Assessing the Risks of Heart Disease

Student 

2024-03-01

1. Introduction

The prevalence of heart disease is an enduring problem in countries across the globe. The World Heart Federation (2023) reports that cardiovascular disease is among the most common causes of death globally, taking 20.5 million lives in 2021 alone. Additionally, the overall spending on heart-related problems ranges between 7.1% and 21% of total health spending world-wide (Santos et al., 2020). Such figures show that understanding the main factors that contribute to the development of heart disease can help policymakers to both save lives and prevent future healthcare expenses. This report presents an attempt to shed light on the factors the most closely associated with heart disease. By utilising a dataset with data on health status of more than 200 thousand Americans, it answers two questions: (1) what are the main risk groups for heart disease? and (2) what are the factors that are the most closely associated with developing heart disease?. The rest of the report is structured as follows. Section 2 provides a brief overview of the health status dataset and the main transformations applied to it. Section 3 presents the methodology of the principal components analysis and gradient boosting machines which are deployed for the analysis. The main results are presented in Section 4. Finally, Section 5 concludes the report.

2. Data

The dataset was taken from Pytlak (2023), who originally built it from the 2022 annual telephone survey conducted in the US by the Center for Disease Control and Prevention (CDC, 2023). The file contains complete cross-sectional observations on 40 variables for 246 022 American citizens, with 34 categorical and 6 numeric variables. The numeric variables list information on characteristics such as height, weight, and BMI, together with several variables on health activity (e.g. the number of physical health days). The categorical variables include information like sex and ethnicity and a collection of health-related indicators (smoker status, having asthma/stroke/angina, and general health status, among others). For the present analysis, the key variable is `HadHeartAttack`, which is a dummy with two levels: "Yes" and "No".

Table 1: Summary statistics for the heart disease data

AgeCategory	Total	% observations	% male	% heart disease	Total heart disease
Age 18 to 24	13122	5.33	57	0.38	50
Age 25 to 29	11109	4.52	54	0.42	47
Age 30 to 34	13346	5.42	51	0.67	90
Age 35 to 39	15614	6.35	49	1.00	156
Age 40 to 44	16973	6.90	47	1.34	228
Age 45 to 49	16753	6.81	48	2.51	420
Age 50 to 54	19913	8.09	48	3.53	703
Age 55 to 59	22224	9.03	48	5.00	1112
Age 60 to 64	26720	10.86	47	5.89	1575
Age 65 to 69	28557	11.61	48	7.55	2155
Age 70 to 74	25739	10.46	48	9.36	2408
Age 75 to 79	18136	7.37	45	11.39	2065
Age 80 or older	17816	7.24	42	13.62	2426
Overall:	246022	100.00	48	5.00	13435

A brief summary of the key variables is presented in Table 1. As can be seen, the sample is unequally represented by different age groups, with the majority of respondents (13.6%) being aged 55 and above. However, the data appear to be relatively balanced in terms of gender with slight variations between age groups. Finally, only 5% of all the respondents reported experiencing heart attack, and the proportion is highly unequal between ages rising from 0.4% in the youngest group to 13.6% in the oldest group. Although these numbers appear to correspond with common sense, having a small fraction of people with heart disease in the dataset can pose additional modelling challenges, which will be discussed in Section 4.

Additional data manipulations included one-hot encoding of the categorical columns with more than two values such that each response got a dummy column with ones and zeros. The variables with two values were re-coded into a numeric dummy variable. These transformations expanded the number of columns to 71. The code for data-preprocessing routines can be found in Appendix A.

3. Method

This report relies on two methods to answer the proposed research questions. First, principal component analysis is used to condense the number of columns in the data to better visualise potential relations within the data and obtain preliminary insights on the groups of variables that are associated with heart disease. Then, a gradient boosting machine is used to assess the individual importance of each variable in predicting heart disease. This section serves to provide a technical overview of both methods.

Principal component analysis (PCA) is a dimensionality reduction technique that is often used for exploratory data analysis. The key idea behind it is to reduce the original number of dimensions (columns) in the dataset while preserving as much original variance as possible with potentially fewer columns, which is achieved by performing a linear transformation of original features based on correlations within them. The resultant columns (principal components) are ordered by the amount of variance captured by each of them. These components can be explored to assess which variables contribute the most to the new columns, which can provide additional insights into potential clusters within the data and facilitate their visualisation in fewer dimensions.

There are multiple ways in which PCA can be explained. Hastie et al. (2009) suggest starting with a set of observations $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, each a vector in R^{p} with elements centered at 0. The goal is to represent them with a linear model of rank q such that $q \leq p$:

$$f(\boldsymbol{\lambda}) = \boldsymbol{\mu} + \mathbf{V}_q \boldsymbol{\lambda},$$

where $\boldsymbol{\lambda}$ is a parameter vector of length q, $\boldsymbol{\mu}$ is a location vector of length p, and $\mathbf{V}_q$ is a $p \times q$ matrix whose columns are given by orthogonal unit vectors. The required rank-q representation of the data is achieved by fitting least squares through solving the minimisation problem:

$$\min_{\mu, \{\lambda_i\}, \mathbf{V}_q} \sum_{i=1}^{n} \left\| \boldsymbol{x}_i - \boldsymbol{\mu} - \mathbf{V}_q \boldsymbol{\lambda}_i \right\|^2.$$

In simple words, the key objective is to find a lower-dimensional representation of original data such that the distance between this representation and the initial data is minimal. With partial optimisation, it can be shown that $\hat{\boldsymbol{\mu}} = 0$ and $\hat{\boldsymbol{\lambda}}_i = \mathbf{V}_q^\top \boldsymbol{x}_i$, assuming that the mean of each column is 0.[1] The latter equation is equivalent to projecting the centred data $\boldsymbol{x}_i$ onto the subspace spanned by the columns of $\mathbf{V}_q$. The final step is then to find the orthogonal matrix $\mathbf{V}_q$ that solves the problem:

$$\min_{\mathbf{V}_q} \sum_{i=1}^{n} \left\| \boldsymbol{x}_i - \mathbf{V}_q \mathbf{V}_q^\top \boldsymbol{x}_i \right\|^2.$$

