Data Description

The original stroke prediction dataset was obtained from the kaggle website from the link: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset
This dataset contains roughly 5,100 entries, each entry represents the unique data of a patient that gives information that would be useful to determine the risk of this individual experiencing a stroke. More specifically, the dataset contains 11 variables that are useful to predict the likelihood of the individual experiencing a stroke. I will also provide a data dictionary that concisely describes the variables and mentions the respective data type.

Name	Description	Data Type
gender	The gender of the patient, 3 levels	Nominal
age	Age of the patient	Numeric
hypertension	If the patient has hypertension	Nominal
heart_disease	If the patient has heart disease	Nominal
ever_married	If the patient has been married	Nominal
work_type	The type of work the patient does, 5 levels	Nominal
Residence_type	The type of area the patient resides, 2 levels	Nominal
avg_glucose_level	Average glucose level in blood	Numeric
bmi	Body mass index of patient	Numeric
smoking_status	The smoking status of the patient, 4 levels	Nominal
stroke	If the patient has had a stroke	Nominal

Research Question

The research question is to determine if a patient's stroke status can be predicted using the variables mentioned above in the data dictionary and to figure out which of the supervised machine learning methods are the most reliable or accurate in this prediction.

Data preparation and cleaning

The data preparation and cleaning were done in the R studio programme. Once the original dataset was read into R, the summary statistics command was run which gave some basic statistics for numerical data types such as median and mean. This dataset has 3 numerical variables which are age, average glucose level and body mass index (bmi). There were no obvious outliers for age and average glucose from viewing the summary statistics, however for bmi it was identified as a character data type.

```
summary(health)
```

```
##        id            gender               age          hypertension
##  Min.   :   67   Length:5110         Min.   : 0.08   Min.   :0.00000
##  1st Qu.:17741   Class :character    1st Qu.:25.00   1st Qu.:0.00000
##  Median :36932   Mode  :character    Median :45.00   Median :0.00000
##  Mean   :36518                       Mean   :43.23   Mean   :0.09746
##  3rd Qu.:54682                       3rd Qu.:61.00   3rd Qu.:0.00000
##  Max.   :72940                       Max.   :82.00   Max.   :1.00000
##  heart_disease     ever_married         work_type          Residence_type
##  Min.   :0.00000   Length:5110         Length:5110         Length:5110
##  1st Qu.:0.00000   Class :character    Class :character    Class :character
##  Median :0.00000   Mode  :character    Mode  :character    Mode  :character
##  Mean   :0.05401
##  3rd Qu.:0.00000
##  Max.   :1.00000
##  avg_glucose_level      bmi             smoking_status         stroke
##  Min.   : 55.12    Length:5110         Length:5110         Min.   :0.00000
##  1st Qu.: 77.25    Class :character    Class :character    1st Qu.:0.00000
##  Median : 91.89    Mode  :character    Mode  :character    Median :0.00000
##  Mean   :106.15                                            Mean   :0.04873
##  3rd Qu.:114.09                                            3rd Qu.:0.00000
##  Max.   :271.74                                            Max.   :1.00000
```

Figure 1: summary function

So to further explore this oddity I used the structure function to receive further general information about the variables in the dataset. This function confirmed that bmi was not the only variable that was read incorrectly; the variables gender, hypertension, heart disease, marriage status, work type, residence type and stroke were also classified wrongly. Further inspection of the bmi variable showed that there were NA entries causing this variable to be classified as a character data type.

```
str(health)
```

```
## 'data.frame':    5110 obs. of  12 variables:
##  $ id                : int  9046 51676 31112 60182 1665 56669 53882 10434 27419 60491 ...
##  $ gender            : chr  "Male" "Female" "Male" "Female" ...
##  $ age               : num  67 61 80 49 79 81 74 69 59 78 ...
##  $ hypertension      : int  0 0 0 0 1 0 1 0 0 0 ...
##  $ heart_disease     : int  1 0 1 0 0 0 1 0 0 0 ...
##  $ ever_married      : chr  "Yes" "Yes" "Yes" "Yes" ...
##  $ work_type         : chr  "Private" "Self-employed" "Private" "Private" ...
##  $ Residence_type    : chr  "Urban" "Rural" "Rural" "Urban" ...
##  $ avg_glucose_level: num  229 202 106 171 174 ...
##  $ bmi               : chr  "36.6" "N/A" "32.5" "34.4" ...
##  $ smoking_status    : chr  "formerly smoked" "never smoked" "never smoked" "smokes" ...
##  $ stroke            : int  1 1 1 1 1 1 1 1 1 1 ...
```

Figure 2: structure function

I then proceeded to convert the variables mentioned above into their correct data types using the as.factor() command as they were all respectively meant to be categorical data types apart from bmi which was converted to a numerical data type using as.numeric(). It also made sense to remove all NA entries from the dataset using the na.omit() command as these missing values could affect further data analysis.

Data preparation and cleaning

```
health$smoking_status <- as.factor(health$smoking_status)
health$bmi <- as.numeric(health$bmi)
```

```
## Warning: NAs introduced by coercion
```

```
health$gender <- as.factor(health$gender)
health$hypertension <- as.factor(health$hypertension)
health$heart_disease <- as.factor(health$heart_disease)
health$ever_married <- as.factor(health$ever_married)
health$work_type <- as.factor(health$work_type)
health$Residence_type <- as.factor(health$Residence_type)
health$stroke <- as.factor(health$stroke)
```

```
cleandf <- na.omit(health)
```

Figure 3: converting variables to correct types

The final step to cleaning this data was to conduct a simple outlier detection on all three numerical variables: age, average glucose level and bmi. The general idea was to first generate a boxplot for each numerical variable to visually detect any outliers. Then to create a summary statistic object from each variables' boxplot which contained each respective variables' outlier values. From these summary statistics we compute the minimum value which can be used as a filter to remove all outlier values for each respective variable.

```
cleandf <- na.omit(health)

bmi_boxplot <- boxplot(cleandf$bmi)
```

```
min_bmi_outlier <- min(bmi_boxplot$out)
bmi_clean <- cleandf[cleandf$bmi < min_bmi_outlier, ]

glucose_boxplot <- boxplot(bmi_clean$avg_glucose_level)
```

```
min_glucose_outlier <- min(glucose_boxplot$out)
glucose_clean <- bmi_clean[bmi_clean$avg_glucose_level < min_glucose_outlier, ]


age_boxplot <- boxplot(glucose_clean$age)
```

```
min_age_outlier <- min(age_boxplot$out)
```

```
## Warning in min(age_boxplot$out): no non-missing arguments to min; returning Inf
```

```
health_clean <- glucose_clean[glucose_clean$age < min_age_outlier, ]
```

Figure 4: creating the outlier filter

Following the removal of these outliers using the filters, I ran the summary statistics for the original dataset and for the dataset with the outliers removed.

Data preparation and cleaning

There was not a significant change to the values of the age variable. However, the maximum value for the average glucose level dropped from 271.74 to 167.41. Also the maximum value for bmi was reduced from 97.60 to 47.50.

