June 8, 2020

Data Analysis for Predicting Medical Patient No-Shows

IST 707

Stephen Bookstaber | Kathryn Egan

Contents

[Introduction 1](#_Toc42511089)

[Dataset Description 1](#_Toc42511090)

[Linear Regression 3](#_Toc42511091)

[Association Rule Mining 4](#_Toc42511092)

[Cluster Analysis 6](#_Toc42511093)

[Classification Analysis 9](#_Toc42511094)

[Overview 9](#_Toc42511095)

[Preprocessing 12](#_Toc42511096)

[Model Functions and Performance 14](#_Toc42511097)

[Classification Summary 19](#_Toc42511098)

[References 23](#_Toc42511099)

[Appendix A: Classification Code 24](#_Toc42511100)

[Appendix B 30](#_Toc42511101)

# Introduction

In 2017 the total estimated cost of missed healthcare appointments each year in the United States is upwards of $150 billion. With no-show rates topping out at 30% and the average 60-minute unused timeslot costing a doctor $200, it’s easy to see how quickly this all can add up (Gier, 2017).

Predicting patients most likely to be a no-show can help medical practices create plans and policies on how to address the issue by concentrating on patients predicted to have a higher likelihood of abandoning their scheduled appointment. The objective of this study is to develop a model that yields the most accurate prediction.

## Dataset Description

The dataset is from Kaggle and consists of 110,527 medical office appointment records being used for developing prediction models (JoniHoppen, 2017). There are 14 variables in the dataset:

|  |  |
| --- | --- |
| Variable | Description |
| PatientId | unique identifier |
| AppointmentID | unique identifier |
| Gender | male or female |
| ScheduledDay | day of the week the appointment was scheduled on |
| AppointmentDay | day of the week the appointment was scheduled for |
| Age | age of patient. |
| Neighborhood | general area where the appointment takes place. |
| Scholarship | Is the patient using social welfare programs for payment, True or False? |
| Hypertension | Does the patient have high blood pressure, True or False? |
| Diabetes | Does the patient have diabetes, True or False? |
| Alcoholism | Does the patient have alcoholism, True or False? |
| Handicap | Does the patient have a physical limitation, True or False? |
| SMS\_received | Did the patient receive a text to remind them of their appointment, True or False? |
| No-show | Did the patient not show up to their appointment, True or False? |

Prior to preparing the data for analysis, we explored the dataset using descriptive statistics to identify any immediately evident trends in age and gender that may help us build better predictive models.

A screenshot of a cell phone

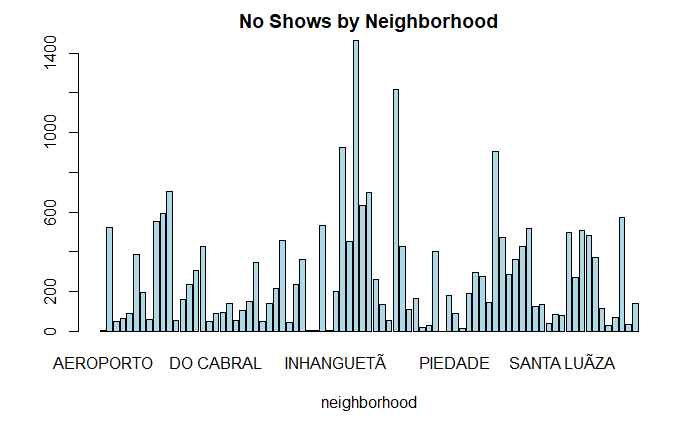
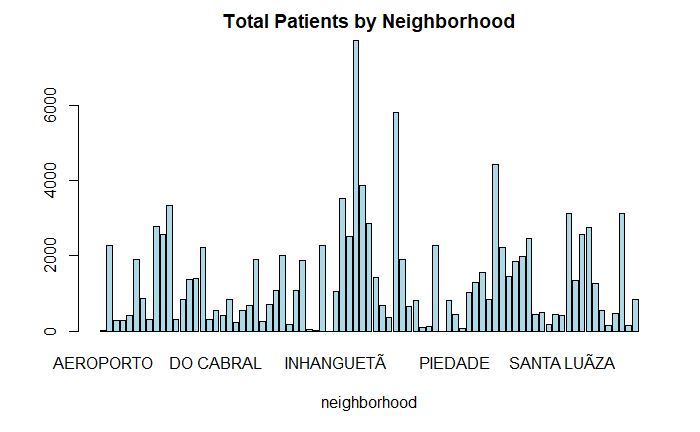
Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