The solution can be obtained by performing the singular value decomposition of the $n \times p$ data matrix $\mathbf{X}$:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^\top,$$

where $\mathbf{U}$ is an $n \times p$ orthogonal matrix with columns $\boldsymbol{u}_j$ (meaning that the product of its transpose by itself is a $p \times p$ identity matrix), $\mathbf{V}$ is a $p \times p$ orthogonal matrix with columns $\boldsymbol{v}_j$. These column vectors represent left and right singular vectors of $\mathbf{X}$, accordingly. Finally, $\mathbf{D}$ is a $p \times p$ diagonal matrix whose diagonal elements d_i are called the singular values and have the property that $d_1 \geq d_2 \geq \cdots \geq d_p$. The principal components of $\mathbf{X}$ are then represented by the columns of the $n \times p$ matrix $\mathbf{UD}$. Returning back to the minimisation problem, the rank-q matrix $\mathbf{V}_q$ is obtained with the first q columns of $\mathbf{V}$, and the n vectors $\boldsymbol{\lambda}_i$ are obtained with the first q principal components $\mathbf{U}_q \mathbf{D}_q$.

Importantly, the singular values of $\mathbf{D}$ are directly related to the variance explained by each principal component (this result can be shown by performing a spectral decomposition of the covariance matrix of $\mathbf{X}$). Therefore, the principal components in $\mathbf{UD}$ are ordered by the amount of variance they explain in the original data. Hence, it follows that if $q = p$, they capture 100% of the original variance, which is given by the sum of diagonal elements of $\mathbf{D}$. However, the goal of PCA is to represent the data in a smaller-dimensional

[1] Note that the book uses formulas accounting for cases where columns are not demeaned. Here, it was decided to drop subtraction of means for ease of notation.

space while retaining as much of its variance as possible. Therefore, one can select the number of components $q < p$ such that the reduced data capture a sufficient amount of total variance, and there exist multiple ways to select the optimal q (Jackson, 1993). However, in this analysis PCA is not used as the main analytical tool, and the task is not to choose the optimal reconstruction matrix but to see whether the first several components can be used to meaningfully visualise data and spot potential cluster structures within them. Hence, the components are used only at the visualsiation stage, and the accuracy of the visualisation is assessed based on the amount of variance explained by each of the components employed.

Another important property of PCA is that the columns of $\mathbf{V}$ represent vectors of variable loadings for each column of $\mathbf{X}$ with respect to the corresponding principal component. This allows to see how much each variable contributes to a given component and can be used to give semantic interpretation to the obtained results. This property is useful in combination with visualisation of PCA, as if any clusters are spotted in the graph, they can be interpreted in terms of the variables that have the greatest component loadings.

As a second tool, this report utilises gradient boosting machines (GBM). Unlike PCA, it is a supervised method that is used for prediction based on the available data. GBM is suitable for prediction of both continuous and categorical variables, but the subsequent explanation focuses only on the case of binary classification. In essence, GBM builds on the idea of classical boosting, which uses a series of "weak learners" that individually predict the outcome variable slightly better than random guessing. In each iteration, the algorithm employs simple learners like 1-stump decision trees to predict the target variable, assigning greater importance to previously misclassified observations. This iterative process is repeated a specified number of times, and the final predictions are given by a weighted average vote of all learners, where each learner has a different amount of "say" based on how accurately they performed in predicting the given observation. Gradient boosting goes one step further by introducing a loss function that quantifies the difference between predicted and actual values. The algorithm optimises its predictions by minimising the gradient of this loss function, adjusting the weak learners accordingly and enhancing the quality of subsequent predictions.

Mathematically, boosting can be explained using the Adaboost.M1 algorithm (Freund & Schapire, 1997). Begin with a two-class problem with a vector of n outcome variables $\boldsymbol{y}$ that can take one of two values: $\{-1, 1\}$. Further suppose that the $n \times p$ matrix $\mathbf{X}$ contains vectors of predictor variables for each observation i as its rows. The algorithm presupposes fitting weak classifiers $G_m(\mathbf{X}), m = 1, 2 \ldots, M$ with updating weights of each observation i based on the accuracy of the previous prediction. The weak classifier can be a shallow tree with 1-3 partitions. At each iteration, we evaluate the classifier's error rate with

$$err_m = \frac{\sum_{i=1}^{n} w_i I(y_i \neq G_m(\boldsymbol{x}_i))}{\sum_{i=1}^{n} w_i},$$

where $\boldsymbol{x}_i$ is the ith row of $\mathbf{X}$, I is an indicator function that equals 1 if the prediction does not match the true value of y_i, and w_i is the weight of observation i. In the first iteration, the weights are set to be equal for all observations. For every iteration m, the classifier is adjusted to assign greater weights to previously misclassified observations. With $G_m(\mathbf{X})$, AdaBoost.M1 calculates the weight of the mth classifier in the final classifier $G(\mathbf{X})$ and uses it in an exponential loss function to update the weights of each observation that are passed to the next iteration. The final classification is obtained by taking the sign of the weighted sum of predictions for all m classifiers. The technical details behind AdaBoost.M1 are not essential for understanding GBM, but the reader is invited to consult the original paper by Freund & Shapire (ibid) for further explanations.

