MACHINE LEARNING AND OPTIMISATION

ASSIGNMENT ONE (REGRESSION ANALYSIS USING R)

JOHNSON OHAKWE

4111678

CT7205

SCHOOL OF COMPUTING AND ENGINEERING

UNIVERSITY OF GLOUCESTERSHIRE, UNITED KINGDOM

MODULE TUTOR:

BHUPESH MISHRA

JULY 2022

TABLE OF CONTENTS

CHAPTER ONE

1 INTRODUCTION

Nowadays, medical insurance is becoming a necessity for many people. Depending on the medical care, insurance companies collect annual premiums. It is difficult to estimate the medical expenses due to various health conditions of payees. Some conditions are, however, more prevalent for certain segments of the population. For instance, lung cancer is more likely among smokers than non-smokers, and heart disease may be more likely among the obese. As a result, insurers invest a great deal of time and money to develop models that accurately forecast medical expenses.

This study is concerned with using both simple and multiple linear regression to model and predict (forecast) medical expenses using age, sex, body mass index (BMI), number of children (Children), smoker, and the region as independent variables.

1.2 AIM AND OBJECTIVES

The main aim of this study is to model and predict (forecast) the medical expenses of an insurance company. Below are the objectives of this study

(a) To state if it does require ML supervised, unsupervised, or semi-supervised learning and why? Which ML task (classification, clustering, regression analysis or any other) is the best in this case and why.
(b) To explore the data and document our observation.
(c) To study the correlation between each predictor and the medical cost. And state our conclusion.
(d) To use the correlation analysis to select the 3 best predictors and build a simple linear regression model based on each of the predictors.
(e) To evaluate the performance with the statistical performance measures to evaluate the statistical significance of your results.
(f) To build two multivariate regression models (1) with the three predictors above and (2) with all the predictors in the dataset. Evaluate and compare the two models.
(g) To state our overall conclusions for this task.

1.3 DEFINING ATTRIBUTES IN THE DATA SET

- **Age:** Age of primary beneficiary.
- **Sex:** Insurance contractor gender: female or male.
- **BMI:** body mass index, providing an understanding of body, weights that are relatively high or low relative to height.
- **Children:** Number of children covered by health insurance.

- **Smoker:** Yes or no if the person is a smoker.
- **Region:** The payees' residential area in the US, northeast, southeast, southwest, northwest.
- **Medical Cost:** Individual medical costs billed by medical insurance.

CHAPTER TWO

2 METHODOLOGY

This chapter talks about the methodology for statistical modelling of the insurance data set collected.

2.1 SOURCE OF THE DATA

The data used for this study is a secondary data set of an insurance company containing the following variables: age, sex, body mass index (BMI), number of children, smokers, regions, and medical cost.

2.2 LINEAR REGRESSION MODEL

Linear regression discusses the relationship and the effect of independent variables on the dependent variable. However, here we shall look at the methodology of linear regression.

This function was created using the matrix approach to linear regression analysis.
Considering both simple and multiple linear regression straight line equations.

For simple linear regression:

$$Y = \beta_0 + \beta_1 X_1 + e \quad (2.1)$$

For multiple linear regression:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k + e \quad (2.2)$$

Where; β_0 = Intercept

β_1 = Coefficient of X_1

β_2 = Coefficient of X_2

β_k = Coefficient of X_k

e = Error term

Estimation of the model parameters using the matrix approach we have.

$$X^T X = \begin{pmatrix} n & \sum X_1 & \sum X_2 \ldots & \sum X_k \\ \sum X_1 & \sum X_1^2 & \sum X_1 X_2 \ldots & \sum X_1 X_k \\ \sum X_2 & \sum X_1 X_2 & \sum X_2^2 \ldots & \sum X_2 X_k \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ \sum X_k & \sum X_1 X_k & \sum X_2 X_k \ldots & \sum X_k^2 \end{pmatrix} \quad (2.3)$$

$$X^T Y = \begin{pmatrix} \sum Y \\ \sum X_1 Y \\ \sum X_2 Y \\ . \\ . \\ . \\ \sum X_k Y \end{pmatrix} \quad (2.4)$$

$$X^T Y = (X^T X)\beta \quad (2.5)$$

To solve for β, we have.

$$\beta = (X^T X)^{-1}(X^T Y) \quad (2.6)$$

Where:

$$(X^T X)^{-1} = \frac{1}{\det (X^T X)} \text{adj } (X^T X) \quad (2.7)$$

$$\beta = \begin{pmatrix} n & \sum X_1 & \sum X_2 \ldots & \sum X_k \\ \sum X_1 & \sum X_1^2 & \sum X_1 X_2 \ldots & \sum X_1 X_k \\ \sum X_2 & \sum X_1 X_2 & \sum X_2^2 \ldots & \sum X_2 X_k \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ \sum X_k & \sum X_1 X_k & \sum X_2 X_k \ldots & \sum X_k^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum Y \\ \sum X_1 Y \\ \sum X_2 Y \\ . \\ . \\ . \\ \sum X_k Y \end{pmatrix} \quad (2.8)$$

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ . \\ . \\ . \\ \beta_k \end{pmatrix} \tag{2.9}$$

2.3 ASSUMPTIONS OF LINEAR REGRESSION

Assumptions of linear regression:

i. Normality.
ii. Linearity.
iii. Homoscedasticity.
iv. Multicollinearity.
v. Autocorrelation.

2.3.1 TEST FOR HOMOSCEDASTICITY

This assumption talks about equal variance across the error terms. The Breusch-Pagan test (Wikipedia, 2021) will be used to test for equal variance assumption. The following Lagrange multiplier (LM) yields the test statistic for the Breusch-Pagan test

The test statistic.

$$LM = \left(\frac{\partial \ell}{\partial \theta}\right)^T \left(-E\left[\frac{\partial^2 \ell}{\partial \theta \partial \theta^t}\right]\right)^{-1} \left(\frac{\partial \ell}{\partial \theta}\right) \tag{2.10}$$

This test can be carried out by means of the accompanying three strategies:

Step 1: Apply OLS in the model

$$y_i = X_i\beta + \varepsilon_i, \quad i = 1,...,\mathrm{n}$$

Step 2: Compute the regression residuals, ε_i, square them, and divide by the Maximum Likelihood estimate of the error variance from the Step 1 regression, to get what Breusch and Pagan call g_i:

$$g_i = \varepsilon_i^2 \Big/ \sigma^2 , \qquad \sigma^2 = \sum \varepsilon_i^2 \Big/ n \tag{2.11}$$

Step 3: Estimate the auxiliary regression

$$g_i = \gamma_1 + \gamma_2 z_{2i} + ... + \gamma_p z_{pi} + \eta_i \quad (2.12)$$

Step 4: The LM test statistic is then half of the explained sum of squares from the auxiliary regression in Step 3:

$$LM = \frac{1}{2}(TSS - SSR) \quad (2.13)$$

where TSS is the sum of squared deviations of the g_i from their mean of 1, and SSR is the sum of squared residuals from the auxiliary regression.

The hypothesis for the test is stated below as:

$H_0: \sigma_1^2 = \sigma_2^2 =, \dots, \sigma_n^2 = 0$ (Homoscedasticity)

$H_1: \sigma_1^2 \neq \sigma_2^2 \neq, \dots, \sigma_n^2 \neq 0$ (Heteroscedasticity)

Decision rule:
If the p-value is greater than 0.05, we then accept H_0, but if the p-value is less than 0.05, we reject H_0 and accept H_1.

2.3.2 TEST FOR AUTOCORRELATION

We test for autocorrelation to find out if the error term (U_t) is correlated in the regression. We will use the Durbin-Watson [DW] statistics to test for the presence of autocorrelation.

Durbin-Watson statistic.

$$d^* = \frac{\sum_{t=2}^{T}(e_t - e_{t-1})^2}{\sum_{t=1}^{T} e_t^2} \quad (2.14)$$

Where:
e_t, t=1, 2, . . ., T

Hypothesis:

$H_0: \rho = 0$

$H_1: \rho > 0$

Decision rule:
If the p-value is greater than 0.05, we then accept H_0, but if the p-value is less than 0.05, we reject H_0 and accept H_1.

2.3.3 TEST FOR NORMALITY

The test will be conducted to find out if the error terms are normally distributed with zero mean and constant variance i.e $\mu \sim N(0, \sigma^2)$. We shall be using the Shapiro-Wilk's test (Wikipedia, 2022).

The test statistic is

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2} \tag{2.15}$$

Where:

$x_{(i)}$ is the *i*th order statistic

$\bar{x} = \left(x_i, ..., x_n\right)/n$ is the sample mean.

The coefficients a_i, are given by.

$$\left(a_i, ..., a_n\right) = \frac{m^T V^{-1}}{C} \tag{2.16}$$

Where C is a vector norm.

$$C = \| V^{-1} m \| = \left(m^T V^{-1} V^{-1} m\right)^{1/2} \tag{2.17}$$

And m is a vector.

$$m = \left(m_1, ..., m_n\right)^T \tag{2.18}$$

is made of the expected values of the order statistics of independent and identically distributed random variables sampled from the standard normal distribution; finally, V is the covariance matrix of those normal order statistics.

The hypothesis is

H_0: $\mu_1 = 0$ Normally distributed

H_1: $\mu_1 \neq 0$ Not normally distributed

Decision rule:
If the p-value is greater than 0.05, we then accept H_0, but if the p-value is less than 0.05, we reject H_0 and accept H_1. The test statistic follows a chi-square distribution.

2.3.4 TEST FOR MULTICOLLINEARITY

Multicollinearity assumes that the independent variables are not highly correlated with each other. This assumption is tested using Variance Inflation Factor (VIF).

Where:

$$VIF = \frac{1}{(1-R^2)} \quad (2.19)$$

If the value of VIF is <10, it is acceptable and can conclude that there is no multicollinearity among the explanatory variables.

2.4 RESIDUAL STANDARD ERROR (RSE)

The residual standard error is used to measure how well a regression model fits a dataset. In simple terms, it measures the standard deviation of the residuals in a regression model.

$$RSE = \sqrt{(y-\hat{y})^2/df} \quad (2.20)$$

Where:

y: The observed value.
ŷ: The predicted value.
df: The degrees of freedom, calculated as the total number of observations – total number of model parameters.

2.5 ADJUSTED R-SQUARE (R^2)

Adjusted R^2 is a corrected goodness-of-fit (model accuracy) measure for linear models. It identifies the percentage of variance in the target field that is explained by the input or inputs.

$$Adjusted\ R^2 = 1 - \frac{(1-R^2)(N-1)}{N-P-1} \quad (2.21)$$

CHAPTER THREE

3 EXPLORATORY DATA ANALYSIS (QUESTION B)

Here we shall do some visualisation and descriptive analysis. Let's start by loading required libraries and importing the data set.

R Code 1.

```
##############  Load Libraries  ##################################################################

library(readr)
library(plyr)
library(lmtest)
library(car)
library(car)
library(tidyverse)
library(dplyr)
library(ggplot2)
library(ggthemes)
library(psych)
library(relaimpo)

######## Importing the Data Set ###################################################################

Ins <- read.csv("./insurance.csv")
Ins
```

The above R code 1 loads the required libraries for this work and helps to import our data set.

Below is a head view of the insurance data set

Table 1.

	age	sex	bmi	children	smoker	region	medicalCost
1	19	female	27.900	0	yes	southwest	16884.924
2	18	male	33.770	1	no	southeast	1725.552
3	28	male	33.000	3	no	southeast	4449.462
4	33	male	22.705	0	no	northwest	21984.471
5	32	male	28.880	0	no	northwest	3866.855
6	31	female	25.740	0	no	southeast	3756.622

R Code 2.

```
###############  Head View Of The Data Set   ########

head(Ins)
```

The above R code 2 is used to perform a head view of our data

3.1 DESCRIPTIVE ANALYSIS AND VISUALISATION

We shall look at the general exploratory data analysis of the insurance data set. Below is a general descriptive statistic of the insurance data set.

Table 2.

```
      age             sex                 bmi           children        smoker
 Min.   :18.00   Length:1338        Min.   :15.96   Min.   :0.000   Length:1338
 1st Qu.:27.00   Class :character   1st Qu.:26.30   1st Qu.:0.000   Class :character
 Median :39.00   Mode  :character   Median :30.40   Median :1.000   Mode  :character
 Mean   :39.21                      Mean   :30.66   Mean   :1.095
 3rd Qu.:51.00                      3rd Qu.:34.69   3rd Qu.:2.000
 Max.   :64.00                      Max.   :53.13   Max.   :5.000
    region           medicalCost
 Length:1338        Min.   : 1122
 Class :character   1st Qu.: 4740
 Mode  :character   Median : 9382
                    Mean   :13270
                    3rd Qu.:16640
                    Max.   :63770
```

We can see in table 2 that categorical variables such as sex, smoker, and region descriptive statistics were not provided because they are categorical variables, so we shall do an individual descriptive analysis for each variable in the insurance data set with respect to medical cost.

R Code 3.

```
############### General Descriptive Statistics Of The Insurance Data Set  ############

summary(Ins)
```

The above R code 3 perform a general descriptive analysis of the insurance data set.

3.1.1 DESCRIPTIVE ANALYSIS AND VISUALISATION FOR SEX VARIABLE

Table 3.

```
     Sex Count
1 female   662
2   male   676
```

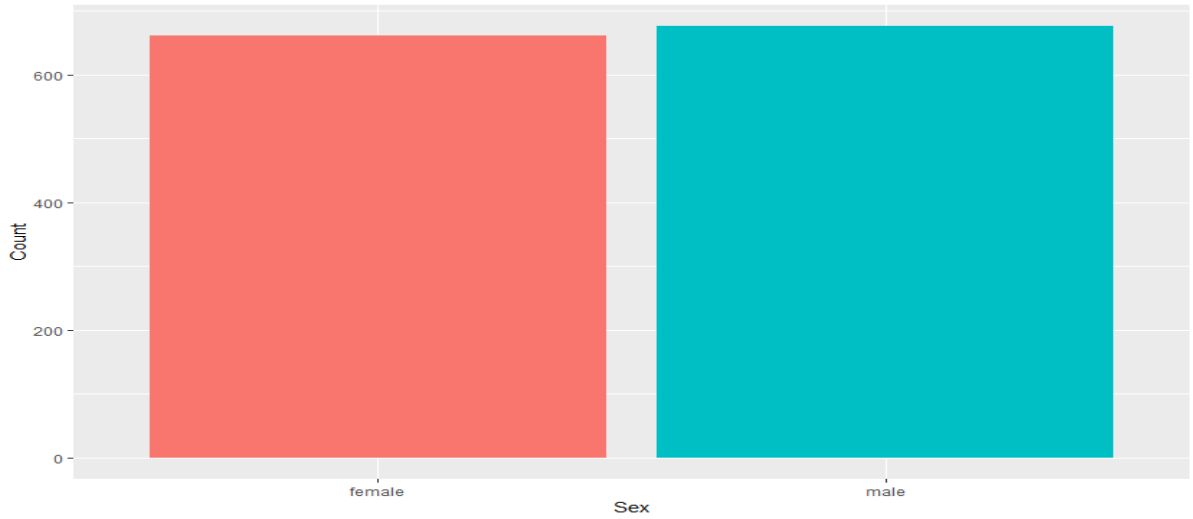


Figure 1: *A Bar chart plot for sex variable.*

Table 4

```
 Descriptive statistics by group
group: female
   vars   n     mean      sd  median  trimmed     mad     min      max    range skew kurtosis
X1    1 662 12569.58 11128.7 9412.96 10455.16 7129.08 1607.51 63770.43 62162.92 1.72     2.71
       se
X1 432.53
------------------------------------------------------------------------------
group: male
   vars   n     mean       sd  median trimmed     mad     min      max range skew kurtosis     se
X1    1 676 13956.75 12971.03 9369.62 11825.4 8121.53 1121.87 62592.87 61471 1.33     0.79 498.89
```

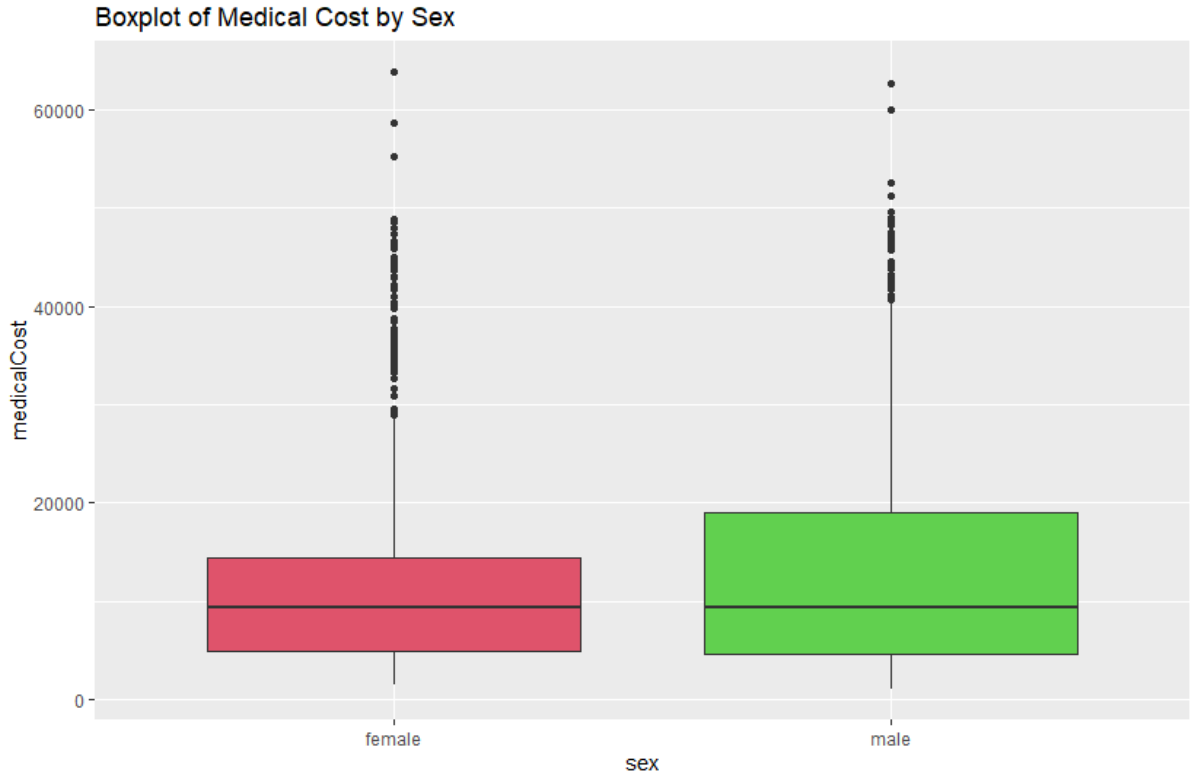


Figure 2: *A Box plot of medical cost sex.*

3.1.1.1 REMARK

From both table 3 and figure 1, we can see that there are 662 number of females and 676 number of males represented in the data set, and it can be observed that there are higher number of male than female. Whilst from table 4 and figure 2 we can notice that medical costs does not seem to be affected by sex.

R Code 4.

```
#########################################################################################
###### Descriptive and Visualisation Analysis For Sex ###################################
#########################################################################################

sx <- table(Ins$sex)
sx
sx1 <- as.data.frame(ag)
names(sx1)[c(1,2)] <- c("Sex", "Count")

sx1 %>%
  ggplot(aes(x = Sex, y = Count, fill = Sex)) +
  geom_col() +
  theme(legend.position = 'none')

describeBy(Ins$medicalCost, Ins$sex)

ggplot(data = Ins,aes(sex,medicalCost)) + geom_boxplot(fill = c(2:3)) +
  theme_classic() + ggtitle("Boxplot of Medical Cost by Sex")
```

The above R code 4 perform descriptive analysis, plot a Bar chart, and a Box plot of the sex variable.

3.1.2 DESCRIPTIVE ANALYSIS AND VISUALISATION FOR AGE VARIABLE

Table 5.

	Age	Count	Age	Count	Age	Count	Age	Count
1	18	69	28	28	38	25	48	29
2	19	68	29	27	39	25	49	28
3	20	29	30	27	40	27	50	29
4	21	28	31	27	41	27	51	29
5	22	28	32	26	42	27	52	29
6	23	28	33	26	43	27	53	28
7	24	28	34	26	44	27	54	28
8	25	28	35	25	45	29	55	26
9	26	28	36	25	46	29	56	26
10	27	28	37	25	47	29	57	26

	Age	Count
41	58	25
42	59	25
43	60	23
44	61	23
45	62	23
46	63	23
47	64	22

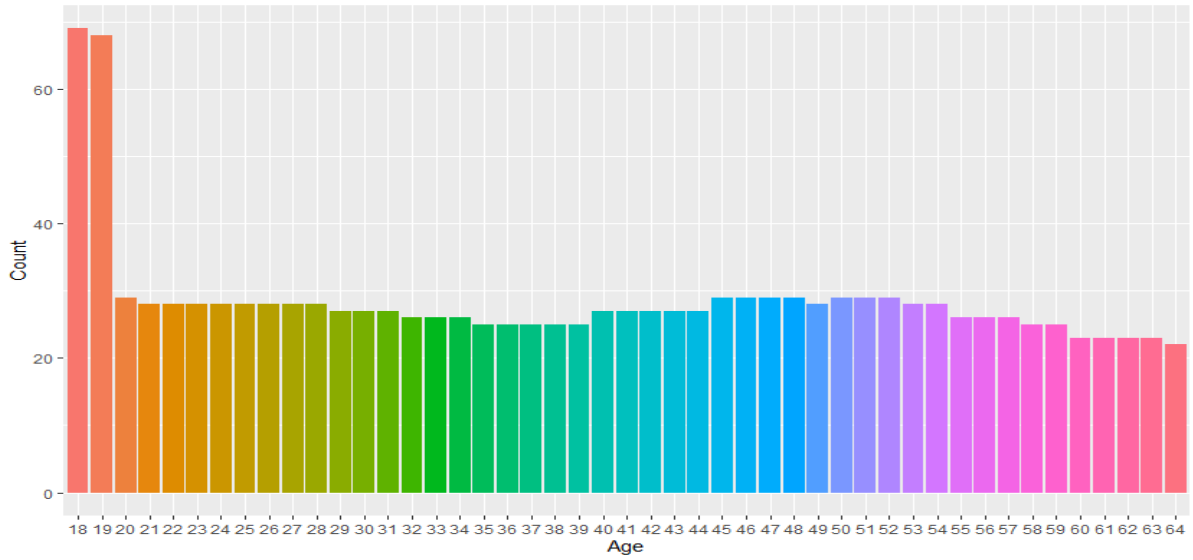


Figure 3: *A Bar chart of age variable.*

3.1.2.1 REMARK

From both table 5 and figure 3, we can observe that this data set contain more of teenagers as both 18 and 19 years old are more represented in the data set with their total number as 69 and 68 respectively.