Exploratory data analysis

The exploratory data analysis was carried out using the R studio programme. To start, I produced a plot for each individual variable in the dataset, a histogram for the numerical variables and a bar chart for the categorical variables. The individual histograms suggest that bmi is pretty much normally distributed, the average glucose levels are normally distributed with a positive skew and that age is randomly distributed with no inherent obvious pattern.

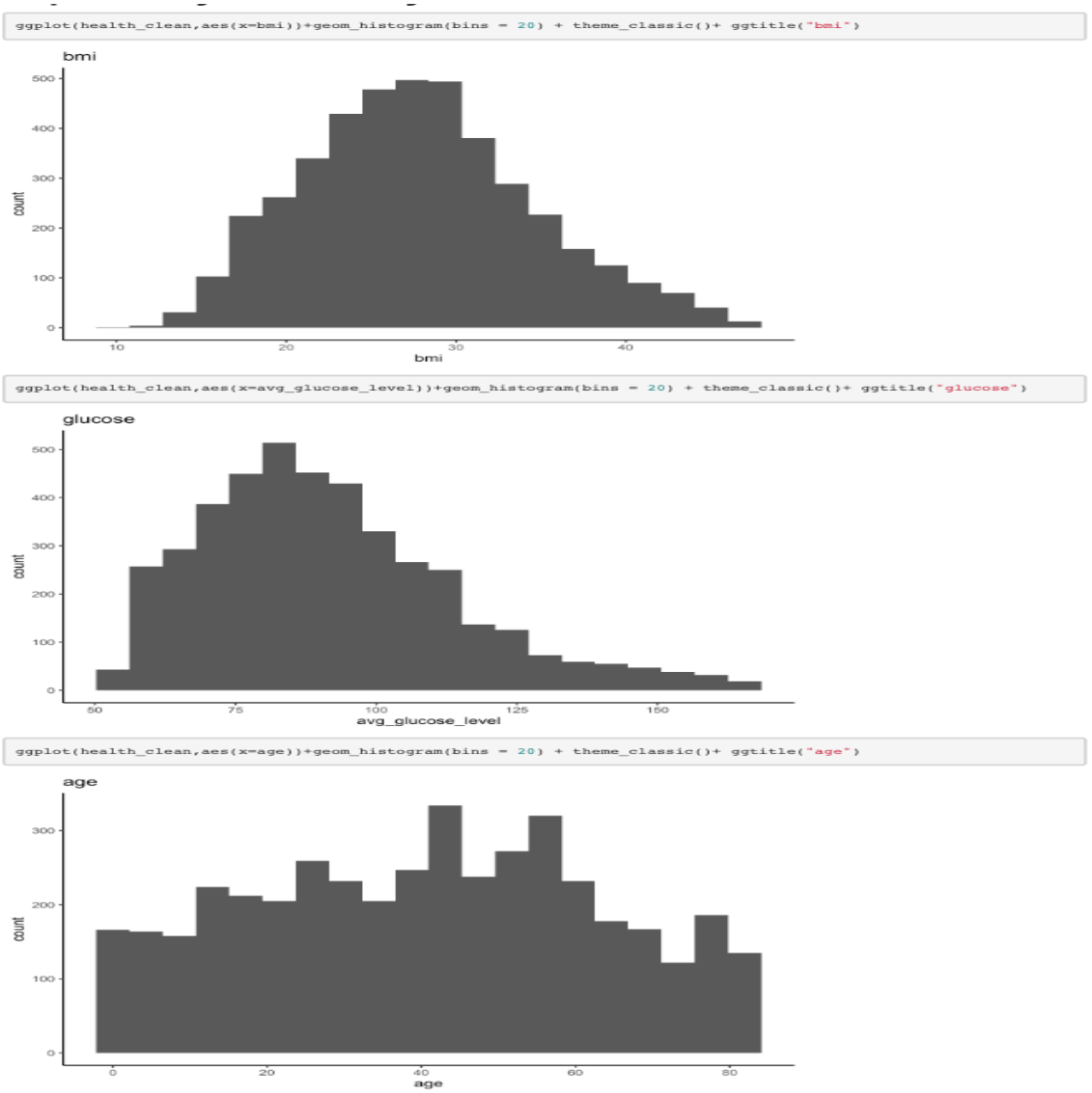


Figure 5: histograms for numerical variables

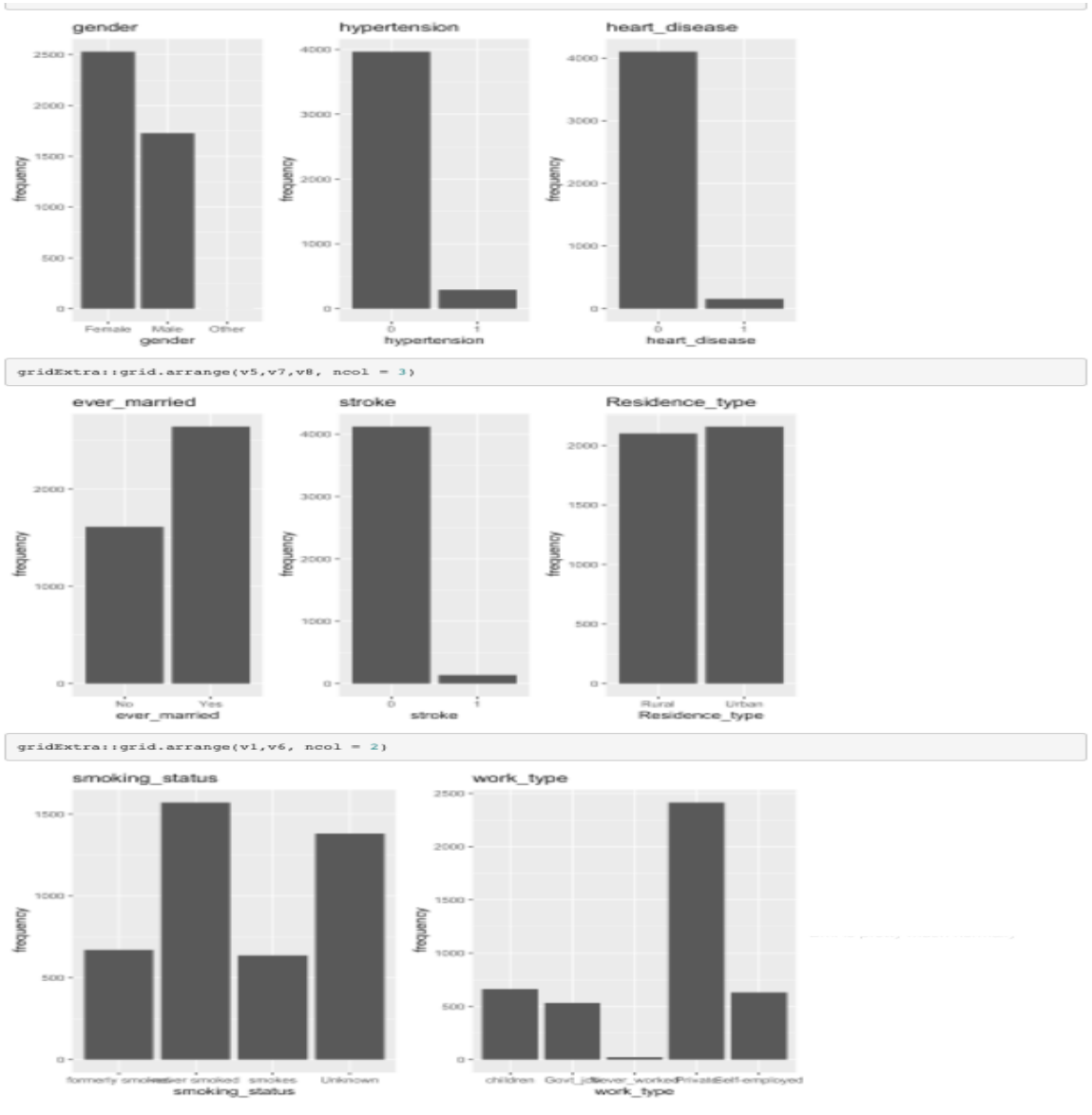
Exploratory data analysis

The barchart for smoking status suggests that the majority of patients never smoked, the second largest group were patients whose smoking status was unknown, the third largest group were patients that were ex-smokers and the group with the fewest numbers were of patients who currently smoke.
The barchart for genders suggests the majority of patients in this dataset were females, then followed by males and although the dataset had a third choice called ‘other’ there was not a single case of a patient being classified under this level.

The barchart for hypertension and heart disease suggests the vast majority of patients did not suffer from these conditions and a very small minority did. Also the barchart for marriage status showed that most patients had been married in comparison to never have been married.

The barchart for work type shows that the majority of patients worked in the private sector, the second largest group were those who worked with children, third largest group was self employment, fourth largest group were those working in government jobs and the least populous group were patients who have never worked.

