## Etical Considaration

### Risks

First, we need to understand the risks of results mistakes or misinterpretations. Medical data has a high sensitivity for the risk. We can calculate the cost of mistakes for many industries that may emerge in the analysis. For example, a mistake in sales analysis may cause the loss of the customer and N-amount of potential profit. However, in medical analysis, especially directly related to patient health, we can not calculate the risk because human life and health are priceless. Considering that we need to prevent any harm which may be caused by analysis and conclusions. All statements and comments must be strictly defied. We need to prevent any misinterpretations of our research and insights. For example, two identical insights from the analysis can be displayed in different ways:

* **"Smoking does not cause Disease A"** This is a hazardous statement which may cause many problems for people who read the study and apply this statement somehow. Additionally, for the company which published the study.
* **"Based on a dataset provided and methods used there is not enough evidence that smoking has a direct impact on physical characteristics related to disease A"** This statement seems savier because we defined that our conclusion is based only on data provided. Also, we mentioned that it has no direct impact, but impact in general may be.

### Bias

Because we do not know how data was being collected, we can not say how accurately the given sample represents the population and if any bias emerged during the collection. This means that machine learning models may perform differently with other datasets.

There are several types of bias which may be in the dataset:

* **Selection Bias:** Because we do not know how data was being collected, we can not say how accurately the given sample represents the population and if any bias emerged during the collection. This means that machine learning models may perform differently with another dataset.
* **Survey Bias:** The dataset includes personal information about each patient. This information was likely provided directly by the patient, which means the information may be not as accurate as we could expect.
* **Measurement Bias:** Some features contain numerical data representing the patient's physical condition. We do not exactly know how those numbers were obtained and by whom.
* **Demographic Bias:** If the given dataset represents a specific demographic group, the models developed in the research may not give the same accuracy if they are applied to diagnose different demographic groups.

## Objective

The dataset offers to analyse many different hypotheses such as a correlation between any tendency of a patient's lifestyle and physical characteristics. For us, it is essential to understand whose problem we are going to solve. In our case, we need to keep in focus the hospital/clinic efficiency rather than directly intervene in patient tendencies and treatments. As objectives, we need to define which information will help medical personnel to do their duties and how. We can define two types of problems we are going to investigate: Rapidity and Quality.

### General Predictability Evaluation

In this section, we are going to apply different classification models to investigate which model performs with the best accuracy in terms of diagnosing. Additionally, we are going to select different features to train models to see which combination benefits accuracy the most.

The aim of the section is similar to a previous stage, specifically to help medical personnel decrease the number of possible diagnoses, to increase the probability of selecting the right one.

Important note: Results of machine learning models can be used only as a supporting or secondary method by medical personnel and should not be considered as a final decision about patient condition.

## Data Source and Methods

### Data Source

We have been provided with 2 files: csv. file with dataset and docx. dictionary file with a brief explanation for each column in the dataset. Mostly we are going to work with csv file. However second file may be helpful in the preparation stage.

### EDA

* Statistical summary
* Distributions
* Skewness
* Correlations
* Feature Importance
* Unique values
* Dimension reduction (PCA, LDA)

**Note:**

EDA and Data Preparation stages were merged. This approach is chosen for clarity purposes. It allows one to detect an issue, and select appropriate methods to solve it.

### Data Preparation

In this section we will try to investigate several methods for dealing with each issue we will find. Additionally, there will be created an Accuracy log, which will help us to track the best combination of Data Preparation methods.

Unfortunately, in the format of this research, we will not be able to test all combinations properly, due to the large number of those combinations. However, we will compare the approaches in each subsection, and keep the one which allows the models to perform better than others.

* Missing values
* Encoding
* Normalisation
* Feature Engineering
* Feature Selection
* Target Selection
* Splitting Data
* Scaling

### Machine Learning Models

In this section, we will apply several different ML Models and a DL Model to each variation of the dataset after the preparation section. We will use GridSearch construction to tune models and find more efficient hyperparameters. However, GridSearch may be quite a time-consuming method, despite the modest hyperparameters we set. We will apply Cross-Validation to pre-evaluate the models' performance.

* Logistic Regression
* Decision Tree
* Random Forest
* Gradient Boosting
* K-Nearest Neighbors
* Multi-layer Perceptron
* SVC
* XGBoost
* PyTorch ANN

**Note:**

The metrics and scores will be recorded as DataFrame and variables

### Evaluation methods

Because we are going to test several models, we need to evaluate them properly to understand which strong and weak sides each model has.

Evaluation metrics:

* Precision
* Recall
* F1 score
* Train/Test Accuracy
* RMSE
* MAE
* ROC-AUC

# EDA

## Summary

df = pd.read\_csv("healthcare\_dataset.csv")

df.dtypes

Age int64  
BMI float64  
BloodPressure float64  
Cholesterol float64  
Glucose int64  
Insulin int64  
HeartRate int64  
Smoker int64  
PhysicalActivity object  
SleepTime int64  
MedicalHistoryScore int64  
IncomeLevel object  
StressLevel object  
HealthIns object  
Disease object  
dtype: object

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Age | BMI | BloodPressure | Cholesterol | Glucose | Insulin | HeartRate | Smoker | SleepTime | MedicalHistoryScore |
| count | 5000.000000 | 4750.000000 | 4750.000000 | 4750.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 |
| mean | 48.805600 | 27.332873 | 129.749053 | 199.611158 | 137.440600 | 146.573200 | 80.342000 | 0.510400 | 7.482200 | 4.485400 |
| std | 17.906991 | 7.190172 | 28.671508 | 58.173633 | 47.448072 | 82.623127 | 13.914919 | 0.499942 | 2.281562 | 2.852405 |
| min | 18.000000 | 15.000291 | 80.000000 | 100.000000 | 70.000000 | 15.000000 | 60.000000 | 0.000000 | 4.000000 | 0.000000 |
| 25% | 34.000000 | 21.049709 | 105.000000 | 149.000000 | 103.000000 | 78.000000 | 70.000000 | 0.000000 | 5.000000 | 2.000000 |
| 50% | 49.000000 | 27.218082 | 130.000000 | 201.000000 | 136.000000 | 144.000000 | 80.000000 | 1.000000 | 7.000000 | 4.000000 |
| 75% | 64.000000 | 33.530189 | 155.000000 | 250.000000 | 167.000000 | 213.000000 | 89.000000 | 1.000000 | 9.000000 | 7.000000 |
| max | 79.000000 | 39.988943 | 179.000000 | 299.000000 | 582.000000 | 744.000000 | 198.000000 | 1.000000 | 11.000000 | 9.000000 |

# Data Preparation

## Missing values - EDA

##### Missingness mechanisms

Firstly, we need to understand the nature of missing values in our dataset. Bhaskaran, K. (2014) highlighted 3 types of missingness, which we need to consider before dealing with missing values:

* **Missing Completely at Random:**

Those missing values do not follow any pattern related to the patient's diagnosis. For example, medical personnel just didn't update the database with measurements or this data got lost after merging two datasets with different scales or abbreviations.

* **Missing at Random:**

When missing values are related to other features they can be explained through them. For example, medical personnel may not prescribe to a patient a specific blood test based on the features they already have. In this case, a patient may have missed a Cholesterol test because medical personnel made a decision based on information from other features.

* **Missing Not at Random:**

When data missing not only at random but also without data-driven decisions. For example, a patient ignored the doctor's prescription to take a blood test.

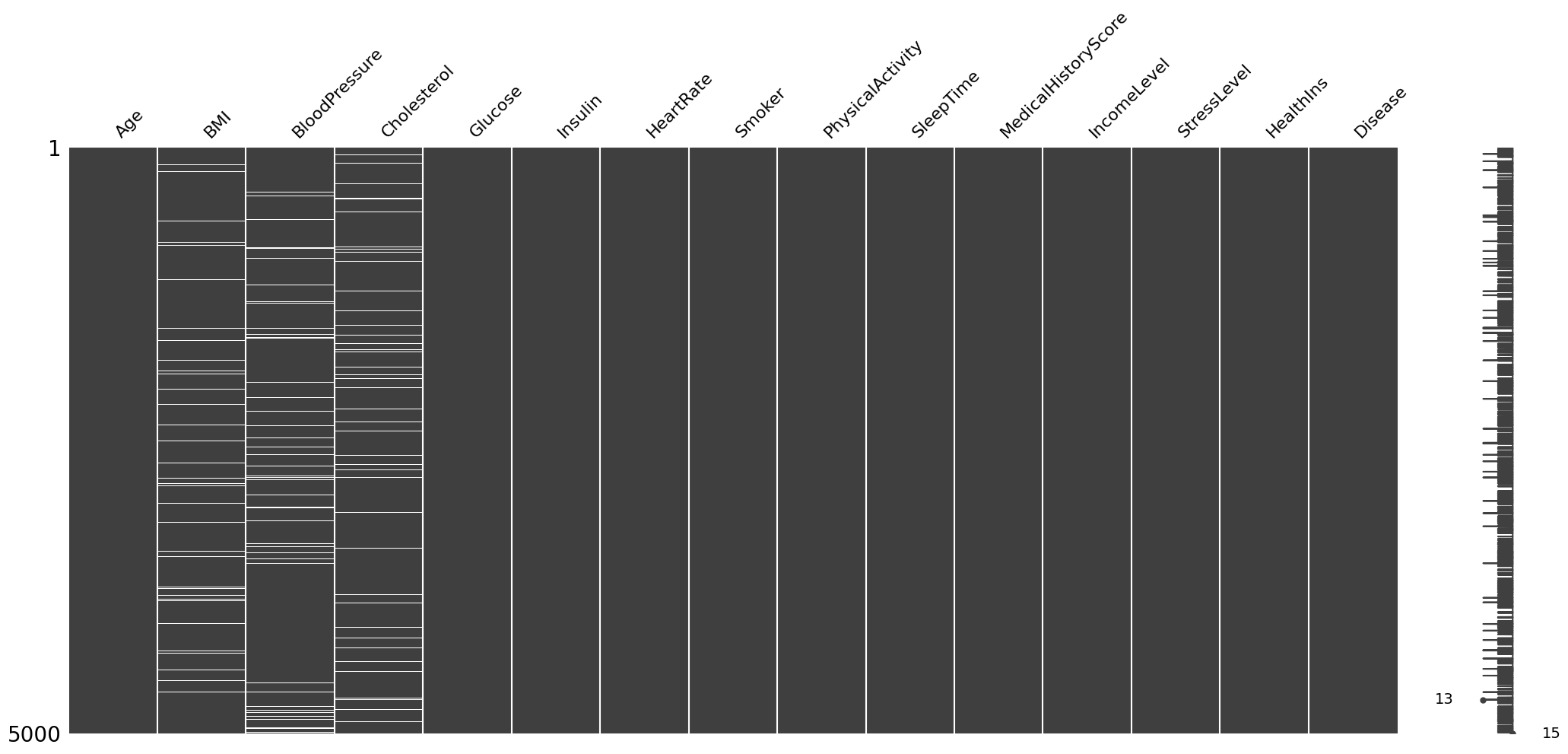
##### Display columns with missing values

As we can see we have 250 missing values in 3 features. Dropping those values may cost us 5-15% from the whole database considering that we have only 5k records. We will try different approaches to cope with this problem. However, we will apply the dropping method as well to compare the methods' efficiency.

Also, we can see that the 4th category in the column 'Disease' was recognised as missing values. There may be 2 reasons why missing values emerged there. The first reason is that patients who have missing values got one of those diseases but data was not recorded. The second reason is patients' diseases were not found. So, before working with missing values in independent columns we need to define all missing values in a "Disease" column.

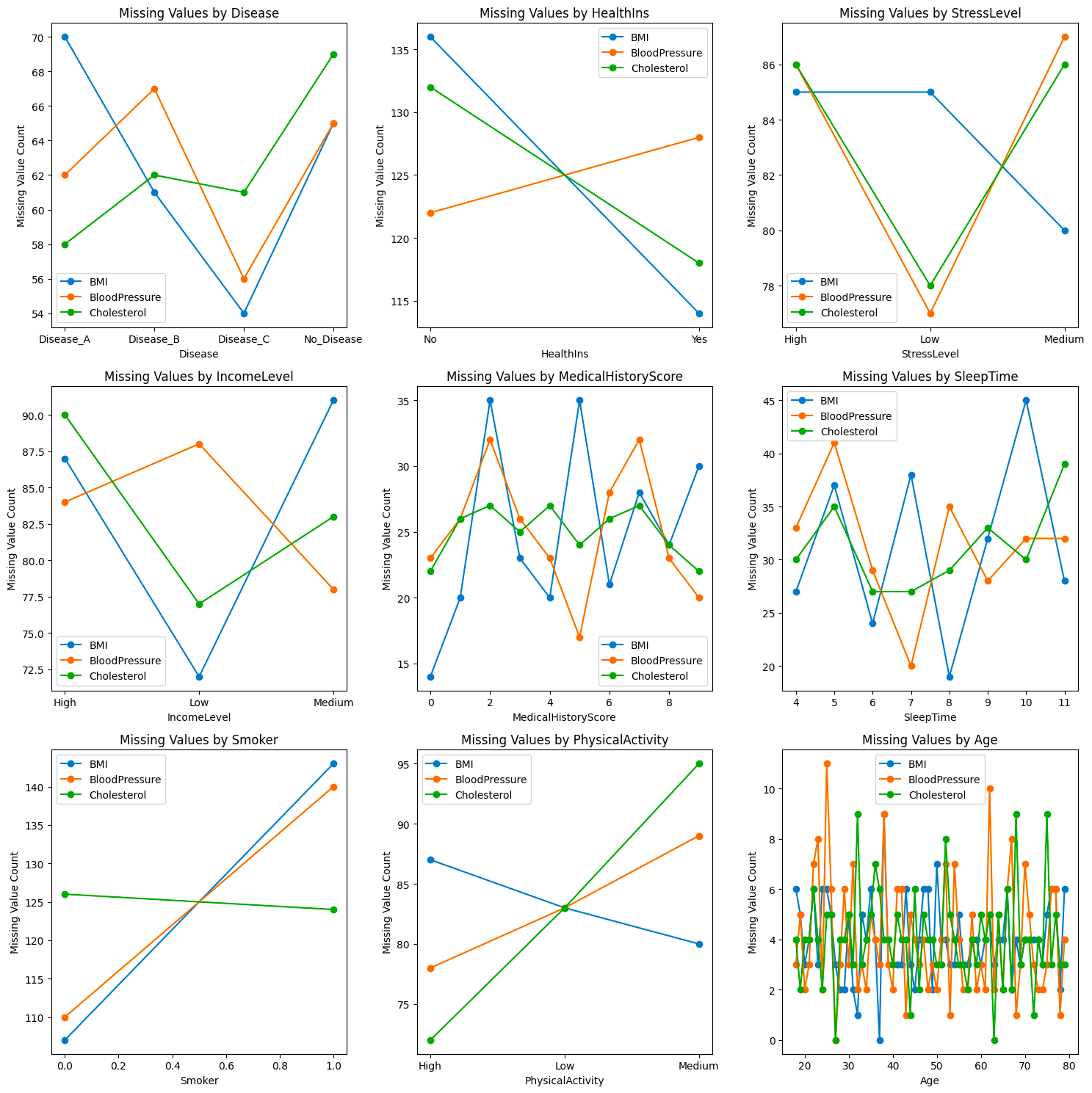
##### Missingness spread

**"missingno"** library will help us to assess amount of missing data and understand how missing values are spread



df1 = df.copy()  
df1['Disease'] = df1['Disease'].fillna('No\_Disease')  
msno.matrix(df1)

##### Correlation between Missingness and other features

To understand if there is any pattern indicating a occur of missing we plot missing values to each categorical feature

## Missing data - Handling

In terms of dealing with missing values, we will try several approaches to data imputation which were explained in detail by Palanivinayagam A. (2023) who applied Machine learning models to impute missing values. Also, authors Makaba T. and Dogo E. (2019) highlighted the pros and cons of many common methods of data imputation.

#### Fill with ML model

For this approach was chosen KNN Regressor because all our missing values are numeric. Additionally, as a temporary imputation method, median imputation. Ideally, we need to try different components in this function such as other ML models, and different imputation methods. However, it could increase the number of variations we need to test significantly.

If we find out that this method benefits the most to final accuracy using the accuracy log, we can experiment with different components.

knn\_regressor = KNeighborsRegressor()  
imputer = SimpleImputer(strategy='median')  
label\_encoder = LabelEncoder()  
cols\_to\_fill = ['BMI', 'BloodPressure', 'Cholesterol']  
**for** column **in** cols\_to\_fill:  
 **if** df\_missing[column].isnull().any():   
 train = df\_missing[~df\_missing[column].isnull()]  
 test = df\_missing[df\_missing[column].isnull()]   
 X2\_train = train.drop(columns=[column])  
 X2\_test = test.drop(columns=[column])   
 categorical\_cols = X2\_train.select\_dtypes(include=['object']).columns   
 X\_train = X2\_train.copy()   
 **for** column1 **in** categorical\_cols:  
 X\_train[column1] = label\_encoder.fit\_transform(X2\_train[column1])   
 categorical\_cols = X2\_test.select\_dtypes(include=['object'])  
 X\_test = X2\_test.copy()   
 **for** column2 **in** categorical\_cols:  
 X\_test[column2] = label\_encoder.fit\_transform(X2\_test[column2])   
 y\_train = train[column]  
 imputer = SimpleImputer(strategy='median')  
 X\_train = pd.DataFrame(imputer.fit\_transform(X\_train), columns=X\_train.columns)  
 X\_test = pd.DataFrame(imputer.fit\_transform(X\_test), columns=X\_test.columns)   
 knn\_regressor.fit(X\_train, y\_train)  
 missing\_values\_pred = knn\_regressor.predict(X\_test)  
 test[column] = missing\_values\_pred  
 df\_missing = pd.concat([train, test], ignore\_index=True)

#### Fill with the median

cols\_to\_fill = ['BMI', 'BloodPressure', 'Cholesterol']  
**for** col **in** cols\_to\_fill:  
 df\_missing[col] = df\_missing[col].fillna(df\_missing[col].median())

#### Drop rows with missing values

df\_missing = df\_missing.dropna()

In terms of accuracy, I think this method benefits the most. Previous methods create artificial data to fill in missing, and that data very likely is not as accurate as the original data. However, the dropping method has 2 serious disadvantages:

1. We lose data, in this case, we can lose 5-15%, depending on how missing values overlap.
2. Bais. We just reject consideration of patients' cases who don't have that information.