R Code 5.

```
##############################################################################################
###### Descriptive and Visualisation Analysis For Age ########################################
##############################################################################################

ag <- table(Ins$age)
ag
ag1 <- as.data.frame(ag)
names(ag1)[c(1,2)] <- c("Age", "Count")
ne <- data.frame(ag1[(1:10),], ag1[(11:20),], ag1[(21:30),], ag1[(31:40),])
names(ne)[1:8] <- c("Age", "Count", "Age", "Count", "Age", "Count", "Age", "Count")
ne
ag1[(41:47),]

ag1 %>%
  ggplot(aes(x = Age, y = Count, fill = Age)) +
  geom_col() +
  theme(legend.position = 'none')
```

The above R code 5 perform descriptive analysis and plot a Bar chart of the age variable.

3.1.3 DESCRIPTIVE ANALYSIS AND VISUALISATION FOR SMOKER VARIABLE

Table 6.

	Smoker	Count
1	no	1064
2	yes	274

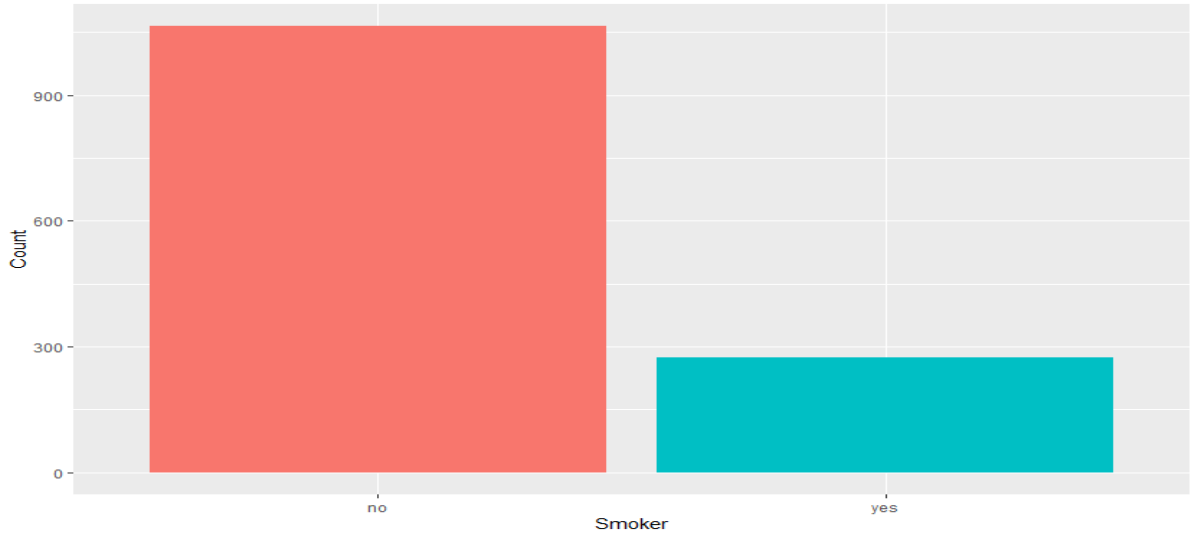


Figure 4: *A Bar chart of smoker variable.*

Table 7.

```
 Descriptive statistics by group
group: no
   vars    n    mean      sd  median trimmed     mad     min      max    range skew kurtosis
X1    1 1064 8434.27 5993.78 7345.41 7599.76 5477.15 1121.87 36910.61 35788.73 1.53     3.12
      se
X1 183.75
------------------------------------------------------------------------
group: yes
   vars   n     mean       sd   median  trimmed      mad      min      max    range skew kurtosis
X1    1 274 32050.23 11541.55 34456.35 31782.89 15167.19 12829.46 63770.43 50940.97 0.13    -1.05
      se
X1 697.25
```

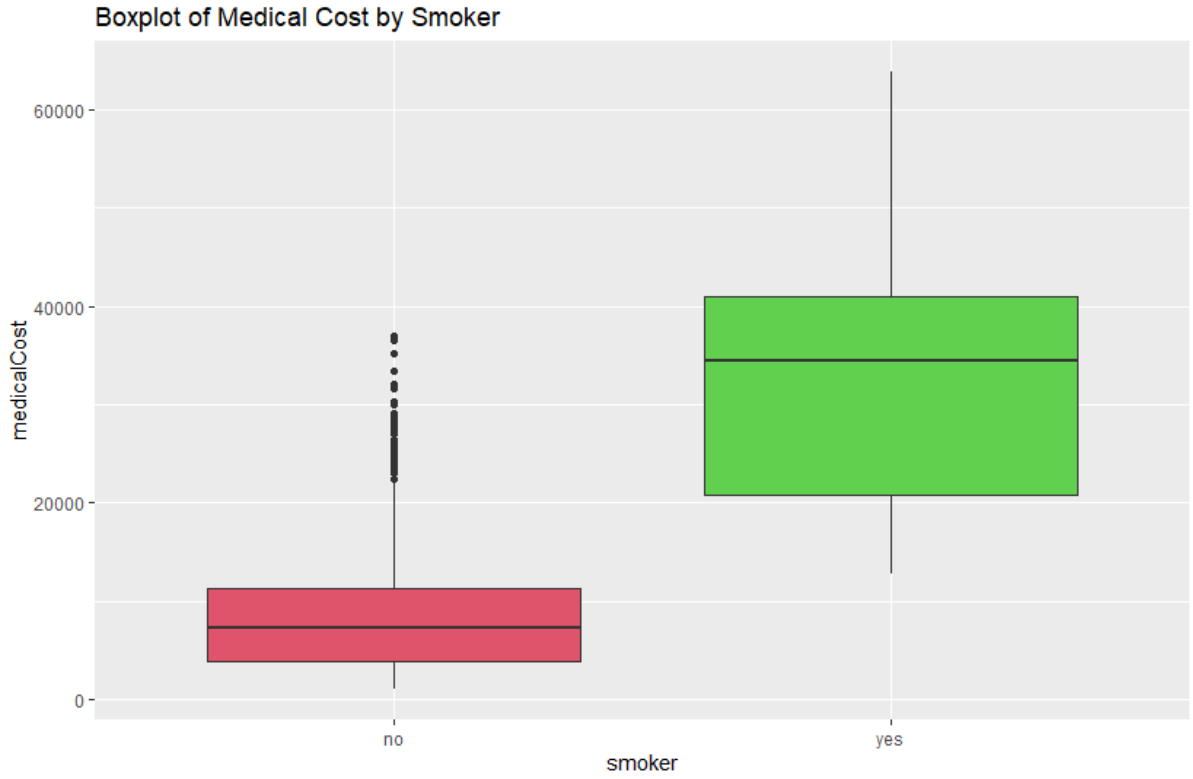


Figure 5: *A Box plot of medical cost by smoker.*

3.1.3.1 REMARK

From both table 6 and figure 4, we can see that there are 1064 number of persons who do not smoke and 274 number of persons who smoke represented in the data set, and it can be observed that there are higher number of persons do not smoke than those who smoke. Whilst from table 7 and figure 5 it can be clearly seen that smokers spends a lot more in terms of medical expenses compared to non-smokers by almost four times.

R Code 6.

```
#############################################################################################
###### Descriptive and Visualisation Analysis For Smokers ###################################
#############################################################################################

sm <- table(Ins$smoker)
sm
sm1 <- as.data.frame(sm)
names(sm1)[c(1,2)] <- c("Smoker", "Count")

sm1 %>%
  ggplot(aes(x = Smoker, y = Count, fill = Smoker)) +
  geom_col() +
  theme(legend.position = 'none')

describeBy(Ins$medicalCost,Ins$smoker)

ggplot(data = Ins,aes(smoker,medicalCost)) + geom_boxplot(fill = c(2:3)) +
   ggtitle("Boxplot of Medical Cost by Smoker")
```

The above R code 6 perform descriptive analysis, plot a Bar chart, and a Box plot of the smoker variable.

3.1.4 DESCRIPTIVE ANALYSIS AND VISUALISATION FOR REGION VARIABLE

Table 8.

	Region	Count
1	northeast	324
2	northwest	325
3	southeast	364
4	southwest	325

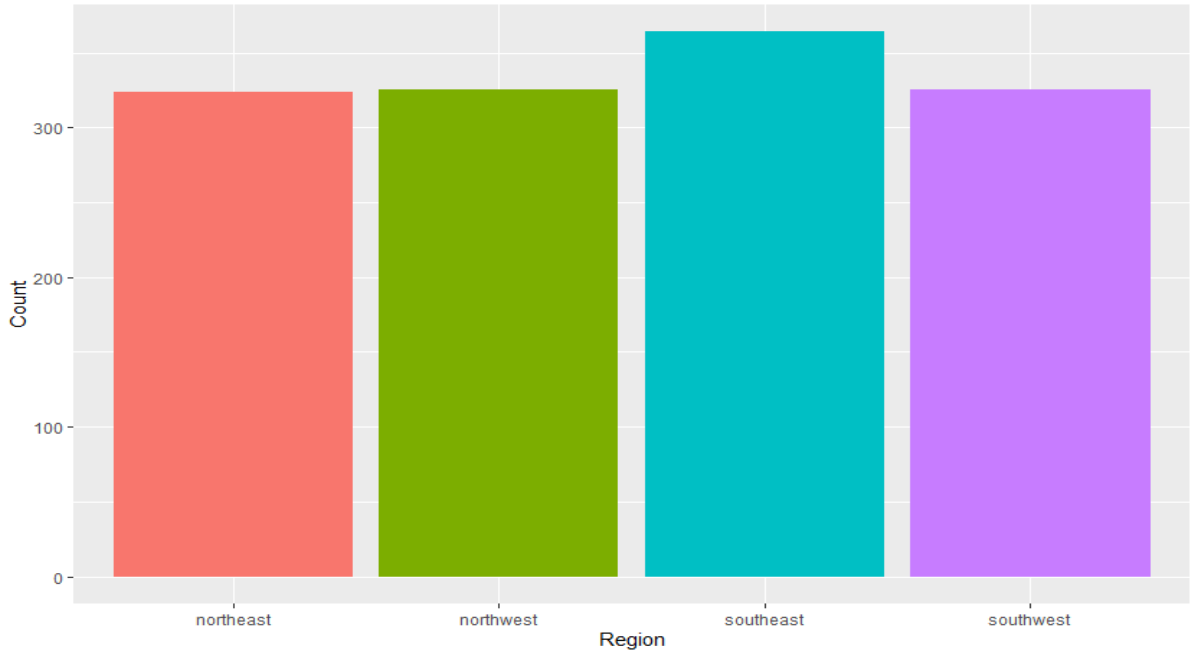


Figure 6: *A Bar chart of the region variable.*

Table 9.

```
 Descriptive statistics by group
group: northeast
   vars   n     mean      sd   median  trimmed     mad    min      max    range skew kurtosis
X1    1 324 13406.38 11255.8 10057.65 11444.31 7806.78 1694.8 58571.07 56876.28 1.48     1.68
       se
X1 625.32
------------------------------------------------------------------------------
group: northwest
   vars   n     mean       sd median  trimmed     mad     min     max    range skew kurtosis
X1    1 325 12417.58 11072.28 8965.8 10414.54 7001.14 1621.34 60021.4 58400.06 1.67     2.53
       se
X1 614.18
------------------------------------------------------------------------------
group: southeast
   vars   n     mean      sd  median  trimmed     mad     min      max    range skew kurtosis
X1    1 364 14735.41 13971.1 9294.13 12563.65 8749.51 1121.87 63770.43 62648.55 1.24     0.48
       se
X1 732.28
------------------------------------------------------------------------------
group: southwest
   vars   n     mean       sd  median  trimmed     mad     min      max    range skew kurtosis
X1    1 325 12346.94 11557.18 8798.59 10120.52 6329.39 1241.57 52590.83 51349.26 1.67     2.03
       se
X1 641.08
```

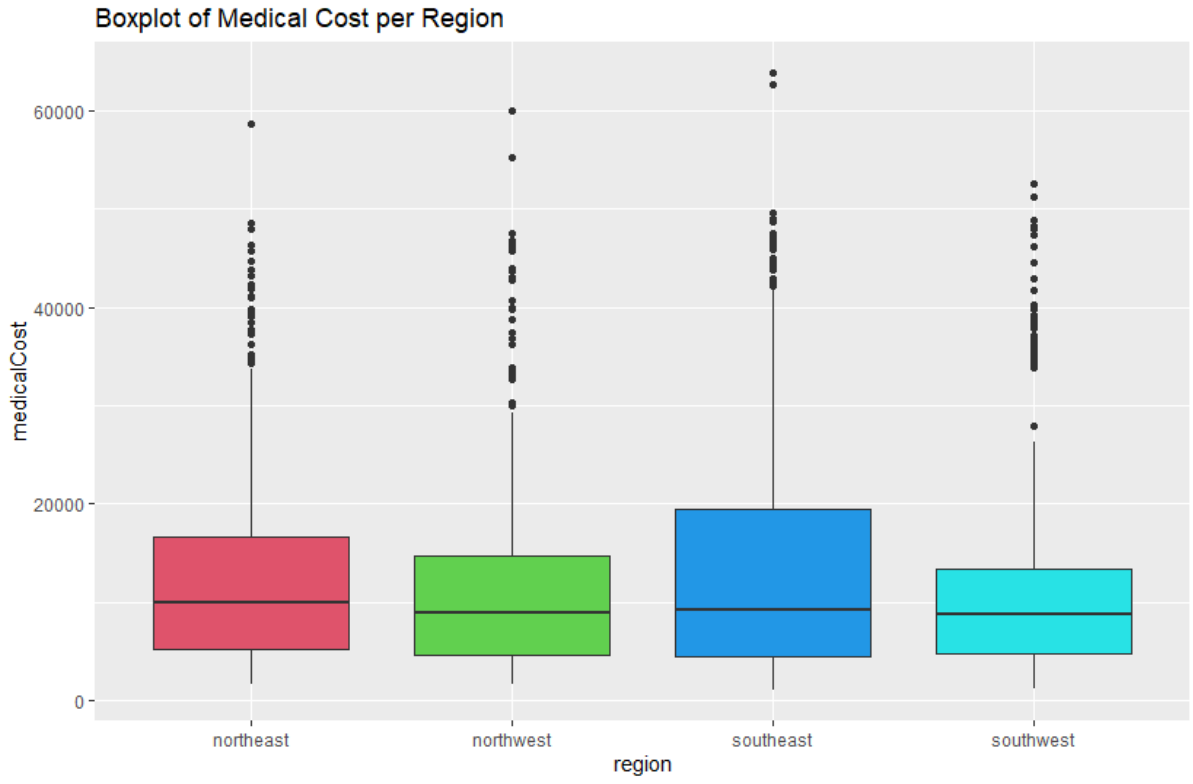


Figure 7: *A Box plot of medical cost per region.*

3.1.4.1 REMARK

From both table 8 and figure 6, we can see that there are 324 number of persons from northeast, 325 number of persons from northwest, 364 number of persons from southeast, and 325 number of persons from southwest represented in the data set, and it can be observed that there are

higher number of persons from southeast than those from other region. Whilst from table 9 and figure 7 we can disclose that region of origin doesn't have much impact with the amount of medical cost.

R Code 7.

```
#################################################################################################
###### Descriptive and Visualisation Analysis For Region ########################################
#################################################################################################

rg <- table(Ins$region)
rg
rg1 <- as.data.frame(rg)
names(rg1)[c(1,2)] <- c("Region", "Count")

rg1 %>%
  ggplot(aes(x = Region, y = Count, fill = Region)) +
  geom_col() +
  theme(legend.position = 'none')

describeBy(Ins$medicalCost, Ins$region)

ggplot(data = Ins,aes(region,medicalCost)) + geom_boxplot(fill = c(2:5)) +
   ggtitle("Boxplot of Medical Cost per Region")
```

The above R code 7 perform descriptive analysis, plot a Bar chart, and a Box plot of the region variable.

3.1.5 DESCRIPTIVE ANALYSIS AND VISUALISATION FOR CHILDREN VARIABLE

Table 10.

	Children	Count
1	0	574
2	1	324
3	2	240
4	3	157
5	4	25
6	5	18

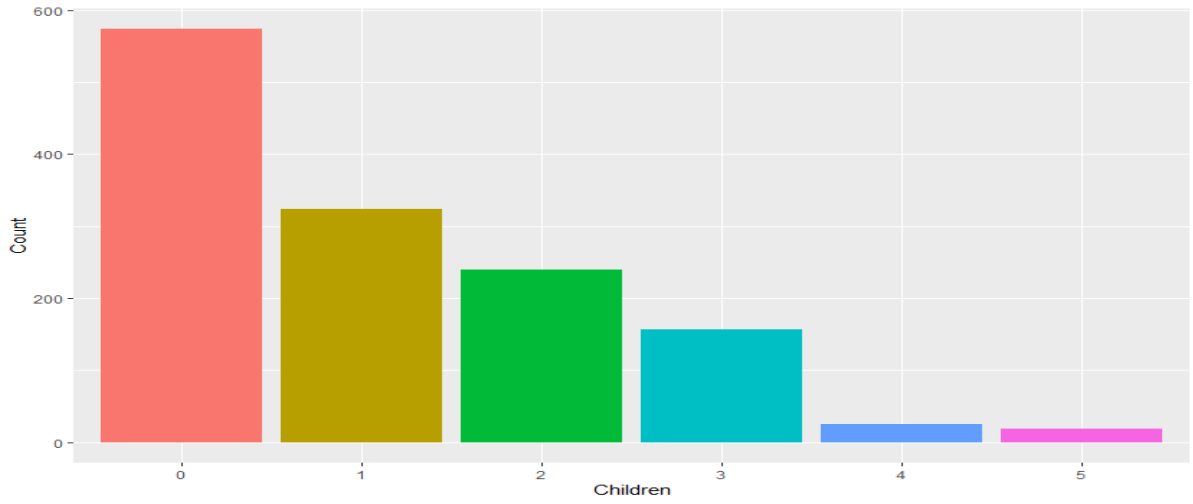


Figure 8: *A Bar chart of children variable.*

Table 11.

```
 Descriptive statistics by group
group: 0
   vars   n     mean       sd  median  trimmed      mad     min      max    range skew kurtosis
X1    1 574 12365.98 12023.29 9856.95 10155.21 10067.29 1121.87 63770.43 62648.55 1.53     1.95
       se
X1 501.84
------------------------------------------------------------------------
group: 1
   vars   n     mean       sd  median trimmed     mad     min      max    range skew kurtosis
X1    1 324 12731.17 11823.63 8483.87 10364.8 5859.46 1711.03 58571.07 56860.05 1.66     1.97
       se
X1 656.87
------------------------------------------------------------------------
group: 2
   vars   n     mean       sd  median  trimmed     mad  min      max    range skew kurtosis     se
X1    1 240 15073.56 12891.37 9264.98 12895.82 6587.43 2304 49577.66 47273.66 1.28     0.35 832.13
------------------------------------------------------------------------
group: 3
   vars   n     mean       sd   median  trimmed     mad     min     max    range skew kurtosis
X1    1 157 15355.32 12330.87 10600.55 13220.71 6918.06 3443.06 60021.4 56578.33 1.45     1.21
       se
X1 984.11
------------------------------------------------------------------------
group: 4
   vars  n     mean      sd   median  trimmed    mad     min      max    range skew kurtosis
X1    1 25 13850.66 9139.22 11033.66 12401.81 7109.3 4504.66 40182.25 35677.58 1.45     1.59
       se
X1 1827.84
------------------------------------------------------------------------
group: 5
   vars  n    mean      sd  median trimmed     mad    min      max    range skew kurtosis     se
X1    1 18 8786.04 3808.44 8589.57 8402.35 3631.71 4687.8 19023.26 14335.46 1.04     0.54 897.66
```

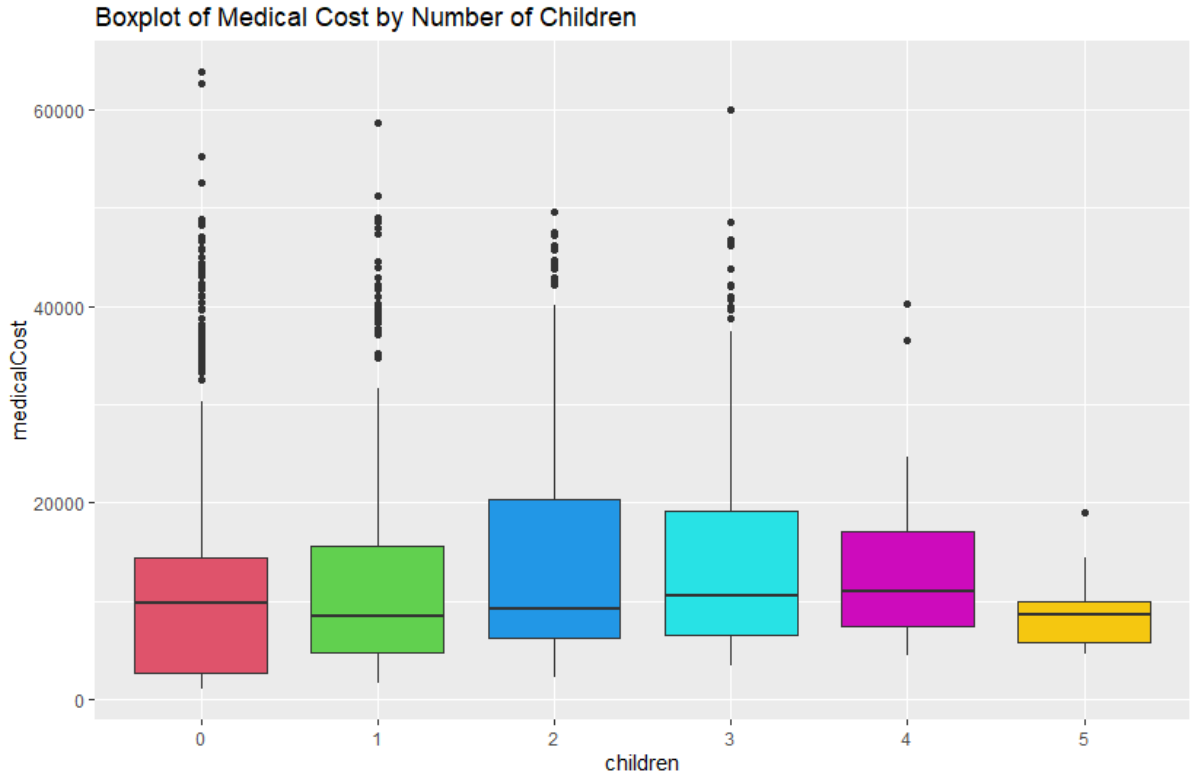


Figure 9: *A Box plot of medical cost by number of children.*

3.1.5.1 REMARK

From both table 10 and figure 8, we can see that there are 574 number of persons had no (zero) children, 324 number of persons had 1 child, 240 number of persons had 2 children, 157 number of persons had 3 children, 15 number of persons had 4 children, and 18 number of persons had 5 children represented in the data set, and it was observed that those with no children had higher number than others with children. Whilst from table 11 and figure 9 it can be seen that people with 5 children, on average, has lesser medical expenditures compared to the other groups.

R Code 8.

```
#################################################################################################
###### Descriptive and Visualisation Analysis For Children ######################################
#################################################################################################

ch <- table(Ins$children)
ch
ch1 <- as.data.frame(ch)
names(ch1)[c(1,2)] <- c("Children", "Count")

ch1 %>%
  ggplot(aes(x = Children, y = Count, fill = Children)) +
  geom_col() +
  theme(legend.position = 'none')

describeBy(Ins$medicalCost, Ins$children)

ggplot(data = Ins,aes(as.factor(children),medicalCost)) + geom_boxplot(fill = c(2:7)) +
    xlab("children") + ggtitle("Boxplot of Medical Cost by Number of Children")
```

The above R code 8 perform descriptive analysis, plot a Bar chart, and a Box plot of the children variable.

