Data Mining:
Practical Exercise Report.

A.S.Wijewardena

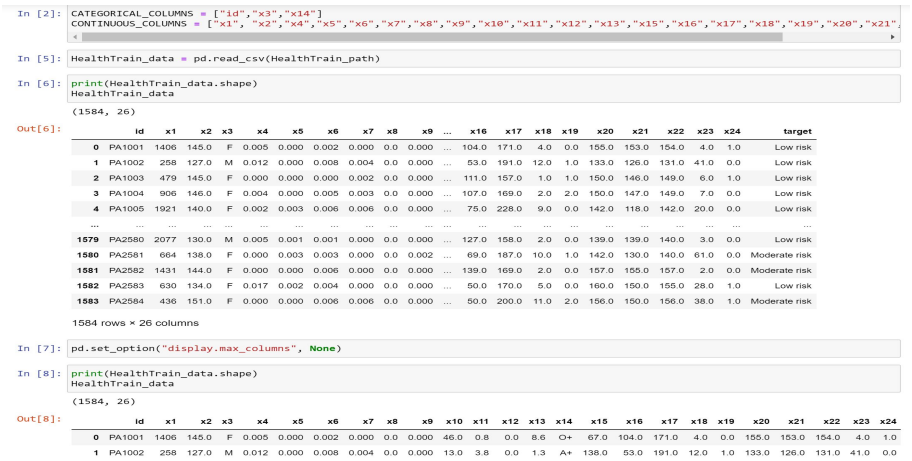
Abstract

Data mining is a broad subject which consists of various techniques and tasks. Throughout the report the implementation of chosen algorithms and tools in each stage of the Knowledge Discovery in Databases(KDD) process has been stated. Data summarizing, cleaning and pre-processing have been carried out according to the provided dataset. Three different classification models have been built and trained to identify which one would have a higher level of accuracy. Summarizing the results of the findings from this practical exercise, Two of the classification models displayed a higher level of accuracy. Furthermore, high-level insights have been observed and mentioned within the initial summarizing step.Finalizing with the conclusion of the conducted practical exercise. The code built for the exercise is attached under the appendix of the report.

Contents

1 Summary of Features

As per Tsai et al. (2012) features of variables in a dataset is crucial to understand the data and analysing. Variables often can be considered to be important features. Initial observations of the data set can be stated that it consists of 1584 rows(observations) and 26 columns(variables).



```
In [2]: CATEGORICAL_COLUMNS = ["id","x3","x14"]
        CONTINUOUS_COLUMNS = ["x1", "x2","x4","x5","x6","x7","x8","x9","x10","x11","x12","x13","x15","x16","x17","x18","x19","x20","x21",

In [5]: HealthTrain_data = pd.read_csv(HealthTrain_path)

In [6]: print(HealthTrain_data.shape)
        HealthTrain_data

        (1584, 26)

Out[6]:
```

	id	x1	x2	x3	x4	x5	x6	x7	x8	x9	...	x16	x17	x18	x19	x20	x21	x22	x23	x24	target
0	PA1001	1406	145.0	F	0.005	0.000	0.002	0.000	0.0	0.000	...	104.0	171.0	4.0	0.0	155.0	153.0	154.0	4.0	1.0	Low risk
1	PA1002	258	127.0	M	0.012	0.000	0.008	0.004	0.0	0.000	...	53.0	191.0	12.0	1.0	133.0	126.0	131.0	41.0	0.0	Low risk
2	PA1003	479	145.0	F	0.000	0.000	0.000	0.002	0.0	0.000	...	111.0	157.0	1.0	1.0	150.0	146.0	149.0	6.0	1.0	Low risk
3	PA1004	906	146.0	F	0.004	0.000	0.005	0.003	0.0	0.000	...	107.0	169.0	2.0	2.0	150.0	147.0	149.0	7.0	0.0	Low risk
4	PA1005	1921	140.0	F	0.002	0.003	0.006	0.006	0.0	0.000	...	75.0	228.0	9.0	0.0	142.0	118.0	142.0	20.0	0.0	Low risk
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1579	PA2580	2077	130.0	M	0.005	0.001	0.001	0.000	0.0	0.000	...	127.0	158.0	2.0	0.0	139.0	139.0	140.0	3.0	0.0	Low risk
1580	PA2581	664	138.0	F	0.000	0.003	0.003	0.000	0.0	0.002	...	69.0	187.0	10.0	1.0	142.0	130.0	140.0	61.0	0.0	Moderate risk
1581	PA2582	1431	144.0	F	0.000	0.000	0.006	0.000	0.0	0.000	...	139.0	169.0	2.0	0.0	157.0	155.0	157.0	2.0	0.0	Moderate risk
1582	PA2583	630	134.0	F	0.017	0.002	0.004	0.000	0.0	0.000	...	50.0	170.0	5.0	0.0	160.0	150.0	155.0	28.0	1.0	Low risk
1583	PA2584	436	151.0	F	0.000	0.000	0.006	0.006	0.0	0.000	...	50.0	200.0	11.0	2.0	156.0	150.0	156.0	38.0	1.0	Moderate risk

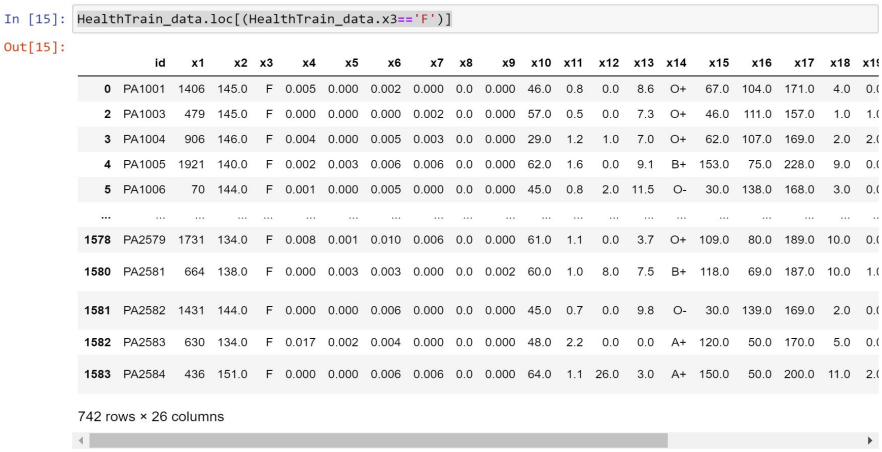
1584 rows × 26 columns

```
In [7]: pd.set_option("display.max_columns", None)

In [8]: print(HealthTrain_data.shape)
        HealthTrain_data

        (1584, 26)

Out[8]:
```

	id	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24
0	PA1001	1406	145.0	F	0.005	0.000	0.002	0.000	0.0	0.000	46.0	0.8	0.0	8.6	O+	67.0	104.0	171.0	4.0	0.0	155.0	153.0	154.0	4.0	1.0
1	PA1002	258	127.0	M	0.012	0.000	0.008	0.004	0.0	0.000	13.0	3.8	0.0	1.3	A+	138.0	53.0	191.0	12.0	1.0	133.0	126.0	131.0	41.0	0.0

Figure 1: Initial look at the data set.

It is a combination of categorical and numerical data. Patient's id, X3 and X14 are categorical data whereas the remaining variables are numerical(continous) data. The target label is also visible which represents the health risk levels as moderate risk, low risk and high risk. Also observe that the "Adult_data.shape" returns the the number of records and attributes in the brackets right below the code.Considering missing values and outliers at this stage, there are some variables which include missing values, through further summarising this could be identified.
Further observations of the the data, shows that the variable x3 represensts the patient's gender. It could be recognized that there are 742 female(F) records and 842 male(M) records in the data set.



```
In [15]: HealthTrain_data.loc[(HealthTrain_data.x3=='F')]

Out[15]:
```

	id	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x15
0	PA1001	1406	145.0	F	0.005	0.000	0.002	0.000	0.0	0.000	46.0	0.8	0.0	8.6	O+	67.0	104.0	171.0	4.0	0.(
2	PA1003	479	145.0	F	0.000	0.000	0.000	0.002	0.0	0.000	57.0	0.5	0.0	7.3	O+	46.0	111.0	157.0	1.0	1.(
3	PA1004	906	146.0	F	0.004	0.000	0.005	0.003	0.0	0.000	29.0	1.2	1.0	7.0	O+	62.0	107.0	169.0	2.0	2.(
4	PA1005	1921	140.0	F	0.002	0.003	0.006	0.006	0.0	0.000	62.0	1.6	0.0	9.1	B+	153.0	75.0	228.0	9.0	0.(
5	PA1006	70	144.0	F	0.001	0.000	0.005	0.000	0.0	0.000	45.0	0.8	2.0	11.5	O-	30.0	138.0	168.0	3.0	0.(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1578	PA2579	1731	134.0	F	0.008	0.001	0.010	0.006	0.0	0.000	61.0	1.1	0.0	3.7	O+	109.0	80.0	189.0	10.0	0.(
1580	PA2581	664	138.0	F	0.000	0.003	0.003	0.000	0.0	0.002	60.0	1.0	8.0	7.5	B+	118.0	69.0	187.0	10.0	1.(
1581	PA2582	1431	144.0	F	0.000	0.000	0.006	0.000	0.0	0.000	45.0	0.7	0.0	9.8	O-	30.0	139.0	169.0	2.0	0.(
1582	PA2583	630	134.0	F	0.017	0.002	0.004	0.000	0.0	0.000	48.0	2.2	0.0	0.0	A+	120.0	50.0	170.0	5.0	0.(
1583	PA2584	436	151.0	F	0.000	0.000	0.006	0.006	0.0	0.000	64.0	1.1	26.0	3.0	A+	150.0	50.0	200.0	11.0	2.(

742 rows × 26 columns

Figure 2: Accessing data with F as gender.

Obtaining summary statistics of the data, I have explored both categorical and numerical variables. It is discovered that the some of the numerical variables consist of missing values, whereas there are no missing values among the categorical variables.
The categorical variables x3 consists of 2 unique values and x14 of 8 unique values. The data in variable x14 seems to be blood type of each patient, corresponding to their own id.

```
Including sum of missing values of each categorical variable
              id    x3   x14
count       1584  1584  1584
unique      1584     2     8
top       PA1001     M    O+
freq           1   842   410
missing        0     0     0
```

Figure 3: Categorical Variables.

Basic visualisations of the initial data have been done during the code. Some of what considered note worthy observations have been included below.Numerical data visualizations are conducted somewhat different than visualizing categorical data. Since there are 22 continous variables the created plots are vast and provides certain outlooks about the data. Noteworthy, through visualizations it can be seen that variable x8 seems to be only related to high risk targets. x24 seems to have higher levels of density among low risk and moderate risk targets.

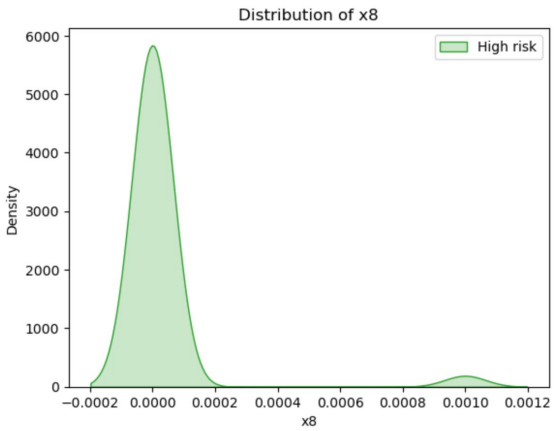


Figure 4: x8 plot

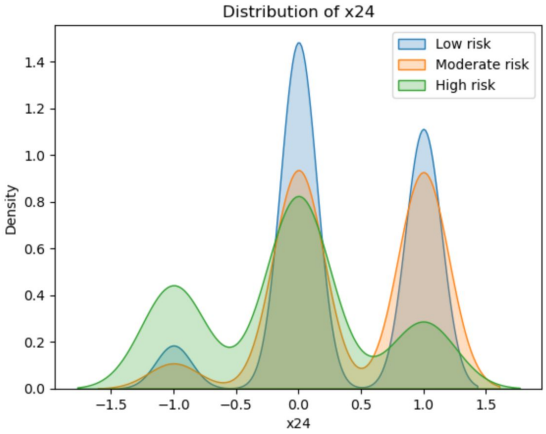


Figure 5: x24 plot

Plotting categorical variables, through observation it can be stated that there is an higher count of male low risk patients compared to female low risk patients. Although female moderate risk patients seems to be greater as opposed to males. Considering variable x14 it indicates that O+ consists of the highest count for low risk, A+ represents the highest count for high risk patients.

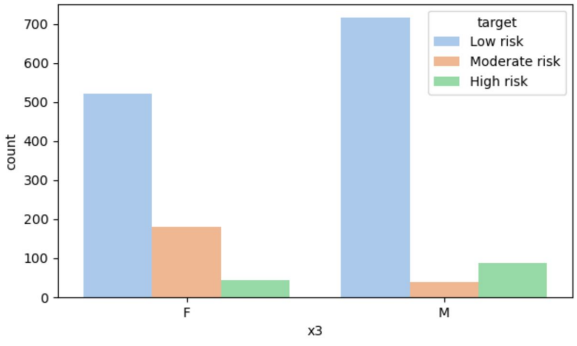


Figure 6: x3 plot

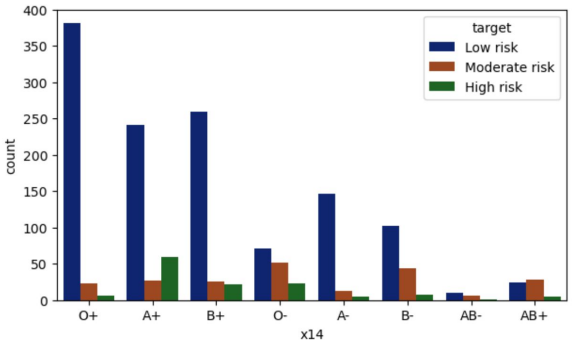


Figure 7: x14 plot

The count of the three targets among the entire data set is plotted below. It can be recognized that the amount of low risk patients are much higher than the other two target classes.

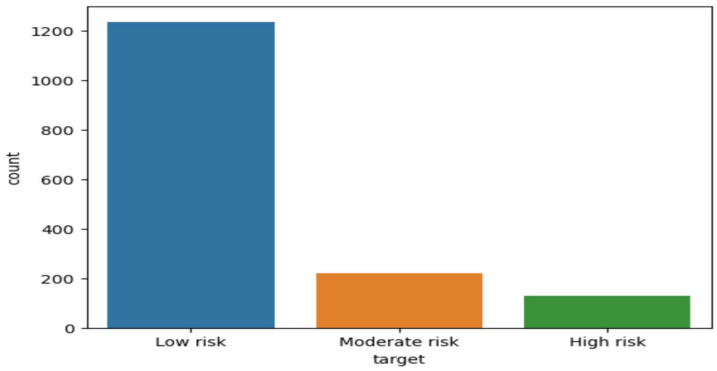


Figure 8: Distribution of the risk levels among the data set

Further, visualizations have been done throughout the code, consists of box, scatter plots among variables.

2 Data Cleaning & pre processing

2.1 Data Cleaning

Data cleaning and pre processing is a necessary task to conduct in order to obtain a clear , more accurate data set. In the study Van den Broeck et al. (2005) explains about four different steps that should be ideally taken during the cleaning stage. More often than so it depends on the data set.
In my practical exercise i have focused on identifying which variables consists of missing values, visualizing them and dropping the rows which contains them.

```
The sum of the counts of the missing values in each variable.
id         0
x1         0
x2         0
x3         0
x4         0
x5        44
x6         0
x7         0
x8        17
x9         0
x10       27
x11        0
x12        0
x13        0
x14        0
x15        0
x16        0
x17        0
x18        0
x19        0
x20        0
x21        0
x22        0
x23        0
x24        0
target     0
dtype: int64
```

Figure 9: sum of counts of the missing values

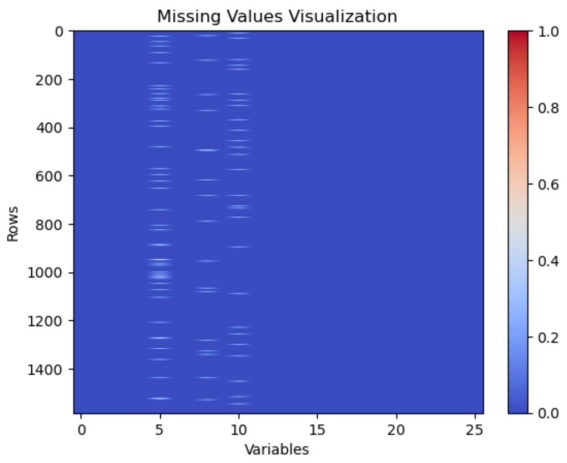


Figure 10: Missing Values Visualization

Once missing values are dealt with implemented the Z score method to detect outliers in the data set. The threshold was indicated as three, due to the value being robust and also assumes a considerable level of normality in the data. Outliers for each numerical variable was plotted using box plots. Filtered the data accordingly to ensure that rows that contain missing values and outliers are retained in a new data frame. As a third step, i removed any duplicate values in the data set.

```
Filtering to ensure only those rows that are not outliers & missing values are
retained in a new data frame for further coding
(1252, 22)
```

```
deduplicated_filtered_data = filtered_data.drop_duplicates()
print(deduplicated_filtered_data.shape)
```

```
(1251, 22)
```

Figure 11: Filtered data frame & removal of duplicates.

2.2 Data Pre-processing

Data cleaning can be identified as a sub step of data pre-processing. After the above tasks are carried out during the pre-processing stage, focused on encoding variables and feature selection. Distribution plots for categorical variables were created. Categorical variables were also encoded. All steps taken in the code.
Feature selection, as indexes decided that all columns would be considered since the understanding of the some data in the columns are vague, while some can be understood clearly. Correlation in the variables were calculated and plotted using heat maps for clearer visualizations.

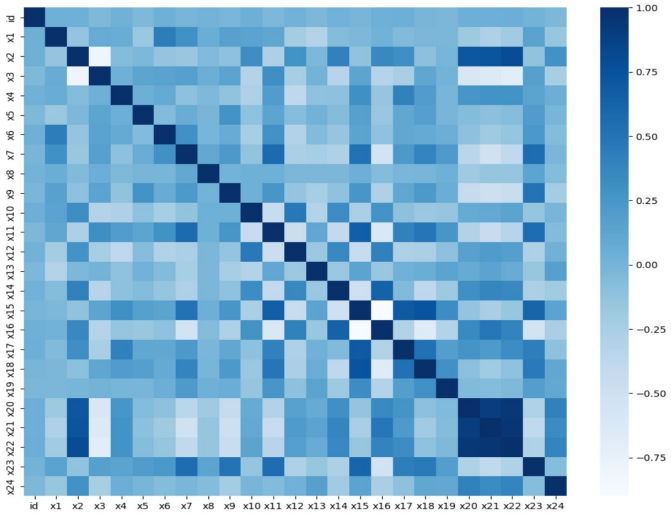


Figure 12: correlation visualization.

Addressing the fact that the codes used to conduct the data cleaning and pre-processing are all under the appendix of this report.

3 Supervised model training, Evaluation, predicted labels

Before building models, the data set obtained after the cleaning and preprocessing tasks, were split in to two sets, training set and validation set. The validation set consists of 20% of the data set and remaining 80% dedicated as the training set. Since a test data set was provided separately, it was two part split.

3.1 Models trained

For the purpose of the practical exercise, three classification models were created and utilized. Linear SVC, Naive Bayes and K-Nearest Neighbour(KNN) are the algorithms used.
Linear Support Vector Classification is a one of the popular classification methods in data mining. It is known to be quite effective and performs well computationally. In the past Linear SVC did have performance issues due to the computing process being rather time-consuming but the research Ho and Lin (2012) conducted around 2012/2013 discusses the improvements of it, one being the faster computation.

The prediction results of the built classifier on the training data set consisted of a number of outputs respective to the rows od the set. values were three different numbers representing the three target classes respectively.

```
y_predict = Classifiersvc.predict(train_X_encoded)
print(y_predict)
```

```
[1 1 1 2 2 1 1 1 1 2 1 1 2 1 1 2 1 2 1 1 1 1 0 1 1 1 1 0 2 1 1 2 1 1 1 1 1
 1 1 2 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1
 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1
 1 2 1 1 0 0 1 1 1 1 1 1 1 0 1 2 1 2 2 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 2 1 2
 1 2 1 2 1 1 0 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 2 1 1 2 2 0 1 1 1 1 2 1 1 1
 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 2 1
 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1
 2 2 1 1 2 1 1 1 1 1 1 0 1 1 0 1 1 1 2 1 1 1 1 1 2 1 1 1 0 1 1 2 2 1 1 1 1
 1 2 1 1 1 1 0 1 1 1 2 1 1 0 1 2 1 1 1 1 1 0 1 0 1 1 1 1 0 2 1 1 1 1 2 1 1
 1 1 1 1 1 1 2 0 1 1 0 1 1 2 1 1 1 1 2 2 1 0 1 1 1 2 1 1 1 1 2 2 0 1 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1
 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 2
 2 1 2 2 1 1 1 1 1 1 1 1 0 1 1 1 2 2 1 1 1 1 0 1 2 1 1 1 1 0 1 2 1 0 0 1 1
 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 0 0 2 1 0 2 1 1 1 0 2 1 2 1 1 2
 0 1 1 2 1 0 1 1 1 1 1 0 1 0 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1
 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1 1 0 1 0 2 0 1
 1 1 0 1 1 2 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 2 1 0 1 2 2 1 1 2
 0 1 2 1 1 1 0 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 0 1 0 1 1 0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 1 1
 1 1 0 1 1 1 1 1 1 1 2 1 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1
 1 1 1 2 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 2 1 1 1 1 1 1 1 1 1 1 2 1
 2 1 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 0 1 1 1 2 1 1 1 1
 1 2 1 2 0 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1
 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2 1 1 0 2 1 1 1 1 1 1 1 1
 1 1 1 0 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 0 1 1 1 1 1 1 2 1 0 2 1 1
 1 1 2 2 1 1 2 1 1 2 1 1 1 0 2 2 1 2 1 0 1 1 1 1 1 0 1 2 0 1 2 1 0 1 1 1 1
 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1
 1 2 1 1 1 1 1 0 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 2 1
 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 2 1 1 1 2 1 1 1 1 1 0 2 0 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 1 1 1 1 0 2 1 1 0 1 2 0 1 1 1 0 1 2 2 1 1 1 1 2 1 1 1 1 1
 1 1 1 2 1 1 0 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 2 1 1 1 2
 1 1 1 1 1 2 1 1 2 1 1 1 2 0 1 1 1 2 1 2 1 1 2 0 1 0 1 1 1 1 1 1 1 1 0 2 1
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 0 2 1 2 1 2 1 1 1 1 1
 1 1 1 1 1 1 1 1 1]
```

Figure 13: Linear SVC prediction. (training data set).

Next model built was the **Naive Bayes Classifier**. Due to its' high performance level and productivity considered it as one option. Yang (2018) states that this particular model is high in demand due to its' applicability to extensive domains of classification models.
k-nearest neighbour(KNN), the third model developed in this practical exercise. KNN holds a reputation of being computationally consuming. Despite that there have been many studies conducted relating to the above issue. The study Wu et al. (2002) researches the importance of pre-processed data being used for KNN, seems to have a better speed at computation.

```
from sklearn.naive_bayes import GaussianNB

naivebayes = GaussianNB()
naivebayes.fit(train_X_encoded.toarray(), train_y_encoded);
```

Figure 14: Naive Bayes Classifier

```
class KNN(BaseEstimator, ClassifierMixin):
    def __init__(self):
        pass

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.classes_ = unique_labels(y)
        self._X = X
        self._y = y
        return self

    def predict(self, X):
        check_is_fitted(self)
        X = check_array(X)
        closest = np.argmin(euclidean_distances(X, self._X), axis=1)
        return self._y[closest]
Classifier_KNN = KNN()
Classifier_KNN.fit(train_X_encoded.toarray(), train_y_encoded);
```

Figure 15: KNN

Running the linear svc model with the validation data set did require additional coding to match up the mixmatch of feature counts in the training set and the validation set. Encoding was done. The prediction output generated was considerabley interpretable. Although the accuracy level was below 30%.

```
accuracy = accuracy_score(val_y_encoded, predictions_Valset)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
[1 2 0 1 0 2 0 1 1 2 2 2 2 2 2 1 0 2 2 2 2 1 1 1 2 1 0 2 0 0 2 2 0 1 0 1
 1 2 0 2 2 1 1 0 0 2 2 0 0 0 0 2 1 2 0 1 0 0 0 1 0 2 2 0 0 1 2 1 0 2 2 2 2
 1 2 0 0 1 1 1 0 0 2 2 0 1 2 0 1 2 2 0 2 2 2 0 0 1 2 2 2 2 2 2 2 1 1 0 0 2
 1 1 2 2 1 2 1 2 0 1 2 2 2 0 0 2 2 0 0 2 0 0 0 1 2 2 1 0 0 2 0 0 1 0 0 2 2
 1 1 1 2 1 2 0 1 1 1 2 2 1 1 1 2 2 2 0 1 2 1 0 2 1 2 0 2 1 1 2 0 0 1 1 1 2
 1 1 2 2 0 0 2 2 1 0 2 2 2 1 2 0 0 0 1 1 0 2 2 1 0 0 2 1 1 1 2 1 0 2 1 1 2
 2 2 1 0 2 2 1 2 2 0 2 1 0 2 2 1 0 1 0 2 0 2 2 2 0 2 1 2 2 1 0 2 2 1 1 2 0
 0 2 1 1 2 0 2 2 0 0 1 2 2 1 2 2 2 0 0 2 0 0 2 2 1 2 2 0 2 2 1 0 0 0 1 0 1
 2 2 0 2 2 0 2 2 1 0 0 0 0 1 2 2 2 2 2 0 1]
Accuracy: 24.92%
```

Figure 16: Linear SVC prediction. (validation data set).

3.2 Evaluation of models

Evaluation of each models was solely based on the accuracy percentage of each model performed using the validation data set(vds).

```
from sklearn.metrics import accuracy_score
classifiers = [Classifiersvc, naivebayes, Classifier_KNN]
scores = [accuracy_score(clf.predict(val_X_encoded.toarray()), val_y_encoded) fo
index = np.argmax(scores)
print(scores)
print(classifiers[index])
print(scores[index])
print("Highest Accuracy percentage: {:.2f}%".format(scores[index]* 100))
```

```
[0.24921135646687698, 0.7476340694006309, 0.6214511041009464]
GaussianNB()
0.7476340694006309
Highest Accuracy percentage: 74.76%
```

Figure 17: Accuracy percentages of model predictions.(vds)

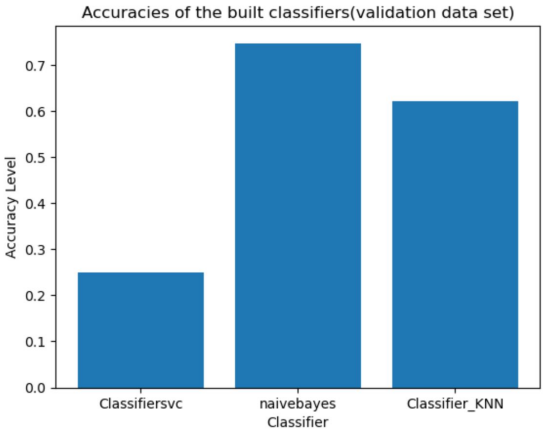


Figure 18: Classifier accuracy level.(vds)

According to the above it is clear that, the Naive Bayes classifier held the highest percentage, next in line is KNN and lastly Linear SVC.The model with the highest accuracy percentage, which is the naive bayes classifier, precision and recall scores were calculated.

```
Accuracy: 74.76%
Precision: 24.92%
Recall: 33.33%
```

Figure 19: Accuracy, Precision and Recall percentages of Naive Bayes.(vds)

3.3 Predicted labels

Predicted labels for the validation data set and training data set using Linear SVC is presented under sub topic models trained.
As for the test data set provided, the predicted values using Linear SVC classification is displayed below. Note that it can be interpreted that 0,1,2 represents low risk, moderate risk and high risk respectively. They are properly labeled in the predictedTarget.csv file provided.

```
[1 0 2 2 2 0 0 2 2 2 1 2 0 1 0 0 2 2 2 2 1 1 2 2 0 1 2 2 0 1 0 0 2 0 0 0 0
 0 2 1 2 1 0 0 1 2 2 1 0 1 0 2 2 1 2 0 2 0 2 2 0 2 2 2 2 2 2 1 2 2 2 1 2 0
 0 1 0 2 2 2 0 1 2 0 2 1 0 0 2 0 2 2 1 2 2 2 2 0 2 0 2 0 1 1 0 2 0 0 0 0 1 1
 1 1 2 0 0 0 2 2 2 0 0 1 2 2 0 0 0 1 1 2 1 1 2 1 0 2 0 2 0 2 0 2 0 2 0 1 1
 1 2 2 0 2 1 0 0 2 1 2 0 2 2 1 1 2 2 2 0 0 2 2 0 0 0 0 0 2 0 1 0 0 1 0 0 2
 1 1 0 1 2 0 2 1 1 0 1 1 2 1 1 1 2 0 0 0 0 1 2 1 2 0 0 0 1 2 1 0 2 2 1 0 0
 1 1 2 1 2 1 2 2 2 2 1 2 1 2 2 1 2 2 0 0 1 1 1 1 2 0 0 0 1 2 1 1 1 2 2 2 1
 1 1 2 2 1 2 2 1 1 2 0 1 1 2 1 2 2 1 0 1 1 0 2 0 2 1 1 2 1 1 0 2 2 1 0 0 1
 1 1 2 2 2 1 2 1 0 2 2 0 0 1 1 0 2 0 2 0 2 2 0 2 1 2 0 0 1 2 1 2 2 2 2 2 2
 0 1 0 1 0 2 0 1 1 2 1 0 0 2 1 0 0 0 2 0 2 2 1 0 2 0 0 0 0 2 1 1 0 1 0 2 1
 2 1 1 0 2 1 1 2 0 1 2 1 1 2 2 0 0 2 2 1 0 1 2 2 1 1 2 0 0 2 2 0 2 2 1 0 0
 2 2 0 0 2 0 1 2 1 1 1 0 2 0 2 0 2 2 0 2 1 0 2 2 2 2 1 1 0 2 2 2 1 0 0 1 0
 1 2 0 0 2 0 1 2 2 1 0 2 0 1 2 2 2 0 0 2 2 1 0 0 1 1 0 1 0 1 2 2 2 0 2 1 2
 1 2 1 0 2 0 2 0 1 0 2 1 0 2 0 1 2 1 0 2 0 2 1 2 0 1 2 2 2 2 2 2 0 0 0 1 2 2
 1 0 2 2 0 1 2 0 1 2]
```

Figure 20: Linear SVC prediction. (test data set).

4 Unsupervised Clustering

Unsupervised clustering was done using the main data set which is the healthtrain data set.
Steps in exploring the data were done once again.
Created kde plots for all the variables in the data set.

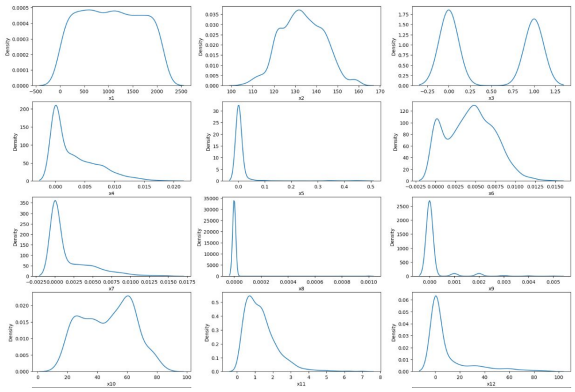


Figure 21: kdeplots

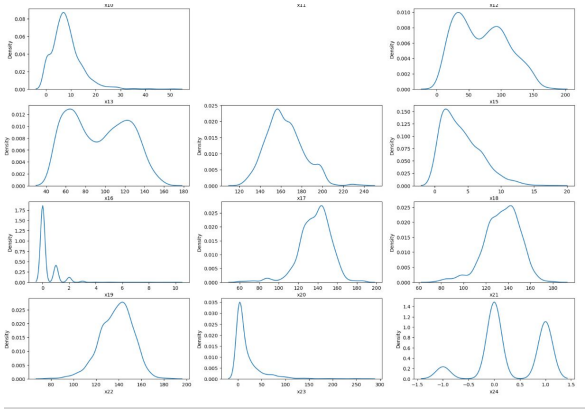


Figure 22: kdeplots

Considering how plots are skewed and the variation across all the plots, redefined just the columns needed to create clusters.
Below presented are the KDE plots for the chosen subset of variables.

