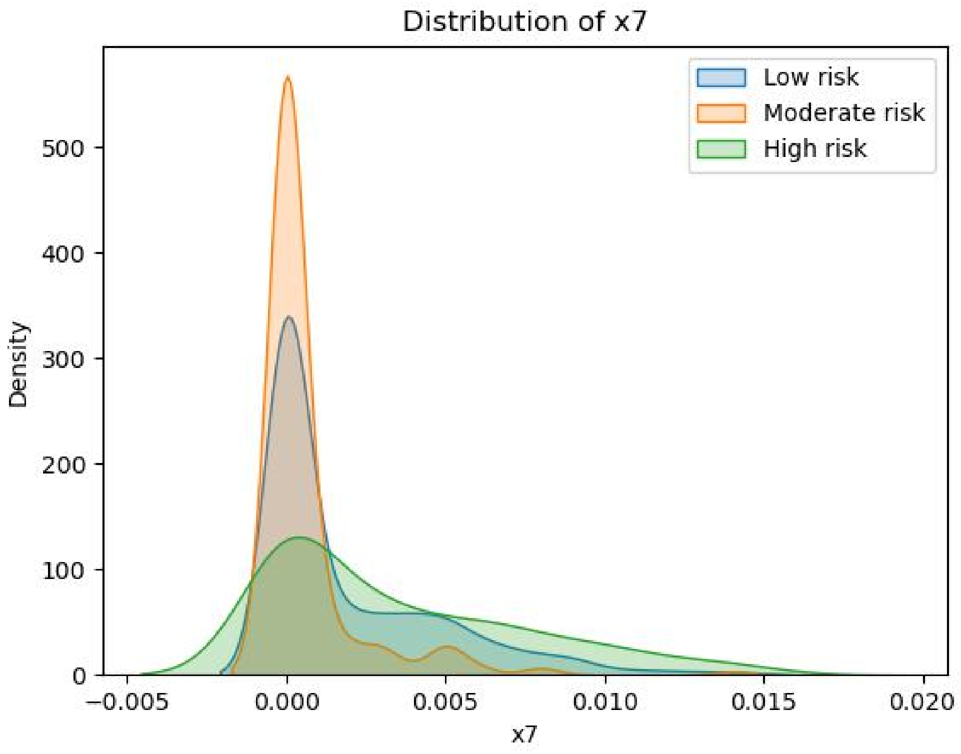

Figure 10: Distribution of x7 with Target

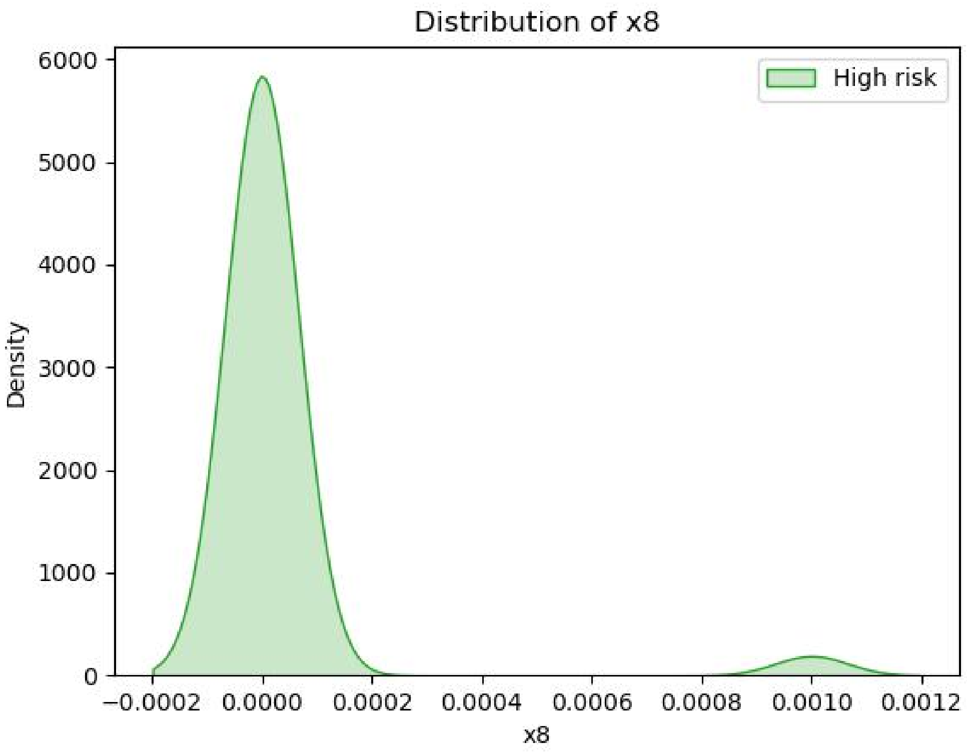

Figure 11: Distribution of x8 with Target

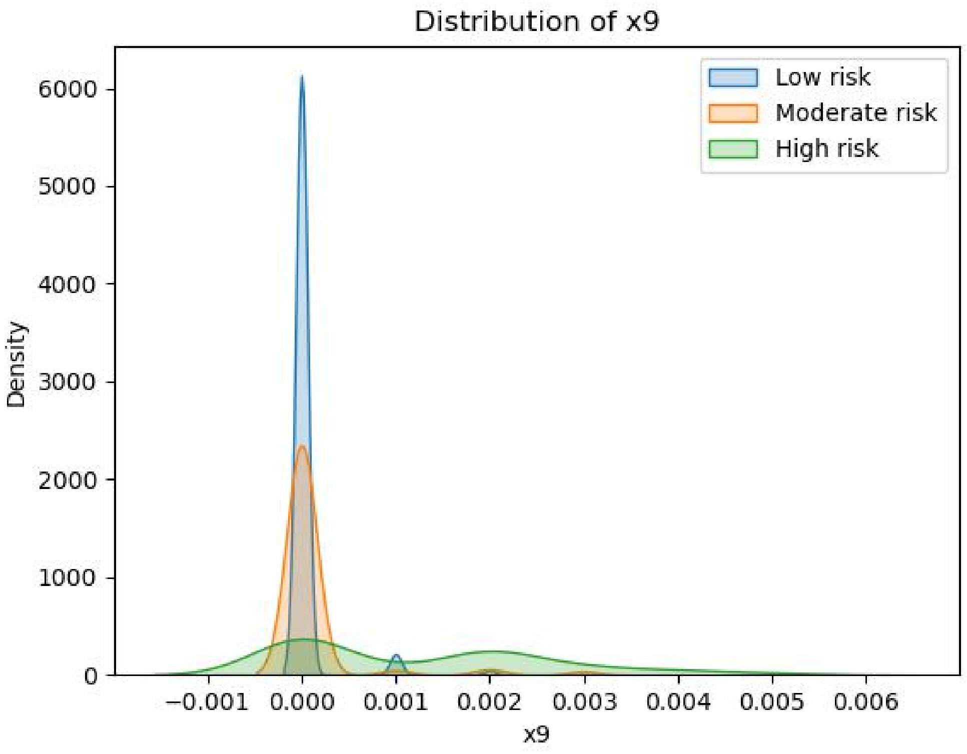

Figure 12: Distribution of x9 with Target

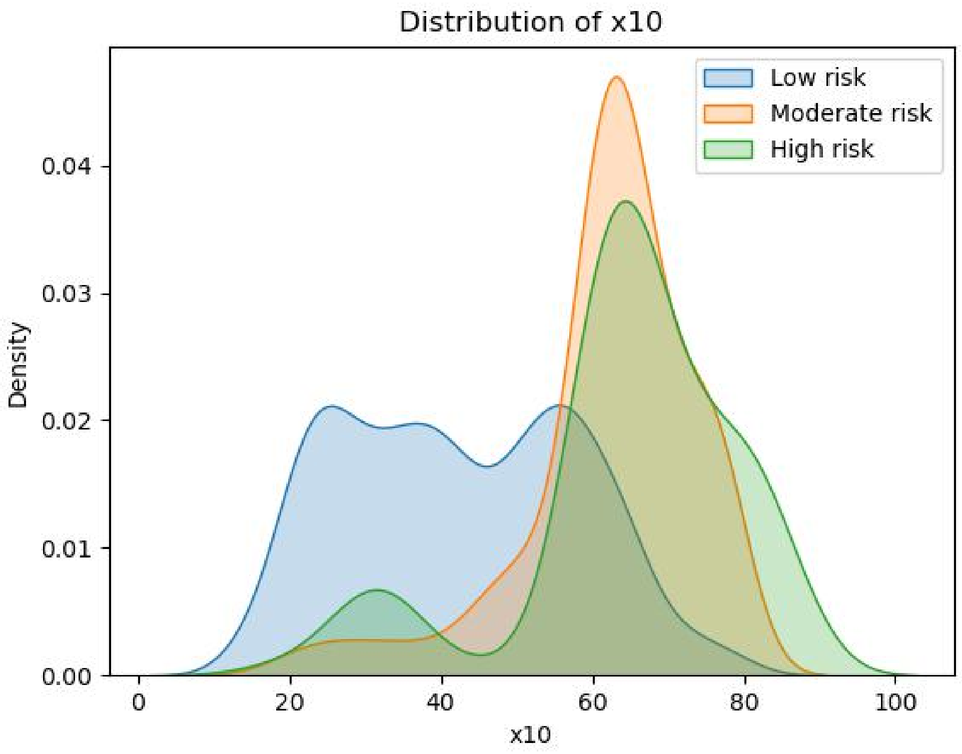

Figure 13: Distribution of x10 with Target

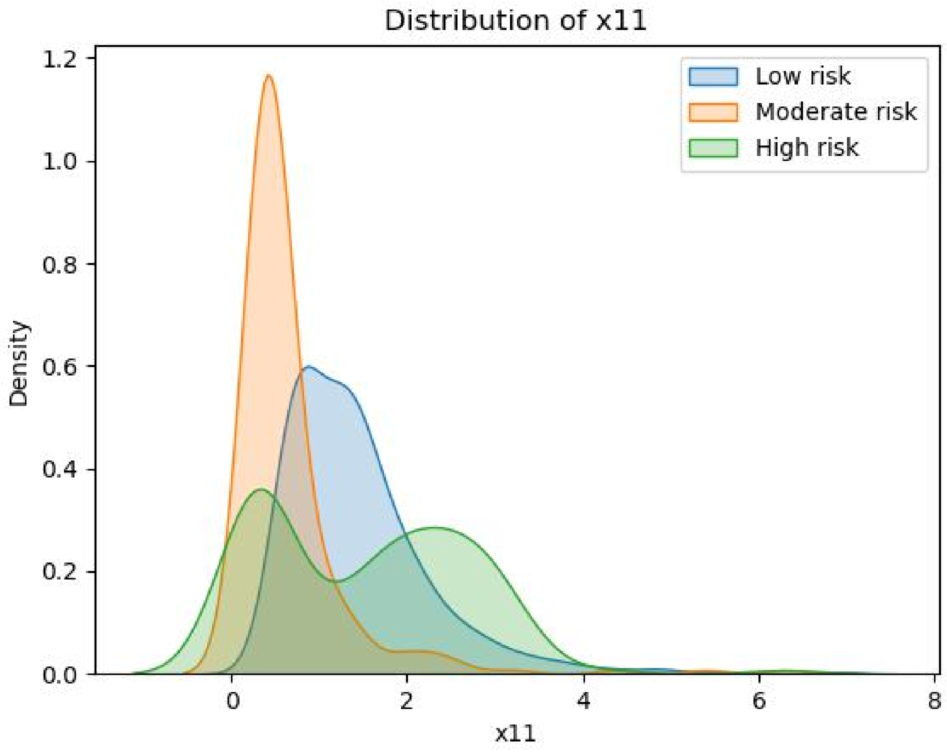

Figure 14: Distribution of x11 with Target

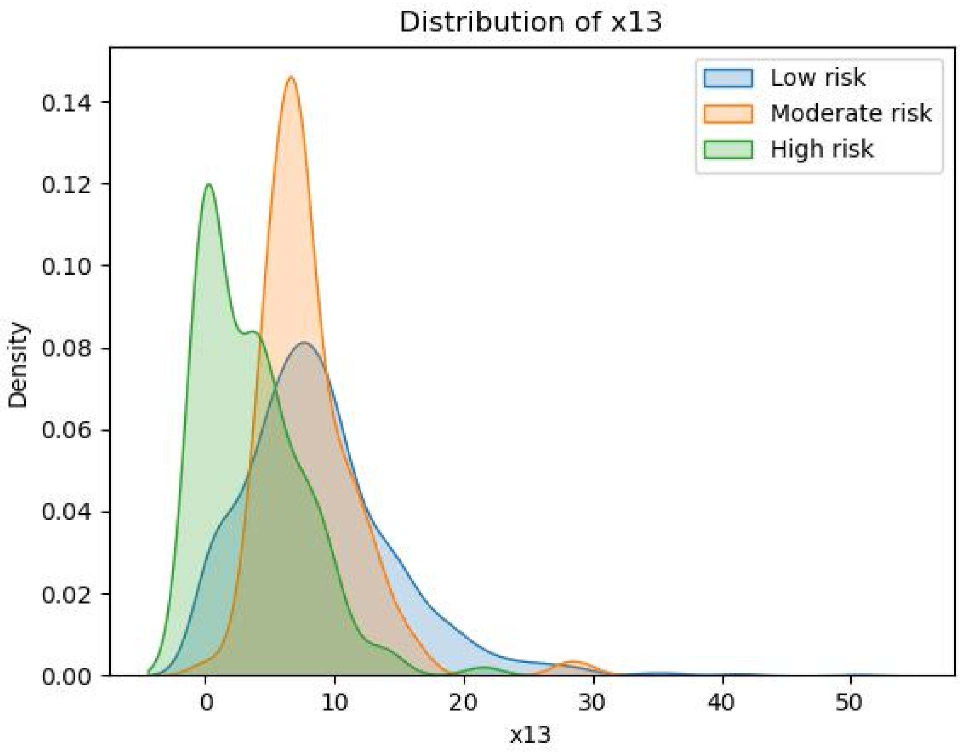

Figure 15: Distribution of x13 with Target

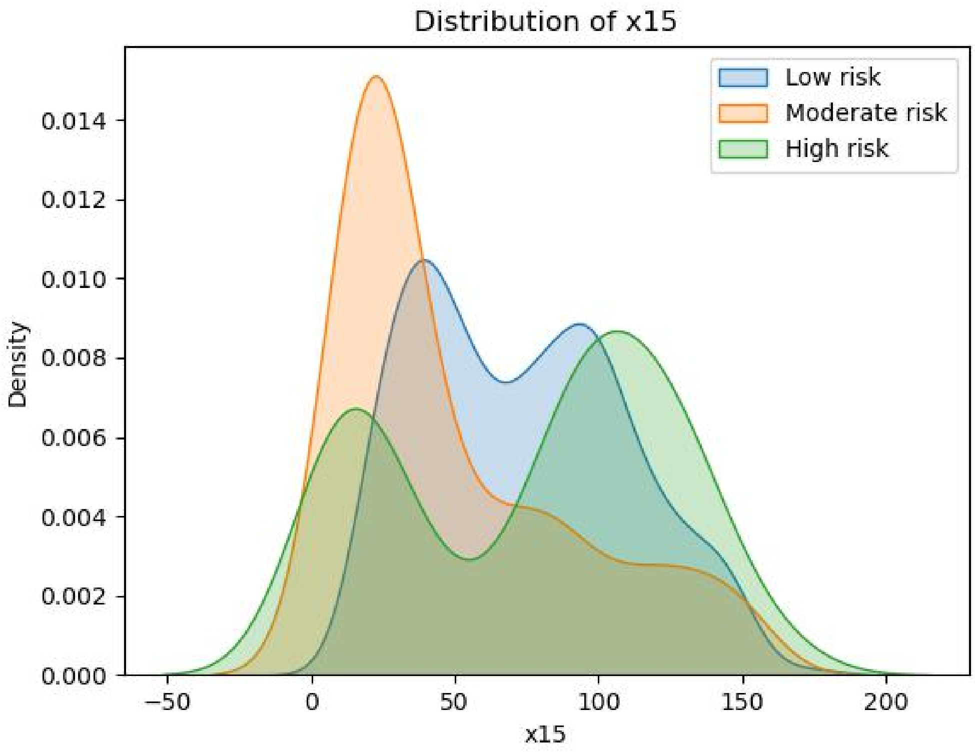

Figure 16: Distribution of x15 with Target