3.1.6 DESCRIPTIVE ANALYSIS AND VISUALISATION FOR BODY MASS INDEX (BMI) VARIABLE

For the sake of exploratory data analysis, we shall do future engineering (create new variable called “Obesity”). It is generally known that any person with body mass index within 30 and above is said to be obese. We shall assign no to value below 30 and yes to values within 30 and above. Below is a head view of the new data set.

Table 12.

	age	sex	bmi	children	smoker	region	medicalCost	obesity
1	19	female	27.900	0	yes	southwest	16884.924	no
2	18	male	33.770	1	no	southeast	1725.552	yes
3	28	male	33.000	3	no	southeast	4449.462	yes
4	33	male	22.705	0	no	northwest	21984.471	no
5	32	male	28.880	0	no	northwest	3866.855	no
6	31	female	25.740	0	no	southeast	3756.622	no

Table 13.

	Obesity	Count
1	no	631
2	yes	707

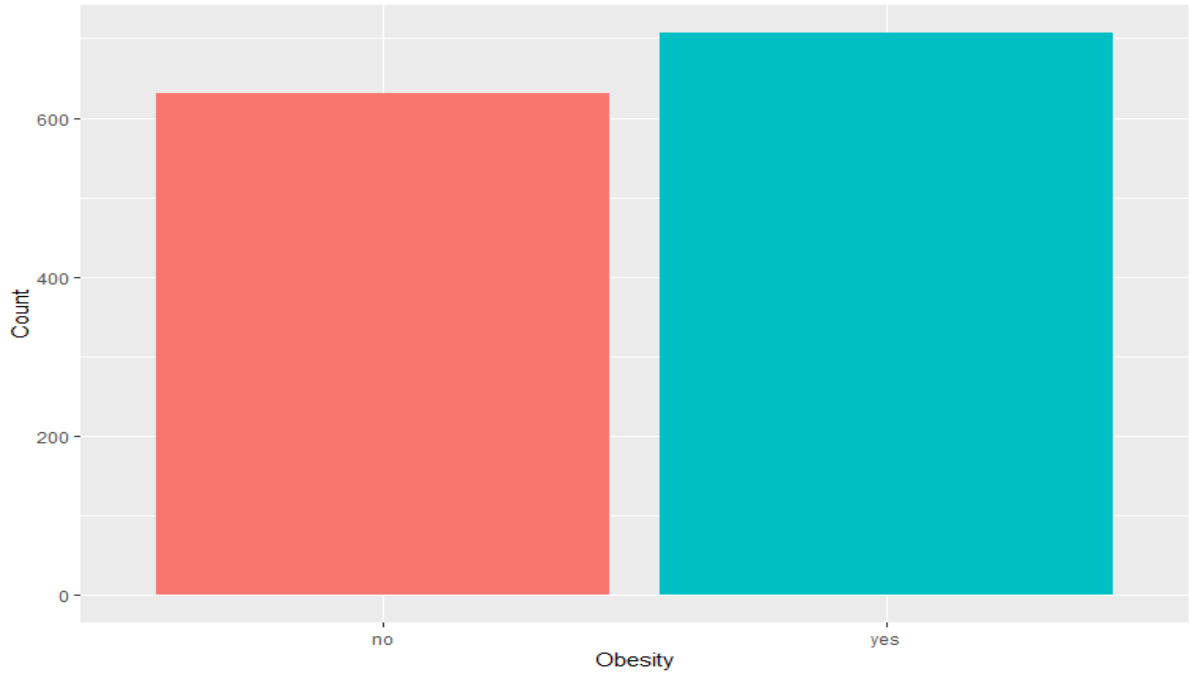


Figure 10: *A Bar chart of obesity.*

Table 14.

```
 Descriptive statistics by group
group: no
   vars   n     mean      sd  median trimmed     mad     min      max    range skew kurtosis     se
X1    1 631 10713.67 7843.54 8604.48 9772.74 7024.01 1121.87 38245.59 37123.72 0.97     0.23 312.25
------------------------------------------------------------------------------
group: yes
   vars   n     mean       sd  median  trimmed     mad     min      max    range skew kurtosis    se
X1    1 707 15552.34 14552.32 9964.06 13451.03 7883.43 1131.51 63770.43 62638.92 1.18     0.08 547.3
```

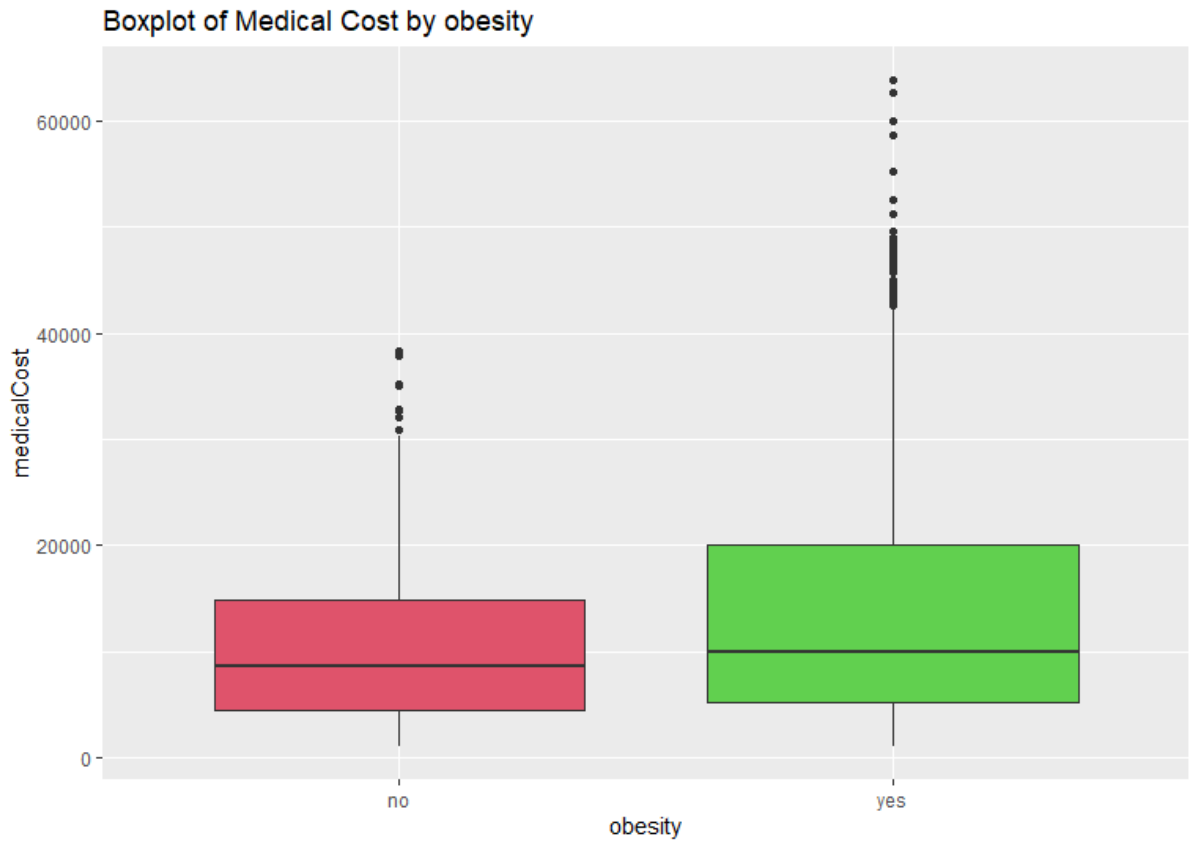


Figure 11: *A Box plot of medical cost by obesity.*

3.1.6.1 REMARK

From table 13 and figure 10 we can see that 631 persons from the data are not smokers and 707 persons from the data set are smokers, which implies that there are higher number of smokers in the data set than those who are not smokers. Whilst from table 14 and figure 11 we can see, although obese and non-obese people have the same median medical costs, their average expenditure differ by almost 5000.

R Code 9.

```
#############################################################################################
###### Descriptive and Visualisation Analysis For BMI ######################################
#############################################################################################

# Create new variable derived from bmi
Ins$obesity <- ifelse(Ins$bmi>=30,"yes","no")

head(Ins)

ob <- table(Ins$obesity)
ob
ob1 <- as.data.frame(ob)
names(ob1)[c(1,2)] <- c("Obesity", "Count")

ob1 %>%
  ggplot(aes(x = Obesity, y = Count, fill = Obesity)) +
  geom_col() +
  theme(legend.position = 'none')

# By obesity status
describeBy(Ins$medicalCost,Ins$obesity)

ggplot(data = Ins,aes(obesity,medicalCost)) + geom_boxplot(fill = c(2:3)) +
  xlab("obesity") + ggtitle("Boxplot of Medical Cost by obesity")
```

The above R code 9 creates a new variable called “Obesity”, performs descriptive analysis, and plots both Bar chart and Box plot.

CHAPTER FOUR

4 REGRESSION MODEL ANALYSIS

4.1 THE REQUIRED MACHINE LEARNING FOR THIS STUDY (QUESTION A)

The required machine learning for this study is a supervised machine learning because the data set of the case study have labels (medical cost as dependent variable labelled as Y and other variables as independent variables labelled as X's).

4.2 CORRELATION BETWEEN VARIABLES (QUESTION C)

Below is the correlation between medical cost and other variables.

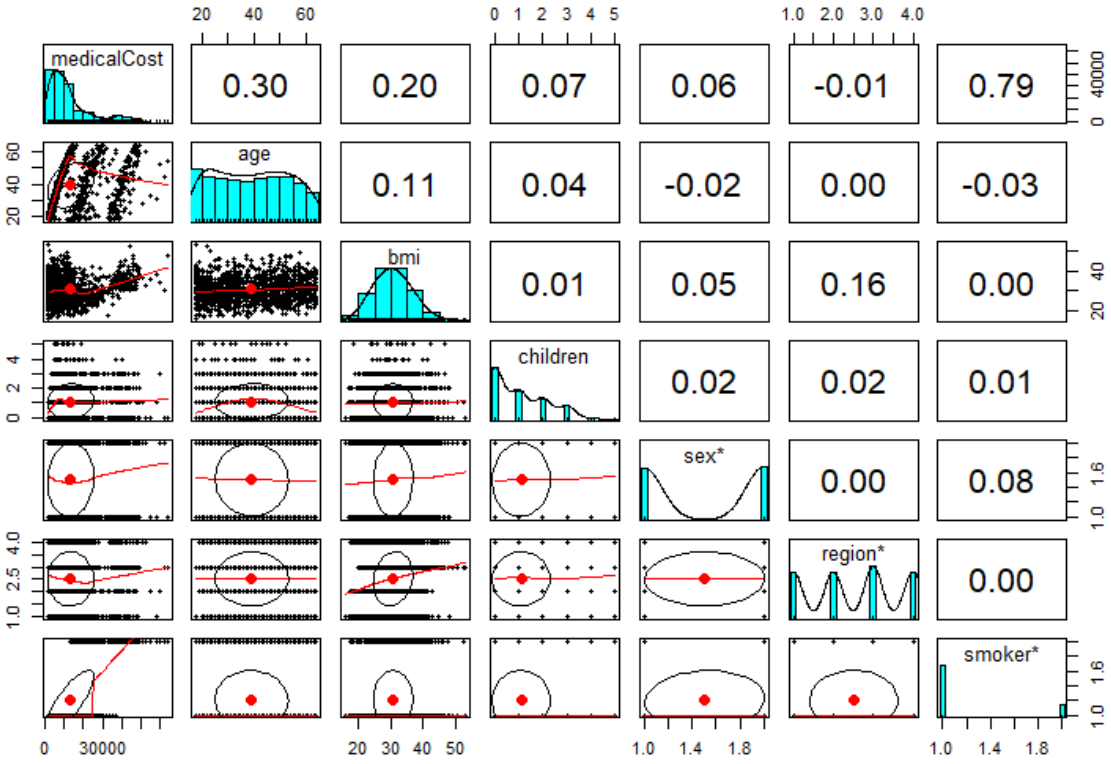


Figure 12: *A Pairs.panel plot showing the correlation between medical cost and other variables.*

R Code 10.

```
##########################################################################################################
#################  Checking Correlation Between Variables  ###############################################
##########################################################################################################

pairs.panels(Ins[c("medicalCost", "age", "bmi", "children",  "sex", "region", "smoker")])
```

The above R code 10 calculates the correlation between medical cost and other variables.

4.2.1 CONCLUSION

From figure 12 it is seen that between all the independent variables (age, bmi, children, sex, region, and smoker) only the smoker variable has a high positive correlation between the medical cost, the age, bmi, children, and sex variables have low positive correlation between the medical cost, and the region variable has negative correlation between the medical cost. Since the relationship between smoker and medical cost variables is high, we can conclude from our findings that smoking have a lot to do with medical cost.

4.3 SIMPLE LINEAR REGRESSION (QUESTION D)

From our findings, the three variables having the highest correlation values with medical cost are smoker, age, and bmi having 0.79, 0.30, and 0.20 respectively as correlation values.

4.3.1 LINEAR REGRESSION ASSUMPTIONS

Before we start any regression analysis, we need to check for the four linear regression assumptions which are normality, homoscedasticity, autocorrelation, and multicollinearity assumptions.

4.3.1.1 NORMALITY ASSUMPTION

Table 15.

```
        Shapiro-Wilk normality test

data:  llm$residuals
W = 0.89894, p-value < 2.2e-16
```

From table 15 normality assumption is failed.

4.3.1.2 HOMOSCEDASTICITY ASSUMPTION

Table 16.

```
        studentized Breusch-Pagan test

data:  llm
BP = 121.74, df = 8, p-value < 2.2e-16
```

From table 16 homoscedasticity assumption is failed.

4.3.1.3 AUTOCORRELATION ASSUMPTION

Table 17.

```
        Durbin-Watson test

data:  llm
DW = 2.0884, p-value = 0.9472
alternative hypothesis: true autocorrelation is greater than 0
```

From table 17 autocorrelation assumption is passed.

4.3.1.4 MULTICOLLINEARITY ASSUMPTION

Table 18.

	GVIF	Df	GVIF^(1/(2*Df))
age	1.016822	1	1.008376
bmi	1.106630	1	1.051965
children	1.004011	1	1.002003
sex	1.008900	1	1.004440
region	1.098893	3	1.015841
smoker	1.012074	1	1.006019

From table 18 multicollinearity assumption is passed.

R Code 11.

```
#########################################################################################################
######################### Assumption Of Linear Regression Checking  ######################################
#########################################################################################################

llm <- lm(medicalCost ~ age + bmi + children + sex + region + smoker, data = Ins)

#############  Normality Assumption Checking  ###################################################

# Perform Shapiro-Wilk Normality Test (W)
shapiro.test(llm$residuals) # Failed assumption

##### Homoscedasticity Assumption  ###########################################################

# Perform Breusch-Pagan Test (BP)
bptest(llm) # Failed Homoscedasticity Assumption

##### Autocorrelation Assumption  ############################################################

# Perform Durbin-Watson test (DW)
dwtest(llm) ## Passed Autocorrelation Assumption

###########   Multicollinearity Assumption ###################################################

# Perform Variance Inflation Factor Test (VIF)
vif(llm)  ##  Passed Multicollinearity Assumption
```

The above R code 11 perform the linear regression assumptions check.

4.3.1.5 CONCLUSION

The autocorrelation and multicollinearity assumptions were met but both normality and homoscedasticity assumptions were not met because the insurance data set contain categorical variables which can neither be normal nor have equal variance across their error terms. So, we can go ahead with our simple linear regression analysis.

4.3.2 SIMPLE LINEAR REGRESSION (SMOKER AND MEDICAL COST)

Table 19.

```
Call:
lm(formula = medicalCost ~ smoker, data = Ins)

Residuals:
   Min     1Q Median     3Q    Max
-19221  -5042   -919   3705  31720

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   8434.3      229.0   36.83   <2e-16 ***
smokeryes    23616.0      506.1   46.66   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 7470 on 1336 degrees of freedom
Multiple R-squared:  0.6198,    Adjusted R-squared:  0.6195
F-statistic:  2178 on 1 and 1336 DF,  p-value: < 2.2e-16
```

We get a straight line y (medical cost) = 8434.3 + 23616 (smoker yes).

4.3.3 SIMPLE LINEAR REGRESSION (AGE AND MEDICAL COST)

Table 20.

```
Call:
lm(formula = medicalCost ~ age, data = Ins)

Residuals:
   Min     1Q Median     3Q    Max
 -8059  -6671  -5939   5440  47829

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3165.9      937.1   3.378 0.000751 ***
age            257.7       22.5  11.453  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 11560 on 1336 degrees of freedom
Multiple R-squared:  0.08941,   Adjusted R-squared:  0.08872
F-statistic: 131.2 on 1 and 1336 DF,  p-value: < 2.2e-16
```

We get a straight line y (medical cost) = 3165.9 + 257.7 (age).

4.3.4 SIMPLE LINEAR REGRESSION (BMI AND MEDICAL COST)

Table 21.

```
Call:
lm(formula = medicalCost ~ bmi, data = Ins)

Residuals:
   Min     1Q Median     3Q    Max
-20956  -8118  -3757   4722  49442

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1192.94    1664.80   0.717    0.474
bmi           393.87      53.25   7.397 2.46e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 11870 on 1336 degrees of freedom
Multiple R-squared:  0.03934,   Adjusted R-squared:  0.03862
F-statistic: 54.71 on 1 and 1336 DF,  p-value: 2.459e-13
```

We get a straight line y (medical cost) = 1192.94 + 393.87 (bmi).

R Code 12.

```
#################################################################################################
#############  Simple Linear Regression (Smoker and Medical Cost)   ##############################
#################################################################################################
slm <- lm(medicalCost ~ smoker, data = Ins)
summary(slm)

#################################################################################################
#############  Simple Linear Regression (Age and Medical Cost)   ################################
#################################################################################################
alm <- lm(medicalCost ~ age, data = Ins)
summary(alm)

#################################################################################################
#############  Simple Linear Regression (BMI and Medical Cost)   ################################
#################################################################################################
blm <- lm(medicalCost ~ bmi, data = Ins)
summary(blm)
```

The above R code 12 execute a simple linear regression of medical cost variable with smoker, age, and bmi variables.

4.3.5 STATISTICAL MODEL EVALUATION (QUESTION E)

To evaluate these models using statistical measures, we shall look at their (the models) residual standard error and adjusted R-square. The residual standard deviation (or residual standard error**)** and the adjusted R-square are measures used to assess how well a linear regression model fits the data. For residual standard error the smaller the value, the better the model fits the data

and for adjusted R-square, the bigger the value, the more involved the independent variable is in predicting the dependent variable. From our simple linear regression models, we can say that the regression model between smoker and medical cost performs better than the rest models because it has the smallest residual standard error value (7470) and the biggest adjusted R-square value (62%).

4.4 MULTIPLE LINEAR REGRESSION (QUESTION F)

Here we shall build two multiple linear regression models, one with the three variables which have the best correlation value with medical cost and the other with all the variables in the insurance data set.

4.4.1 MULTIPLE LINEAR REGRESSION (BMI, AGE, AND SMOKER)

Table 22.

```
Call:
lm(formula = medicalCost ~ bmi + age + smoker, data = Ins)

Residuals:
     Min       1Q   Median       3Q      Max
-12415.4  -2970.9   -980.5   1480.0  28971.8

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -11676.83     937.57  -12.45   <2e-16 ***
bmi            322.62      27.49   11.74   <2e-16 ***
age            259.55      11.93   21.75   <2e-16 ***
smokeryes    23823.68     412.87   57.70   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6092 on 1334 degrees of freedom
Multiple R-squared:  0.7475,    Adjusted R-squared:  0.7469
F-statistic:  1316 on 3 and 1334 DF,  p-value: < 2.2e-16
```

We get a straight line y (medical cost) = -11676.83 + 322.62 (bmi) + 259.55 (age) + 23823.68 (smoker yes).

4.4.2 MULTIPLE LINEAR REGRESSION (BMI, AGE, SMOKER, SEX, CHILDREN, AND REGION)

Table 23.

```
Call:
lm(formula = medicalCost ~ age + bmi + children + sex + region +
    smoker, data = Ins)

Residuals:
     Min       1Q   Median       3Q      Max
-11304.9  -2848.1   -982.1   1393.9  29992.8

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     -11938.5      987.8 -12.086  < 2e-16 ***
age                256.9       11.9  21.587  < 2e-16 ***
bmi                339.2       28.6  11.860  < 2e-16 ***
children           475.5      137.8   3.451 0.000577 ***
sexmale           -131.3      332.9  -0.394 0.693348
regionnorthwest   -353.0      476.3  -0.741 0.458769
regionsoutheast  -1035.0      478.7  -2.162 0.030782 *
regionsouthwest   -960.0      477.9  -2.009 0.044765 *
smokeryes        23848.5      413.1  57.723  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6062 on 1329 degrees of freedom
Multiple R-squared:  0.7509,    Adjusted R-squared:  0.7494
F-statistic: 500.8 on 8 and 1329 DF,  p-value: < 2.2e-16
```

We get a straight line y (medical cost) = -11938.5 + 339.2 (bmi) + 256.9 (age) + 23848.5 (smoker yes) + 475.5 (children) – 131.3 (sex male) – 353 (region northwest) – 1035 (region southeast) – 960 (region southwest).

R Code 13.

```
#########################################################################################################
#############  Multiple Linear Regression (BMI, AGE, AND SMOKER)   ##############################
#########################################################################################################

mlm1 <- lm(medicalCost ~ bmi + age + smoker, data = Ins)
summary(mlm1)

#########################################################################################################
########  Multiple Linear Regression (BMI, AGE, SEX, CHILDREN, REGION, AND SMOKER)   #############
#########################################################################################################

ins <- Ins

ins$sex <- as.factor(ins$sex)
ins$smoker <- as.factor(ins$smoker)
ins$region <- as.factor(ins$region)

mlm2 <- lm(medicalCost ~ age + bmi + children + sex + region + smoker, data = ins)
summary(mlm2)
```

The above R code 13 is used to execute the two multiple linear regression models.

4.4.3 MODEL EVALUATION AND COMPARISON

In the first model, all the independent variables (age, bmi, and smoker (yes)) were highly statistically significant in the model in explaining the dependent variable (medical cost), while in the second model, some independent variables (age, bmi, children, and smoker (yes)) were highly statistically significant, two independent variables (sex (male) and region (northwest)) are not statistically significant, and two independent variables (region (southeast) and region (southwest)) are statistically significant in explaining the dependent variable (medical cost). Comparing the two models, the second model performed better looking at some statistical measures such as the residual standard error and the adjusted R-square which is slightly better than that of the first model. Again, in a nutshell, in real life, there are a lot of variables that can affect or influence medical cost, so it is only normal that the second model performs better than the first model because the second model was performed with more independent variables than the first model.