## Encoding

At this stage, we will encode categorical features using two common approaches a Standard encoder and a One-Hot encoder. Each method has its impact on machine learning model efficiency. The application of the standard encoding method was mentioned by Gillam R. (2003) where the author summarised all essential information regarding the encoding process.

Another study conducted by Choong A. and Lee N. (2017) compares the impacts of ordinal and one-hot encoding methods on model performance.

### Standad Encoder

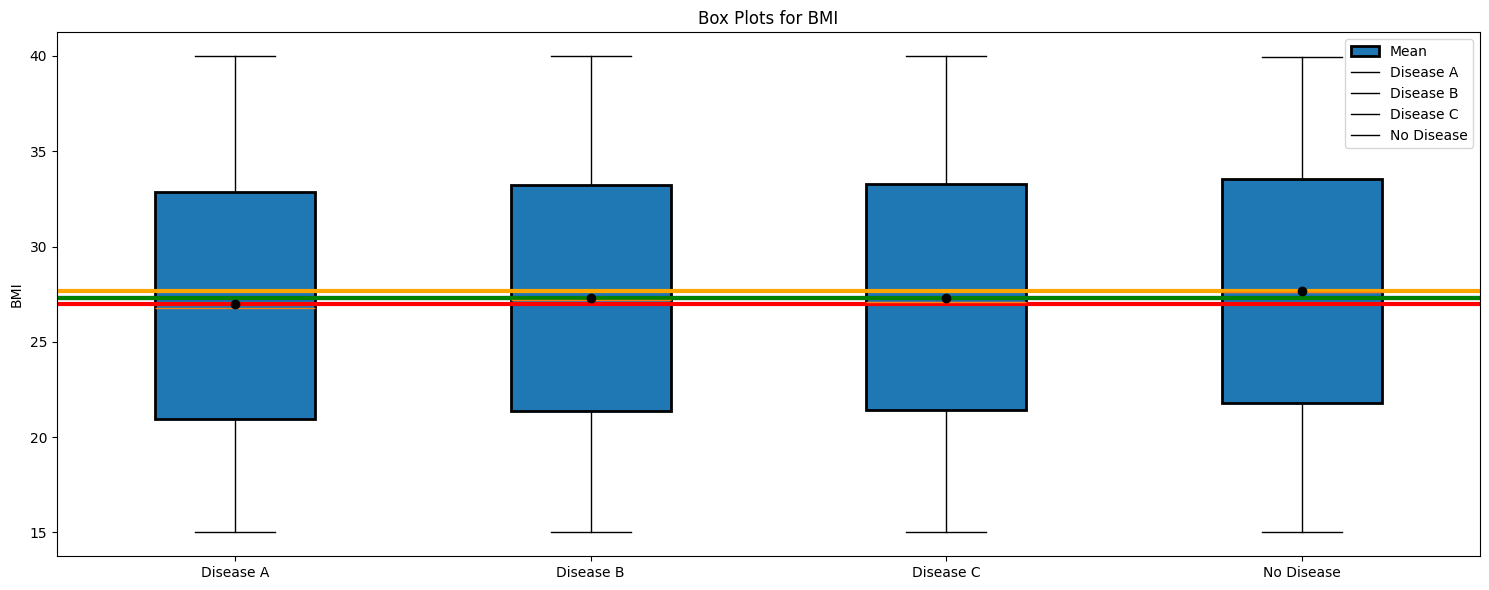
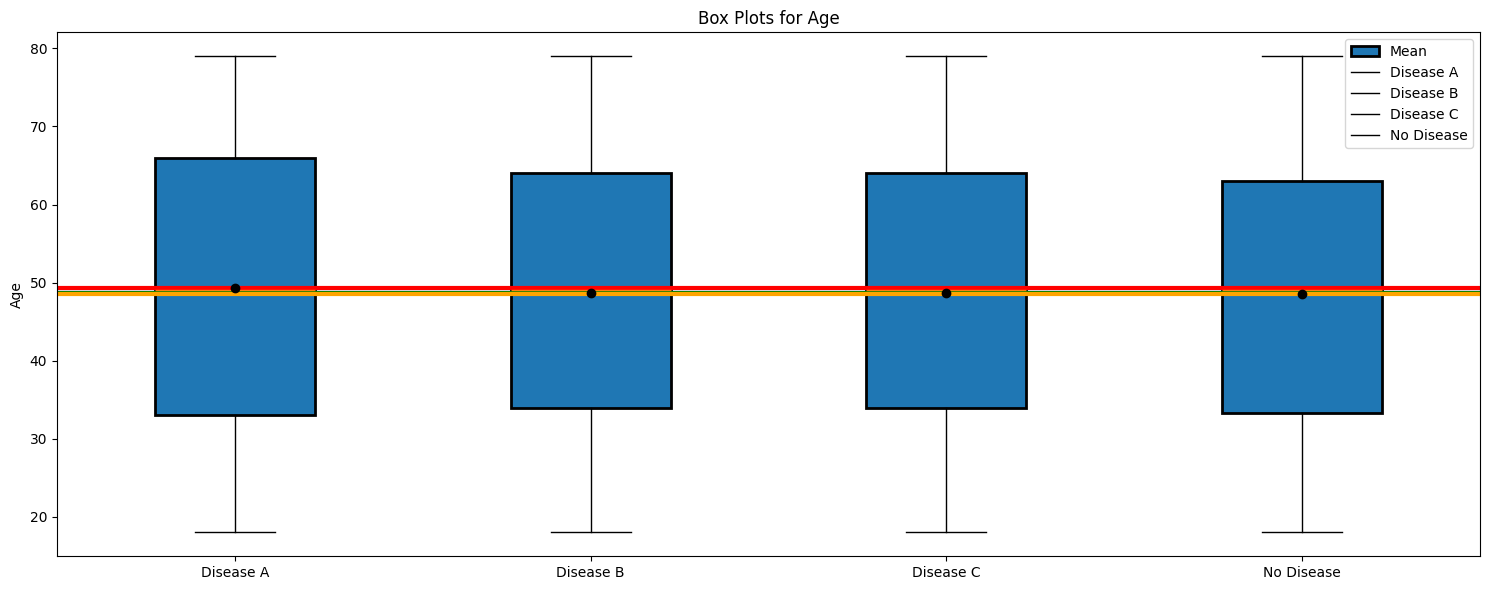
cat\_columns = df\_missing.select\_dtypes(include=['object'])  
df\_encoded = df\_missing.copy()  
label\_encoder = LabelEncoder()  
**for** column **in** cat\_columns:  
 **if** column != 'Disease':  
 df\_encoded[column] = label\_encoder.fit\_transform(df\_missing[column])

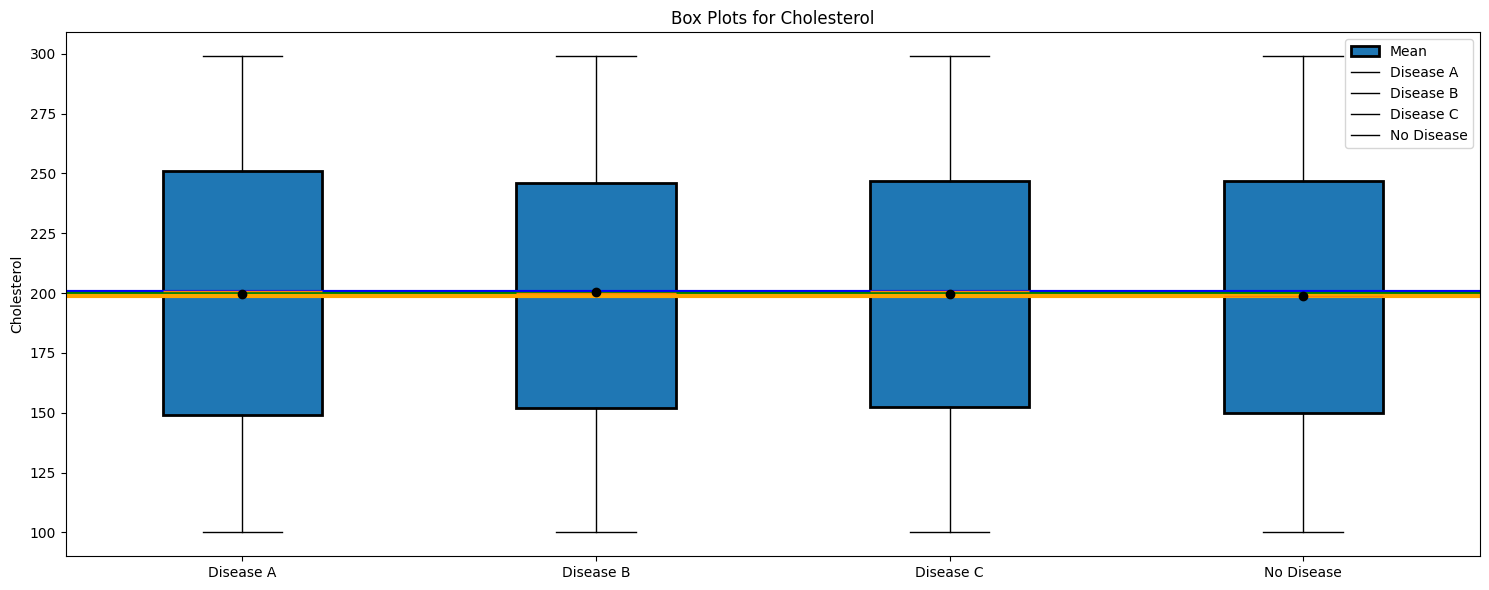
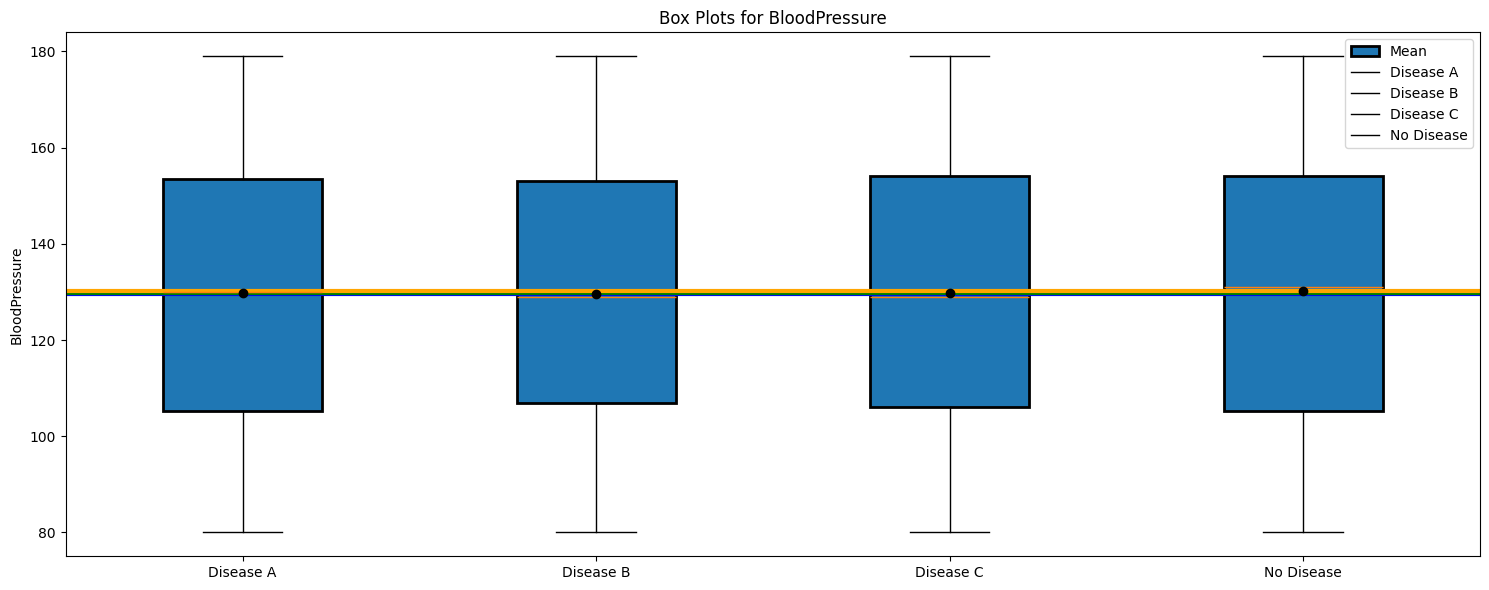
### One Hot Encoder

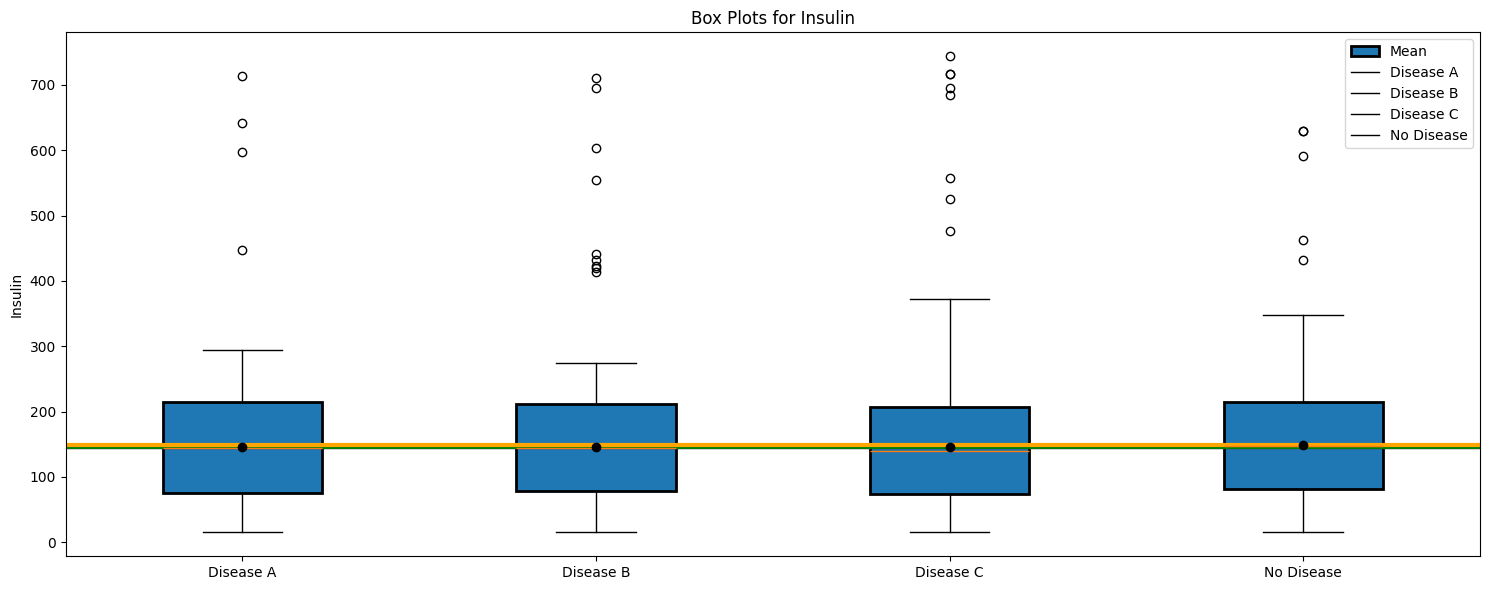
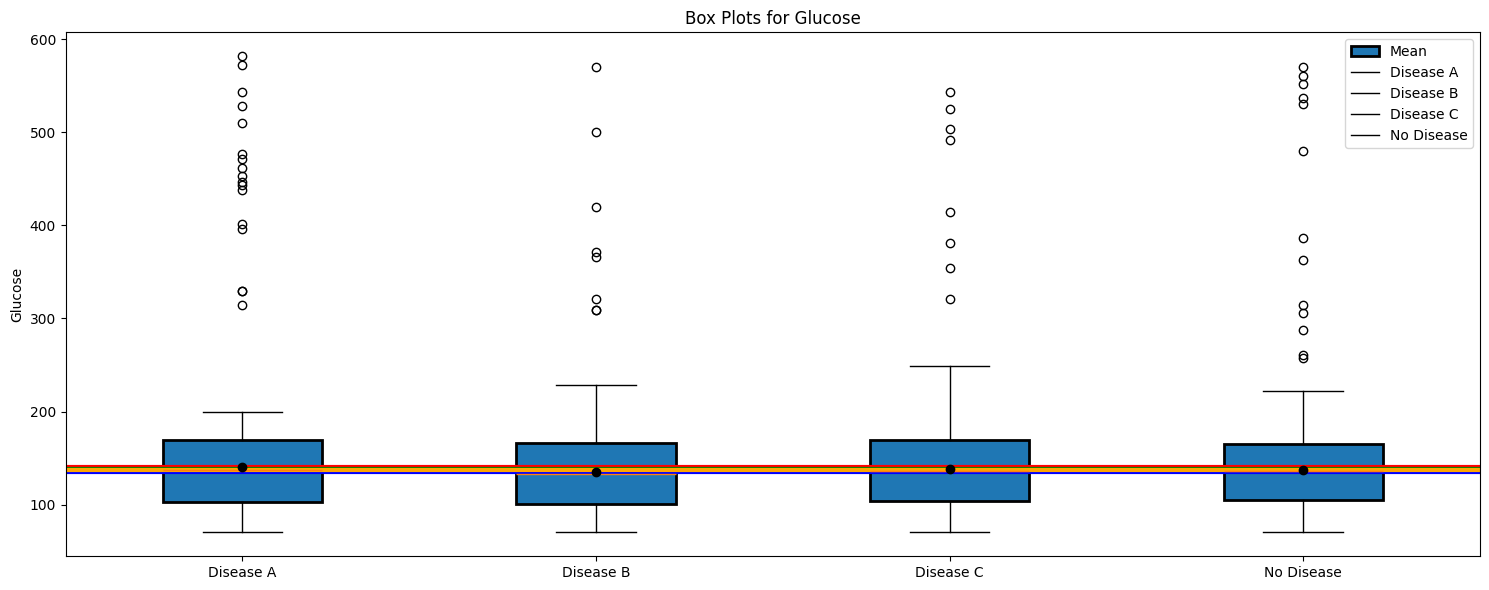
cat\_columns = df\_missing.select\_dtypes(include=['object'])  
df\_encoded = df\_missing.copy()  
**for** column **in** cat\_columns:  
 **if** column != 'Disease':  
 one\_hot = pd.get\_dummies(df\_missing[column], prefix=column)  
 df\_encoded = pd.concat([df\_encoded, one\_hot], axis=1)  
 df\_encoded.drop(column, axis=1, inplace=True)