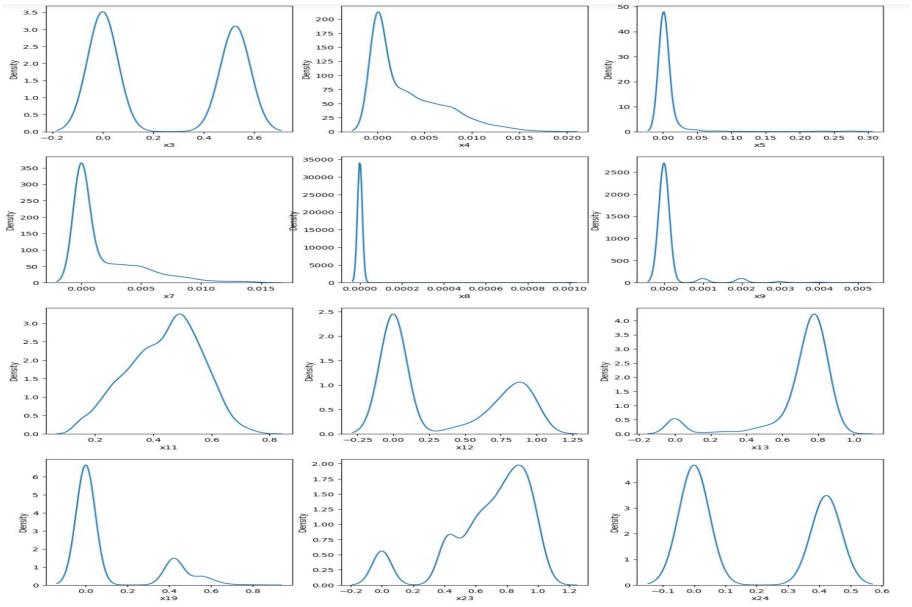


Figure 23: KDE plots for the chosen subset of variables

Heatmaps of the correlations of the columns in the dataset.

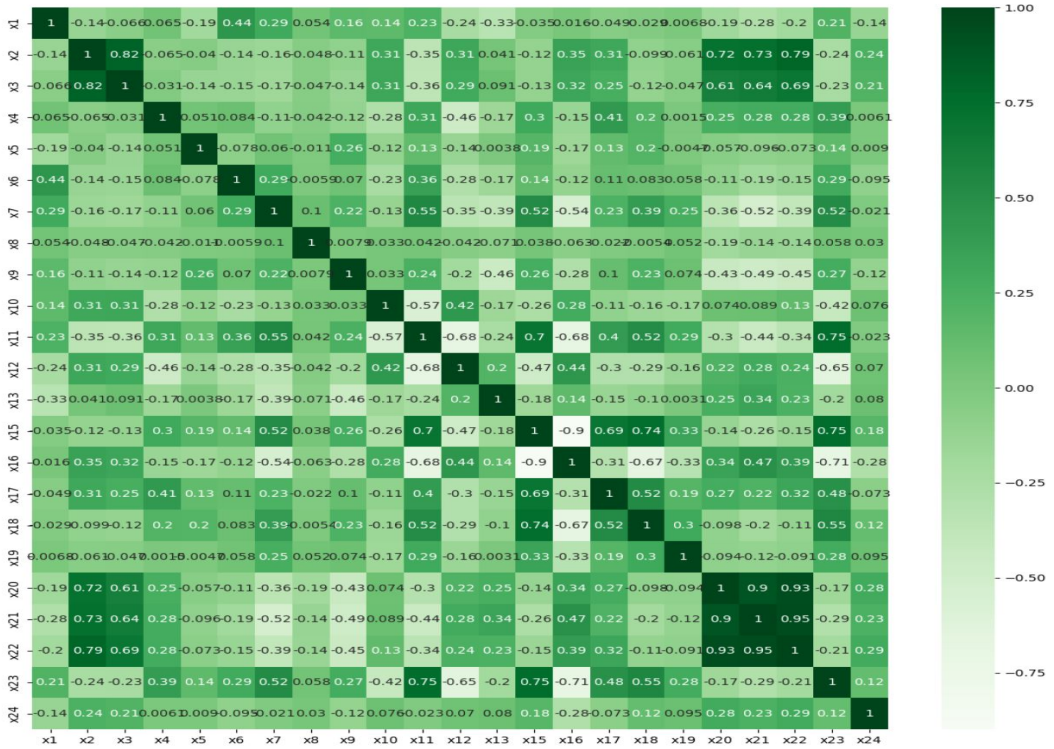


Figure 24: Heatmaps of the correlations of the columns

5 Conclusion

The healthtrain data set is a large data set, which required summarization of it initially, identifying features. The cleaning process conducted removed missing values, filtered out detected outliers and dropped duplicated values. The pre processing stage, mainly focused on encoding categorical variables since it was vital for building classification models.Considering the classification models built , Linear SVC did have a considerably lower accuracy rate, whereas KNN classifier and Naive Bayes classifier did obtain a higher level of accuracy, when the models were performed using the validation data set. Out of the two models it could be recognized the Naive Bayes classification performed well due to its' accuracy percentage(74.76%). Countering in the output state of the models, LinearSVC did have a more interpretable aspect to it, although the accuracy was below 30%.

In conclusion, KNN algorithm while the implementation is easy it is known to be computationally expensive" in regards to when handling large data sets. Linar SVC is an much more efficient approach since its computationally reasonable and works well with linearly separable classes. As for the Naive Bayes classifier holds a higher level of performance, only disadvantage to be considered here is that this particular model makes the assumption that the variables(features) are conditionally independent.

There are various other classification models that could be explored with the data set further. Decision trees, Logistic regression and even deep learning models like Convolutional Neural Networks also can be utilized to develop a health condition classifier.

References

Ho, C.-H. and Lin, C.-J. (2012). Large-scale linear support vector regression. *The Journal of Machine Learning Research*, 13(1):3323–3348.

Tsai, C.-F., Lu, Y.-H., and Yen, D. C. (2012). Determinants of intangible assets value: The data mining approach. *Knowledge-Based Systems*, 31:67–77.

Van den Broeck, J., Argeseanu Cunningham, S., Eeckels, R., and Herbst, K. (2005). Data cleaning: detecting, diagnosing, and editing data abnormalities. *PLoS medicine*, 2(10):e267.

Wu, Y., Ianakiev, K., and Govindaraju, V. (2002). Improved k-nearest neighbor classification. *Pattern recognition*, 35(10):2311–2318.

Yang, F.-J. (2018). An implementation of naive bayes classifier. In *2018 International conference on computational science and computational intelligence (CSCI)*, pages 301–306. IEEE.

A Appendix:Jupyter Notebook

The code and outputs have in under the appendix in the following pages.

100404197DataMining

May 17, 2023

[1]:
```
import pandas as pd
import numpy as np
pd.reset_option("^display")
pd.set_option('display.max_rows', 10)
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup complete.")
import matplotlib.image as mpimg
```

```
Setup complete.
```

[2]:
```
HealthTrain_path = "C:/Users/JT/DataSetsforDM/health_train.csv"
```

[3]:
```
CATEGORICAL_COLUMNS = ["id","x3","x14"]
CONTINUOUS_COLUMNS = ["x1",␣
 ↪"x2","x4","x5","x6","x7","x8","x9","x10","x11","x12","x13","x15","x16","x17","x18","x19","x
```

[4]:
```
HealthTrain_data = pd.read_csv(HealthTrain_path)
```

[5]:
```
print(HealthTrain_data.shape)
HealthTrain_data
```

```
(1584, 26)
```

[5]:

	id	x1	x2	x3	x4	x5	x6	x7	x8	x9	…	\
0	PA1001	1406	145.0	F	0.005	0.000	0.002	0.000	0.0	0.000	…	
1	PA1002	258	127.0	M	0.012	0.000	0.008	0.004	0.0	0.000	…	
2	PA1003	479	145.0	F	0.000	0.000	0.000	0.002	0.0	0.000	…	
3	PA1004	906	146.0	F	0.004	0.000	0.005	0.003	0.0	0.000	…	
4	PA1005	1921	140.0	F	0.002	0.003	0.006	0.006	0.0	0.000	…	
…	…	…	…	..	…	…	…	…	…	…	…	
1579	PA2580	2077	130.0	M	0.005	0.001	0.001	0.000	0.0	0.000	…	
1580	PA2581	664	138.0	F	0.000	0.003	0.003	0.000	0.0	0.002	…	
1581	PA2582	1431	144.0	F	0.000	0.000	0.006	0.000	0.0	0.000	…	
1582	PA2583	630	134.0	F	0.017	0.002	0.004	0.000	0.0	0.000	…	
1583	PA2584	436	151.0	F	0.000	0.000	0.006	0.006	0.0	0.000	…	

```
        x16    x17   x18  x19    x20    x21    x22   x23  x24         target
0     104.0  171.0   4.0  0.0  155.0  153.0  154.0   4.0  1.0       Low risk
1      53.0  191.0  12.0  1.0  133.0  126.0  131.0  41.0  0.0       Low risk
2     111.0  157.0   1.0  1.0  150.0  146.0  149.0   6.0  1.0       Low risk
3     107.0  169.0   2.0  2.0  150.0  147.0  149.0   7.0  0.0       Low risk
4      75.0  228.0   9.0  0.0  142.0  118.0  142.0  20.0  0.0       Low risk
…       …     …     …   …      …      …      …    …   …             …
1579  127.0  158.0   2.0  0.0  139.0  139.0  140.0   3.0  0.0       Low risk
1580   69.0  187.0  10.0  1.0  142.0  130.0  140.0  61.0  0.0  Moderate risk
1581  139.0  169.0   2.0  0.0  157.0  155.0  157.0   2.0  0.0  Moderate risk
1582   50.0  170.0   5.0  0.0  160.0  150.0  155.0  28.0  1.0       Low risk
1583   50.0  200.0  11.0  2.0  156.0  150.0  156.0  38.0  1.0  Moderate risk

[1584 rows x 26 columns]
```

[6]:
```
pd.set_option("display.max_columns", None)
```

[7]:
```
print(HealthTrain_data.shape)
HealthTrain_data
```

```
(1584, 26)
```

[7]:
```
           id    x1     x2 x3     x4     x5     x6     x7   x8     x9    x10  \
0     PA1001  1406  145.0  F  0.005  0.000  0.002  0.000  0.0  0.000  46.0
1     PA1002   258  127.0  M  0.012  0.000  0.008  0.004  0.0  0.000  13.0
2     PA1003   479  145.0  F  0.000  0.000  0.000  0.002  0.0  0.000  57.0
3     PA1004   906  146.0  F  0.004  0.000  0.005  0.003  0.0  0.000  29.0
4     PA1005  1921  140.0  F  0.002  0.003  0.006  0.006  0.0  0.000  62.0
…       …     …      …  ..    …      …      …      …   …    …      …
1579  PA2580  2077  130.0  M  0.005  0.001  0.001  0.000  0.0  0.000  72.0
1580  PA2581   664  138.0  F  0.000  0.003  0.003  0.000  0.0  0.002  60.0
1581  PA2582  1431  144.0  F  0.000  0.000  0.006  0.000  0.0  0.000  45.0
1582  PA2583   630  134.0  F  0.017  0.002  0.004  0.000  0.0  0.000  48.0
1583  PA2584   436  151.0  F  0.000  0.000  0.006  0.006  0.0  0.000  64.0

      x11   x12  x13 x14    x15    x16    x17   x18  x19    x20    x21    x22  \
0     0.8   0.0  8.6  O+   67.0  104.0  171.0   4.0  0.0  155.0  153.0  154.0
1     3.8   0.0  1.3  A+  138.0   53.0  191.0  12.0  1.0  133.0  126.0  131.0
2     0.5   0.0  7.3  O+   46.0  111.0  157.0   1.0  1.0  150.0  146.0  149.0
3     1.2   1.0  7.0  O+   62.0  107.0  169.0   2.0  2.0  150.0  147.0  149.0
4     1.6   0.0  9.1  B+  153.0   75.0  228.0   9.0  0.0  142.0  118.0  142.0
…   …   …   …  ..    …      …      …      …   …   …      …      …      …
1579  0.9   8.0  4.4  B-   31.0  127.0  158.0   2.0  0.0  139.0  139.0  140.0
1580  1.0   8.0  7.5  B+  118.0   69.0  187.0  10.0  1.0  142.0  130.0  140.0
1581  0.7   0.0  9.8  O-   30.0  139.0  169.0   2.0  0.0  157.0  155.0  157.0
1582  2.2   0.0  0.0  A+  120.0   50.0  170.0   5.0  0.0  160.0  150.0  155.0
1583  1.1  26.0  3.0  A+  150.0   50.0  200.0  11.0  2.0  156.0  150.0  156.0
```

```
       x23  x24         target
0      4.0  1.0       Low risk
1     41.0  0.0       Low risk
2      6.0  1.0       Low risk
3      7.0  0.0       Low risk
4     20.0  0.0       Low risk
…     …  …             …
1579   3.0  0.0       Low risk
1580  61.0  0.0  Moderate risk
1581   2.0  0.0  Moderate risk
1582  28.0  1.0       Low risk
1583  38.0  1.0  Moderate risk

[1584 rows x 26 columns]
```

[8]:
```
HealthTrain_data['x3']
```

[8]:
```
0       F
1       M
2       F
3       F
4       F
       ..
1579    M
1580    F
1581    F
1582    F
1583    F
Name: x3, Length: 1584, dtype: object
```

[9]:
```
HealthTrain_data.loc[:,['id','x3','x14','target']]
```

[9]:
```
           id x3 x14         target
0     PA1001  F  O+       Low risk
1     PA1002  M  A+       Low risk
2     PA1003  F  O+       Low risk
3     PA1004  F  O+       Low risk
4     PA1005  F  B+       Low risk
…      … ..  ..            …
1579  PA2580  M  B-       Low risk
1580  PA2581  F  B+  Moderate risk
1581  PA2582  F  O-  Moderate risk
1582  PA2583  F  A+       Low risk
1583  PA2584  F  A+  Moderate risk

[1584 rows x 4 columns]
```

[10]:
```
HealthTrain_data.loc[0:15,['id','x3','x14','target']]
```

[10]:
```
         id x3 x14         target
0   PA1001  F  O+       Low risk
1   PA1002  M  A+       Low risk
2   PA1003  F  O+       Low risk
3   PA1004  F  O+       Low risk
4   PA1005  F  B+       Low risk
..     … ..  ..            …
11  PA1012  F  O+       Low risk
12  PA1013  F  O-  Moderate risk
13  PA1014  M  A+       Low risk
14  PA1015  M  A+      High risk
15  PA1016  M  B-  Moderate risk

[16 rows x 4 columns]
```

[11]:
```
HealthTrain_data.loc[:,['id','x3','x6','x10','x14','x24','target']]
```

[11]:
```
           id x3     x6   x10 x14  x24         target
0     PA1001  F  0.002  46.0  O+  1.0       Low risk
1     PA1002  M  0.008  13.0  A+  0.0       Low risk
2     PA1003  F  0.000  57.0  O+  1.0       Low risk
3     PA1004  F  0.005  29.0  O+  0.0       Low risk
4     PA1005  F  0.006  62.0  B+  0.0       Low risk
…       … ..    …    …  ..  …             …
1579  PA2580  M  0.001  72.0  B-  0.0       Low risk
1580  PA2581  F  0.003  60.0  B+  0.0  Moderate risk
1581  PA2582  F  0.006  45.0  O-  0.0  Moderate risk
1582  PA2583  F  0.004  48.0  A+  1.0       Low risk
1583  PA2584  F  0.006  64.0  A+  1.0  Moderate risk

[1584 rows x 7 columns]
```

[12]:
```
HealthTrain_data.loc[(HealthTrain_data.x3=='F')]
```

[12]:
```
           id    x1     x2 x3     x4     x5     x6     x7   x8     x9   x10  \
0     PA1001  1406  145.0  F  0.005  0.000  0.002  0.000  0.0  0.000  46.0
2     PA1003   479  145.0  F  0.000  0.000  0.000  0.002  0.0  0.000  57.0
3     PA1004   906  146.0  F  0.004  0.000  0.005  0.003  0.0  0.000  29.0
4     PA1005  1921  140.0  F  0.002  0.003  0.006  0.006  0.0  0.000  62.0
5     PA1006    70  144.0  F  0.001  0.000  0.005  0.000  0.0  0.000  45.0
…       …     …      … ..    …      …      …      …   …      …    …
1578  PA2579  1731  134.0  F  0.008  0.001  0.010  0.006  0.0  0.000  61.0
1580  PA2581   664  138.0  F  0.000  0.003  0.003  0.000  0.0  0.002  60.0
1581  PA2582  1431  144.0  F  0.000  0.000  0.006  0.000  0.0  0.000  45.0
1582  PA2583   630  134.0  F  0.017  0.002  0.004  0.000  0.0  0.000  48.0
```

```
1583  PA2584   436  151.0  F  0.000  0.000  0.006  0.006  0.0  0.000  64.0

      x11   x12   x13 x14    x15    x16    x17   x18  x19    x20    x21  \
0     0.8   0.0   8.6  O+   67.0  104.0  171.0   4.0  0.0  155.0  153.0
2     0.5   0.0   7.3  O+   46.0  111.0  157.0   1.0  1.0  150.0  146.0
3     1.2   1.0   7.0  O+   62.0  107.0  169.0   2.0  2.0  150.0  147.0
4     1.6   0.0   9.1  B+  153.0   75.0  228.0   9.0  0.0  142.0  118.0
5     0.8   2.0  11.5  O-   30.0  138.0  168.0   3.0  0.0  162.0  157.0
…   …   …   …  ..    …    …    …   …  …    …    …
1578  1.1   0.0   3.7  O+  109.0   80.0  189.0  10.0  0.0  156.0  144.0
1580  1.0   8.0   7.5  B+  118.0   69.0  187.0  10.0  1.0  142.0  130.0
1581  0.7   0.0   9.8  O-   30.0  139.0  169.0   2.0  0.0  157.0  155.0
1582  2.2   0.0   0.0  A+  120.0   50.0  170.0   5.0  0.0  160.0  150.0
1583  1.1  26.0   3.0  A+  150.0   50.0  200.0  11.0  2.0  156.0  150.0

        x22   x23  x24         target
0     154.0   4.0  1.0       Low risk
2     149.0   6.0  1.0       Low risk
3     149.0   7.0  0.0       Low risk
4     142.0  20.0  0.0       Low risk
5     160.0   5.0  1.0       Low risk
…     …   …  …            …
1578  151.0  61.0  0.0       Low risk
1580  140.0  61.0  0.0  Moderate risk
1581  157.0   2.0  0.0  Moderate risk
1582  155.0  28.0  1.0       Low risk
1583  156.0  38.0  1.0  Moderate risk

[742 rows x 26 columns]
```

[13]:
```
HealthTrain_data.loc[(HealthTrain_data.x3=='M')]
```

[13]:
```
          id    x1     x2 x3     x4     x5     x6     x7   x8   x9   x10  x11  \
1     PA1002   258  127.0  M  0.012  0.000  0.008  0.004  0.0  0.0  13.0  3.8
7     PA1008   712  126.0  M  0.005  0.003  0.005  0.000  0.0  0.0  41.0  1.7
9     PA1010  1147  122.0  M  0.000  0.000  0.006  0.009  0.0  0.0  18.0  1.9
10    PA1011  1062  127.0  M  0.003  0.000  0.008  0.000  0.0  0.0  33.0  1.0
13    PA1014   561  128.0  M  0.001  0.000  0.010  0.003  0.0  0.0  37.0  2.8
…      …   …    …  ..    …    …    …   …  …   …   …  …
1574  PA2575   646  123.0  M  0.000  0.000  0.004  0.000  0.0  0.0  60.0  0.6
1575  PA2576  1790  121.0  M  0.000  0.002  0.002  0.015  0.0  0.0  61.0  2.0
1576  PA2577  1665  106.0  M  0.006  0.000  0.006  0.000  0.0  0.0  64.0  0.7
1577  PA2578  1234  125.0  M  0.002  0.000  0.004  0.001  0.0  0.0  30.0  1.1
1579  PA2580  2077  130.0  M  0.005  0.001  0.001  0.000  0.0  0.0  72.0  0.9

      x12   x13 x14    x15    x16    x17   x18  x19    x20    x21    x22  \
1     0.0   1.3  A+  138.0   53.0  191.0  12.0  1.0  133.0  126.0  131.0
```

```
7     0.0  21.4  A+  100.0   50.0  150.0   4.0  0.0  133.0  130.0  132.0
9     0.0   8.8  O+   44.0   91.0  135.0   4.0  0.0  126.0  117.0  121.0
10    0.0   7.6  A-   32.0  114.0  146.0   0.0  0.0  129.0  130.0  132.0
13    0.0  16.0  A+  136.0   55.0  191.0   9.0  1.0  129.0  130.0  129.0
…   …   …   ..   …    …   …   …   …   …    …    …
1574  0.0  10.4  O+   52.0   84.0  136.0   4.0  0.0  125.0  124.0  126.0
1575  0.0   4.9  A+   98.0   62.0  160.0   8.0  0.0  114.0  100.0  106.0
1576  0.0  12.6  O+   54.0  100.0  154.0   3.0  0.0  112.0  116.0  114.0
1577  3.0  11.0  O+   45.0  102.0  147.0   1.0  0.0  126.0  127.0  128.0
1579  8.0   4.4  B-   31.0  127.0  158.0   2.0  0.0  139.0  139.0  140.0

       x23  x24         target
1     41.0  0.0       Low risk
7     10.0  1.0       Low risk
9     19.0  1.0       Low risk
10     3.0  0.0       Low risk
13    37.0  0.0       Low risk
…    …  …            …
1574   2.0  1.0  Moderate risk
1575  59.0  0.0      High risk
1576  14.0 -1.0       Low risk
1577   3.0  0.0       Low risk
1579   3.0  0.0       Low risk

[842 rows x 26 columns]
```

[14]:
```
HealthTrain_data['id'].describe()
```

[14]:
```
count       1584
unique      1584
top       PA1001
freq           1
Name: id, dtype: object
```

[15]:
```
HealthTrain_data['x1'].describe()
```

[15]:
```
count    1584.000000
mean     1053.188131
std       615.996716
min         0.000000
25%       523.750000
50%      1049.500000
75%      1583.250000
max      2125.000000
Name: x1, dtype: float64
```

[16]:
```
HealthTrain_data['x3'].describe()
```

[16]:
```
count     1584
unique       2
top          M
freq       842
Name: x3, dtype: object
```

[17]:
```
HealthTrain_data['x14'].describe()
```

[17]:
```
count     1584
unique       8
top         O+
freq       410
Name: x14, dtype: object
```

[18]:
```
HealthTrain_data['x14'].unique()
```

[18]:
```
array(['O+', 'A+', 'B+', 'O-', 'A-', 'B-', 'AB-', 'AB+'], dtype=object)
```

[19]:
```
HealthTrain_data['x14'].value_counts()
```

[19]:
```
O+     410
A+     328
B+     307
A-     165
B-     154
O-     146
AB+     57
AB-     17
Name: x14, dtype: int64
```

[20]:
```
HealthTrain_data['x3'].value_counts()
```

[20]:
```
M    842
F    742
Name: x3, dtype: int64
```

[21]:
```
NumericalD= HealthTrain_data[CONTINUOUS_COLUMNS].describe()
print(NumericalD)
```

```
                x1           x2           x4           x5           x6  \
count  1584.000000  1584.000000  1584.000000  1540.000000  1584.000000
mean   1053.188131   133.297980     0.003169     0.009906     0.004347
std     615.996716    10.002632     0.003821     0.048627     0.002948
min       0.000000   106.000000     0.000000     0.000000     0.000000
25%     523.750000   126.000000     0.000000     0.000000     0.002000
50%    1049.500000   133.000000     0.002000     0.000000     0.004000
75%    1583.250000   141.000000     0.006000     0.003000     0.007000
max    2125.000000   160.000000     0.019000     0.477000     0.014000
```

```
                x7           x8           x9          x10          x11  \
count  1584.000000  1567.000000  1584.000000  1557.000000  1584.000000
mean      0.001854     0.000003     0.000157    47.094412     1.337816
std       0.002940     0.000050     0.000593    17.269621     0.899092
min       0.000000     0.000000     0.000000    13.000000     0.200000
25%       0.000000     0.000000     0.000000    32.000000     0.700000
50%       0.000000     0.000000     0.000000    49.000000     1.200000
75%       0.003000     0.000000     0.000000    61.000000     1.700000
max       0.015000     0.001000     0.005000    86.000000     7.000000

               x12          x13          x15          x16          x17  \
count  1584.000000  1584.000000  1584.000000  1584.000000  1584.000000
mean     10.008838     8.255240    70.409091    93.496843   163.905934
std      18.520206     5.784579    38.993892    29.593370    17.908749
min       0.000000     0.000000     3.000000    50.000000   122.000000
25%       0.000000     4.600000    36.000000    67.000000   152.000000
50%       0.000000     7.400000    68.000000    93.000000   162.000000
75%      11.000000    10.700000   100.000000   120.000000   174.000000
max      91.000000    50.700000   176.000000   158.000000   238.000000

               x18          x19          x20          x21          x22  \
count  1584.000000  1584.000000  1584.000000  1584.000000  1584.000000
mean      4.063763     0.324495   137.333965   134.542929   137.935606
std       2.950268     0.718499    16.461643    15.729735    14.622680
min       0.000000     0.000000    60.000000    73.000000    77.000000
25%       2.000000     0.000000   128.000000   125.000000   128.000000
50%       3.000000     0.000000   139.000000   136.000000   139.000000
75%       6.000000     0.000000   148.000000   145.250000   148.000000
max      18.000000    10.000000   187.000000   182.000000   186.000000

               x23          x24
count  1584.000000  1584.000000
mean     18.448232     0.309343
std      28.375002     0.615868
min       0.000000    -1.000000
25%       2.000000     0.000000
50%       7.000000     0.000000
75%      23.000000     1.000000
max     269.000000     1.000000
```

[22]:
```
for column in CONTINUOUS_COLUMNS:
    NumericalD.loc['missing',column] = HealthTrain_data[column].isnull().sum()
print("Including sum of missing values of each numerical variable")
print(NumericalD)
```

```
Including sum of missing values of each numerical variable
                x1           x2           x4           x5           x6  \
count  1584.000000  1584.000000  1584.000000  1540.000000  1584.000000
```

mean	1053.188131	133.297980	0.003169	0.009906	0.004347
std	615.996716	10.002632	0.003821	0.048627	0.002948
min	0.000000	106.000000	0.000000	0.000000	0.000000
25%	523.750000	126.000000	0.000000	0.000000	0.002000
50%	1049.500000	133.000000	0.002000	0.000000	0.004000
75%	1583.250000	141.000000	0.006000	0.003000	0.007000
max	2125.000000	160.000000	0.019000	0.477000	0.014000
missing	0.000000	0.000000	0.000000	44.000000	0.000000

	x7	x8	x9	x10	x11	\
count	1584.000000	1567.000000	1584.000000	1557.000000	1584.000000	
mean	0.001854	0.000003	0.000157	47.094412	1.337816	
std	0.002940	0.000050	0.000593	17.269621	0.899092	
min	0.000000	0.000000	0.000000	13.000000	0.200000	
25%	0.000000	0.000000	0.000000	32.000000	0.700000	
50%	0.000000	0.000000	0.000000	49.000000	1.200000	
75%	0.003000	0.000000	0.000000	61.000000	1.700000	
max	0.015000	0.001000	0.005000	86.000000	7.000000	
missing	0.000000	17.000000	0.000000	27.000000	0.000000	

	x12	x13	x15	x16	x17	\
count	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000	
mean	10.008838	8.255240	70.409091	93.496843	163.905934	
std	18.520206	5.784579	38.993892	29.593370	17.908749	
min	0.000000	0.000000	3.000000	50.000000	122.000000	
25%	0.000000	4.600000	36.000000	67.000000	152.000000	
50%	0.000000	7.400000	68.000000	93.000000	162.000000	
75%	11.000000	10.700000	100.000000	120.000000	174.000000	
max	91.000000	50.700000	176.000000	158.000000	238.000000	
missing	0.000000	0.000000	0.000000	0.000000	0.000000	

	x18	x19	x20	x21	x22	\
count	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000	
mean	4.063763	0.324495	137.333965	134.542929	137.935606	
std	2.950268	0.718499	16.461643	15.729735	14.622680	
min	0.000000	0.000000	60.000000	73.000000	77.000000	
25%	2.000000	0.000000	128.000000	125.000000	128.000000	
50%	3.000000	0.000000	139.000000	136.000000	139.000000	
75%	6.000000	0.000000	148.000000	145.250000	148.000000	
max	18.000000	10.000000	187.000000	182.000000	186.000000	
missing	0.000000	0.000000	0.000000	0.000000	0.000000	

	x23	x24
count	1584.000000	1584.000000
mean	18.448232	0.309343
std	28.375002	0.615868
min	0.000000	-1.000000
25%	2.000000	0.000000

```
50%          7.000000     0.000000
75%         23.000000     1.000000
max        269.000000     1.000000
missing      0.000000     0.000000
```

[23]:
```
CategoricalD= HealthTrain_data[CATEGORICAL_COLUMNS].describe()
print(CategoricalD)
```

```
            id    x3   x14
count     1584  1584  1584
unique    1584     2     8
top     PA1001     M    O+
freq         1   842   410
```

[24]:
```
for column in CATEGORICAL_COLUMNS:
    CategoricalD.loc['missing',column] = HealthTrain_data[column].isnull().sum()
print("Including sum of missing values of each categorical variable")
print(CategoricalD)
```

```
Including sum of missing values of each categorical variable
            id    x3   x14
count     1584  1584  1584
unique    1584     2     8
top     PA1001     M    O+
freq         1   842   410
missing      0     0     0
```

[25]:
```
print("Let's visualize the data!")
```

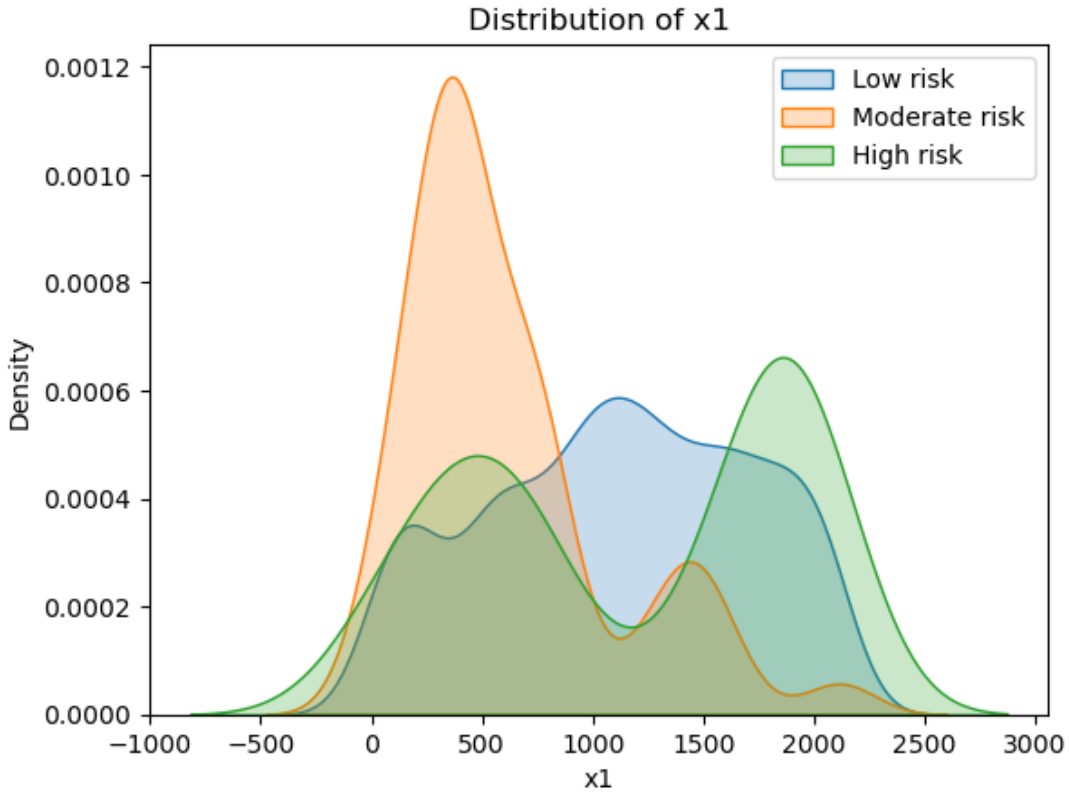
```
Let's visualize the data!
```

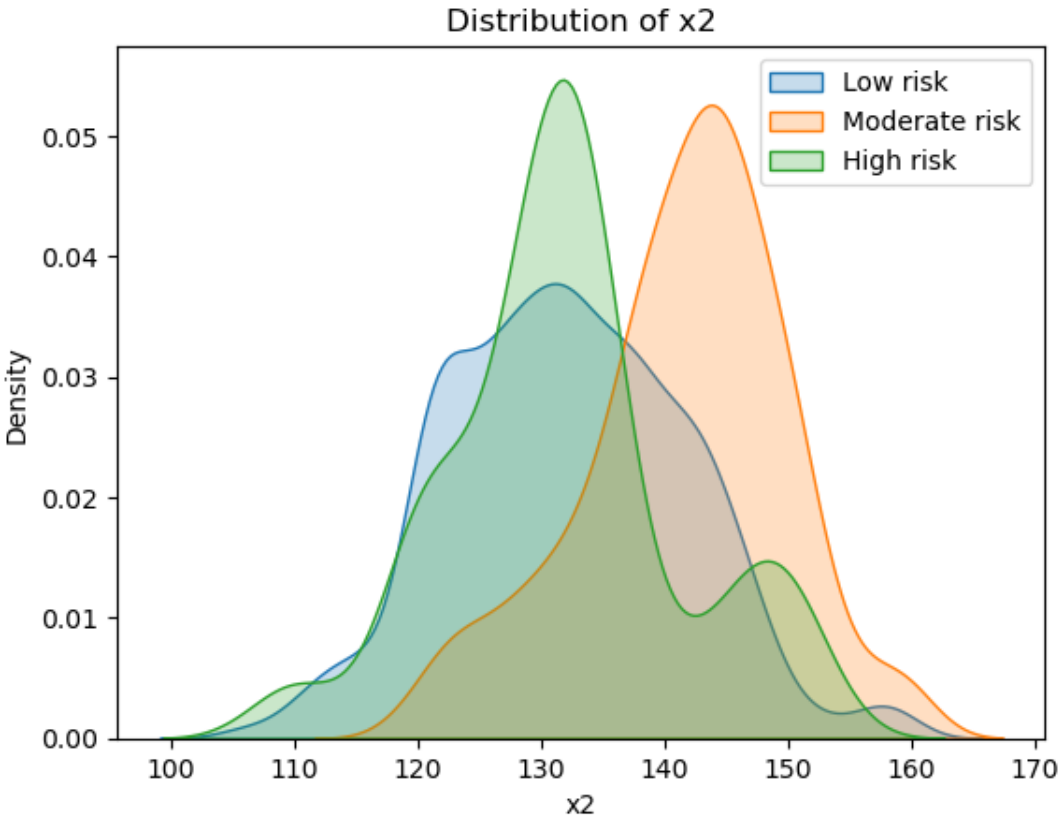
[26]:
```
HealthTrain_data_LR = HealthTrain_data.loc[(HealthTrain_data.target.str.
 ↪strip()=='Low risk')]
HealthTrain_data_MR = HealthTrain_data.loc[(HealthTrain_data.target.str.
 ↪strip()=='Moderate risk')]
HealthTrain_data_HR = HealthTrain_data.loc[(HealthTrain_data.target.str.
 ↪strip()=='High risk')]
```