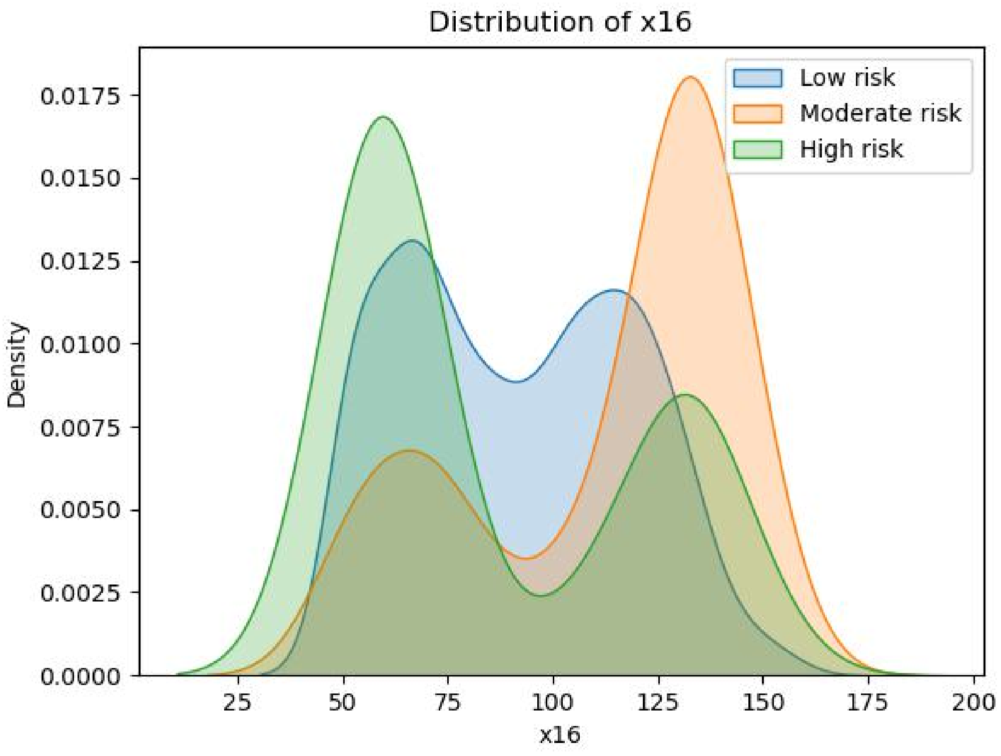

Figure 17: Distribution of x16 with Target

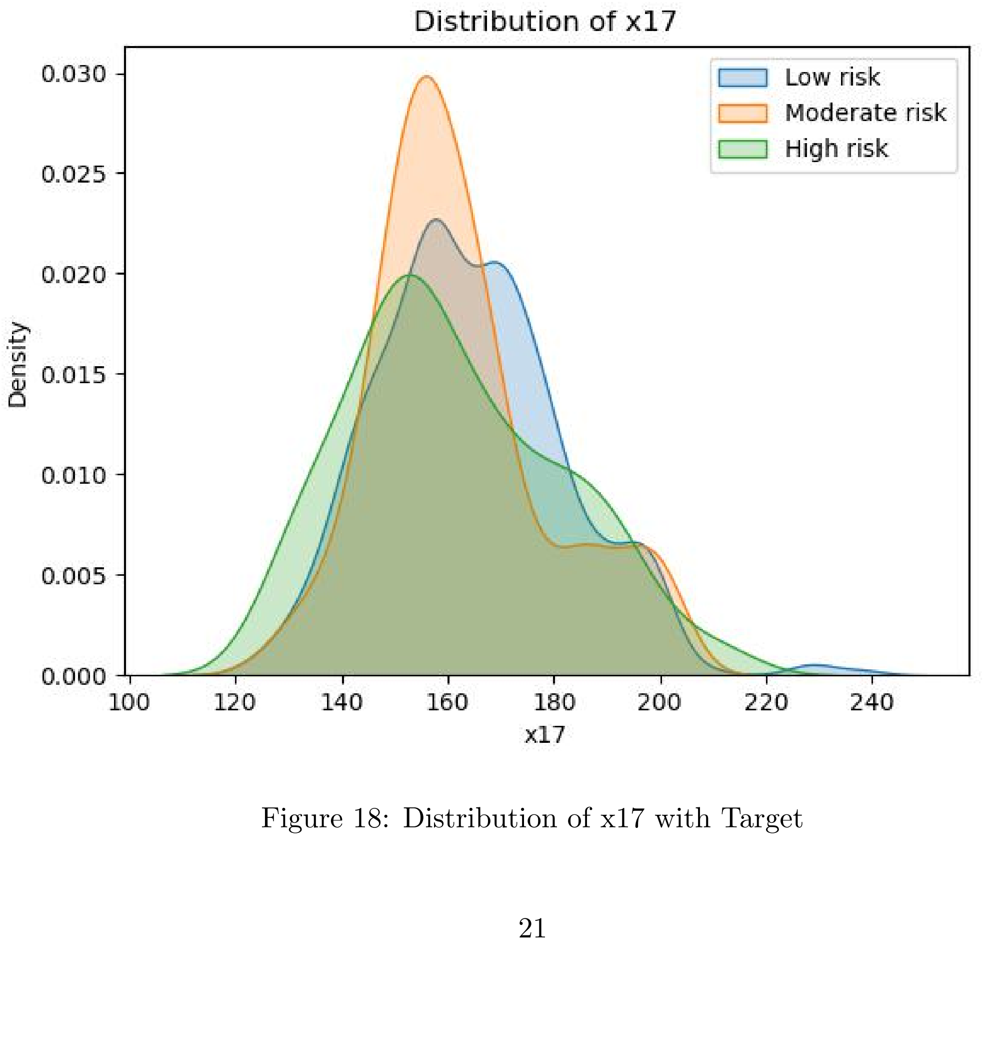

Figure 18: Distribution of x17 with Target

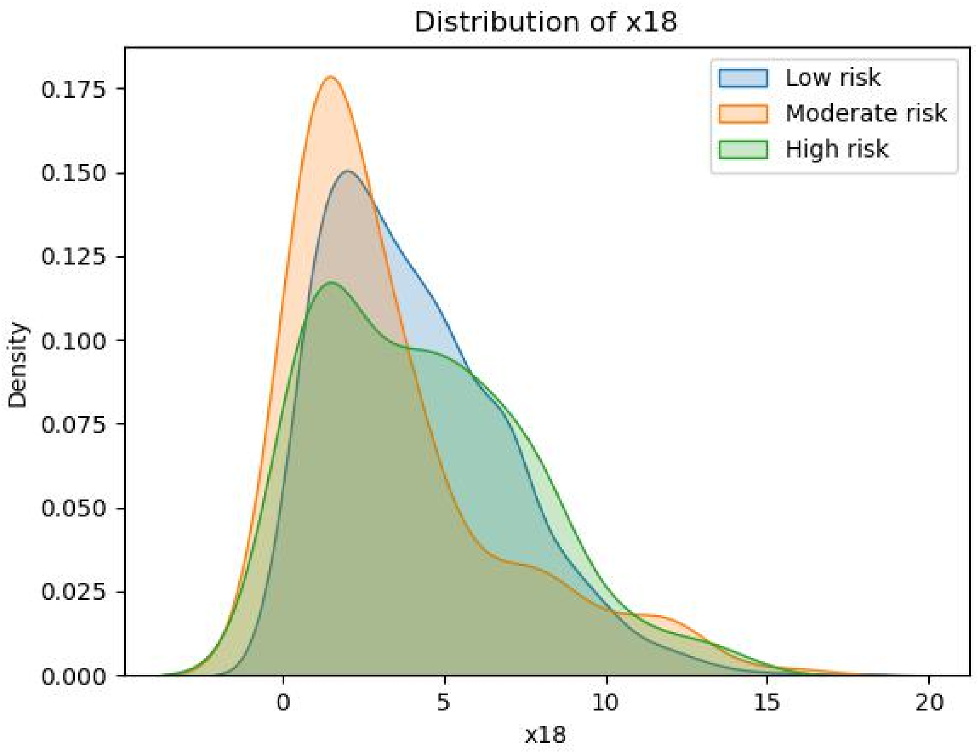

Figure 19: Distribution of x18 with Target

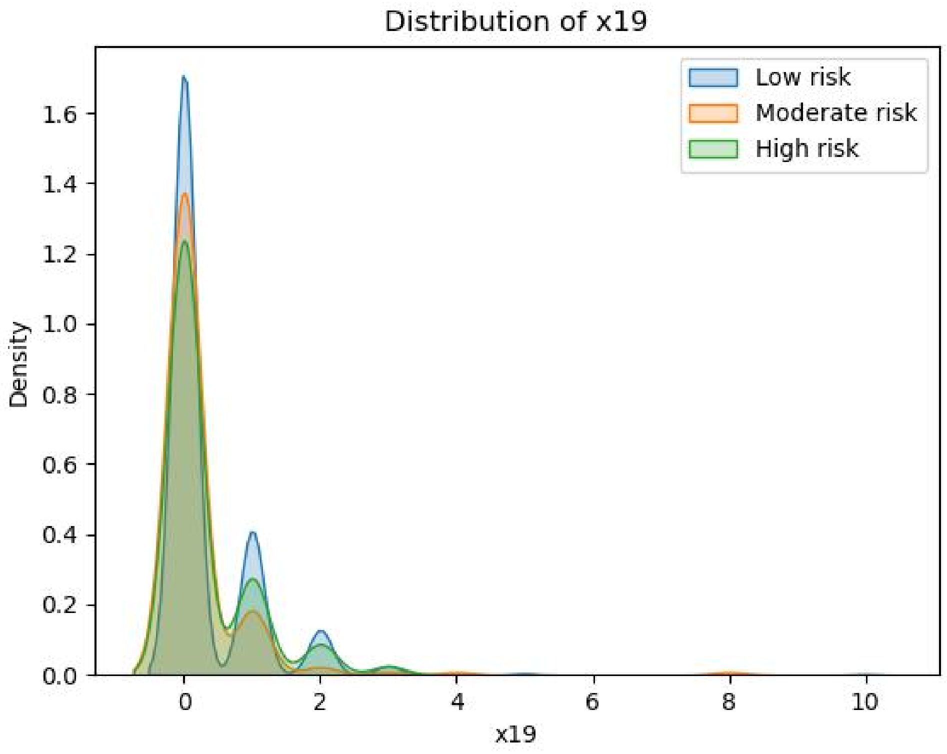

Figure 20: Distribution of x19 with Target

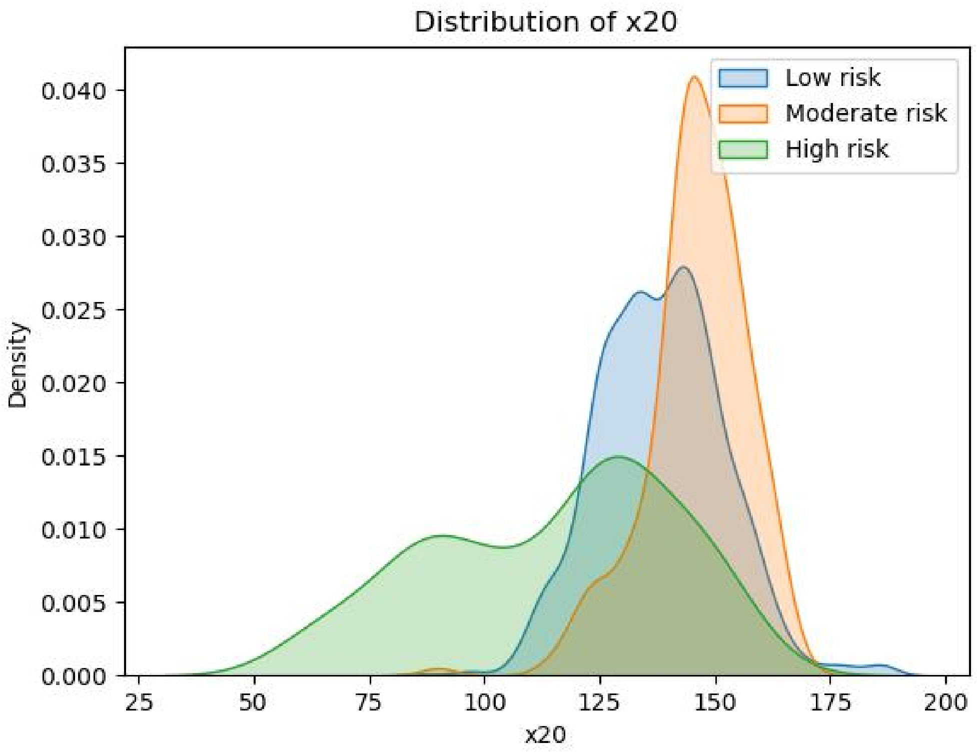

Figure 21: Distribution of x20 with Target

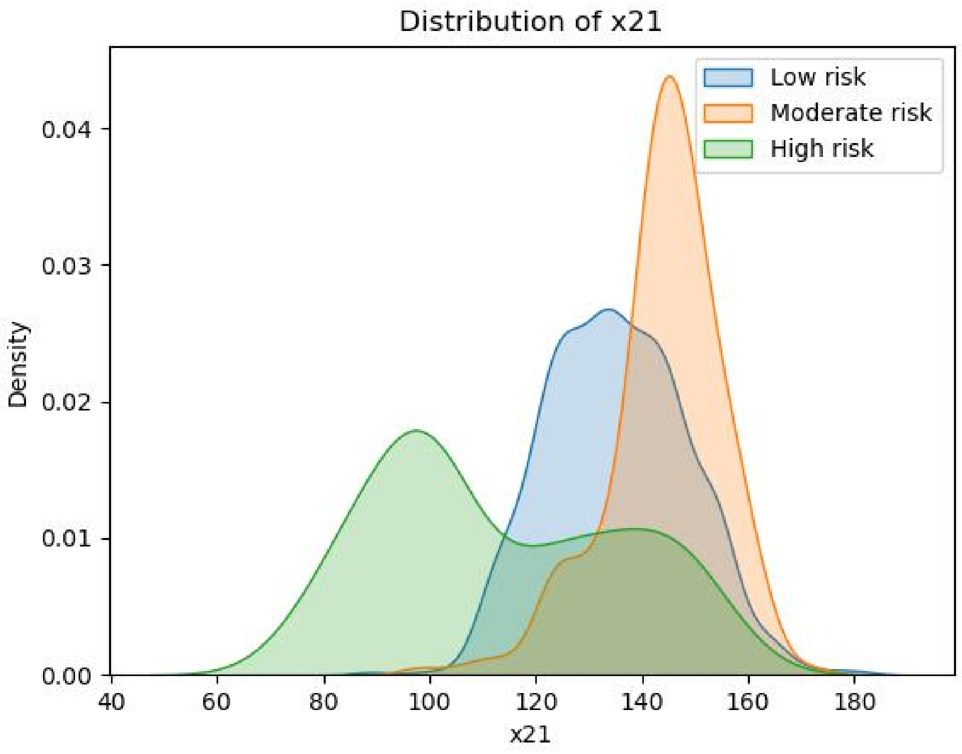

Figure 22: Distribution of x21 with Target

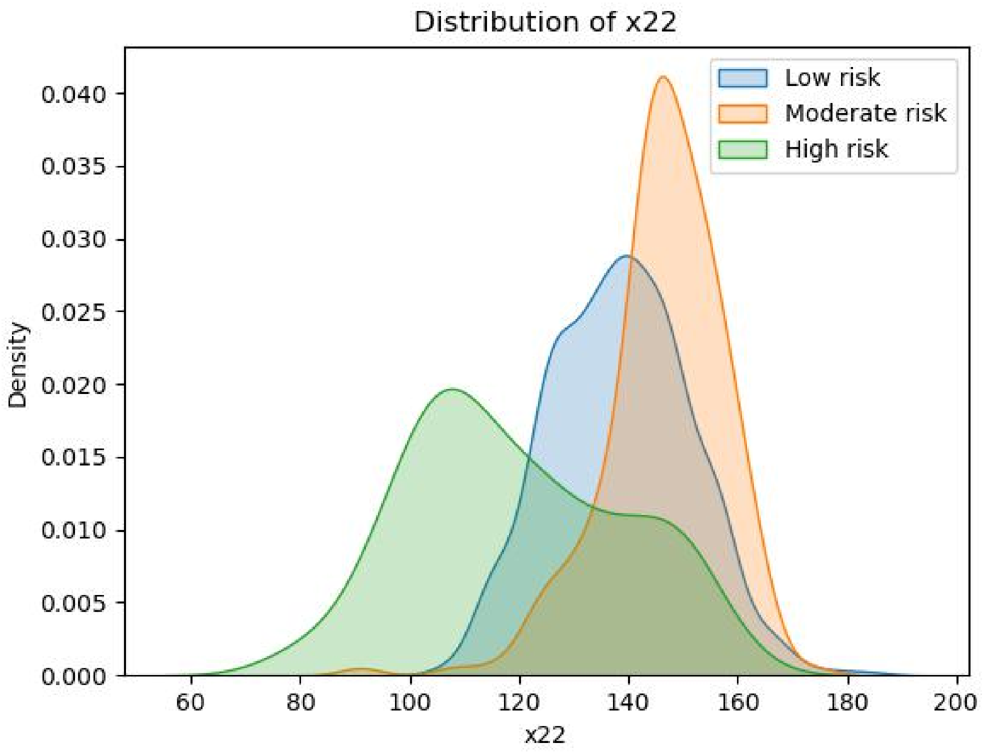

Figure 23: Distribution of x22 with Target

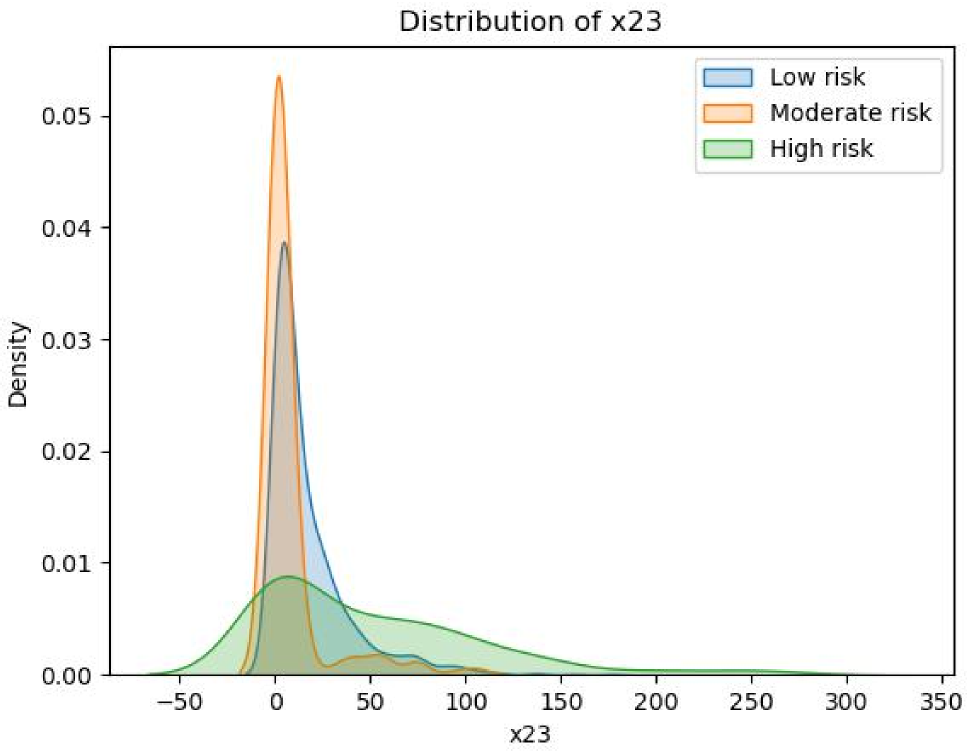