Gradient boosting is a technique that builds on the principles of boosting but focuses on adjusting a user-specified loss function, rather than weights, to enhance classification accuracy. Similar to traditional boosting, the model is iteratively adjusted by fitting weak learners, such as 1-stump decision trees. However, in each iteration, a new tree is fitted to the residual errors of the previous model (Hastie et al., 2009). The direction in which the algorithm refines accuracy is defined by the loss function $L(y_i, f(\boldsymbol{x}_i))$, where y_i is the true value and $f(\boldsymbol{x}_i)$ is the prediction of the model. The specific form of the loss function varies and serves as a parameter to tune. Nevertheless, the optimisation routine is the same for all loss functions. At each iteration m, the negative gradient r_{im} of the loss function is calculated using the following equation:

$$r_{im} = -\left[\frac{\partial L(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}\right]_{f=f_{m-1}}.$$

This negative gradient indicates the direction in which the model needs adjustment to minimise the error. Notably, the current solution for the gradient is evaluated using values obtained from the previous learner, f_{m-1}. The next step involves fitting a classification tree to each of the obtained gradient values. This tree is denoted as $T(\boldsymbol{x}_i; \Theta_m)$, where Θ_m is a vector of parameters used in constructing the mth tree to make its predictions as close as possible to the negative gradient. The tree divides the input space into regions R_{jm}. The goal is to find a value γ_{jm} that represents the modal class of observations falling in the jth region obtained at the mth iteration. This constant is found by minimising the following expression:

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{\boldsymbol{x}_i \in R_{jm}} L\left(y_i, f_{m-1}\left(\boldsymbol{x}_i\right) + \gamma_{jm}\right).$$

Here, $\hat{\gamma}_{jm}$ is the value that, when added to the previous model's predictions, minimises the overall loss within the j-th region. The model is then updated as follows:

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I\left(x \in R_{jm}\right),$$

where J_m is the number of terminal nodes in the tree. The final predictions of the algorithm, $\hat{f}(x)$, are obtained with the model estimated in the last iteration. In binary classification, the output of the algorithm is a vector of class probabilities, and the classification is determined using an arbitrarily chosen cutoff point.

One problem of GBM is that it takes multiple parameters that need to be tuned. Apart from the number of iterations M, one can also introduce a penalty to the loss function which slows down the convergence time but makes it less likely that the algorithm "jumps over" the most optimal parameters. Further, to prevent overfitting, there exists an extension to GBM in the form of stochastic gradient boosting, for which at each iteration, only a proportion of training data is used for estimating the update, with this proportion being a parameter to tune. Finally, the choice of the loss function can also play an important role for the accuracy of the final model. Taken together, these parameters can significantly complicate the search for the most optimal specification and require a computationally efficient searching strategy.

Usually, GBM is trained on a subsample of data, and its performance is evaluated on the quality of predictions for the data not used for training. To assess the classification performance of the algorithm, one can rely on conventional metrics such as the accuracy rate (the percentage of correctly predicted classes), the true positives/negatives rate (fraction of correctly predicted positive/negative classes), the false positives/negatives rate (fraction of incorrectly predicted positive/negative classes), and their derivatives such as precision ($\frac{TP}{TP+FP}$) and recall ($\frac{TP}{TP+FN}$). The relative importance of these metrics is defined by the task at hand and can be used as guidance for further model tuning, as will be demonstrated in the next section.

Finally, one advantage of GBM is that it allows to assess the relative importance of variables in the estimated model. Breiman (2001) suggested a method that assesses the variable's importance based on the aggregated degree of improvement obtained at the nodes where it was used for splitting compared to predictions made with a constant fit.[2] This allows to see which variables contribute the most to the algorithm's accuracy and can provide further insights into potential relations within the data, although the interpretation remains rather limited since the method does not indicate the direction of the relation, nor does it imply the existence of a causal link between the predicted variable and its predictor.

4. Results

Before running PCA on the health data, the variables `HadHeartAttack` and `state` were removed from the dataset. The rest of the variables were scaled to have mean 0 and standard deviation of 1 so that the variables measured on a different scale and exercising greater variance (such as height and weight) did not distort the results of PCA. However, the performance of PCA in terms of dimensionality reduction can be at most assessed as modest. Figure 1B in Appendix B shows that the first principal component explains 3.42% of total variance in the data, and the values are even smaller for each subsequent component. Despite this, a visualisation of these results still allows us to see some meaningful patterns in the data. Panel (a) of Figure 1 shows a visualisation of a random subsample of the data based on the scores of the first two

[2] The details of the method are rather technical, and the reader is invited to consult Breiman's paper for a detailed explanation.

principal components,[3] coloured by whether the person has had a heart disease. As can be seen, people with no heart attack tend to cluster around smaller values of the first component, and the units with heart attack are concentrated around higher values of this component. However, the image is less clear for the second component. To add more clarity, Panel (b) of Figure 1 visualises all the 13 435 people with heart disease and shows that indeed, the cases seem to cluster around larger values of PC1. Additionally, one can see some signs of clustering around the second component, although these signs are not as clear as for the first one.

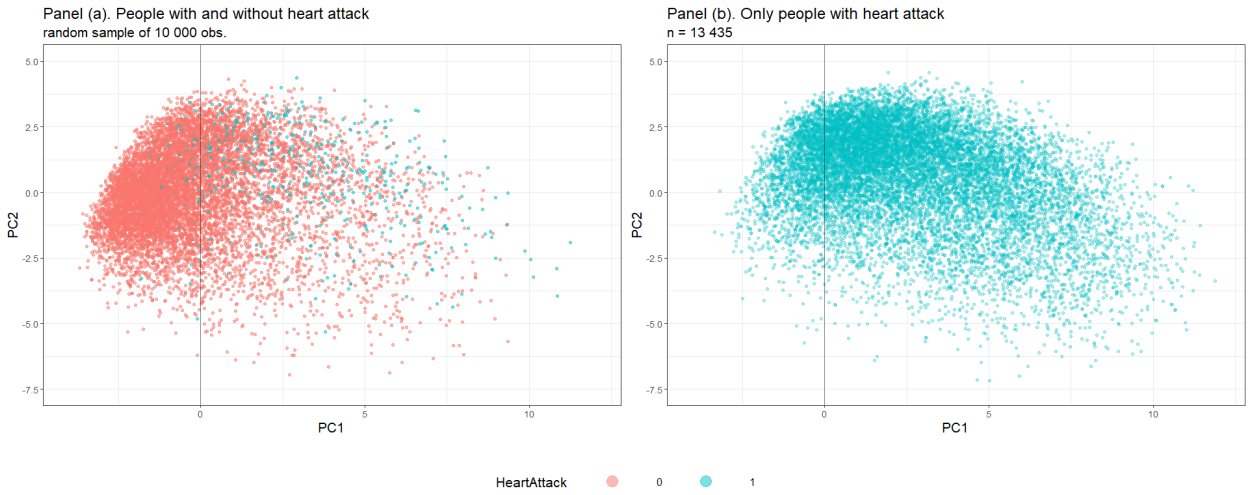

Figure 1: PCA results (first two components)

To provide further insights into the obtained components, Table 2 shows 5 variables with the largest (top 5) and smallest, or most negative, loadings (bottom 5) for the first two principal components. The table indicates that the first component can be interpreted as "general health", with the variables associated with a healthy lifestyle (such as undertaking physical activities or not smoking) contributing the least to this component while the variables associated with poor health have the largest contributions (e.g., presence of arthritis and `PhysicalHealthDays`, which indicates the number of days spent out of activity due to health issues). Again, such a clear distinction cannot be made for the second component, whose composition does not appear to be easily interpretable. The interpretation of the first component, however, needs to be seen as rather approximate, as it explains only 3.4% of total variance in the dataset.