4.5 VARIABLE IMPORTANCE

Here we shall perform a test to check for the importance of the explanatory variables in explaining the response variable. The Shapley value regression will be used for this task. Shapley value regression is a variance decomposition method by means of computing the marginal contribution of each attribute (variable) in a regression model. The last multiple linear regression will be used in this test be it contains all the variables in the insurance data set. Below is the Shapley value regression result and a plot.

Table 24.

region	age	bmi	children	sex	smoker
0.003348641	0.088865634	0.032102191	0.003418587	0.001590924	0.621587058

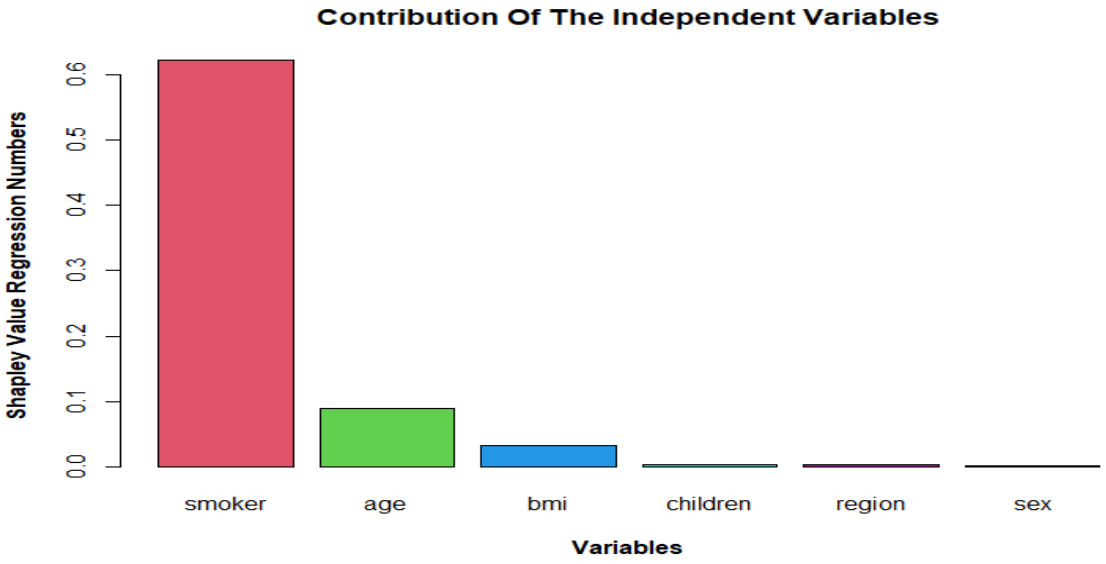


Figure 13: *A Bar chart showing the Shapley value regression numbers of variables.*

Note that the sum of Shapley value regression is the R-square (which explain the contribution of the explanatory variables in predicting the response variable in the model) value of the regression model. Below is the sum of the Shapley value regression (which is equivalent to the R-square value).

Table 25.

```
[1] 0.750913
```

From table 24 and figure 13 the Shapley value regression test shows that the smoker variable had the highest contribution in explaining the response variable (medical cost) with 0.621587058 and the sex variable had the lowest contribution with 0.001590924. Also, we can see that the sum of the Shapley value regression numbers is equal to the R-square value of our regression model of interest.

R Code 14.

```
#########################################################################################################
#################  Shapley Value Regression   ###########################################################
#########################################################################################################

mlm2_shapley <-calc.relimp(mlm2, type = "lmg")
mlm2_shapley$lmg

barplot(sort(mlm2_shapley$lmg, decreasing = TRUE),col=c(2:10),
        main="Contribution Of The Independent Variables",xlab="Variables",
        ylab="Shapley Value Regression Numbers",font.lab=2)

sum(mlm2_shapley$lmg)
```

The above R code 14 performs a Shapley value regression, plot a bar chart of the results, and sum the results.

CHAPTER FIVE

5 OVERALL CONCLUSION (QUESTION G)

This study categorised under the supervised machine learning constitute of linear regression models with dependent variable as medical cost and independent variables as age, sex, bmi (body mass index), children (numbers of children), smoker, and region. A correlation analysis has been carried out and age, bmi, and smoker variables was discovered to have the best correlation (relationship) with the dependent variable (medical cost). A simple linear regression was performed each with the three variables with the best correlation with medical cost and the model with the smoker variable preforms better. Also, two multiple linear regression analysis was performed, one with the three best correlated variables (age, bmi, and smoker), and the other with all the variables in the data set. We find out that the model implemented with all the variables in the data set tends to slightly perform better than the model implemented with the three best correlated variables. A Shapley value regression was executed using the last multiple linear regression model to see the contribution of all the explanatory variables in explaining and prediction the response variable (medical cost) and we find out that the “smoker variable” plays a very high role compared with others, in explaining the response variable (medical cost).

We then conclude that in the health insurance industry, those who are smokers will most likely have to pay more health insurance fee because smokers are highly prone to a lot of diseases like heart and kidney diseases, thereby increasing the medical costs the insurance company will incur on him or her.

MACHINE LEARNING AND OPTIMISATION

ASSIGNMENT TWO (MACHINE LEARNING AND OPTIMISATION USING PYTHON)

JOHNSON OHAKWE

4111678

CT7205

SCHOOL OF COMPUTING AND ENGINEERING

UNIVERSITY OF GLOUCESTERSHIRE, UNITED KINGDOM

MODULE TUTOR:

BHUPESH MISHRA

JULY 2022

TABLE OF CONTENTS

CHAPTER ONE

1 INTRODUCTION

Census has been seen as one of the most media through which information about a population can be ascertained. Census is traditionally known as counting each person in a geographical area or environment but in the modern world or in recent times, censuses have been conducted and used not just to know the total population there is in a country but other information such as the sex of a person, their age, their educational level, their race, their native or original country, their marital status, their relationship, their family, their income per annual amongst others.

This study will be focused on the USA census data with the aim of predicting if an adult makes more than $50,000 a year or not (income) based on the number of variables available in the data set.

1.2 AIM AND OBJECTIVES

The main aim of this study is to model and predict (forecast) the income of the USA census data. Below are the objectives of this study

(a) Load and explore the data (note your observations).

(b) Use appropriate methods to handle categorical data.

(c) Investigate and train at least 5 ML models including Classification (to predict if an individual going to earn more $50,000 annually or not), Clustering and Neural Networks. You are free to choose any ML algorithms.

(d) Optimise your models, evaluate the models, and compare the models' results as: i. How does optimisation improve the performance of the model? Which parameter do you use for optimisation? ii. Compare the results among the models of similar types (e.g. If you are using two classification models compare their performances).

(e) State your overall conclusions for this task.

1.3 DEFINING ATTRIBUTES IN THE DATA SET

- **Age:** The age of an individual.
- **Work Class:** Employment status of an individual.
- **Fnlwgt:** Final weight. In other words, this is the number of people the census believes the entry represents.
- **Education:** The highest level of education achieved by an individual.
- **Education Number:** The highest level of education achieved in numerical form.
- **Marital Status:** Marital status of an individual.

- **Occupation:** The general type of occupation of an individual.
- **Relationship:** Represents what this individual is relative to others.
- **Sex:** The biological sex of the individual.
- **Capital Gain:** Capital gains for an individual.
- **Capital Loss:** Capital loss for an individual.
- **Hours Per Week:** The hours an individual has reported to work per week.
- **Native Country:** Country of origin for an individual.
- **Income:** Whether an individual makes more than $50,000 annually.

CHAPTER TWO

2 METHODOLOGY

This chapter talks about the methodology for statistical machine learning modelling of the census data set collected.

2.1 SOURCE OF THE DATA

The data set used for this study is secondary data of the US Census dataset from the Census Bureau which is publicly available online.

2.2 CLUSTERING MODEL

A process of organizing objects into groups such that data points in the same groups are like the data points in the same group. A cluster is a collection of objects where these objects are similar and dissimilar to the other cluster. Here we shall discuss the methodology of the K-Means Algorithm, Hierarchy algorithm, and Density-Based Spatial Clustering of Applications with Noise algorithm (DBSCAN).

2.2.1 K-MEANS ALGORITHM

K-Means clustering is a type of unsupervised learning. The main aim of this algorithm is to discover groups in data and the number of groups is addressed as K. This clustering algorithm separates data into the most appropriate group based on the information the algorithm already has. Data is separated in K different clusters, which are usually chosen to be far enough apart from each other spatially, in Euclidian_Distance, to be able to produce effective data mining results. Each cluster has a centre, called the centroid**,** and a data point is clustered into a certain cluster based on how close the features are to the centroid. K-means algorithm iteratively minimizes the distances between every data point and its centroid to find the most optimal solution for all the data points.

Given two points $A(x_2, x_1)$ and $\mathrm{B}(\mathrm{y}_2, \mathrm{y}_1)$ the Euclidean distance is estimated using

$$D(A,B) = \sqrt{(x_2 - x_1) + (y_2 - y_1)} \tag{2.0}$$

The centroid of each cluster v_j is estimated using

$$v_j = \frac{1}{m} \sum_{x \in v_j} X \tag{2.1}$$

The objective function for the K-means clustering algorithm is the squared error function:

$$J = \sum_{i=1}^{k} \sum_{j=1}^{n} \left(\| x_i - v_j \| \right)^2 = 1 \quad (2.2)$$

Where:

$\| x_i - v_j \|$ is the Euclidian distance between a point, x_i, and a centroid, v_j, iterated over all *k* points in the i^{th} cluster, for all n clusters. In simpler terms, the objective function attempts to pick centroids that minimize the distance to all points belonging to its respective cluster so that the centroids are more symbolic of the surrounding cluster of data points.

2.2.2 HIERARCHY ALGORITHM

Hierarchical clustering is just *K*-means, but instead of there being a fixed number of clusters, the number changes in every iteration. When the number increases, we talk about divisive clustering: all data instances start in one cluster, and splits are performed in each iteration, resulting in a hierarchy of clusters. Agglomerative clustering, on the other hand, is a bottom-up approach: each instance is a cluster at the beginning, and clusters are merged in every iteration. With the use of either method, the hierarchy will have *N*−1 levels (Hastie et al., 2008)

The step involves are:

- Calculating Euclidean distance in equation (2.0).
- Using the linkage criteria.

(1) Complete linkage:

$$\max \{d(a, b) : a \in A, b \in B\} \quad (2.3)$$

(2) Single linkage:

$$\min \{d(a, b) : a \in A, b \in B\} \quad (2.4)$$

(3) Average linkage:

$$d(c_A, c_B) \quad (2.5)$$

Where: c_A and c_B are the centroids of the clusters A and B and d is the chosen metric.

2.2.3 DENSITY-BASED SPATIAL CLUSTERING OF APPLICATIONS WITH NOISE ALGORITHM (DBSCAN)

DBSCAN (Density-Based Spatial Clustering of Applications with Noise algorithm) is a clustering algorithm or model used for the purpose of discovering clusters of random shapes. Its runtime is defined to be O(nlog(n)). Mathematically, it is based on the formal idea of density-reachability for the k-dimensional points in a region. DBSCAN (Density-Based Spatial

Clustering of Applications with Noise algorithm) is frequently used on Noisy data for clustering operations.

The 2 major parameters associated with DBSCAN are

- Epsilon (Eps) and
- minimum Points (MinPts).

$$Eps = \varepsilon_0 \tag{2.6}$$

minimum *minPts* can be derived from the number of dimensions *D* in the data set, as *minPts* ≥ *D* + 1

2.3 CLASSIFICATION MODEL

Classification is an arranged set of related categories used to group data according to its similarities. Here we shall discuss the methodology of the logistic regression algorithm and the support vector machine algorithm (SVM).

2.3.1 LOGISTIC REGRESSION ALGORITHM

Logistic regression model is typically used to model a binary dependent variable with the help of logistic function. Another name for the logistic function is a sigmoid function and is given by:

$$F(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} \tag{2.7}$$

This function enables the logistic regression model to squeeze the values from (-k, k) to (0, 1). Logistic regression is mainly used for binary classification tasks; however, it can be used for multi-class classification.

2.3.2 SUPPORT VECTOR MACHINE ALGORITHM (SVM)

The support vector machine (SVM) is one of the most effective and powerful out-of-the-box supervised machine learning algorithms, unlike many other machine learning algorithms such as neural networks, you don't have to do a lot of tweaks to obtain good results with SVM. This algorithm consists of some basic linear algebra, vectors, length of the vectors, direction of the vectors, dot product of the vectors, and hyperplane.

Vectors:

$$V = \overrightarrow{OA} \tag{2.8}$$

Length of the vectors:

$$|V| = \sqrt{O^2 + A^2} \tag{2.9}$$

The direction of the vectors:

$$\theta = tan^{-1}\left({}^{O}/_{A}\right) \tag{2.10}$$

The dot product of the vectors:

$$O \cdot A = |O||A|cos\theta \tag{2.11}$$

And the hyperplane of the linear data.

2.4 NEURAL NETWORK MODEL

A neural network is a series of algorithms that endeavours to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so, the network generates the best possible result without needing to redesign the output criteria (definition by Investopedia, 2021). Here we shall look at the methodology of the Multi-Layer Perceptron Classifier (MLP).

2.4.1 MULTI-LAYER PERCEPTRON ALGORITHM (MLP)

A Multi-Layer Perceptron (MLP) is a composition of an input layer, at least one hidden layer of the linear threshold units (LTUs) and an output layer of the linear threshold units (LTUs). If an MLP has two or more hidden layer, it is called a deep neural network (DNN).

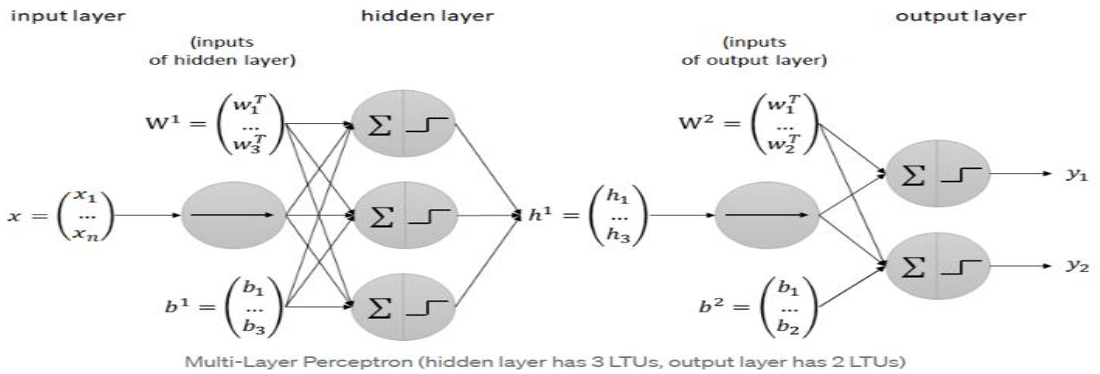


Figure 14: *This shows the methodology the multi-layers perceptron Algorithm.*

The calculations for MLP are the same as for a perceptron, just that there exist more layers of the linear threshold units (LTUs) to combine until one reaches the output y:

$$h^1 = step(z^1) = step(W^1 \cdot x + b^1)$$

$$y = step(z^2) = step(W^2 \cdot h^1 + b^2)$$

Figure 15: *This shows the methodology of the step process of both the perceptron and multi-layers perceptron algorithm.*

2.5 CONFUSION MATRIX

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix looks like the square matrix below:

	Actually Positive (1)	*Actually Negative* (2)
Pr*edicted Positive* (1)	*True Positives* (*TPs*)	*False Positives* (*FPs*)
Pr*edicted Negative* (2)	*False Negatives* (*FNs*)	*True Negatives* (TN *s*)

(2.12)

Accuracy (all correct / all) = TP + TN / TP + TN + FP + FN (2.13)

Where: TP is True Positive, TN is True Negative, FP is False Positive, and FN is False Negative.

CHAPTER THREE

3 DATA EXPLORATION, VISUALISATION, LABEL ENCODER, AND DESCRIPTIVE ANALYSIS (QUESTION A)

Here we shall do some data exploration, visualisation, and descriptive analysis of our data. Let's start by loading required libraries and importing the data set.

Python Code 1.

```
# Load Libraries

import math
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn import svm
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.cluster import AgglomerativeClustering
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
import scipy.cluster.hierarchy as sch
from sklearn.metrics import silhouette_score
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import DBSCAN
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

The above Python code 1 is used to import libraries used for this study.

Python Code 2.

```
# Import Data and View Data

cen_dat = pd.read_csv (r'C:\Users\ohakw\Desktop\Assignment\Machine Learning and Optimsation CT7205\CensusDB.csv')
```

The above Python code 2 is used for importing and loading of the census data set.

Let’s check if there are missing values in the data set.

Table 26.

```
age               0
workclass         0
fnlwgt            0
education         0
education-num     0
marital-status    0
occupation        0
relationship      0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
income            0
dtype: int64
```

From table 1 it is obvious that our data set have no missing values.

Python Code 3.

```
#Checking for missing values

cen_dat.isnull().sum()
```

The above Python code 3 is used to check for missing values in the data set.

3.1 EXPLORATORY DATA ANALYSIS

Below is a head view of the census data.

Table 27.

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	Female	0	4356	40	United-States	<=50K
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	Female	0	4356	18	United-States	<=50K
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Female	0	4356	40	United-States	<=50K
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	Female	0	3900	40	United-States	<=50K
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	Female	0	3900	40	United-States	<=50K

From table 2 we notice that the census data contain age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, sex, capital-gain, capital-loss, hours-per-week, and income as variables.

Python Code 4.

```
# Head view of the Census data set

cen_dat.head()
```

The above Python code 4 is used to perform a head view of the data.

Let's see the dimension of the data.

Table 28.

```
(32561, 14)
```

From table 3, we can see that the census data has 32561 number of rows and 14 number of columns.

Python Code 5.

```
# Checking the dimension of the Census data set

cen_dat.shape
```

The above Python code 5 execute the checking of the dimension of the data.

Let's see the class or structure of the attributes in the data set.

Table 29.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education-num   32561 non-null  int64
 5   marital-status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   sex             32561 non-null  object
 9   capital-gain    32561 non-null  int64
 10  capital-loss    32561 non-null  int64
 11  hours-per-week  32561 non-null  int64
 12  native-country  32561 non-null  object
 13  income          32561 non-null  object
dtypes: int64(6), object(8)
memory usage: 3.5+ MB
```

We can see from table 4 that the census data contain 6 continuous variables as integers which are age, fnlwgt, education-num, capital-gain, capital-loss, and hours-per-week variables, and it 8 categorical variables as object (strings or characters) which are workclass, education, marital-status, occupation, relationship, sex, native-country, and income variables.

Python Code 6.

```
# Checking the information of the Census data set

cen_dat.info()
```

The above Python code 6 is used for checking the class or structure the attributes in the census belongs to.

We shall play around with the data; we change the column name of the native-country attribute to country. Below is a head view of the new data.

Table 30.

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	sex	capital-gain	capital-loss	hours-per-week	country	income
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	Female	0	4356	40	United-States	<=50K
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	Female	0	4356	18	United-States	<=50K
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Female	0	4356	40	United-States	<=50K
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	Female	0	3900	40	United-States	<=50K
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	Female	0	3900	40	United-States	<=50K

From table 5, the native-country attribute name has been changed to country.

Python Code 7.

```
cen_dat.columns=['age','workclass','fnlwgt','education','education-num','marital-status','occupation','relationship',
                 'sex','capital-gain','capital-loss','hours-per-week','country','income']
cen_dat.head()
```

The above Python Code 7 implements the attribute name change and the head view of the new data set.

3.2 DATA VISUALISATION

Here we shall do separate visualisation for both the continuous and categorical variables. Below are the categorical and the continuous variables.

Table 31.

```
Continous variables in the dataset are:
age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week,

Categorical variable in the dataset are:
workclass,  education,  marital-status,  occupation,  relationship,  sex,  country,  income,
```

Table 6 shows the separation of the attributes into categorical and continuous variables.

Python Code 8.

```
# Separating the continuous variables from the categorical variables

print('Continous variables in the dataset are: ')
cont = []
for i in cen_dat.columns:
  if cen_dat[i].dtype == 'int64':
    cont.append(i)
    print(i, end = ', ')
print('\n\nCategorical variable in the dataset are: ')
catg=[]
for i in cen_dat.columns:
  if cen_dat[i].dtype == 'O':
    catg.append(i)
    print(i, end = ',  ')
```

The above Python code 8 implements the separation of the attribute into continuous and categorical variables.

3.2.1 DATA VISUALISATION FOR THE CONTINUOUS VARIABLES

Here we shall only visualise the continuous variables which are age, fnlwgt, education-num, c apital-gain, capital-loss, and hours-per-week variables. Below are some visualisations of the c ontinuous variables.