Figure 24: Distribution of x23 with Target

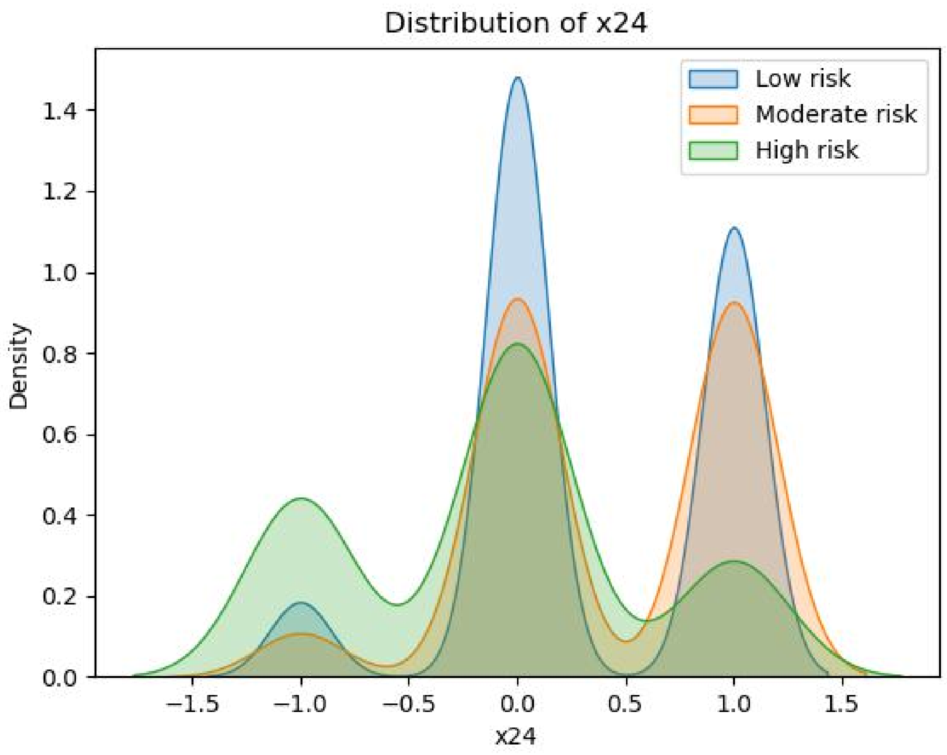

Figure 25: Distribution of x24 with Target

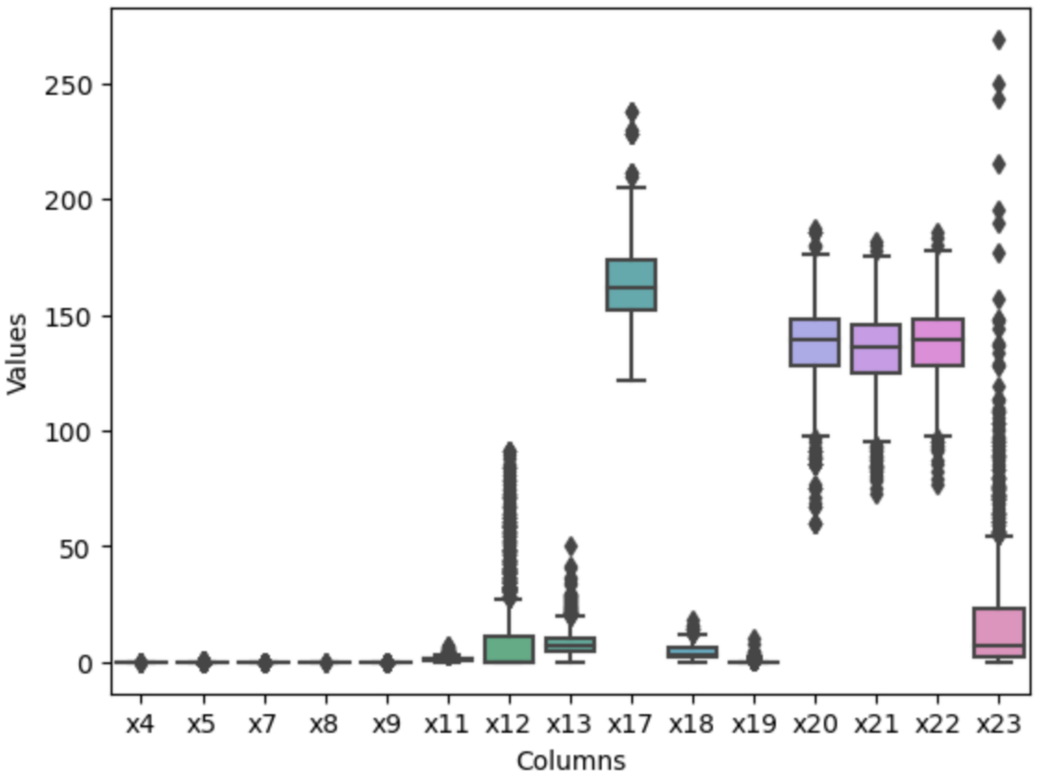

Figure 26: Detecting outliers in individual fields using a Boxplot

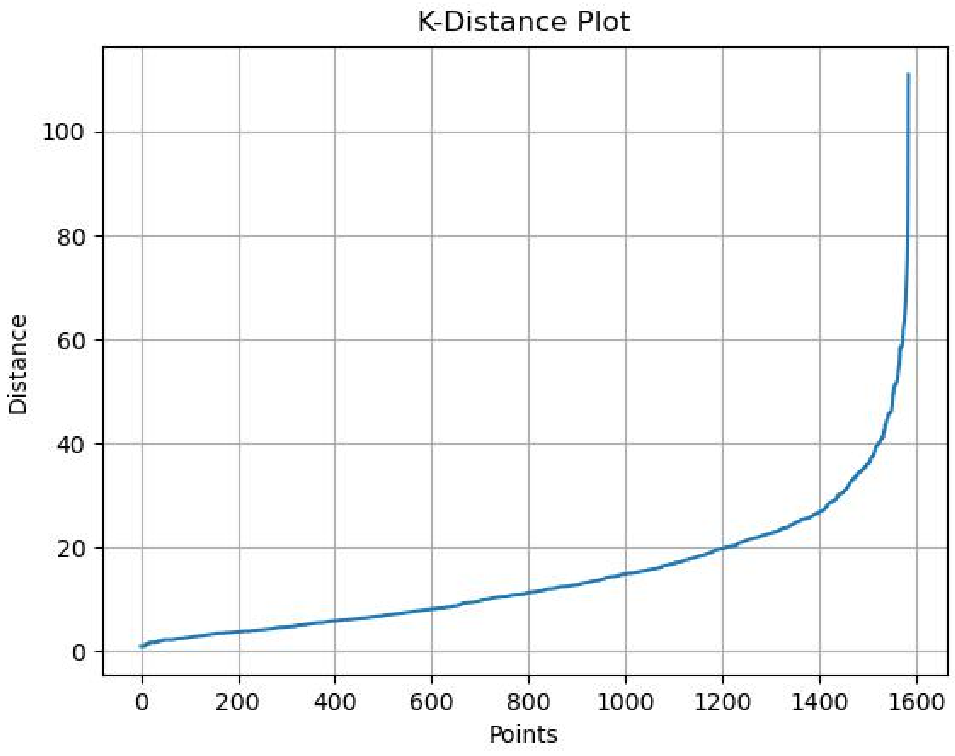

Figure 27: K-distance Plot for *eps* parameter determination

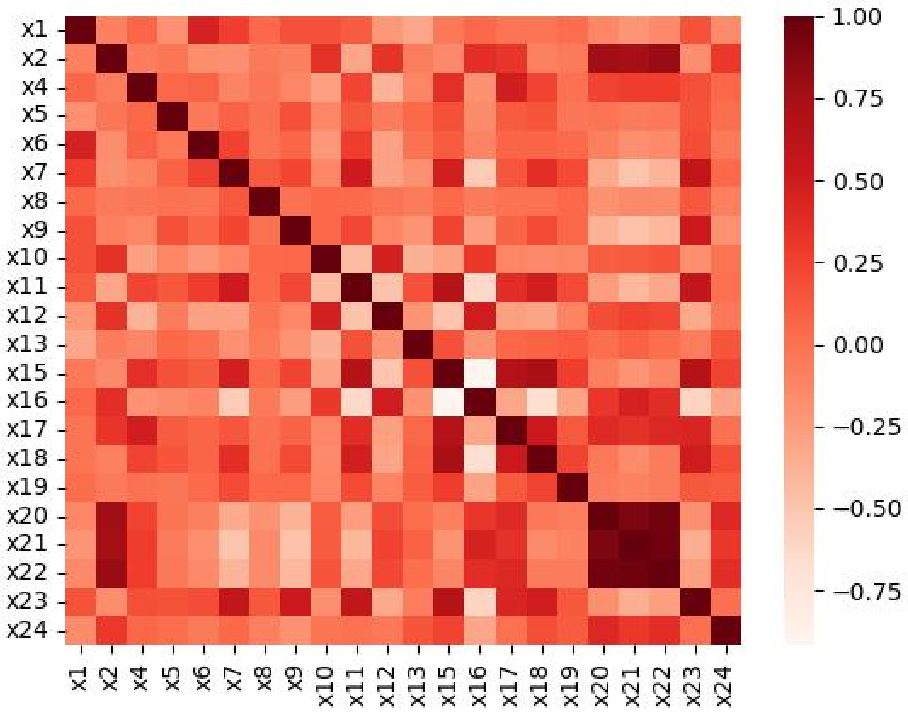

Figure 28: Heatmap showing the correlational analysis of numerical columns

Columns	VIF
x1	9.030821138418537
x2	1291.5304844175646
x4	5.187949376119607
x5	1.1741426885051276
x6	5.051290342423822
x7	4.73321161586137