The barchart for stroke suggests that the vast majority of patients in this dataset have never experienced a stroke in comparison to those who have. Also the barchart for residence type shows that the number of patients who live in rural areas are fractionally less than those who live in urban areas.



Exploratory data analysis

Now for other visualisations, due to the research question I will produce plots for stroke as the dependent variable or as the target variable. I will produce box plots of stroke status against the numerical/continuous variables and then mosaic plots for stroke status against other categorical variables.
The box plots suggest that:

The median bmi for patients who have had a stroke is slightly higher than the median bmi for patients who have not had a stroke.

The median average glucose levels for patients who have had a stroke is no different than those who have not had a stroke.

The median age for patients who have had a stroke is substantially higher than the median age for patients who have not had a stroke.

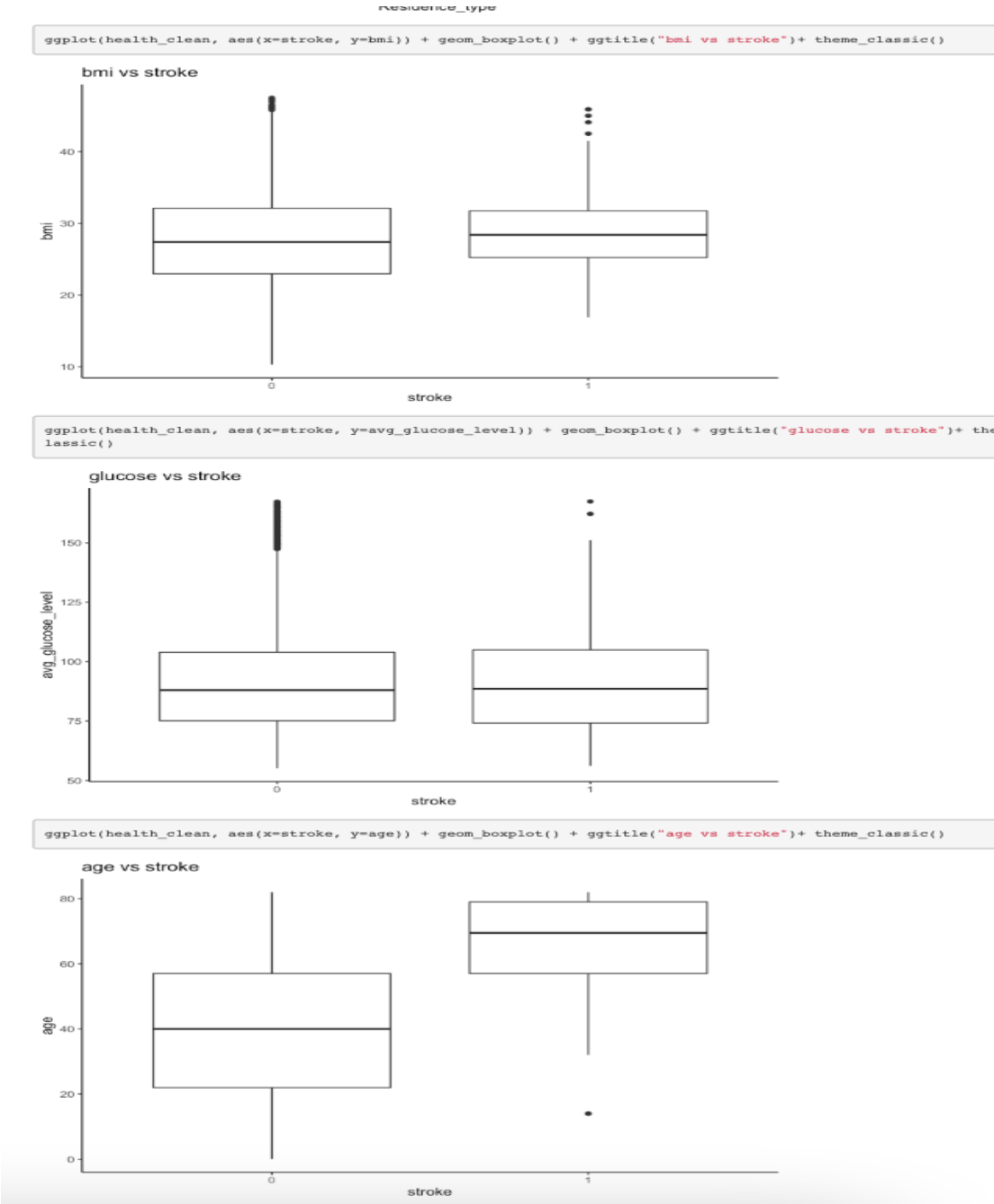


Figure 7: box plots for numerical variables against stroke

Exploratory data analysis

The mosaic plots suggest that:

We can see that the proportion of females that have stroke is larger than that for males that have stroke, the same statement can be said for the demographic with no stroke.

We can see that the proportion of people with hypertension and stroke is smaller than that of those with no hypertension and stroke,the same statement can be said for the demographic with no stroke.