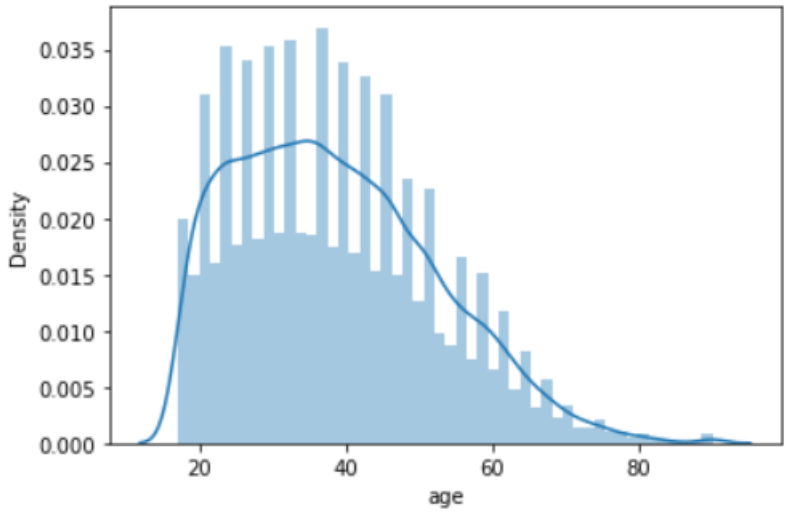


Figure 16: *A plot of age variable.*

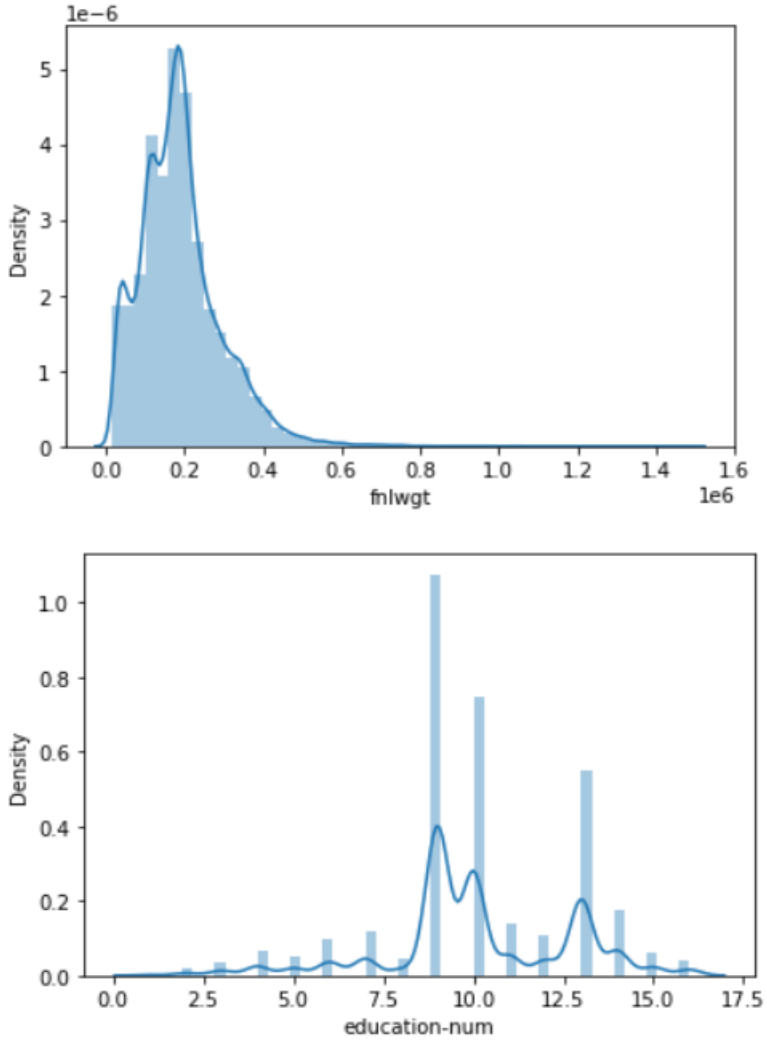


Figure 17: *A mixed plot of fnlwgt and education-num variables.*

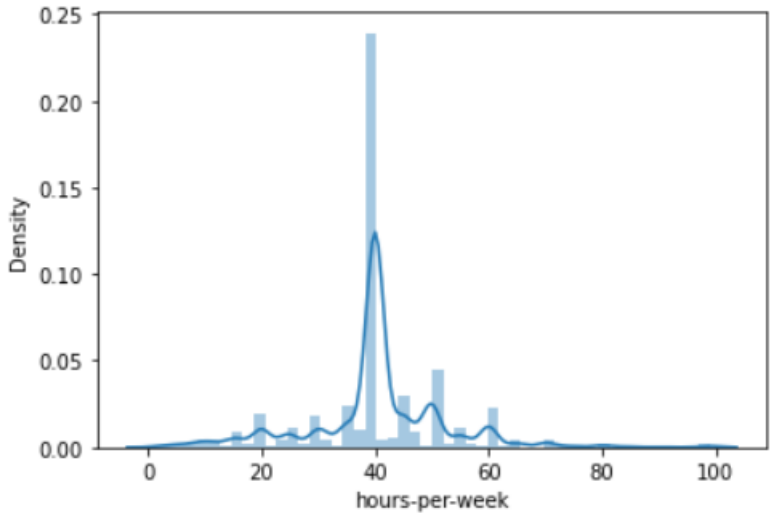


Figure 18: *A plot of hours-per-week variable.*

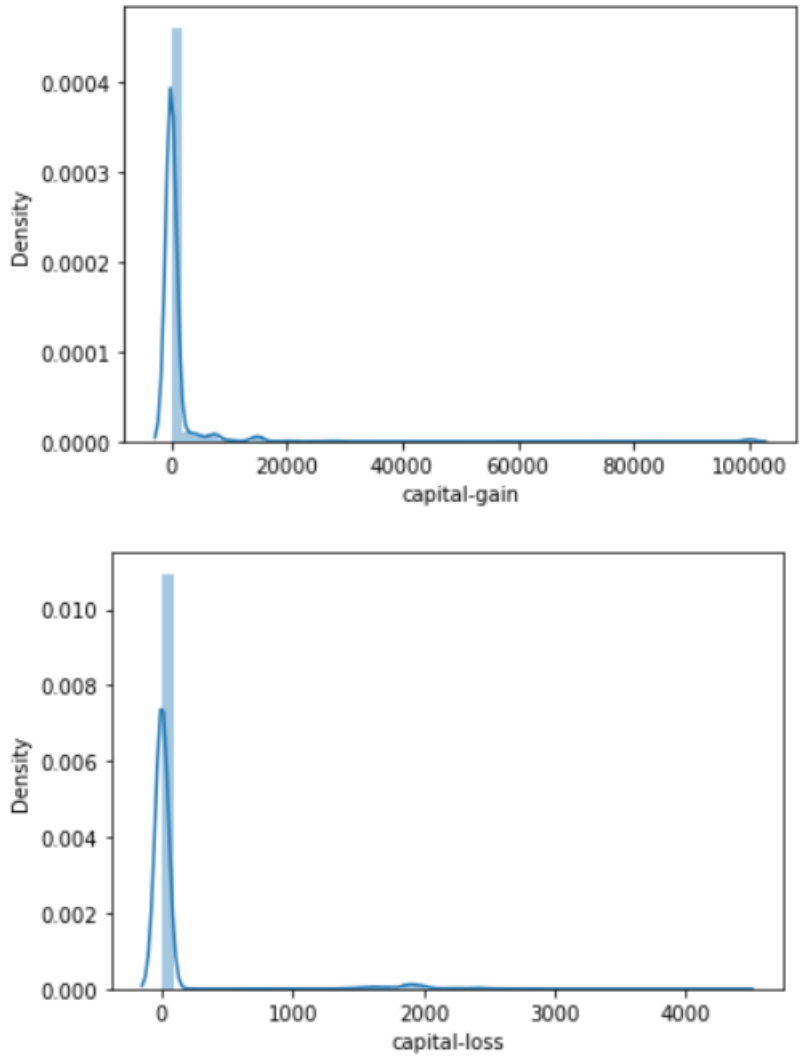


Figure 19: *A mixed plot of both capital-gain and capital-loss variables.*

From the above plots we can observe that:

The fnlwgt variable is just a census number and it is not useful for this study (income prediction), the education-num variable is an ordered-encoding of education variable, and there is presence of outliers in the capital-gain and capital-loss variables (columns).

Python Code 9.

```
# Visualisation for the continuous variables

for i in cont:
  sns.distplot(cen_dat[i])
  plt.show()
```

The above Python code 9 implements the plot of all the continuous variables.

Python Code 10.

```
# Copying the old data into a new container

census = cen_dat.copy(deep = True)
```

```
# Our Final continuous variables.

cont=['age','education-num','capital-gain','capital-loss','hours-per-week']
```

The above Python code 10 is used to copy our original data into a new container called “census” and the code is also used to save or store the continuous variables we are going to be using in this study. Note that the fnlwgt variable was dropped because it is not relevant for the model prediction we want to perform for this study.

3.2.2 DATA VISUALISATION FOR THE CATEGORICAL VARIABLES

Here we shall only visualise the categorical variables which are workclass, education, marital-status, occupation, relationship, sex, country, and income variables. Below are some visualisa tions of the categorical variables with respect to income.

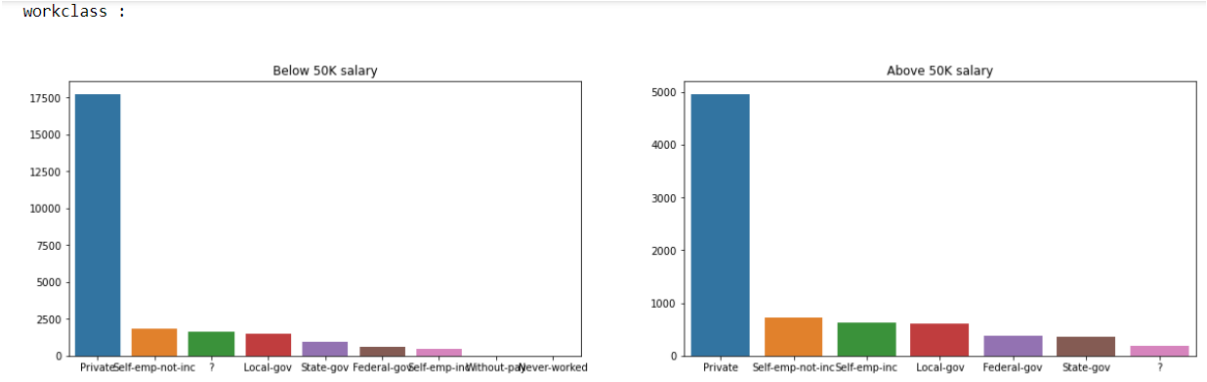


Figure 20: *A mixed Bar plot of workclass for below or equal to $50k and above $50k.*

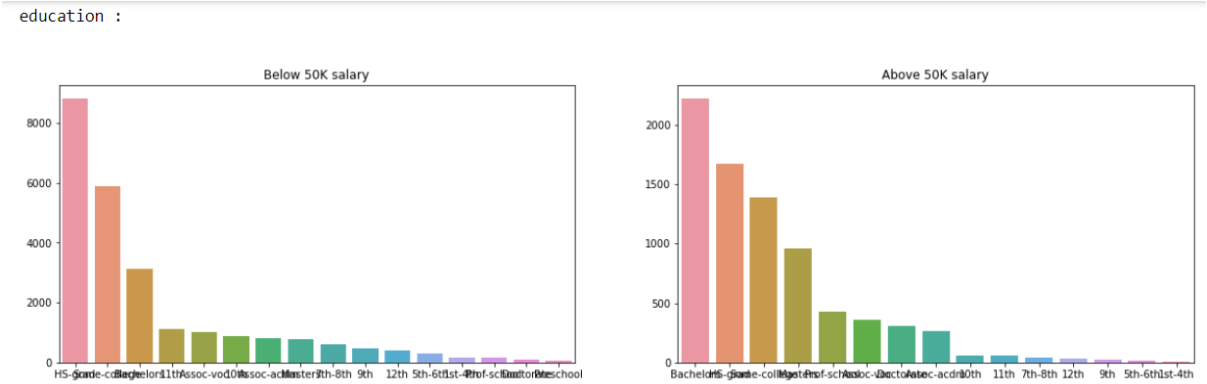


Figure 21: *A mixed Bar plot of education for below or equal to $50k and above $50k.*

marital-status :

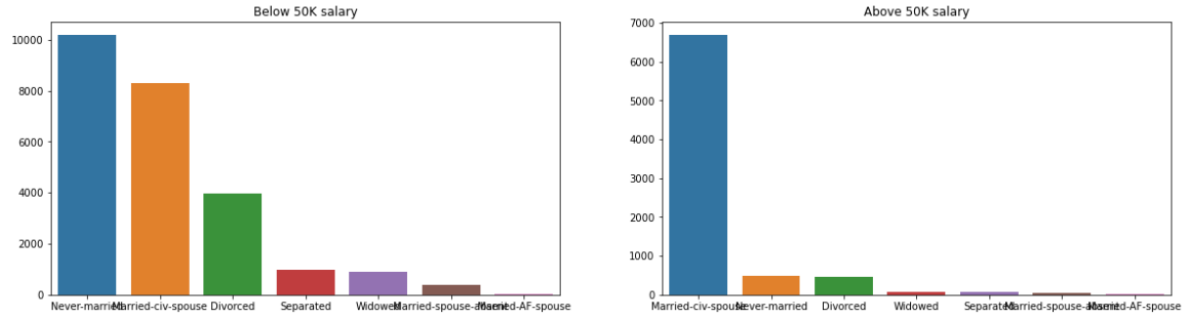


Figure 22: *A mixed Bar plot of marital-status for below or equal to $50k and above $50k.*

occupation :

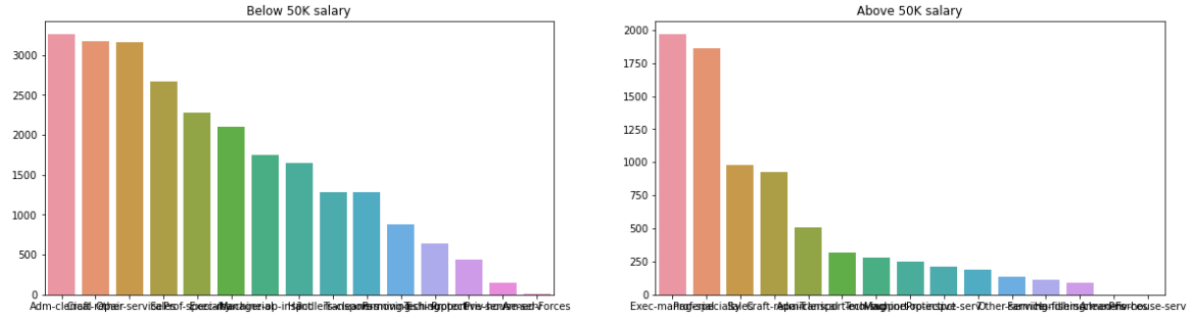


Figure 23: *A mixed Bar plot of occupation for below or equal to $50k and above $50k.*

relationship :

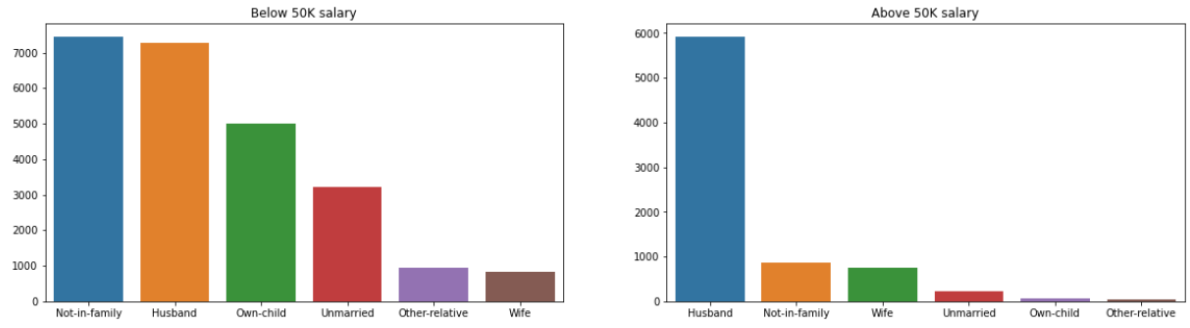


Figure 24: *A mixed Bar plot of relationship for below or equal to $50k and above $50k.*

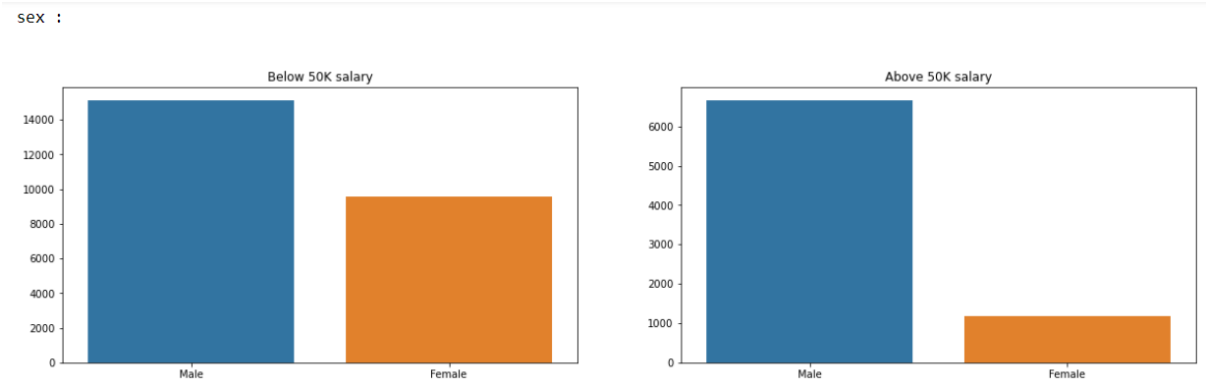


Figure 25: *A mixed Bar plot of sex for below or equal to $50k and above $50k.*

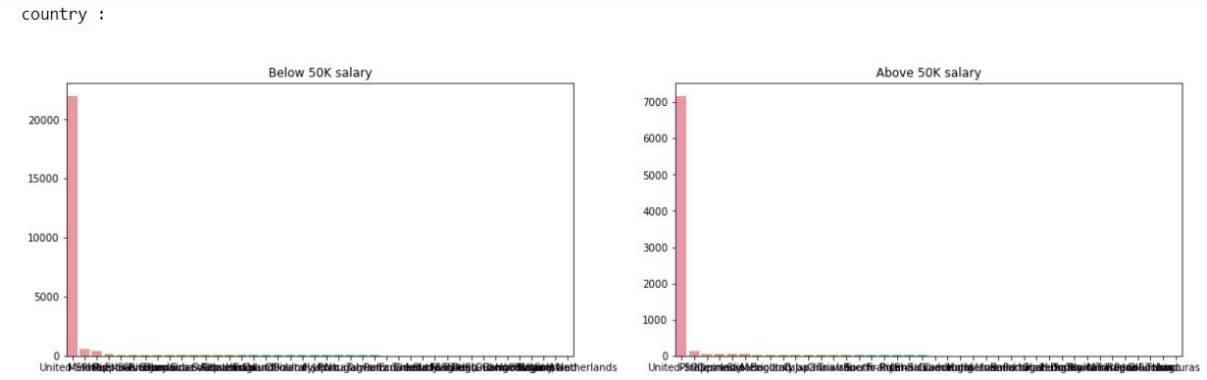


Figure 26: *A mixed Bar plot of country for below or equal to $50k and above $50k.*

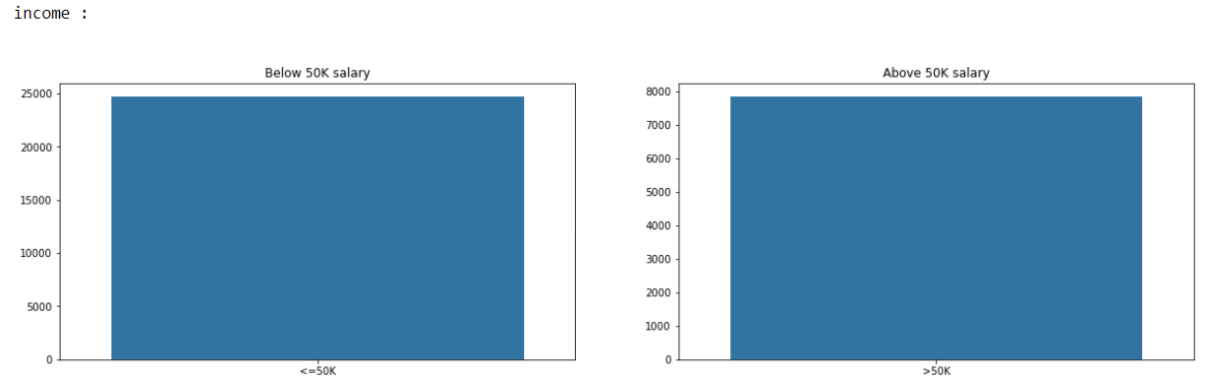


Figure 27: *A mixed Bar plot of income for below or equal to $50k and above $50k.*

From the visualization of the categorical variables, we noticed and conclude that:

We shall drop workclass, marital-status, relationship, and country because they are not relevant in explaining income variable, we shall drop education as education-num (continous variable) is already present.

Python Code 11.

```
# Creating a Visualisation function

def catg_vis(dt,cg):
  plt.figure(figsize = (20,5))
  a = dt[dt['income'] == '<=50K'][cg].value_counts() # Below 50k salary
  b = dt[dt['income'] == '>50K'][cg].value_counts() # Above 50k salary
  plt.subplot(1,2,1)
  plt.title('Below 50K salary')
  sns.barplot(a.index,a.values)
  plt.subplot(1,2,2)
  plt.title('Above 50K salary')
  sns.barplot(b.index,b.values)
  plt.show()
```

```
# Visualisation

for i in catg:
  print(i, ':\n')
  catg_vis(cen_dat,i)
```

The above Python code 11 create a visualisation function and perform a for loop for plotting all the categorical variables.

3.3 LABEL ENCODER (QUESTION B)

Here the categorical variables will be converted or transformed from object or strings, or characters to numbers called zeros and ones (if they are binary). First, let's see a head view of the final data we shall be using in this study.

Table 32.

	age	education-num	capital-gain	capital-loss	hours-per-week	occupation	sex	income
0	90	9	0	4356	40	?	Female	<=50K
1	82	9	0	4356	18	Exec-managerial	Female	<=50K
2	66	10	0	4356	40	?	Female	<=50K
3	54	4	0	3900	40	Machine-op-inspct	Female	<=50K
4	41	10	0	3900	40	Prof-specialty	Female	<=50K

Table 7 is our final for this study containing 8 variables.

Python Code 12.

```
# The final categorical variables we shall using in this study

catg = ['occupation', 'sex', 'income']
```

```
# Our new data set

census = census[cont+catg]
census.head()
```

The above Python code 12 is used to store or save our final categorical variables and the other combine both the stored continuous and categorical variables together and perform a head view of our final data.

The next to do is transform the occupation, sex, and income variables to categorical variables with numbers. Firstly, we convert them from objects to categories.

Table 33.

```
age                int64
education-num      int64
capital-gain       int64
capital-loss       int64
hours-per-week     int64
occupation      category
sex             category
income          category
dtype: object
```

In table 8, all the object class variables have been converted or transformed to category class.

Then we move forward to converting them into numbers of zeros and ones (variables with binary observations). Below is a head view of the proper categorical variables and continuous variable (as our final data for this study).

Table 34.

	age	education-num	capital-gain	capital-loss	hours-per-week	occupation	sex	income
0	90	9	0	4356	40	0	0	0
1	82	9	0	4356	18	4	0	0
2	66	10	0	4356	40	0	0	0
3	54	4	0	3900	40	7	0	0
4	41	10	0	3900	40	10	0	0

Table 9 contains our final data set for this study with properly encoded categorical variables.

Python Code 13.

```
# Coding the categorical variables in numbers (One hot Encoding / Label Encoder)
# Converting the objects in the data to category

census['occupation'] = census['occupation'].astype('category')
census['sex'] = census['sex'].astype('category')
census['income'] = census['income'].astype('category')

census.dtypes

# To view the numbers in the categorical variables
# Head view

census['occupation'] = census['occupation'].cat.codes
census['sex'] = census['sex'].cat.codes
census['income'] = census['income'].cat.codes

census.head()
```

The above Python code 13 implement the encoding of the categorical variables.

3.4 DESCRIPTIVE ANALYSIS

Here we will perform some descriptive statistical analysis of our final data set. Below are the basic descriptive analysis of our final data set.

Table 35.

	age	education-num	capital-gain	capital-loss	hours-per-week	occupation	sex	income
count	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	10.080679	1077.648844	87.303830	40.437456	6.572740	0.669205	0.240810
std	13.640433	2.572720	7385.292085	402.960219	12.347429	4.228857	0.470506	0.427581
min	17.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000	3.000000	0.000000	0.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000	7.000000	1.000000	0.000000
75%	48.000000	12.000000	0.000000	0.000000	45.000000	10.000000	1.000000	0.000000
max	90.000000	16.000000	99999.000000	4356.000000	99.000000	14.000000	1.000000	1.000000

Table 10 contain some basic statistics of our data such as count, mean, standard deviation, minimum value, maximum value, 25%, 50%, and 75% percentile.

Python Code 14.

```
census.describe()
```

The above Python code 14 is used to execute the basic descriptive analysis of our data set.

Let’s look at the correlation or let’s see the relationship between variables. Below is a heat map plot with the correlation values of all the variables.

Figure 28: *A heat map plot with correlation values.*

From figure 15, we can see that there seems to be no relationship between variables as their correlation values are very small and some are negative. The best correlation we have in the data is the correlation between education-num and income with 0.34. We can then conclude that there are no strong relationships between variables.