x8	1.2141667077999245
x9	3.4983831819481863
x10	18.726374516663544
x11	10.152249348563718
x12	2.8051894804813977
x13	6.749581666320324
x15	inf
x16	inf
x17	inf
x18	6.82220568596364
x19	1.4224406925621829
x20	1491.162341758748
x21	3188.4043769619266
x22	5124.809691240905
x23	5.577853700708042
x24	3.72152211563566

Table 18: VIF values for each columns to determine highly correlated variables

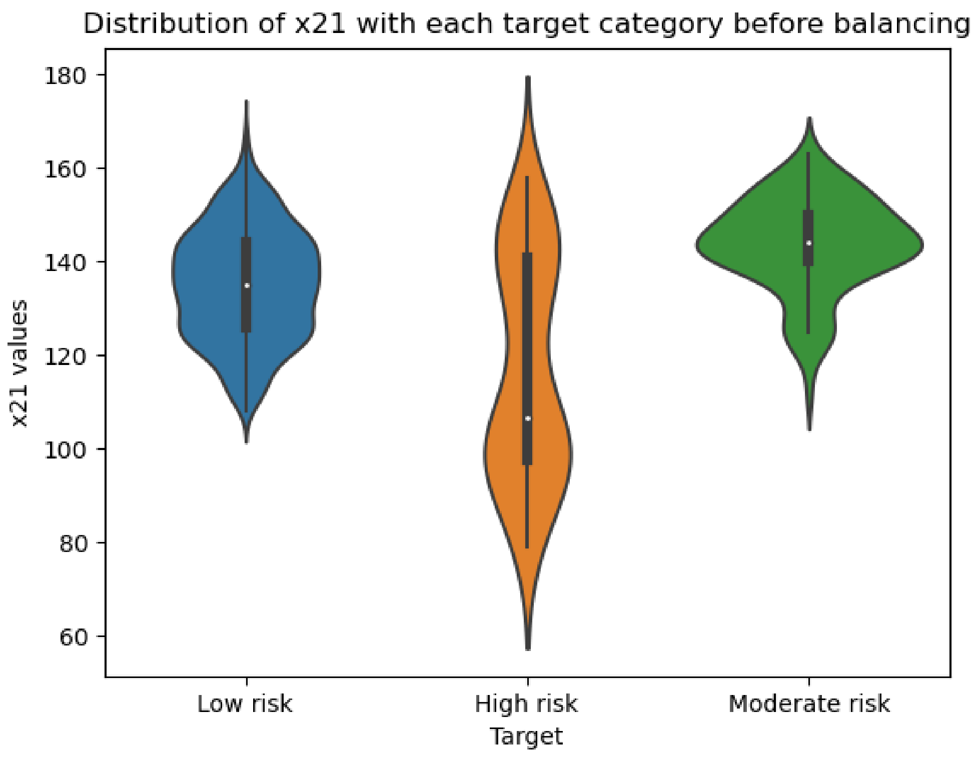

Figure 29: Distribution of 'x21' before balancing

Feat$_{names}$	ANOVA F measure
x9	358.0340410683420
x10	256.38234931668600
x12	250.04233317801500
x4	147.79181607461900