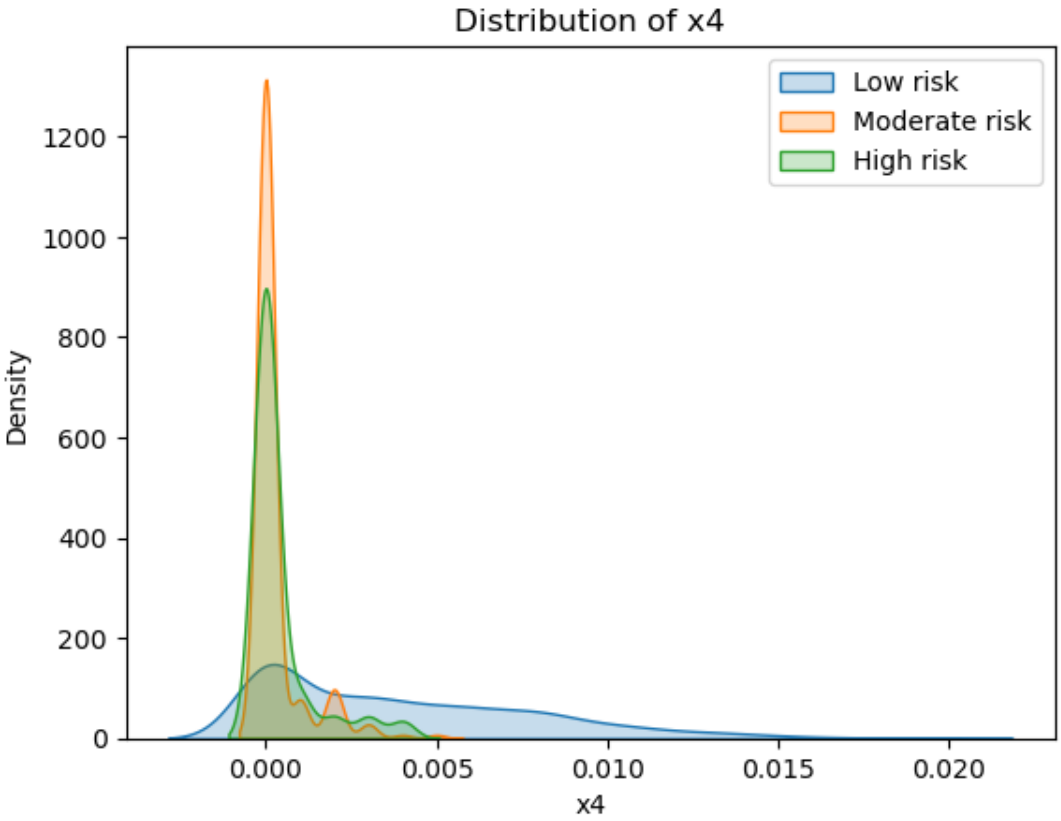
[27]:
```
#KDE plots for the continous varibales of the data set
for column in CONTINUOUS_COLUMNS:
    plt.figure()

    sns.kdeplot(data=HealthTrain_data_LR[column],label="Low risk", shade=True)
    sns.kdeplot(data=HealthTrain_data_MR[column],label="Moderate risk",␣
 ↪shade=True)
    sns.kdeplot(data=HealthTrain_data_HR[column],label="High risk", shade=True)

    plt.legend()
```

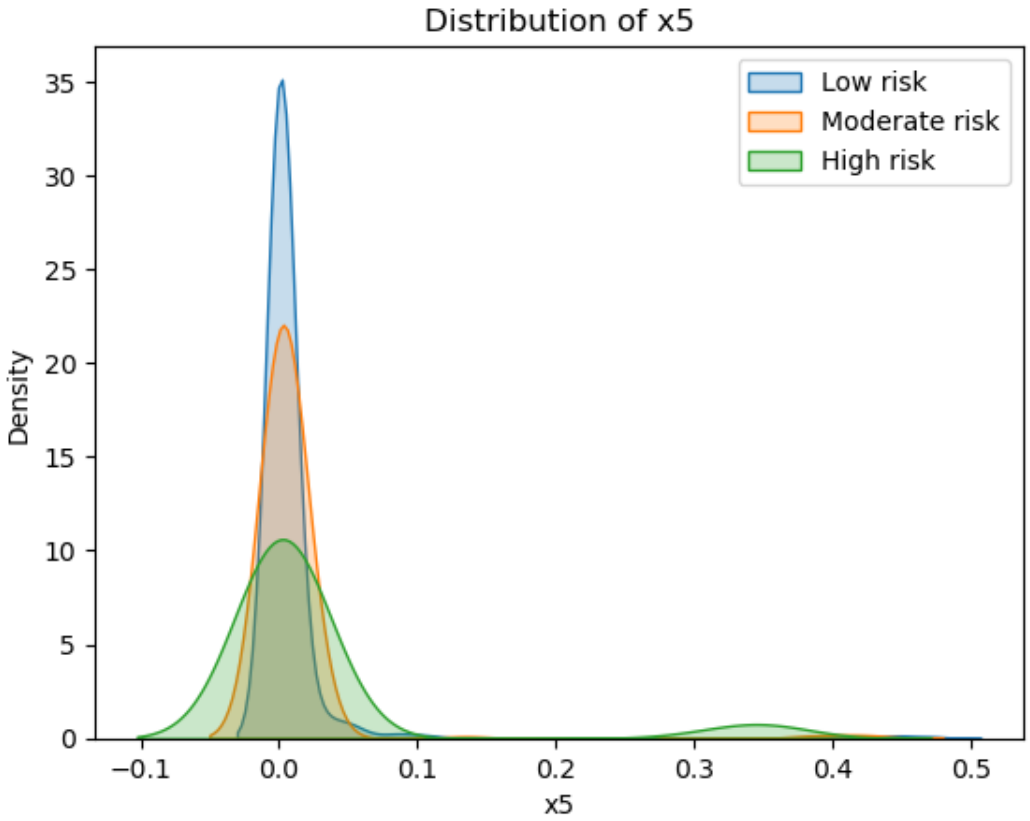
```
    plt.title("Distribution of " +column)

    plt.savefig(column+'.jpg')

   # plt.show()
```

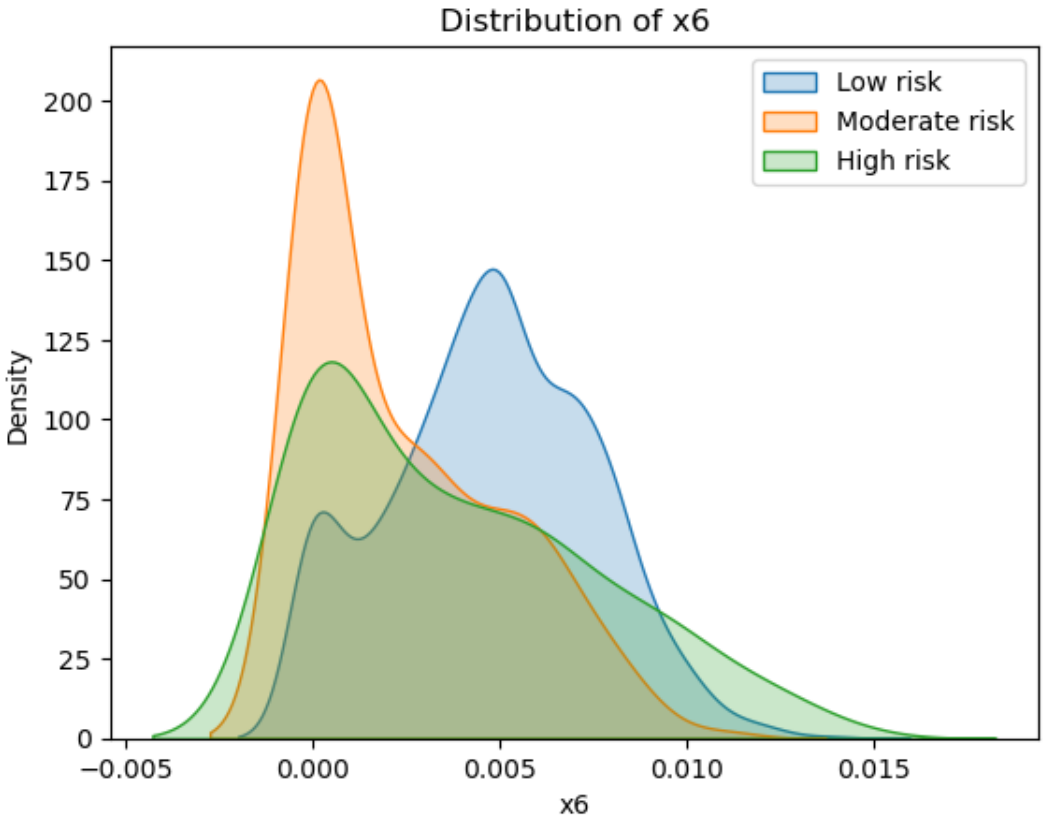
```
C:\Users\JT\anaconda3\lib\site-packages\seaborn\distributions.py:316:
UserWarning: Dataset has 0 variance; skipping density estimate. Pass
`warn_singular=False` to disable this warning.
  warnings.warn(msg, UserWarning)
C:\Users\JT\anaconda3\lib\site-packages\seaborn\distributions.py:316:
UserWarning: Dataset has 0 variance; skipping density estimate. Pass
`warn_singular=False` to disable this warning.
  warnings.warn(msg, UserWarning)
C:\Users\JT\AppData\Local\Temp\ipykernel_17320\3408765006.py:3: RuntimeWarning:
More than 20 figures have been opened. Figures created through the pyplot
interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and
may consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  plt.figure()
```

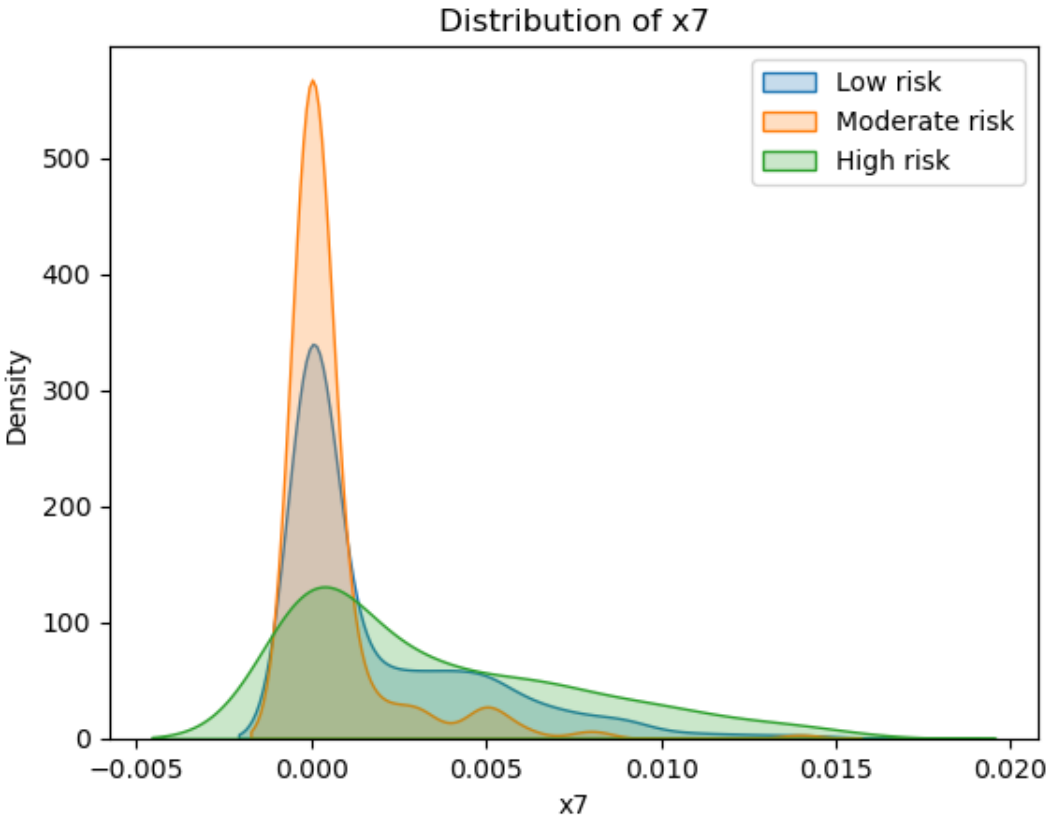


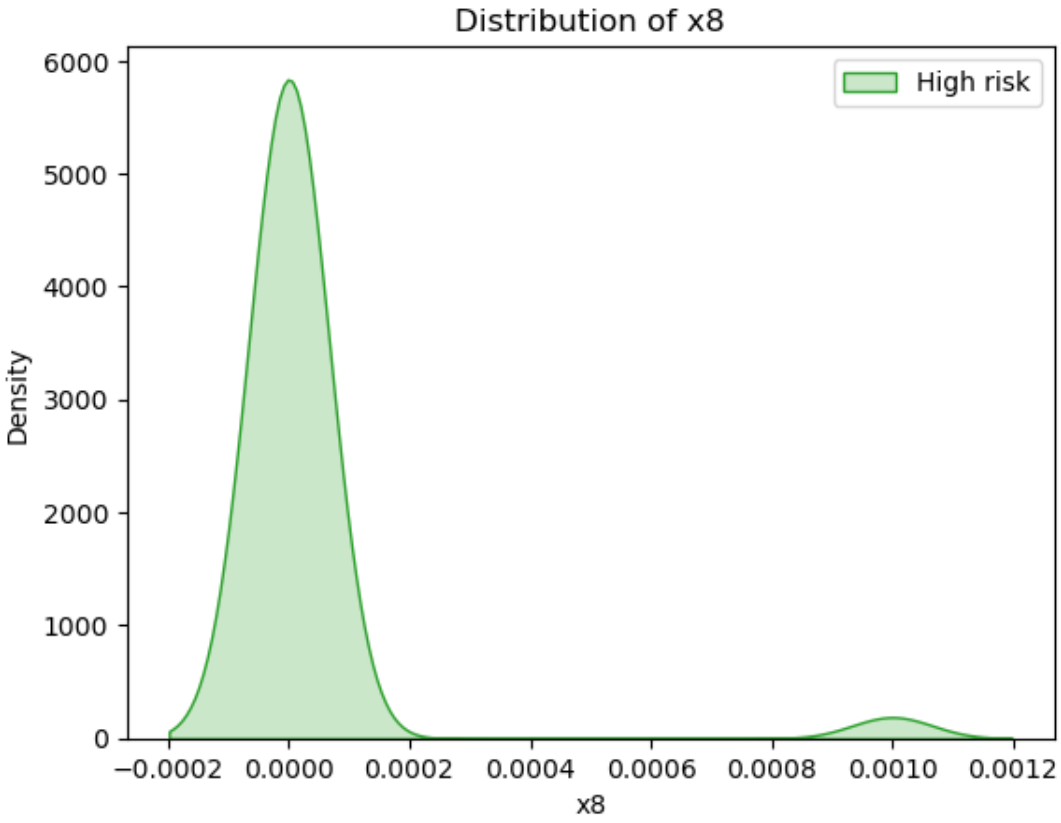

Distribution of x2
Low risk
Moderate risk
High risk
Density
0.00
0.01
0.02
0.03
0.04
0.05
100
110
120
130
140
150
160
170
x2

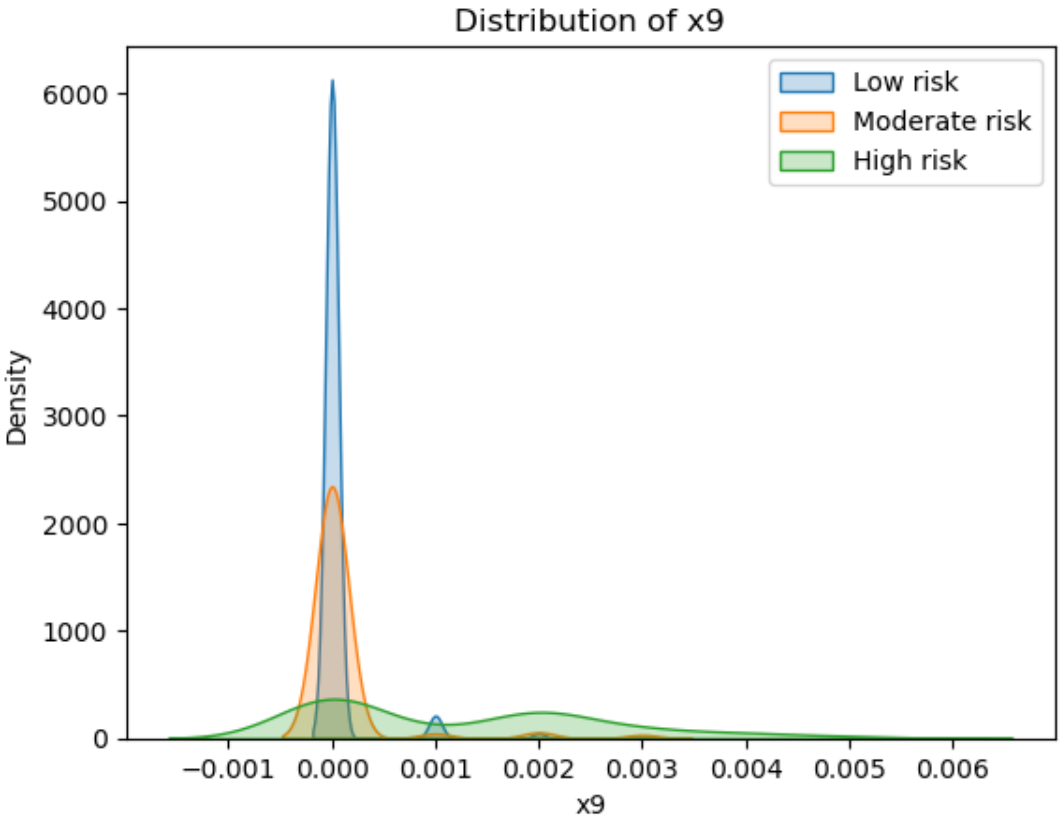

Distribution of x4
Low risk
Moderate risk
High risk
Density
1200
1000
800
600
400
200
0
0.000
0.005
0.010
0.015
0.020
x4

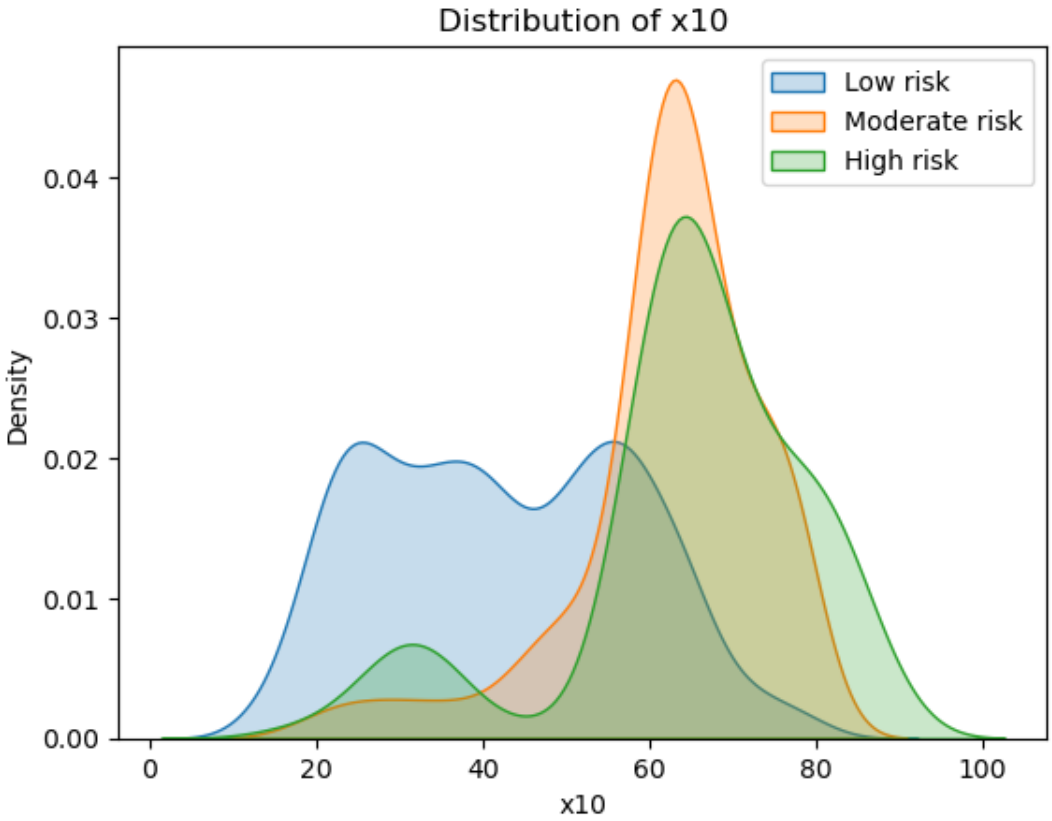

Distribution of x5
Low risk
Moderate risk
High risk
Density
x5
35
30
25
20
15
10
5
0
−0.1
0.0
0.1
0.2
0.3
0.4
0.5

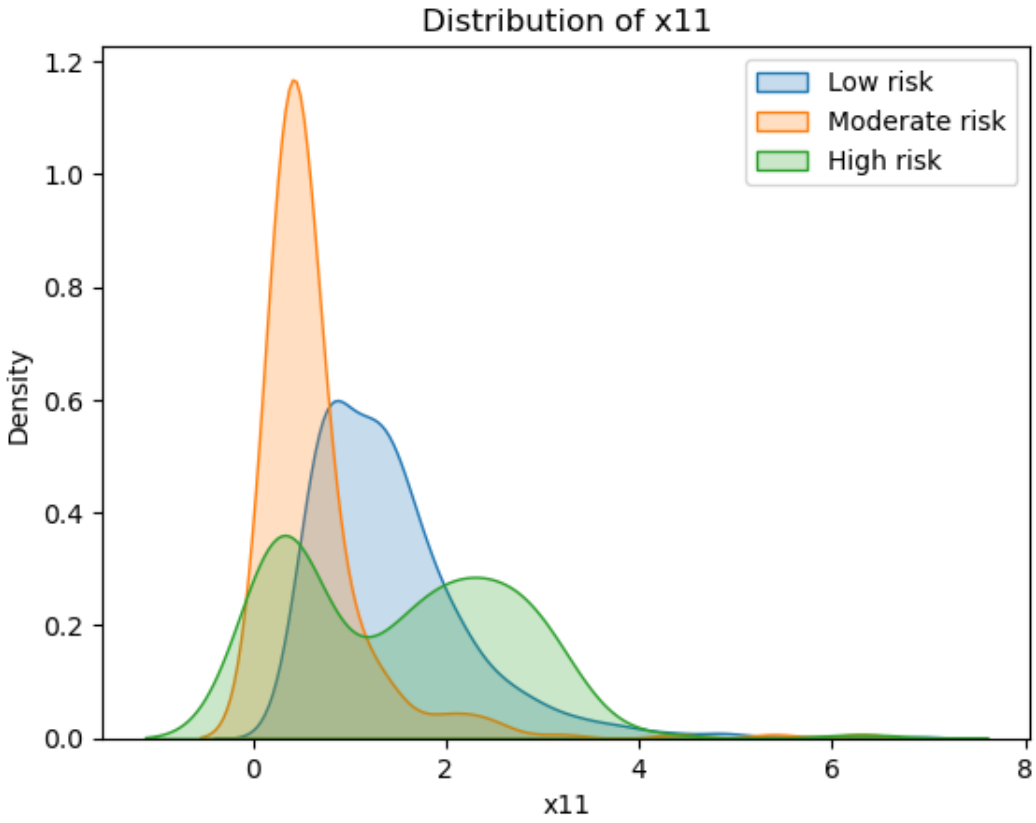

Distribution of x6
Low risk
Moderate risk
High risk
Density
200
175
150
125
100
75
50
25
0
−0.005
0.000
0.005
0.010
0.015
x6

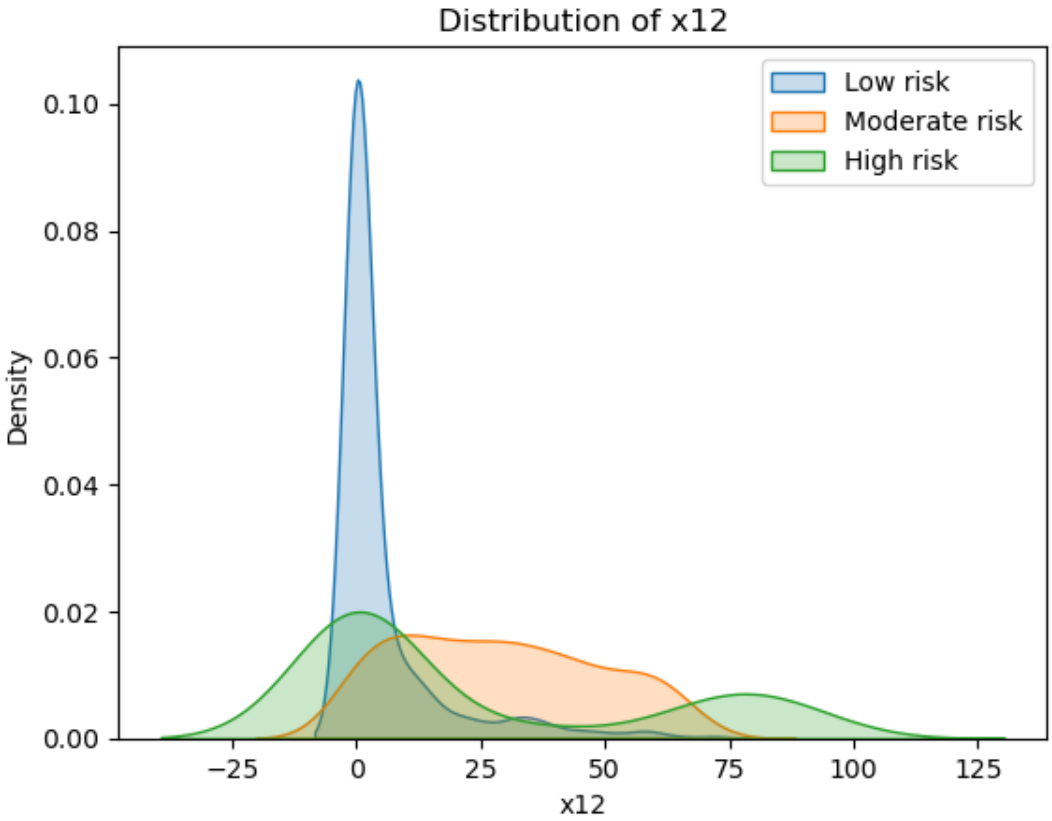

Distribution of x7
Low risk
Moderate risk
High risk
Density
500
400
300
200
100
0
−0.005
0.000
0.005
0.010
0.015
0.020
x7

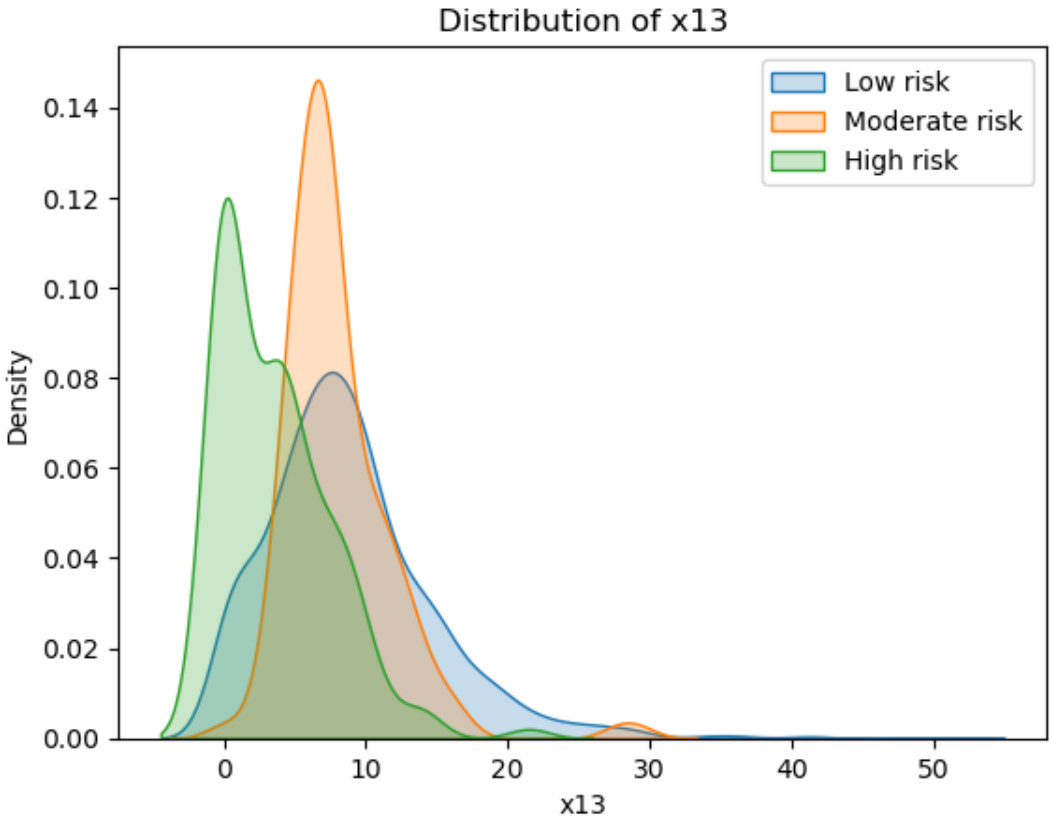

Distribution of x8
High risk
Density
6000
5000
4000
3000
2000
1000
0
−0.0002
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0.0012
x8

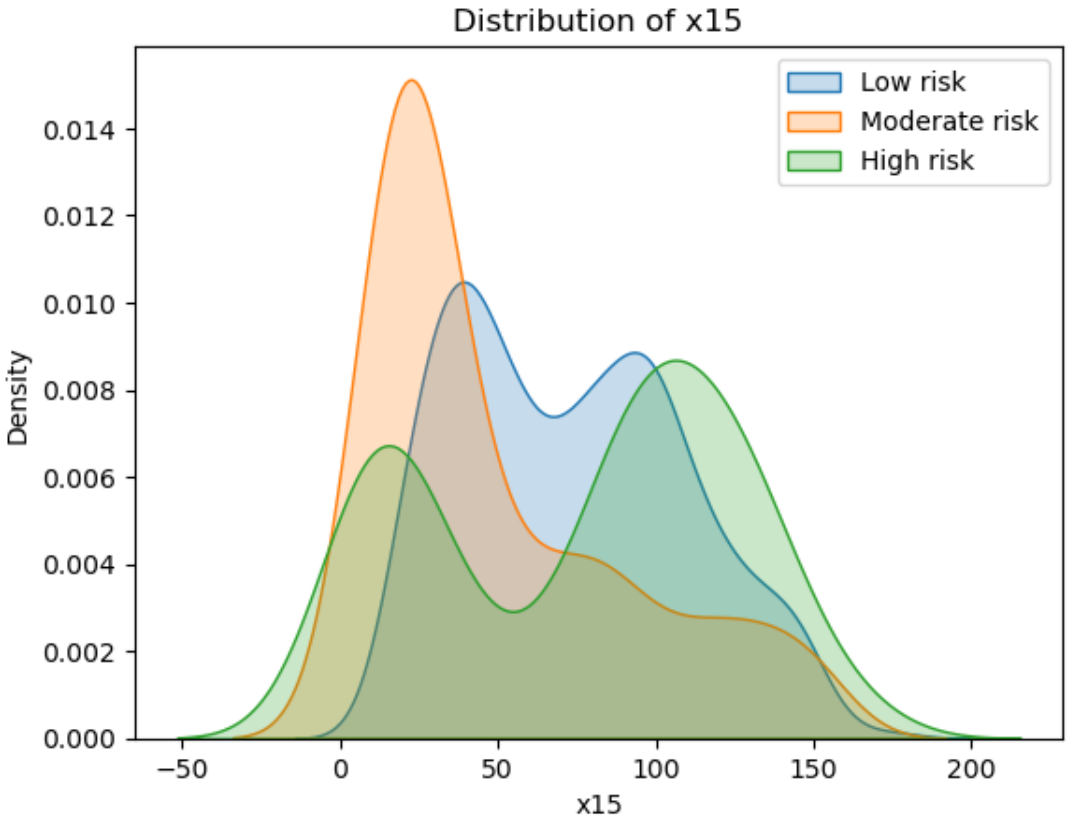

Distribution of x9
Low risk
Moderate risk
High risk
Density
6000
5000
4000
3000
2000
1000
0
−0.001
0.000
0.001
0.002
0.003
0.004
0.005
0.006
x9