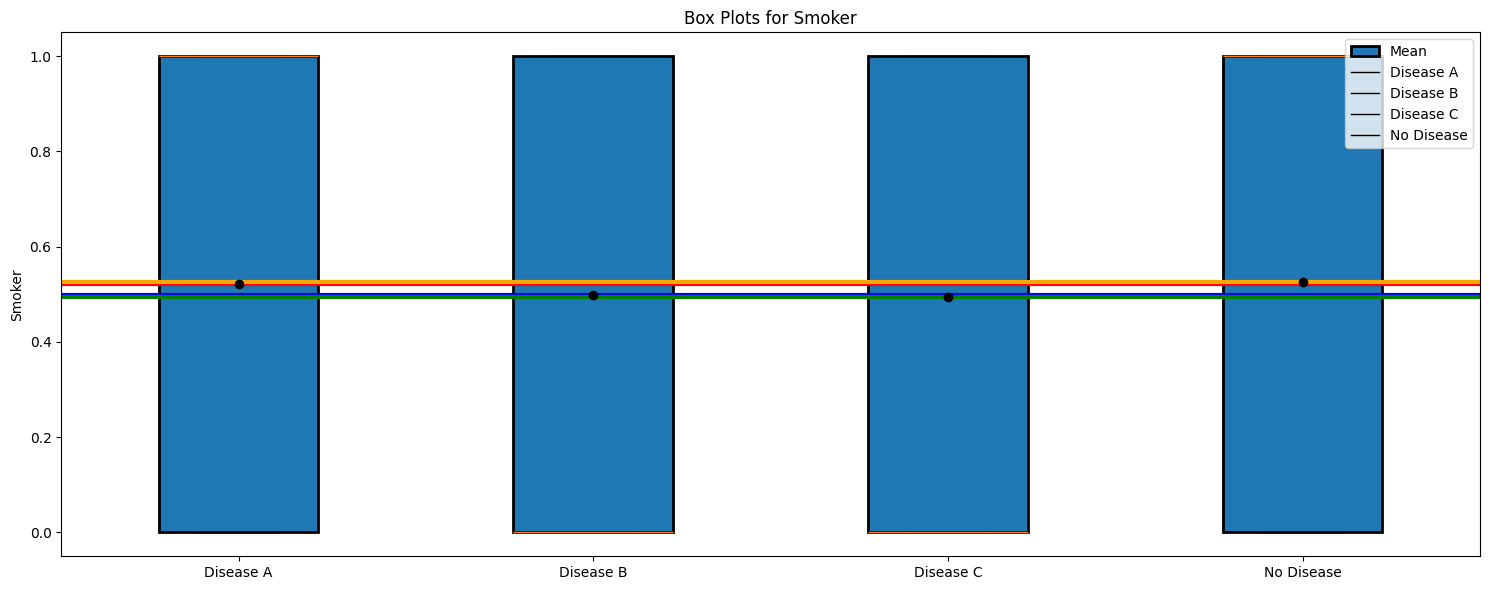
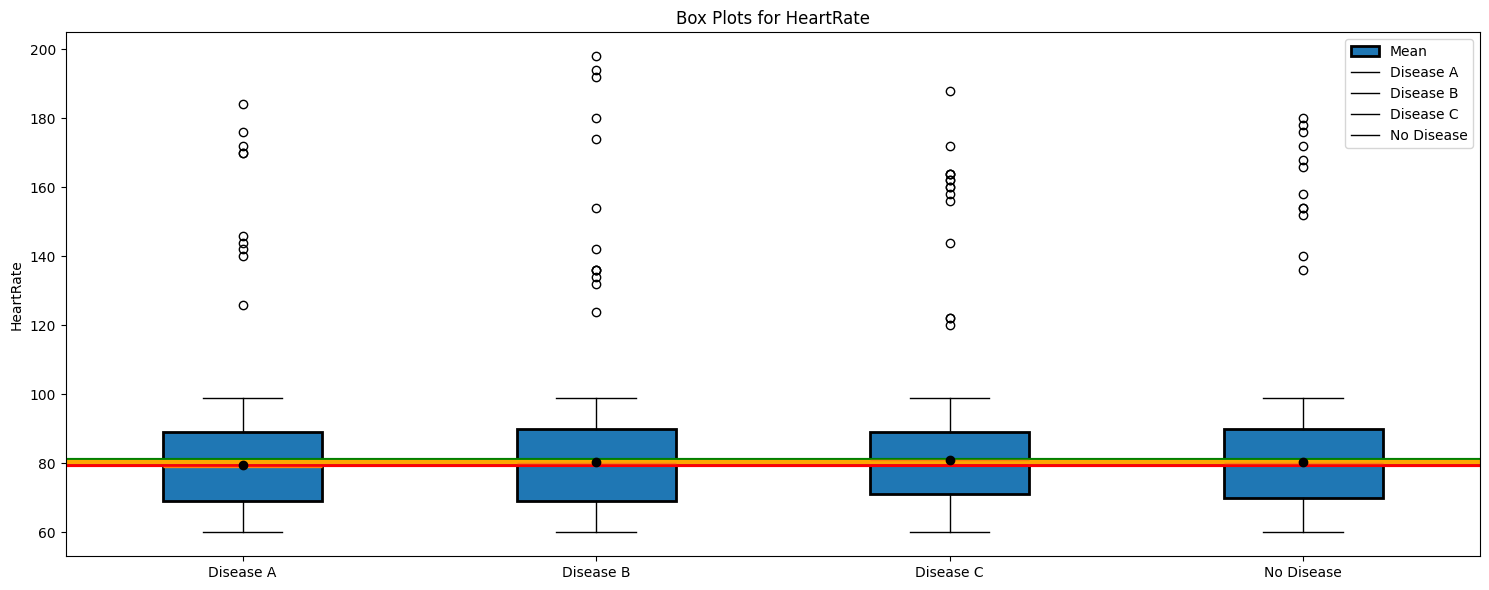
## Distributions

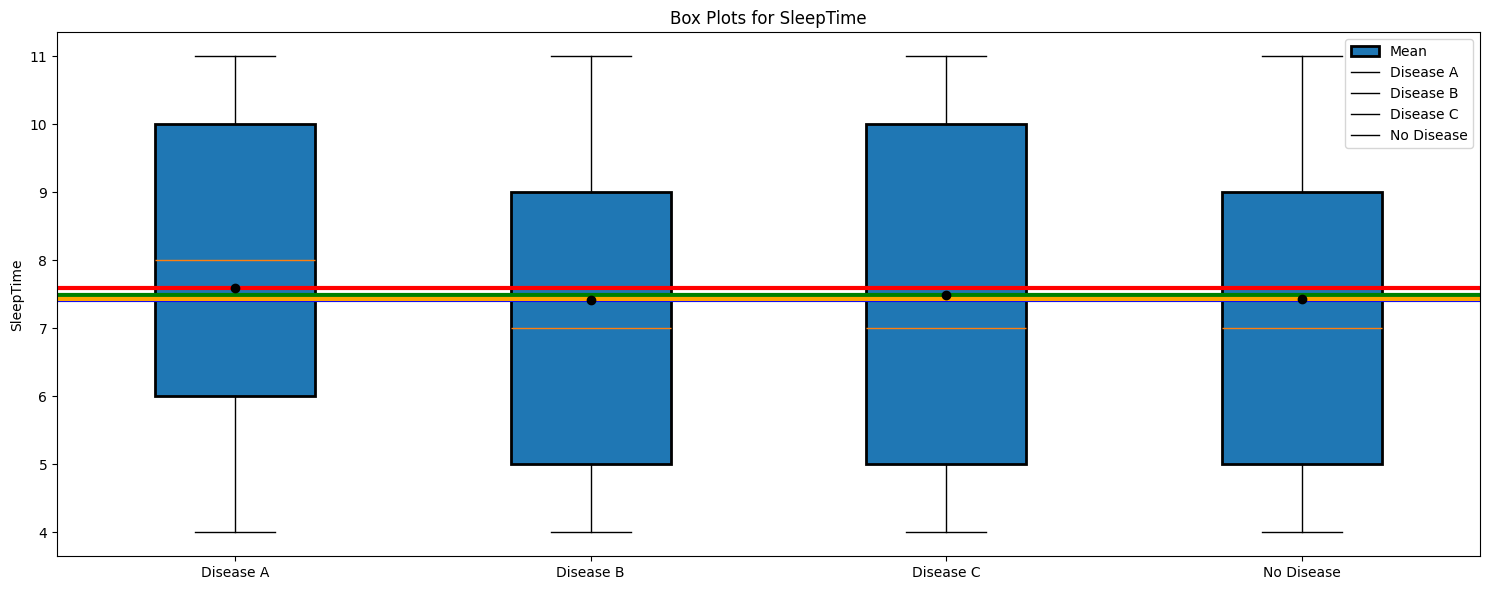
We are going to use box plots to visualise statistical data and some basic insights about our dataset. However, we will split data according to disease features to find out the differences.

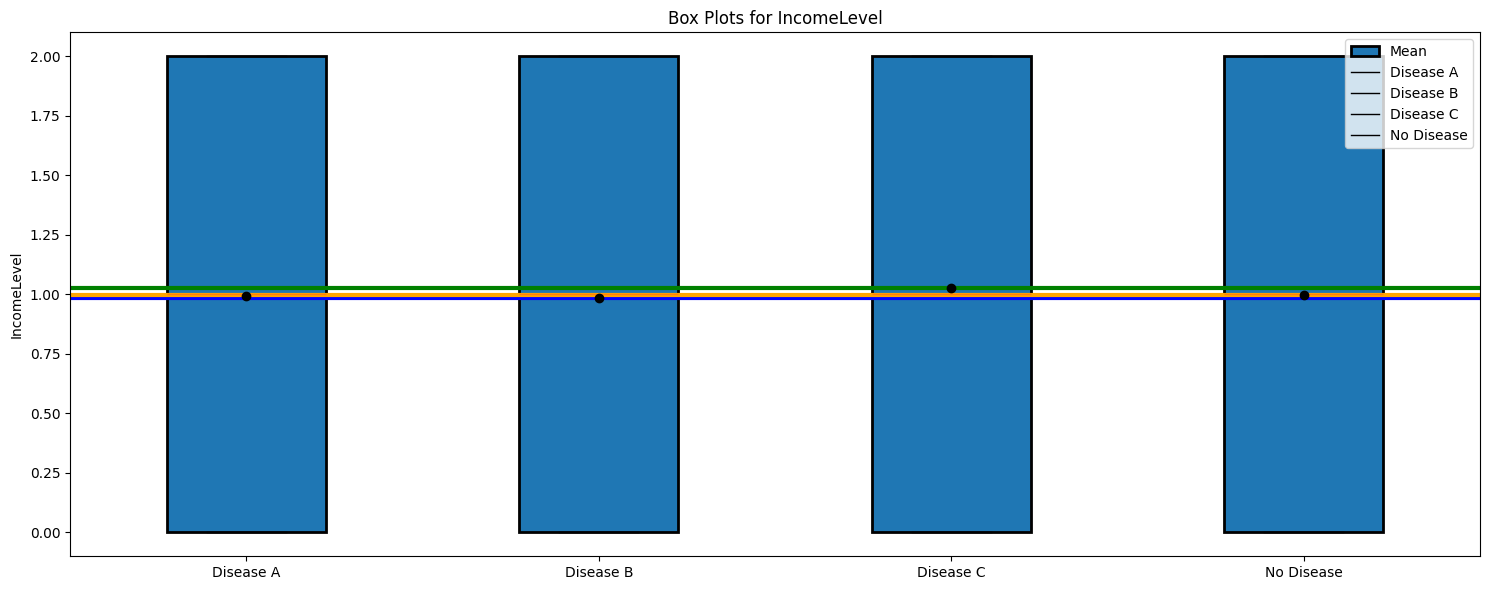
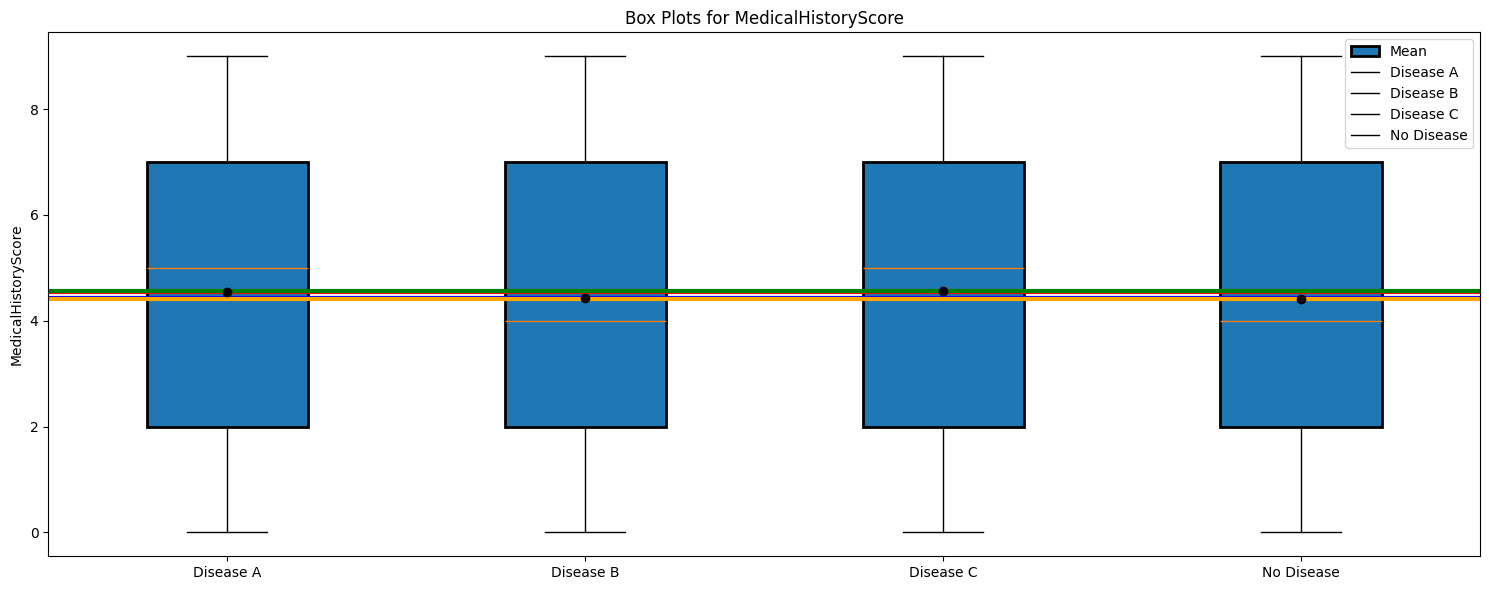


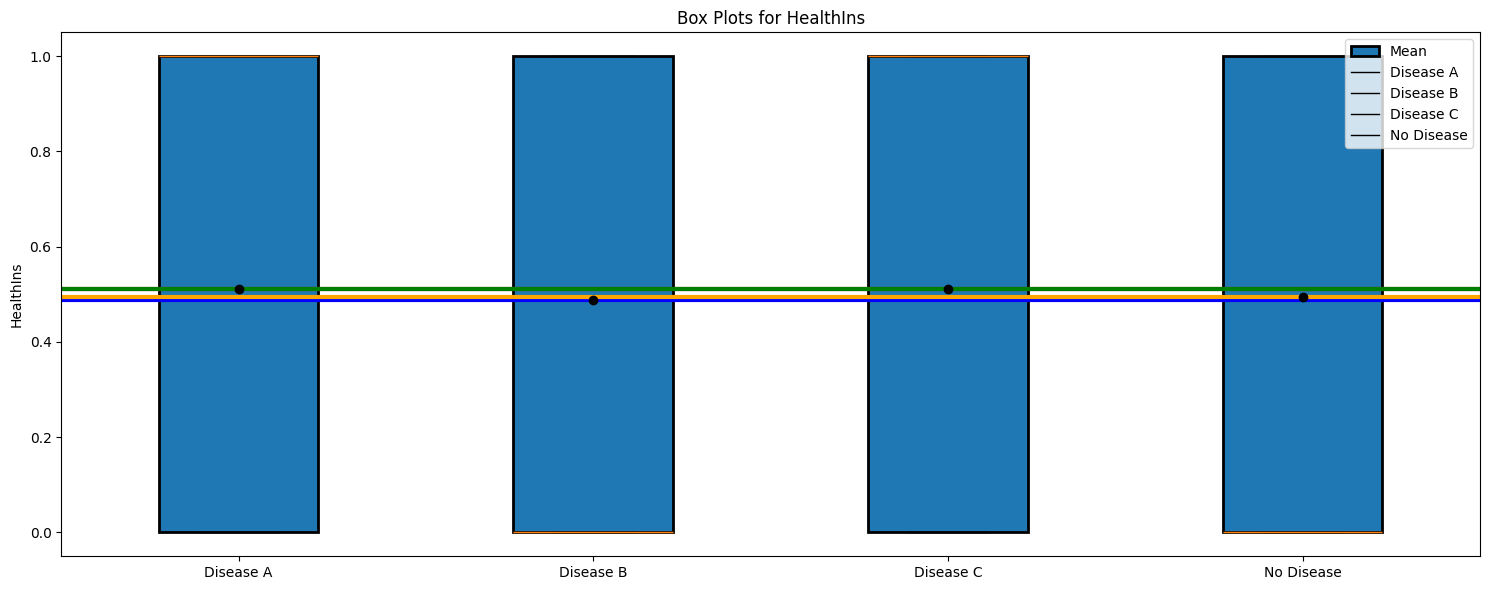












## Unique values

At this stage, we will take a look if there are any misspellings in the records or other issues which may occur in the categorical data.