x23	111.92504642361400
x11	80.4092530162967
S$_M$	71.89952362273220
x1	71.8308139513011
x6	65.46352110410450
x16	57.05716890280420
x13	52.453220913904600
B$_O-$	44.34902575735340
x7	41.55655432776500
B$_O+$	38.487165699837800
x15	36.799977381727700
x24	34.31432774533360
B$_AB+$	32.773975169487100
B$_A+$	32.04076231791860
x8	23.03565416065420
B$_B-$	16.156359241366000
x5	8.528675397691210
B$_A-$	6.870743395920380
x18	6.78849064737927
B$_B+$	5.227064982278980
B$_AB-$	3.3324120442885200
x19	3.2880396499449600
x17	3.1716461400949200

Table 19: Ranked Anova F-statistic scores with their respective columns

Feature	Kendall's Correlation
x12	0.3001072676398560
x21	0.2553248326714320
x22	0.2517033633247700
x20	0.2381891644915190
S_F	0.22753810971411700
x16	0.21379683123257300
x2	0.2052112851532840
x24	0.11554632030118700
B_tar	0.11105236195026600
x10	0.10701905160127800
x13	0.0919757583758995
x5	0.07296680757616770
x17	-0.01717542307854690
x19	-0.08559617772435110
x8	-0.08969147294581910
x18	-0.10102151505476800
x4	-0.1216367635913370
x6	-0.14348732043388400
x15	-0.18666115810924200
x1	-0.2024108922613900
x7	-0.20555983708999200
S_M	-0.22753810971411700
x23	-0.2571207835608040
x9	-0.26660179642318700
x11	-0.296878441397752