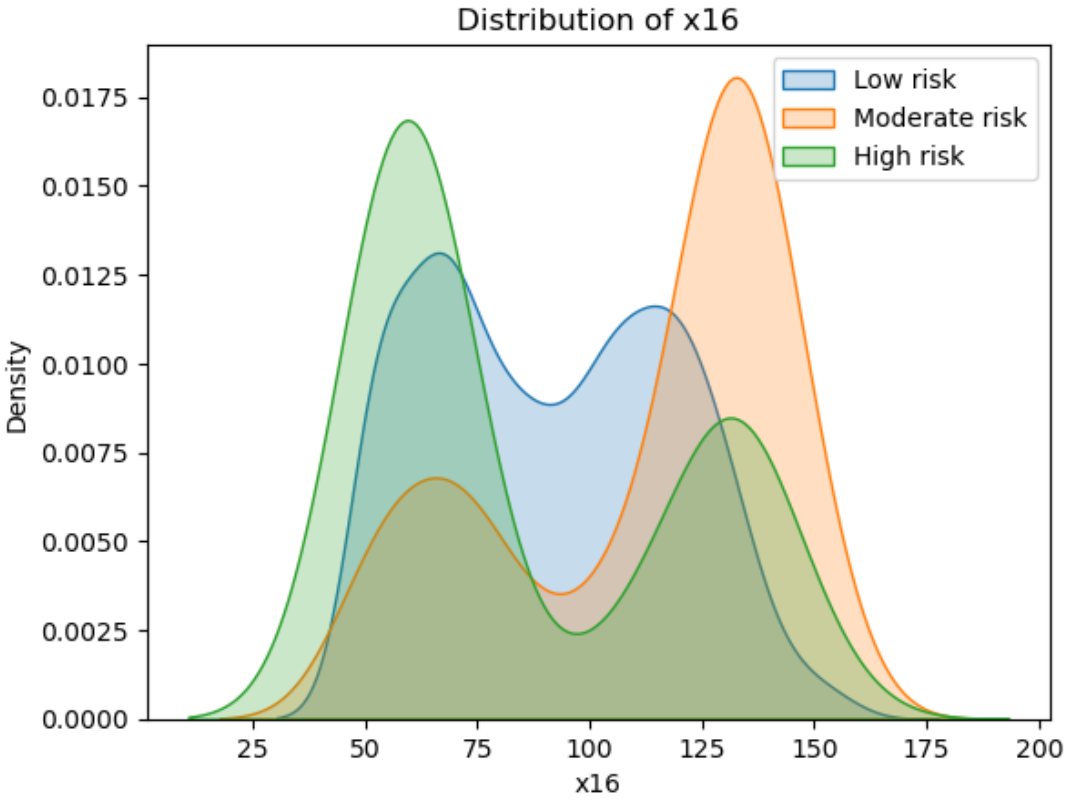

Distribution of x10
Low risk
Moderate risk
High risk
Density
0.00
0.01
0.02
0.03
0.04
0
20
40
60
80
100
x10

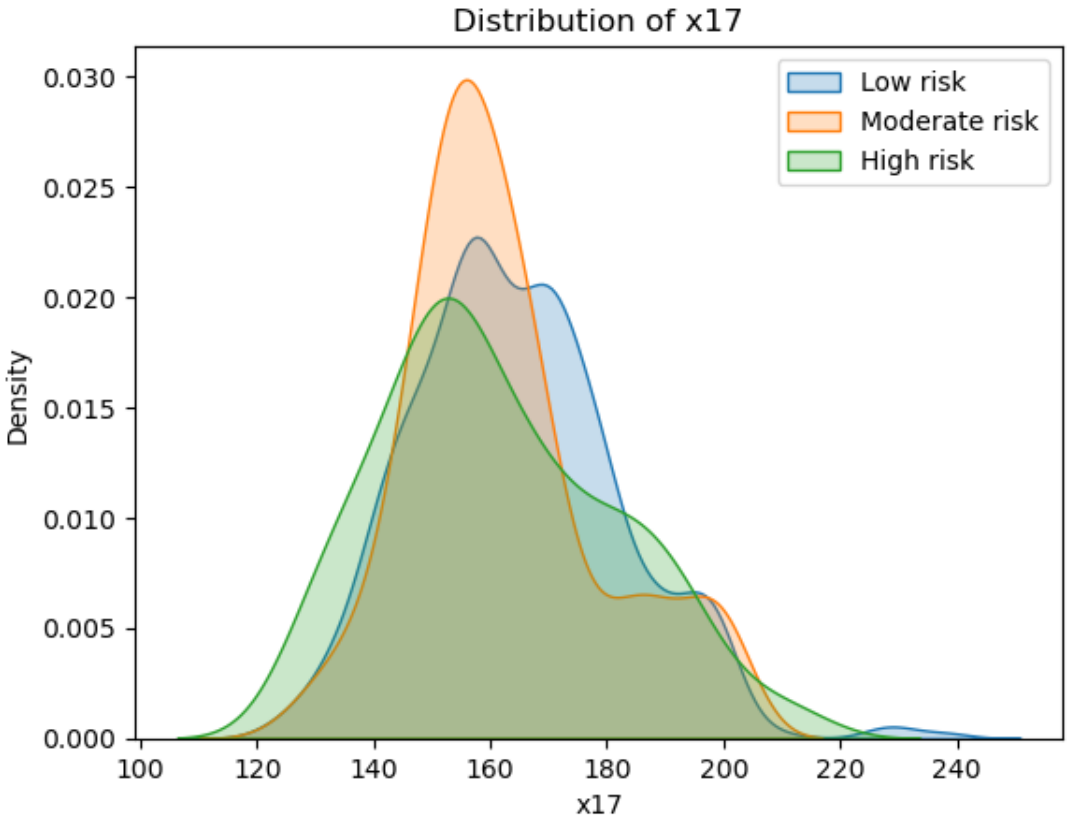

Distribution of x11
Low risk
Moderate risk
High risk
Density
x11
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
2
4
6
8

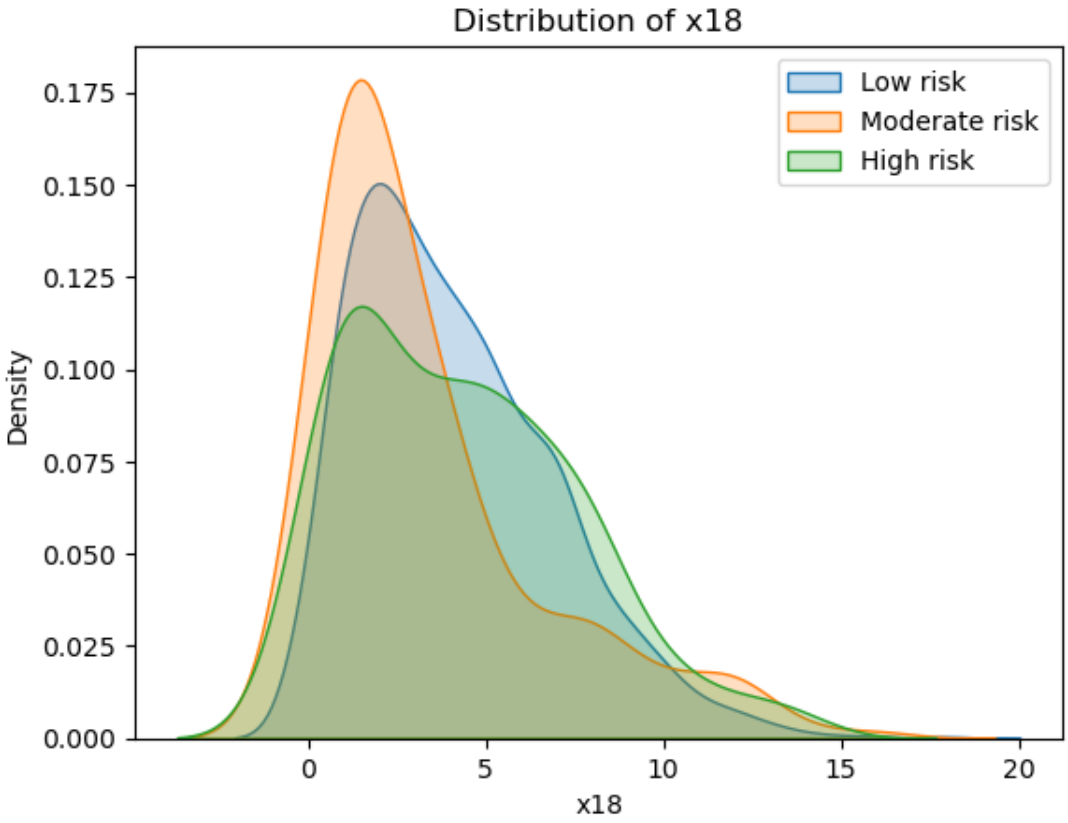

Distribution of x12
Low risk
Moderate risk
High risk
Density
0.10
0.08
0.06
0.04
0.02
0.00
−25
0
25
50
75
100
125
x12

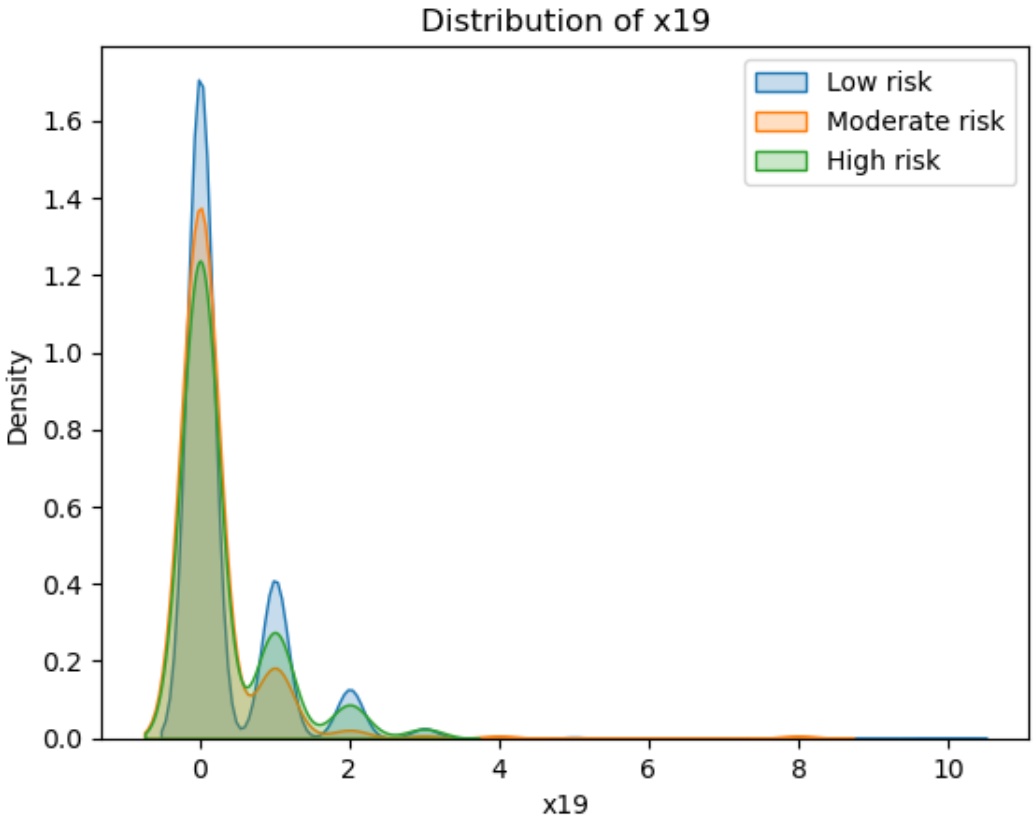

Distribution of x13
Low risk
Moderate risk
High risk
Density
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
10
20
30
40
50
x13

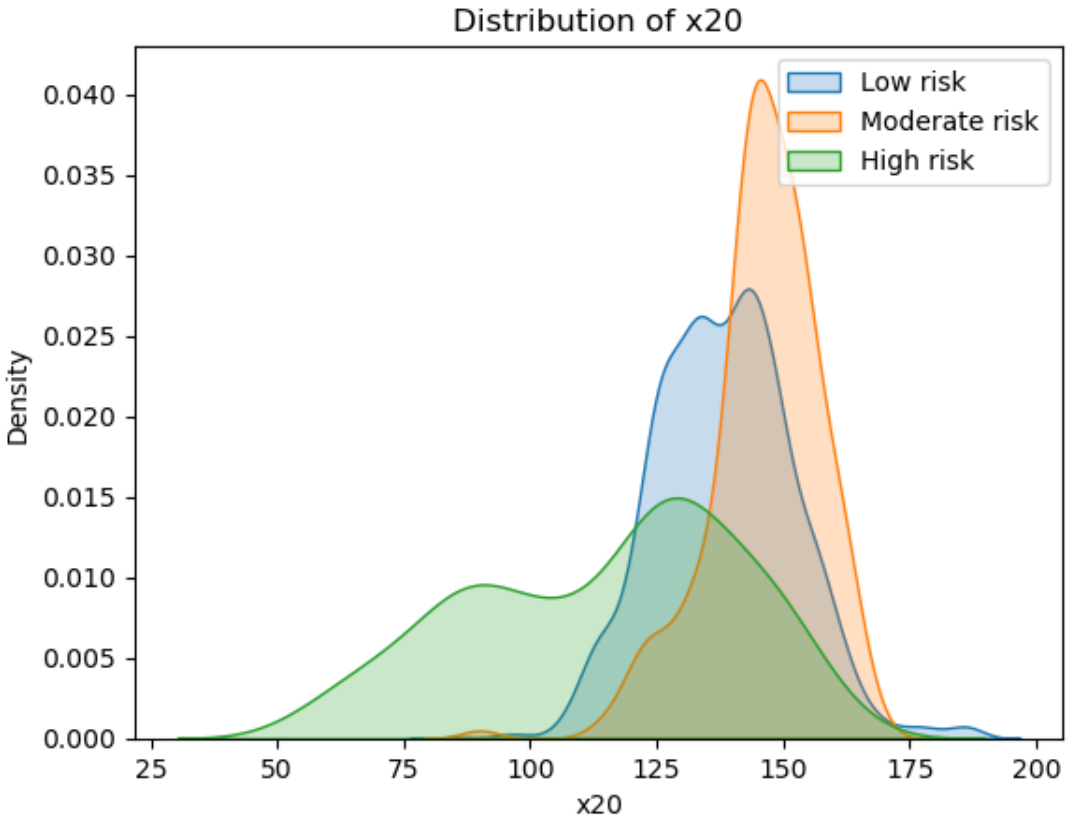

Distribution of x15
Low risk
Moderate risk
High risk
Density
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
−50
0
50
100
150
200
x15

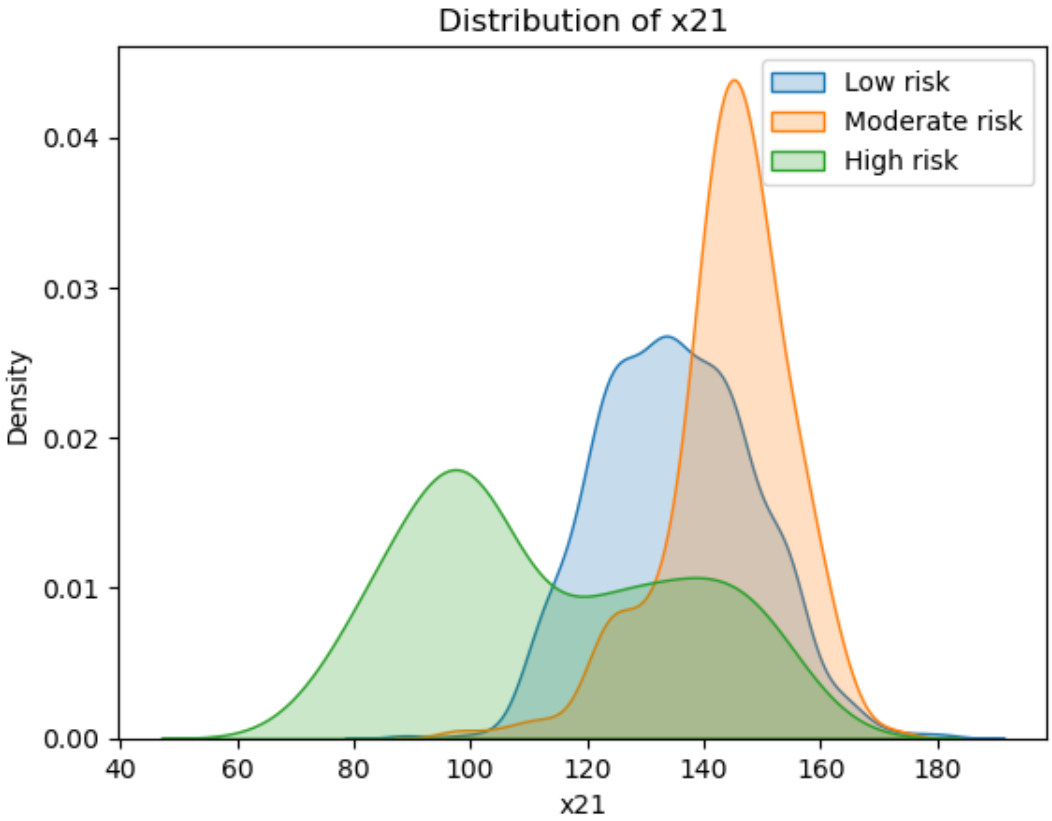

Distribution of x16
Low risk
Moderate risk
High risk
Density
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
25
50
75
100
125
150
175
200
x16

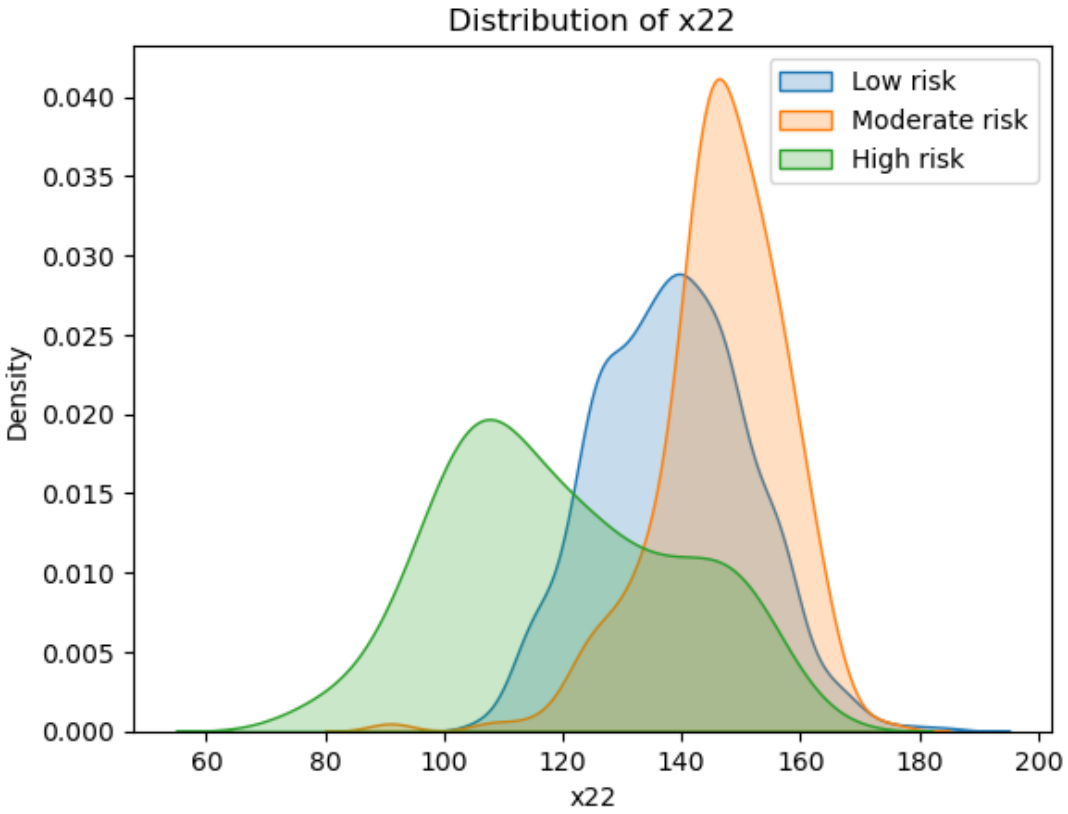

Distribution of x17
Low risk
Moderate risk
High risk
Density
0.030
0.025
0.020
0.015
0.010
0.005
0.000
100
120
140
160
180
200
220
240
x17

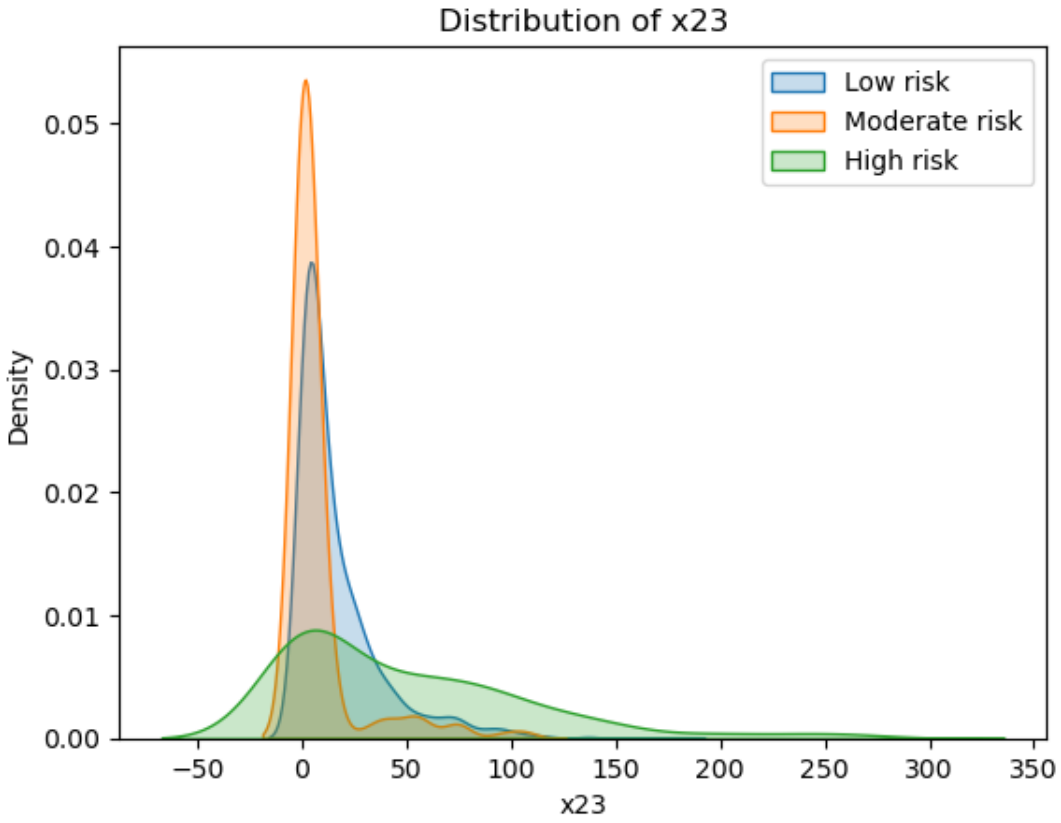

Distribution of x18
Low risk
Moderate risk
High risk
Density
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
5
10
15
20
x18

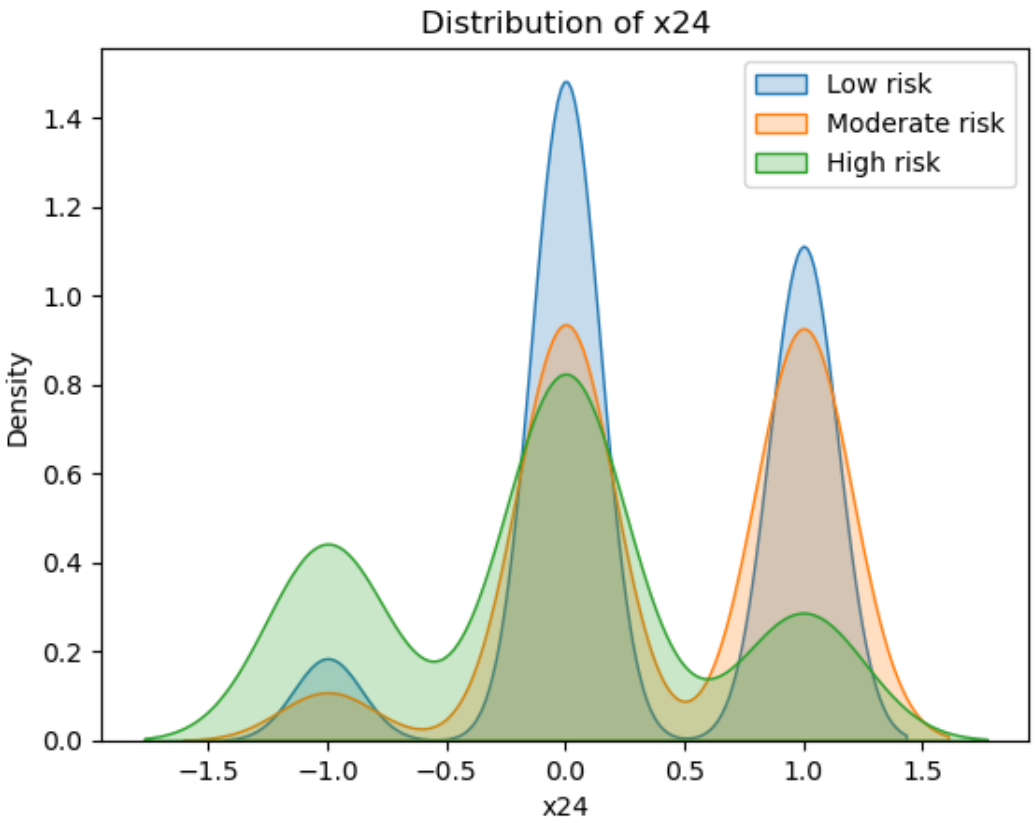

Distribution of x19
Low risk
Moderate risk
High risk
Density
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
x19


Distribution of x20
Low risk
Moderate risk
High risk
Density
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
25
50
75
100
125
150
175
200
x20


Distribution of x21
Low risk
Moderate risk
High risk
Density
0.00
0.01
0.02
0.03
0.04
40
60
80
100
120
140
160
180
x21


Distribution of x22
Low risk
Moderate risk
High risk
Density
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
60
80
100
120
140
160
180
200
x22

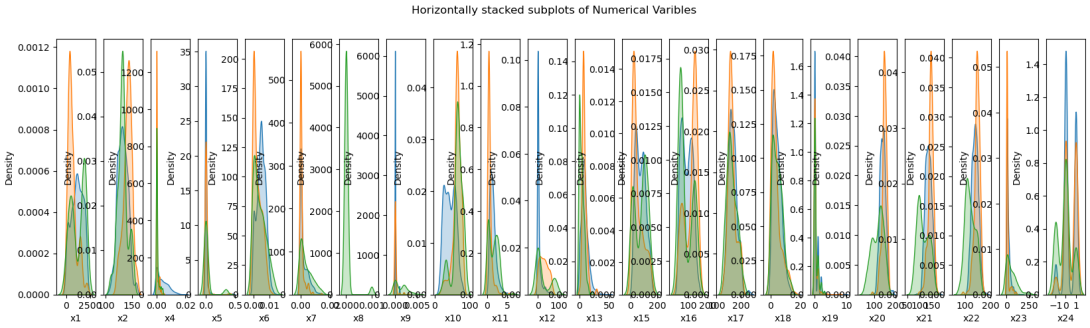

Distribution of x23
Low risk
Moderate risk
High risk
Density
0.05
0.04
0.03
0.02
0.01
0.00
−50
0
50
100
150
200
250
300
350
x23



[28]:
```
fig, (ax1) = plt.subplots(1,22, figsize=(20,5))
fig.suptitle('Horizontally stacked subplots of Numerical Varibles')


for i in range(len(CONTINUOUS_COLUMNS)):
    sns.kdeplot(data=HealthTrain_data_LR[CONTINUOUS_COLUMNS[i]],label="Low␣
 ↪risk", shade=True,ax=ax1[i])
    sns.kdeplot(data=HealthTrain_data_MR[CONTINUOUS_COLUMNS[i]],label="Moderate␣
 ↪risk", shade=True,ax=ax1[i])
    sns.kdeplot(data=HealthTrain_data_HR[CONTINUOUS_COLUMNS[i]],label="High␣
 ↪risk", shade=True,ax=ax1[i])

plt.show()
```

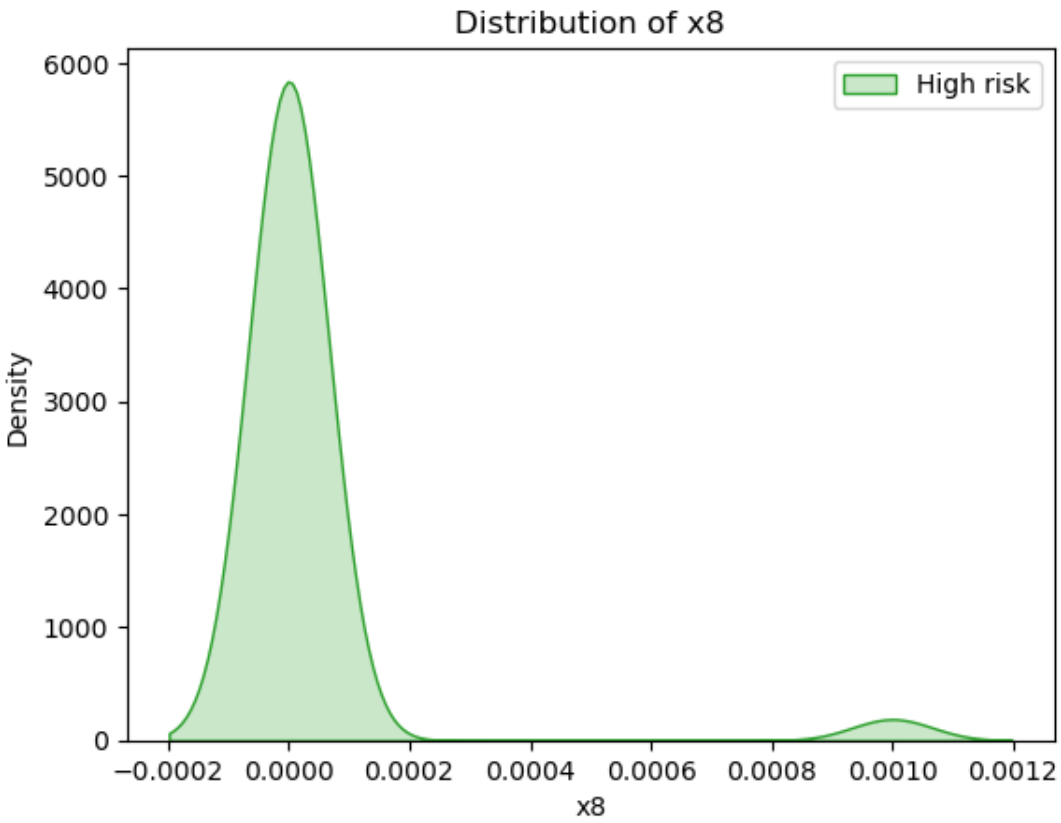
```
C:\Users\JT\anaconda3\lib\site-packages\seaborn\distributions.py:316:
UserWarning: Dataset has 0 variance; skipping density estimate. Pass
`warn_singular=False` to disable this warning.
  warnings.warn(msg, UserWarning)
C:\Users\JT\anaconda3\lib\site-packages\seaborn\distributions.py:316:
UserWarning: Dataset has 0 variance; skipping density estimate. Pass
```

```
`warn_singular=False` to disable this warning.
  warnings.warn(msg, UserWarning)
```