**for** col **in** df.columns:  
 **if** df[col].dtype == 'object':   
 print(f"Unique values in {col}: {df[col].unique()}")

Unique values in PhysicalActivity: ['Medium' 'High' 'Low']  
Unique values in IncomeLevel: ['High' 'Medium' 'Low']  
Unique values in StressLevel: ['Low' 'Medium' 'High']  
Unique values in HealthIns: ['No' 'Yes']  
Unique values in Disease: ['Disease\_B' 'Disease\_A' 'Disease\_C' nan]

**Comment:**

Here we have unique values for some categorical features. However, although some features are in numeric format they still indicate categorical data. So, we will manually select features which indicate categorical features.

cat\_cols = ['PhysicalActivity','IncomeLevel' ,’StressLevel’,'HealthIns','Smoker','SleepTime','MedicalHistoryScore']

**for** col **in** cat\_cols:   
 print(f"Unique values in {col}: {df[col].unique()}”){df[col].unique()}")

Unique values in PhysicalActivity: ['Medium' 'High' 'Low']  
Unique values in IncomeLevel: ['High' 'Medium' 'Low']  
Unique values in StressLevel: ['Low' 'Medium' 'High']  
Unique values in HealthIns: ['No' 'Yes']  
Unique values in Smoker: [0 1]  
Unique values in SleepTime: [ 9 5 7 11 4 10 8 6]  
Unique values in MedicalHistoryScore: [5 3 0 1 7 4 8 6 9 2]

**Comment:**

Here we displayed all unique values of features which belong to a patient's lifestyle rather than data which displays medical data.

**Note:**

We can calculate all possible combinations which classify patients. In this particular case, we have 8640 possible patient types based on the social information provided. This information gives us some insights into further analysis.

## Skewed features - EDA

Those features were chosen due to the previous box plot displaying that several features have high skewness. One of the approaches we would like to apply is log transformation.

Several authors highlighted the impacts of using skewed data and some solutions to deal with it. Kazerouni A. et al. (2020) and Monard M. (2002), discuss how skewed data may affect classification tasks.

Authors Ha T. et al. (2021), highlighted the advantages of using log transform to deal with skewed data.

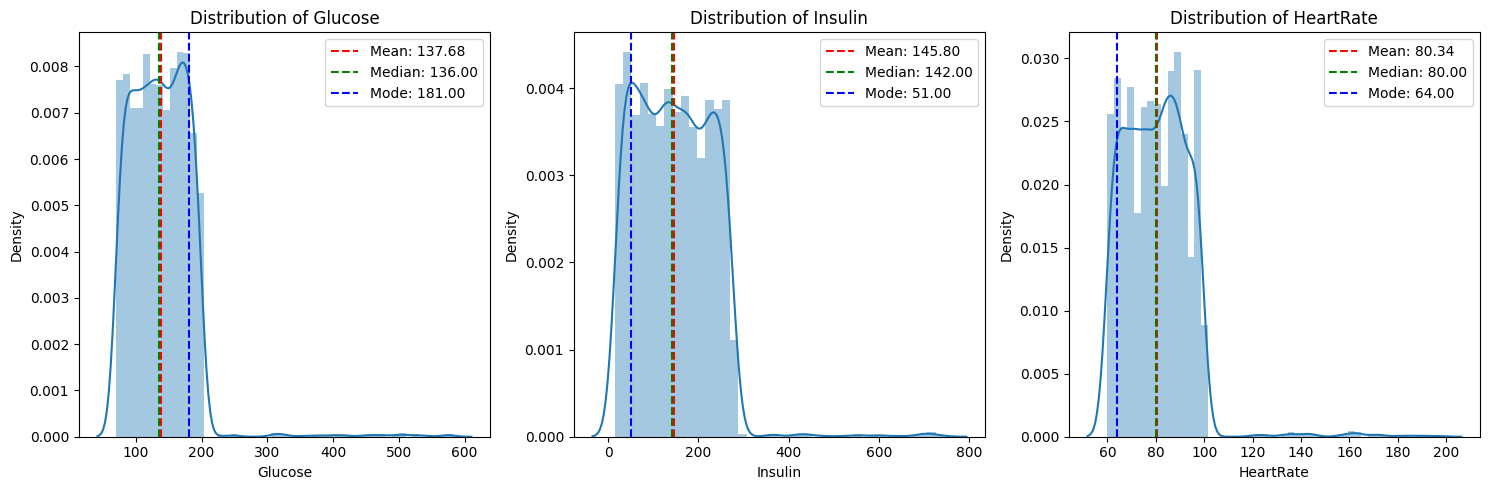
Briefly, the logarithm can be defined as a power which we need to transform log base to a number which we need to get the logarithm from.

* Skewness is less than -1 or greater than 1, the distribution is highly skewed.
* Skewness is -1 and -0.5 or between 0.5 and 1, the distribution is moderately skewed.
* Skewness is between -0.5 and 0.5, the distribution is approximately symmetric.

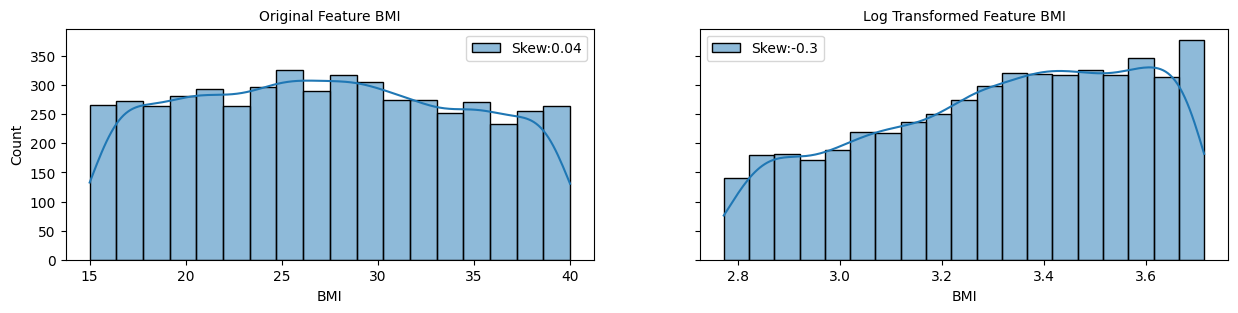
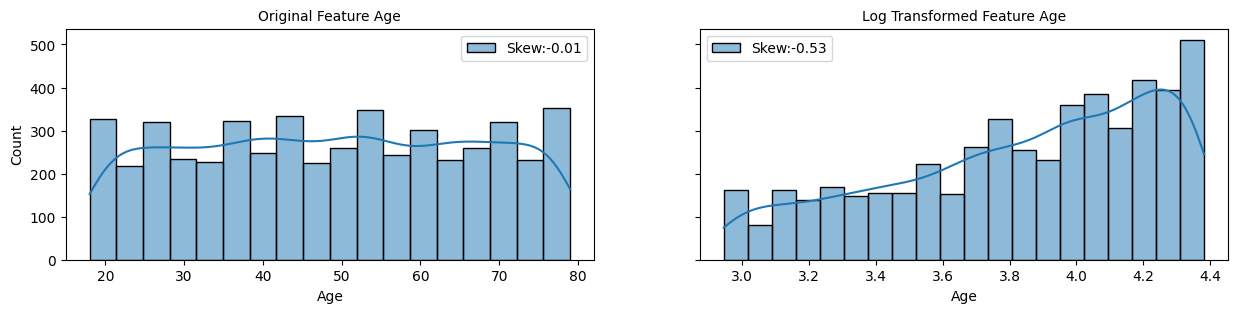
Skewed data may affect a machine learning model's performance which expects symmetric data.

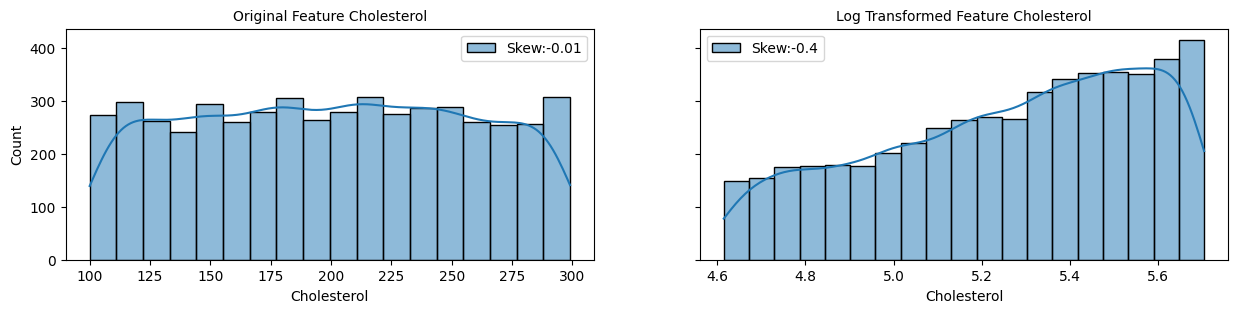
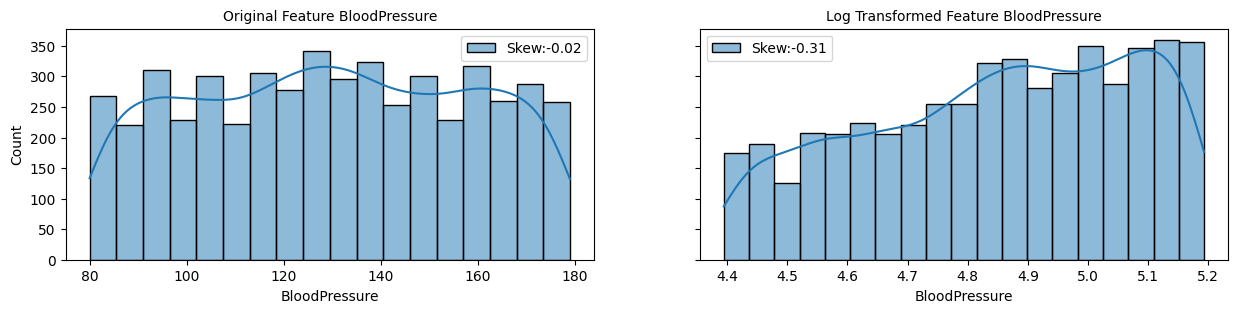
df\_skewed = df\_encoded.copy()

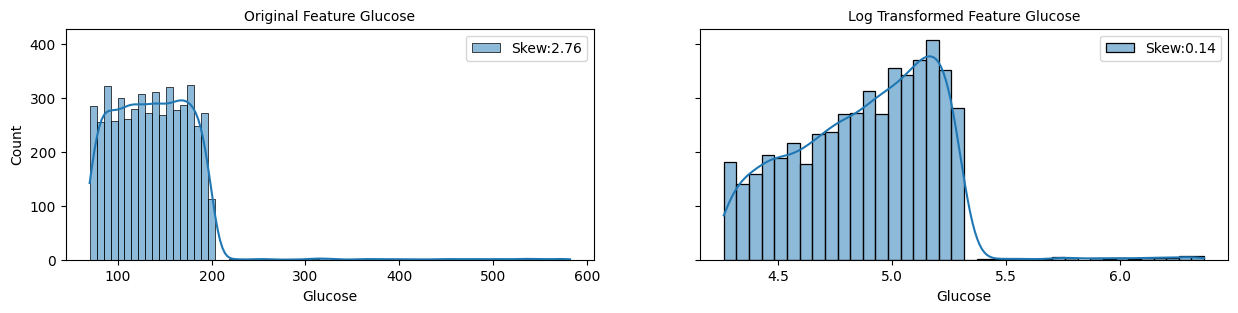
fig, axes = plt.subplots(1, 3, figsize=(15, 5))  
sns.distplot(df\_skewed['Glucose'], ax=axes[0])  
sns.distplot(df\_skewed['Insulin'], ax=axes[1])  
sns.distplot(df\_skewed['HeartRate'], ax=axes[2])  
**for** i, feature **in** enumerate(['Glucose', 'Insulin', 'HeartRate']):  
 mean\_age = df\_skewed[feature].mean()  
 median\_age = df\_skewed[feature].median()  
 mode\_age = df\_skewed[feature].mode()[0]   
 axes[i].axvline(mean\_age, color='r', linestyle='--', label=f'Mean: {mean\_age:.2f}')  
 axes[i].axvline(median\_age, color='g', linestyle='--', label=f'Median: {median\_age:.2f}')  
 axes[i].axvline(mode\_age, color='b', linestyle='--', label=f'Mode: {mode\_age:.2f}')   
 axes[i].set\_title(f'Distribution of {feature}')  
 axes[i].legend()  
plt.tight\_layout()  
plt.show()

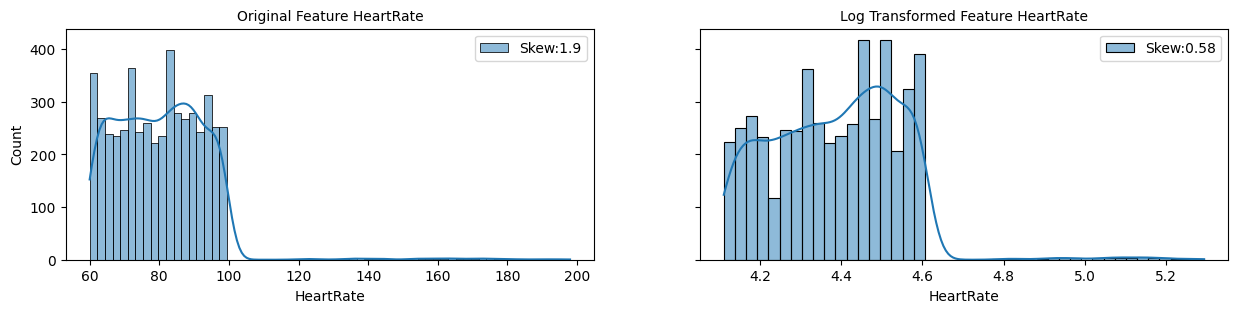


cols\_list = ['Age', 'BMI', 'BloodPressure', 'Cholesterol', 'Glucose', 'Insulin', 'HeartRate']  
**for** col **in** cols\_list:  
 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 3), sharey=True)  
 sns.histplot(ax=ax1, x=df\_skewed[col], kde=True, palette='crest', label="Skew:" + str(round(skew(df\_skewed[col]), 2)))  
 ax1.set\_title("Original Feature " + col, fontsize=10)  
 ax1.legend()  
 sns.histplot(ax=ax2, x=np.log1p(df\_skewed[col]), kde=True, palette='crest', label="Skew:" + str(round(skew(np.log1p(df\_skewed[col])), 2)))  
 ax2.set\_title("Log Transformed Feature " + col, fontsize=10)  
 ax2.legend()



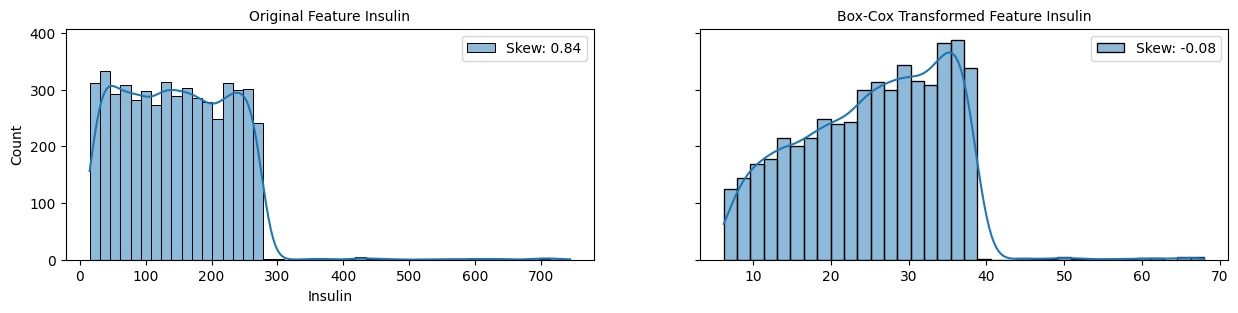






**Comment:** It is seen that log transformation successfully made skewed features more symmetrical, however, it seems that log transformation works only for two features. The Insulin is still skewed, it just changed the side. We will try another method for this feature. Box-Cox transformation belongs to the same family of power transformations.

transformed\_Insulin, optimal\_lambda = boxcox(df\_encoded["Insulin"])  
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 3), sharey=True)  
sns.histplot(ax=ax1, x=df\_skewed["Insulin"], kde=True, palette='crest', label="Skew: " + str(round(skew(df\_skewed["Insulin"]), 2)))  
ax1.set\_title("Original Feature Insulin", fontsize=10)  
ax1.legend()  
sns.histplot(ax=ax2, x=transformed\_Insulin, kde=True, palette='crest', label="Skew: " + str(round(skew(transformed\_Insulin), 2)))  
ax2.set\_title("Box-Cox Transformed Feature Insulin", fontsize=10)  
ax2.legend()  
plt.show()



**Comment:**

A skewness value of 2.76 indicates that the distribution of the data is highly skewed to the right (positive skew).

Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean.

If the skewness is less than -1 or greater than 1, the distribution is highly skewed. If the skewness is between -1 and -0.5 or between 0.5 and 1, the distribution is moderately skewed. If the skewness is between -0.5 and 0.5, the distribution is approximately symmetric. In your case, with a skewness of 2.76, the distribution is highly skewed to the right, meaning that it has a long tail on the right side and the majority of the data points are clustered on the left side of the distribution.

When the data is highly skewed, it can affect the performance of certain statistical analyses and machine learning algorithms that assume normality or symmetric distributions. It's important to be aware of the skewness of the data and consider appropriate transformations or adjustments to account for it, if necessary, before conducting further analysis.

## Skewed features - Handling

### Box-Cox Transform

cols\_list = ['Glucose', 'Insulin', 'HeartRate']  
**for** col **in** cols\_list:  
 transformed\_feature1, lambda\_value = boxcox(df\_skewed[col])  
 df\_skewed[col] = transformed\_feature1

### Log Transform

cols\_list = ['Glucose', 'Insulin', 'HeartRate']  
**for** col **in** cols\_list:  
 df\_skewed[col] = np.log1p(df\_skewed[col])

### Removing outliers

We need to understand if those outliers are anomalies. Even if they are out of normal distribution they may still contain important information for our research. We may lose trust in our data. We have outliers which cause skewness in those 3 features we were working on log transform We can try to see if there is any pattern or correlation of outliers occurring, however, we will likely discover the right way only after comparing the models' performance using different approaches to deal with this skewness.