We can see that the proportion of people with heart disease and stroke is smaller than that of those with no heart disease and stroke,the same statement can be said for the demographic with no stroke.

We can see that the proportion of people who have been married and had a stroke is larger than that of those who have not married and had a stroke, the same statement can be said for the demographic with no stroke.

We can see that the proportion of rural people who have had a stroke is roughly the same as urban people who have had a stroke, the same statement can be said for the demographic with no stroke.

We can also apply principal component analysis to the three numerical variables. This will give us the choice to use these principle components or the raw separate variables when applying our supervised methods to reduce the input and possibly increase the performance and load times of the supervised methods such as classification and regression.

We begin by removing the categorical variables from the dataset and apply principal component analysis to the remaining subset which only contains the numerical variables. The output gives the standard deviation of each component. Also the principal components are given in order of which explain the most variance. The next step is to run a summary on the individual principal components which will give us statistics such as the proportion of variance each component holds as well as the cumulative proportion, which is the sum of these individual proportions.

From this we found that the first component explains roughly 46% of the variance, the second explains 33% of the variance and the last explains 21% of the variance.

We also calculated the actual value of the variance which will be useful in plotting the cumulative proportion of variance (PEV) plot with a dotted line representing a threshold of our choice. This plot will suggest how many principal components are sufficient for capturing a certain threshold of total variance which in our case we have chosen to be 80%.

From the plot we found PC1 and PC2 to account for roughly 80% of the variance, so we will continue to work with these two components to plot a biplot.

Exploratory data analysis

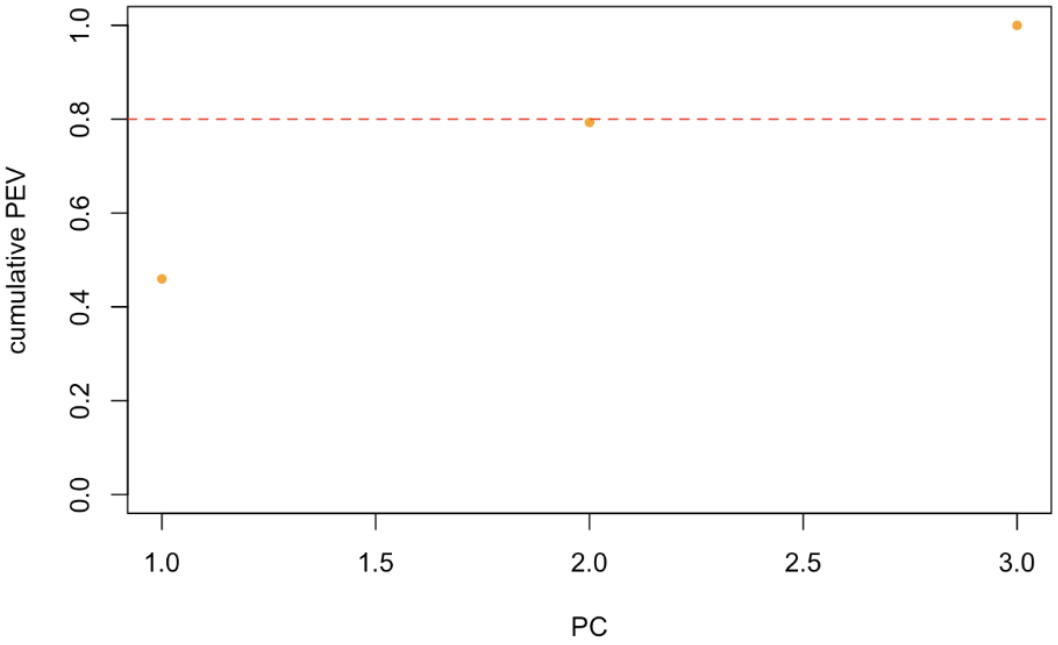


Figure 8: threshold plot for the cumulative PEV

On the biplot the length of the vectors/arrows are directly proportional to the variability included in the two components, PC1 and PC2. Hence the average glucose level contributes to a higher proportion of the variability to PC1 and PC2 than bmi and age. The angles between the vectors/arrows constitutes the correlation between variables which are represented by said vectors/arrows. The larger the angle the weaker the correlation between the variables. Hence bmi and age are strongly positively correlated, whereas the correlation between average glucose level and age is lacking and much weaker.

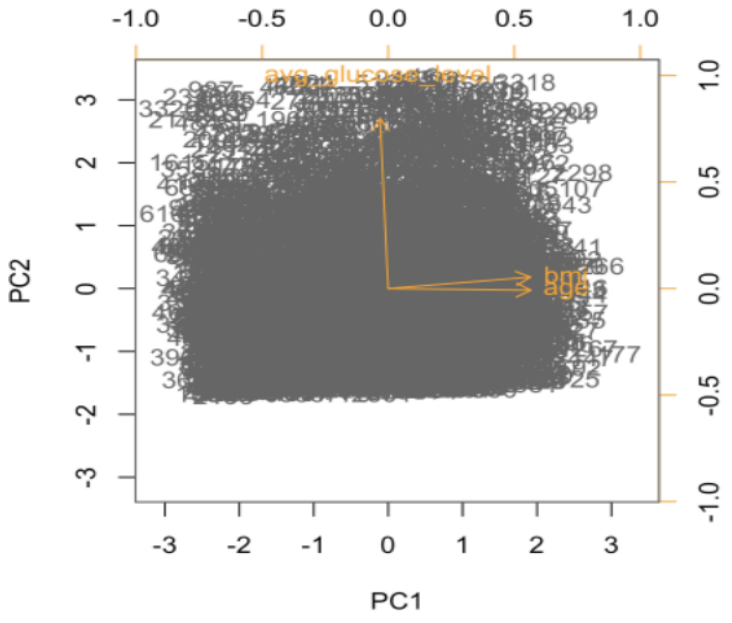


Figure 9: biplot for PC1 against PC2

High performance computational implementation

I decided to use PySpark on a virtual machine running Ubuntu for my implementation to distribute the supervised machine learning methods. To start I read in my cleaned dataset from the hadoop cluster. To do this I first had to format and then start the hadoop services using the start-dfs and start-yarn functions. Which starts services such as the resourcemanager, nodemanager, datanode, namenode, secondarynamenode and jps. The last thing I had to do before reading in the dataset was to create an input directory in HDFS which is where the dataset could be accessed.

After the dataset was read in successfully using the hadoop cluster, we had to start the Spark services by changing the directory to Spark home and then start both the master and workers which handles all the processes and allows us to monitor the clusters. If the status of the workers is alive we can proceed to start PySpark. Once we create a new notebook in jupyter, we import the SparkSession and SparkConf libraries and configure SparkSession and SparkContext respectively.

When the dataset was imported into the notebook I chose to do some extra exploratory data analysis but this time heavily focusing on the integer/double data types. Due to the fact we were going to predict the likelihood of stroke we checked the frequency of each level in the stroke variable and found that only 136 patients had experienced a stroke whereas the majority of patients had not, numbering at 4120.