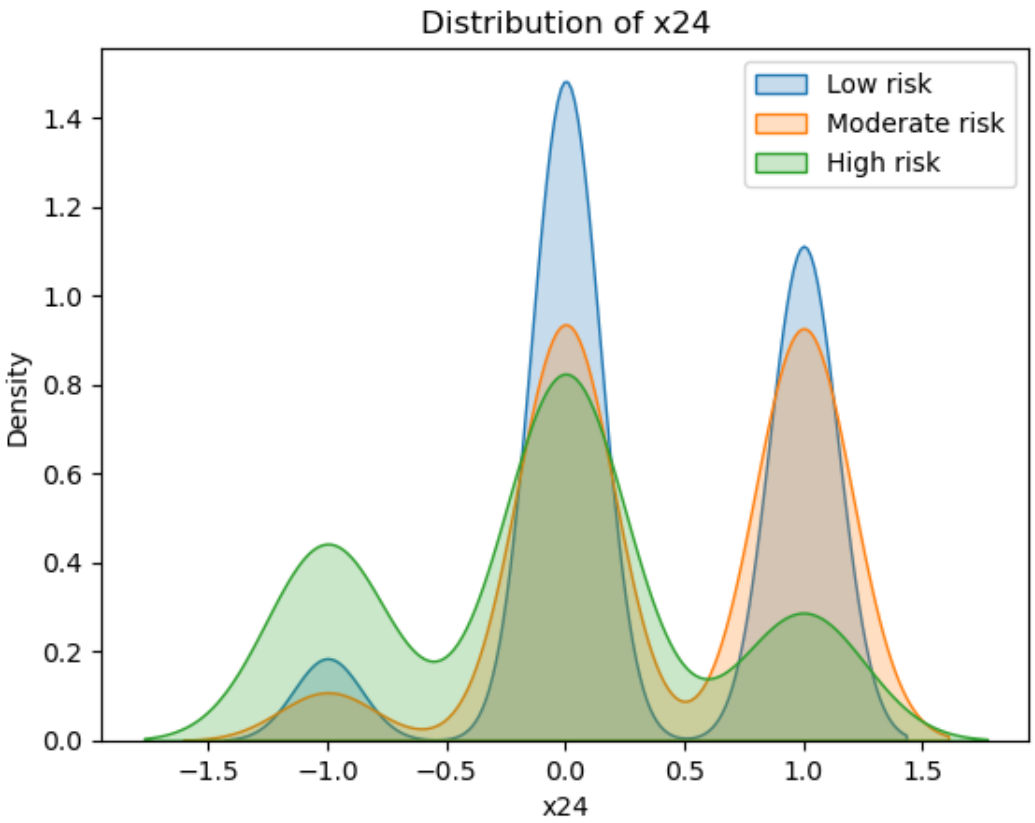


[29]:
```
sns.kdeplot(data=HealthTrain_data_LR['x8'],label="Low risk", shade=True)
sns.kdeplot(data=HealthTrain_data_MR['x8'],label="Moderate risk", shade=True)
sns.kdeplot(data=HealthTrain_data_HR['x8'],label="High risk", shade=True)

plt.title("Distribution of x8")
plt.legend()

plt.savefig('x8_dis'+'.jpg')
```

```
C:\Users\JT\anaconda3\lib\site-packages\seaborn\distributions.py:316:
UserWarning: Dataset has 0 variance; skipping density estimate. Pass
`warn_singular=False` to disable this warning.
  warnings.warn(msg, UserWarning)
C:\Users\JT\anaconda3\lib\site-packages\seaborn\distributions.py:316:
UserWarning: Dataset has 0 variance; skipping density estimate. Pass
`warn_singular=False` to disable this warning.
  warnings.warn(msg, UserWarning)
```



[30]:
```
sns.kdeplot(data=HealthTrain_data_LR['x24'],label="Low risk", shade=True)
sns.kdeplot(data=HealthTrain_data_MR['x24'],label="Moderate risk", shade=True)
sns.kdeplot(data=HealthTrain_data_HR['x24'],label="High risk", shade=True)

plt.title("Distribution of x24")
plt.legend()

plt.savefig('x24_dis'+'.jpg')
```



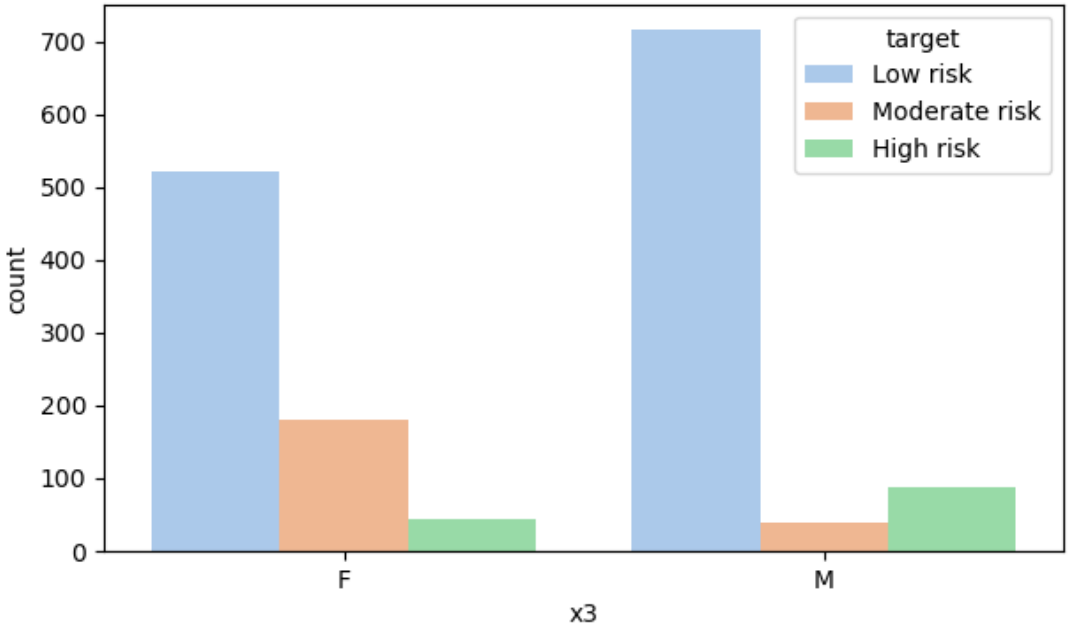
[31]:
```
for column in CATEGORICAL_COLUMNS:
    CategoricalD.loc['missing',column] = HealthTrain_data[column].isnull().sum()
    CategoricalD.loc['values',column] = HealthTrain_data[column].unique()

print(CategoricalD)
```

```
                                                      id      x3  \
count                                               1584    1584
unique                                              1584       2
top                                                PA1001       M
freq                                                   1     842
missing                                                0       0
values   [PA1001, PA1002, PA1003, PA1004, PA1005, PA100…  [F, M]

                                          x14
count                                    1584
unique                                      8
top                                        0+
freq                                      410
missing                                     0
```

```
values    [O+, A+, B+, O-, A-, B-, AB-, AB+]
```

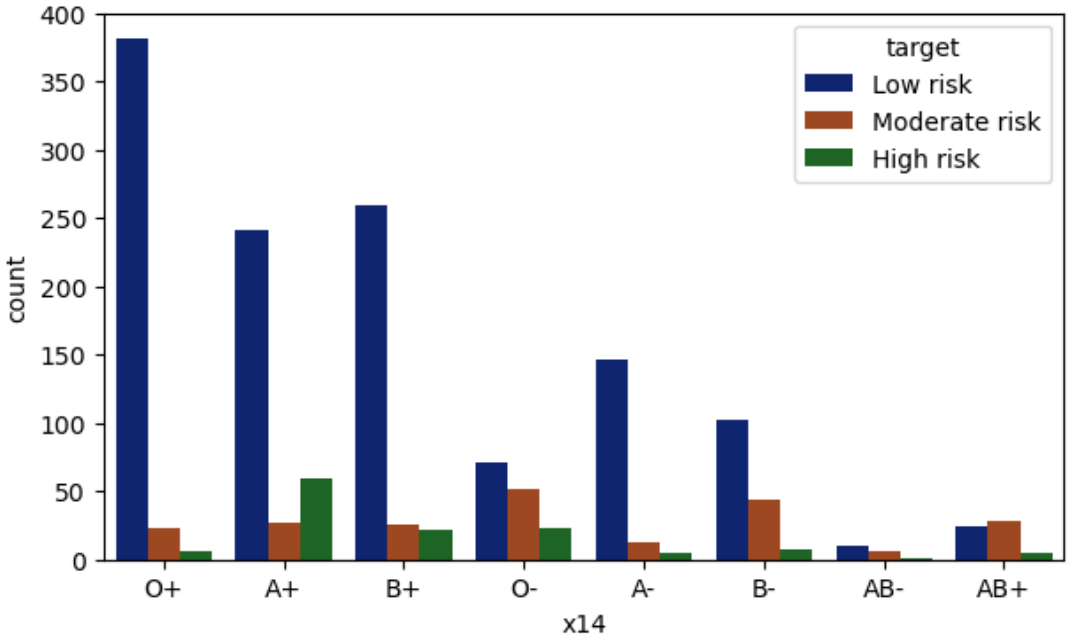
[32]:
```
plt.figure(figsize=(7,4))
sns.countplot(x="x3", hue='target', palette="pastel", data=HealthTrain_data)
```

[32]:
```
<AxesSubplot:xlabel='x3', ylabel='count'>
```

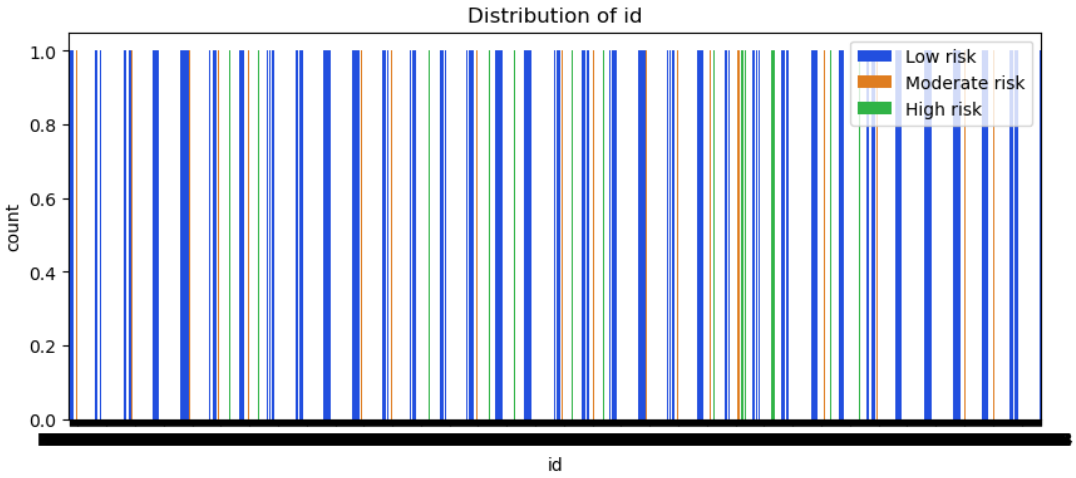


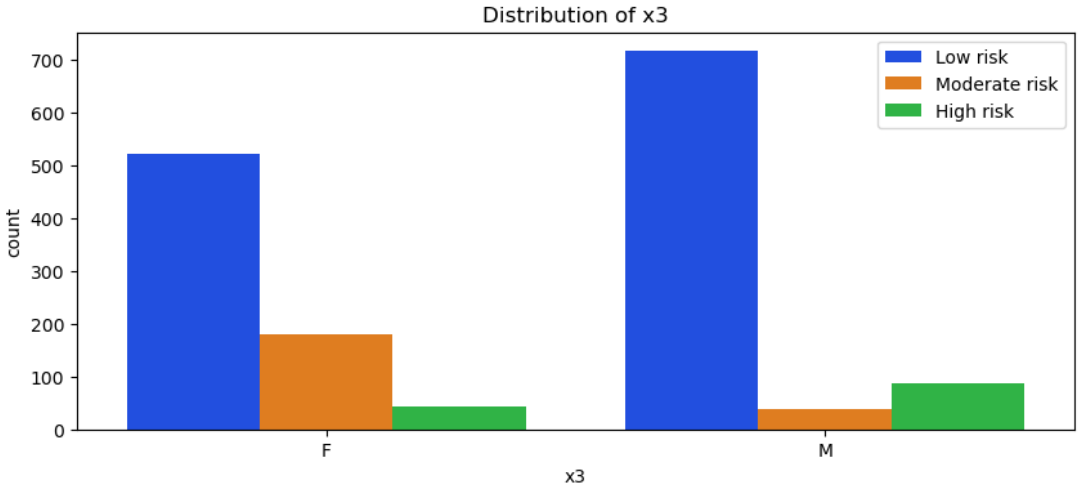
[33]:
```
plt.figure(figsize=(7,4))
sns.countplot(x="x14", hue='target', palette="dark", data=HealthTrain_data)
```

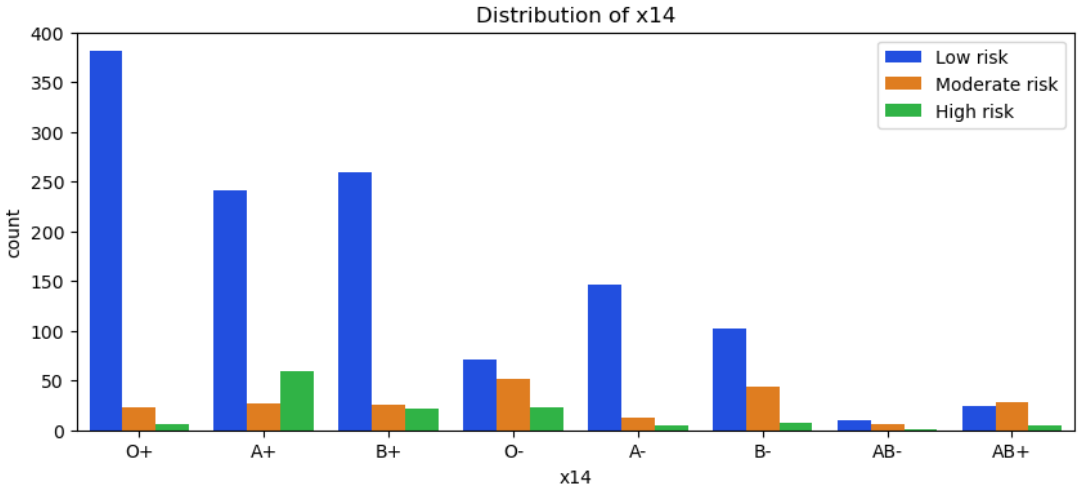
[33]:
```
<AxesSubplot:xlabel='x14', ylabel='count'>
```



[34]:
```
for column in CATEGORICAL_COLUMNS:
    plt.figure(figsize=(10,4))
    #plt.xticks(rotation=90)
    sns.countplot(x=column, hue='target', palette="bright",␣
 ↪data=HealthTrain_data)
    plt.legend()
    plt.title("Distribution of " +column)
    plt.savefig(column+'.jpg')
```

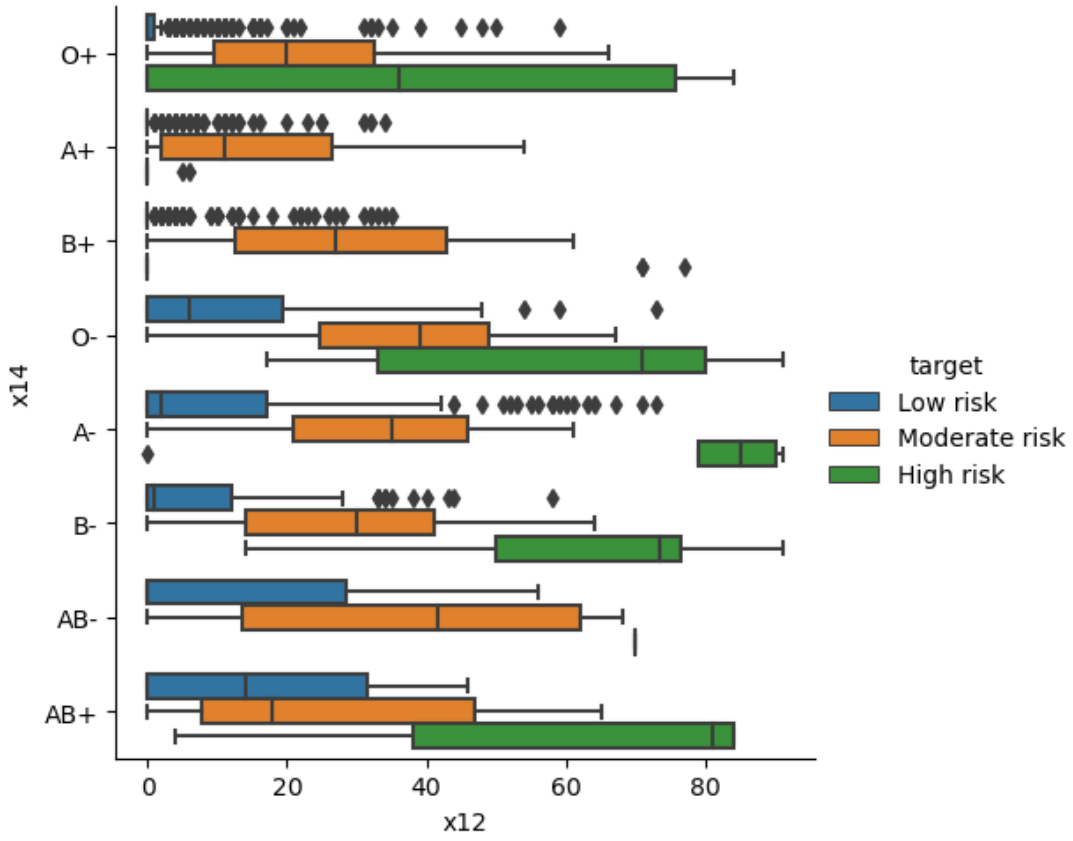






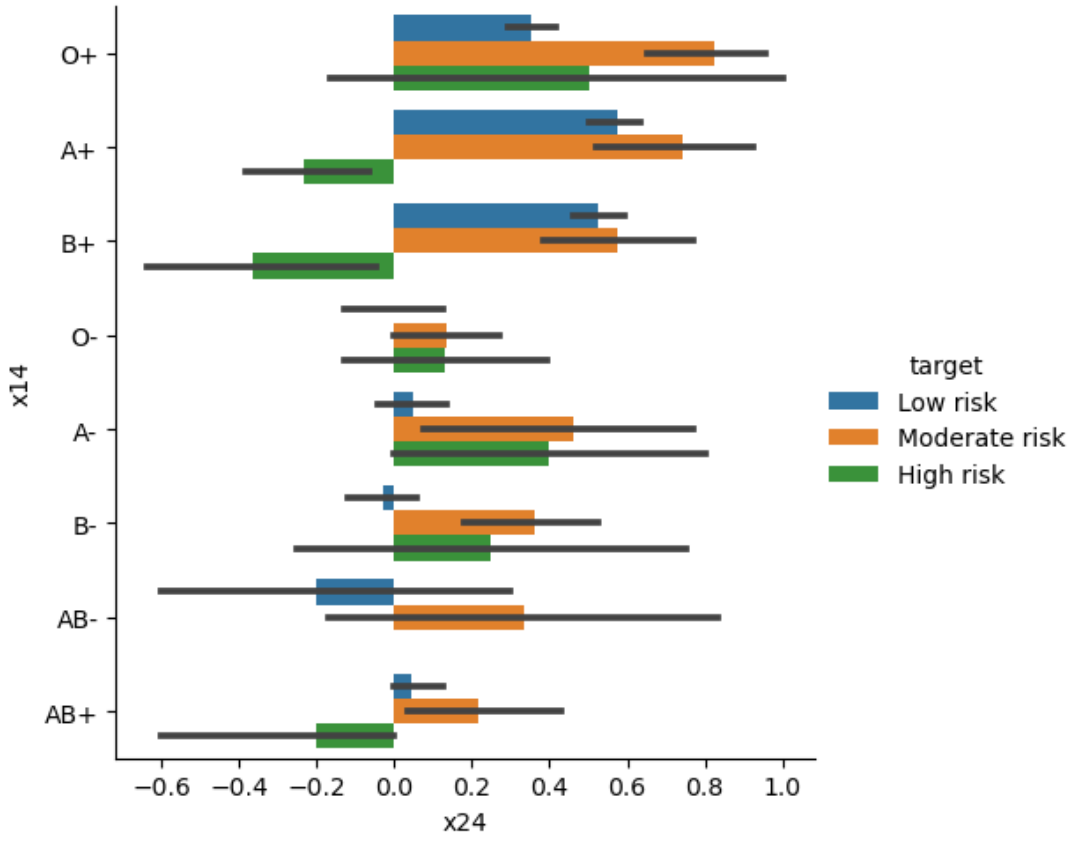
[35]:
```
sns.catplot(x="x12",y="x14", hue="target", kind="box", data=HealthTrain_data)
```

[35]: <seaborn.axisgrid.FacetGrid at 0x214b7ffa550>



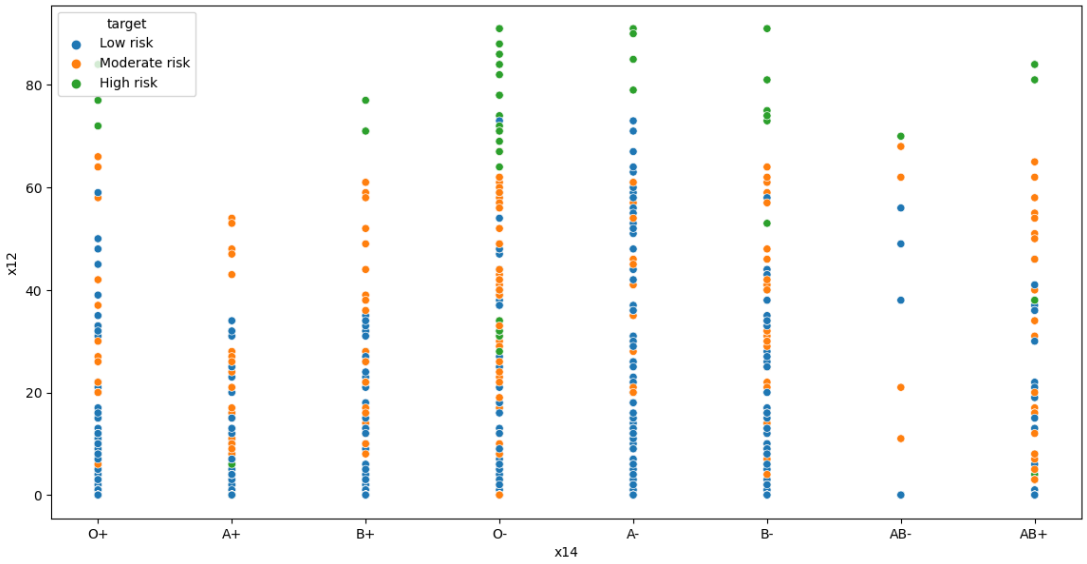
[36]:
```
sns.catplot(x="x24",y="x14", hue="target", kind="bar", data=HealthTrain_data)
```

[36]:
```
<seaborn.axisgrid.FacetGrid at 0x214b38a5700>
```



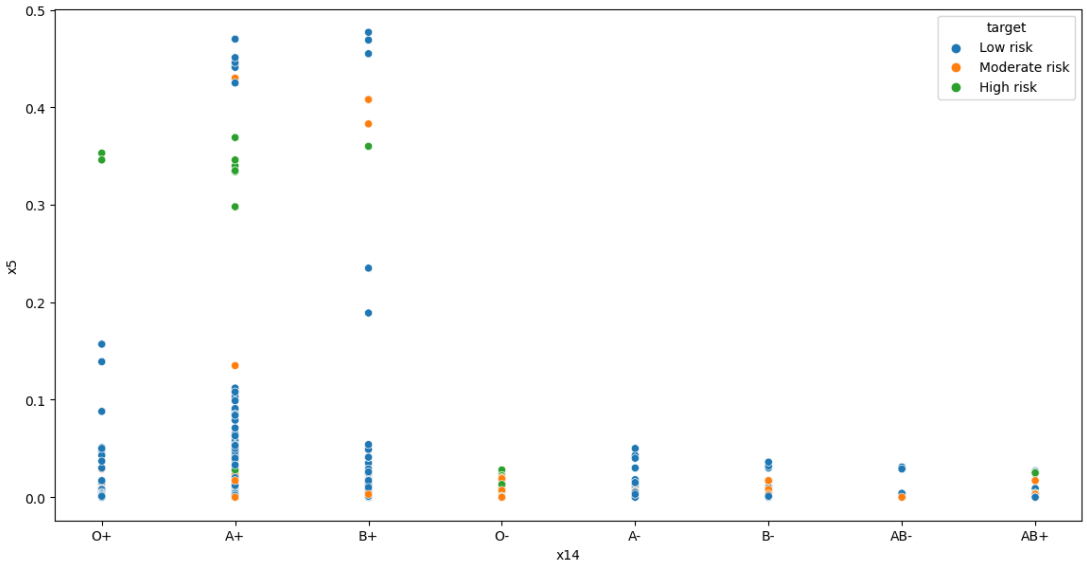
[37]:
```
plt.figure(figsize=(14,7))

#sns.lmplot(x="x12", y="x24", hue="target", data=HealthTrain_data)
sns.scatterplot(x=HealthTrain_data['x14'],␣
↪y=HealthTrain_data['x12'],hue=HealthTrain_data['target'])
```

[37]:
```
<AxesSubplot:xlabel='x14', ylabel='x12'>
```



[38]:
```
plt.figure(figsize=(14,7))

sns.scatterplot(x=HealthTrain_data['x14'],␣
↪y=HealthTrain_data['x5'],hue=HealthTrain_data['target'])
```

[38]: <AxesSubplot:xlabel='x14', ylabel='x5'>



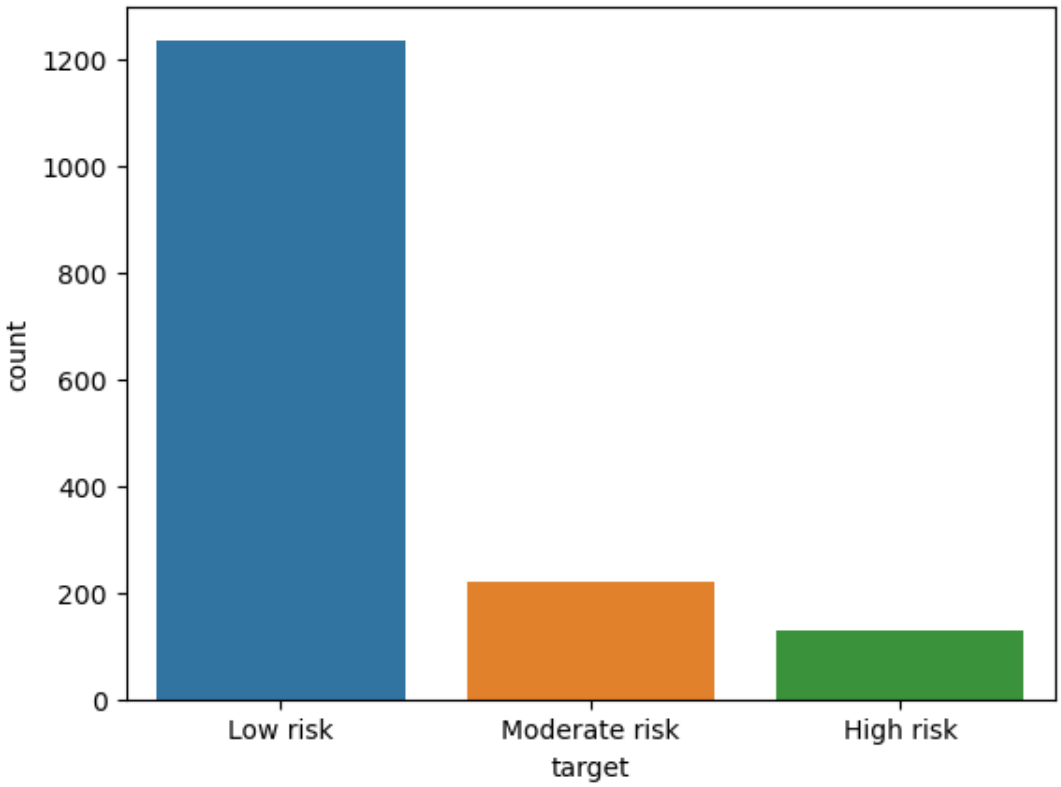
[39]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
print("Vis setup Complete")
```

```
Vis setup Complete
```

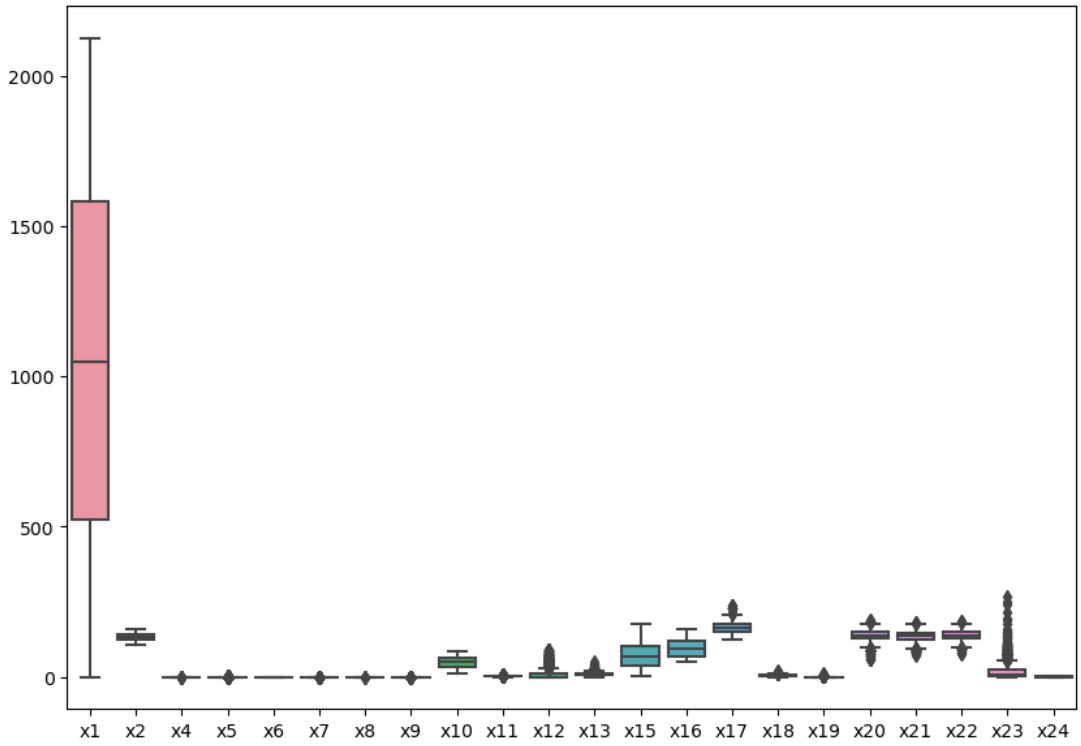
[40]:
```
print("Distribution of the risk levels among the data set")
sns.countplot(x= 'target', data=HealthTrain_data)
```

```
Distribution of the risk levels among the data set
```

[40]:
```
<AxesSubplot:xlabel='target', ylabel='count'>
```



[41]:
```
plt.figure(figsize=(10,7))
sns.boxplot(data=HealthTrain_data)
plt.show()
```

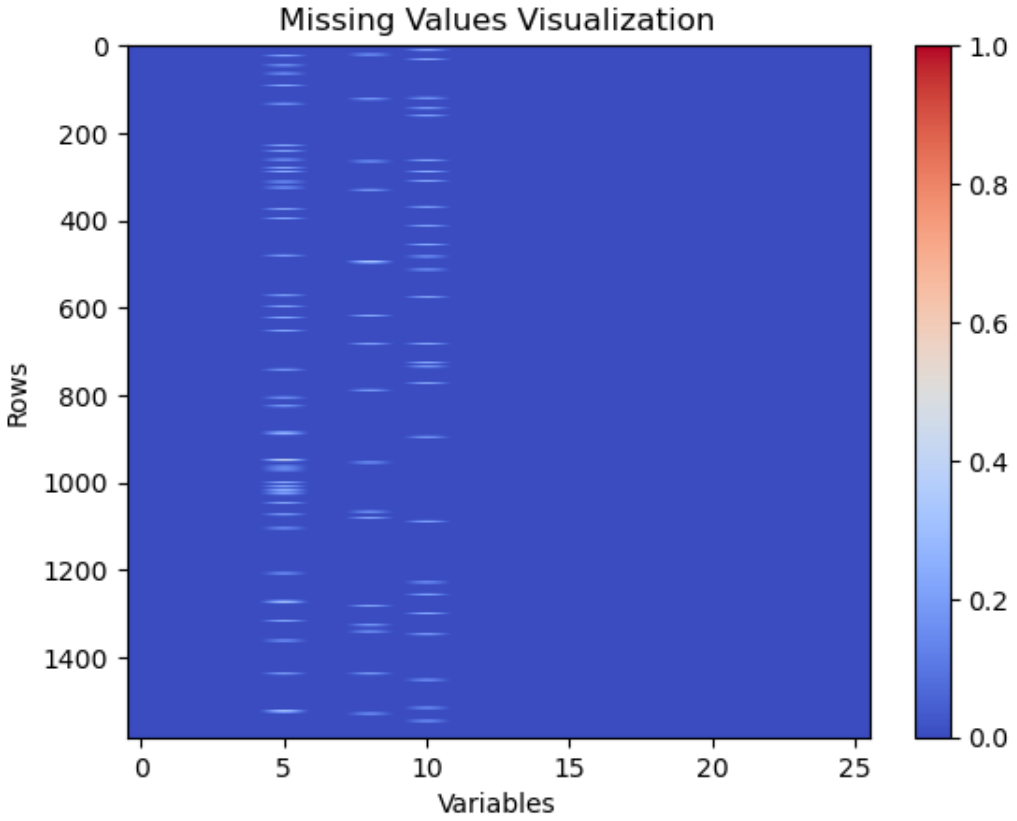


[42]:
```
missing_v = HealthTrain_data.isnull()
missing_vcount = missing_v.sum()
pd.set_option('display.max_rows',None)
print("The sum of the counts of the missing values in each variable.")
print(missing_vcount)
```

```
The sum of the counts of the missing values in each variable.
id          0
x1          0
x2          0
x3          0
x4          0
x5         44
x6          0
x7          0
x8         17
x9          0
x10        27
x11         0
x12         0
x13         0
x14         0
x15         0
```

