ADVANCED ANALYSIS OF ANNUAL MEDICAL PREMIUM IN INDIA

GROUP 05

SANUJI DEVASURENDRA - S14810

SAHAN MADHUSHANKA - S14835

ISINI ASALYA - S15023

Abstract

This report brings out the advanced analysis of the medical premium dataset sourced from the Kaggle website. The main objective of this project is to provide a worthwhile health insurance plan with maximum coverage and a minimum annual premium for a citizen of India. Therefore, to predict the best annual premium that an Indian citizen can obtain, several statistical models were fitted, and for model comparison, the root mean square error (MSE) and the mean absolute percentage error (MAPE) were utilized. Additionally, prior to fitting statistical models, cluster analysis was performed using the k-prototype clustering technique and concluded that there are no clearly specified clusters. Afterwards, it was concluded that out of all the models, the random forest model was identified as the best model to explain our model.

Table of Contents

List of Figures

List of Tables

1. Introduction

Health is wealth. Despite how much our quality of life has increased with the advancement of modern technology, it is difficult to predict the uncertainty into which our lives might fall. As a result, having medical insurance to cover the costs of a medical emergency is essential, especially with the high rate of medical inflation.

But in recent years, the healthcare industry has undergone significant changes, and as a result, health insurance premiums have also steadily risen, affecting both individuals and businesses.

Hence, this report aims to analyze annual health insurance premiums in India in the year 2021, identifying the key drivers behind premium increases and building a predictive model with the most accurate statistical model with the objective of providing maximum coverage at the best (minimum) annual premium price for Indian citizens.

2. Description of the question

Due to the wide discrepancies in financial standing across India, it is practically infeasible to afford medical insurance for every resident of the country. Moreover, due to one or many factors, some beneficiaries have to pay a high premium price, but it is not always in the favor of the beneficiary party. As a result, it is clear that we must closely monitor what factors may influence premium prices, with the ultimate goal of "providing a worthwhile health insurance package to Indian citizens that provides maximum coverage at the best (minimum) annual premium price."

Therefore, in order to achieve the main objective, we can classify the sub-objectives of this project as follows:

To perform an exploratory analysis observing the behavior of different factors affecting the premium price and thereby identify the key factors that most influence the variable “Premium Price”.

To develop a model that incorporates the most significant predictors to forecast the optimum premium price of a health insurance plan in India.

As a result of achieving the above objectives, insurers will be in a better position to make better decisions in risk management when suggesting insurance plans to customers.

3. Description of the dataset

The medical insurance premium prediction dataset is sourced from the Kaggle website and is a collection of 986 observations with 10 predictor variables. This dataset focuses on the annual premium price of an Indian medical insurance plan measured in Indian rupees (symbol) in the year 2021.

No	Variable Name	Description
1	Age	In years.
2	Diabetes	Categories: 0 = absence of diabetes; 1 = presence of diabetes.
3	BloodPressureProblems	Categories: 0 = absence of blood pressure problems; 1 = presence of blood pressure problems.
4	AnyTransplants	Categories: 0 = transplant has not been done; 1 = transplant has been done.
5	AnyChronicDiseases	Categories: 0 = absence of a chronic disease; 1 = presence of a chronic disease.
6	Height	In centimeters
7	Weight	In kilograms
8	KnownAllergies	Categories: 0 = absence of a known allergy; 1 = presence of a known allergy.
9	HistoryOfCancerInFamily	Categories: 0 = There is no family history for cancer; 1 = There is a family history for cancer.
10	NumberOfMajorSurgeries	The number of major surgeries that the beneficiary party has undergone.
11	PremiumPrice	Premium price of the medical insurance plan in Indian rupees (₹).

Table 3.1: Description of Variables

3.1. Data Preprocessing

The dataset did not contain any corrupted data or missing values. Since the variables "Height" and "Weight" had no meaningful interpretation towards the premium price paid by a customer, those two variables were utilized to create the "BMI" variable, which has a more meaningful interpretation.

4. Important results from the descriptive analysis

- The response "Premium Price" in relation to the predictors was examined considering the context of the objective. Due to the minimal skewness (<1), the "Premium Price" distribution is approximately normal. However, due to the extreme observations, the premium varies over a greater range of ₹ 0 to ₹ 40,000, with the majority of prices falling between ₹ 20,000 and ₹ 35,000.

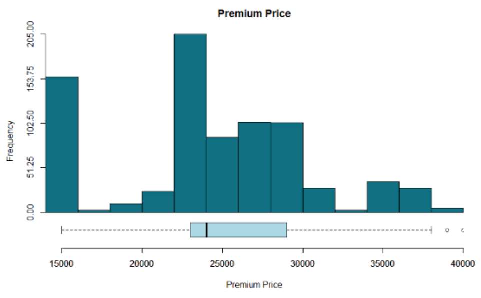


Figure 4.1: Distribution of the premium price

- The applicant's age was then emphasized as a crucial component that affects the premium, as it was clearly highlighted that the higher the age, the higher the premium price.