Table 20: Ranked Kendall's correlation coefficient with their respective columns

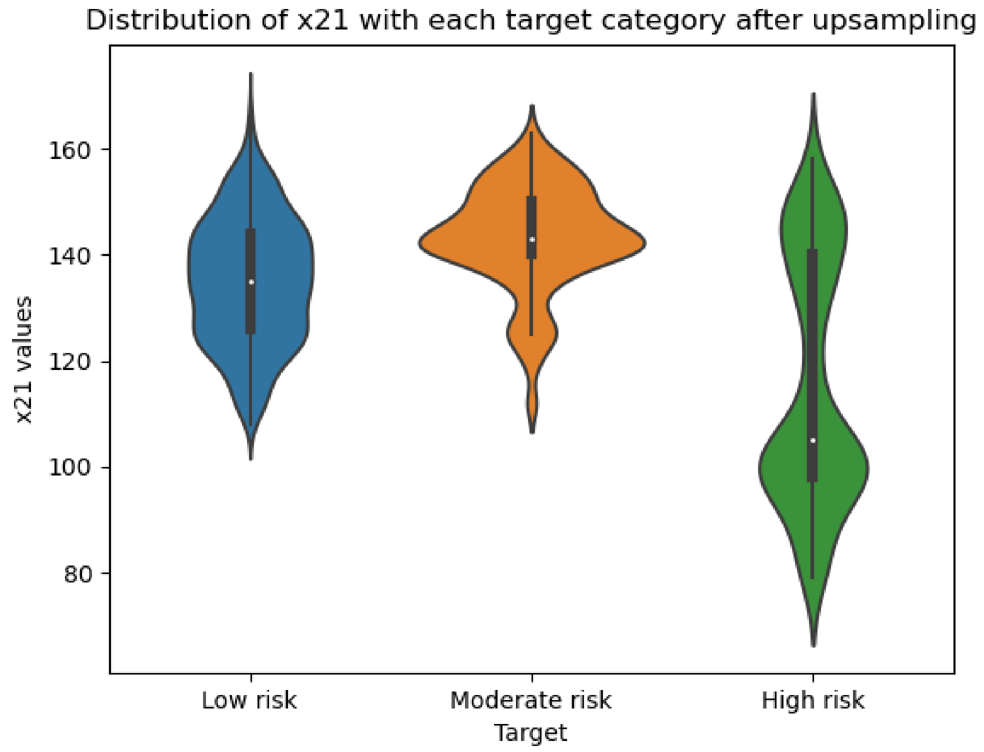

Figure 30: Upsampled data showing the ‘x21’ column

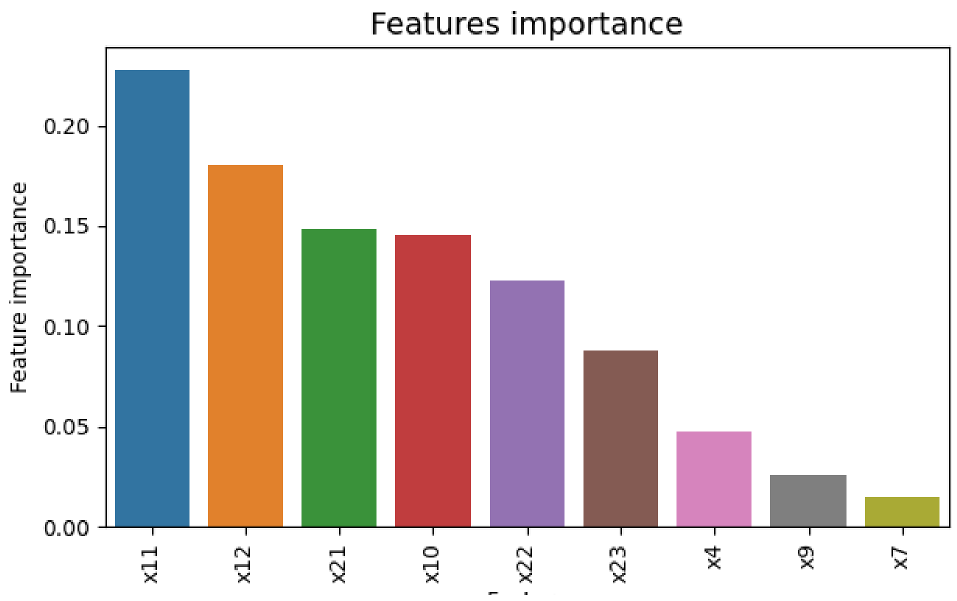

Figure 31: Ranked Feature importance of selected columns

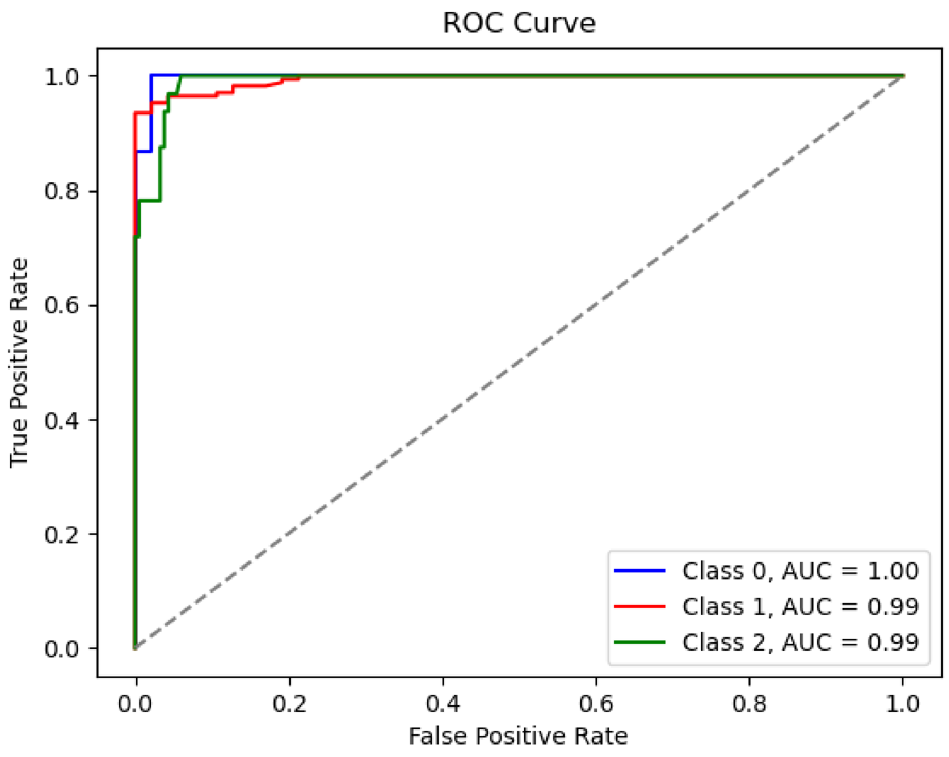

Figure 32: ROC curve plot of the Random Forest classification model

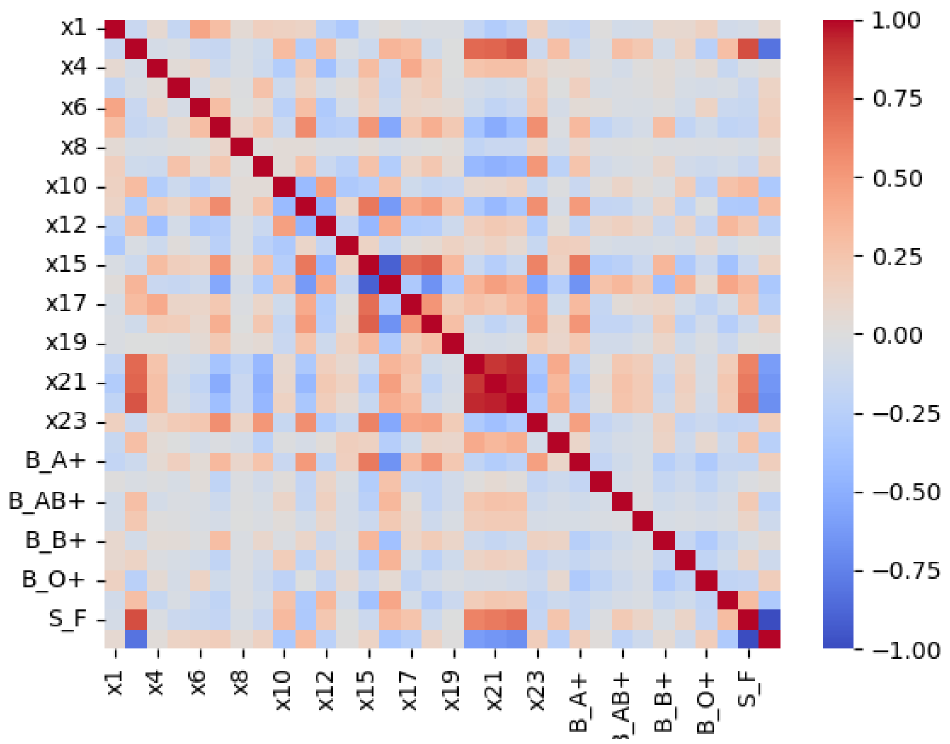

Figure 33: Heatmap showing the correlational analysis of numerical columns for unsupervised learning