Table 2: Variable loadings for the first 2 components

Variable name (PC1)	Loading	Variable name (PC2)	Loading
DifficultyWalking	0.298	PneumoVaxEver	0.171
PhysicalHealthDays	0.271	FluVaxLast12	0.071
HadArthritis	0.227	RaceEthnicityCategoryWhite only, Non-Hispanic	0.013
DifficultyErrands	0.223	Checkup within past year	0.122
GeneralHealthPoor	0.205	AgeCategoryAge 80 or older	0.091
GeneralHealthVery good	-0.139	MentalHealthDays	0.142
GeneralHealthExcellent	-0.140	DifficultyConcentrating	0.167
SmokerStatusNever smoked	-0.147	HIVTesting	0.009
PhysicalActivities	-0.180	HadDepressiveDisorder	0.140
RemovedTeethNone of them	-0.200	RaceEthnicityCategoryHispanic	-0.019

Finally, the variable `HadHeartAttack` was predicted using GBM. As mentioned in Section 2, the dataset is highly imbalanced with respect to the predicted variable, with only 5% of the respondents having a history of heart attack. Without considering this, GBM achieves 95% accuracy by classifying all predictions as "no heart attack" but shows extremely poor results with respect to recall (due to a large number of false

[3]Sumbsampling was used to avoid cluttering of the graph. However, using the whole sample produces a very similar image.

negatives). Hence, it was decided to perform basic undersampling (Mohammed et al., 2020) by modifying the data such that all the observations from the minority class remained, and the same number of observations was randomly sampled from the majority class. This yielded a dataset where both classes were equally represented but implied that in the current context, the model used only 10% of the original data.

After obtaining a balanced sample with 26 870 observations, 80% of them were randomly allocated to a training set and the remaining 20% were reserved for a test set. Then, optimal values for the shrinkage parameter (0.1) , number of iterations (150), and depth of trees (3) were obtained using the R `caret` package (Kuhn, 2008). The parameters for the fraction of observations used for training in each iteration (0.5) and the loss function (Bernoulli) were found through manual iteration given the values of other parameters found with `caret`. Finally, the classification cutoff point was set at the conventional value of 0.5. The final model was estimated with the `gbm` package (Greenwell et. al, 2022) and achieved overall accuracy of around 80% in predicting whether the person has had heart attack both with training and test data. More importantly, the model performed relatively well with respect to recall achieving ≈ 0.78 with both sets, indicating that the model correctly predicted the class for people with heart attack in 78% of the cases. Given these results, it is now possible to see which variables GBM sees as contributing the most to heart attack. Figure 2 shows a bar plot with the 10 most important variables. As can be seen, these variables differ substantially from those we observed with PCA. Having angina is by far the most influential predictor, followed by having a chest scan and stroke. It also can be seen that many other important variables are closely related to health (being aged above 80, having diabetes/arthritis), as well as the person's sex.

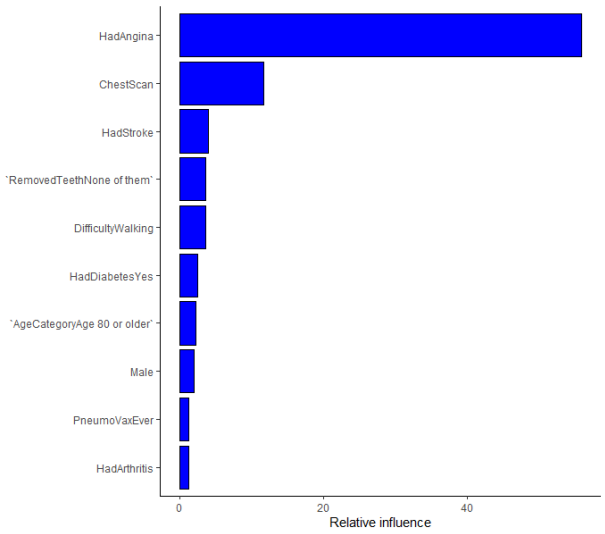

Figure 2: Variable importance (GBM)

However, the interpretation of these results remains limited. First, from the importance measure one cannot see how exactly the variables are related to heart disease. For example, it is not clear whether men are more or less likely to have one, nor is it obvious how having a vaccine against pneumonia is associated with heart attack. Second, GBM does not allow for making inference on causal relations within variables. Figure 2 shows that angina is strongly associated with heart disease, but one does not know which of the two diseases precedes, and the same logic applies to stroke. The high importance of chest scan can also be interpreted in two ways: people with heart disease are more likely to have a chest scan or, alternatively, those who have a chest scan are aware of potential heart issues and undertake timely actions to prevent them. Such complications limit the applicability of the obtained results for explaining and predicting heart disease but can indicate potentially meaningful correlations for further investigation by medical professionals.

5. Discussion and Conclusions

This report employed PCA and GBM to assess the factors that are the most closely related to heart disease. Overall, the obtained results can be summarised as follows. With PCA, one could see that factors associated with healthy lifestyle (such as physical activity) are the least associated with the risk of heart disease, but this relation explains only a minor fraction of this risk. GBM presented a different narrative showing that heart disease is closely related to other conditions such as angina, stroke, and diabetes, but could not provide insights into how these diseases are interconnected, making the results rather inconclusive. Nevertheless, both of these results can guide further research that will rely on better data and more elaborated methods to address the proposed question. Therefore, it remains to be researched whether stronger clusters of factors can be found to better narrow down the groups that are the most likely to suffer from heart illness. An additional challenge for further works lies in addressing the overall unbalancedness of health data, ensuring that the employed algorithms correctly predict and explain heart disease as a minority class in large datasets.