Python Code 15.

```
# Calculate correlations between variables

corrmat = census.corr()
fig, ax = plt.subplots(figsize = (10, 7))
sns.heatmap(corrmat, annot = True)
```

The above Python code 15 is used to plot a heatmap plot with correlation values of variables.

CHAPTER FOUR

4 CLUSTERING, CLASSIFICATION, AND NEURAL NETWORK MODEL (QUESTION C AND D)

Here we shall look at clustering model, classification model, and neural network model.

4.1 CLUSTERING MODEL

Here we shall discuss the three clustering algorithms namely k-means, hierarchy, and the DBSCAN algorithms.

4.1.1 K-MEANS ALGORITHM

The first thing we need do here is to drop the response variable (income) in our data, as what we do in clustering is to predict the response variable. Below is a head view of our new data after drop the income variable.

Table 36.

	age	education-num	capital-gain	capital-loss	hours-per-week	occupation	sex
0	90	9	0	4356	40	0	0
1	82	9	0	4356	18	4	0
2	66	10	0	4356	40	0	0
3	54	4	0	3900	40	7	0
4	41	10	0	3900	40	10	0

Table 11 is our new data set without the income variable.

Python Code 16.

```
# Since clustering is an unsupervised method, the response variable is usually not there
# Therefore, we will drop the response column (income)

census1 = census.copy(deep = True)
census1.drop(['income'], axis = 1, inplace = True)
census1.head()
```

The above Python code 16 is used to execute the dropping of the income variable and the head view of the new data set.

Next is to scale, standardise, or normalise our data before we fit the model to it. Below is the scaled data set.

Table 37.

```
array([[ 3.76961234, -0.42005962, -0.14592048, ..., -0.03542945,
        -1.55428326, -1.42233076],
       [ 3.18311167, -0.42005962, -0.14592048, ..., -1.81720429,
        -0.60838662, -1.42233076],
       [ 2.01011032, -0.03136003, -0.14592048, ..., -0.03542945,
        -1.55428326, -1.42233076],
       ...,
       [ 0.10398314, -0.42005962, -0.14592048, ..., -0.03542945,
         0.10103586,  0.70307135],
       [ 1.42360965, -0.42005962, -0.14592048, ..., -0.03542945,
        -1.3178091 , -1.42233076],
       [-1.21564337, -0.42005962, -0.14592048, ..., -1.65522476,
        -1.3178091 ,  0.70307135]])
```

Table 12 is the scaled data we shall be using to fit a k-means cluster algorithm.

Python Code 17.

```
#Perform feature scaling

sc = StandardScaler()
census2 = sc.fit_transform(census1)
census2
```

The above Python code 17 is used to scale our data set.

We went ahead to fit the k-means cluster algorithm to the scaled data using k as 3. Below are the cluster centres of the fitted model.

Table 38.

```
array([[-0.13223143, -0.02908213, -0.07021054, -0.21272614, -0.33785141,
        -0.12017165, -1.42233076],
       [ 0.05050883, -0.01067351,  0.04582813, -0.21605415,  0.15316397,
         0.05431079,  0.70091191],
       [ 0.22416041,  0.35288381, -0.14592048,  4.50430853,  0.23664811,
         0.0865253 ,  0.21435785]])
```

Table 13 contain the centres of the three clusters of our model.

Python Code 18.

```
# Get the cluster centres

kmeans.cluster_centers_
```

The above Python code 18 is used to extract the centres of the three clusters of our model.

Let calculate the model accuracy to ascertain the performance of our model in the predicting the response variable. Below the calculated accuracy of our model.

Table 39.

```
0.47087005927336384
```

From table 14, our model accuracy is at 47%. For the sake of this study, we need to optimise our model.

Python Code 19.

```
# Apply K-means clustering using sklearn.cluster

actual = census['income']
kmeans = KMeans(n_clusters = 3, random_state = 42)
y_kmeans = kmeans.fit_predict(census2)

# Model Acuracy

kmeans_performance = metrics.accuracy_score(y_kmeans , actual)
kmeans_performance
```

The above Python code 19 is used to fit the k-mean model and to calculate the accuracy level.

Let look at the predicted verses the actual income values. Below is a table showing this predicted income against the actual income.

Table 40.

	Predicted Income	Actual Income
0	2	0
1	2	0
2	2	0
3	2	0
4	2	0
...	...	...
32556	1	0
32557	0	0
32558	1	1
32559	0	0
32560	1	0

32561 rows × 2 columns

Table 15 shows the predicted income against the actual income.

Python Code 20.

```
# Comparing the predicted income with the actual income

com = pd.DataFrame({'Predicted Income':y_kmeans, 'Actual Income': actual})
com
```

The above Python code 20 is used to create a data frame of the predicted income against the actual income.

Below is the cluster plot of the k-means algorithm.

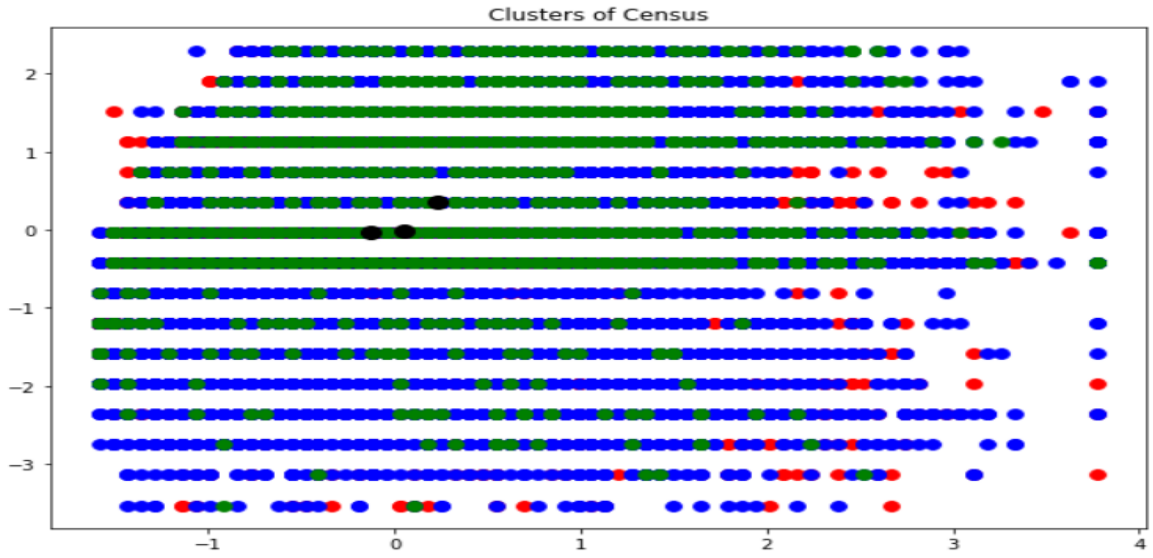


Figure 29: *The cluster plot of the k-means algorithm.*

Python Code 21.

```
# Plot the points and calculated cluster centers in a scatter plot which differentiate clusters by colour

plt.figure(figsize = (10, 7))
plt.scatter(census2[y_kmeans == 0, 0], census2[y_kmeans == 0, 1], s = 60, c = 'red', label = 'Cluster 1')
plt.scatter(census2[y_kmeans == 1, 0], census2[y_kmeans == 1, 1], s = 60, c = 'blue', label = 'Cluster 2')
plt.scatter(census2[y_kmeans == 2, 0], census2[y_kmeans == 2, 1], s = 60, c = 'green', label = 'Cluster 3')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 100, c = 'black', label = 'Centroids')
plt.title('Clusters of Census')
plt.show()
```

The above Python code 21 implement the cluster plot of the k-means algorithm.

4.1.1.1 OPTIMISATION OF THE K-MEANS ALGORITHM

To optimise the k-means algorithm we first need to run the elbow method test for best value of k. Below is the elbow method plot.

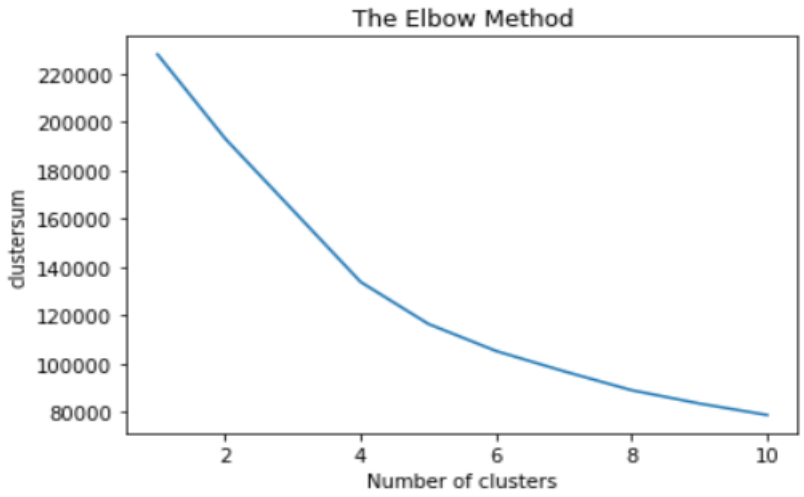


Figure 30: *An Elbow method plot.*

From the Elbow method plot, we select k value to be 2. We will fit another k-means model using the k as 2 and calculate the new model accuracy.

Python Code 22.

```
# Elbow method test for optimal K selection

from sklearn.cluster import KMeans
clustersum = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(census2)
    clustersum.append(kmeans.inertia_)
plt.plot(range(1, 11), clustersum)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('clustersum')
plt.show()
```

The above Python code 22 execute the Elbow method plot for selecting the best value of k.

We fit again another k-means model with k = 2. Below is the optimise model accuracy.

Table 41

```
0.4987868922944627
```

From table 16, we can notice an improve model accuracy which is 50%. The model have been optimise because the current model accuracy (50%) after optimisation is better than the previous model accuracy (47%).

Python Code 23.

```
# Apply K-means clustering using sklearn.cluster (Optimising)

kmeans = KMeans(n_clusters = 2, random_state = 42)
y_kmeans1 = kmeans.fit_predict(census2)

# Model Acuracy (After Optimisation)

kmeans_performance1 = metrics.accuracy_score(y_kmeans1 , actual)
kmeans_performance1
```

The above Python code 23 is used to fit the new k-means algorithm and to calculate the model accuracy.

Again, let's look at the predicted income against the actual income after optimising the model.

Table 42.

	Predicted Income	Actual Income
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
32556	1	0
32557	0	0
32558	1	1
32559	0	0
32560	1	0

32561 rows × 2 columns

The predicted income in table 17 is better than the previous one when the model has not been optimised.

Python Code 24.

```
# Comparing the predicted income with the actual income

com1 = pd.DataFrame({'Predicted Income':y_kmeans1, 'Actual Income': actual})
com1
```

The above Python code 24 is used to create a data frame of the predicted income against the actual income after optimising the model.

Below is the centres values and the plot of the cluster after optimisation.

Table 43.

```
array([[-0.12761426, -0.01834553, -0.07136642, -0.06454352, -0.33443324,
        -0.11498434, -1.42233076],
       [ 0.06291478,  0.00904448,  0.03518418,  0.03182044,  0.16487809,
         0.05668814,  0.70121969]])
```

Table 18 contain the centres values of the k-mean model after optimisation.

Python Code 25.

```
# Get the cluster centres (After Optimisation)

kmeans1.cluster_centers_
```

The above Python code 25 is used to extract the centre values from the k-means model.

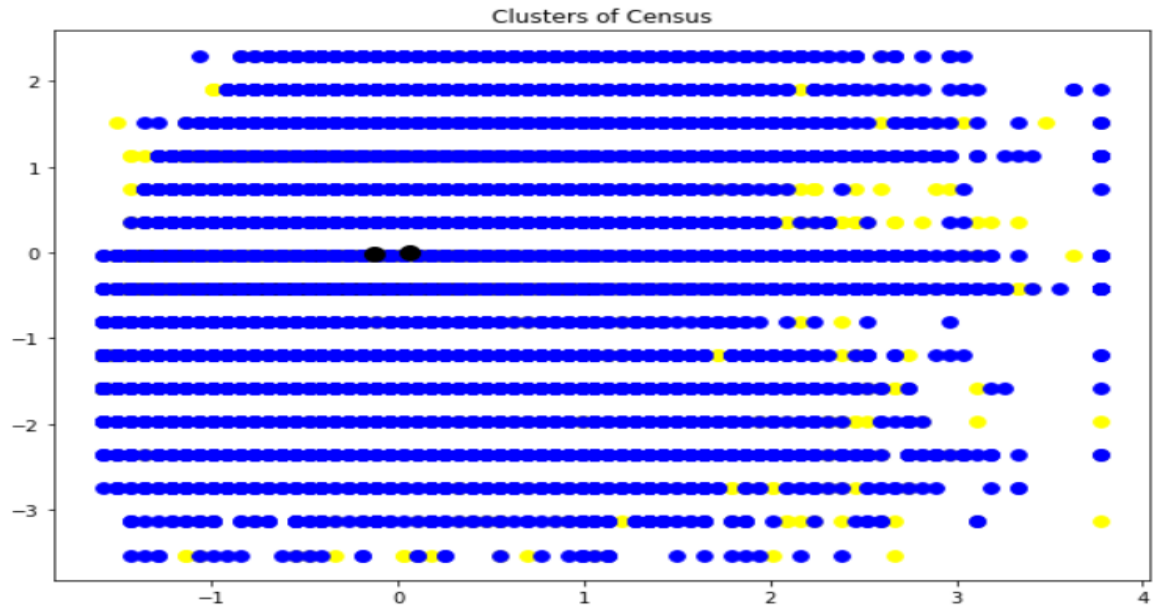


Figure 31: *The cluster plot of the k-means algorithm after optimisation.*

Python Code 26.

```
# Plot the points and calculated cluster centers in a scatter plot which differentiate clusters by colour (After Optimisation)

plt.figure(figsize = (10, 7))
plt.scatter(census2[y_kmeans1 == 0, 0], census2[y_kmeans1 == 0, 1], s = 60, c = 'yellow', label = 'Cluster 1')
plt.scatter(census2[y_kmeans1 == 1, 0], census2[y_kmeans1 == 1, 1], s = 60, c = 'blue', label = 'Cluster 2')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 100, c = 'black', label = 'Centroids')
plt.title('Clusters of Census')
plt.show()
```

The above Python code 26 is used for plotting the clusters of the model after optimisation.

4.1.2 HIERARCHY ALGORITHM

Firstly, we shall look at the hierarchy plot and select a value for k and fit our model. Below is the plot.

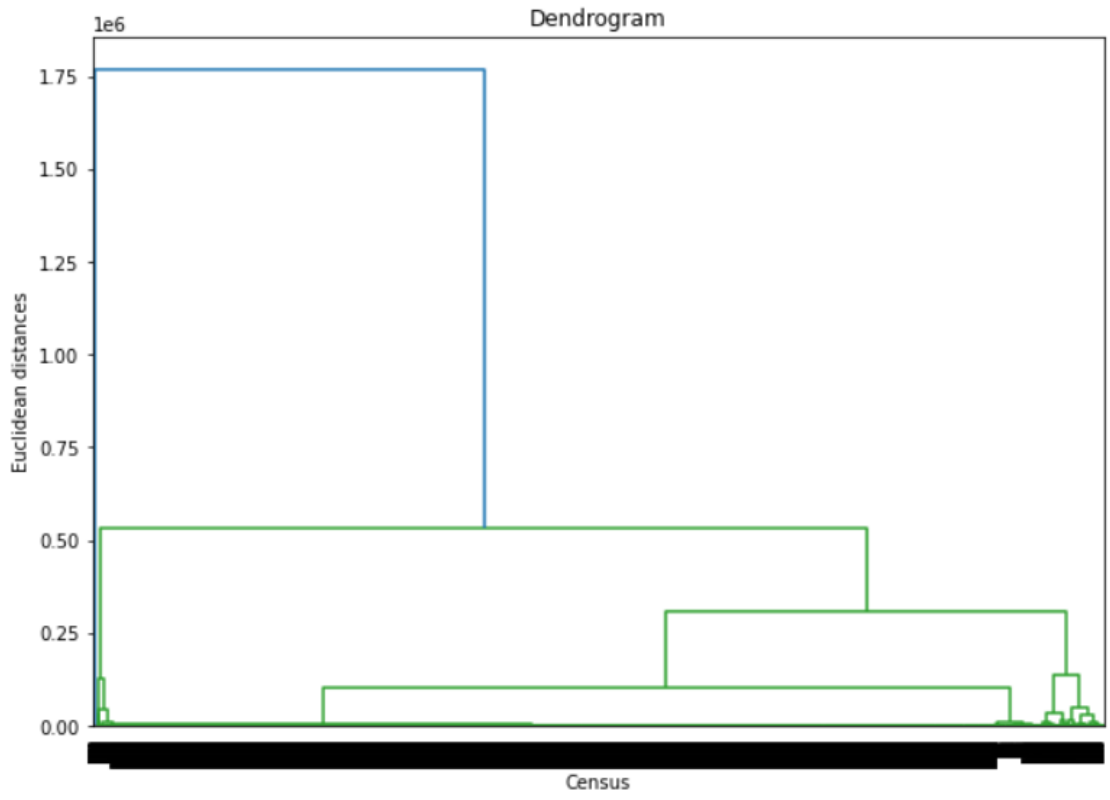


Figure 32: *A hierarchy plot.*

Python Code 27.

```
plt.figure(figsize = (10, 7))

dendrogram = sch.dendrogram(sch.linkage(census1, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Census')
plt.ylabel('Euclidean distances')
plt.show()
```

The above Python code 27 is used for plotting the dendrogram diagram.

We will now fit the model with k value as 3 and calculate the model accuracy. Below is the fitted model accuracy.

Table 44.

```
0.4910168606615276
```

From table 19 we can see that the model accuracy is 49%. We shall try to optimise this model to get a better model accuracy.

Python Code 28.

```
# Perform Agglomerative Clustering

cluster = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage = 'ward')
cl = cluster.fit_predict(census2)

# Model Accuracy

hcl_performance = metrics.accuracy_score(cl , actual)
hcl_performance
```

The above Python code 28 is used to fit the model and to calculate the model accuracy.

Let's look at the predicted income against the actual income.

Table 45.

	Predicted Income	Actual Income
0	2	0
1	2	0
2	2	0
3	2	0
4	2	0
...	...	...
32556	0	0
32557	1	0
32558	0	1
32559	1	0
32560	0	0

32561 rows × 2 columns

Table 20 contains the predicted income against the actual income.

Python Code 29.

```
# Comparing the predicted income with the actual income

hcl_com = pd.DataFrame({'Predicted Income':cl, 'Actual Income': actual})
hcl_com
```

The above Python code 29 is used to create a data frame of predicted income against the actual income.

Below is the cluster plot of the model.

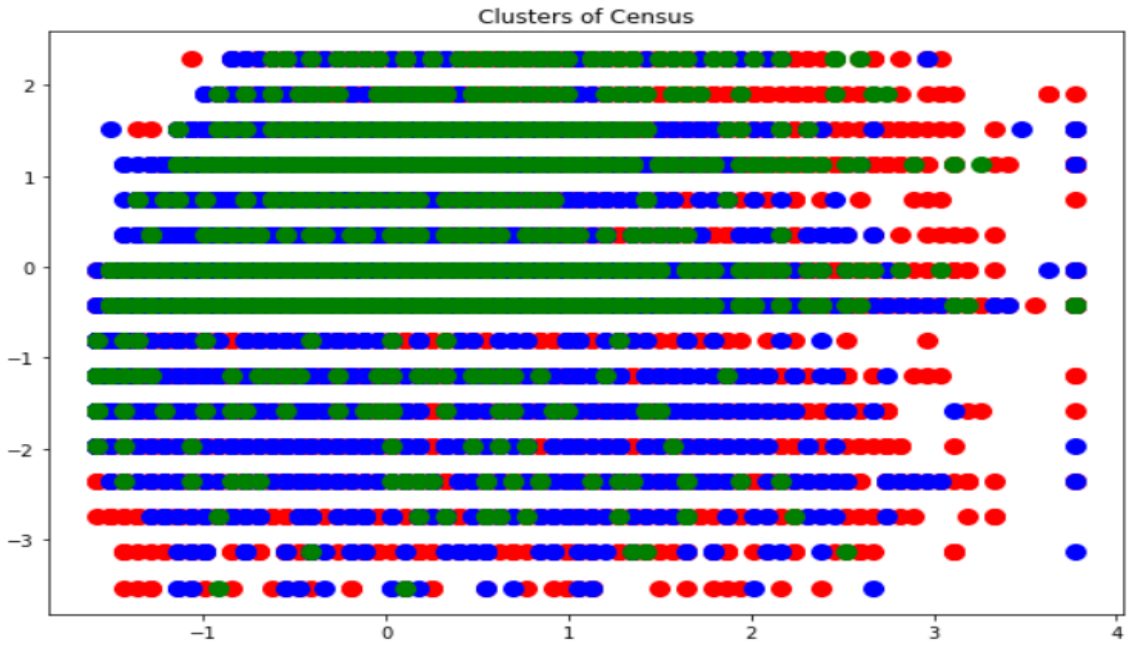


Figure 33: *A cluster of hierarchy algorithm.*

Python Code 30.

```
plt.figure(figsize = (10, 7))
plt.scatter(census2[cl == 0, 0], census2[cl == 0, 1], s = 100, c = 'red', label = 'cluster 1')
plt.scatter(census2[cl == 1, 0], census2[cl == 1, 1], s = 100, c = 'blue', label = 'cluster 2')
plt.scatter(census2[cl == 2, 0], census2[cl == 2, 1], s = 100, c = 'green', label = 'cluster 3')
plt.title('Clusters of Census')
plt.show()
```

The above Python code 30 is used to visualise the clusters in the model.

4.1.2.1 OPTIMISATION OF THE HIERARCHY ALGORITHM

Tuning the value of k to be 2 just as we did in the previous clustering model, might help in optimising our model accuracy. Using 2 as the value of k, we fit another model. Below is the model accuracy.

Table 46..

```
0.5129449341236448
```

From table 21 we can notice that the model accuracy has increase to 51% better than the previous model accuracy 49%.

Python Code 31.

```
# Perform Agglomerative Clustering (Optimising)

cluster = AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean', linkage = 'ward')
cl1 = cluster.fit_predict(census2)

# Model Accuracy (After Optimisation)

hcl_performance1 = metrics.accuracy_score(cl1 , actual)
hcl_performance1
```

The above Python code 31 is used to fit the model and to calculate the model accuracy after optimisation.

Again, let's look at the model plot and the predicted income against the actual income after optimising the model.

Table 47.

	Predicted Income	Actual Income
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
32556	0	0
32557	1	0
32558	0	1
32559	1	0
32560	0	0

32561 rows × 2 columns

Table 22 shows a better predicted income than the previous one.

Python Code 32.

```
# Comparing the predicted income with the actual income

hcl_com1 = pd.DataFrame({'Predicted Income':cl1, 'Actual Income': actual})
hcl_com1
```

The above Python code 32 is used to create a data frame of predicted income against actual income.