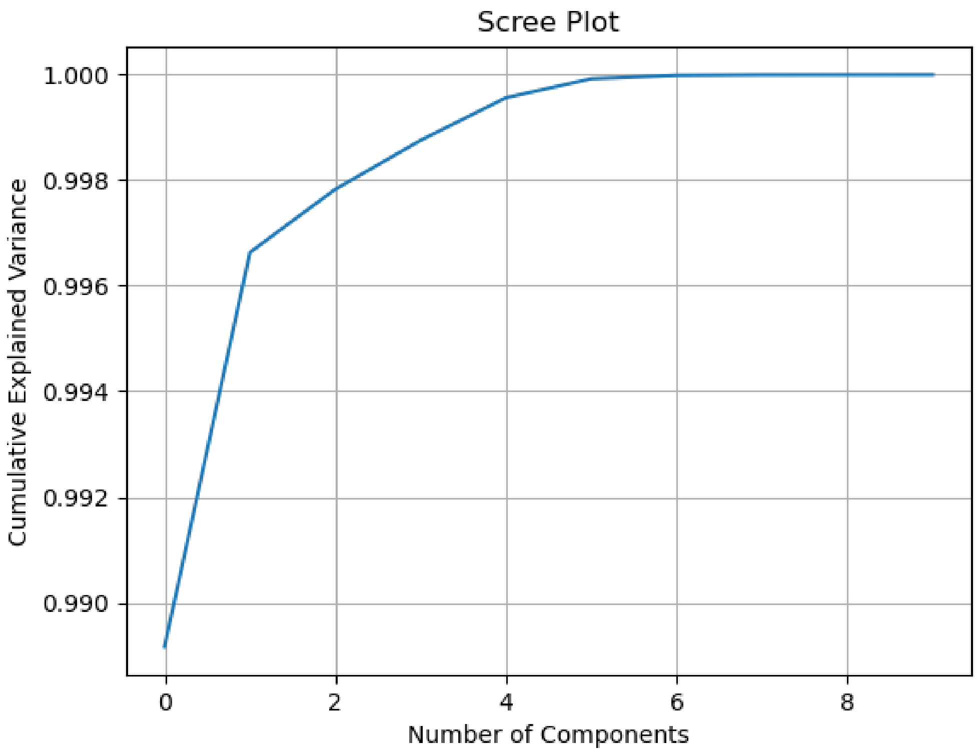

Figure 34: Scree plot for n_component determination for PCA

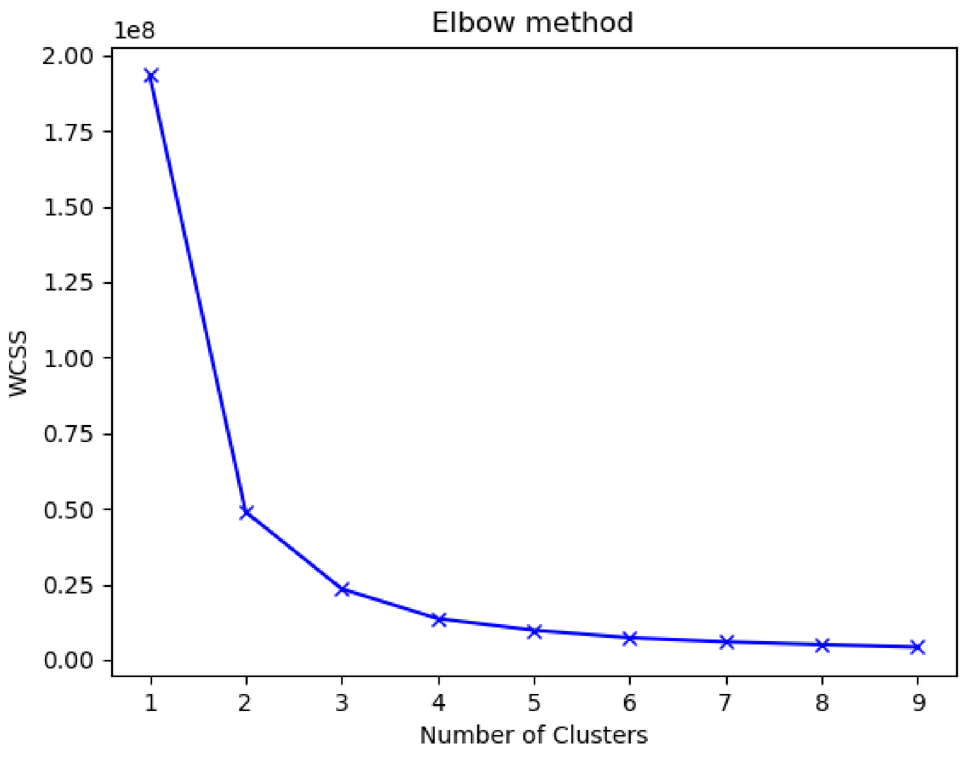

Figure 35: Elbow Method Plot

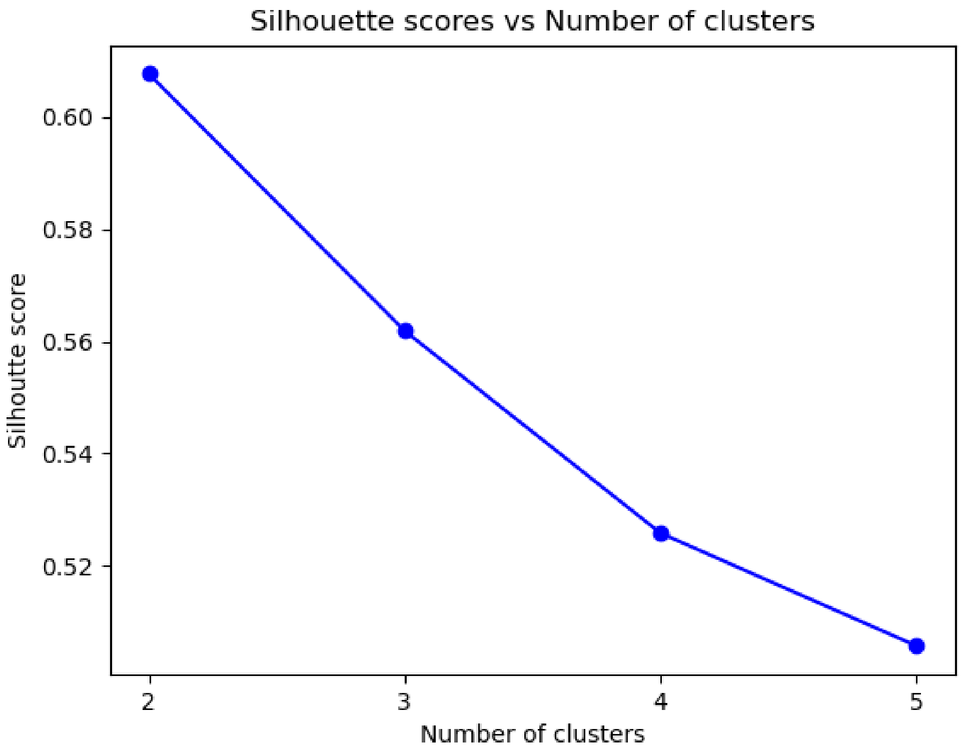

Figure 36: Silhouette Method Plot

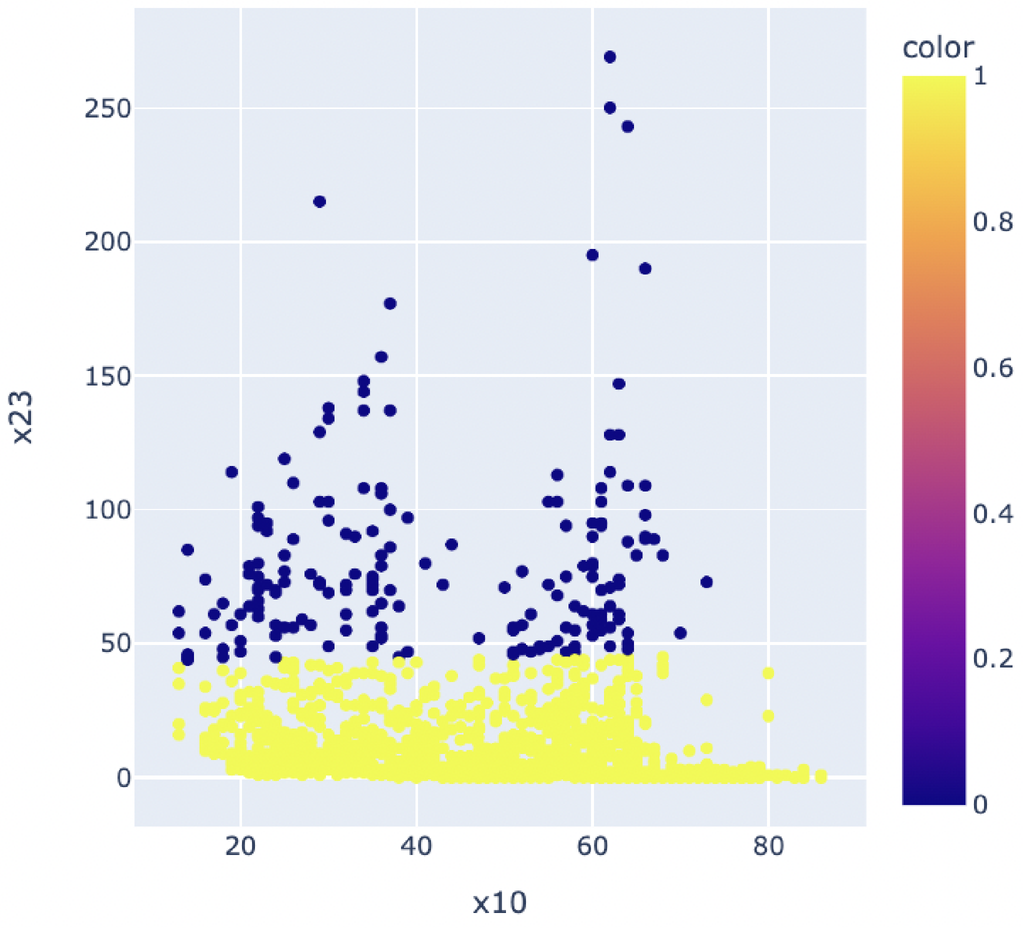

Figure 37: Formation of 2 clusters using the K-Means algorithm