With the help of importing the scatter matrix and pandas functions and using the seaborn package. We were able to produce visualisations showing correlation in scatter plots and heat maps respectively. The scatter plots showed a random pattern of points with no obvious correlation and the heat maps confirmed this by showing the correlation between all numeric variables to be weak.

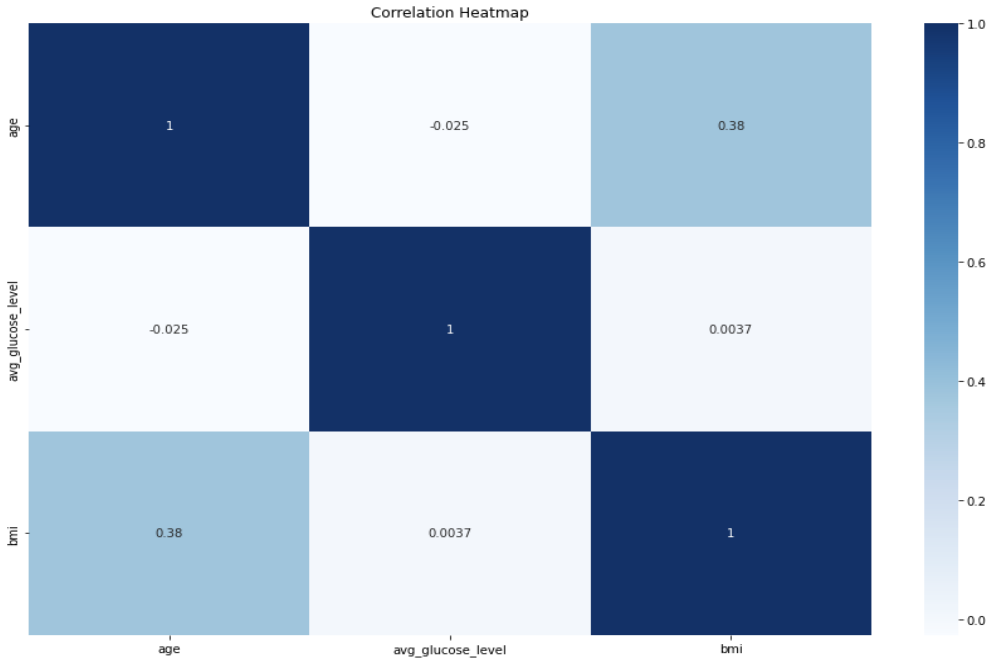


Figure 10: Heat map matrix of all numeric variables

Machine learning prediction

Machine learning prediction

Once all correlations between all numeric variables are confirmed to be negligible, we can progress to start the process of preparing the dataset to be used in a variety of supervised machine learning methods. Initially, I selected all variables from the dataset to be a part of my machine learning process except for the principal components. This was a personal choice as there are only a few numeric variables in this particular dataset and hence dimensionality reduction is not imperative for this dataset. As I believe using principal components will not reduce the processing time for any particular machine learning method as this dataset does not have an excessive amount of variables to begin with.

Before any specific machine learning method is applied, we have a problem in which the machine learning library in spark only handles datasets that contain numeric variables. So the categorical variables have to be prepared before they can be used in Spark and the simplest way to do this is to encode each categorical variable. The method of encoding that will be used is called one hot encoding.

To start this process the OneHotEncoder, StringIndexer and VectorAssembler functions must be imported. First each categorical variable is indexed by the StringIndexer and then the indexed categorical variables are converted to one hot variables using the OneHotEncoder. The StringIndexer is used again to encode the label, as the label variable represents the outcome of whether the patient has a stroke or not. The final step is to create a features variable by combining all features into a single vector using the VectorAssemble function.

Now that the dataset is ready to be used in spark we import the Pipeline function, this tool is used to optimise and enhance the performance of the machine learning method and it does this by combining multiple stages of the machine learning process and this essentially streamlines the process into a smooth singular workflow.

	label	features	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1.0	(0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, ...	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
1	1.0	(0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, ...	Male	74.0	1	1	Yes	Private	Rural	70.09	27.4	never smoked	1
2	1.0	(1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, ...	Female	69.0	0	0	No	Private	Urban	94.39	22.8	never smoked	1
3	1.0	[1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, ...	Female	78.0	0	0	Yes	Private	Urban	58.57	24.2	Unknown	1
4	1.0	(1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, ...	Female	81.0	1	0	Yes	Private	Rural	80.43	29.7	never smoked	1
5	1.0	(1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, ...	Female	61.0	0	1	Yes	Govt_job	Rural	120.46	36.8	smokes	1
6	1.0	(1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, ...	Female	54.0	0	0	Yes	Private	Urban	104.51	27.3	smokes	1
7	1.0	(1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, ...	Female	50.0	1	0	Yes	Self-employed	Rural	167.41	30.9	never smoked	1
8	1.0	(1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, ...	Female	60.0	0	0	No	Private	Urban	89.22	37.8	never smoked	1
9	1.0	[0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, ...	Male	71.0	0	0	Yes	Private	Urban	102.87	27.2	formerly smoked	1

Figure 11: Table showing the hot encoded variables

Machine learning prediction

The final step before applying any particular machine learning method was to randomly split the data into training and test datasets using a standard 70/30 split as well as setting a seed so the results could be easily reproduced.

The first supervised machine learning method that I chose to predict whether a patient had a stroke or not was the support vector machines (SVM) method although it is fairly difficult to interpret the method's raw output, it's highly accurate in its prediction given that the a sufficient kernel function and other parameters are chosen, it is immune or impartial to noise and this method is not susceptible to overfitting to the training data. To deploy this method the linear support vector machine classifier must be imported and configured. Also the classifier must be trained using the training data and then its performance must be tested using the testing data. To optimise the accuracy of the prediction, I found setting the maximum number of iterations to 110 and the kernel function was left to the default, which was linear.

To assess the performance of this method, the model evaluator must be imported which will give a percentage of how accurate the SVM method was. This method performed decently well with the models' prediction being roughly 81% accurate.

```
from pyspark.ml.classification import LinearSVC

lsvc = LinearSVC(maxIter=110, regParam=0.1)
lsvcModel = lsvc.fit(train)
```

```
print("Coefficients: " + str(lsvcModel.coefficients))
print("Intercept: " + str(lsvcModel.intercept))

predictions = lsvcModel.transform(test)
```

```
Coefficients: [-0.10149005418245566,-0.10123589821019571,-0.00034933422809724934,0.0005144208331199358,0.005711954123502509,-0.04298214329381941,-0.0
11773941821754905,-0.04497145088240285,-0.044542263634571956,-0.0010317527618000908,-0.0032741235305366397,-0.004407263379057498,-0.00307667263253505
25,0.0004382278835826 62,4.718025722060206e-06,3.737023067885311e-05]
Intercept: -0.8934336243589422
```

```
print('Test Area Under ROC', evaluator.evaluate(predictions))
```

```
Test Area Under ROC 0.8125793650793663
```

Figure 12: result for linear function SVM

I chose to implement a second SVM model. However, I opted to use the radial basis kernel function. This was not possible using the SVM classifier in PySpark as its only kernel function that could be picked was the linear function. So, I chose to implement this second SVM using R studio. The only major difference between implementing this SVM to the previous one was that I had to use the caret R library to one hot encode my categorical variables. After this stage, I again configured the 70/30 training-test data split. I gave the SVM a cost of 1 and used the before mentioned radial basis kernel function.