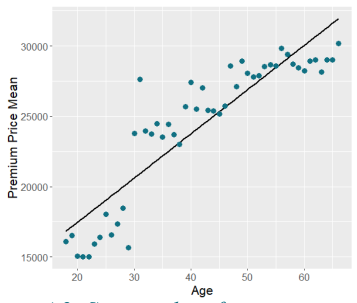


Figure 4.2: Scatter plot of mean premium price and age

- The body mass index did not clearly indicate an increase or decrease in the annual premium when taking into account the existing level of healthiness.
- The premium goes increased if a candidate has diabetes, high blood pressure, or other chronic conditions.
- The applicant will also be required to pay a greater premium than those who have not if they have had a transplant or one, two, or numerous surgeries.
- The history of cancer in the family also has a significant effect on the increase in the annual premium.
- Since the data did not contain any significant outliers, applying a transformation was not a necessity.
- Since there are no observational clusters, according to the score plot of the partial least squares, cluster analysis can be disregarded.

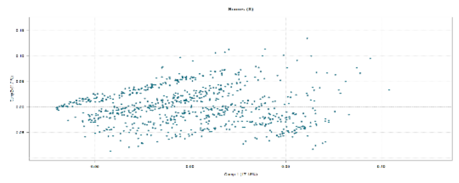


Figure 4.3: Score plot using PLS

- Further, correlations among the variables can be identified through the loading plot of partial least squares. Additionally, the existence of variable clusters can be explained by the loading plot.

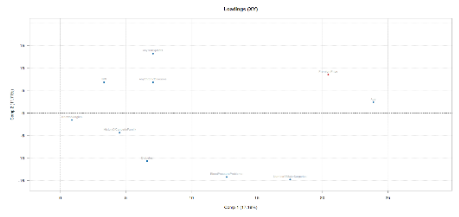


Figure 4.4: Loading plot using PLS

- Given that all VIF values are less than 10, indicate that there is no multicollinearity in the data.

Age	Diabetes	BP problems	Any Transplants	Any Chronic Diseases
1.3041	1.0791	1.1345	1.005	1.0112

Known Allergies	History of cancer in family	Number of major surgeries	BMI
1.0323	1.090	1.358	1.004

Table 4.2: VIF values of the predictive variables

5. Important results from the advanced analysis

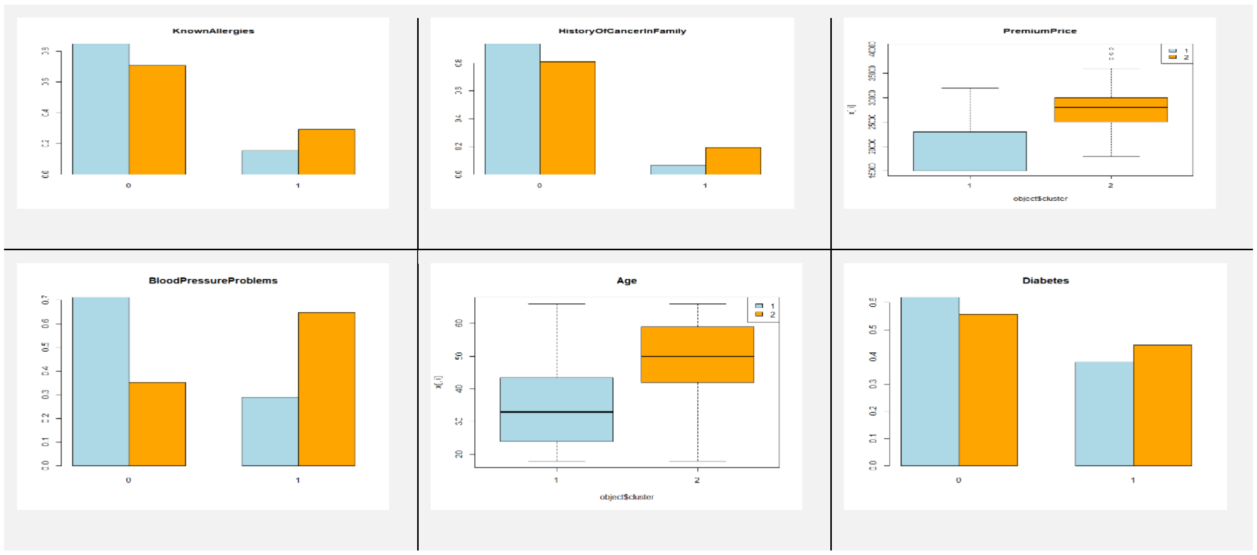
Since our study focuses on a continuous response (Premium Price), regression models were applied to the dataset.

Even though the score plot obtained from the partial least squares method suggested that there are no clusters in the dataset, we further used the K prototype clustering technique, which can handle mixed data types (both numerical and categorical).

Following the technique, we found that the dataset can be clustered into two clusters. However, by further analyzing the two clusters, it was identified that the silhouette mean of the two clusters is *0.3556671,* which implies that the two clusters are not properly clustered.