```
x16         0
x17         0
x18         0
x19         0
x20         0
x21         0
x22         0
x23         0
x24         0
target      0
dtype: int64
```

[43]:
```
missing_v = HealthTrain_data.isnull()
plt.imshow(missing_v, cmap='coolwarm', aspect='auto')
plt.colorbar()
plt.title('Missing Values Visualization')
plt.xlabel('Variables')
plt.ylabel('Rows')
plt.show()
```



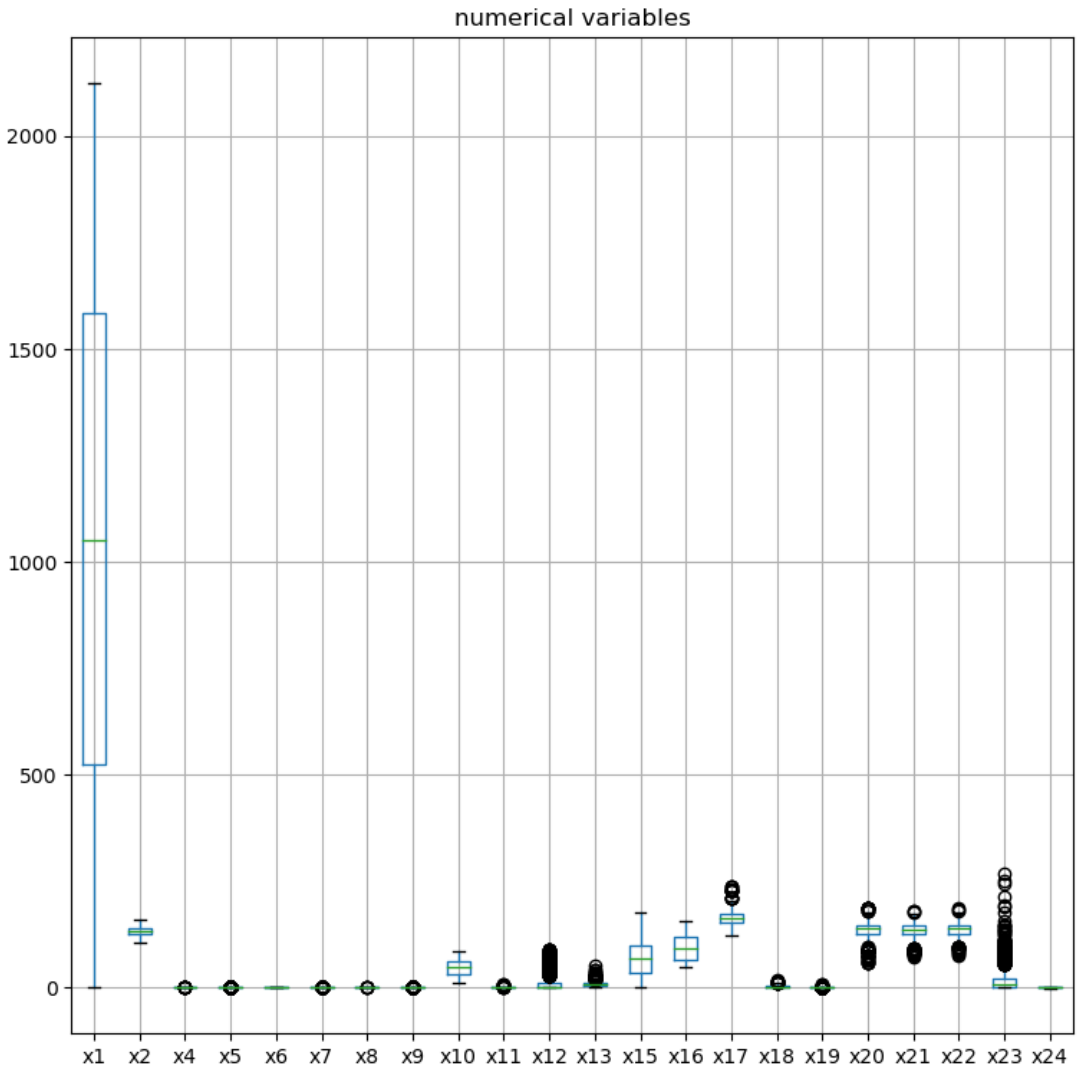
[44]:
```
pd.set_option('display.max_rows',10)
htrain= HealthTrain_data[CONTINUOUS_COLUMNS]
htrain_nomissingv = htrain.dropna()
print(htrain_nomissingv)
```

```
        x1     x2     x4     x5     x6     x7   x8     x9   x10  x11   x12  \
0     1406  145.0  0.005  0.000  0.002  0.000  0.0  0.000  46.0  0.8   0.0
1      258  127.0  0.012  0.000  0.008  0.004  0.0  0.000  13.0  3.8   0.0
2      479  145.0  0.000  0.000  0.000  0.002  0.0  0.000  57.0  0.5   0.0
3      906  146.0  0.004  0.000  0.005  0.003  0.0  0.000  29.0  1.2   1.0
4     1921  140.0  0.002  0.003  0.006  0.006  0.0  0.000  62.0  1.6   0.0
...    ...    ...    ...    ...    ...    ...  ...    ...   ...  ...   ...
1579  2077  130.0  0.005  0.001  0.001  0.000  0.0  0.000  72.0  0.9   8.0
1580   664  138.0  0.000  0.003  0.003  0.000  0.0  0.002  60.0  1.0   8.0
1581  1431  144.0  0.000  0.000  0.006  0.000  0.0  0.000  45.0  0.7   0.0
1582   630  134.0  0.017  0.002  0.004  0.000  0.0  0.000  48.0  2.2   0.0
1583   436  151.0  0.000  0.000  0.006  0.006  0.0  0.000  64.0  1.1  26.0

      x13    x15    x16    x17   x18  x19    x20    x21    x22   x23  x24
0     8.6   67.0  104.0  171.0   4.0  0.0  155.0  153.0  154.0   4.0  1.0
1     1.3  138.0   53.0  191.0  12.0  1.0  133.0  126.0  131.0  41.0  0.0
2     7.3   46.0  111.0  157.0   1.0  1.0  150.0  146.0  149.0   6.0  1.0
3     7.0   62.0  107.0  169.0   2.0  2.0  150.0  147.0  149.0   7.0  0.0
4     9.1  153.0   75.0  228.0   9.0  0.0  142.0  118.0  142.0  20.0  0.0
...   ...    ...    ...    ...   ...  ...    ...    ...    ...   ...  ...
1579  4.4   31.0  127.0  158.0   2.0  0.0  139.0  139.0  140.0   3.0  0.0
1580  7.5  118.0   69.0  187.0  10.0  1.0  142.0  130.0  140.0  61.0  0.0
1581  9.8   30.0  139.0  169.0   2.0  0.0  157.0  155.0  157.0   2.0  0.0
1582  0.0  120.0   50.0  170.0   5.0  0.0  160.0  150.0  155.0  28.0  1.0
1583  3.0  150.0   50.0  200.0  11.0  2.0  156.0  150.0  156.0  38.0  1.0

[1498 rows x 22 columns]
```

[45]:
```
htrain= HealthTrain_data[CONTINUOUS_COLUMNS]
htrain.shape
```

[45]: (1584, 22)

[46]:
```
htrain.boxplot(figsize=(9,9))
plt.title('numerical variables')
plt.show()
```

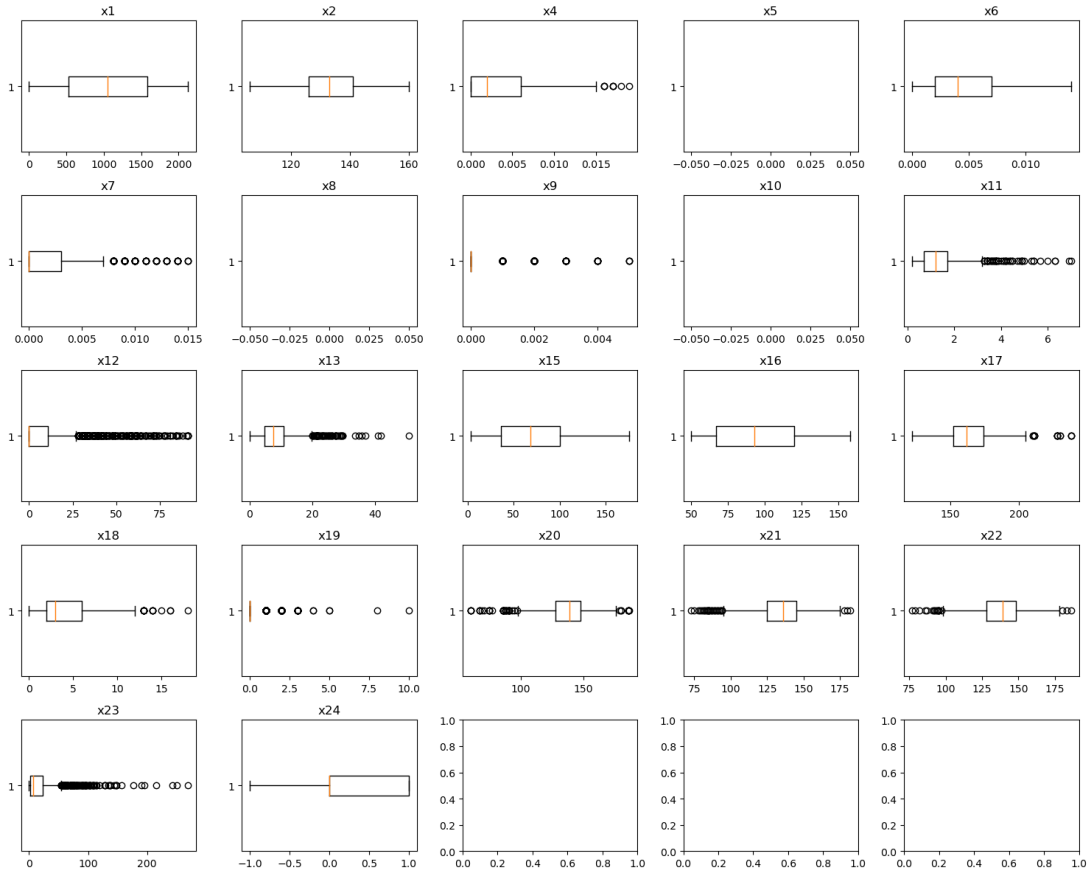


[47]:
```
print("Detecting Outliers using the Zscore method")

zscore = (htrain - htrain.mean())/ htrain.std()
threshold = 3
outliers_Mask = np.abs(zscore) > threshold
sum_outliers = outliers_Mask.sum()
pd.set_option('display.max_rows',None)
print("Sum number of outliers for each variable are:")
print(sum_outliers)
```

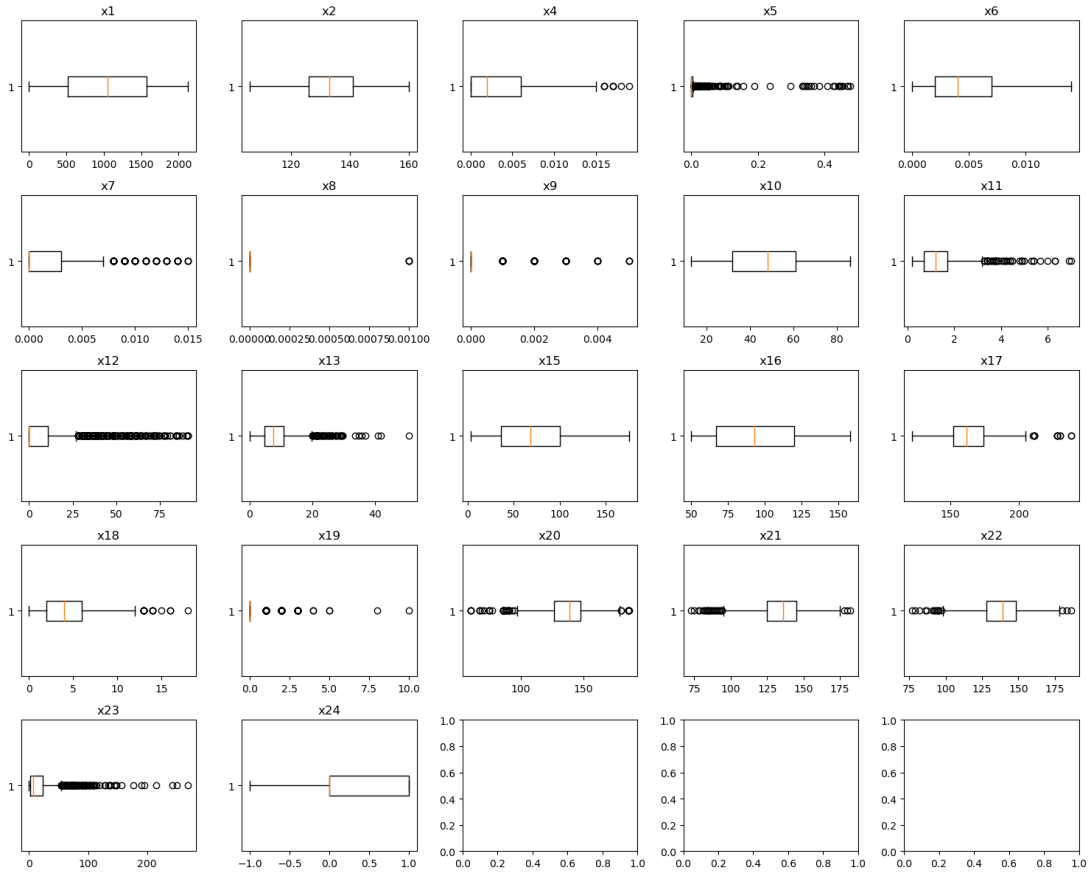
```
Detecting Outliers using the Zscore method
Sum number of outliers for each variable are:
x1      0
```

```
x2      0
x4     16
x5     24
x6      2
x7     31
x8      4
x9     79
x10     0
x11    25
x12    42
x13    24
x15     0
x16     0
x17    10
x18    16
x19    23
x20    27
x21    20
x22    13
x23    30
x24     0
dtype: int64
```

[48]:
```
fig, axs = plt.subplots(nrows=5, ncols=5, figsize=(15, 12))
for i, col in enumerate(htrain.columns):
    ax = axs[i // 5, i % 5]  # Get the corresponding axis
    ax.boxplot(htrain[col], vert=False)
    ax.set_title(col)
plt.tight_layout()
plt.show()
```



[49]:
```
fig, axs = plt.subplots(nrows=5, ncols=5, figsize=(15, 12))
for i, col in enumerate(htrain_nomissingv.columns):
    ax = axs[i // 5, i % 5]  # Get the corresponding axis
    ax.boxplot(htrain_nomissingv[col], vert=False)
    ax.set_title(col)
plt.tight_layout()
plt.show()
```



[50]:
```
print("Filtering to ensure only those rows that are not outliers are retained␣
 ↪in a new data frame for further coding")

zscores = np.abs((htrain - htrain.mean())/ htrain.std())
threshold = 3

masked_data = zscores[~zscores.isnull()].lt(threshold).all(axis=1)

filtered_data = htrain[masked_data]
print(filtered_data.shape)
```

```
Filtering to ensure only those rows that are not outliers are retained in a new
data frame for further coding
(1251, 22)
```

[51]:
```
print("Filtering to ensure only those rows that are not outliers & missing␣
 ↪values are retained in a new data frame for further coding")
```

```
zscores = np.abs((htrain_nomissingv - htrain_nomissingv.mean())/␣
 ↪htrain_nomissingv.std())
threshold = 3

masked_datanomissingv = zscores[~zscores.isnull()].lt(threshold).all(axis=1)

filtered_datanomissingv = htrain_nomissingv[masked_datanomissingv]
print(filtered_datanomissingv.shape)
```

Filtering to ensure only those rows that are not outliers & missing values are retained in a new data frame for further coding
(1252, 22)

[52]:
```
deduplicated_filtered_data = filtered_data.drop_duplicates()
print(deduplicated_filtered_data.shape)
```

(1251, 22)

[53]:
```
pd.set_option('display.max_rows',10)
pd.set_option('display.max_columns',None)

print(deduplicated_filtered_data)
```

```
        x1     x2     x4     x5     x6     x7   x8   x9   x10  x11   x12  \
0     1406  145.0  0.005  0.000  0.002  0.000  0.0  0.0  46.0  0.8   0.0
1      258  127.0  0.012  0.000  0.008  0.004  0.0  0.0  13.0  3.8   0.0
2      479  145.0  0.000  0.000  0.000  0.002  0.0  0.0  57.0  0.5   0.0
3      906  146.0  0.004  0.000  0.005  0.003  0.0  0.0  29.0  1.2   1.0
5       70  144.0  0.001  0.000  0.005  0.000  0.0  0.0  45.0  0.8   2.0
…     …     …      …      …      …      …    …  …  …    …  …    …
1577  1234  125.0  0.002  0.000  0.004  0.001  0.0  0.0  30.0  1.1   3.0
1578  1731  134.0  0.008  0.001  0.010  0.006  0.0  0.0  61.0  1.1   0.0
1579  2077  130.0  0.005  0.001  0.001  0.000  0.0  0.0  72.0  0.9   8.0
1581  1431  144.0  0.000  0.000  0.006  0.000  0.0  0.0  45.0  0.7   0.0
1583   436  151.0  0.000  0.000  0.006  0.006  0.0  0.0  64.0  1.1  26.0

        x13    x15    x16    x17   x18  x19    x20    x21    x22   x23  x24
0       8.6   67.0  104.0  171.0   4.0  0.0  155.0  153.0  154.0   4.0  1.0
1       1.3  138.0   53.0  191.0  12.0  1.0  133.0  126.0  131.0  41.0  0.0
2       7.3   46.0  111.0  157.0   1.0  1.0  150.0  146.0  149.0   6.0  1.0
3       7.0   62.0  107.0  169.0   2.0  2.0  150.0  147.0  149.0   7.0  0.0
5      11.5   30.0  138.0  168.0   3.0  0.0  162.0  157.0  160.0   5.0  1.0
…     …     …      …      …     …    …    …      …      …      …    …
1577  11.0   45.0  102.0  147.0   1.0  0.0  126.0  127.0  128.0   3.0  0.0
1578   3.7  109.0   80.0  189.0  10.0  0.0  156.0  144.0  151.0  61.0  0.0
1579   4.4   31.0  127.0  158.0   2.0  0.0  139.0  139.0  140.0   3.0  0.0
1581   9.8   30.0  139.0  169.0   2.0  0.0  157.0  155.0  157.0   2.0  0.0
1583   3.0  150.0   50.0  200.0  11.0  2.0  156.0  150.0  156.0  38.0  1.0
```

```
[1251 rows x 22 columns]
```

[54]:
```
print("Preprocessing")
```

```
Preprocessing
```

[55]:
```
for col in HealthTrain_data[CATEGORICAL_COLUMNS]:
    print ("---- %s ---" % col)
    print (HealthTrain_data[col].value_counts())
```

```
---- id ---
PA1001    1
PA2065    1
PA2063    1
PA2062    1
PA2061    1
         ..
PA1526    1
PA1525    1
PA1524    1
PA1523    1
PA2584    1
Name: id, Length: 1584, dtype: int64
---- x3 ---
M    842
F    742
Name: x3, dtype: int64
---- x14 ---
O+     410
A+     328
B+     307
A-     165
B-     154
O-     146
AB+     57
AB-     17
Name: x14, dtype: int64
```

[56]:
```
for col in deduplicated_filtered_data:
    print ("---- %s ---" % col)
    print (deduplicated_filtered_data[col].value_counts())
```

```
---- x1 ---
1406    1
886     1
4       1
264     1
1545    1
       ..
```

```
1786    1
2007    1
927     1
1496    1
436     1
Name: x1, Length: 1251, dtype: int64
---- x2 ---
122.0    77
125.0    62
138.0    61
133.0    60
130.0    57
         ..
154.0     4
116.0     3
160.0     1
156.0     1
117.0     1
Name: x2, Length: 48, dtype: int64
---- x4 ---
0.000    496
0.003    101
0.002     96
0.001     79
0.004     77
        …
0.010     29
0.011     27
0.013     16
0.012     14
0.014     10
Name: x4, Length: 15, dtype: int64
---- x5 ---
0.000    812
0.001     96
0.002     54
0.003     54
0.004     35
        …
0.028      1
0.079      1
0.065      1
0.024      1
0.053      1
Name: x5, Length: 55, dtype: int64
---- x6 ---
0.000    197
0.005    184
```

```
0.004    152
0.007    133
0.006    124
        …
0.009     48
0.010     23
0.011      6
0.012      4
0.013      1
Name: x6, Length: 14, dtype: int64
---- x7 ---
0.000    786
0.001     98
0.002     65
0.005     64
0.003     63
        …
0.006     35
0.007     31
0.009     21
0.008     20
0.010      8
Name: x7, Length: 11, dtype: int64
---- x8 ---
0.0    1251
Name: x8, dtype: int64
---- x9 ---
0.000    1217
0.001      34
Name: x9, dtype: int64
---- x10 ---
58.0    41
22.0    32
65.0    32
60.0    32
51.0    31
        ..
14.0     2
80.0     2
86.0     2
83.0     2
82.0     1
Name: x10, Length: 72, dtype: int64
---- x11 ---
0.7    88
0.8    84
0.6    80
0.5    79
```

```
0.4    76
       ..
3.7     2
3.1     1
3.6     1
3.9     1
4.0     1
Name: x11, Length: 39, dtype: int64
---- x12 ---
0.0     690
5.0      31
1.0      31
2.0      29
4.0      24
       …
50.0      1
65.0      1
47.0      1
63.0      1
45.0      1
Name: x12, Length: 66, dtype: int64
---- x13 ---
0.0     27
6.5     18
6.6     17
5.2     17
7.8     17
        ..
16.3     1
24.2     1
14.7     1
13.9     1
19.2     1
Name: x13, Length: 214, dtype: int64
---- x15 ---
39.0     29
42.0     21
102.0    20
27.0     20
33.0     19
         ..
147.0     1
131.0     1
143.0     1
135.0     1
150.0     1
Name: x15, Length: 140, dtype: int64
---- x16 ---
```

```
120.0    33
68.0     28
103.0    26
64.0     24
51.0     23
         ..
151.0     2
152.0     2
150.0     2
156.0     1
154.0     1
Name: x16, Length: 105, dtype: int64
---- x17 ---
157.0    45
158.0    41
171.0    40
156.0    40
159.0    39
         ..
123.0     1
190.0     1
134.0     1
205.0     1
122.0     1
Name: x17, Length: 81, dtype: int64
---- x18 ---
1.0     218
2.0     214
3.0     173
4.0     148
5.0     125
       …
8.0      53
9.0      31
10.0     19
12.0     10
11.0     10
Name: x18, Length: 13, dtype: int64
---- x19 ---
0.0    985
1.0    210
2.0     56
Name: x19, dtype: int64
---- x20 ---
133.0    72
144.0    56
142.0    53
126.0    49
```

```
150.0    48
         ..
105.0     1
164.0     1
186.0     1
180.0     1
179.0     1
Name: x20, Length: 65, dtype: int64
---- x21 ---
141.0    49
144.0    47
140.0    42
125.0    40
132.0    40
         ..
108.0     1
172.0     1
91.0      1
170.0     1
175.0     1
Name: x21, Length: 68, dtype: int64
---- x22 ---
145.0    50
146.0    46
142.0    44
147.0    41
141.0    40
         ..
172.0     1
168.0     1
170.0     1
176.0     1
178.0     1
Name: x22, Length: 66, dtype: int64
---- x23 ---
1.0     169
3.0     120
2.0     120
0.0      97
4.0      72
       …
62.0      1
56.0      1
63.0      1
98.0      1
94.0      1
Name: x23, Length: 82, dtype: int64
---- x24 ---
```

```
 0.0    666
 1.0    501
-1.0     84
Name: x24, dtype: int64
```

[57]:
```
for col in HealthTrain_data[CONTINUOUS_COLUMNS]:
    print ("---- %s ---" % col)
    print (HealthTrain_data[col].value_counts())
```

```
---- x1 ---
1406    1
1495    1
183     1
1240    1
1625    1
       ..
333     1
1786    1
2007    1
927     1
436     1
Name: x1, Length: 1584, dtype: int64
---- x2 ---
133.0    97
122.0    87
130.0    79
138.0    75
125.0    67
         ..
157.0     4
116.0     4
156.0     3
160.0     1
117.0     1
Name: x2, Length: 48, dtype: int64
---- x4 ---
0.000    662
0.003    127
0.002    116
0.001    105
0.004     92
        …
0.015      8
0.017      3
0.016      3
0.019      1
0.018      1
Name: x4, Length: 20, dtype: int64
```

```
---- x5 ---
0.000    959
0.001    120
0.002     73
0.003     67
0.004     36
        …
0.451      1
0.430      1
0.065      1
0.425      1
0.088      1
Name: x5, Length: 90, dtype: int64
---- x6 ---
0.000    249
0.005    228
0.004    187
0.007    158
0.003    157
        …
0.010     35
0.011     11
0.012     10
0.013      2
0.014      2
Name: x6, Length: 15, dtype: int64
---- x7 ---
0.000    923
0.001    120
0.003     92
0.002     87
0.005     85
        …
0.011      8
0.014      7
0.012      7
0.013      6
0.015      3
Name: x7, Length: 16, dtype: int64
---- x8 ---
0.000    1563
0.001       4
Name: x8, dtype: int64
---- x9 ---
0.000    1453
0.002      53
0.001      52
0.003      16
```

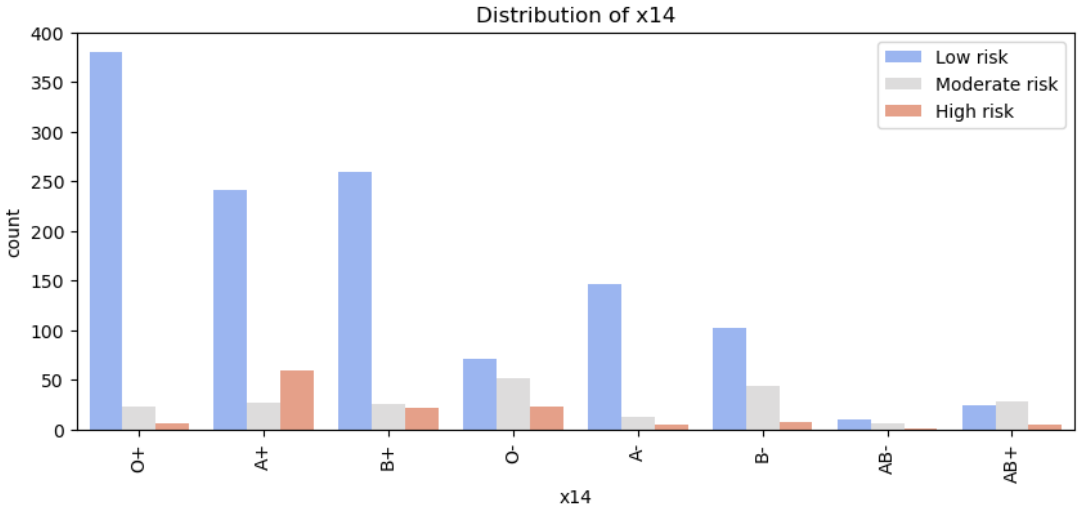
```
0.004       7
0.005       3
Name: x9, dtype: int64
---- x10 ---
58.0    48
63.0    46
60.0    44
65.0    43
64.0    42
        ..
14.0     4
80.0     4
86.0     3
83.0     2
82.0     1
Name: x10, Length: 72, dtype: int64
---- x11 ---
0.7    95
0.8    94
1.3    92
0.4    90
0.5    88
       ..
6.0     1
5.0     1
4.7     1
4.8     1
4.3     1
Name: x11, Length: 55, dtype: int64
---- x12 ---
0.0     920
5.0      36
1.0      35
2.0      31
4.0      29
       …
75.0      1
63.0      1
65.0      1
88.0      1
90.0      1
Name: x12, Length: 87, dtype: int64
---- x13 ---
0.0     105
7.1      22
7.8      21
6.7      20
6.5      20
```

```
       …
16.6      1
23.1      1
24.9      1
19.0      1
19.2      1
Name: x13, Length: 240, dtype: int64
---- x15 ---
39.0     30
102.0    24
27.0     23
98.0     22
42.0     22
         ..
137.0     1
139.0     1
3.0       1
6.0       1
131.0     1
Name: x15, Length: 151, dtype: int64
---- x16 ---
50.0     58
52.0     38
120.0    37
71.0     34
68.0     34
         ..
151.0     2
155.0     2
156.0     1
158.0     1
149.0     1
Name: x16, Length: 107, dtype: int64
---- x17 ---
157.0    53
158.0    50
171.0    48
156.0    46
159.0    43
         ..
123.0     2
210.0     2
128.0     1
205.0     1
122.0     1
Name: x17, Length: 85, dtype: int64
---- x18 ---
1.0     260
```

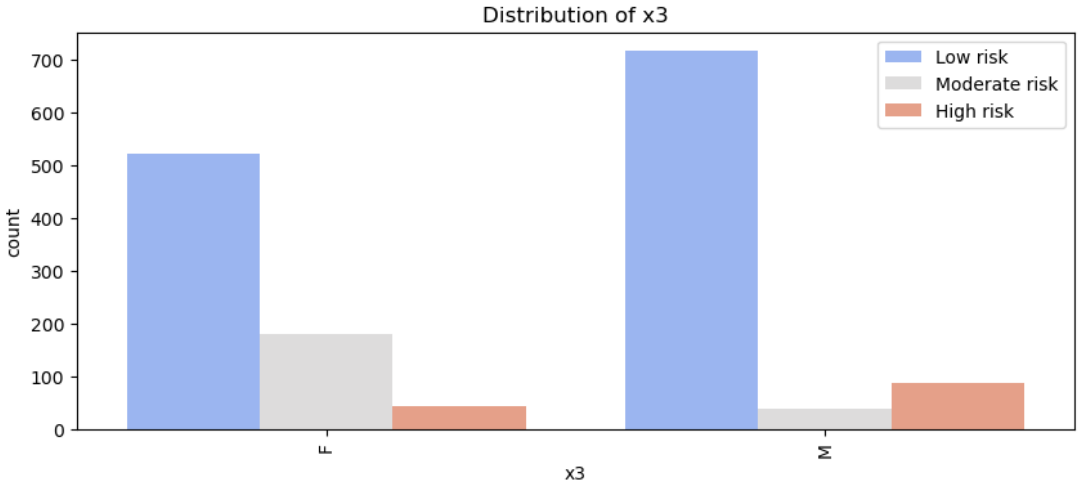
```
2.0     251
3.0     202
4.0     187
5.0     157
       …
13.0      8
14.0      4
16.0      2
18.0      1
15.0      1
Name: x18, Length: 18, dtype: int64
---- x19 ---
0.0     1213
1.0      269
2.0       79
3.0       17
4.0        2
5.0        2
8.0        1
10.0       1
Name: x19, dtype: int64
---- x20 ---
133.0    97
150.0    63
144.0    61
142.0    61
125.0    59
         ..
106.0     1
71.0      1
179.0     1
90.0      1
93.0      1
Name: x20, Length: 86, dtype: int64
---- x21 ---
144.0    53
141.0    50
132.0    46
143.0    46
140.0    46
         ..
178.0     1
81.0      1
180.0     1
170.0     1
88.0      1
Name: x21, Length: 100, dtype: int64
---- x22 ---
```