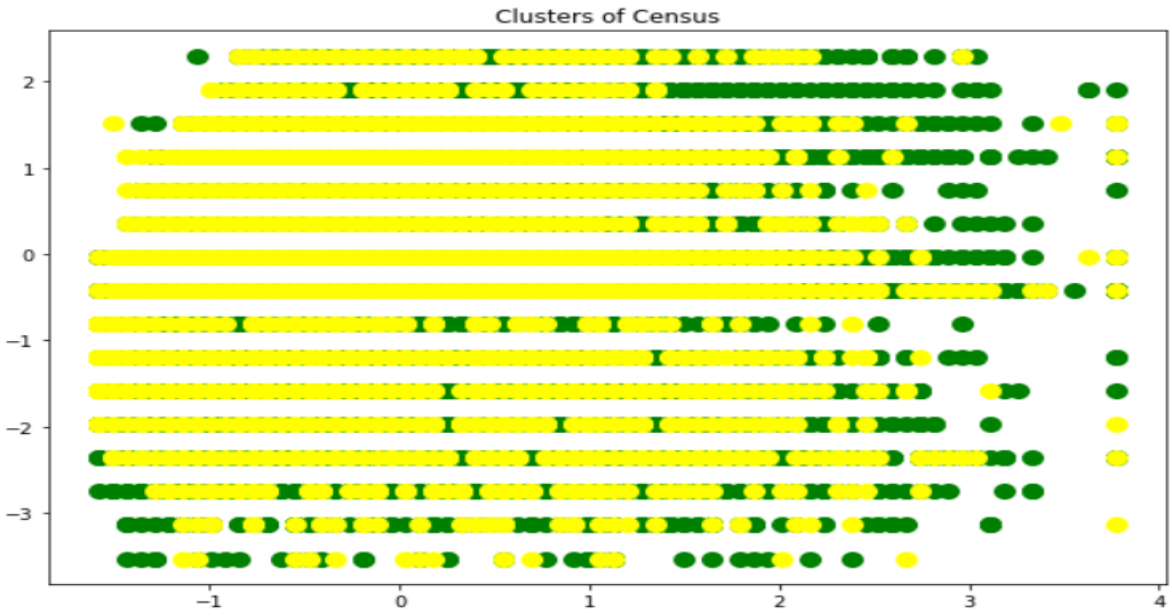


Figure 34: *The cluster plot of the model.*

Python Code 33.

```
plt.figure(figsize = (10, 7))
plt.scatter(census2[cl1 == 0, 0], census2[cl1 == 0, 1], s = 100, c = 'green', label = 'cluster 1')
plt.scatter(census2[cl1 == 1, 0], census2[cl1 == 1, 1], s = 100, c = 'yellow', label = 'cluster 2')
plt.title('Clusters of Census')
plt.show()
```

The above Python code 33 is used to plot the clusters of the model after optimising the model.

4.1.3 DENSITY-BASED SPATIAL CLUSTERING OF APPLICATIONS WITH NOISE ALGORITHM (DBSCAN)

Here, before we fit model, we first need to get the calculate the distance and visualise it. Below is a plot of the distance.

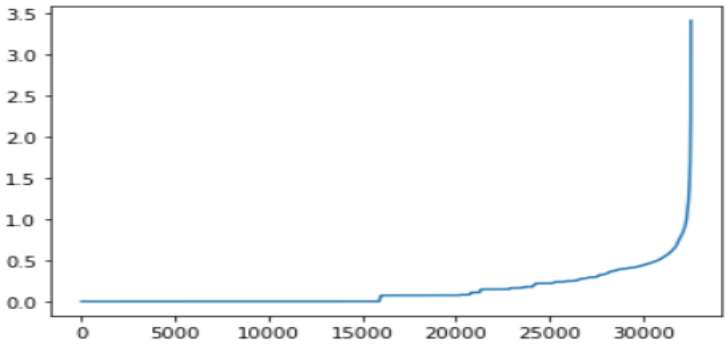


Figure 35: *A line plot of the distance.*

Python Code 34.

```
# Trying DBSCAN

neigh = NearestNeighbors(n_neighbors = 3)
nbrs = neigh.fit(census2)
distances, indices = nbrs.kneighbors(census2)
```

```
# Distance

distances = np.sort(distances, axis = 0)
distances = distances[:, 1]
plt.plot(distances)
```

The above Python code 34 is used for calculating distance and for plot the line plot.

Next is to fit the DBSCAN model, check how many clusters was created by the model, check for model accuracy fixing eps = 2, and min_samples = 3 in model. From the output, we discovered that 5 clusters were formed by the DBSCAN model. Below is the model accuracy.

Table 48.

```
0.49989250944381314
```

From table 23 we can see that the model accuracy is 50%. For this study, we shall further perform optimisation for the DBSCAN model.

Python Code 35.

```
# Fitting the DBSCAN

dbscancluster = DBSCAN(eps = 2, min_samples = 3)
dbscancluster.fit(census2)
clusters = dbscancluster.labels_

# Checking for number of clusters

len(set(clusters))

# Model Accuracy

dbscan_performance = metrics.accuracy_score(clusters , actual)
dbscan_performance
```

The above Python code 35 is used to fit the model, to check for the number clusters in the model, and to calculate model accuracy.

Let’s look at the clusters plot and the predicted income against the actual income.

Table 49.

	Predicted Income	Actual Income
0	-1	0
1	-1	0
2	-1	0
3	-1	0
4	-1	0
...	...	...
32556	0	0
32557	1	0
32558	0	1
32559	1	0
32560	0	0

32561 rows × 2 columns

Table 24 is the predicted income against the actual income.

Python Code 36.

```
# Comparing the predicted income with the actual income

dbscan_com = pd.DataFrame({'Predicted Income':clusters, 'Actual Income': actual})
dbscan_com
```

The above Python code 36 is used to create a data frame of predicted income against actual income.

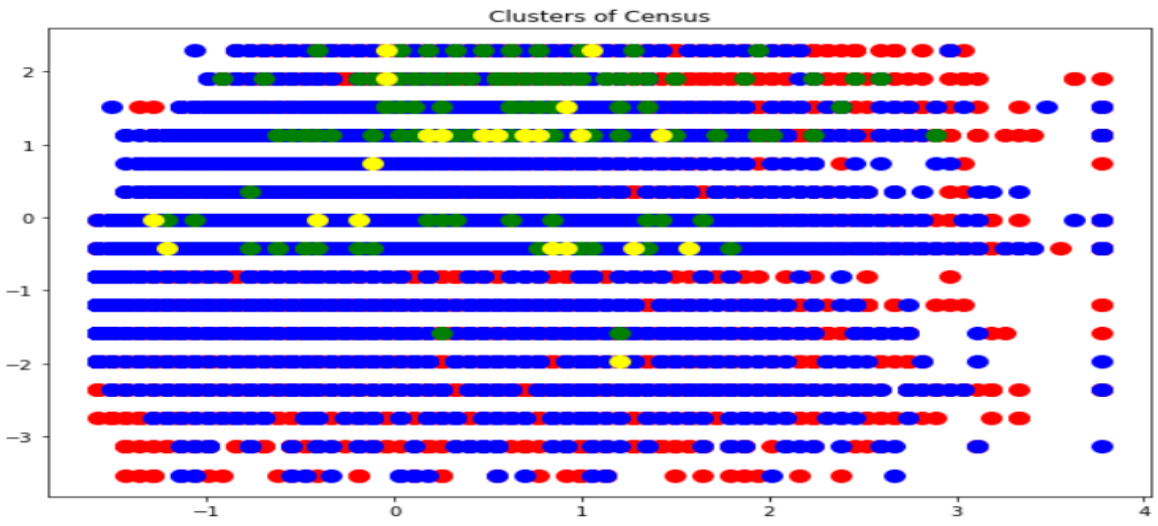


Figure 36: *The clusters plot of the DBSCAN model.*

Python Code 37.

```
plt.figure(figsize = (10, 7))
plt.scatter(census2[clusters == 0, 0], census2[clusters == 0, 1], s = 100, c = 'red', label = 'cluster 1')
plt.scatter(census2[clusters == 1, 0], census2[clusters == 1, 1], s = 100, c = 'blue', label = 'cluster 2')
plt.scatter(census2[clusters == 2, 0], census2[clusters == 2, 1], s = 100, c = 'green', label = 'cluster 3')
plt.scatter(census2[clusters == 3, 0], census2[clusters == 3, 1], s = 100, c = 'yellow', label = 'cluster 4')
plt.scatter(census2[clusters == 4, 0], census2[clusters == 4, 1], s = 100, c = 'purple', label = 'cluster 5')
plt.title('Clusters of Census')
plt.show()
```

The above Python code 37 is used to execute the DBSCAN model plot.

4.1.3.1 OPTIMISATION OF THE DENSITY-BASED SPATIAL CLUSTERING OF APPLICATIONS WITH NOISE ALGORITHM (DBSCAN)

To optimise the DBSCAN model we tune the parameters by setting eps = 7 and min_samples = 5 before fitting the model. After tuning and fitting the DBSCAN model, the new model create 2 clusters and below is the model accuracy.

Table 50.

```
0.7640735849636068
```

From table 25 we can see that there was a huge model improvement, the model accuracy of 76% is quite high compared to the previous DBSCAN model before optimisation.

Python Code 38.

```
# Fitting the DBSCAN (Optimising)

dbscancluster1 = DBSCAN(eps = 7, min_samples = 5)
dbscancluster1.fit(census2)
clusters1 = dbscancluster1.labels_

# Checking for number of clusters

len(set(clusters1))

# Model Accuracy (After Optimisation)

dbscan_performance1 = metrics.accuracy_score(clusters1 , actual)
dbscan_performance1
```

The above Python code 38 is used to fit the model, to check the number of clusters in the model, and to calculate the model accuracy.

Again, let’s look at the plot and the predicted income against the actual income.

Table 51.

	Predicted Income	Actual Income
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
32556	0	0
32557	0	0
32558	0	1
32559	0	0
32560	0	0

32561 rows × 2 columns

The prediction in table 26 is far better than the previous one in the DBSCAN model.

Python Code 39.

```
# Comparing the predicted income with the actual income

dbscan_com1 = pd.DataFrame({'Predicted Income':clusters1, 'Actual Income': actual})
dbscan_com1
```

The above Python code 39 is used to create a data frame of the predicted income against the actual income.

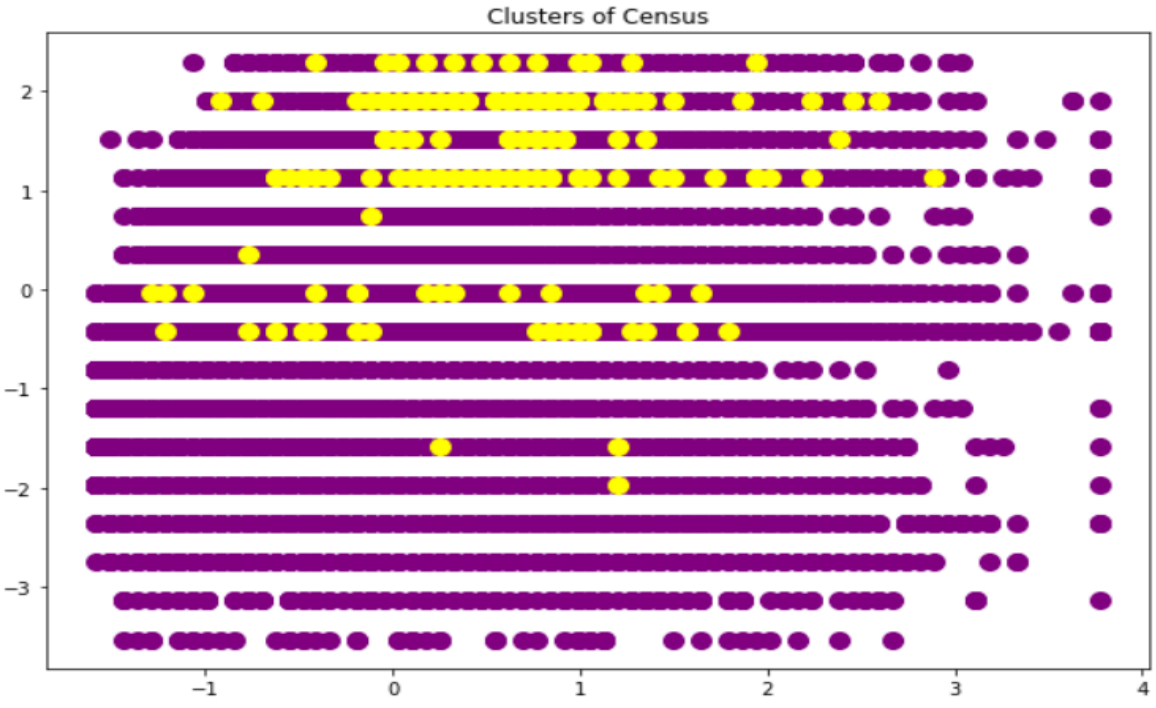


Figure 37: *The clusters plot of the DBSCAN model after optimisation.*

Python Code 40.

```
plt.figure(figsize = (10, 7))
plt.scatter(census2[clusters1 == 0, 0], census2[clusters1 == 0, 1], s = 100, c = 'purple', label = 'cluster 1')
plt.scatter(census2[clusters1 == 1, 0], census2[clusters1 == 1, 1], s = 100, c = 'yellow', label = 'cluster 2')
plt.title('Clusters of Census')
plt.show()
```

The above Python code 40 is used to execute the DBSCAN model plot after optimisation.

4.1.4 CONCLUSION FOR THE CLUSTERING MODELS

The three clustering algorithms that have been studied and discussed above here are K-means algorithm, Hierarchy algorithm, and the DBSCAN algorithm. Their model has been fitted with our data set, setting the number of clusters (k) as 3 for both k-means and hierarchy algorithms, and for the DBSCAN algorithm, its parameters (eps and min_samples) was set to 2 and 3 respectively, producing 5 clusters and their (the three algorithms) model accuracy were calculated. We went further to optimise the three algorithms by tuning their parameters, for K-means and hierarchy algorithms, we tuned the number of clusters (k) to 2 and for the DBSCAN algorithm, we tuned the eps and min_samples parameters to 7 and 5 respectively, producing 2 as the number of clusters (k) and we noticed improvement in the model performance (accuracy) of all the three algorithms. Comparing the three algorithms, the DBSCAN algorithm seems to have performed better than the others (k-means and hierarchy algorithms), having a better model accuracy both before and after optimisation. Therefore, following the model performance and accuracy level in predicting whether an adult in the census data set will earn above $50k or not, we conclude that the DBSCAN algorithm is best for modelling this census data set.

4.2 CLASSIFICATION MODEL

Here we shall study and discuss the logistic regression algorithm and the support vector machine algorithm.

4.2.1 LOGISTIC REGRESSION ALGORITHM

To perform logistic regression on this data set, we first need to see some scatter plots of some variables against the response variable (income). Note that the main purpose of the implementing logistic regression was in the case were there is no linear relationship between variables (low correlation between variables) and there are categorical binaries. Below are some scatter plots.

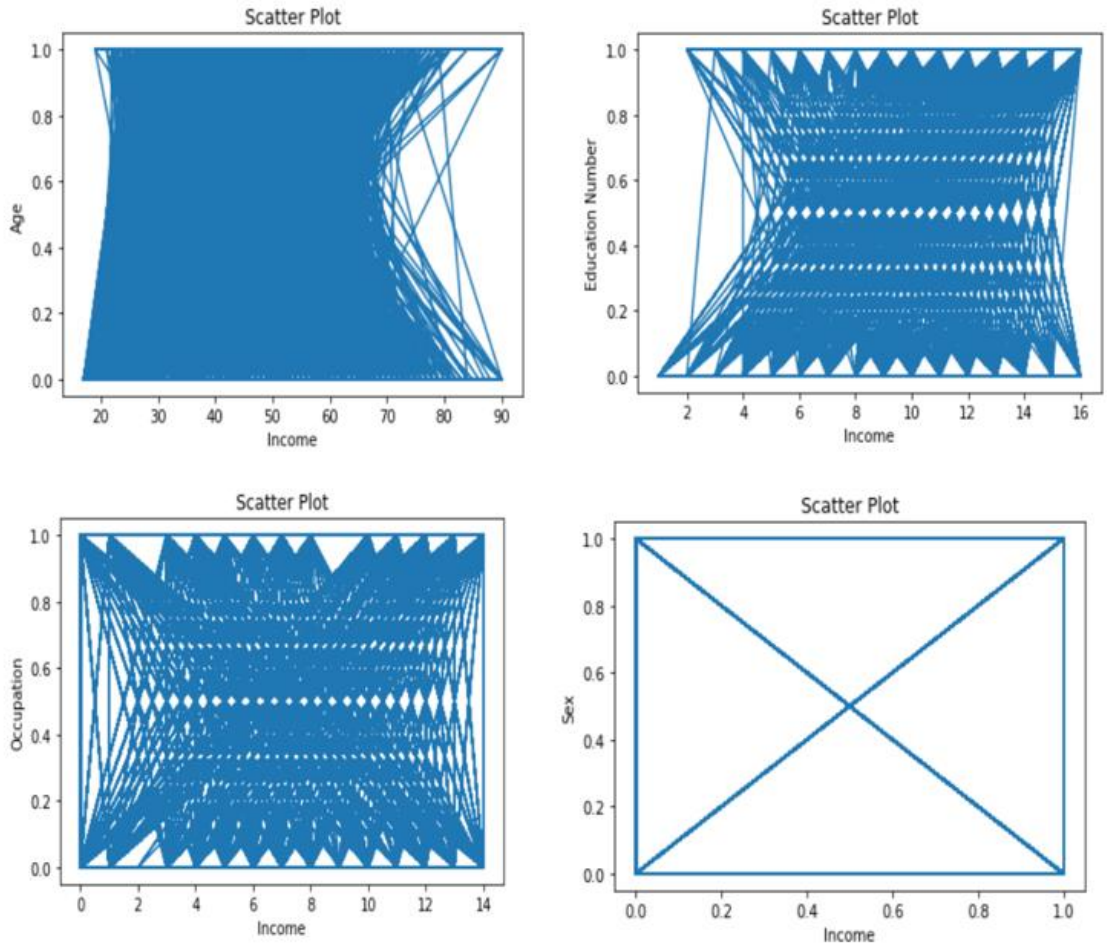


Figure 38: *A mixed scatter plots of age, sex, occupation, and education number variables against the income variable.*

It can be seen in both our previous heatmap plot and figure 25 that there no linear relationship between variables in this census data set, so logistic regression algorithm can be used to fit this data set.

Python Code 41.

```
# A scattered plot to see reason why we want to you logistic regression (Income is dependent variable)

plt.plot(census['age'], census['income'])
plt.xlabel('Income')
plt.ylabel('Age')
plt.title('Scatter Plot')
plt.show()

plt.plot(census['education-num'], census['income'])
plt.xlabel('Income')
plt.ylabel('Education Number')
plt.title('Scatter Plot')
plt.show()

plt.plot(census['occupation'], census['income'])
plt.xlabel('Income')
plt.ylabel('Occupation')
plt.title('Scatter Plot')
plt.show()

plt.plot(census['sex'], census['income'])
plt.xlabel('Income')
plt.ylabel('Sex')
plt.title('Scatter Plot')
plt.show()
```

The above Python code 41 is used to plot those scattered plots.

The next to do is to split our data into x (the predictors) and y (the response) variables, and then we can further split each (x and y) into train and test, allocating 80% of the data set to train and 20% to test. Below is the head view of both x and y data set respectively.

Table 52.

	age	education-num	capital-gain	capital-loss	hours-per-week	occupation	sex
0	90	9	0	4356	40	0	0
1	82	9	0	4356	18	4	0
2	66	10	0	4356	40	0	0
3	54	4	0	3900	40	7	0
4	41	10	0	3900	40	10	0

And

Table 53.

```
0    0
1    0
2    0
3    0
4    0
Name: income, dtype: int8
```

Python Code 42.

```
# Head view of both X (predictor) and y (response) variables

x.head()
y.head()
```

The above Python code 42 is used to split the data into x (predictor), y (response) variables.

We then further split these data into train and test and then scale them before we can fit our model.

Python Code 43.

```
# Divide the dataset into train and test data

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 2)
```

```
# Feature scaling (To scale the data)

scale = StandardScaler()
x_train = scale.fit_transform(x_train)
x_test = scale.transform(x_test)
```

The above Python code 43 is used to split these data into train and test and to scale them.

We then move further to fitting the logistic regression algorithm but that, we check for class imbalance. Below are our findings.

Table 54.

```
0    0.759521
1    0.240479
Name: income, dtype: float64
```

From table 29, we can see that our response variable (Income) had more adult who earn less than or equal to $50k represented in the data set than those who earn above $50k. This is a big problem as it will produce a model with high accuracy level which may not be effective, realistic, and might be bias when used for future. There are various ways or methods of solving this problem such as dropping some of the majority data points or adding more data points to the minority, but all these methods lead to creating another problem, as dropping, or adding data points will lead to loss of vital information or overfitting of the mode. The best approach to solve this is to assign more weight to the minority when fitting the model.

Python Code 44.

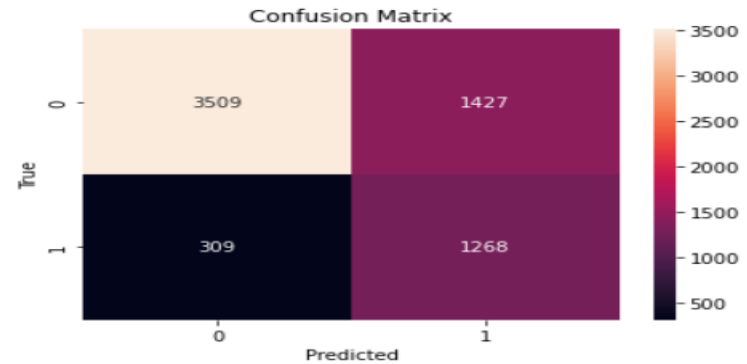
```
# check for distribution of labels

y_train.value_counts(normalize = True)
```

The above Python code 44 is used to check for class imbalance.

We then forge ahead to fit the model and calculate model accuracy using confusion matrix.

Table 55.

	precision	recall	f1-score	support
0	0.92	0.71	0.80	4936
1	0.47	0.80	0.59	1577
accuracy			0.73	6513
macro avg	0.69	0.76	0.70	6513
weighted avg	0.81	0.73	0.75	6513



We can see from table 30 that 4777 were predicted correctly and 1736 were predicted wrongly, giving us a model accuracy of 73%. Shall further optimise this model for better accuracy level.

Python Code 45.

```
# Perform logistic regression
# Make instance of model with default parameters except class weight
# As we will add class weights due to class imbalance problem

lr_model = LogisticRegression(class_weight = {0:0.2, 1:0.8})
lr_model.fit(x_train, y_train)
```

```
# Predicting using the testing dataset

y_pred = lr_model.predict(x_test)
```

```
# Calculating model accuracy using confusion matrix

matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(matrix, annot = True, fmt = "d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
print(classification_report(y_test, y_pred))
```

The above Python 45 is used to fit the model and calculate model accuracy using confusion matrix.

4.2.1.1 OPTIMISATION OF THE LOGISTIC REGRESSION ALGORITHM

To optimising the model, we did hyperparameter tuning, tuning some of the parameters, such as weight, class-weight, C, penalty, and we increased the cv (cross-validation). Below is the view of the optimised model.