For a further analysis of the characteristics of the two clusters, the following plots were obtained, and by analyzing them, it can be identified that the two clusters do not contain significant differences in the number of observations containing different characteristics.

Therefore, considering the below grid of variable plots, it can be concluded that the two clusters we got from the k prototype clustering are not perfectly clustered.



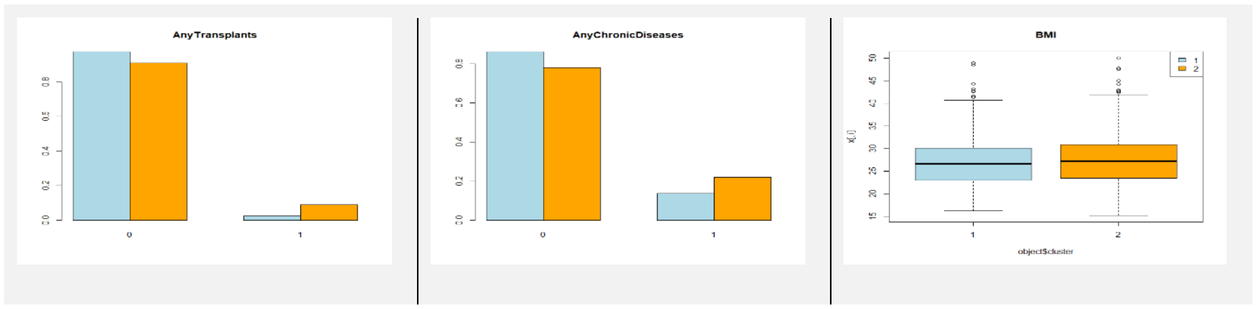


Figure 5.1 : Characteristics of the two clusters

5.1. K-Nearest Neighbors Regression

K-Nearest Neighbors (KNN) is a non-parametric machine learning model that can be used to make predictions based on the k closest observations in the training set. Compared to parametric models, KNN is more flexible as it does not make assumptions about the underlying data distribution.

In this study, we applied KNN to predict premiums, using hyperparameter tuning to optimize the model's performance. However, we found that the root mean squared error (RMSE = 24487.07) for KNN was relatively high compared to other statistical models.

RMSE value	24,487.07

Table 5.1.1 RMSE Value – KNN

This high RMSE may be explained by the fact that KNN performs well for nonlinear datasets, number of parameters are less than 5 and when the dataset is large, whereas our dataset is only relatively small. Then, focus was placed on parametric regression models.

5.2. Multiple Linear Regression Model

Starting with the simplest form of fitting data into multiple predictors, a multiple linear regression model is fitted to the dataset. Then the obtained RMSE value is as follows:

RMSE value	3844.983

Table 5.2.1 RMSE Value - Multiple Linear Regression

5.2.1. Best Subset Selection Method

It is possible that all predictors are associated with the response, but it is more often the case that the response is only associated with a subset of the predictors. Then, for the task of determining which predictors are associated with the response, the best subset selection method under the variable selection method is utilized, as it considers all possible combinations of independent variables.

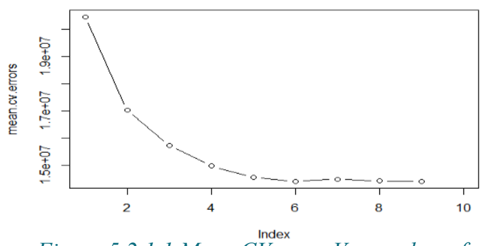


Figure 5.2.1.1 Mean CV error Vs. number of parameters - Best subset

According to figure 5.2.1.1, we can see that the mean cross validation error is at its minimum when the number of parameters is 9. Hence, the best subset selection method suggests keeping all the features in the model. Hence, the RMSE of the best subset selection method is the same as the value of multiple linear regression model, as we did not reduce the features.

RMSE value	3844.983

Table 5.2.1.2 RMSE Value - Best Subset Selection

However, it is known that there are alternative methods to perform feature selection. Hence, now let's focus on shrinkage techniques, which include ridge, lasso and elastic net statistical models.

5.3. Ridge Regression

According to Figure 5.3.1, it can be observed that all the coefficient values tend to zero as lambda increases, but not exactly zero. Hence, all the variables (features) remain the same in the model. The faster the shrinkage of the coefficient towards 0, the lower the importance of the objective of minimizing the premium.