```
145.0    54
142.0    52
146.0    51
147.0    50
151.0    48
         ..
98.0      1
87.0      1
180.0     1
99.0      1
77.0      1
Name: x22, Length: 92, dtype: int64
---- x23 ---
1.0      190
0.0      139
3.0      124
2.0      124
4.0       76
        …
68.0       1
215.0      1
50.0       1
134.0      1
106.0      1
Name: x23, Length: 118, dtype: int64
---- x24 ---
 0.0    832
 1.0    621
-1.0    131
Name: x24, dtype: int64
```

[58]:
```
def DistPlot(df,hue,column):

    plt.figure(figsize=(10,4))
    plt.xticks(rotation=90)

    sns.countplot(x=column, hue=hue,palette="coolwarm", data=df)

    plt.title("Distribution of " +column)

    plt.legend()
    return
DistPlot(HealthTrain_data,'target','x14')
```



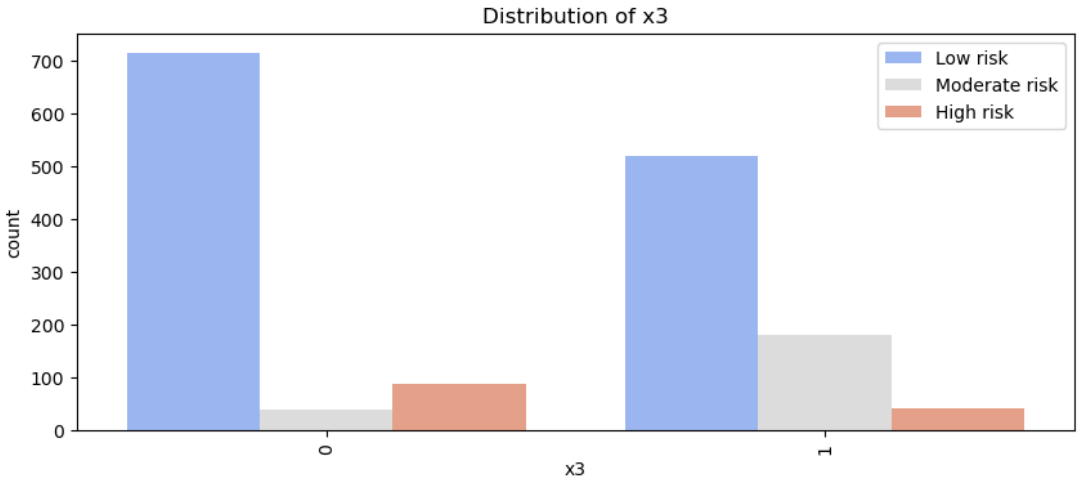
[59]:
```
def DistPlot(df,hue,column):

    plt.figure(figsize=(10,4))
    plt.xticks(rotation=90)

    sns.countplot(x=column, hue=hue,palette="coolwarm", data=df)
    plt.title("Distribution of " +column)
    plt.legend()
    return
DistPlot(HealthTrain_data,'target','x3')
```



[60]:
```
HealthTrain_data["x3"] = np.where(HealthTrain_data["x3"].str.contains("F"), 1,␣
↪0)
print(HealthTrain_data['x3'].dtypes)
```

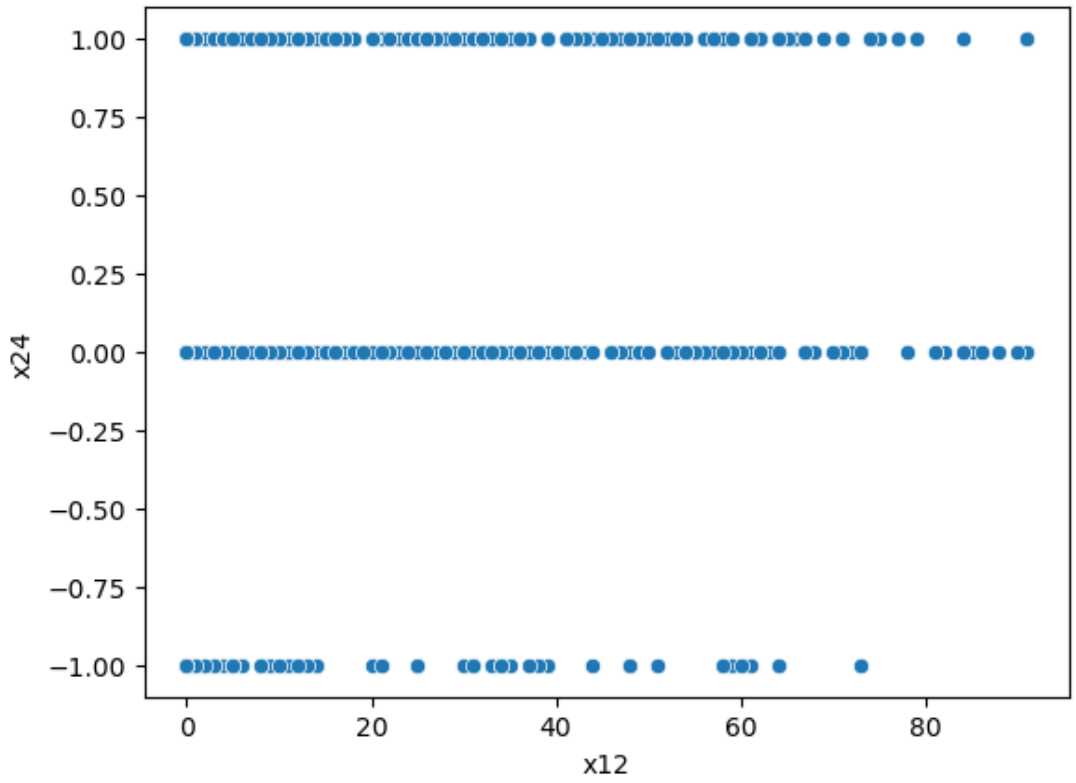
```
int32
```

[61]:
```
DistPlot(HealthTrain_data,'target','x3')
```



[62]:
```
sns.scatterplot(x=HealthTrain_data['x12'], y=HealthTrain_data['x24'])
```

[62]:
```
<AxesSubplot:xlabel='x12', ylabel='x24'>
```



[63]:
```
HealthTrain_data.select_dtypes(include=['object']).columns
```

[63]:
```
Index(['id', 'x14', 'target'], dtype='object')
```

[98]:
```
!pip install category_encoders

import category_encoders as ce
enc = ce.OrdinalEncoder(cols=['id','x3',␣
 ↪'x14'],handle_missing='return_nan',return_df= True)

X=enc.fit_transform(HealthTrain_data)
type(X)
```

```
Requirement already satisfied: category_encoders in
c:\users\jt\anaconda3\lib\site-packages (2.6.1)
Requirement already satisfied: scipy>=1.0.0 in c:\users\jt\anaconda3\lib\site-
packages (from category_encoders) (1.9.1)
Requirement already satisfied: scikit-learn>=0.20.0 in
c:\users\jt\anaconda3\lib\site-packages (from category_encoders) (1.0.2)
Requirement already satisfied: pandas>=1.0.5 in c:\users\jt\anaconda3\lib\site-
packages (from category_encoders) (1.4.4)
Requirement already satisfied: statsmodels>=0.9.0 in
```

```
c:\users\jt\anaconda3\lib\site-packages (from category_encoders) (0.13.2)
Requirement already satisfied: patsy>=0.5.1 in c:\users\jt\anaconda3\lib\site-
packages (from category_encoders) (0.5.2)
Requirement already satisfied: numpy>=1.14.0 in c:\users\jt\anaconda3\lib\site-
packages (from category_encoders) (1.21.5)
Requirement already satisfied: pytz>=2020.1 in c:\users\jt\anaconda3\lib\site-
packages (from pandas>=1.0.5->category_encoders) (2022.1)
Requirement already satisfied: python-dateutil>=2.8.1 in
c:\users\jt\anaconda3\lib\site-packages (from pandas>=1.0.5->category_encoders)
(2.8.2)
Requirement already satisfied: six in c:\users\jt\anaconda3\lib\site-packages
(from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=0.11 in c:\users\jt\anaconda3\lib\site-
packages (from scikit-learn>=0.20.0->category_encoders) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\jt\anaconda3\lib\site-packages (from scikit-
learn>=0.20.0->category_encoders) (2.2.0)
Requirement already satisfied: packaging>=21.3 in
c:\users\jt\anaconda3\lib\site-packages (from
statsmodels>=0.9.0->category_encoders) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
c:\users\jt\anaconda3\lib\site-packages (from
packaging>=21.3->statsmodels>=0.9.0->category_encoders) (3.0.9)
```

[98]:
```
pandas.core.frame.DataFrame
```

[99]:
```
X.head()
```

[99]:

	id	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	\
0	1.0	1406	145.0	1.0	0.005	0.000	0.002	0.000	0.0	0.0	46.0	0.8	
1	2.0	258	127.0	2.0	0.012	0.000	0.008	0.004	0.0	0.0	13.0	3.8	
2	3.0	479	145.0	1.0	0.000	0.000	0.000	0.002	0.0	0.0	57.0	0.5	
3	4.0	906	146.0	1.0	0.004	0.000	0.005	0.003	0.0	0.0	29.0	1.2	
4	5.0	1921	140.0	1.0	0.002	0.003	0.006	0.006	0.0	0.0	62.0	1.6	

	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	\
0	0.0	8.6	1.0	67.0	104.0	171.0	4.0	0.0	155.0	153.0	154.0	4.0	
1	0.0	1.3	2.0	138.0	53.0	191.0	12.0	1.0	133.0	126.0	131.0	41.0	
2	0.0	7.3	1.0	46.0	111.0	157.0	1.0	1.0	150.0	146.0	149.0	6.0	
3	1.0	7.0	1.0	62.0	107.0	169.0	2.0	2.0	150.0	147.0	149.0	7.0	
4	0.0	9.1	3.0	153.0	75.0	228.0	9.0	0.0	142.0	118.0	142.0	20.0	

	x24	target
0	1.0	Low risk
1	0.0	Low risk
2	1.0	Low risk
3	0.0	Low risk
4	0.0	Low risk

[66]:
```
print("Feature selection")
objnum_df = X.select_dtypes(exclude=['object']).copy()
FS_Colsnum=objnum_df.columns
FS_Colsnum
```

```
Feature selection
```

[66]:
```
Index(['id', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10',
       'x11', 'x12', 'x13', 'x14', 'x15', 'x16', 'x17', 'x18', 'x19', 'x20',
       'x21', 'x22', 'x23', 'x24'],
      dtype='object')
```

[67]:
```
X1=X[FS_Colsnum]
y=X.target
```

[68]:
```
print("Correlation in variables")
correlated_v = X1.corr()
correlated_v
```

```
Correlation in variables
```

[68]:

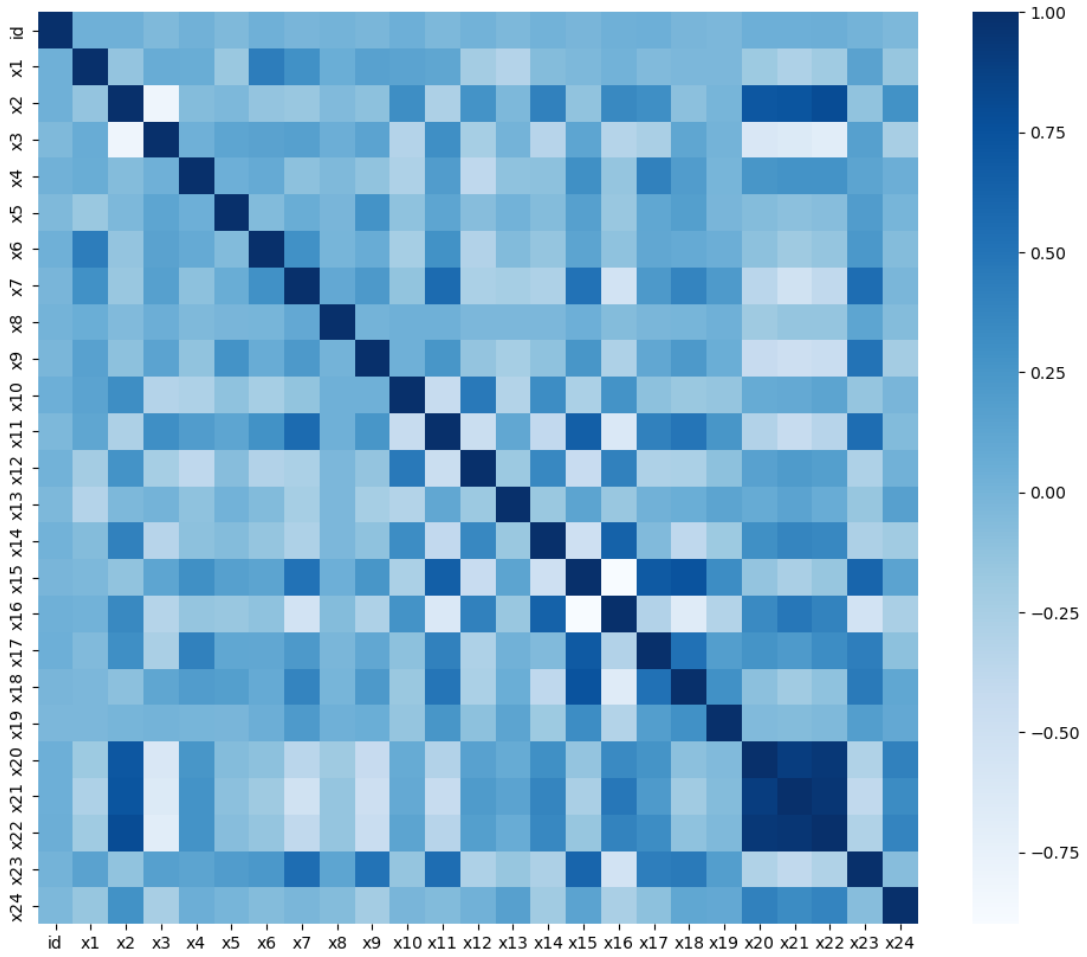
	id	x1	x2	x3	x4	x5	x6	\
id	1.000000	0.029566	0.031876	-0.041106	0.027292	-0.041190	0.029051	
x1	0.029566	1.000000	-0.140849	0.065654	0.064611	-0.171225	0.441475	
x2	0.031876	-0.140849	1.000000	-0.816558	-0.064462	-0.036019	-0.141572	
x3	-0.041106	0.065654	-0.816558	1.000000	0.030652	0.131673	0.149680	
x4	0.027292	0.064611	-0.064462	0.030652	1.000000	0.035747	0.083795	
..	…	…	…	…	…	…	…	
x20	0.040581	-0.186488	0.716999	-0.609279	0.253762	-0.063722	-0.107098	
x21	0.035536	-0.281965	0.730456	-0.642774	0.279833	-0.099306	-0.191931	
x22	0.047157	-0.199382	0.794640	-0.686039	0.279454	-0.077966	-0.145553	
x23	0.008057	0.153051	-0.125936	0.176244	0.132532	0.204304	0.229555	
x24	-0.031305	-0.154393	0.287434	-0.245493	0.045396	-0.006158	-0.063031	

	x7	x8	x9	x10	x11	x12	x13	\
id	-0.012578	0.006168	-0.020131	0.040984	-0.037858	0.015674	-0.031959	
x1	0.290645	0.054606	0.160610	0.141690	0.123397	-0.218873	-0.316328	
x2	-0.161659	-0.048030	-0.107664	0.313653	-0.275446	0.276581	-0.033334	
x3	0.174493	0.047244	0.142323	-0.317180	0.306515	-0.233389	0.006101	
x4	-0.111122	-0.042002	-0.123082	-0.282130	0.204907	-0.380133	-0.117809	
..	…	…	…	…	…	…	…	
x20	-0.355754	-0.191844	-0.431633	0.075051	-0.304893	0.156685	0.077902	
x21	-0.522467	-0.142918	-0.489333	0.090029	-0.435390	0.212512	0.139702	
x22	-0.390918	-0.143435	-0.450616	0.136198	-0.333253	0.178859	0.069769	
x23	0.554405	0.124649	0.508488	-0.149117	0.549538	-0.280306	-0.150921	
x24	-0.021714	-0.066576	-0.223341	-0.010472	-0.057076	0.023520	0.164932	

	x14	x15	x16	x17	x18	x19	x20	\

```
id   0.017125 -0.009603  0.034864  0.036703 -0.014020 -0.028469  0.040581
x1  -0.062672 -0.034931  0.016311 -0.049105 -0.028743 -0.027576 -0.186488
x2   0.410054 -0.123930  0.349450  0.307608 -0.098621 -0.003442  0.716999
x3  -0.330656  0.127566 -0.321904 -0.254173  0.118956  0.006650 -0.609279
x4  -0.105601  0.300302 -0.147888  0.409489  0.202866 -0.003213  0.253762
..        …       …       …       …       …       …
x20  0.300408 -0.138368  0.344051  0.267251 -0.098188 -0.047142  1.000000
x21  0.381483 -0.258080  0.473544  0.220574 -0.202156 -0.068363  0.896618
x22  0.356017 -0.152112  0.393444  0.318945 -0.113037 -0.039798  0.934683
x23 -0.272176  0.608519 -0.539875  0.432849  0.459140  0.189649 -0.297319
x24 -0.206723  0.146532 -0.259174 -0.109218  0.112561  0.098500  0.412951

          x21       x22       x23       x24
id   0.035536  0.047157  0.008057 -0.031305
x1  -0.281965 -0.199382  0.153051 -0.154393
x2   0.730456  0.794640 -0.125936  0.287434
x3  -0.642774 -0.686039  0.176244 -0.245493
x4   0.279833  0.279454  0.132532  0.045396
..        …       …       …       …
x20  0.896618  0.934683 -0.297319  0.412951
x21  1.000000  0.950510 -0.390207  0.330283
x22  0.950510  1.000000 -0.294820  0.389421
x23 -0.390207 -0.294820  1.000000 -0.082045
x24  0.330283  0.389421 -0.082045  1.000000

[25 rows x 25 columns]
```

[69]:
```
plt.figure(figsize=(12,10))
sns.heatmap(correlated_v, annot=False, cmap=plt.cm.Blues)
plt.show()
```



[102]:
```
print("Spliting the data into training set and validation set.")

from sklearn.model_selection import train_test_split

train_X, val_X, train_y, val_y= train_test_split (X,y, test_size=0.
↪2,random_state=1)

print("Train set description \n", train_X.describe())

print("Validation set description \n",val_X.describe())
```

```
Spliting the data into training set and validation set.
Train set description
                id           x1           x2           x3           x4  \
count  1267.000000  1267.000000  1267.000000  1267.000000  1267.000000
mean    789.262036  1063.404893   133.053670     1.540647     0.003197
std     457.381484   617.024162    10.083159     0.498542     0.003841
```

```
min       1.000000     0.000000   106.000000     1.000000     0.000000
25%     391.500000   543.500000   125.000000     1.000000     0.000000
50%     789.000000  1069.000000   133.000000     2.000000     0.002000
75%    1188.500000  1598.500000   140.000000     2.000000     0.006000
max    1584.000000  2125.000000   160.000000     2.000000     0.019000

                x5           x6           x7           x8           x9  \
count  1238.000000  1267.000000  1267.000000  1255.000000  1267.000000
mean      0.010211     0.004430     0.001878     0.000003     0.000160
std       0.049419     0.002909     0.002933     0.000056     0.000582
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.002000     0.000000     0.000000     0.000000
50%       0.000000     0.005000     0.000000     0.000000     0.000000
75%       0.003000     0.007000     0.003000     0.000000     0.000000
max       0.470000     0.014000     0.015000     0.001000     0.005000

               x10          x11          x12          x13          x14  \
count  1244.000000  1267.000000  1267.000000  1267.000000  1267.000000
mean     47.034566     1.355170     9.846093     8.281058     3.077348
std      17.285532     0.897827    18.387117     5.831938     1.930961
min      13.000000     0.200000     0.000000     0.000000     1.000000
25%      32.000000     0.700000     0.000000     4.600000     1.000000
50%      49.000000     1.200000     0.000000     7.500000     3.000000
75%      61.000000     1.800000    10.000000    10.700000     5.000000
max      86.000000     6.900000    91.000000    50.700000     8.000000

               x15          x16          x17          x18          x19  \
count  1267.000000  1267.000000  1267.000000  1267.000000  1267.000000
mean     70.951066    93.126283   164.077348     4.138122     0.323599
std      39.188056    29.491716    18.224561     2.993520     0.683433
min       3.000000    50.000000   122.000000     0.000000     0.000000
25%      37.000000    66.000000   151.500000     2.000000     0.000000
50%      69.000000    93.000000   162.000000     4.000000     0.000000
75%     100.000000   120.000000   175.000000     6.000000     0.000000
max     176.000000   156.000000   238.000000    18.000000     8.000000

               x20          x21          x22          x23          x24
count  1267.000000  1267.000000  1267.000000  1267.000000  1267.000000
mean    137.059984   134.108129   137.612470    18.534333     0.289661
std      16.470843    15.912780    14.742463    28.154410     0.622571
min      60.000000    73.000000    77.000000     0.000000    -1.000000
25%     127.000000   124.000000   128.000000     2.000000     0.000000
50%     139.000000   135.000000   139.000000     8.000000     0.000000
75%     148.000000   145.000000   147.500000    24.000000     1.000000
max     187.000000   180.000000   183.000000   269.000000     1.000000
Validation set description
                id           x1           x2           x3           x4  \
count   317.000000   317.000000  317.000000  317.000000  317.000000
```

```
mean    805.441640  1012.353312  134.274448    1.495268    0.003060
std     457.996601   611.135016    9.627743    0.500768    0.003741
min       4.000000     6.000000  112.000000    1.000000    0.000000
25%     419.000000   485.000000  127.000000    1.000000    0.000000
50%     803.000000   951.000000  134.000000    1.000000    0.001000
75%    1186.000000  1548.000000  142.000000    2.000000    0.006000
max    1579.000000  2124.000000  158.000000    2.000000    0.015000

                x5          x6          x7     x8          x9          x10  \
count   302.000000  317.000000  317.000000  312.0  317.000000  313.000000
mean      0.008659    0.004016    0.001757    0.0    0.000145   47.332268
std       0.045291    0.003080    0.002969    0.0    0.000635   17.231776
min       0.000000    0.000000    0.000000    0.0    0.000000   13.000000
25%       0.000000    0.001000    0.000000    0.0    0.000000   32.000000
50%       0.000000    0.004000    0.000000    0.0    0.000000   50.000000
75%       0.002000    0.006000    0.003000    0.0    0.000000   61.000000
max       0.477000    0.013000    0.014000    0.0    0.005000   84.000000

               x11         x12         x13         x14         x15         x16  \
count   317.000000  317.000000  317.000000  317.000000  317.000000  317.000000
mean      1.268454   10.659306    8.152050    3.141956   68.242902   94.977918
std       0.902220   19.058637    5.599088    1.884826   38.192065   29.997513
min       0.200000    0.000000    0.000000    1.000000    6.000000   50.000000
25%       0.600000    0.000000    4.800000    2.000000   35.000000   68.000000
50%       1.100000    0.000000    7.300000    3.000000   63.000000   95.000000
75%       1.600000   14.000000   10.700000    4.000000   99.000000  120.000000
max       7.000000   91.000000   33.500000    8.000000  150.000000  158.000000

               x17         x18         x19         x20         x21         x22  \
count   317.000000  317.000000  317.000000  317.000000  317.000000  317.000000
mean    163.220820    3.766562    0.328076  138.429022  136.280757  139.227129
std      16.596147    2.755159    0.845465   16.405044   14.873306   14.082448
min     129.000000    0.000000    0.000000   71.000000   84.000000   99.000000
25%     152.000000    2.000000    0.000000  129.000000  127.000000  130.000000
50%     161.000000    3.000000    0.000000  140.000000  137.000000  140.000000
75%     172.000000    5.000000    0.000000  150.000000  147.000000  149.000000
max     200.000000   14.000000   10.000000  186.000000  182.000000  186.000000

               x23         x24
count   317.000000  317.000000
mean     18.104101    0.388013
std      29.283077    0.582646
min       0.000000   -1.000000
25%       2.000000    0.000000
50%       6.000000    0.000000
75%      21.000000    1.000000
max     215.000000    1.000000
```

[71]:
```
train_y.value_counts()
```

[71]:
```
Low risk         998
Moderate risk    170
High risk         99
Name: target, dtype: int64
```

[72]:
```
from sklearn.svm import LinearSVC
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

labelencoder = LabelEncoder()
one_hencoder = OneHotEncoder()

train_y_encoded = labelencoder.fit_transform(train_y)
train_X_encoded = one_hencoder.fit_transform(train_X)


Classifiersvc = LinearSVC()
Classifiersvc.fit(train_X_encoded, train_y_encoded);
```

[73]:
```
print("LinearSVC Impletented \n",Classifiersvc)
```

```
LinearSVC Impletented
 LinearSVC()
```

[74]:
```
np.set_printoptions(threshold=np.inf)
y_predict = Classifiersvc.predict(train_X_encoded)
print(y_predict)
```

```
[1 1 1 2 2 1 1 1 1 2 1 1 2 1 1 2 1 2 1 1 1 1 0 1 1 1 1 0 2 1 1 2 1 1 1 1 1
 1 1 2 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1
 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1
 1 2 1 1 0 0 1 1 1 1 1 1 1 0 1 2 1 2 2 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 1 2
 1 2 1 2 1 1 0 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 2 1 1 2 2 0 1 1 1 1 2 1 1 1
 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 2 1
 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1
 2 2 1 1 2 1 1 1 1 1 1 0 1 1 0 1 1 1 2 1 1 1 1 1 2 1 1 1 0 1 1 2 2 1 1 1 1
 1 2 1 1 1 1 0 1 1 1 2 1 1 0 1 2 1 1 1 1 1 0 1 0 1 1 1 1 0 2 1 1 1 1 2 1 1
 1 1 1 1 1 1 2 0 1 1 0 1 1 2 1 1 1 1 2 2 1 0 1 1 1 2 1 1 1 1 2 2 0 1 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1
 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 2
 2 1 2 2 1 1 1 1 1 1 1 1 0 1 1 1 2 2 1 1 1 1 0 1 2 1 1 1 1 0 1 2 1 0 0 1 1
 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 0 0 2 1 0 2 1 1 1 0 2 1 2 1 1 2
 0 1 1 2 1 0 1 1 1 1 1 0 1 0 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1
 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1 1 0 1 0 2 0 1
 1 1 0 1 1 2 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 2 1 0 1 2 2 1 1 2
```

```
0 1 2 1 1 1 0 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 0 1 0 1 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1 2 1 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1
1 1 1 2 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 2 1 1 1 1 1 1 1 1 1 1 2 1
2 1 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 0 1 1 1 2 1 1 1 1
1 2 1 2 0 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1
2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2 1 1 0 2 1 1 1 1 1 1 1 1
1 1 1 0 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 0 1 1 1 1 1 1 2 1 0 2 1 1
1 1 2 2 1 1 2 1 1 2 1 1 1 0 2 2 1 2 1 0 1 1 1 1 1 0 1 2 0 1 2 1 0 1 1 1 1
1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1
1 2 1 1 1 1 1 0 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 2 1
1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 2 1 1 1 2 1 1 1 1 1 0 2 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 0 2 1 1 0 1 2 0 1 1 1 0 1 2 2 1 1 1 1 2 1 1 1 1 1
1 1 1 2 1 1 0 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 2 1 1 1 2
1 1 1 1 1 2 1 1 2 1 1 1 2 0 1 1 1 2 1 2 1 1 2 0 1 0 1 1 1 1 1 1 1 1 1 0 2 1
1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 0 2 1 2 1 2 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1]
```

[75]:
```
from sklearn.naive_bayes import GaussianNB

naivebayes = GaussianNB()
naivebayes.fit(train_X_encoded.toarray(), train_y_encoded);
```

[76]:
```
print("NaiveBayes classifier Implemented \n",naivebayes)
```

```
NaiveBayes classifier Implemented
 GaussianNB()
```

[77]:
```
print("Creating KNN Classifier")
```

```
Creating KNN Classifier
```

[78]:
```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.utils import *
from sklearn.utils.validation import check_is_fitted
from sklearn.utils.multiclass import unique_labels
from sklearn.metrics.pairwise import euclidean_distances

class KNN(BaseEstimator, ClassifierMixin):
    def __init__(self):
        pass

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.classes_ = unique_labels(y)
        self._X = X
        self._y = y
        return self
```

```
    def predict(self, X):
        check_is_fitted(self)
        X = check_array(X)
        closest = np.argmin(euclidean_distances(X, self._X), axis=1)
        return self._y[closest]
Classifier_KNN = KNN()
Classifier_KNN.fit(train_X_encoded.toarray(), train_y_encoded);
print("KNN implemented, when k=1 \n", Classifier_KNN)
```

```
KNN implemented, when k=1
 KNN()
```

[79]:
```
from sklearn.metrics import accuracy_score

classifiers = [Classifiersvc, naivebayes, Classifier_KNN]


scores = [accuracy_score(clf.predict(train_X_encoded.toarray()),␣
 ↪train_y_encoded) for clf in classifiers]

index = np.argmax(scores)

print(scores)

print(classifiers[index])
print(scores[index])
```

```
[1.0, 1.0, 1.0]
LinearSVC()
1.0
```

[80]:
```
y_svc = Classifiersvc.predict(train_X_encoded)

accuracy = accuracy_score(train_y_encoded, y_svc)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
Accuracy: 100.00%
```

[81]:
```
print(val_X)
```

```
          id    x1     x2   x3     x4     x5     x6     x7   x8   x9   x10  \
1298  1299.0   726  129.0  2.0  0.000  0.008  0.002  0.000  0.0  0.0  55.0
654    655.0  1206  136.0  1.0  0.005  0.000  0.006  0.000  0.0  0.0  46.0
572    573.0  1966  138.0  1.0  0.011    NaN  0.005  0.000  0.0  0.0  57.0
1132  1133.0   935  126.0  2.0  0.003  0.000  0.005  0.001  0.0  0.0  30.0
628    629.0   884  136.0  1.0  0.001  0.000  0.004  0.000  0.0  0.0  39.0
…        …    …      …    …      …      …      …      …    …    …    …
1289  1290.0   691  133.0  2.0  0.007  0.455  0.003  0.003  0.0  0.0  25.0
1196  1197.0   376  141.0  1.0  0.005  0.023  0.002  0.001  0.0  0.0  53.0
```

```
1184  1185.0  1548  149.0  1.0  0.000  0.000  0.007  0.000  0.0  0.0  62.0
321    322.0   522  158.0  1.0  0.008  0.027  0.002  0.000  0.0  0.0  41.0
56      57.0  1110  122.0  2.0  0.005  0.000  0.000  0.000  0.0  0.0  20.0

      x11   x12   x13  x14    x15    x16    x17   x18  x19    x20    x21  \
1298  0.6   0.0  10.3  1.0   58.0  101.0  159.0   3.0  0.0  140.0  138.0
654   0.8   1.0   7.4  5.0   39.0  119.0  158.0   1.0  1.0  144.0  143.0
572   0.7   0.0   3.7  4.0   48.0  135.0  183.0   3.0  0.0  152.0  160.0
1132  1.2   0.0  10.8  1.0   52.0   95.0  147.0   3.0  1.0  136.0  131.0
628   0.8  28.0   6.6  6.0   33.0  124.0  157.0   4.0  0.0  141.0  139.0
…    …    …    …    …  …     …      …      …    …  …    …      …
1289  1.7   0.0  13.3  3.0  106.0   68.0  174.0   7.0  1.0  143.0  132.0
1196  1.0  28.0   5.2  2.0  123.0   57.0  180.0  13.0  0.0  154.0  149.0
1184  0.4  46.0   5.5  8.0   14.0  146.0  160.0   1.0  0.0  154.0  153.0
321   0.9   0.0  13.2  8.0   47.0  151.0  198.0   4.0  0.0  186.0  178.0
56    2.1   0.0   9.8  1.0   55.0  101.0  156.0   4.0  1.0  127.0  126.0

        x22   x23  x24         target
1298  140.0   2.0  0.0  Moderate risk
654   145.0   1.0  0.0       Low risk
572   161.0  21.0  1.0       Low risk
1132  134.0   7.0  1.0       Low risk
628   141.0   1.0  0.0       Low risk
…       …    …  …             …
1289  138.0  41.0  0.0       Low risk
1196  154.0  10.0  1.0  Moderate risk
1184  155.0   0.0  0.0  Moderate risk
321   180.0  15.0  0.0       Low risk
56    127.0   7.0  0.0       Low risk

[317 rows x 26 columns]
```

[110]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from scipy.sparse import hstack,csr_matrix


labelencoder = LabelEncoder()
one_hencoder = OneHotEncoder()

val_y_encoded = labelencoder.fit_transform(val_y)
val_X_encoded = one_hencoder.fit_transform(val_X)

if val_X_encoded.shape[1] != train_X_encoded.shape[1]:
    train_columns = train_X_encoded.shape[1]
    val_columns = val_X_encoded.shape[1]
```

```
    if val_columns < train_columns:
        missing_columns = train_columns - val_columns
        missing_columns_num = [f'missing_feature_{i}' for i in␣
↪range(missing_columns)]
        val_X_encoded = hstack([val_X_encoded, csr_matrix((val_X_encoded.
↪shape[0], missing_columns))], format='csr')

predictions_Valset = Classifiersvc.predict(val_X_encoded)
print(predictions_Valset)

accuracy = accuracy_score(val_y_encoded, predictions_Valset)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
[1 2 0 1 0 2 0 1 1 2 2 2 2 2 2 2 1 0 2 2 2 2 1 1 1 2 1 0 2 0 0 2 2 0 1 0 1
 1 2 0 2 2 1 1 0 0 2 2 0 0 0 0 2 1 2 0 1 0 0 0 1 0 2 2 0 0 1 2 1 0 2 2 2 2
 1 2 0 0 1 1 1 0 0 2 2 0 1 2 0 1 2 2 0 2 2 2 0 0 1 2 2 2 2 2 2 2 1 1 0 0 2
 1 1 2 2 1 2 1 2 0 1 2 2 2 0 0 2 2 0 0 2 0 0 0 1 2 2 1 0 0 2 0 0 1 0 0 2 2
 1 1 1 2 1 2 0 1 1 1 2 2 1 1 1 2 2 2 0 1 2 1 0 2 1 2 0 2 1 1 2 0 0 1 1 1 2
 1 1 2 2 0 0 2 2 1 0 2 2 2 1 2 0 0 0 1 1 0 2 2 1 0 0 2 1 1 1 2 1 0 2 1 1 2
 2 2 1 0 2 2 1 2 2 0 2 1 0 2 2 1 0 1 0 2 0 2 2 2 0 2 1 2 2 1 0 2 2 1 1 2 0
 0 2 1 1 2 0 2 2 0 0 1 2 2 1 2 2 2 0 0 2 0 0 2 2 1 2 2 0 2 2 1 0 0 0 1 0 1
 2 2 0 2 2 0 2 2 1 0 0 0 0 1 2 2 2 2 2 0 1]
Accuracy: 24.92%
```

[83]:
```
print(train_X_encoded.shape)
```

```
(1267, 3913)
```

[84]:
```
print(val_X_encoded.shape)
```

```
(317, 1659)
```

[133]:
```
from sklearn.metrics import accuracy_score
classifiers = [Classifiersvc, naivebayes, Classifier_KNN]
scores = [accuracy_score(clf.predict(val_X_encoded.toarray()), val_y_encoded)␣
↪for clf in classifiers]
index = np.argmax(scores)
print(scores)
print(classifiers[index])
print(scores[index])
print("Highest Accuracy percentage: {:.2f}%".format(scores[index]* 100))
```

```
[0.24921135646687698, 0.7476340694006309, 0.6214511041009464]
GaussianNB()
0.7476340694006309
Highest Accuracy percentage: 74.76%
```