Table 56.

```
CPU times: total: 8min 43s
Wall time: 9min 8s

GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=42, shuffle=True),
             estimator=LogisticRegression(),
             param_grid={'C': [0.1, 0.5, 1, 10, 15, 20],
                         'class_weight': [{0: 0.0, 1: 1.0},
                                          {0: 0.0019839679358717435,
                                           1: 0.9980160320641283},
                                          {0: 0.003967935871743487,
                                           1: 0.9960320641282565},
                                          {0: 0.005951903807615305,
                                           1: 0.9940480961923848},
                                          {0: 0.007935871743486974,
                                           1: 0.99206412825...
                                           1: 0.9543687374749499},
                                          {0: 0.047615230460921844,
                                           1: 0.9523847695390781},
                                          {0: 0.04959919839679359,
                                           1: 0.9504008016032064},
                                          {0: 0.05158316633266533,
                                           1: 0.9484168336673346},
                                          {0: 0.05356713426853708,
                                           1: 0.9464328657314629},
                                          {0: 0.055551102204408814,
                                           1: 0.9444488977955912},
                                          {0: 0.05753507014028056,
                                           1: 0.9424649298597194}, ...],
                         'penalty': ['l1', 'l2']},
             return_train_score=True, scoring='f1')
```

Table 31 is a view of the optimised model.

Python Code 46.

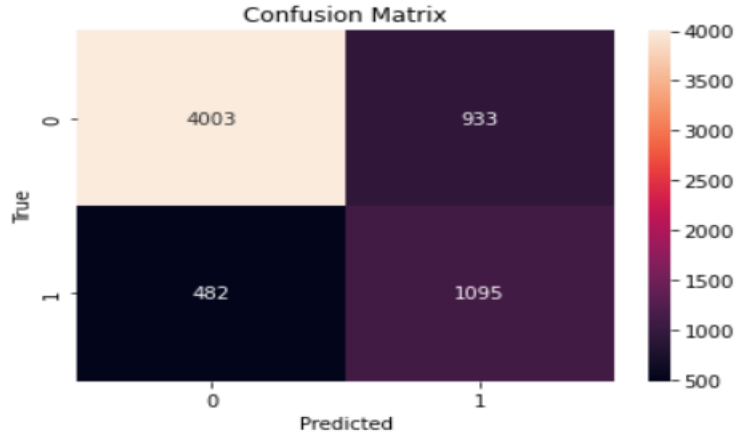
```
%%time

# Hyperparameter tuning
# Define model/create instance

lr = LogisticRegression()

# Tuning weight for minority class then weight for majority class will be 1-weight of minority class
# Setting the range for class weights

weights = np.linspace(0.0,0.99,500)

# Specifying all hyperparameters with possible values

param = {'C': [0.1, 0.5, 1,10,15,20], 'penalty': ['l1', 'l2'],"class_weight":[{0:x, 1:1.0 -x} for x in weights]}

# Create 5 folds

folds = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 42)

# Gridsearch for hyperparam tuning

model = GridSearchCV(estimator = lr, param_grid = param, scoring = "f1", cv = folds, return_train_score = True)

# Train model to learn relationships between x and y

model.fit(x_train, y_train)
```

The above Python code 46 is used for optimising the model.

We then use the optimised model to predict and calculate the model accuracy using confusion matrix.

Table 57.

	precision	recall	f1-score	support
0	0.89	0.81	0.85	4936
1	0.54	0.69	0.61	1577
accuracy			0.78	6513
macro avg	0.72	0.75	0.73	6513
weighted avg	0.81	0.78	0.79	6513



From table 32 we can see that 5098 were predicted correctly and 1415 were predicted wrongly giving us an accuracy level of 78% which is better than the previous accuracy level of the previous model.

Python Code 47.

```
# Predicting using the testing dataset (After Optimisation)

y_pred1 = model.predict(x_test)

# Calculating model accuracy using confusion matrix (After Optimisation)

matrix = confusion_matrix(y_test, y_pred1)
sns.heatmap(matrix, annot = True, fmt = "d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
print(classification_report(y_test, y_pred1))
```

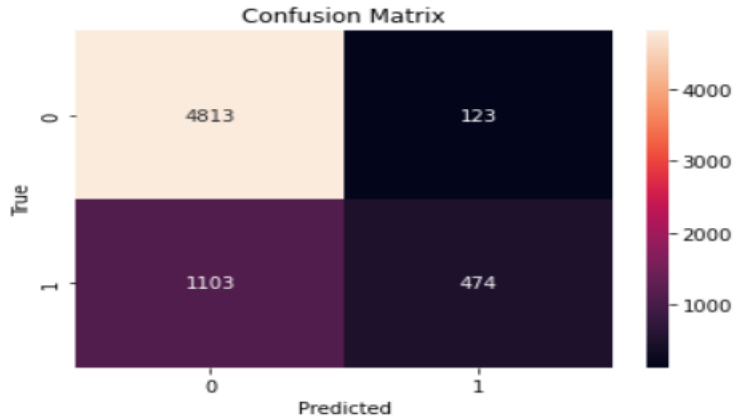
The above Python code 47 is used to predict and to calculate model accuracy using confusion matrix.

4.2.2 SUPPORT VECTOR MACHINE ALGORITHM (SVM)

As our data set has been split into train and test, and scaled, we will go straight to fitting the model and calculating model accuracy using confusion.

Table 58.

	precision	recall	f1-score	support
0	0.81	0.98	0.89	4936
1	0.79	0.30	0.44	1577
accuracy			0.81	6513
macro avg	0.80	0.64	0.66	6513
weighted avg	0.81	0.81	0.78	6513



From table 33 we can see that 5287 were predicted correctly and 1226 were wrongly predicted, giving us a model accuracy of 81%. This is high but we shall still try to optimise the model by means of hyperparameter tuning.

Python Code 48.

```
# Fit an SVM model with sigmoid kernel
# Create a SVM Classifier

clf = SVC(kernel = 'linear', gamma = 'auto') # linear sets

# Train the model using the training sets

clf.fit(x_train, y_train)
```

```
# Predicting using the testing dataset

y1_pred = clf.predict(x_test)
```

```
# Calculating model accuracy using confusion matrix

matrix = confusion_matrix(y_test, y1_pred)
sns.heatmap(matrix, annot = True, fmt = "d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
print(classification_report(y_test, y1_pred))
```

The above Python code 48 is used to fit, predict, and calculate model accuracy using confusion matrix.

4.2.2.1 OPTIMISATION OF THE SUPPORT VECTOR MACHINE ALGORITHM (SVM)

To optimise the model, we used hyperparameter tuning, by tuning some of the parameters of the model such as C, n_iter (number of iterations), cv (cross-validation), gamma, kernel, and verbose. Below is a view of the optimised model.

Table 59.

```
Fitting 2 folds for each of 2 candidates, totalling 4 fits
[CV] END ..................C=1, gamma=0.001, kernel=sigmoid; total time=   5.4s
[CV] END ..................C=1, gamma=0.001, kernel=sigmoid; total time=   5.4s
[CV] END ............................C=1, gamma=1, kernel=rbf; total time=   8.2s
[CV] END ............................C=1, gamma=1, kernel=rbf; total time=   8.0s
CPU times: total: 48.1 s
Wall time: 48.3 s

RandomizedSearchCV(cv=2, estimator=SVC(), n_iter=2,
                   param_distributions={'C': [0.1, 1, 10, 100],
                                        'gamma': [1, 0.1, 0.01, 0.001],
                                        'kernel': ['rbf', 'poly', 'sigmoid']},
                   random_state=2, verbose=2)
```

Table 34 is a view of the optimised model.

Python Code 49.

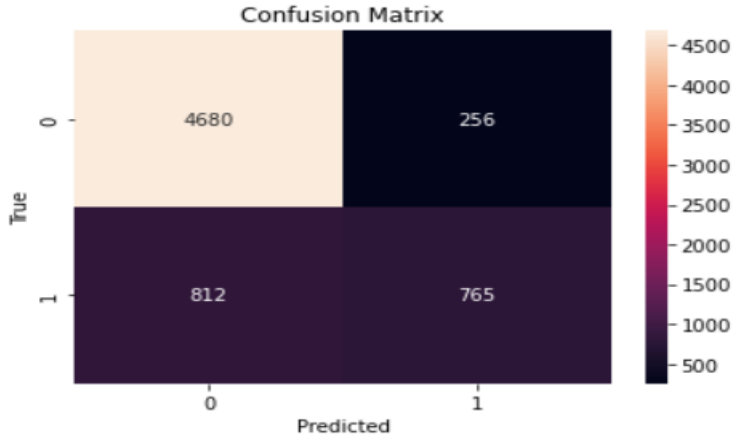
```
%%time

param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly', 'sigmoid']}

grid = RandomizedSearchCV(SVC(), param_grid, n_iter = 2, cv = 2, verbose = 2, random_state = 2)
grid.fit(x_train, y_train)
```

The above Python code 49 is used to run optimisation of the model.

We can predict using the optimised model and calculate model accuracy using confusion matrix.

Table 60.

	precision	recall	f1-score	support
0	0.85	0.95	0.90	4936
1	0.75	0.49	0.59	1577
accuracy			0.84	6513
macro avg	0.80	0.72	0.74	6513
weighted avg	0.83	0.84	0.82	6513



From table 35 we can notice that 5445 were correctly predicted and 1068 were predicted wrongly, giving us a model accuracy of 84% which is better than the previous accuracy level of the previous model before optimisation was carried out.

Python Code 50.

```
# Predicting with the optimised model

g_pred = grid.predict(x_test)
# Calculating model accuracy using confusion matrix

matrix = confusion_matrix(y_test, g_pred)
sns.heatmap(matrix, annot = True, fmt = "d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
print(classification_report(y_test, g_pred))
```

The above Python code 50 is used to predict using the optimised model and to calculate the model accuracy using confusion matrix.

4.2.3 CONCLUSION FOR THE CLASSIFICATION MODELS

The two classification algorithms studied and discussed here are the logistic regression and the support vector machine algorithms. Both models have been fitted with the data set and have been seen to predict income variable with some nice accuracy levels. Also the both algorithms were optimised using hyperparameter tuning by tuning some of their parameters such as weight, class-weight, C, penalty, and we increased the cv (cross-validation) for logistic regression algorithm, and C, n_iter (number of iterations), cv (cross-validation), gamma, kernel, and verbose for support vector machine algorithm, and we have notice good improvement in the model performance and accuracy levels in the both models in predicting income. Following the performance and accuracy level, we can conclude that the support vector machine algorithm is best when compared to the logistic regression algorithm in predicting whether an adult will earn above $50k or not.

4.3 NEURAL NETWORK MODEL

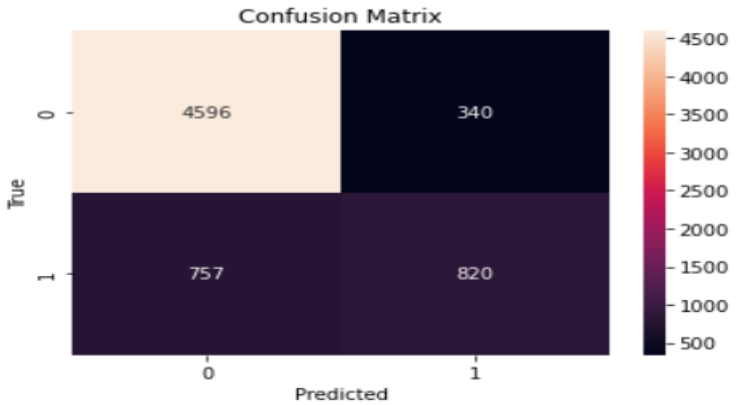
Here we shall study and discuss the multi-layers perceptron algorithm.

4.3.1 MULTI-LAYER PERCEPTRON ALGORITHM (MLP)

We will move straight to fitting the model and calculating the model accuracy by using confusion matrix.

Table 61

	precision	recall	f1-score	support
0	0.86	0.93	0.89	4936
1	0.71	0.52	0.60	1577
accuracy			0.83	6513
macro avg	0.78	0.73	0.75	6513
weighted avg	0.82	0.83	0.82	6513



From table 36 we can see that 5416 were predicted correctly and 1097 were wrongly predicted, giving us a model accuracy level of 83%. The model performance and accuracy level are quite high, but we shall still perform model optimisation to see if we can improve the model.

Python Code 51.

```
# Fitting the MLP Classifier model

mlp_nn = MLPClassifier(random_state = 2, max_iter = 10)
mlp_nn.fit(x_train, y_train)
```

```
# Predicting with the model

y2_pred = mlp_nn.predict(x_test)
```

```
# Checking for Accuracy

matrix = confusion_matrix(y_test, y2_pred)
sns.heatmap(matrix, annot = True, fmt = "d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
print(classification_report(y_test, y2_pred))
```

The above Python code 51 is used to fit, predict, and calculate model accuracy using the confusion matrix.

4.3.1.1 OPTIMISATION OF THE MULTI-LAYER PERCEPTRON ALGORITHM (MLP)

Hyperparameter tuning was also used here to optimise the model by tuning some the model parameter such as n_iter (number of iterations), max_iter (maximum number of iterations), hidden_layer_size (the hidden layer size), activation, solver, alpha, learning_rate (learning rate), verbose, and n_jobs (the number of jobs). Below is a view of the optimised model.

Table 62

```
Fitting 10 folds for each of 30 candidates, totalling 300 fits
CPU times: total: 5min 2s
Wall time: 48min 52s

RandomizedSearchCV(cv=10, estimator=MLPClassifier(max_iter=400, random_state=2),
                   n_iter=30, n_jobs=-1,
                   param_distributions={'activation': ['tanh', 'relu',
                                                      'logistic', 'identity'],
                                        'alpha': [0.0001, 0.05, 0.5, 0.1],
                                        'hidden_layer_sizes': [(60, 60, 60),
                                                               (60, 120, 60),
                                                               (150,)],
                                        'learning_rate': ['constant',
                                                          'adaptive'],
                                        'solver': ['sgd', 'adam', 'lbfgs']},
                   random_state=2, verbose=2)
```

Table 37 is a view of the optimised model.

Python Code 52.

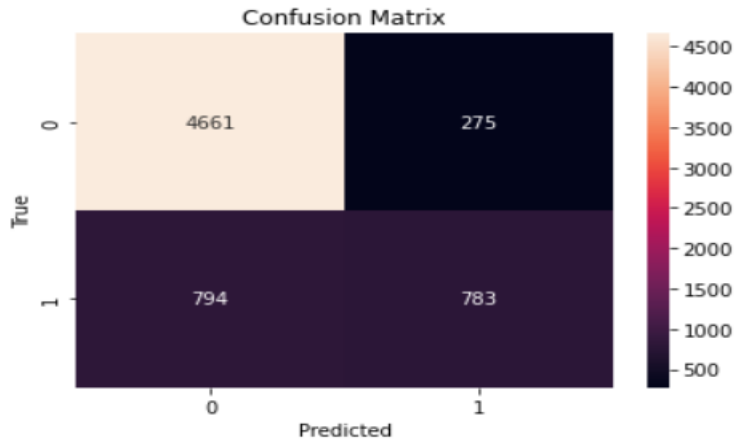
```
%%time

# Hyperparameter tuning for model optimisation.

mlp_gs = MLPClassifier(max_iter = 400, random_state = 2)
parameter_space = {
    'hidden_layer_sizes': [(60,60,60),(60,120,60),(150,)],
    'activation': ['tanh', 'relu', 'logistic', 'identity'],
    'solver': ['sgd', 'adam', 'lbfgs'],
    'alpha': [0.0001, 0.05, 0.5, 0.1],
    'learning_rate': ['constant','adaptive'],
}

mlf = RandomizedSearchCV(estimator = mlp_gs, param_distributions = parameter_space, n_iter = 30,
                         n_jobs = -1, cv = 10, random_state = 2, verbose = 2)
mlf.fit(x_train, y_train)
```

The above Python code 52 is used to optimise the model.

Let's us used the optimised model to predict and calculate the model accuracy using confusion matrix.

Table 63.

	precision	recall	f1-score	support
0	0.85	0.94	0.90	4936
1	0.74	0.50	0.59	1577
accuracy			0.84	6513
macro avg	0.80	0.72	0.75	6513
weighted avg	0.83	0.84	0.82	6513



From table 38 we can see that 5444 were correctly predicted and 1069 were predicted wrongly, giving us a model accuracy level of 84% which is slightly better than the previous accuracy level of the previous model before optimisation was carried out.

Python Code 53.

```
# Predicting Optimised model

mlf_pred = mlf.predict(x_test)
```

```
# Checking for Accuracy (After Optimisation)

matrix = confusion_matrix(y_test, mlf_pred)
sns.heatmap(matrix, annot = True, fmt = "d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
print(classification_report(y_test, mlf_pred))
```

The above Python code 53 is used to predict using the optimised model and to calculate the model accuracy using the confusion matrix.

4.3.2 CONCLUSION FOR THE NEURAL NETWORK MODEL

The one neural network algorithm studied and discussed here is the multi-layer perceptron algorithm (MLP). The model was fitted to our data set and was found to perform high with 83% as accuracy level, predicting 5416 of the income observations correctly. We went forward to optimise the model by the means of hyperparameter tuning, tuning some of the parameters of the model such as n_iter (number of iterations), max_iter (maximum number of iterations), hidden_layer_size (the hidden layer size), activation, solver, alpha, learning_rate (learning rate), verbose, and n_jobs (the number of jobs), and we discovered some improvement in the model which was found to have slightly perform better than the previous model producing an accuracy level of 84% predicting 5444 of the income observations correctly. Though it was not much but we conclude that the model was optimised.

4.4 MODEL EVALUATION

Here we are going through all the models used or applied in this study and evaluate to see which model performed better before and after optimisation was performed on them. Below are the plots to show this.

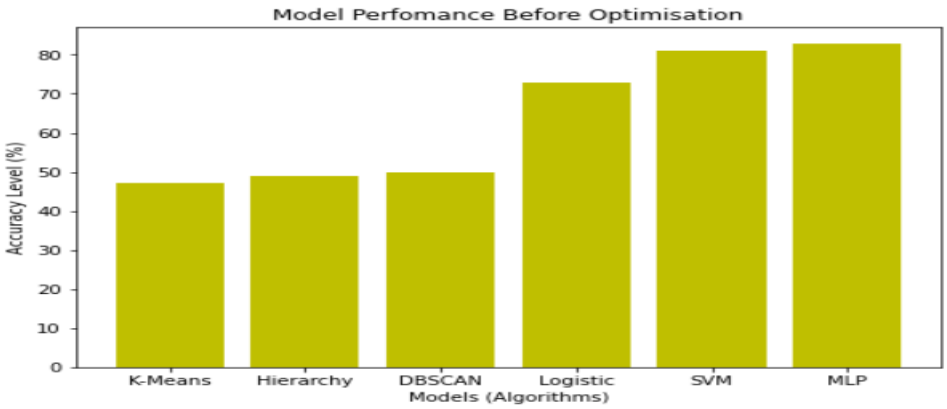


Figure 39: *A Bar plot showing the accuracy level of models before optimisation.*

From figure 26 we can see that the MLP model had the best accuracy level before optimisation with 83%.

Python Code 54.

```
# Before Optimisation.

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
models = ['K-Means', 'Hierarchy', 'DBSCAN', 'Logistic', 'SVM', 'MLP']
model_score = [47, 49, 50, 73, 81, 83]
ax.bar(models, model_score, color = 'y')
plt.xlabel('Models (Algorithms)')
plt.ylabel('Accuracy Level (%)')
plt.title('Model Perfomance Before Optimisation')
plt.show()
```

The above Python code 54 is used to plot a performance Bar chart of models before optimisation.

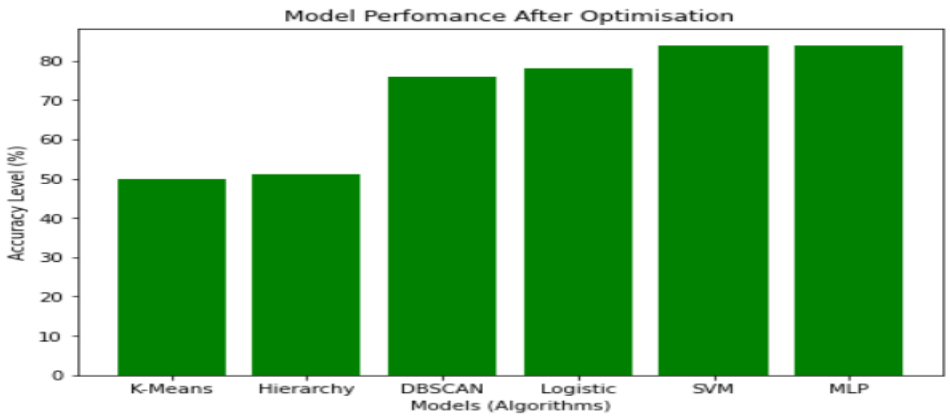


Figure 40: *A Bar plot showing the accuracy level of models after optimisation.*

From figure 27 we can see that both the SVM and the MLP models had the best accuracy level after optimisation with 84% each.

Python Code 55.

```
# After Optimisation.

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
models = ['K-Means', 'Hierarchy', 'DBSCAN', 'Logistic', 'SVM', 'MLP']
model_score = [50, 51, 76, 78, 84, 84]
ax.bar(models, model_score, color = 'g')
plt.xlabel('Models (Algorithms)')
plt.ylabel('Accuracy Level (%)')
plt.title('Model Perfomance After Optimisation')
plt.show()
```

The above Python code 55 is used to plot a performance Bar chart of models after optimisation.

CHAPTER FIVE

5 OVERALL CONCLUSION (QUESTION E)

This study is categorised under supervised and unsupervised machine learning, neural networks, and optimisation. The data set used for this study is the US census data which consist of age, sex, relationship, education, education-num, occupation, native-country, capital-gain, capital-loss, fnlwgt, marital-status, workclass, hours-per-week, and income variables. Some of the variables were dropped because they were not relevant in the predicting of income. In the case of unsupervised machine learning, three clustering algorithms which are the k-means, hierarchy, and the density-based spatial clustering of applications with noise (DBSCAN) algorithms were used to fit the data set with the aim of predicting whether an adult will earn above $50k annually (income), and it was discovered that the DBSCAN algorithm performs better in predicting the response variable (income) both in before and after optimisation. In the case of supervised machine learning, two classification algorithms which are logistic regression and support vector machine (SVM) algorithms were used to fit the data set with the aim of predicting whether an adult will earn above $50k annually (income), and it was discovered that the SVM algorithm performs better in predicting the response variable (income) both in before and after optimisation. While, in the case of neural networks, one algorithm which is the multi-layer perceptron (MLP) algorithm was used to fit the data set with the aim of predicting whether an adult will earn $50k annually or not and it discovered to have performed high in predicting the response variable (income) before and after optimisation. Finally, model evaluation was carried out within all the models used in this study (including neural network, unsupervised, and supervised machine learning models) to select the model that best fit and predict the response variable (income), and MLP model was discovered to be the best because of its high performance and accuracy level in predicting whether an adult will earn $50k annually both before and after the model was optimised. We therefore conclude that for predicting whether an adult will earn $50k annually, the MLP algorithm is best.