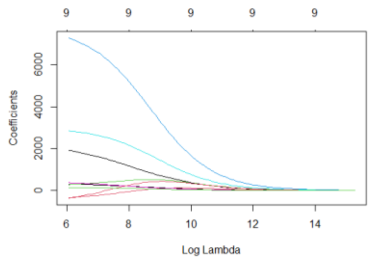


Figure 5.3.1 Variation of coefficient with log lambda - Ridge

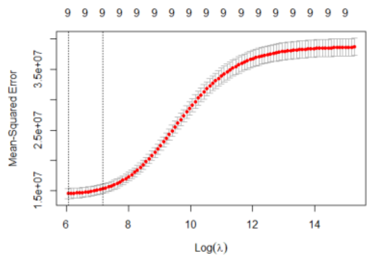


Figure 5.3.2 10 fold cross validation MSE Vs. log lambda - Ridge

5.4. Lasso Regression

In Lasso, it selects a subset of variables, which implies that some variables shrink exactly to zero. Figure 5.4.1 illustrates that fact. As a result of the above fact, the model becomes simpler and more interpretable. As in the figure 5.4.2 if we consider the lambda with 1 SD, it will highlight 5 parameters which is in favor of model simplicity over the accuracy.

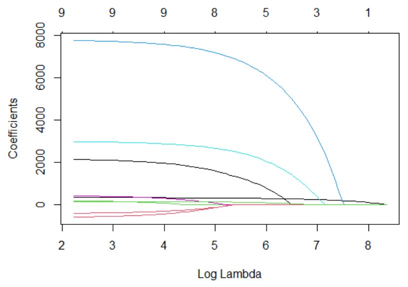


Figure 5.4.1 Variation of coefficient with log lambda - Lasso

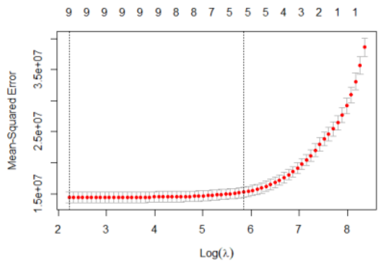


Figure 5.4.2 10 fold cross validation MSE Vs. log lambda - Lasso

5.5. Elastic Net Regression

This model combines both L1 (Lasso) and L2 (Ridge) regularizations to overcome their individual limitations. Figure 5.5.1 illustrates that it can be utilized as a feature selection method. In similar to the Lasso Regression, according to the figure only five parameters

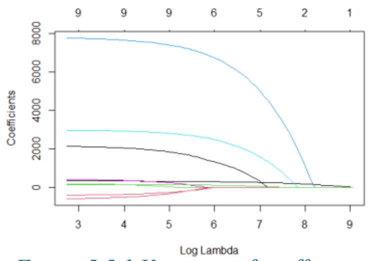


Figure 5.5.1 Variation of coefficient with log lambda - Elastic Net

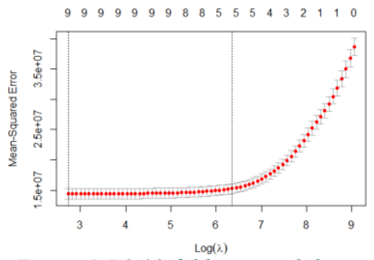


Figure 5.5.2 10-fold cross validation MSE Vs. log lambda – Elastic Net

5.5.2 if we consider the lambda with 1 standard deviation, it will highlight which is in favor of model simplicity over the accuracy.

Now, let's focus on the values for the best lambda, RMSE, and MAPE values of the Ridge, Lasso, and Elastic Net regressions.

	Best Lambda	Test RMSE	MAPE
Ridge	427.1633	3875.3	0.1279603
Lasso	9.202955	3843.689	0.122715
Elastic Net	15.28091	3844.187	0.1227796

Table 5.5.1 - Best Lambda, RMSE and MAPE of shrinkage methods

As per Table 5.5.1, it can be concluded that out of these 3 shrinkage models, Lasso regression has the lowest test RMSE and MAPE, although Elastic Net too has very close RMSE and MAPE to Lasso.

5.6. Boosting

Boosting is another popular ensemble learning technique that can be applied to both regression and classification problems.

There are several types of boosting techniques, and two types of techniques were applied (gradient boosting and XG boosting).

5.6.1. Gradient Boosting

The technique uses gradient descent optimization to train a sequence of weak models. Since the dataset is small, to avoid overfitting, hyperparameter tuning was followed in this technique. The results obtained from the gradient boosting method are as follows:

RMSE	3047.607
MAPE	0.0733202

Table 5.6.1.1 RMSE, MAPE values - Gradient Boosting

5.6.2. XGBoost (Extreme Gradient Boosting)

XGBoosting is based on the gradient boosting framework but introduces some key enhancements to improve performance and speed which can be used for both regression and classification tasks. For the analysis we selected two boosting techniques,

1. The booster function is based on decision tree models, which is similar to the gradient boosting technique (xgbTree).
2. This booster function is based on linear models, specifically regularized linear regression (xgbLinear).

	RMSE	MAPE
xgbTree	8306.259	0.3122753
xgbLinear	8386.285	0.3140013

Table 5.6.2.1 RMSE, MAPE values – XGBoosting

In general, xgbTree performs better than xgbLinear, and that can be verified using Table 5.6.2.1, as both lower RMSE and MAPE are for xgbTree.