References

Breiman, L. (2001). "Random Forests." *Machine Learning*, 45, 5-32. https://doi.org/10.1023/A:1010933404324

CDC, Centers for Disease Control and Prevention. (2023). "Know Your Risk for Heart Disease." Available online at: https://www.cdc.gov/heartdisease/risk_factors.htm. [Accessed: Feb 28, 2024].

Greenwell, B., Boehmke, B., and Cunningham, J. (2022). "GBM: Generalized Boosted Regression Models." R package version 2.1.8.1, https://CRAN.R-project.org/package=gbm.

Freund, Y. and Schapire, R. E. (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting." *Journal of Computer and System Sciences*, 55(1), 119-139. https://doi.org/10.1006/jcss.1997.1504.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). "The Elements of Statistical Learning. Data Mining, Inference, and Prediction (2nd edition)." *Springer Series in Statistics*, New York, USA.

Jackson, D. A. (1993). "Stopping Rules in Principal Components Analysis: A Comparison of Heuristical and Statistical Approaches". *Ecology*, 74(8), 2204–2214. https://doi.org/10.2307/1939574

Kuhn, M. (2008). "Building Predictive Models in R Using the caret Package." *Journal of Statistical Software*, 28(5), 1–26. https://doi.org/10.18637/jss.v028.i05

Mohammed, M., Rawashdeh, J., and Abdullah, M. (2020). "Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results." *11th International Conference on Information and Communication Systems (ICICS)*, 243-248. https://doi.org/10.1109/ICICS49469.2020.239556

Pytlak, K. (2023). "Indicators of Heart Disease (2022 UPDATE)". Dataset. Available online at: https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease/data. [Accessed: Feb 28, 2024].

Santos, J. V., Vandenberghe, D., Lobo, M., and Freitas, A. (2020). "Cost of cardiovascular disease prevention: towards economic evaluations in prevention programs." *Annals of Translational Medicine*, 8(7), 512. https://doi.org/10.21037/atm.2020.01.20

World Health Federation. (2023). "Deaths from cardiovascular disease surged 60% globally over the last 30 years: report". Available online at: https://world-heart-federation.org/news/deaths-from-cardiovascular-disease-surged-60-globally-over-the-last-30-years-report. [Accessed: Feb 28, 2024].

Code used for the report

Packages

```
library(tidyverse)
library(ggpubr)
library(cluster)
library(mltools)
library(data.table)
library(caret)
library(gbm)
```

PCA

```
PCA_data<- heart_data %>%
  select(-State, -HadHeartAttack) %>% #Drop the predictor and state
  scale(.)

PCA_out<- prcomp(as.matrix(PCA_data))

# Calculate variance and cumulative variance
PCA_variances<- PCA_out$sdev
total_variance<- sum(PCA_variances)
percentage_explained<- (PCA_variances/total_variance)*100
cumsum(percentage_explained)
```

Figure 1

```
set.seed(1234)

text_size<- theme(text = element_text(size = rel(4)),
                  plot.title = element_text(size = 18))

legend_size<- theme(legend.key.size = unit(3, 'cm'),
      legend.text = element_text(size=12),
      legend.title = element_text(size=16))

panel_a<-as.data.frame(PCA_out$x) %>%
  mutate(HeartAttack=as.factor(heart_data$HadHeartAttack)) %>%
  sample_n(10000) %>% # Sample to avoid cluttering the plot
  ggplot(., aes(x=PC1, y=PC2, colour=HeartAttack)) +
  geom_point(alpha=0.5) +
  theme_bw() +
  xlim(c(-4, 12)) +
  ylim(c(-7.5, 5)) +
  geom_vline(xintercept=0,
             colour="black",
             alpha=0.5) +
  text_size +
  legend_size +
  guides(colour = guide_legend(override.aes = list(size=6))) +
```

```
  labs(title="Panel (a). People with and without heart attack",
       subtitle="random sample of 10 000 obs.")

panel_b<- as.data.frame(PCA_out$x) %>%
  mutate(HeartAttack=as.factor(heart_data$HadHeartAttack)) %>%
  filter(HeartAttack==1) %>%
  ggplot(., aes(x=PC1, y=PC2)) +
  geom_point(alpha=0.3,
             colour='#00BFC4') +
  theme_bw() +
  xlim(c(-4, 12)) +
  ylim(c(-7.5, 5))+
  geom_vline(xintercept=0,
             colour="black",
             alpha=0.5) +
  text_size +
  labs(title="Panel (b). Only people with heart attack", subtitle="n = 13 435")


figure_1<-ggarrange(panel_a, panel_b, nrow=1, common.legend=TRUE, legend="bottom")
```

GBM - undersampling and splitting

```
set.seed(1998)

# PERFORM UNDER-SAMPLING
heart_data<-data_copy
have_HA<- heart_data %>% filter(HadHeartAttack==1)
no_HA_sampled<- heart_data %>%
  filter(HadHeartAttack==0) %>%
  sample_n(nrow(have_HA))

heart_data<- rbind(have_HA, no_HA_sampled)

# Drop State and re-factor the predicted variable
heart_data$State <- NULL
heart_data$HadHeartAttack<- as.factor(heart_data$HadHeartAttack)

#Set 80% for training, 20% for test
train_index<- sample(c(1:nrow(heart_data)),
                     size =round(nrow(heart_data)*0.8,),
                     replace=FALSE)

heart_train<- heart_data[train_index, ]
heart_test<- heart_data[-train_index, ]
```

GBM - tuning

```
set.seed(1998)

fitControl <- trainControl(method = "repeatedcv",
```

```
                           number = 5,
                           repeats = 10)

gbmFit1 <- train(HadHeartAttack ~ ., data = heart_train,
                 method = "gbm",
                 trControl = fitControl,
                 weights=weights_vec,
                 verbose = FALSE)

# Optimal values:

# Num trees - 150, interaction.depth=3, shrinkage = 0.1
```