Q1 = df\_skewed['HeartRate'].quantile(0.25)  
Q3 = df\_skewed['HeartRate'].quantile(0.75)  
IQR = Q3 - Q1  
threshold = 1.5  
outliers = df\_skewed[(df\_skewed['HeartRate'] < (Q1 - threshold \* IQR)) | (df\_skewed['HeartRate'] > (Q3 + threshold \* IQR))]  
  
outlier = df\_encoded.HeartRate.quantile(0.95)  
df\_skewed = df\_encoded.loc[df.HeartRate < outlier, :]  
outlier = df\_encoded.Insulin.quantile(0.95)  
df\_skewed = df\_encoded.loc[df.Insulin < outlier, :]  
outlier = df\_encoded.Glucose.quantile(0.95)  
df\_skewed = df\_encoded.loc[df.Glucose < outlier, :]

## Feature Engeneering

### Adding columns

df\_engen = df\_skewed.copy()

df\_engen['Age\_vs\_BloodPressure'] = df\_encoded['Age'] / (df\_encoded['BloodPressure'])  
df\_engen['Age\_vs\_Cholesterol'] = df\_encoded['Age'] / (df\_encoded['Cholesterol'])  
df\_engen['Age\_vs\_Glucose'] = df\_encoded['Age'] / (df\_encoded[‘Glucose’])

>>

## Correlations

### HeatmapImage

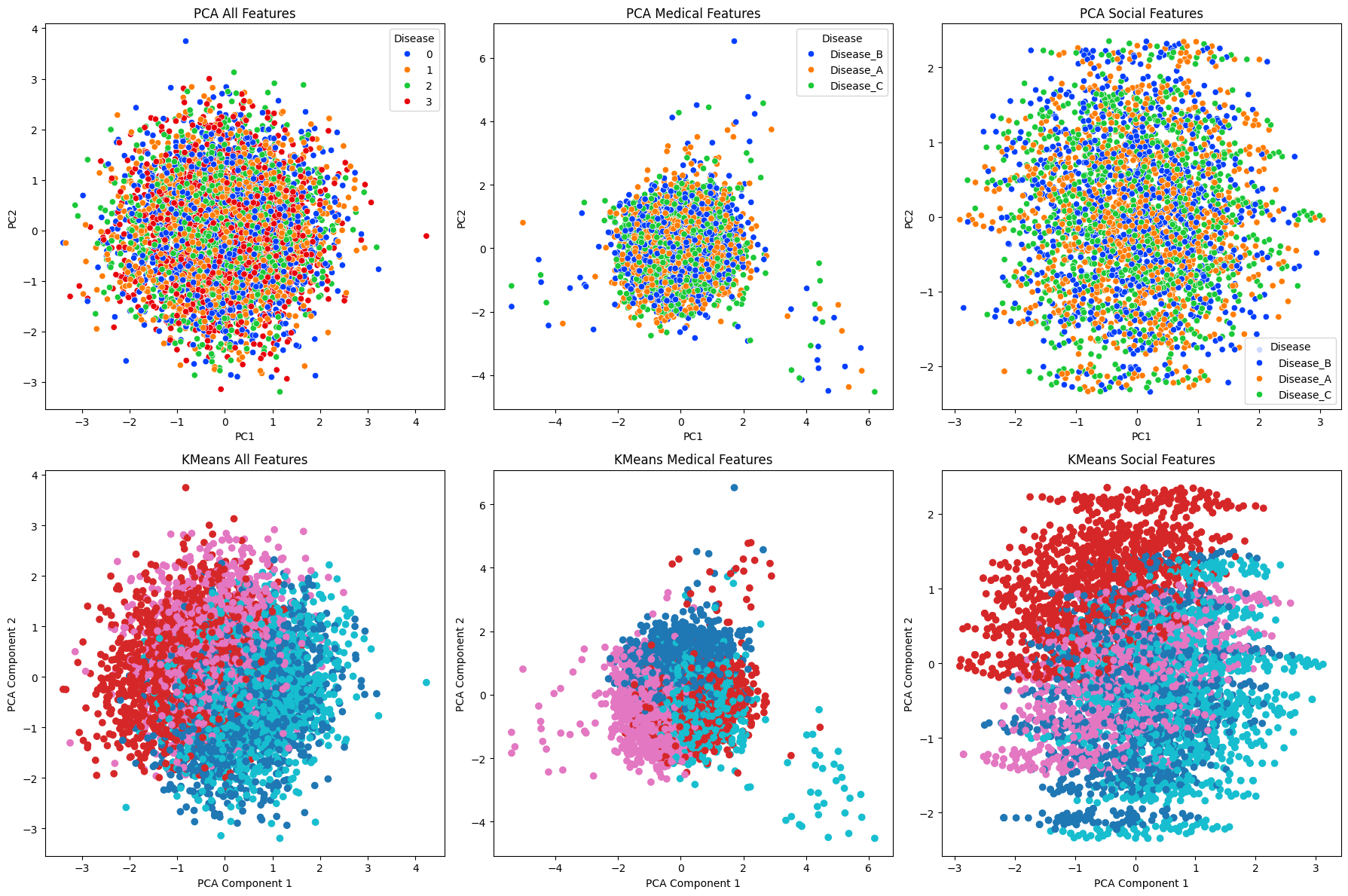
Correlations with Disease  
Age -0.013555  
BMI 0.037779  
BloodPressure 0.008862  
Cholesterol -0.011784  
Glucose -0.014706  
Insulin 0.007705  
HeartRate 0.021838  
Smoker 0.006983  
PhysicalActivity -0.011412  
SleepTime -0.015536  
MedicalHistoryScore -0.013772  
IncomeLevel 0.005515  
StressLevel 0.004620  
HealthIns -0.007615  
Disease 1.000000  
Age\_vs\_BMI -0.033194  
Age\_vs\_BloodPressure -0.012356  
Age\_vs\_Cholesterol -0.002936  
Age\_vs\_Glucose -0.002323  
Age\_vs\_Insulin -0.011002  
Age\_vs\_HeartRate -0.020938  
BMI\_vs\_BloodPressure 0.022958  
BMI\_vs\_Cholesterol 0.028515  
BMI\_vs\_Glucose 0.027584  
BMI\_vs\_Insulin -0.000434  
BMI\_vs\_HeartRate 0.021793  
BloodPressure\_vs\_Cholesterol 0.006682  
BloodPressure\_vs\_Glucose 0.008681  
BloodPressure\_vs\_Insulin -0.006443  
BloodPressure\_vs\_HeartRate -0.007391  
Cholesterol\_vs\_Glucose -0.002848  
Cholesterol\_vs\_Insulin -0.012249  
Cholesterol\_vs\_HeartRate -0.017332  
Glucose\_vs\_Insulin -0.015011  
Glucose\_vs\_HeartRate -0.022791  
Insulin\_vs\_HeartRate 0.002609  
Name: Disease, dtype: float64

label\_encoder2 = LabelEncoder()  
df\_encoded['Disease'] = label\_encoder.fit\_transform(df['Disease'])  
df\_encoded2 = df\_encoded.drop('Disease', axis=1)   
sns.heatmap(df\_encoded.corr(), cmap='coolwarm')  
plt.title('Heatmap')  
plt.show()

corr\_matrix = df\_encoded.corr()  
target\_feature = 'Disease'  
correlations\_with\_target = corr\_matrix[target\_feature]  
print("Correlations with", target\_feature)  
print(correlations\_with\_target)

# Data Modeling - Unsupervised algorithms

## PCA + KMean

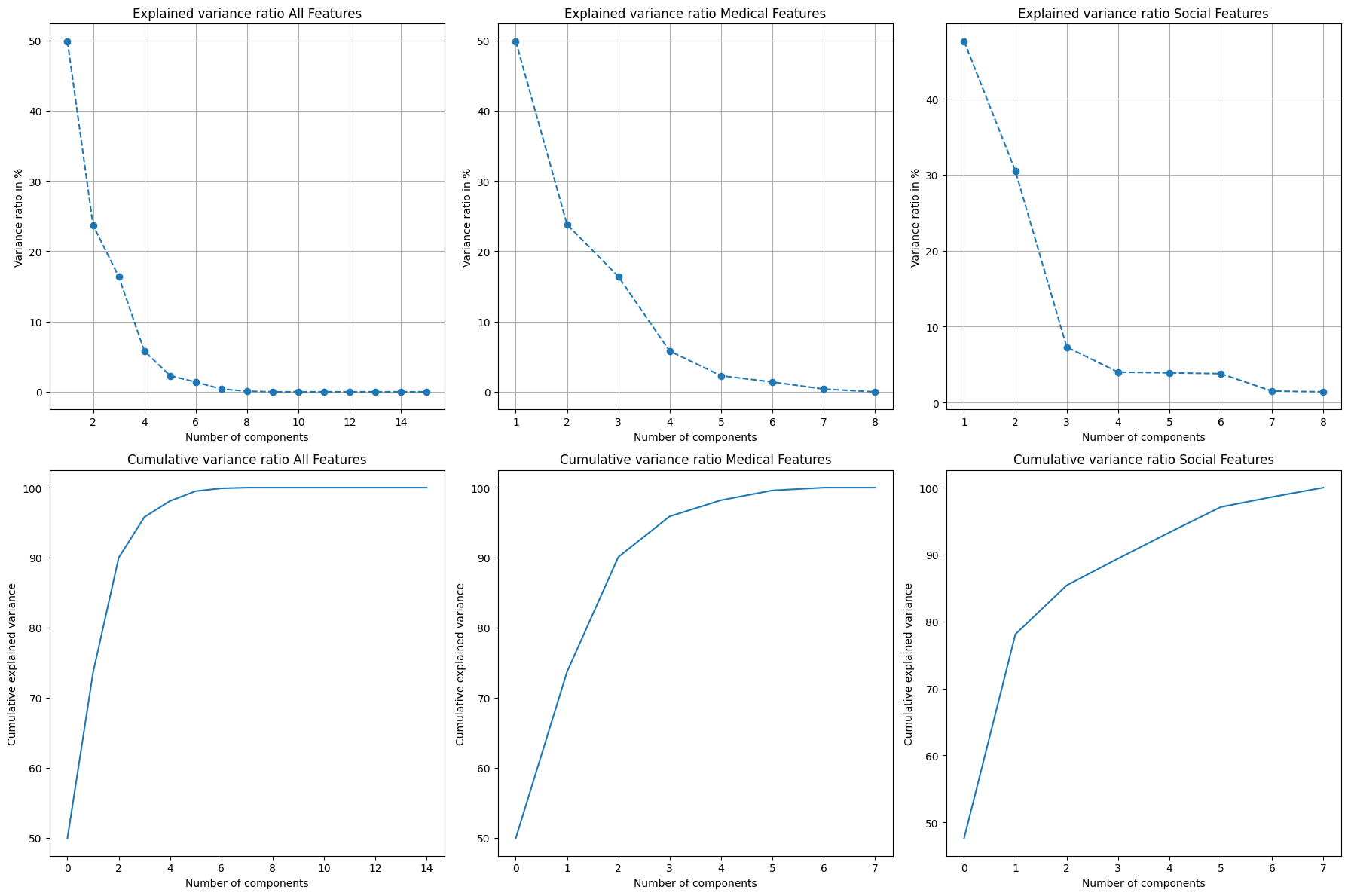


fig, axs = plt.subplots(2, 3, figsize=(18, 12))  
**def** plot\_pca\_and\_kmeans(ax\_pca, ax\_kmeans, features, target, title):  
 scaled\_features = scaler.fit\_transform(features)  
   
 pca = PCA(n\_components=2)  
 principal\_components = pca.fit\_transform(scaled\_features)  
 principal\_df = pd.DataFrame(data=principal\_components, columns=['PC1', 'PC2'])  
 final\_df = pd.concat([principal\_df, target], axis=1)  
   
 sns.scatterplot(data=final\_df, x='PC1', y='PC2', hue=target.name, palette='bright', ax=ax\_pca)  
 ax\_pca.set\_title(f'PCA {title}')  
   
 kmeans = KMeans(n\_clusters=4, random\_state=42)  
 clusters = kmeans.fit\_predict(scaled\_features)  
   
 ax\_kmeans.scatter(principal\_components[:, 0], principal\_components[:, 1], c=clusters, cmap='tab10')  
 ax\_kmeans.set\_title(f'KMeans {title}')  
 ax\_kmeans.set\_xlabel('PCA Component 1')  
 ax\_kmeans.set\_ylabel('PCA Component 2')  
  
plot\_pca\_and\_kmeans(axs[0, 0], axs[1, 0], df\_encoded2, df\_target['Disease'], 'All Features')  
plot\_pca\_and\_kmeans(axs[0, 1], axs[1, 1], df\_medical\_cols\_pca, df['Disease'], 'Medical Features')  
plot\_pca\_and\_kmeans(axs[0, 2], axs[1, 2], df\_social\_cols\_pca, df['Disease'], 'Social Features')  
plt.tight\_layout()  
plt.show()

### Explained and Cumulative variance ratio

fig, axs = plt.subplots(2, 3, figsize=(18, 12))  
pca = PCA()  
  
df\_transform = scaler.fit\_transform(df\_encoded)  
pca.fit(df\_encoded)  
variations1 = np.round(pca.explained\_variance\_ratio\_ \* 100, decimals=1)  
labels = [str(x) **for** x **in** range(1, len(variations1) + 1)]  
axs[0, 0].plot(range(1, len(variations1) + 1), variations1, marker='o', linestyle='--')  
axs[0, 0].set\_xlabel('Number of components')  
axs[0, 0].set\_ylabel('Variance ratio in %')  
axs[0, 0].set\_title('Explained variance ratio All Features’)

axs[0, 0].grid(True)



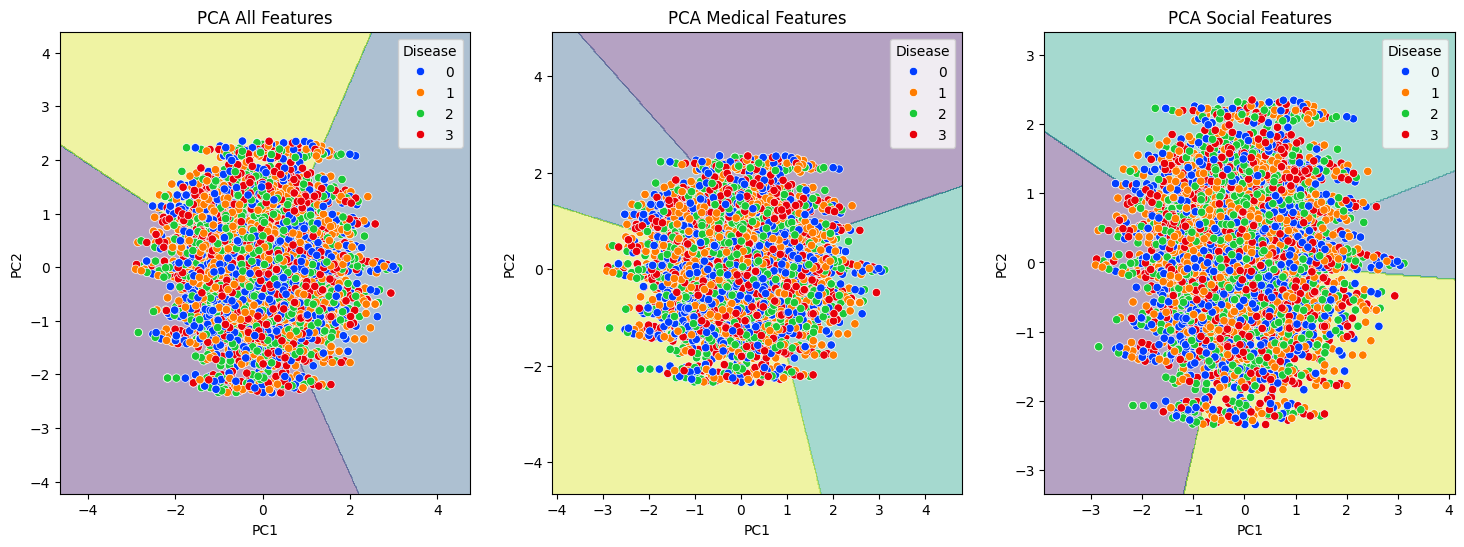
### Decision boundary

**def** plot\_decision\_boundary(clf, X, y, ax, title):  
 h = .02   
 x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1  
 y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1  
 xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),  
 np.arange(y\_min, y\_max, h))  
 Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])  
  
 Z = Z.reshape(xx.shape)  
 ax.contourf(xx, yy, Z, alpha=0.4, cmap='viridis')  
  
le = LabelEncoder()  
df\_encoded['Disease'] = le.fit\_transform(df\_encoded['Disease'])  
  
fig, axs = plt.subplots(1, 3, figsize=(18, 6))  
  
*# All Features*  
scaled\_features = scaler.fit\_transform(df\_encoded.drop(columns=['Disease']))  
pca = PCA(n\_components=2)  
principal\_components = pca.fit\_transform(scaled\_features)  
X = principal\_components  
y = df\_encoded['Disease']  
clf = SVC(kernel='linear')  
clf.fit(X, y)  
plot\_decision\_boundary(clf, X, y, axs[0], 'PCA All Features')  
sns.scatterplot(data=final\_df, x='PC1', y='PC2', hue='Disease', palette='bright', ax=axs[0])  
axs[0].set\_title('PCA All Features')

axs[0, 0].grid(True)