However, it can be observed that RMSE values for XGBoosting are comparably high when compared to other models (except for KNN regression), and probable reasons are that XGBoosting performs well when there are a large number of features that are relevant to the target variable and the data are nonlinear.

5.7. Random Forest

Random forest is an ensemble learning technique that combines multiple decision trees to improve the accuracy and robustness of the prediction which is relatively easy to use and interpret. With respect to our analysis, the following results were found.

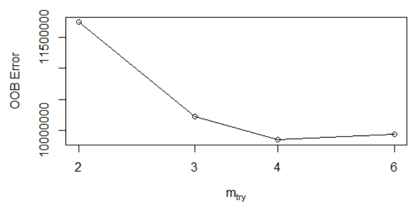


Figure 5.7.1 Out of Bag Error Vs. mtry - Random Forest

Since the random forest model is sensitive to the number of variables that are randomly sampled at each split in the decision trees, first found the number of variables (m_{try}) that has the minimum OOB error.
According to Figure 5.7.1, it can be observed that optimal m_{try} is four (4).

Then we refitted the model using tuned parameters and obtained the following results.

RMSE	3033.353
MAPE	0.06985066

Table 5.7.2.1 RMSE, MAPE values - Random Forest

6. Discussion and Conclusions

6.1. Pre-Processing for Advanced Analysis

The BMI variable is created using the existing variables "Height" and "Weight" and does not require any further analysis for missing values or outliers as the dataset did not contain such.

6.2. Best Model

Considering all the statistical models, it can be clearly seen that both the root mean square error (RMSE) and mean absolute percentage error (MAPE) are at their lowest for the random forest model.

Hence, the best model for our analysis is the random forest model, and the results of the model are as follows.

Best Model	RMSE	MAPE	Accuracy of the fitted model
Random Forest	3033.353	0.06985066	77.08579%

Table 6.2.1 Best Model

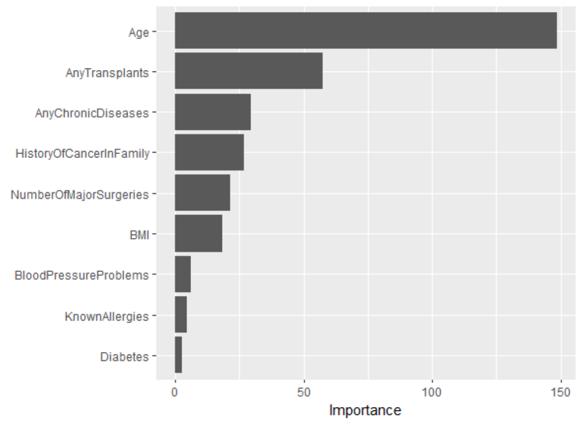


Figure 6.2.1 VIP - Random Forest

```
Age                        148.556356
Diabetes                     2.794697
BloodPressureProblems        6.218071
AnyTransplants              57.357058
AnyChronicDiseases          29.614634
KnownAllergies               4.717590
HistoryOfCancerInFamily     26.619932
NumberOfMajorSurgeries      21.333365
BMI                         18.237416
```

Table 6.2.2 Importance -Random Forest

According to Figure 6.1, it can be identified that age has a significant importance to our model, which we observed the same in our descriptive analysis, implying that the higher the age, higher the premium. Similar to the descriptive analysis, the VIP and Table 6.2 also imply that there is a significant effect on the premium of having undergone surgery, having a chronic illness, having a family history of cancer, and the number of surgeries an applicant has had.

To discuss whether the model is overfitted, the following results were obtained.

	Train	Test
RMSE	1737.872	3033.353
MAPE	0.03736194	0.06985066

Table 6.2.3 Test and Train RMSE and MAPE Values - Random Forest

The test set has higher RMSE and MAPE than the training set, demonstrated in Table 6.2. When we take RMSE into account, the most likely explanation for a larger difference in RMSE is that RMSE is scale-dependent, as the bulk of responses vary between ₹ 20,000 and ₹ 35,000, but when we take MAPE into account, we are unable to see a larger difference. In spite of that, we may say that the model is not overfitted. Further, since our final statistical model is a random forest, it is a known fact that it can handle overfitting better than some other machine algorithms as well.

Finally, our final statistical model, random forest, is flexible for mixed models, improves performance, and reduces variance since it combines the results of multiple decision trees. This makes it more resilient to noise and random variations in the data.

7. Issues encountered and proposed solutions.

Even though the k-prototype clustering technique suggests two clusters, the characteristics of the variable plots do not clearly specify a clustering variable, and the silhouette mean does not suggest well-separated clusters.