GBM - final model

```
set.seed(1998)

heart_train$HadHeartAttack<- as.double(heart_train$HadHeartAttack)-1
final_gbm <- gbm(HadHeartAttack ~ ., data = heart_train,
                 n.trees = 150,
                 shrinkage = 0.1,
                 interaction.depth = 3,
                 distribution = "bernoulli",
                 bag.fraction=0.5,
                 verbose=FALSE)

#NOTE: the model was trained on 27 combinations:
# with bag.fraction = {0.1, 0.9} and distribution
# {bernoulli, huberized, adaboost}

#The one specified here achieved the higherst training accuracy
```

GBM - prediction and assessment

```
# Training performance
train_pred <- predict(final_gbm,
                          newdata = heart_train,
                          n.trees = 150,
                            type = "response")
#Re-classify predictions
train_pred<- ifelse(train_pred<0.5, 0, 1)
unique(train_pred)

train_confusion<-confusionMatrix(as.factor(train_pred),
                                 as.factor(heart_train$HadHeartAttack),
                                 mode = "everything",
                                 positive="1")

## Predict for all ensemble sizes
heart_test$HadHeartAttack<- as.double(heart_test$HadHeartAttack)-1
pred_final_gbm <- predict(final_gbm,
```

```
                          newdata = heart_test,
                          n.trees = 150,
                            type = "response")
#Re-classify predictions
pred_final_gbm<- ifelse(pred_final_gbm<0.5, 0, 1)
unique(pred_final_gbm)

# Summary of results: Confusion matrix
gbm_confusion<-confusionMatrix(as.factor(pred_final_gbm),
                               as.factor(heart_test$HadHeartAttack),
                               mode = "everything",
                               positive="1")
```

Figure 2

```
# Variable importance
gbm_summary<- summary.gbm(final_gbm)
importance_vec<-gbm_summary$rel.inf
var_vec<- gbm_summary$var

importance_table<- data.frame(Variable=var_vec, Importance=importance_vec)

importance_graph<- importance_table %>%
  head(10) %>%
  ggplot(aes(x=reorder(Variable, Importance), y=Importance)) +
  geom_bar(stat="identity", fill="blue", colour="black") +
  theme_classic() + coord_flip() +
  labs(y="Relative influence", x=" ")

importance_table %>%
  arrange(desc(Importance)) %>%
  head(10)
```

Table 2

```
pc1_head<-as.data.frame(PCA_loadings) %>%
  arrange(desc(PC1)) %>%
  head(5) %>%
  round(., 3) %>%
  select(PC1)

pc1_tail<-as.data.frame(PCA_loadings) %>%
  arrange(desc(PC1)) %>%
  tail(5) %>%
  round(., 3) %>%
  select(PC1)

comp_1<- rbind(pc1_head, pc1_tail)

pc2_head<-as.data.frame(PCA_loadings) %>%
  arrange(desc(PC2)) %>%
  head(5) %>%
```

```
  round(., 3) %>%
  select(PC1)

pc2_tail<-as.data.frame(PCA_loadings) %>%
  arrange((PC2)) %>%
  head(5) %>%
  round(., 3) %>%
  select(PC1)

comp_2<- rbind(pc2_head, pc2_tail)

comps_loadings<-data.frame(PC_1_var=rownames(comp_1),
                           PC_1_value=comp_1[,1],
                           PC_2_var= rownames(comp_2),
                           PC_2_value=comp_2[,1])

colnames(comps_loadings)<- c("Variable name (PC1)",
                             "Loading",
                             "Variable name (PC2)",
                             "Loading")

comps_loadings[4, 3]<- "Checkup within past year"
knitr::kable(comps_loadings,
             caption="Variable loadings for the first 2 components")
```

Appendix A. Data pre-processing

```r
heart_data<-read.csv("heart_data_cleaned.csv")

heart_data<- read.csv("2022/heart_2022_no_nans.csv")

heart_data$Male<- ifelse(heart_data$Sex=="Male", 1, 0)

heart_data$Sex<- NULL

categorical_variables<- c("GeneralHealth", "LastCheckupTime",
                          "RemovedTeeth", "HadDiabetes",
                          "SmokerStatus", "ECigaretteUsage",
                          "RaceEthnicityCategory", "AgeCategory",
                          "TetanusLast10Tdap", "CovidPos")
categorical_columns<- heart_data[, categorical_variables]

# One-hot encode categorical variables
dummified_columns<- as.data.frame(
  model.matrix(~.-1,
               data=categorical_columns))

heart_data[, categorical_variables]<- NULL

for (i in c(2:ncol(heart_data))){
  if (is.character(heart_data[, i]) & length(unique(heart_data[, i]))==2){
  heart_data[, i]<- as.double(case_when(heart_data[, i]=="Yes" ~ "1",
                              heart_data[, i]=="No"~"0",
                              TRUE~heart_data[, i]))
  }
}

heart_data<- cbind(heart_data, dummified_columns) %>% na.omit()
```