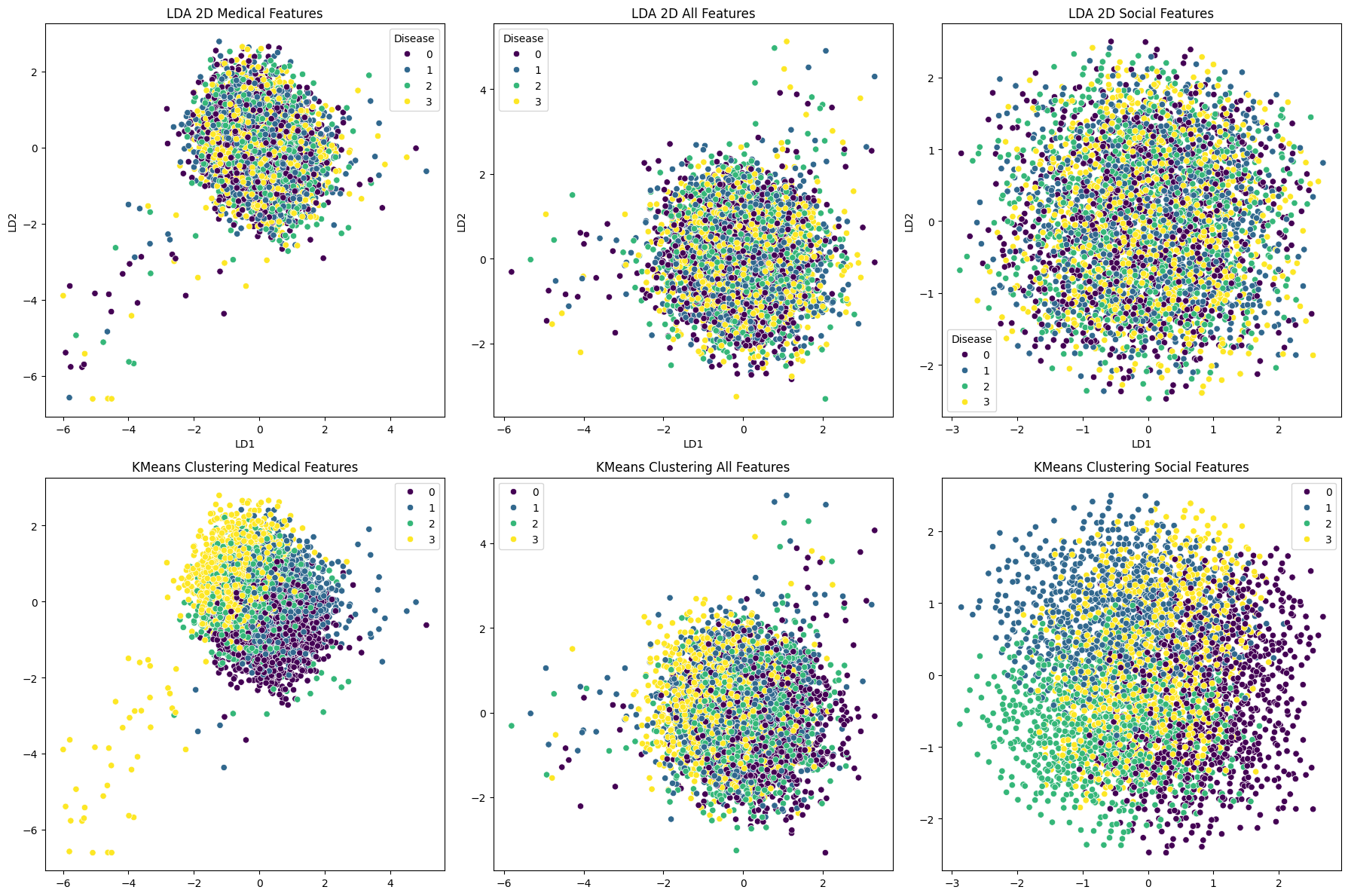
*# Medical Features*

*# Social Features*



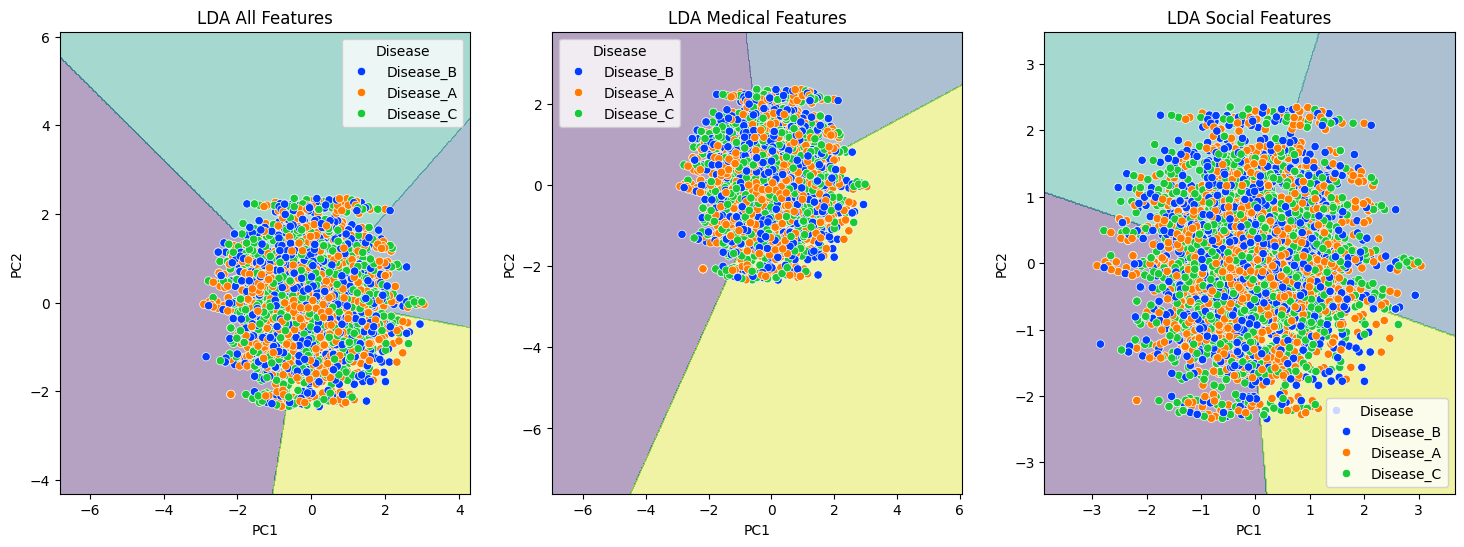
## LDA + KMean

fig, axs = plt.subplots(2, 3, figsize=(18, 12))  
  
*# Medical Features - LDA*  
scaler = StandardScaler()  
scaled\_features = scaler.fit\_transform(df\_medical\_cols.drop(columns=['Disease']))  
lda = LDA(n\_components=2)  
lda\_components = lda.fit\_transform(scaled\_features, df\_medical\_cols['Disease'])  
lda\_df = pd.DataFrame(data=lda\_components, columns=['LD1', 'LD2'])  
final\_lda\_df = pd.concat([lda\_df, df\_medical\_cols['Disease']], axis=1)  
sns.scatterplot(data=final\_lda\_df, x='LD1', y='LD2', hue='Disease', palette='viridis', ax=axs[0, 0])  
axs[0, 0].set\_title('LDA 2D Medical Features')  
  
*# KMeans Clustering - Medical Features*  
kmeans = KMeans(n\_clusters=4, random\_state=42)  
clusters = kmeans.fit\_predict(scaled\_features)  
sns.scatterplot(x=lda\_components[:, 0], y=lda\_components[:, 1], hue=clusters, palette='viridis', ax=axs[1, 0])  
axs[1, 0].set\_title('KMeans Clustering Medical Features')



### Decision Boundary

**def** plot\_decision\_boundary(clf, X, y, ax, title):  
 h = .02 *# step size in the mesh*  
 x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1  
 y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1  
 xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),  
 np.arange(y\_min, y\_max, h))  
 Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])  
  
 Z = Z.reshape(xx.shape)  
 ax.contourf(xx, yy, Z, alpha=0.4, cmap='viridis')  
  
le = LabelEncoder()  
df\_encoded['Disease'] = le.fit\_transform(df\_encoded['Disease'])  
  
fig, axs = plt.subplots(1, 3, figsize=(18, 6))  
  
*# All Features*  
lda = LDA(n\_components=2)  
X = lda.fit\_transform(df\_encoded.drop(columns=['Disease']), df\_encoded['Disease'])  
y = df\_encoded['Disease']  
clf = SVC(kernel='linear')  
clf.fit(X, y)  
plot\_decision\_boundary(clf, X, y, axs[0], 'LDA All Features')  
sns.scatterplot(data=final\_df, x='PC1', y='PC2', hue='Disease', palette='bright', ax=axs[0])  
axs[0].set\_title('LDA All Features’)

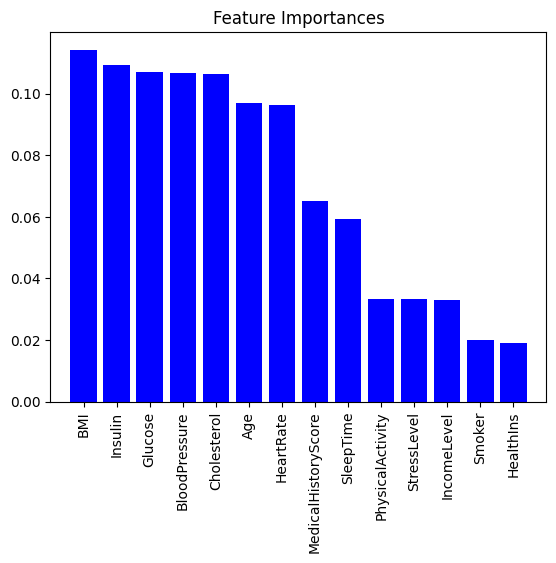


## Feature importance

model = RandomForestClassifier(n\_estimators=100, random\_state=42)  
model.fit(X, y)  
importances = model.feature\_importances\_  
indices = np.argsort(importances)[::-1]  
plt.figure()  
plt.title('Feature Importances’)

plt.bar(range(X.shape[1]), importances[indices], color='b', align='center')  
plt.xticks(range(X.shape[1]), [X.columns[i] **for** i **in** indices], rotation=90)  
plt.xlim([-1, X.shape[1]])  
plt.show()

Feature Importance  
1 BMI 0.202450  
3 Cholesterol 0.124421  
4 Glucose 0.116953  
5 Insulin 0.114707  
0 Age 0.111388  
6 HeartRate 0.098907  
2 BloodPressure 0.076149  
10 MedicalHistoryScore 0.046438  
9 SleepTime 0.039164  
12 StressLevel 0.027063  
8 PhysicalActivity 0.012560  
7 Smoker 0.011204  
13 HealthIns 0.010536  
11 IncomeLevel 0.008061



X=df\_engen.drop(columns=['Disease'],axis = 1)  
y=df\_engen['Disease']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)  
gb\_model = GradientBoostingClassifier()  
gb\_model.fit(X\_train, y\_train)  
feature\_importances = gb\_model.feature\_importances\_  
pd.set\_option('display.max\_rows', None)   
pd.set\_option('display.max\_columns', None)   
importance\_df = pd.DataFrame({'Feature': X\_train.columns, 'Importance': feature\_importances})  
importance\_df = importance\_df.sort\_values(by='Importance', ascending=False)  
print(importance\_df)

## Evaluation

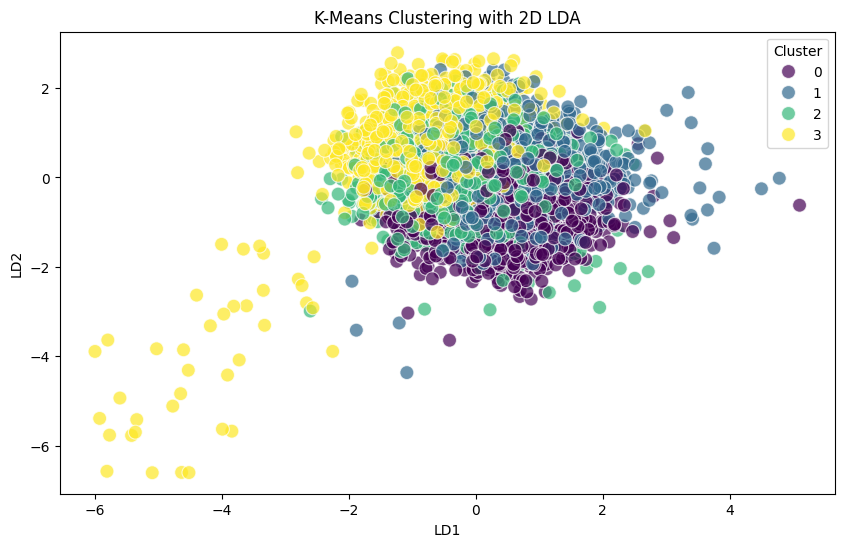
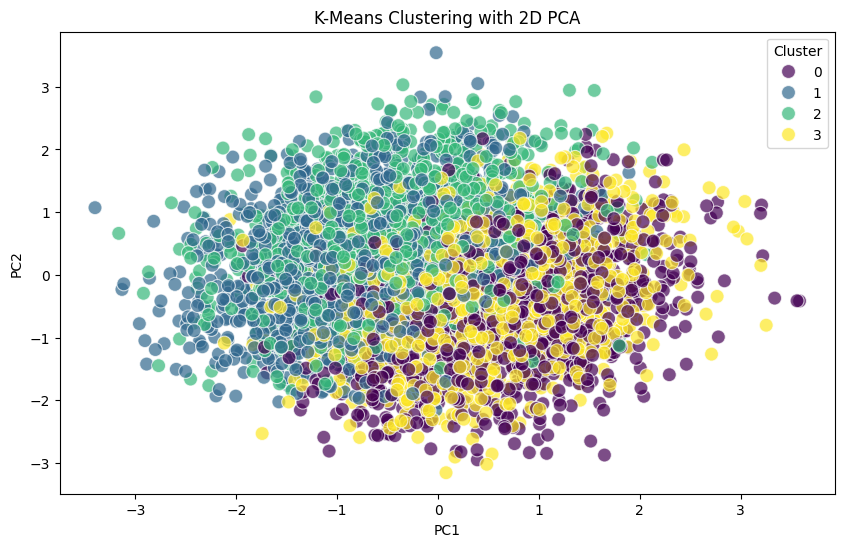
scaler = StandardScaler()  
df\_scaled = scaler.fit\_transform(df\_encoded2)  
kmeans = KMeans(n\_clusters=4, random\_state=42)  
clusters = kmeans.fit\_predict(df\_scaled)  
df\_with\_clusters = pd.DataFrame(df\_scaled, columns=df\_encoded2.columns)  
df\_with\_clusters['Cluster'] = clusters  
pca = PCA(n\_components=2)  
pca\_components = pca.fit\_transform(df\_scaled)  
pca\_df = pd.DataFrame(data=pca\_components, columns=['PC1', 'PC2'])  
pca\_df['Cluster'] = clusters  
plt.figure(figsize=(10, 6))  
sns.scatterplot(data=pca\_df, x='PC1', y='PC2', hue='Cluster', palette='viridis', alpha=0.7, s=100)  
plt.title('K-Means Clustering with 2D PCA')  
plt.show()  
silhouette\_avg = silhouette\_score(df\_scaled, clusters)  
print(f'Silhouette Score: {silhouette\_avg}’)

### PCA + Kmean

Silhouette Score: 0.05385180947545493

### LDA + KMean

Silhouette Score: 0.09985965416766922

As we can see separation of clusters is not clear in both algorithms (PCA, LDA). Even though we can visually separate classes it does not mean that KMean assigns correct classes to each record.

scaler = StandardScaler()  
df\_scaled = scaler.fit\_transform(df\_medical\_cols.drop(columns=['Disease']))  
k = 4  
kmeans = KMeans(n\_clusters=k, random\_state=42)  
kmeans.fit(df\_scaled)  
clusters = kmeans.predict(df\_scaled)  
df\_medical\_cols\_pca['Cluster'] = clusters  
lda = LDA(n\_components=2)  
lda\_components = lda.fit\_transform(df\_scaled, df\_medical\_cols['Disease'])  
lda\_df = pd.DataFrame(data=lda\_components, columns=['LD1', 'LD2'])  
lda\_df['Cluster'] = clusters  
plt.figure(figsize=(10, 6))  
sns.scatterplot(data=lda\_df, x='LD1', y='LD2', hue='Cluster', palette='viridis', alpha=0.7, s=100)  
plt.title('K-Means Clustering with 2D LDA')  
plt.show()  
silhouette\_avg = silhouette\_score(df\_scaled, clusters)  
print(f'Silhouette Score: {silhouette\_avg}’)

# Data Modeling - Supervised algorithms

#### Scaling

##### Standard Scaler

df\_scaled = pd.DataFrame(df\_scaled)  
df\_scaled = df\_scaled.astype(int)  
pca = PCA(n\_components=2)  
X = df\_scaled.drop('Disease', axis=1)  
pca\_Disease = pca.fit\_transform(X)  
X = df\_scaled.drop('HealthIns', axis=1)  
pca\_HealthIns = pca.fit\_transform(X)  
X = df\_scaled.drop('StressLevel', axis=1)  
pca\_StressLevel = pca.fit\_transform(X)  
X = df\_scaled.drop('MedicalHistoryScore', axis=1)  
pca\_MedicalHistoryScore = pca.fit\_transform(X)  
X = df\_scaled.drop('PhysicalActivity', axis=1)  
pca\_PhysicalActivity = pca.fit\_transform(X)  
X = df\_scaled.drop('Smoker', axis=1)  
pca\_Smoker = pca.fit\_transform(X)

numeric\_columns = ['Age', 'BMI', 'BloodPressure', 'Cholesterol', 'Glucose', 'Insulin', 'HeartRate']   
scaler = StandardScaler()  
column\_transformer = ColumnTransformer(  
 transformers=[('scaler', scaler, numeric\_columns)], remainder='passthrough')  
df\_scaled = column\_transformer.fit\_transform(df\_engen)  
df\_scaled = pd.DataFrame(df\_scaled, columns=numeric\_columns + [col **for** col **in** df\_engen.columns **if** col **not** **in** numeric\_columns])

numeric\_columns = ['Age', 'BMI', 'BloodPressure', 'Cholesterol', 'Glucose', 'Insulin', 'HeartRate']   
scaler = MinMaxScaler()  
column\_transformer = ColumnTransformer(  
 transformers=[('scaler', scaler, numeric\_columns)], remainder='passthrough')  
df\_scaled = column\_transformer.fit\_transform(df\_engen)  
df\_scaled = pd.DataFrame(df\_scaled, columns=numeric\_columns + [col **for** col **in** df\_engen.columns **if** col **not** **in** numeric\_columns])