8. Appendix

```
Medicalpremium =
read.csv("C:\\Users\\Devasurendra\\Desktop\\Project
1\\Medicalpremium.csv",header=TRUE, stringsAsFactors=FALSE)
attach(Medicalpremium)
#Cleaning data
sum(is.na(Medicalpremium))
sum(duplicated(Medicalpremium))
#Recode Variables
Medicalpremium$BMI<-
Medicalpremium$Weight/(Medicalpremium$Height/100)^2

#Spliting data for trainin and testing
set.seed(1)
train =
sample(1:nrow(Medicalpremium),(nrow(Medicalpremium)*8)/10)
test = (-train)
MedicalpremiumTrain = Medicalpremium[train,]
MedicalpremiumTest = Medicalpremium[test,]
##For training data
yTrain = MedicalpremiumTrain$PremiumPrice
y = yTrain
xTrain = data.frame(MedicalpremiumTrain[-c(6,7,13,14)])
xTrain =
transform(xTrain,Diabetes=as.factor(Diabetes),BloodPressureProblems=
as.factor(BloodPressureProblems),          AnyTransplants=as.factor(Any
Transplants),AnyChronicDiseases=as.factor(AnyChronicDiseases),
     KnownAllergies=as.factor(KnownAllergies),HistoryOfCancerInFamily=
as.factor(HistoryOfCancerInFamily))
x = data.matrix(xTrain[-9])
##For testing data
xTest = data.frame(MedicalpremiumTest[-c(6,7,13,14)])
xTest =
transform(xTest,Diabetes=as.factor(Diabetes),BloodPressureProblems=a
s.factor(BloodPressureProblems),          AnyTransplants=as.factor(AnyTr
ansplants),AnyChronicDiseases=as.factor(AnyChronicDiseases),
KnownAllergies=as.factor(KnownAllergies),HistoryOfCancerInFamily=as.f
actor(HistoryOfCancerInFamily))
#Advanced Analysis
##KNN Regression
library(tidyverse)
library(tidymodels)
library(gridExtra)
library(caret)
set.seed(123)
#Creating a preprocessing pipeline
preProc = preProcess(xTrain,method = c("center","scale"))
#APply preprocessing pipline to the training and testing sets
trainDataPreProc = predict(preProc,xTrain)
testDataPreProc = predict(preProc,xTest)
#Train the KNN regression model using trainingset
knn_model =
train(PremiumPrice~.,data=trainDataPreProc,method="knn",
              trControl = trainControl(method="cv",number=5),
              tuneGrid = expand.grid(k = 1:20))
#Predictions on the test set
knn_predict = predict(knn_model,testDataPreProc)
RMSE_knn = RMSE(xTest$PremiumPrice,knn_predict)
RMSE_knn #24487.07
```

```
###Libraries for Ridge, Lasso and Elastic Net
library(glmnet)
library(mdatools)
library(Metrics)
##Ridge Regression
set.seed(1)
ridge_model = glmnet(x,y, alpha = 0)
cv_model1 = cv.glmnet(x,y,alpha =0)
plot(cv_model1)
best_lambda1 = cv_model1$lambda.min
best_lambda1 #427.1633
best_model1 = glmnet(x,y,alpha = 0,lambda =
best_lambda1)
coef(best_model1)
ridge_predict = predict(ridge_model,s=best_lambda1,newx
= data.matrix(xTest[-9]))
plot(ridge_model,"lambda")
rmse_ridge =
rmse(MedicalpremiumTest$PremiumPrice,ridge_predict)
rmse_ridge #3875.3
mape_ridge = mape(xTest$PremiumPrice,ridge_predict)
mape_ridge #0.1279603
##Lasso Regression
set.seed(1)
lasso_model = glmnet(x,y, alpha = 1)
cv_model2 = cv.glmnet(x,y,alpha =1)
plot(cv_model2)
best_lambda2 = cv_model2$lambda.min
best_lambda2 #9.202955
best_model2 = glmnet(x,y,alpha = 1,lambda =
best_lambda2)
coef(best_model2)
lasso_predict = predict(lasso_model,s=best_lambda2,newx
= data.matrix(xTest[-9]))
plot(lasso_model,"lambda")
rmse_lasso = rmse(xTest$PremiumPrice,lasso_predict)
rmse_lasso #3843.689
mape_lasso = mape(xTest$PremiumPrice,lasso_predict)
mape_lasso #0.122715
##Elastic Net
set.seed(1)
elastic_model = glmnet(x,y, alpha = 0.5)
cv_model3 = cv.glmnet(x,y,alpha =0.5)
plot(cv_model3)
best_lambda3 = cv_model3$lambda.min
best_lambda3 #15.28091
best_model3 = glmnet(x,y,alpha = 0.5,lambda =
best_lambda3)
coef(best_model3)
elastic_predict =
predict(elastic_model,s=best_lambda3,newx =
data.matrix(xTest[-9]))
rmse_elastic = rmse(xTest$PremiumPrice,elastic_predict)
rmse_elastic
mape_elastic = mape(xTest$PremiumPrice,elastic_predict)
mape_elastic #0.1227796
##Gradient Boosting
set.seed(1)
library(gbm)
library(caret)
library(Metrics)
```

```
#Hyper parameter tuning
hyper_grid = expand.grid(
  interaction.depth = c(1, 3, 5),
  n.trees = c(50, 100, 200),
  shrinkage = c(0.01, 0.1, 0.3),