This second SVM implementation was again evaluated by measuring the accuracy of its predictions. Fortunately, this method performed substantially higher than the linear SVM with the model's prediction being roughly 96.5% accurate.

```
acc_svm_results
```

```
## [1] 0.9655442
```

Figure 13: accuracy score for radial basis SVM

Performance evaluation and comparison of methods

In this section, I will compare my supervised machine learning (and high performance computational techniques) results with those of my fellow project members. A decision tree was implemented in PySpark by one of these members, it performed extremely poorly with the prediction being exactly 50% accurate. This was the highest accuracy the member could achieve by tinkering with the max depth value and the optimum value for this parameter was found to be 3. Also, the max depth value can be described as the maximum number of times a path on the decision tree can branch.

Another member of my group implemented a random forest through the use of R studio. This method performed much better than the decision tree with the prediction being roughly 97% accurate. This member also implemented k-prototype cluster analysis in dask which created two large distinct sets of clusters, with five smaller sets of clusters that were insignificant.

The third method that was implemented by another member of the group was the gradient boosting tree (GBT) through the use of PySpark. This method performed with the prediction accuracy being roughly 87% accurate. This was the highest accuracy this member could achieve by setting the maximum number of iterations set to a value of 10. Also, the maximum number of iterations is the number of trees created by the algorithm.

Discussion of the findings

The poor performance of the decision tree method could be put down to the fact that decision trees tend to be highly sensitive to noise and that decision trees tend to fit closer to the training data than adapting to the test data.

The difference in performance between the decision tree and random forest method suggests that the training data must have contained a fair amount of noise as the major difference between these two methods was that the random forest is resistant to noise and missing data. An alternative reasoning for this major difference in performance could be that a single decision tree was not complex enough to handle the substantial amount of features this dataset contained in predicting our target feature, the stroke status.

The accuracy score for the gradient boosted tree is surprising because if they are configured correctly they tend to perform better than the random forest method. This oddity could be due to the fact that the data was too noisy and the random forest may have overfitted towards the noise, however this is just speculation. The most probable cause was due to the fact that the categorical variables were encoded differently for random forest and GBT by the individual users.

From these performance evaluations we can say that the supervised methods that performed the best and were the most reliable in predicting the stroke status of patients in this dataset were (best performing to least):

1. Random forest
2. Support vector machine (Radial basis kernel function)
3. Gradient boosted tree
4. Support vector machine (Linear kernel function)
5. Decision tree

Appendix

The data management plan and author contributions can be found in the appendix below.

Author Contribution statement

The data collection was planned and executed by Abhishek Thapa and Aditya Parvati Parvati. The data cleaning process was done by Hamza Bin Wasi and Ian Garland Ofori-Addo. The exploratory data analysis was completed by Valentina Ekeleme and Manraj Rai.

The decision tree was implemented by Aditya Parvati Parvati, the random forest and k-prototype cluster analysis was implemented by Valentina Ekeleme, the gradient boosting tree was implemented by Hamza Bin Wasi and the support vector machines were implemented by Manraj Rai.

Data Management Plan

1. Overview

Researcher: Abhishek Thapa, Aditya Parvati Parvati, Hamza Bin Wasi, Ian Garland Ofori-Addo, Manraj Rai, Valentina Ekeleme
Project title: Distributed Data Analysis
Project duration: 150 Hours
Project context: The purpose of this coursework is to analyse a large dataset using a combination of machine learning methods and high-performance computation methods. The context was if a patient's stroke status could be predicted using the many variables found in our dataset and which supervised methods were the most accurate in predicting this outcome correctly.

Appendix

2. Defining your data/research sources

2.1 Where will your data/research sources come from?

We found our dataset online via https://www.kaggle.com/datasets/

2.2 How often will you get new data?

We will be using a one-off dataset as we have decided as a group that our research question does not require more than a single batch of data.

2.3 How much data/information will you generate?

We aim to use data that is no more than 1MB in size.

2.4 What file formats will you use?

As we will be using R studio for data cleaning and exploratory data analysis and high-performance computational techniques, our data will be stored as a .csv file extension which can easily be read into software such as R studio, Jupyter notebook and the Hadoop distributed file system.

Appendix

3. Organising your data

3.1 How will you structure and name your folders and files?

As we will be using a singular dataset, we will give the file a simple name and will be placed in a singular easily accessible directory. Any changes to the dataset will be saved as a new file going with the naming convention of ‘V1’ for version 1, ‘V2’ for version 2, etc.

3.2 What additional information is required to understand each data file?

A data dictionary that explains the variable types and provides the meta data and any explanation to the code we create will be produced alongside the raw code.

3.3 What different versions of each data file or source will you create?

The different versions of the file will be created according to the needs of the task we are completing. For example, if we are required to do principal component analysis, we will create a version of the dataset with the new principal components and name it accordingly to 3.1

4. Looking after your data

4.1 Where will you store your data?

The file will be saved in a directory on our personal computers and backups of the original file and new files will be stored on an online file storage service such as google drive, one drive or dropbox.

4.2 How will your data be backed up?

The data will be backed up on the online file storage services mentioned in 4.1 and if necessary, even on our individual thumb drives or hard disk drives

Appendix

4.3 How will you test whether you can restore from your backups?

A weekly inspection will take place in which we download the data from these online file storages to check whether the backed-up data is corrupted or not and whether it is still accessible.

5. Sharing your data

5.1 Who owns the data you generate?

Any files created due to the changes made to the data will belong to the group or the individual member who made the personal changes.

5.2 Who else has a right to see or use this data?

The members of this group are allowed to view and use this data for this project as the project requires the collaboration between the individual members. Also, the module leaders also have the right when grading the quality of the data generated.

5.3 Who else should reasonably have access to this data when you share it?

The public or anyone that seems to be interested in the insights found from the newly generated data, this is of course only after the project has been completed and graded due to the issues that may arise surrounding the topic of plagiarism.

5.4 What should/shouldn't be shared and why?

Any confidential and sensitive information protected by the data protection act of 2018 should not be shared for safety reasons. However, due to the dataset not disclosing any personal information the new data and insights produced from this project will inherently be non intrusive in nature.

6. Archiving your data

6.1 What should be archived beyond the end of your project?

Any new relevant insights found from the data produced that could possibly aid further research in the chosen topic in hand.

Appendix

6.2 For how long should it be stored?

Subject to how useful it may be to the individual members within the group, this could be a wide ranging period of time differing from member to member. Some members may wish to explore a new area of study using the insights found in this project or some may wish to archive or delete the findings as it is no longer useful to them.