##### Min-Max Scaler

## Categorical targets

### PCA Transform

### Define Targets to test

### Define hyperparameter to tune

datasets = {  
 'Disease': (df\_scaled.drop('Disease', axis=1), df\_scaled['Disease']),  
 'HealthIns': (df\_scaled.drop('HealthIns', axis=1), df\_scaled['HealthIns']),  
 'StressLevel': (df\_scaled.drop('StressLevel', axis=1), df\_scaled['StressLevel']),  
 'MedicalHistoryScore': (df\_scaled.drop('MedicalHistoryScore', axis=1), df\_scaled['MedicalHistoryScore']),  
 'PhysicalActivity': (df\_scaled.drop('PhysicalActivity', axis=1), df\_scaled['PhysicalActivity']),  
 'Smoker': (df\_scaled.drop('Smoker', axis=1), df\_scaled['Smoker']),  
 'pca\_Disease': (pca\_Disease, df\_scaled['Disease']),  
 'pca\_HealthIns': (pca\_HealthIns, df\_scaled['HealthIns']),  
 'pca\_StressLevel': (pca\_StressLevel, df\_scaled['StressLevel']),  
 'pca\_MedicalHistoryScore': (pca\_MedicalHistoryScore, df\_scaled['MedicalHistoryScore']),  
 'pca\_PhysicalActivity': (pca\_PhysicalActivity, df\_scaled['PhysicalActivity']),  
 'pca\_Smoker': (pca\_Smoker, df\_scaled['Smoker'])  
}

param\_svc\_model = {'C': [0.1, 1],'kernel': ['linear', 'rbf', 'sigmoid'],'gamma': ['scale', 'auto']}  
param\_xgb\_model = {'learning\_rate': [0.01, 0.1, 0.3],'n\_estimators': [100, 200, 300],'max\_depth': [3, 5, 7]}  
param\_log\_model = {'C': [0.01, 0.1, 1, 10],'penalty': ['l1', 'l2', 'none'],'solver': ['newton-cg', 'saga']}  
param\_dt\_model = {'criterion': ['gini', 'entropy'],'max\_depth': [5, 10, 15],'min\_samples\_split': [2, 5, 10],'min\_samples\_leaf':[1, 2, 4]}  
param\_rf\_model = {'n\_estimators': [100, 200],'criterion': ['gini', 'entropy'],'max\_depth': [5, 10],'min\_samples\_split': [5, 10], 'min\_samples\_leaf': [2, 4]}  
param\_gb\_model = {'n\_estimators': [200, 300],'max\_depth': [5, 7]}  
param\_knn\_model = {'n\_neighbors': [3, 5, 7],'weights': ['uniform', 'distance'],p': [1, 2]}  
param\_mlp\_model = {'activation': ['identity', 'logistic'],'solver': ['adam', 'lbfgs'],'alpha': [0.001, 0.01]}

### Define Models for GridSearch

models\_params = {  
 'Logistic Regression': (LogisticRegression(), param\_log\_model),  
 'Decision Tree': (DecisionTreeClassifier(), param\_dt\_model),  
 'Random Forest': (RandomForestClassifier(), param\_rf\_model),  
 'Gradient Boosting': (GradientBoostingClassifier(), param\_gb\_model),  
 'KNN': (KNeighborsClassifier(), param\_knn\_model),  
 'MLP': (MLPClassifier(), param\_mlp\_model),  
 'SVC': (SVC(), param\_svc\_model),  
 'XGBoost': (XGBClassifier(), param\_xgb\_model)  
}

### Fit Models

In this stage, we will apply ML models to different categorical features to assess which feature is being predicted better.

**Code comment:**

We will create a loop function to collect metrics indicating models' performance together for evaluation and test different target features, define lists with models and data with targets, and record metrics for evaluation to a data frame.

accuracy\_classif = []  
confusion\_matrixes = {}  
dataset\_progress = tqdm(datasets.items(), desc="Datasets", position=0, leave=True)  
**for** dataset\_name, (X, y) **in** dataset\_progress:  
 dataset\_progress.set\_description(f"Processing {dataset\_name}")  
   
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
  
 model\_progress = tqdm(models\_params.items(), desc="Models", position=1, leave=False)  
   
 **for** model\_name, (model, params) **in** model\_progress:  
 model\_progress.set\_description(f"Training {model\_name}")  
   
 GS\_model = GridSearchCV(estimator=model,   
 param\_grid=params,  
 scoring=["accuracy", 'neg\_mean\_squared\_error'],  
 refit="accuracy",  
 cv=5,  
 verbose=0)  
   
 GS\_model.fit(X\_train, y\_train)  
 y\_train\_pred = GS\_model.predict(X\_train)  
 y\_test\_pred = GS\_model.predict(X\_test)  
 train\_accuracy = accuracy\_score(y\_train, y\_train\_pred)  
 test\_accuracy = accuracy\_score(y\_test, y\_test\_pred)  
 precision = precision\_score(y\_test, y\_test\_pred, average='weighted')  
 recall = recall\_score(y\_test, y\_test\_pred, average='weighted')  
 f1 = f1\_score(y\_test, y\_test\_pred, average='weighted')  
 best\_estimator = GS\_model.best\_estimator\_  
 best\_parameters = GS\_model.best\_params\_  
 accuracy\_classif.append({  
 'Dataset': dataset\_name,  
 'Model': model\_name,  
 'Train accuracy': train\_accuracy,  
 'Test accuracy': test\_accuracy,  
 'Precision': precision,  
 'Recall': recall,  
 'F1': f1,  
 'Best estimator': best\_estimator,  
 'Best parameters': best\_parameters  
 })  
 confusion\_key = f'{dataset\_name}\_{model\_name}'  
 confusion\_matrixes[confusion\_key] = confusion\_matrix(y\_test, y\_test\_pred)

## Numeric targets

### Define Target to test

### Define hyperparameter to tune

datasets = {  
 'HeartRate': (df\_scaled.drop('HeartRate', axis=1), df\_scaled['HeartRate']),  
 'Insulin': (df\_scaled.drop('Insulin', axis=1), df\_scaled['Insulin']),  
 'Glucose': (df\_scaled.drop('Glucose', axis=1), df\_scaled['Glucose']),  
 'Cholesterol': (df\_scaled.drop('Cholesterol', axis=1), df\_scaled['Cholesterol']),  
 'BloodPressure': (df\_scaled.drop('BloodPressure', axis=1), df\_scaled['BloodPressure']),  
 'BMI': (df\_scaled.drop('BMI', axis=1), df\_scaled['BMI']),  
 'Age': (df\_scaled.drop('Age', axis=1), df\_scaled['Age'])}

linear\_params = {}  
ridge\_params = {'alpha': [0.1, 1.0, 10.0]}  
lasso\_params = {'alpha': [0.1, 1.0, 10.0]}  
tree\_params = {'max\_depth': [None, 10, 20],'min\_samples\_split': [2, 5, 10]}  
forest\_params = {'n\_estimators': [100, 200, 300],  
 'max\_depth': [None, 10, 20],'min\_samples\_split': [2, 5, 10]}  
gboost\_params = {'n\_estimators': [100, 200, 300],'learning\_rate': [0.01, 0.1, 0.5]}  
svr\_params = {'C': [0.1, 1.0, 10.0], 'epsilon': [0.1, 0.2, 0.5]}  
knn\_params = {'n\_neighbors': [3, 5, 7],'weights': ['uniform', 'distance']}

### Define Models for GridSearch

models\_params = {  
 'Linear Regression': (LinearRegression(), linear\_params),  
 'Ridge Regression': (Ridge(), ridge\_params),  
 'Lasso Regression': (Lasso(), lasso\_params),  
 'Decision Tree Regression': (DecisionTreeRegressor(), tree\_params),  
 'Random Forest Regression': (RandomForestRegressor(), forest\_params),  
 'Gradient Boosting Regression': (GradientBoostingRegressor(), gboost\_params),  
 'SVR': (SVR(), svr\_params),  
 'k-NN Regression': (KNeighborsRegressor(), knn\_params)}

### Fit Models

accuracy\_regress = []  
dataset\_progress = tqdm(datasets.items(), desc="Datasets", position=0, leave=True)  
**for** dataset\_name, (X, y) **in** dataset\_progress:  
 dataset\_progress.set\_description(f"Processing {dataset\_name}")  
   
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
  
 model\_progress = tqdm(models\_params.items(), desc="Models", position=1, leave=False)  
 **for** model\_name, (model, params) **in** model\_progress:  
 model\_progress.set\_description(f"Training {model\_name}")   
 GS\_model = GridSearchCV(estimator=model,   
 param\_grid=params,  
 scoring={'neg\_mean\_squared\_error': 'neg\_mean\_squared\_error',   
 'neg\_mean\_absolute\_error': 'neg\_mean\_absolute\_error',   
 'r2': 'r2'},   
 refit='r2',  
 cv=5,  
 verbose=0)   
   
 GS\_model.fit(X\_train, y\_train)  
 y\_train\_pred = GS\_model.predict(X\_train)  
 y\_test\_pred = GS\_model.predict(X\_test)  
 mse\_train = mean\_squared\_error(y\_train, y\_train\_pred)  
 mse\_test = mean\_squared\_error(y\_test, y\_test\_pred)  
 mae\_train = mean\_absolute\_error(y\_train, y\_train\_pred)  
 mae\_test = mean\_absolute\_error(y\_test, y\_test\_pred)  
 r2\_train = r2\_score(y\_train, y\_train\_pred)  
 r2\_test = r2\_score(y\_test, y\_test\_pred)  
 best\_estimator = GS\_model.best\_estimator\_  
 best\_parameters = GS\_model.best\_params\_  
 accuracy\_regress.append({  
 'Dataset': dataset\_name,  
 'Model': model\_name,  
 'Training MSE': mse\_train,  
 'Testing MSE': mse\_test,  
 'Training MAE': mae\_train,  
 'Testing MAE': mae\_test,  
 'Training R²': r2\_train,  
 'Testing R²': r2\_test,  
 'Best estimator': best\_estimator,  
 'Best parameters': best\_parameters})

## Data Modeling - ANN (PyTorch) + CV

Fold 20 Accuracy: 0.2640  
Mean Cross-Validation Accuracy: 0.2486  
[{'Dataset': 'Disease', 'Model': 'PyTorch ANN', 'Cross-Validation Accuracy': 0.2486}]

X = df\_scaled.drop('Disease', axis=1)  
y = df\_scaled['Disease']  
X = X.astype(float)  
y = y.astype(int)  
accuracy\_dl = []  
scaler = StandardScaler()  
X = scaler.fit\_transform(X)  
X\_tensor = torch.tensor(X, dtype=torch.float32)  
y\_tensor = torch.tensor(y.values, dtype=torch.long)  
num\_classes = 4  
num\_epochs = 400  
batch\_size = 4  
k\_folds = 20  
**class** ANN(nn.Module):  
 **def** \_\_init\_\_(self, input\_size, num\_classes):  
 super(ANN, self).\_\_init\_\_()  
 self.fc1 = nn.Linear(input\_size, 64)  
 self.relu = nn.ReLU()  
 self.fc2 = nn.Linear(64, num\_classes)   
 **def** forward(self, x):  
 x = self.fc1(x)  
 x = self.relu(x)  
 x = self.fc2(x)  
 **return** x  
kf = KFold(n\_splits=k\_folds, shuffle=True, random\_state=42)  
fold\_accuracies = []  
**for** fold, (train\_index, test\_index) **in** enumerate(kf.split(X\_tensor)):  
 print(f'Fold {fold + 1}')  
   
 X\_train\_fold = X\_tensor[train\_index]  
 y\_train\_fold = y\_tensor[train\_index]  
 X\_test\_fold = X\_tensor[test\_index]  
 y\_test\_fold = y\_tensor[test\_index]

input\_size = X\_train\_fold.shape[1]  
 model = ANN(input\_size, num\_classes)  
 criterion = nn.CrossEntropyLoss()  
 optimizer = optim.Adam(model.parameters(), lr=0.001)  
   
 **for** epoch **in** range(num\_epochs):  
 model.train()  
 **for** i **in** range(0, len(X\_train\_fold), batch\_size):  
 inputs = X\_train\_fold[i:i+batch\_size]  
 targets = y\_train\_fold[i:i+batch\_size]  
 optimizer.zero\_grad()  
 outputs = model(inputs)  
 loss = criterion(outputs, targets)  
 loss.backward()  
 optimizer.step()  
 **if** (epoch + 1) % 10 == 0:  
 print(f'Epoch [{epoch + 1}/{num\_epochs}], Loss: {loss.item():.4f}')  
 model.eval()  
 **with** torch.no\_grad():  
 outputs\_test = model(X\_test\_fold)  
 \_, y\_test\_pred = torch.max(outputs\_test, 1)  
 y\_test\_pred\_np = y\_test\_pred.numpy()  
 y\_test\_np = y\_test\_fold.numpy()  
 fold\_accuracy = accuracy\_score(y\_test\_np, y\_test\_pred\_np)  
 fold\_accuracies.append(fold\_accuracy)  
 print(f'Fold {fold + 1} Accuracy: {fold\_accuracy:.4f}')  
mean\_accuracy = np.mean(fold\_accuracies)  
print(f'Mean Cross-Validation Accuracy: {mean\_accuracy:.4f}')  
accuracy\_dl.append({  
 'Dataset': 'Disease',  
 'Model': 'PyTorch ANN',  
 'Cross-Validation Accuracy': mean\_accuracy})  
print(accuracy\_dl)

## Evaluation

### Predictability by TaImagerget (Classification)

#### Accuracy

#### Confusion metrix

MedicalHistoryScore\_Logistic Regression:  
[[ 2 2 4 55 14 0 12 56 1 3]  
 [ 1 4 0 57 10 0 17 59 2 2]  
 [ 4 2 5 60 12 1 18 48 1 2]  
 [ 3 2 0 63 11 0 14 41 1 3]  
 [ 2 0 3 53 15 1 20 61 0 2]  
 [ 6 1 4 53 13 0 22 58 1 3]  
 [ 4 1 2 64 15 1 15 45 1 4]  
 [ 3 1 1 41 17 0 19 59 2 3]  
 [ 6 2 1 61 10 1 10 57 0 5]  
 [ 3 1 1 54 13 0 15 48 1 3]]

MedicalHistoryScore\_Decision Tree:  
[[ 5 3 1 75 2 9 21 33 0 0]  
 [ 3 4 4 90 4 10 15 22 0 0]  
 [ 7 3 3 72 2 12 24 30 0 0]  
 [ 5 3 1 84 1 7 18 19 0 0]  
 [ 6 2 2 75 4 9 24 35 0 0]  
 [ 2 6 4 87 6 10 20 26 0 0]  
 [ 8 6 1 72 1 8 27 29 0 0]  
 [ 2 3 3 84 1 10 18 25 0 0]  
 [ 9 3 4 81 6 5 27 18 0 0]  
 [ 4 1 3 76 1 8 24 22 0 0]]

MedicalHistoryScore\_Random Forest:  
[[ 8 3 6 60 6 5 11 45 2 3]  
 [ 8 0 5 76 11 1 14 34 2 1]  
 [12 1 6 69 5 6 16 33 4 1]  
 [ 8 1 2 71 5 0 12 34 3 2]  
 [ 9 1 3 70 5 3 24 38 2 2]  
 [ 7 3 10 71 8 3 23 31 4 1]  
 [ 8 1 5 70 8 3 16 36 2 3]  
 [ 9 3 2 63 9 5 8 41 5 1]  
 [ 8 3 10 67 11 4 12 34 0 4]  
 [10 1 5 59 7 0 19 35 1 2]]

MedicalHistoryScore\_Gradient Boosting:  
[[13 12 15 19 10 11 11 24 17 17]  
 [16 23 13 14 13 12 11 24 15 11]  
 [12 19 12 21 10 17 21 12 17 12]  
 [16 13 11 13 12 10 15 16 16 16]  
 [15 12 10 24 13 12 23 20 16 12]  
 [15 14 23 17 21 16 12 20 10 13]  
 [13 17 10 24 11 9 17 24 14 13]  
 [16 17 11 13 11 17 17 24 14 6]  
 [19 12 14 17 21 13 21 19 6 11]  
 [14 14 12 13 12 11 20 15 17 11]]

MedicalHistoryScore\_KNN:  
[[16 13 19 28 12 11 16 18 11 5]  
 [30 25 14 13 10 14 11 15 13 7]  
 [16 28 14 21 17 12 11 10 13 11]  
 [18 18 19 20 16 10 9 10 10 8]  
 [19 20 20 23 15 13 14 13 17 3]  
 [25 26 22 15 16 11 12 18 10 6]  
 [23 28 21 18 10 13 12 12 7 8]  
 [19 29 18 20 8 16 14 11 4 7]  
 [21 24 17 18 19 15 12 17 5 5]  
 [27 21 19 19 11 11 4 12 11 4]]

MedicalHistoryScore\_MLP:  
[[15 9 15 22 12 14 16 19 18 9]  
 [13 19 10 19 17 17 14 17 20 6]  
 [13 11 12 21 18 18 21 15 13 11]  
 [14 11 11 21 13 17 16 18 8 9]  
 [14 9 9 29 15 13 22 17 14 15]  
 [20 11 10 27 14 17 21 21 10 10]  
 [12 9 9 23 17 13 19 18 19 13]  
 [22 8 9 22 15 15 13 21 14 7]  
 [14 10 15 12 16 19 20 20 18 9]  
 [12 6 12 18 18 15 16 22 11 9]]

MedicalHistoryScore\_SVC:  
[[ 0 0 0 116 0 0 0 33 0 0]  
 [ 0 0 0 123 0 0 0 29 0 0]  
 [ 0 0 0 115 0 0 0 38 0 0]  
 [ 0 0 0 113 0 0 0 25 0 0]  
 [ 0 0 0 126 0 0 0 31 0 0]  
 [ 0 0 0 133 0 0 0 28 0 0]  
 [ 0 0 0 122 0 0 0 30 0 0]  
 [ 0 0 0 111 0 0 0 35 0 0]  
 [ 0 0 0 117 0 0 0 36 0 0]  
 [ 0 0 0 106 0 0 0 33 0 0]]

MedicalHistoryScore\_XGBoost:  
[[16 12 16 23 9 14 11 22 17 9]  
 [15 14 12 13 13 13 14 32 16 10]  
 [11 16 9 18 15 19 19 16 19 11]  
 [14 11 9 23 14 8 12 17 16 14]  
 [18 18 10 17 15 16 16 25 12 10]  
 [14 15 22 15 19 15 11 17 17 16]  
 [11 12 9 20 9 17 20 30 14 10]  
 [14 17 12 14 13 14 15 22 17 8]  
 [14 18 12 19 16 18 14 15 13 14]  
 [11 14 13 12 14 9 16 15 21 14]]

Disease\_Logistic Reg.   
[[ 11 0 0 348]  
 [ 14 0 0 366]  
 [ 14 0 0 386]  
 [ 9 0 0 352]]