Appendix B: Variance explained with PCA

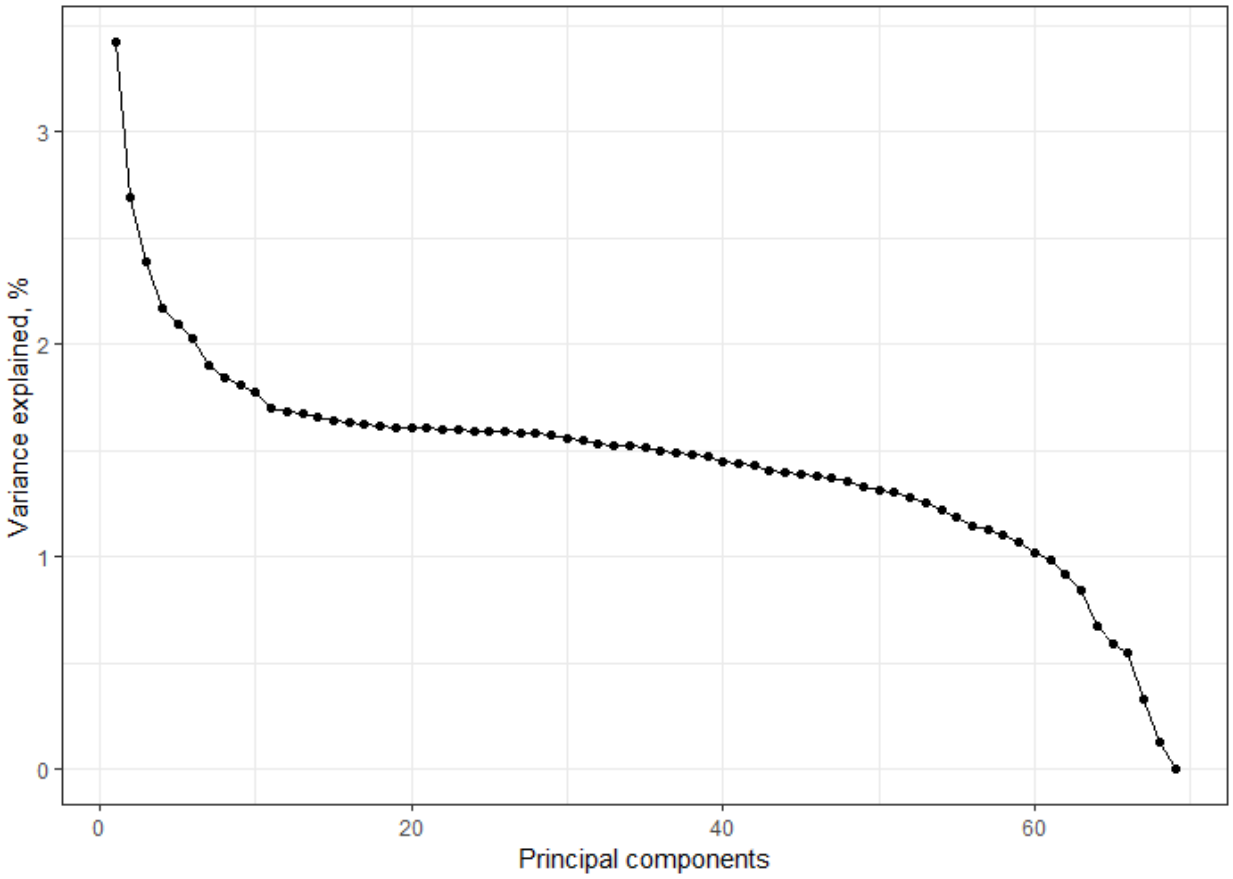

Figure 1B. Variance explained by principal components

Implementation of Hierarchical Clustering

The algorithm

For this assignment, I did a manual implementation of hierarchical clustering. The function `hierarchical_clustering()` (available as a separate R file and duplicated at the end of the report) takes 3 arguments: `dist_mat` - a matrix of (Euclidean) distances; `linkage_method` - the linkage method (average, complete, or single); and `verbose`, which specifies whether the clustering progess should be printed in real time.

The function exploits the fact that all the required distances are already present in the distance matrix, and because of this it does not re-calculate the distance matrix at each iteration, which largely economises on computation time. Additionally, it was ensured that the function cuts the number of loops to a minimum. Since the distance matrix is symmetric, duplicate calculations are excluded, and all the computations inside the loop are performed using `R`'s vectorised functions (e.g., `apply()`). Together, these modelling steps ensure fast convergence of the algorithm (see the note at the end of the report).

Testing

To test the algorithm's performance, I used the `cityWeather` dataset and randomly sampled 10 observations from it. Then, a comparison was made to the `hclust()` function from the `cluster` package. Table 3 reports the distance measures that were used at each merging stage by my function (labelled as "Own") and the package for each of the implemented linkage methods. As can be seen, the results are identical.

```
source("Hierarchical Clustering - FULLY FUNCTIONAL.R")
load("cityweather.RData")
set.seed(1234)
test_data<- cityweather %>% sample_n(.,10)
test_data_raw<-test_data
test_data<-as.matrix(dist(test_data_raw))
```

```
# Test================================================================================
#Average
own_res_avg<-hierarchical_clustering(dist_mat=test_data, linkage_method="average")
package_res_avg<-hclust(dist(test_data_raw), method="average")
own_avg_3_clusters<-own_res_avg$num_clust[[7]]
pcg_avg_3_clusters<- cutree(package_res_avg, k=3)

#Complete
own_res_cmp<-hierarchical_clustering(dist_mat=test_data, linkage_method="complete")
package_res_cmp<-hclust(dist(test_data_raw), method="complete")
own_cmp_3_clusters<-own_res_cmp$num_clust[[7]]
pcg_cmp_3_clusters<- cutree(package_res_cmp, k=3)

#Single
own_res_sgl<-hierarchical_clustering(dist_mat=test_data, linkage_method="single")
package_res_sgl<-hclust(dist(test_data_raw), method="single")
own_sgl_3_clusters<-own_res_sgl$num_clust[[7]]
pcg_sgl_3_clusters<- cutree(package_res_sgl, k=3)
```

Table 3: Comparison of clustering heights (own vs. package)

Own (average)	Package (average)	Own (complete)	Package (complete)	Own (single)	Package (single)
35.63	35.63	35.63	35.63	35.63	35.63
40.61	40.61	40.61	40.61	40.61	40.61
68.69	68.69	68.69	68.69	49.06	49.06
68.88	68.88	80.90	80.90	56.86	56.86
68.91	68.91	88.76	88.76	68.69	68.69
122.13	122.13	125.72	125.72	107.09	107.09
152.71	152.71	194.69	194.69	113.64	113.64
278.24	278.24	427.53	427.53	116.49	116.49
742.02	742.02	975.01	975.01	550.47	550.47