6.3 When will files be moved into the data archive/repository?

This will be when the project has been completed, graded and ratified.

6.4 Where will the data be stored?

Either virtually on an online file storage service or physically on a thumb drive/ hard disk drive. If the data is chosen to be shared, perhaps an online publishing service such as a website would suffice.

6.5 Who is responsible for moving data to the data archive and maintaining it?

The individual members in the group who choose to share it online and if the data is chosen not to be shared there will not be a need for the data to be archived and maintained.

6.6 Who should have access and under what conditions?

The public as the original dataset used for this project was made free for all to use. This is subject to the data/insights being correctly cited and referenced to the rightful sources.

7. Executing your plan

7.1 Who is responsible for making sure this plan is followed?

All the individual members in this group must take personal responsibility in following this plan.

7.2 How often will this plan be reviewed and updated?

Once a week or until an obstruction occurs within the project which halts the group from progressing forward.

Appendix

7.3 What actions have you identified from the rest of this plan? N/A
7.4 What further information do you need to carry out these actions? N/A

Appendix

(Raw Code for EDA, HPC and ML implementation)

Data preparation, data cleaning and exploratory data analysis code for R studio:

````
---
title: "health"
output:
  html_document: default
  pdf_document: default
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

```{r}
library("ggplot2","gridExtra")
health <- read.csv("healthcare-dataset-stroke-data.csv")
```
# Data Cleaning
```{r}
summary(health)
str(health)

health$smoking_status <- as.factor(health$smoking_status)
health$bmi <- as.numeric(health$bmi)

health$gender <- as.factor(health$gender)
health$hypertension <- as.factor(health$hypertension)
health$heart_disease <- as.factor(health$heart_disease)
health$ever_married <- as.factor(health$ever_married)
health$work_type <- as.factor(health$work_type)
health$Residence_type <- as.factor(health$Residence_type)
health$stroke <- as.factor(health$stroke)

```

```{r}
which(is.na(health$bmi == 'NA'))
```

```{r}
cleandf <- na.omit(health)

bmi_boxplot <- boxplot(cleandf$bmi)
min_bmi_outlier <- min(bmi_boxplot$out)
bmi_clean <- cleandf[cleandf$bmi < min_bmi_outlier, ]

glucose_boxplot <- boxplot(bmi_clean$avg_glucose_level)
min_glucose_outlier <- min(glucose_boxplot$out)
glucose_clean <- bmi_clean[bmi_clean$avg_glucose_level < min_glucose_outlier, ]
````
````

Appendix

```
age_boxplot <- boxplot(glucose_clean$age)
min_age_outlier <- min(age_boxplot$out)
health_clean <- glucose_clean[glucose_clean$age < min_age_outlier, ]
```

Exploratory Data analysis

```{r}
ggplot(health_clean,aes(x=bmi))+geom_histogram(bins = 20) + theme_classic()+
ggtitle("bmi")
ggplot(health_clean,aes(x=avg_glucose_level))+geom_histogram(bins = 20) +
theme_classic()+ ggtitle("glucose")
ggplot(health_clean,aes(x=age))+geom_histogram(bins = 20) + theme_classic()+
ggtitle("age")

ggplot(health_clean,aes(x=smoking_status,y=frequency(smoking_status)))+geom_bar(stat
="identity")+labs(title="smoking_status",x="smoking_status",y=" frequency")
ggplot(health_clean,aes(x=gender,y=frequency(gender)))+geom_bar(stat="identity")+labs(
title="gender",x="gender",y=" frequency")
ggplot(health_clean,aes(x=hypertension,y=frequency(hypertension)))+geom_bar(stat="ide
ntity")+labs(title="hypertension",x="hypertension",y=" frequency")
ggplot(health_clean,aes(x=heart_disease,y=frequency(heart_disease)))+geom_bar(stat="i
dentity")+labs(title="heart_disease",x="heart_disease",y=" frequency")
ggplot(health_clean,aes(x=ever_married,y=frequency(ever_married)))+geom_bar(stat="id
entity")+labs(title="ever_married",x="ever_married",y=" frequency")
ggplot(health_clean,aes(x=work_type,y=frequency(work_type)))+geom_bar(stat="identity"
)+labs(title="work_type",x="work_type",y=" frequency")
ggplot(health_clean,aes(x=stroke,y=frequency(stroke)))+geom_bar(stat="identity")+labs(tit
le="stroke",x="stroke",y=" frequency")
ggplot(health_clean,aes(x=Residence_type,y=frequency(Residence_type)))+geom_bar(st
at="identity")+labs(title="Residence_type",x="Residence_type",y=" frequency")
```

```{r}
ggplot(health_clean, aes(x=stroke, y=bmi)) + geom_boxplot() + ggtitle("bmi vs stroke")+
theme_classic()
ggplot(health_clean, aes(x=stroke, y=avg_glucose_level)) + geom_boxplot() +
ggtitle("glucose vs stroke")+ theme_classic()
ggplot(health_clean, aes(x=stroke, y=age)) + geom_boxplot() + ggtitle("age vs stroke")+
theme_classic()

mosaicplot(health_clean$stroke~health_clean$gender, main ="Mosaic Plot of stroke
against gender",ylab="gender", xlab="stroke") #mosaic plots with categorical variable vs
stroke
mosaicplot(health_clean$stroke~health_clean$smoking_status, main ="Mosaic Plot of
stroke against smoking status",ylab="smoking_status", xlab="stroke")
mosaicplot(health_clean$stroke~health_clean$hypertension, main ="Mosaic Plot of stroke
against hypertension",ylab="hypertension", xlab="stroke")
mosaicplot(health_clean$stroke~health_clean$heart_disease, main ="Mosaic Plot of
stroke against heart disease",ylab="heart_disease", xlab="stroke")
```

Appendix

```
mosaicplot(health_clean$stroke~health_clean$ever_married, main ="Mosaic Plot of stroke against marriage status",ylab="ever_married", xlab="stroke")
mosaicplot(health_clean$stroke~health_clean$work_type, main ="Mosaic Plot of stroke against work type",ylab="work_type", xlab="stroke")
mosaicplot(health_clean$stroke~health_clean$Residence_type, main ="Mosaic Plot of stroke against residence type",ylab="Residence_type", xlab="stroke")
```

```{r}
summary(cleandf$age)
summary(health_clean$age)

summary(cleandf$avg_glucose_level)
summary(health_clean$avg_glucose_level)

summary(cleandf$bmi)
summary(health_clean$bmi)
```

```{r}
stroke_variables <- health_clean[, -c(1,2,4,5,6,7,8,11,12)]
```

```{r}
pc_stroke_variables <- prcomp(stroke_variables, center = T, scale. = T)
pc_stroke_variables
```

```{r}
summary(pc_stroke_variables)
```

```{r}
pc_stroke_variables_var <- pc_stroke_variables$sdev^2
pc_stroke_variables_var
pc_stroke_variables_PEV <- pc_stroke_variables_var / sum(pc_stroke_variables_var)
```

```{r}
opar <- par(no.readonly = TRUE)
plot(
  cumsum(pc_stroke_variables_PEV),
  ylim = c(0,1),
  xlab = 'PC',
  ylab = 'cumulative PEV',
  pch = 20,
  col = 'orange'
)
abline(h = 0.8, col = 'red', lty = 'dashed')
par(opar)
```