Disease\_Decision Tree:  
[[108 94 80 77]  
 [122 118 71 69]  
 [117 109 91 83]  
 [104 113 54 90]]

Disease\_Random Forest:  
[[ 88 99 66 106]  
 [ 97 96 75 112]  
 [ 94 110 86 110]  
 [101 77 74 109]]

Disease\_Random Forest:  
[[ 88 99 66 106]  
 [ 97 96 75 112]  
 [ 94 110 86 110]  
 [101 77 74 109]]

Disease\_Gradient Boost:  
[[ 84 110 87 78]  
 [103 104 97 76]  
 [102 115 100 83]  
 [109 85 79 88]]

Disease\_KNN:  
[[118 110 68 63]  
 [141 98 87 54]  
 [126 111 80 83]  
 [116 93 86 66]]

Disease\_MLP:  
[[ 67 88 98 106]  
 [ 80 88 116 96]  
 [104 86 113 97]  
 [ 86 83 101 91]]

Disease\_SVC:  
[[154 0 0 205]  
 [170 0 0 210]  
 [185 0 0 215]  
 [159 0 0 202]]

Disease\_XGBoost:  
[[ 89 94 86 90]  
 [107 109 87 77]  
 [104 100 105 91]  
 [105 83 82 91]]

PhysicalActivity\_Logistic Regression:  
[[ 0 0 536]  
 [ 0 0 454]  
 [ 0 0 510]]

PhysicalActivity\_Decision Tree:  
[[101 148 287]  
 [ 96 115 243]  
 [ 86 124 300]]

PhysicalActivity\_Random Forest:  
[[ 32 146 358]  
 [ 29 112 313]  
 [ 26 115 369]]

PhysicalActivity\_Gradient Boosting:  
[[162 198 176]  
 [146 151 157]  
 [149 168 193]]

PhysicalActivity\_KNN:  
[[244 162 130]  
 [221 147 86]  
 [209 169 132]]

PhysicalActivity\_MLP:  
[[165 146 225]  
 [118 123 213]  
 [130 136 244]]

PhysicalActivity\_SVC:  
[[ 0 2 534]  
 [ 0 1 453]  
 [ 0 0 510]]

PhysicalActivity\_XGBoost:  
[[148 208 180]  
 [142 155 157]  
 [142 177 191]]

StressLevel\_Logistic Regression:  
[[ 0 500 0]  
 [ 0 486 0]  
 [ 0 514 0]]

StressLevel\_Decision Tree:  
[[173 200 127]  
 [177 174 135]  
 [174 205 135]]

StressLevel\_Random Forest:  
[[152 202 146]  
 [120 202 164]  
 [129 225 160]]

StressLevel\_Gradient Boosting:  
[[167 173 160]  
 [147 183 156]  
 [161 180 173]]

StressLevel\_KNN:  
[[213 166 121]  
 [188 181 117]  
 [196 187 131]]

StressLevel\_MLP:  
[[167 185 148]  
 [150 192 144]  
 [171 184 159]]

StressLevel\_SVC:  
[[ 1 468 31]  
 [ 0 467 19]  
 [ 2 483 29]]

StressLevel\_XGBoost:  
[[159 174 167]  
 [144 187 155]  
 [164 190 160]]

HealthIns\_Logistic Regression:  
[[ 0 746]  
 [ 0 754]]

HealthIns\_Decision Tree:  
[[392 354]  
 [398 356]]

HealthIns\_Random Forest:  
[[360 386]  
 [353 401]]

HealthIns\_Gradient Boosting:  
[[365 381]  
 [364 390]]

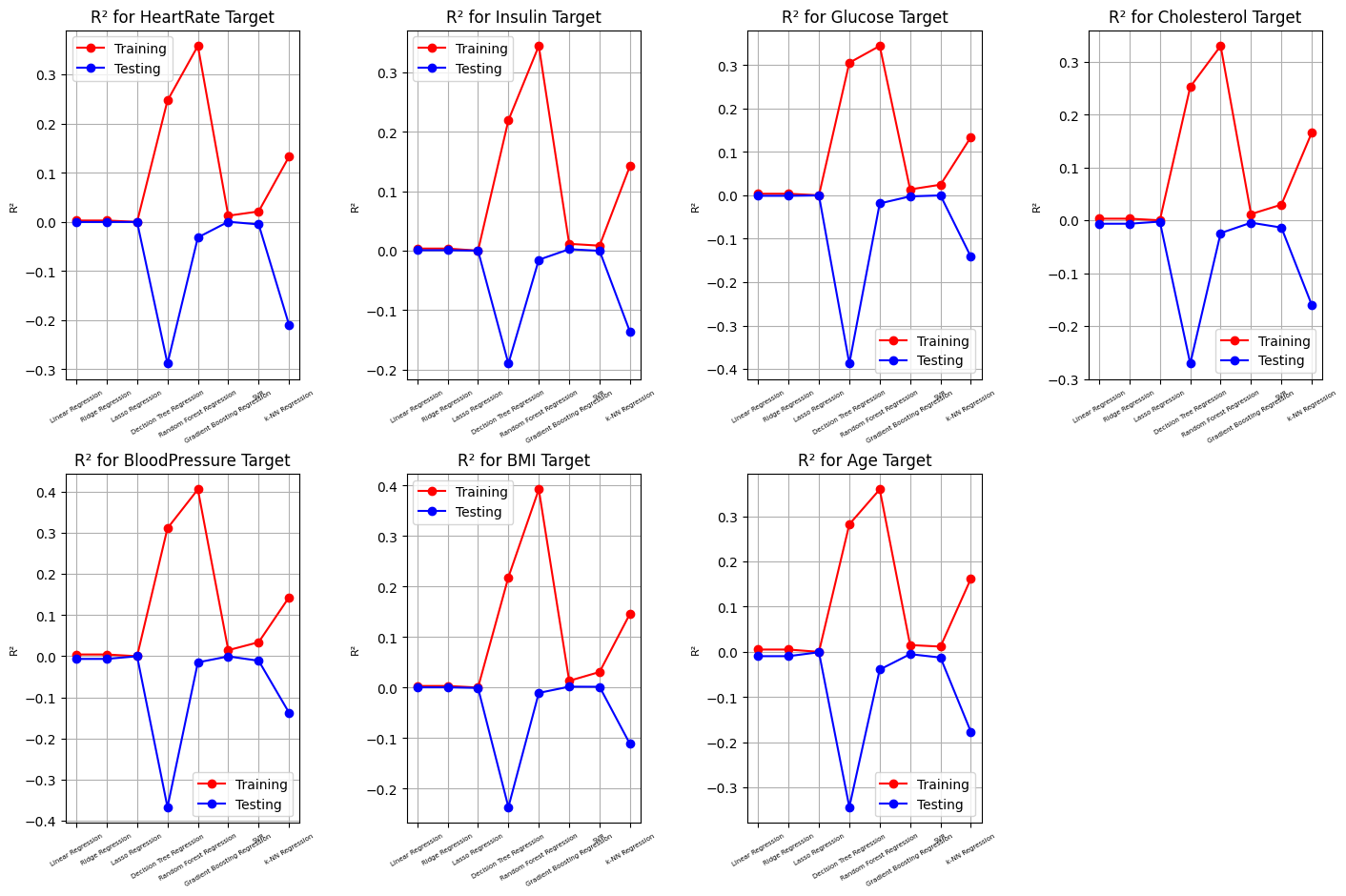
HealthIns\_KNN:  
[[366 380]  
 [365 389]]

HealthIns\_MLP:  
[[356 390]  
 [354 400]]

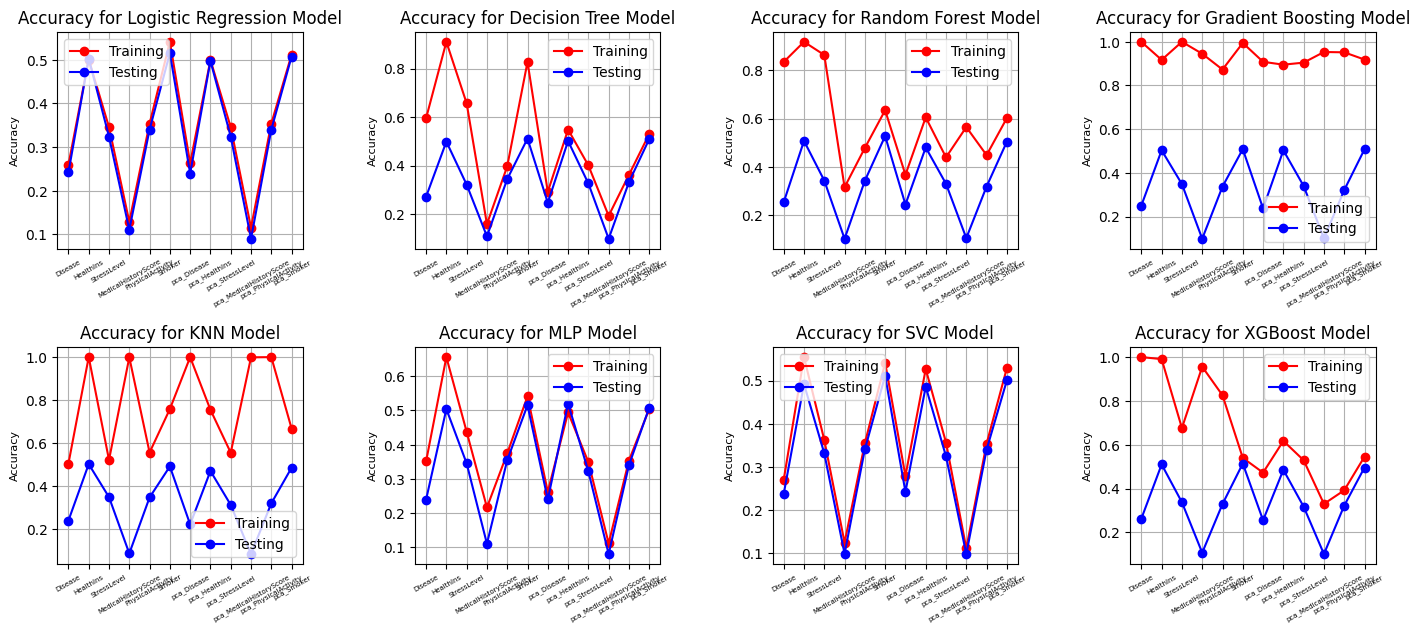
HealthIns\_SVC:  
[[269 477]  
 [285 469]]

HealthIns\_XGBoost:  
[[370 376]  
 [359 395]]

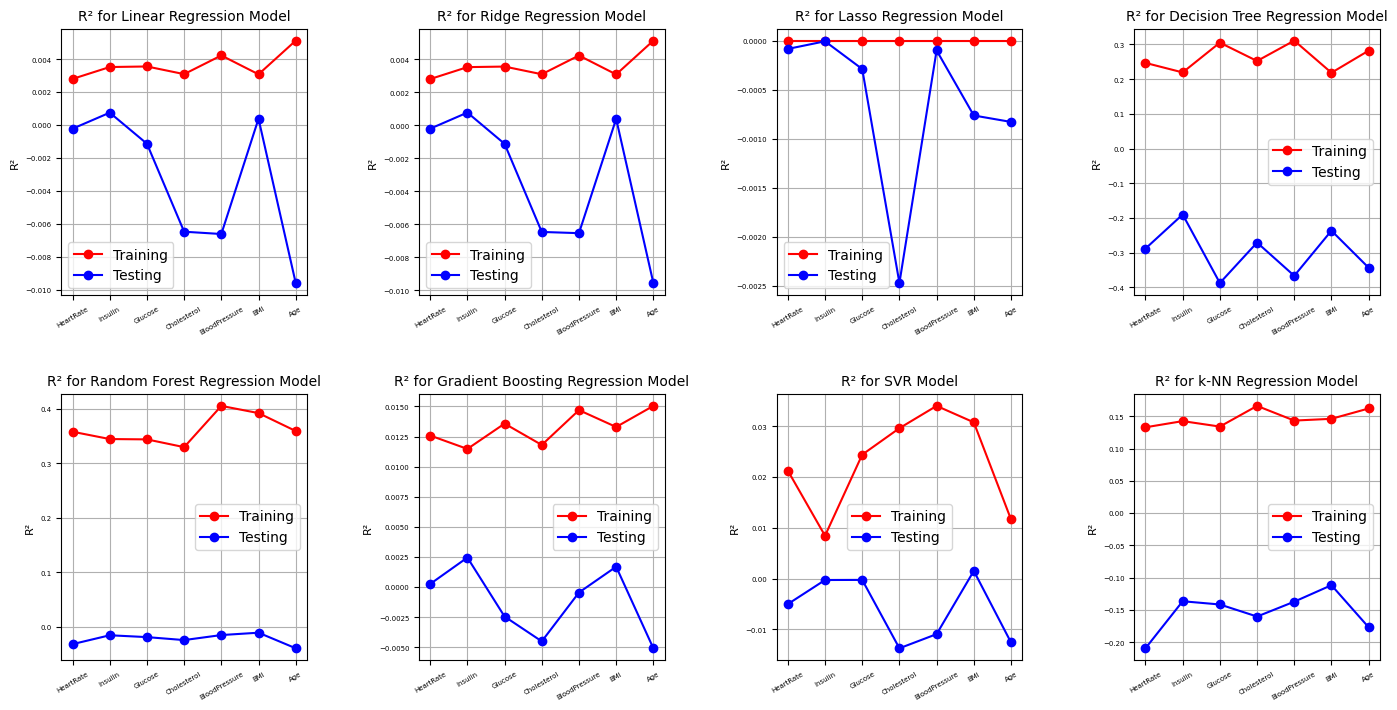
### Predictability by Target (Regression)



### Predictability by Model (Classification)



### Predictability by Model (Regression)



# Conclusion

Based on the dataset provided and the methods applied, we cannot firmly declare that the features in the dataset are predictive. While the classification models performed some accuracy scores, these scores are close to what would be expected using random chance, given the number of classes. This suggests that the features do not provide strong predictive power for the target feature.

There may be several reasons for this outcome:

* **Lack of domain knowledge**
* The dataset may contain information or signals which can be recognised easily by medical professionals. For example, some diseases may not affect a patient's physical condition as much as machine learning models can recognise it, but a domain expert can. In this case, an analyst should have some medical expertise or be advised during the analysis by experts.
* **Not enough records in the dataset**
* To get statistically approved results dataset must contain enough variations. However, in this data set numeric features are not distributed as it required. Additional records might complete distributions and make it easier for ML models to find differences between classes. Authors Vabalas A. et al. (2019) discovered that sample size is directly related to biased ML performance.
* **Not enough features in the dataset**
* Another reason is that the features themselves do not represent the target variable. For instance, if we assign as a target variable the "Disease" column, we can end up trying to predict health issues which can not be detected by using provided features. As an example, predicting mental health issues or a broken leg using blood pressure figures. There may be a correlation between the mentioned variables, but it can be detected by using a large number of records to find a difference. However, if we are not able to get enough records, we can get more features which may be more related to a target variable. For example, using the previous case of a broken leg, data about hobbies or occupations can be more helpful than blood tests. Authors Dahiwade D. et al. (2019) in their study suggested including data such as symptoms, living habits and previous checkups which may be essential for predicting disease.
* **Dataset does not represent population**
* As we mentioned before, we do not how data was collected, and how this data represents the population. If we assume that numeric features in the population are normally distributed we can conclude that the given dataset does not represent the population accurately, and does not contain enough data to predict the target variable. However, this statement should be tested using hypothesis analysis and some statistics from the population.
* **Not appropriate methodology**
* Another reason may be that we have not considered all methods which could be applied to this task. There is likely a method which more appropriate for the task and which can predict the target more accurately. To exclude this item we should test more models with different hyperparameters. For example, authors Uddin S. et al. (2019) compared the performance of 9 machine-learning models and concluded that for their dataset Random Forest and SVM are the most effective models to predict disease.
* **Not enough preparation tests**
* There may be thousands of different variations of how to prepare data. Each variation has its impact on each model's performance. So, testing different data preparation stages may improve the models' performance.
* **Not enough EDA**
* Similar to previous statements more detailed investigation of the dataset on the EDA stage may help to find the correlations or patterns which help to focus a further analysis.

# References

Bhaskaran, K. (2014). What is the difference between missing completely at random and missing at random? IJE 2014. 43(4): 1336-1339.

Choong A. and Lee N. (2017). Evaluation of convolutionary neural networks modeling of DNA sequences using ordinal versus one-hot encoding method. 2017 International Conference on Computer and Drone Applications (IConDA), 60-65.

Dahiwade D., Patle G. and Meshram E. (2019). Designing disease prediction model using machine learning approach. 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), 1211-1215.

Gillam R. (2003). Unicode demystified: a practical programmer's guide to the encoding standard. Addison-Wesley Professional.

Ha, T. N., Lubo-Robles, D., Marfurt, K., and Wallet, B. (2021). In-depth analysis of logarithmic data transformation and per-class normalization in machine learning. Interpretation, 9(3), T685-T710.

Kazerouni, A., Zhao, Q., Xie, J., Tata, S., and Najork, M. (2020). Active learning for skewed data sets. arXiv preprint arXiv:2005.11442.

Makaba, T., and Dogo, E. (2019). A comparison of strategies for missing values in data on machine learning classification algorithms. International multidisciplinary information technology and engineering conference (IMITEC), 1-7.

Monard, M. C. (2002). Learning with skewed class distributions. GEAPA Batista, 85, 173-180.

Palanivinayagam, A. (2023). Effective handling of missing values in datasets for classification using machine learning methods. Information, 14(2), 92.

Uddin S., Khan A., Hossain E. and Moni M. (2019). Comparing different supervised machine learning algorithms for disease prediction. BMC medical informatics and decision making 19 (1), 1-16.

Vabalas A., Gowen E., Poliakoff E. and Casson A. (2019). Machine learning algorithm validation with a limited sample size. PloS one 14 (11), e0224365.

#### Author: Ilia Grishkin

<https://github.com/CCT-Dublin/data-preparation-ca1-Ilia-Grishkin.git>

<https://github.com/Ilia-Grishkin/Machine_learning.git>

<https://github.com/Ilia-Grishkin/Data_Preparation.git>

<https://github.com/CCT-Dublin/integrated-ca2-dp-and-ml-Ilia-Grishkin.git>