Further, Table 4 shows the clusters that are obtained if the functions are required to return vectors of clusters with 3 clusters in each. Although the numbers differ due to the function's design, the patterns are identical.

```
cluster_table<- cbind(own_avg_3_clusters, pcg_avg_3_clusters,
                      own_cmp_3_clusters, pcg_cmp_3_clusters,
                      own_sgl_3_clusters, pcg_sgl_3_clusters)
colnames(cluster_table)<- c("Own (average)", "Package (average)",
                            "Own (complete)", "Package (complete)",
                            "Own (single)", "Package (single)")
```

Table 4: Comparison of final clusters (own vs. package), k=3

Own (average)	Package (average)	Own (complete)	Package (complete)	Own (single)	Package (single)
1	1	1	1	1	1
2	2	2	2	1	1
1	1	1	1	1	1
2	2	2	2	4	2
2	2	2	2	1	1
1	1	1	1	1	1
7	3	7	3	7	3
1	1	1	1	1	1
1	1	1	1	1	1
2	2	2	2	1	1

Note on performance

The function shows relatively fast performance. With 10 observations, clustering is done in almost no time; with 50 observations, the function produces the results in under 5 seconds, and with 100 observations, the output is returned in around 15 seconds. An interesting lesson to learn: matrices are much less computationally expensive than dataframes (the same function with dataframes took about 10 minutes to compute clustering for 100 observations).

Manual implementation - code

```
hierarchical_clustering<- function(dist_mat,
                                   linkage_method="average",
                                   verbose=FALSE){
  #'@description
  #'Performs hierarchical clustering based on a given linkage method
  #'@param dist_mat a symmetric matrix of (Euclidean) distances
  #'@param lankinage_method, character. One of c("complete", "average", "single")

  #'@param verbose, boolean. Should progress be printed in the console?
  #'@return out. List with 3 elements: num_list - vector of class assignments,
  #'                                   partitions - the clusers merged at each step
  #'                                   h - the distance value used at the mergind stage


  #To avoid mistaking identical observations for those within clusters
  diag(dist_mat)<- NA
  rownames(dist_mat)<- as.character(c(1:nrow(dist_mat)))

  # Initialise the encoder function
  clusters<- c(1:nrow(dist_mat))

  # Add cluster labels
  dist_mat<- cbind(dist_mat, clusters)

  # Initialise j
  j <- nrow(dist_mat)

  count<-1

  # Prepare storage for results
  partitions<-data.frame(Step=c(1:(j-1)),
                         Merged_1 = c(1:(j-1)),
                         Merged_2 = c(1:(j-1)))

  clusters_list<-list()
  decisions<- c()
  while(j>1){
    j<- j-1

    # Prepare variables to store
    distance_criteria<-c()
    vector_is<-c()
    vector_ks<-c()
    #Try all possible candidates for merging
    for (i in unique(clusters)){
      for (k in unique(clusters)){
        if (i!=k){ #Avoid duplicates
          if (verbose==TRUE){
            cat("Trying to merge ", i, " and ", k, "\n")
          }
```

```
      cluster_data<- dist_mat

      # Assume we are in cluster i
      # We need to calculate  distances to all points in k

      #Select all points that are NOT in K (including i)
      observations_not_in_cluster_k<-as.numeric(rownames(
        cluster_data[cluster_data[,  ncol(cluster_data)]!=k, ,drop=FALSE ]))

      # Needed for further computations
      n_obs_in_k<- nrow(cluster_data[cluster_data[,  ncol(cluster_data)]==k, ,drop=FALSE ])

      # Needed for further computations
      n_obs_in_i<- nrow(cluster_data[cluster_data[,  ncol(cluster_data)]==i, ,drop=FALSE ])

      # Now select all points that are ONLY in i
      cluster_data<- cluster_data[cluster_data[,  ncol(cluster_data)]==i, , drop=FALSE]

      # Drop the cluster variable for easy of referencing
      cluster_data[,  ncol(cluster_data)]<- NA

      if (linkage_method=="average"){
        # Collect the mean of sums of distances for each element in i away from k
        cluster_data[, observations_not_in_cluster_k]<- NA
        distance_criterion<- mean((apply(cluster_data, MARGIN=1,
                                         FUN=sum,na.rm=TRUE)/n_obs_in_k))
      }

      if (linkage_method=="complete"){
        # Collect the max of max distances for each element in i away from k
        cluster_data[, observations_not_in_cluster_k]<- NA
        distance_criterion<- max(apply(cluster_data, MARGIN=1,
                                       FUN=max, na.rm=TRUE))
      }

      if (linkage_method=="single"){
        # Collect the min of min distances for each element in i away from k
        cluster_data[, observations_not_in_cluster_k]<- NA
        distance_criterion<- min(apply(cluster_data, MARGIN=1,
                                       FUN=min, na.rm=TRUE))
      }

      if(verbose==TRUE){
        cat(linkage_method, "distance is ", distance_criterion, "\n")
      }

      #  Store the results for the given pair i-k
      distance_criteria<-append(distance_criteria, distance_criterion)
      vector_is<- append(vector_is, i)
      vector_ks<-append(vector_ks, k)

    }
  }
```

```
  } #End main loop


  #==================================
  # Make the final splitting decision
  #==================================
  minimal_distance_pair<- which.min(distance_criteria)

  # Find the values of i and k to merge
  i<- vector_is[minimal_distance_pair]
  k<-vector_ks[minimal_distance_pair]

  #Merge clusters
  dist_mat[, ncol(dist_mat)] <- ifelse(dist_mat[, ncol(dist_mat)]==k,
                                       i,
                                       dist_mat[, ncol(dist_mat)])

  clusters<- dist_mat[, ncol(dist_mat)]
  if (verbose==TRUE){
    cat("(", count, ")",
        linkage_method,
        "distance",
        min(distance_criteria),
        "is achieved by merging ",
        i , " and ", k , "\n")
    print(clusters)
  }

  #Store results
  partitions[count, "Merged_1"]<- i
  partitions[count, "Merged_2"]<- k
  clusters_list[[count]]<- clusters
  decisions<- append(decisions, min(distance_criteria))
  count<-count+1
}

# Render the output
out<- list(num_clust=clusters_list, partitions=partitions, h=decisions)
return(out)
```