  n.minobsinnode = c(10, 20, 30)
)
ctrl <- trainControl(method = "repeatedcv",
number = 5, repeats = 3)
gbm.MedicalPremium <- train(
  PremiumPrice~.,
  data = xTrain,
  method = "gbm",
  trControl = ctrl,
  tuneGrid = hyper_grid,
  verbose = FALSE
)
print(gbm.MedicalPremium$bestTune)
yhat.gbm =
predict(gbm.MedicalPremium,newdata=xTest
)
RMSE_gbm = (mean((yhat.gbm-
(xTest$PremiumPrice))^2))^(0.5)
RMSE_gbm #3047.607
num_xTest =
as.numeric(xTest$PremiumPrice)
MAPE_gbm = mape(num_xTest,yhat.gbm)
MAPE_gbm #0.0733202
##Random Forest
set.seed(1)
library(randomForest)
library(Metrics)
require(caTools)
#mtry tuning
bestmtry = tuneRF(xTrain[-9],
xTrain$PremiumPrice, stepFactor=1.5,
improve=1e-5, ntree=500)
print(bestmtry) #4
rf.Medicalpremium =
randomForest(PremiumPrice~.,data = xTrain,
importance = TRUE,mtry=4)
yhat.rf = predict(rf.Medicalpremium,newdata
= xTest[-9])
RMSE_rf = (mean((yhat.rf-
(xTest$PremiumPrice))^2))^(0.5)
RMSE_rf #3033.353
num_xTest =
as.numeric(xTest$PremiumPrice)
mape_rf = mape(num_xTest,yhat.rf)
mape_rf #0.06985066
trainRMSE_rf = sqrt(mean((yhat.rfTrain-
(xTrain$PremiumPrice))^2))
trainRMSE_rf #1737.872
mape_rfTrain =
mape(num_xTrain,yhat.rfTrain)
mape_rfTrain #0.03736194
yhat.rfTrain =
predict(rf.Medicalpremium,newdata =
xTrain[-9])
rf.R2 = (cor(yhat.rf,xTest$PremiumPrice))^2
rf.R2 #0.7708579
## Var Imporatance
library(vip)
varImp(rf.Medicalpremium)
vip(rf.Medicalpremium)
```

```
Medicalpremium <- read.csv("D:/UOC/Z. Academic/3rd
year/semester 2/ST 3082/a. Projects/Project 2/Advanced
Analysis/Medicalpremium.csv")

#recording variables and adding to the dataframe
Medicalpremium$BMI<-
Medicalpremium$Weight/(Medicalpremium$Height/100)^2
BMI =
Medicalpremium$Weight/(Medicalpremium$Height/100)^2
Medicalpremium$BMI<-BMI
Medicalpremium = transform(Medicalpremium,
                           Diabetes=as.factor(Diabetes),
BloodPressureProblems=as.factor(BloodPressureProblems),
                           AnyTransplants=as.factor(AnyTransplants),

AnyChronicDiseases=as.factor(AnyChronicDiseases),
                           KnownAllergies=as.factor(KnownAllergies),
HistoryOfCancerInFamily=as.factor(HistoryOfCancerInFamily),
                           Height = as.numeric(Height),
                           Weight=as.numeric(Weight))
data = subset(Medicalpremium, select = -c(Height,Weight) )
sapply(data,class)
library(tidyverse)
data %>%
  is.na() %>%
  sum()
#training and testing data sets
set.seed(1)
train <- sample(1:nrow(data),(nrow(data)*8)/10)
test <- (-train)
Medicalpremium.train<-data[train,]
Medicalpremium.test<-data[test,]
##K-Prototype Clustering
sapply(Medicalpremium.train,class)
library(clustMixType)
#calculate optimal number of cluster, index values and
clusterpartition with Silhouette-index
val = validation_kproto(method = "silhouette", data =
Medicalpremium.train, k = 2:5, nstart = 5)
val$k_opt #2clusters
# apply k-prototypes
kpres1 = kproto(Medicalpremium.train,val$k_opt)
clprofiles(kpres1, Medicalpremium.train)
clust1 = predict(kpres1,Medicalpremium.train)
clust1$cluster
Medicalpremium.train$clus1 = clust1$cluster
head(Medicalpremium.train)
sapply(Medicalpremium.train,class)
##multiple Linear
mul_lin = lm(PremiumPrice ~.,data = Medicalpremium.train)
mul_lin
predicted_lm = predict(mul_lin ,newdata = Medicalpremium.test)
#rmse (scale dependent)
library(Metrics)
rmse_mul =
rmse(Medicalpremium.test$PremiumPrice,predicted_lm)
rmse_mul   #3844.983
#mape (scale independent)
mape_mul =
mape(Medicalpremium.test$PremiumPrice,predicted_lm)
mape_mul  #0.1225629
#rsq
lm_r2 =
(cor(predicted_lm,Medicalpremium.test$PremiumPrice)^2)
lm_r2    # 0.631023
###best subset selection
library(leaps)
model.regfit_full = regsubsets(PremiumPrice~.,
data = Medicalpremium.train, nvmax = 13)
reg_summary = summary(model.regfit_full)
reg_summary
# k fold cross validation to select the models with the best no of
predictors
#ref: ref book
k <- 10
n <- nrow (Medicalpremium.train)
set.seed (1)
folds <- sample ( rep (1:k, length = n))
cv.errors <- matrix (NA, k, 10,dimnames = list (NULL , paste