Appendix

```{r}
opar <- par(no.readonly = TRUE)
biplot(
  pc_stroke_variables,
  scale = 0,
  col = c('grey40','orange')
)
par(opar)
```

```{r}
pc_health_clean <- cbind(health_clean,pc_stroke_variables$x)
```

Appendix

Code to starting HDFS and Spark services:

```
hadoop@hadoop-VirtualBox:~$ hdfs namenode -format
hadoop@hadoop-VirtualBox:~$ start-dfs.sh
hadoop@hadoop-VirtualBox:~$ start-yarn.sh
hadoop@hadoop-VirtualBox:~$ jps


hadoop@hadoop-VirtualBox:~$ hdfs dfs -mkdir /input
hadoop@hadoop-VirtualBox:~$ hdfs dfs -put Downloads/ pc_health_clean.csv/input

hadoop@hadoop-VirtualBox:~$ cd /usr/local/Spark
hadoop@hadoop-VirtualBox:/usr/local/Spark$ ./sbin/start-all.sh
hadoop@hadoop-VirtualBox:/usr/local/Spark$ pyspark
```

Appendix

Further exploratory data exploration using high performance computational techniques (HPC code for PySpark in jupyter notebook):

```
from pyspark.sql import SparkSession
from pyspark.conf import SparkConf


In [ ]:
spark = SparkSession.builder\
 .master("local[*]")\
 .appName("ML Term Deposit Pred")\
 .getOrCreate()
sc = spark.sparkContext


In [ ]:
df = spark.read.csv('hdfs://localhost:9000/input/pc_health_clean.csv', header = True,
inferSchema = True)
df.printSchema()


In [ ]:
import pandas as pd
pd.DataFrame(df.take(10), columns = df.columns)


In [ ]:
df.groupby('stroke').count().toPandas()


In [ ]:
numeric_features = ['age', 'avg_glucose_level', 'bmi']

df.select(numeric_features).describe().toPandas()


In [ ]:
from pandas.plotting import scatter_matrix

numeric_data = df.select(numeric_features).toPandas()

axs = scatter_matrix(numeric_data, figsize = (15, 15))


In [ ]:
import seaborn as sns
import matplotlib.pyplot as plt

fig = plt.figure(figsize = (15,10))
sns.heatmap(numeric_data.corr(method = 'pearson'), annot = True, cmap = "Blues")
plt.title("Correlation Heatmap")
```

Appendix

```
In [ ]:
df = df.select('gender', 'age', 'hypertension', 'heart_disease', 'ever_married', 'work_type',
'Residence_type', 'avg_glucose_level', 'bmi',
'smoking_status', 'stroke')

cols = df.columns

df.printSchema()
```

Appendix

Linear kernel function SVM ML code for PySpark in jupyter notebook:

```
In [ ]:
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler

categoricalColumns = ['gender', 'hypertension', 'heart_disease', 'ever_married',
'work_type', 'Residence_type',
'smoking_status']

stages = []

for categoricalCol in categoricalColumns:
 stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol = categoricalCol +
'Index')
 encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()],
outputCols=[categoricalCol + "classVec"])
 stages = stages + [stringIndexer, encoder]

label_stringIdx = StringIndexer(inputCol = 'stroke', outputCol = 'label')
stages = stages + [label_stringIdx]

numericCols = ['age', 'avg_glucose_level', 'bmi']
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol = "features")
stages = stages + [assembler]


In [ ]:
from pyspark.ml import Pipeline

pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(df)
df = pipelineModel.transform(df)
selectedCols = ['label', 'features'] + cols
df = df.select(selectedCols)
df.printSchema()


In [ ]:
pd.DataFrame(df.take(10), columns = df.columns)


In [ ]:
train, test = df.randomSplit([0.7, 0.3], seed = 2022)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))




In [ ]:
```

Appendix

```
from pyspark.ml.classification import LinearSVC

lsvc = LinearSVC(maxIter=110, regParam=0.1)
lsvcModel = lsvc.fit(train)


In [ ]:
print("Coefficients: " + str(lsvcModel.coefficients))
print("Intercept: " + str(lsvcModel.intercept))

predictions = lsvcModel.transform(test)


In [ ]:
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()


print('Test Area Under ROC', evaluator.evaluate(predictions))
```

Appendix

Radial basis kernel function SVM ML code for R studio:

````
---
title: "Untitled"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
# Removing unnecessary variables
```{r}
library(kernlab)
library(class)
stroke <- read.csv("pc_health_clean.csv")
stroke_data <- stroke[ -c(1,13:15) ]
```

# One hot encoding to the categorical variables
```{r}
library(dplyr)
library(magrittr)
library(caret)


class(stroke_data)
stroke_data %<>%
  mutate_if(is.character,as.factor)

levels(stroke_data$hypertension)
levels(stroke_data$gender)
levels(stroke_data$heart_disease)
levels(stroke_data$ever_married)
levels(stroke_data$work_type)
levels(stroke_data$Residence_type)
levels(stroke_data$smoking_status)

dummy <- dummyVars(" ~ .", data = stroke_data)
newdata <- data.frame(predict(dummy, newdata = stroke_data))

stroke_data_encoded <- as.data.frame(newdata)

stroke_data_encoded$hypertension <- as.factor(stroke_data_encoded$hypertension)
stroke_data_encoded$heart_disease <- as.factor(stroke_data_encoded$heart_disease)
stroke_data_encoded$stroke <- as.factor(stroke_data_encoded$stroke)

```



# 70/30 Training/Test set split
```{r}
set.seed(2000)
````
````

Appendix

```
n_rows <- nrow(stroke_data_encoded)
training_idx <- sample(n_rows, n_rows * 0.7)
training_stroke_data <- stroke_data_encoded[training_idx,]
test_stroke_data <- stroke_data_encoded[-training_idx,]
```

SVM Training
```{r}
stroke_data_formula <- reformulate(names(training_stroke_data[, -22]), response =
'stroke')

svm_stroke_data <- ksvm(stroke_data_formula, data = training_stroke_data, kernel =
'rbfdot', C = 1)
svm_stroke_data
```

SVM Prediction
```{r}
svm_stroke_data_pred <- predict(svm_stroke_data, test_stroke_data[,-22], type=
"response")


stroke_data_results <- data.frame(
  actual = test_stroke_data$stroke,
  svm = svm_stroke_data_pred)
stroke_data_results


table_svm_results <- table(stroke_data_results[,c('actual','svm')])
table_svm_results


acc_svm_results <- sum(diag(table_svm_results)) / sum(table_svm_results)
```



```{r}
acc_svm_results
```