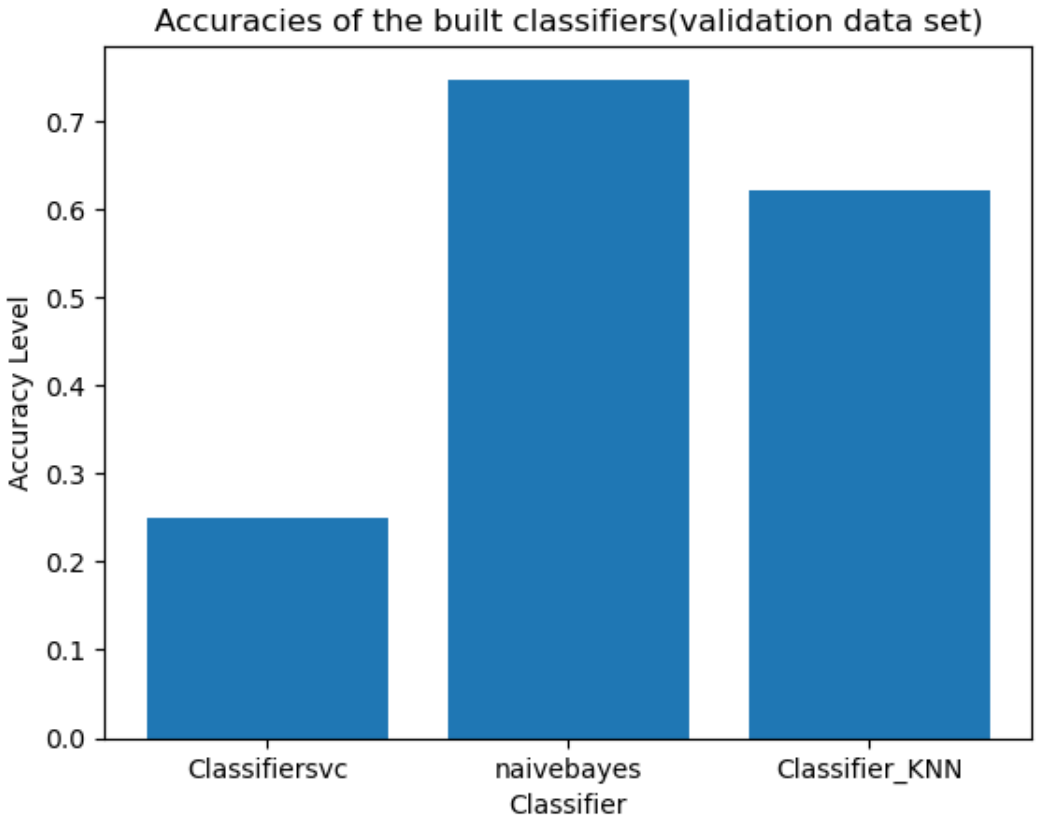
[177]:
```
from sklearn.metrics import precision_score,recall_score
predictions_Valset = naivebayes.predict(val_X_encoded.toarray())
```

```
#print(predictions_Valset)
accuracy = accuracy_score(val_y_encoded, predictions_Valset)
precision = precision_score(val_y_encoded, predictions_Valset, average='macro')
recall = recall_score(val_y_encoded, predictions_Valset, average='macro')
print("Accuracy: {:.2f}%".format(accuracy * 100))
print("Precision: {:.2f}%".format(precision * 100))
print("Recall: {:.2f}%".format(recall * 100))
```

```
Accuracy: 74.76%
Precision: 24.92%
Recall: 33.33%
```

```
C:\Users\JT\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[135]:
```
classifiers_names = ['Classifiersvc', 'naivebayes', 'Classifier_KNN']
accuracy_p = [0.24921135646687698, 0.7476340694006309, 0.6214511041009464]
plt.bar(classifiers_names,accuracy_p)
plt.xlabel('Classifier')
plt.ylabel('Accuracy Level')
plt.title('Accuracies of the built classifiers(validation data set)')

plt.show()
```



[136]:
```
print("Test data and predictions")
```

```
Test data and predictions
```

[137]:
```
HealthTest_path = "C:/Users/JT/DataSetsforDM/health_test.csv"
HealthTest_data = pd.read_csv(HealthTest_path)
print(HealthTest_data.shape)
HealthTest_data
```

```
(528, 25)
```

[137]:

	id	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	\
0	PA3001	767	135	F	0.000	0.000	0.000	0.000	0.0	0.0	67.0	0.4	
1	PA3002	1592	134	F	0.000	0.000	0.010	0.000	0.0	0.0	27.0	2.8	
2	PA3003	1115	122	M	0.000	0.000	0.000	0.000	0.0	0.0	19.0	1.9	
3	PA3004	299	148	F	0.000	NaN	0.000	0.000	0.0	0.0	72.0	0.3	
4	PA3005	1194	133	M	0.003	0.000	0.005	0.000	0.0	0.0	38.0	1.1	
..	…	…	…	..	…	…	…	…	…	…	…	…	
523	PA3524	1071	133	M	0.001	0.000	0.007	0.004	0.0	0.0	27.0	1.7	
524	PA3525	364	135	F	0.000	0.016	0.002	0.000	0.0	0.0	70.0	0.3	
525	PA3526	531	142	F	0.016	0.060	0.004	0.000	0.0	0.0	38.0	1.3	

```
526  PA3527   878  136  F  0.002  0.000  0.006  0.000  0.0  0.0  39.0  0.9
527  PA3528  1733  134  F  0.008    NaN  0.009  0.004  0.0  0.0  60.0  1.2

     x12   x13 x14  x15  x16  x17  x18  x19  x20  x21  x22  x23  x24
0     69   4.6  B+   76   67  143    2    0  137  136  138    0    1
1      8  26.1  O+   74   89  163    7    1  138  134  138   13    0
2      0  15.1  O+   39  103  142    1    0  120  120  122    3    0
3     59   4.3  O-   14  139  153    1    0  150  148  150    0    1
4      0  12.5  A-   58  113  171    5    1  150  147  149    5    0
..   …    …   ..   …   …   …   …   …   …   …   …   …   …
523   13  11.4  O+   95   82  177    4    0  147  133  138   43    0
524   84   3.5  O-    9  132  141    1    0  136  136  137    0    0
525    0   0.0  B+  130   68  198    5    0  180  173  177   14    1
526    8  11.2  O+   47  107  154    1    0  138  139  139    2    0
527    0   4.9  O+  109   80  189    4    1  156  147  151   40    0

[528 rows x 25 columns]
```

[142]:
```
print("Unsupervised Clustering")
HealthTrain_data.head()
```

```
Unsupervised Clustering
```

[142]:
```
       id    x1     x2  x3     x4     x5     x6     x7   x8   x9   x10  x11  \
0  PA1001  1406  145.0   1  0.005  0.000  0.002  0.000  0.0  0.0  46.0  0.8
1  PA1002   258  127.0   0  0.012  0.000  0.008  0.004  0.0  0.0  13.0  3.8
2  PA1003   479  145.0   1  0.000  0.000  0.000  0.002  0.0  0.0  57.0  0.5
3  PA1004   906  146.0   1  0.004  0.000  0.005  0.003  0.0  0.0  29.0  1.2
4  PA1005  1921  140.0   1  0.002  0.003  0.006  0.006  0.0  0.0  62.0  1.6

   x12  x13 x14    x15    x16    x17   x18  x19    x20    x21    x22   x23  \
0  0.0  8.6  O+   67.0  104.0  171.0   4.0  0.0  155.0  153.0  154.0   4.0
1  0.0  1.3  A+  138.0   53.0  191.0  12.0  1.0  133.0  126.0  131.0  41.0
2  0.0  7.3  O+   46.0  111.0  157.0   1.0  1.0  150.0  146.0  149.0   6.0
3  1.0  7.0  O+   62.0  107.0  169.0   2.0  2.0  150.0  147.0  149.0   7.0
4  0.0  9.1  B+  153.0   75.0  228.0   9.0  0.0  142.0  118.0  142.0  20.0

   x24    target
0  1.0  Low risk
1  0.0  Low risk
2  1.0  Low risk
3  0.0  Low risk
4  0.0  Low risk
```

[144]:
```
HealthTrain_data.describe()
```

[144]:
```
                x1           x2           x3           x4           x5  \
count  1584.000000  1584.000000  1584.000000  1584.000000  1540.000000
```

mean	1053.188131	133.297980	0.468434	0.003169	0.009906
std	615.996716	10.002632	0.499160	0.003821	0.048627
min	0.000000	106.000000	0.000000	0.000000	0.000000
25%	523.750000	126.000000	0.000000	0.000000	0.000000
50%	1049.500000	133.000000	0.000000	0.002000	0.000000
75%	1583.250000	141.000000	1.000000	0.006000	0.003000
max	2125.000000	160.000000	1.000000	0.019000	0.477000

	x6	x7	x8	x9	x10 \
count	1584.000000	1584.000000	1567.000000	1584.000000	1557.000000
mean	0.004347	0.001854	0.000003	0.000157	47.094412
std	0.002948	0.002940	0.000050	0.000593	17.269621
min	0.000000	0.000000	0.000000	0.000000	13.000000
25%	0.002000	0.000000	0.000000	0.000000	32.000000
50%	0.004000	0.000000	0.000000	0.000000	49.000000
75%	0.007000	0.003000	0.000000	0.000000	61.000000
max	0.014000	0.015000	0.001000	0.005000	86.000000

	x11	x12	x13	x15	x16 \
count	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000
mean	1.337816	10.008838	8.255240	70.409091	93.496843
std	0.899092	18.520206	5.784579	38.993892	29.593370
min	0.200000	0.000000	0.000000	3.000000	50.000000
25%	0.700000	0.000000	4.600000	36.000000	67.000000
50%	1.200000	0.000000	7.400000	68.000000	93.000000
75%	1.700000	11.000000	10.700000	100.000000	120.000000
max	7.000000	91.000000	50.700000	176.000000	158.000000

	x17	x18	x19	x20	x21 \
count	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000
mean	163.905934	4.063763	0.324495	137.333965	134.542929
std	17.908749	2.950268	0.718499	16.461643	15.729735
min	122.000000	0.000000	0.000000	60.000000	73.000000
25%	152.000000	2.000000	0.000000	128.000000	125.000000
50%	162.000000	3.000000	0.000000	139.000000	136.000000
75%	174.000000	6.000000	0.000000	148.000000	145.250000
max	238.000000	18.000000	10.000000	187.000000	182.000000

	x22	x23	x24
count	1584.000000	1584.000000	1584.000000
mean	137.935606	18.448232	0.309343
std	14.622680	28.375002	0.615868
min	77.000000	0.000000	-1.000000
25%	128.000000	2.000000	0.000000
50%	139.000000	7.000000	0.000000
75%	148.000000	23.000000	1.000000
max	186.000000	269.000000	1.000000

[145]:
```
HealthTrain_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1584 entries, 0 to 1583
Data columns (total 26 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      1584 non-null   object
 1   x1      1584 non-null   int64
 2   x2      1584 non-null   float64
 3   x3      1584 non-null   int32
 4   x4      1584 non-null   float64
 5   x5      1540 non-null   float64
 6   x6      1584 non-null   float64
 7   x7      1584 non-null   float64
 8   x8      1567 non-null   float64
 9   x9      1584 non-null   float64
 10  x10     1557 non-null   float64
 11  x11     1584 non-null   float64
 12  x12     1584 non-null   float64
 13  x13     1584 non-null   float64
 14  x14     1584 non-null   object
 15  x15     1584 non-null   float64
 16  x16     1584 non-null   float64
 17  x17     1584 non-null   float64
 18  x18     1584 non-null   float64
 19  x19     1584 non-null   float64
 20  x20     1584 non-null   float64
 21  x21     1584 non-null   float64
 22  x22     1584 non-null   float64
 23  x23     1584 non-null   float64
 24  x24     1584 non-null   float64
 25  target  1584 non-null   object
dtypes: float64(21), int32(1), int64(1), object(3)
memory usage: 315.7+ KB
```

[149]:
```
HealthTrain_data.drop(['id'], axis=1, inplace=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_17320\3496117139.py in <module>
----> 1 HealthTrain_data.drop(['id'], axis=1, inplace=True)

~\anaconda3\lib\site-packages\pandas\util\_decorators.py in wrapper(*args,␣
 ↪**kwargs)
    309                     stacklevel=stacklevel,
    310                 )
--> 311             return func(*args, **kwargs)
```

```
    312
    313         return wrapper

~\anaconda3\lib\site-packages\pandas\core\frame.py in drop(self, labels, axis,␣
 ↪index, columns, level, inplace, errors)
   4955                 weight  1.0     0.8
   4956         """
-> 4957         return super().drop(
   4958             labels=labels,
   4959             axis=axis,

~\anaconda3\lib\site-packages\pandas\core\generic.py in drop(self, labels, axis␣
 ↪index, columns, level, inplace, errors)
   4265         for axis, labels in axes.items():
   4266             if labels is not None:
-> 4267                 obj = obj._drop_axis(labels, axis, level=level,␣
 ↪errors=errors)
   4268
   4269         if inplace:

~\anaconda3\lib\site-packages\pandas\core\generic.py in _drop_axis(self, labels␣
 ↪axis, level, errors, consolidate, only_slice)
   4309                 new_axis = axis.drop(labels, level=level, errors=errors
   4310             else:
-> 4311                 new_axis = axis.drop(labels, errors=errors)
   4312             indexer = axis.get_indexer(new_axis)
   4313

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in drop(self, labels,␣
 ↪errors)
   6659         if mask.any():
   6660             if errors != "ignore":
-> 6661                 raise KeyError(f"{list(labels[mask])} not found in axis")
   6662             indexer = indexer[~mask]
   6663         return self.delete(indexer)

KeyError: "['id'] not found in axis"
```

[150]:
```
HealthTrain_data.describe()

HealthTrain_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1584 entries, 0 to 1583
Data columns (total 25 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
```

```
 0   x1      1584 non-null   int64
 1   x2      1584 non-null   float64
 2   x3      1584 non-null   int32
 3   x4      1584 non-null   float64
 4   x5      1540 non-null   float64
 5   x6      1584 non-null   float64
 6   x7      1584 non-null   float64
 7   x8      1567 non-null   float64
 8   x9      1584 non-null   float64
 9   x10     1557 non-null   float64
 10  x11     1584 non-null   float64
 11  x12     1584 non-null   float64
 12  x13     1584 non-null   float64
 13  x14     1584 non-null   object
 14  x15     1584 non-null   float64
 15  x16     1584 non-null   float64
 16  x17     1584 non-null   float64
 17  x18     1584 non-null   float64
 18  x19     1584 non-null   float64
 19  x20     1584 non-null   float64
 20  x21     1584 non-null   float64
 21  x22     1584 non-null   float64
 22  x23     1584 non-null   float64
 23  x24     1584 non-null   float64
 24  target  1584 non-null   object
dtypes: float64(21), int32(1), int64(1), object(2)
memory usage: 303.3+ KB
```

[153]:
```
pd.set_option('display.max_rows', None)

HealthTrain_data.isna().mean() * 100
```

[153]:
```
x1        0.000000
x2        0.000000
x3        0.000000
x4        0.000000
x5        2.777778
x6        0.000000
x7        0.000000
x8        1.073232
x9        0.000000
x10       1.704545
x11       0.000000
x12       0.000000
x13       0.000000
x14       0.000000
x15       0.000000
```

```
x16       0.000000
x17       0.000000
x18       0.000000
x19       0.000000
x20       0.000000
x21       0.000000
x22       0.000000
x23       0.000000
x24       0.000000
target    0.000000
dtype: float64
```

[154]:
```
HealthTrain_data['x5'].fillna(HealthTrain_data['x5'].median(), inplace=True)
HealthTrain_data['x8'].fillna(HealthTrain_data['x8'].median(), inplace=True)
HealthTrain_data['x10'].fillna(HealthTrain_data['x10'].median(), inplace=True)

HealthTrain_data.describe()
```

[154]:

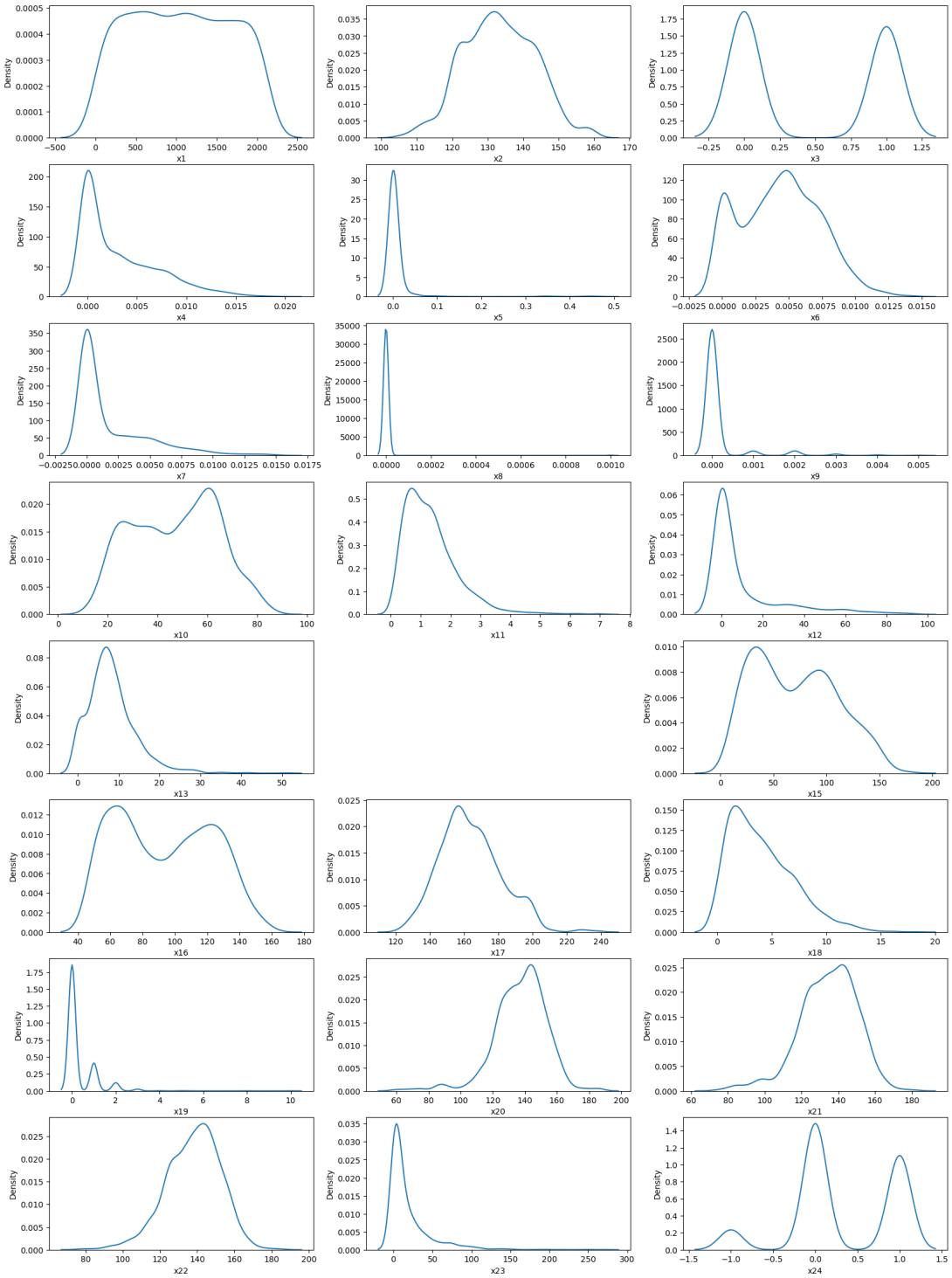
	x1	x2	x3	x4	x5	\
count	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000	
mean	1053.188131	133.297980	0.468434	0.003169	0.009631	
std	615.996716	10.002632	0.499160	0.003821	0.047974	
min	0.000000	106.000000	0.000000	0.000000	0.000000	
25%	523.750000	126.000000	0.000000	0.000000	0.000000	
50%	1049.500000	133.000000	0.000000	0.002000	0.000000	
75%	1583.250000	141.000000	1.000000	0.006000	0.002000	
max	2125.000000	160.000000	1.000000	0.019000	0.477000	

	x6	x7	x8	x9	x10	\
count	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000	
mean	0.004347	0.001854	0.000003	0.000157	47.126894	
std	0.002948	0.002940	0.000050	0.000593	17.123488	
min	0.000000	0.000000	0.000000	0.000000	13.000000	
25%	0.002000	0.000000	0.000000	0.000000	32.000000	
50%	0.004000	0.000000	0.000000	0.000000	49.000000	
75%	0.007000	0.003000	0.000000	0.000000	61.000000	
max	0.014000	0.015000	0.001000	0.005000	86.000000	

	x11	x12	x13	x15	x16	\
count	1584.000000	1584.000000	1584.000000	1584.000000	1584.000000	
mean	1.337816	10.008838	8.255240	70.409091	93.496843	
std	0.899092	18.520206	5.784579	38.993892	29.593370	
min	0.200000	0.000000	0.000000	3.000000	50.000000	
25%	0.700000	0.000000	4.600000	36.000000	67.000000	
50%	1.200000	0.000000	7.400000	68.000000	93.000000	
75%	1.700000	11.000000	10.700000	100.000000	120.000000	
max	7.000000	91.000000	50.700000	176.000000	158.000000	

```
                x17          x18          x19          x20          x21  \
count  1584.000000  1584.000000  1584.000000  1584.000000  1584.000000
mean    163.905934     4.063763     0.324495   137.333965   134.542929
std      17.908749     2.950268     0.718499    16.461643    15.729735
min     122.000000     0.000000     0.000000    60.000000    73.000000
25%     152.000000     2.000000     0.000000   128.000000   125.000000
50%     162.000000     3.000000     0.000000   139.000000   136.000000
75%     174.000000     6.000000     0.000000   148.000000   145.250000
max     238.000000    18.000000    10.000000   187.000000   182.000000

                x22          x23          x24
count  1584.000000  1584.000000  1584.000000
mean    137.935606    18.448232     0.309343
std      14.622680    28.375002     0.615868
min      77.000000     0.000000    -1.000000
25%     128.000000     2.000000     0.000000
50%     139.000000     7.000000     0.000000
75%     148.000000    23.000000     1.000000
max     186.000000   269.000000     1.000000
```

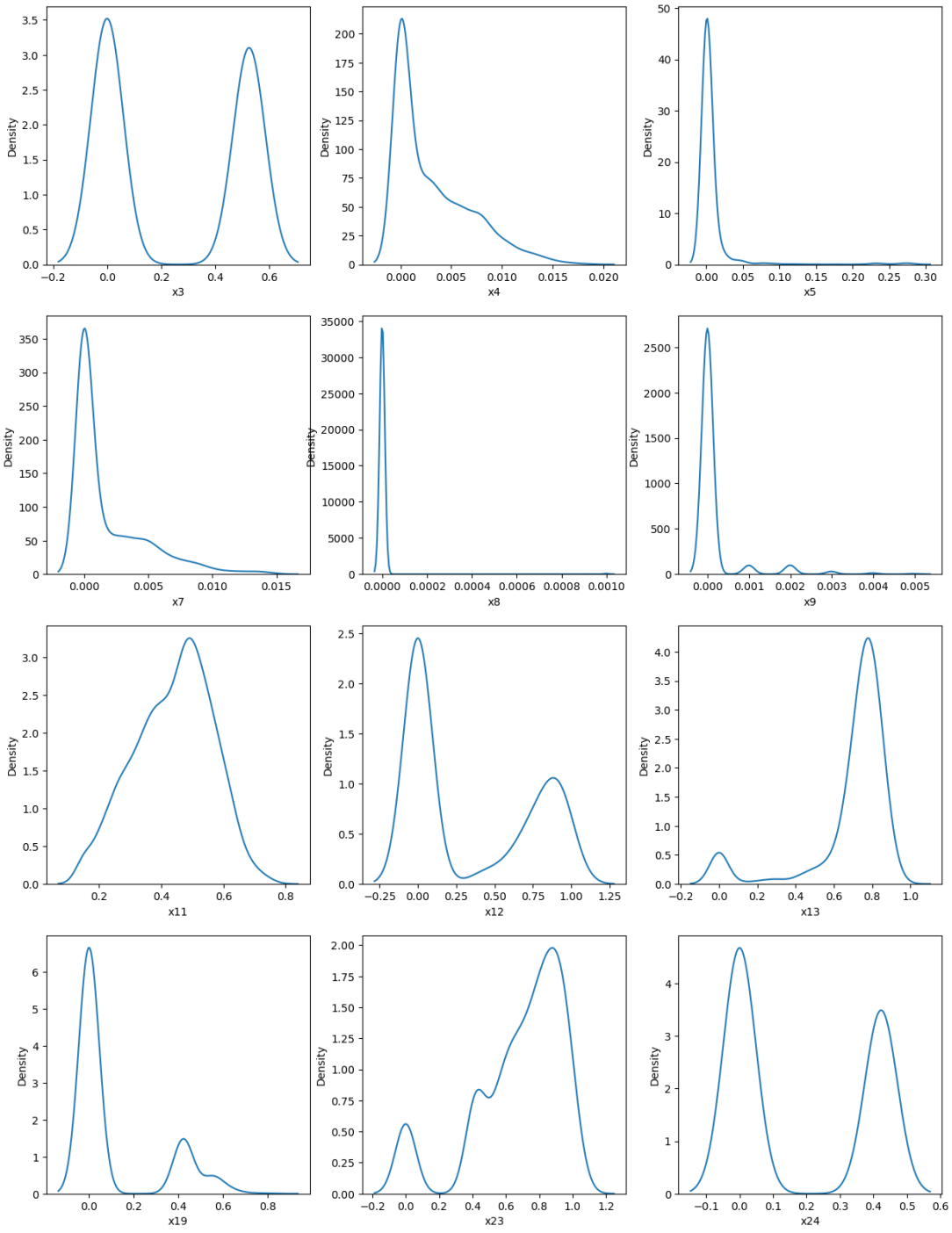
[156]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(20,35))
for i, col in enumerate(HealthTrain_data.columns):
    if HealthTrain_data[col].dtype != 'object':
        ax = plt.subplot(10, 3, i+1)
        sns.kdeplot(HealthTrain_data[col], ax=ax)
        plt.xlabel(col)

plt.show()
```



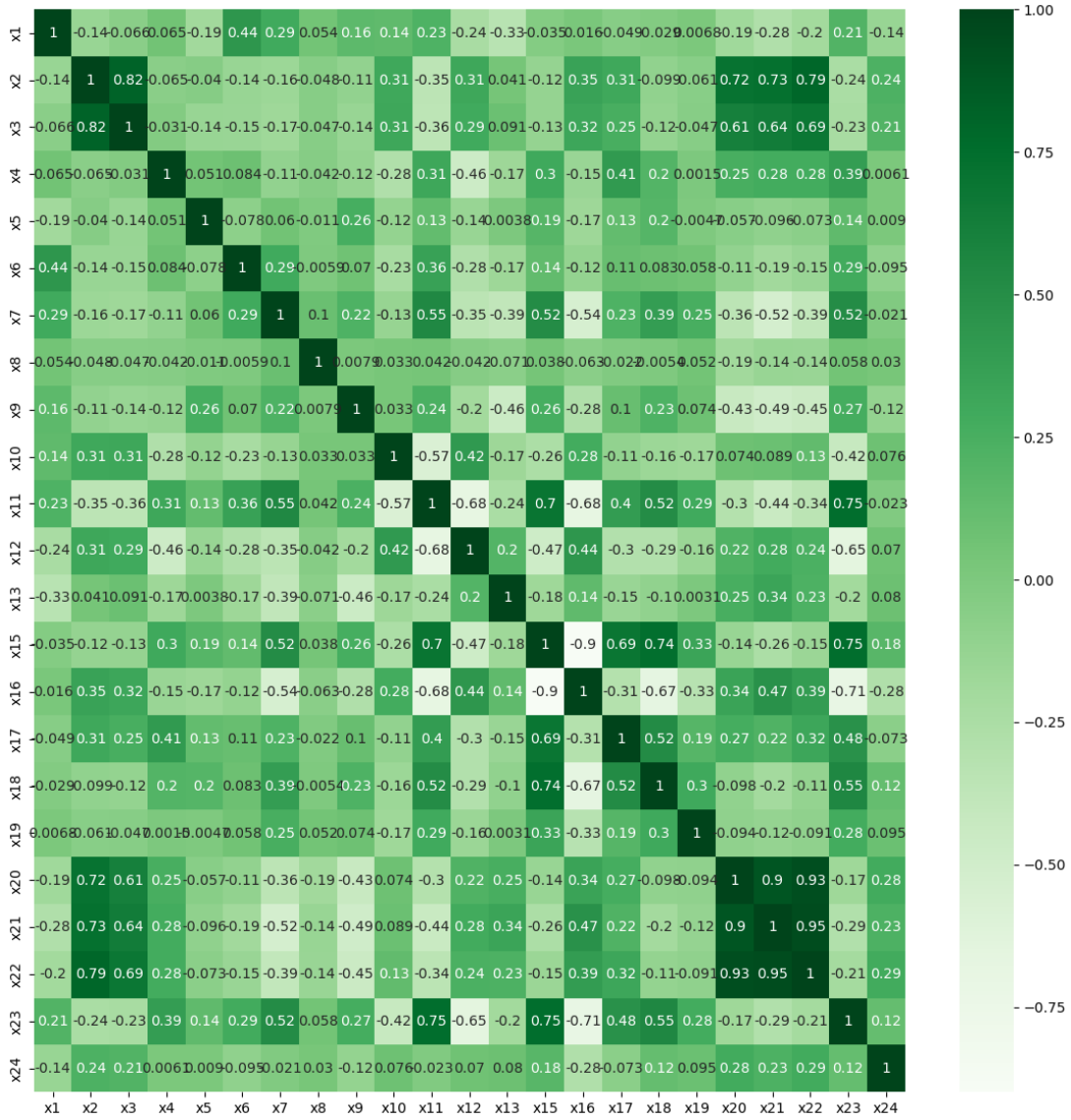
[173]:
```
cols = ['x3','x4', 'x5', 'x7', 'x8','x9','x11','x12','x13','x19','x23','x24']
```

[174]:
```
for col in cols:
    HealthTrain_data[col] = np.log1p(HealthTrain_data[col])
```

[175]:
```
plt.figure(figsize=(15,20))
for i, col in enumerate(cols):
    ax = plt.subplot(4, 3, i+1)
    sns.kdeplot(HealthTrain_data[col], ax=ax)
plt.show()
```



[176]:
```
plt.figure(figsize=(14,14))
sns.heatmap(HealthTrain_data.corr(), annot=True, cmap='Greens')
plt.show()
```



[195]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from scipy.sparse import hstack,csr_matrix
one_tencoder = OneHotEncoder()
test_X_encoded = one_hencoder.fit_transform(HealthTest_data)
if test_X_encoded.shape[1] < train_X_encoded.shape[1]:
    missing_columns = train_X_encoded.shape[1] - test_X_encoded.shape[1]
    test_X_encoded = hstack([test_X_encoded, csr_matrix((test_X_encoded.
 ↪shape[0], missing_columns))], format='csr')
#test_predictionsnb = naivebayes.predict(test_X_encoded.toarray())
#print(test_predictionsnb)
```

```
test_predictionssvc = Classifiersvc.predict(test_X_encoded)
print(test_predictionssvc)
```

```
[1 0 2 2 2 0 0 2 2 2 1 2 0 1 0 0 2 2 2 2 1 1 2 2 0 1 2 2 0 1 0 0 2 0 0 0 0
 0 2 1 2 1 0 0 1 2 2 1 0 1 0 2 2 1 2 0 2 0 2 2 0 2 2 2 2 2 2 1 2 2 2 1 2 0
 0 1 0 2 2 2 0 1 2 0 2 1 0 0 2 0 2 2 1 2 2 2 0 2 0 2 0 1 1 0 2 0 0 0 0 1 1
 1 1 2 0 0 0 2 2 2 0 0 1 2 2 0 0 0 1 1 2 1 1 2 1 0 2 0 2 0 2 0 2 0 2 0 1 1
 1 2 2 0 2 1 0 0 2 1 2 0 2 2 1 1 2 2 2 0 0 2 2 0 0 0 0 0 2 0 1 0 0 1 0 0 2
 1 1 0 1 2 0 2 1 1 0 1 1 2 1 1 1 2 0 0 0 0 1 2 1 2 0 0 0 1 2 1 0 2 2 1 0 0
 1 1 2 1 2 1 2 2 2 2 1 2 1 2 2 1 2 2 0 0 1 1 1 1 2 0 0 0 1 2 1 1 1 2 2 2 1
 1 1 2 2 1 2 2 1 1 2 0 1 1 2 1 2 2 1 0 1 1 0 2 0 2 1 1 2 1 1 0 2 2 1 0 0 1
 1 1 2 2 2 1 2 1 0 2 2 0 0 1 1 0 2 0 2 0 2 2 0 2 1 2 0 0 1 2 1 2 2 2 2 2 2
 0 1 0 1 0 2 0 1 1 2 1 0 0 2 1 0 0 0 2 0 2 2 1 0 2 0 0 0 0 2 1 1 0 1 0 2 1
 2 1 1 0 2 1 1 2 0 1 2 1 1 2 2 0 0 2 2 1 0 1 2 2 1 1 2 0 0 2 2 0 2 2 1 0 0
 2 2 0 0 2 0 1 2 1 1 1 0 2 0 2 0 2 2 0 2 1 0 2 2 2 2 1 1 0 2 2 2 1 0 0 1 0
 1 2 0 0 2 0 1 2 2 1 0 2 0 1 2 2 2 0 0 2 2 1 0 0 1 1 0 1 0 1 2 2 2 0 2 1 2
 1 2 1 0 2 0 2 0 1 0 2 1 0 2 0 1 2 1 0 2 0 2 1 2 0 1 2 2 2 2 2 0 0 0 1 2 2
 1 0 2 2 0 1 2 0 1 2]
```

```
[ ]:
```