(1:10)))
predict.regsubsets <- function (object , newdata , id, ...) {
  form <- as.formula (object$call[[2]])
  mat <- model.matrix(form , newdata)
  coefi <- coef (object , id = id)
  xvars <- names (coefi)
  mat[, xvars] %*% coefi}
for (j in 1:k) {
  best.fit <- regsubsets (PremiumPrice~.,data =
Medicalpremium.train[folds != j, ],nvmax = 9)
for (i in 1:9) {
    pred <- predict (best.fit ,  Medicalpremium.train[folds == j,
], id = i)
    cv.errors[j, i] <-mean ((
Medicalpremium.train$PremiumPrice[folds == j] - pred)^2)
  }
}
mean.cv.errors <- apply (cv.errors , 2, mean)
which.min(mean.cv.errors)
par (mfrow = c(1, 1))
plot (mean.cv.errors , type = "b")
#model with 9 predictors is selected
reg.best <- regsubsets (PremiumPrice ~ ., data =
Medicalpremium.train ,nvmax = 9)
summary(reg.best)
coeffi = coef (reg.best , 9 )
#model accuracy
selected_vars <- names(coef (reg.best , 9))[-1]
formu = paste0("PremiumPrice ~ ", paste(selected_vars,
collapse = " + "))
formu
model_subset = lm(PremiumPrice ~ Age + Diabetes +
BloodPressureProblems + AnyTransplants +
AnyChronicDiseases + KnownAllergies +
HistoryOfCancerInFamily + NumberOfMajorSurgeries +
BMI,  data = Medicalpremium.train)
car::vif(model_subset)
#accuracy
predicted_sub = predict(model_subset ,newdata =
Medicalpremium.test)
#rmse (scale dependent)
library(Metrics)
rmse_sub =
rmse(Medicalpremium.test$PremiumPrice,predicted_sub)
rmse_sub   #3844.983
#mape (scale independent)
mape_sub =
mape(Medicalpremium.test$PremiumPrice,predicted_sub)
mape_sub   #0.1225629
#rsq
sub_r2 =
(cor(predicted_sub,Medicalpremium.test$PremiumPrice)^2)
sub_r2   #0.631023
##XGBoosting
library(caret)
library(xgboost)
#xgbLinear
xgb_caret = train(PremiumPrice ~ .,
                  data = Medicalpremium.train,
                  method = "xgbLinear",
              #specify how we want to define tuning parameters
                  trControl = trainControl(method = "repeatedcv",
                                           number = 3, #k
                                           repeats = 2,
                                           verboseIter = TRUE),
                  #hyper parameter tuning
                  tuneGrid = expand.grid(nrounds = c(1000,1500),
                                         eta = c(0.01,0.05), #how quickly
the algorithm will learn from data
                                         alpha = c(0,1,100),
                                         lambda = c(0,1,100)),
                  objective = "reg:squarederror")
xgbL_pred = predict(xgb_caret, trainX)
library(Metrics)
#rmse
rmse_xgb = rmse(Medicalpremium.test$PremiumPrice,
xgbL_pred)
rmse_xgb  #8386.285
#mape (scale independent)
mape_xgb = mape(Medicalpremium.test$PremiumPrice,
xgbL_pred)
mape_xgb  #0.3140013
#xgbTree
xgb_Tree = train(PremiumPrice ~ .,
                 data = Medicalpremium.train,
                  method = "xgbTree",
                  #specify how we want to define tuning parameters
                  trControl = trainControl(method = "cv",
                                           number = 5, #k
                                           #repeats = 2,
                                           verboseIter = TRUE),
                  #hyper parameter tuning
                  tuneGrid = expand.grid(nrounds = c(500,1000),
                                         eta = 0.1, #how quickly the
algorithm will learn from data
                                         max_depth = c(2,4,6),
                                         colsample_bytree = c(0.5,0.6),
                                         subsample = c(0.5,0.6),
                                         gamma = 0.1,
                                         min_child_weight = c(2,4,6)),
                  objective = "reg:squarederror")
xgbT_pred = predict(xgb_Tree, trainX)
library(Metrics)
#rmse
rmse_xgbTree =
rmse(Medicalpremium.test$PremiumPrice,
xgbT_pred)
rmse_xgbTree  #8306.259
#mape (scale independent)
mape_xgbTree =
mape(Medicalpremium.test$PremiumPrice,xgbT_pred)
mape_xgbTree  #0.3122753
```