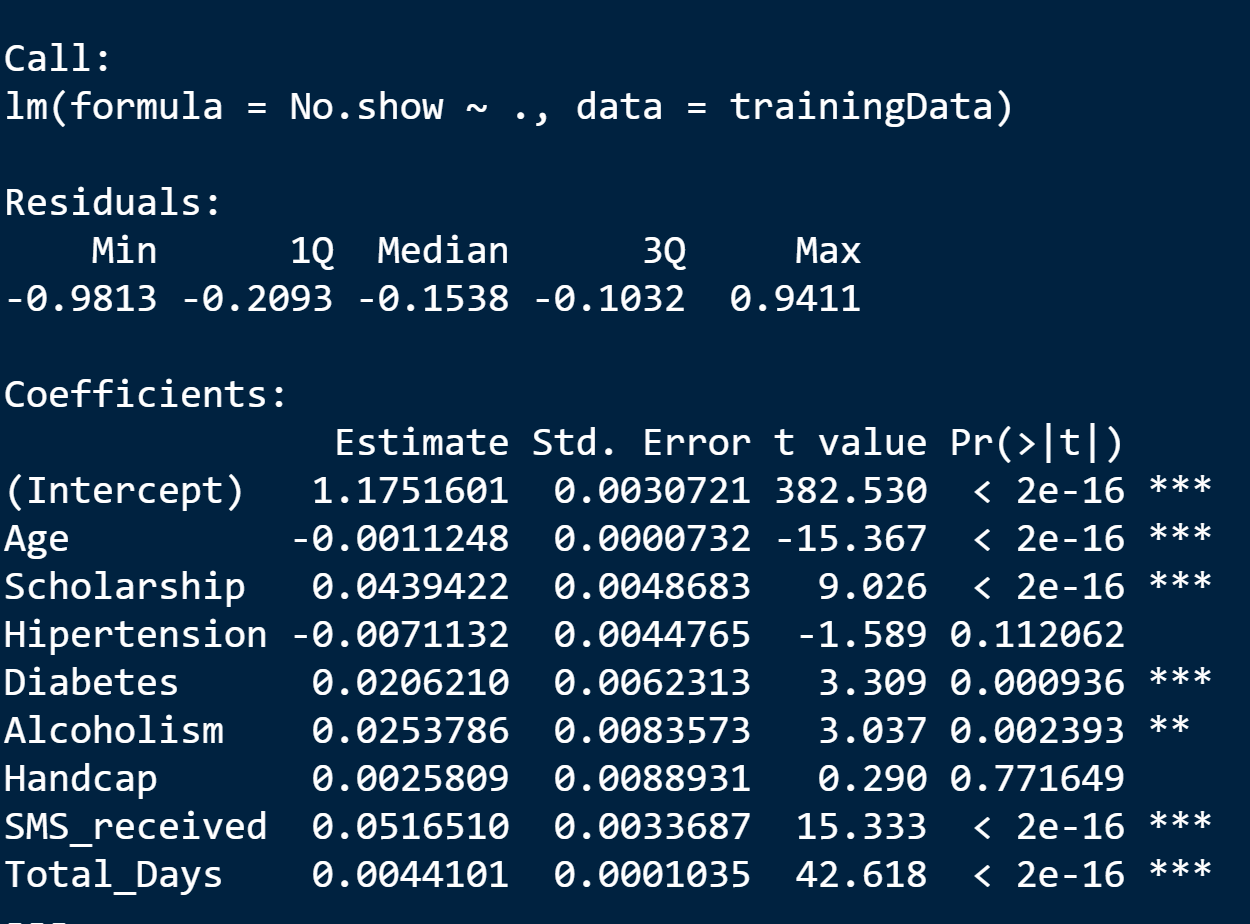
Description automatically generated



After data exploration, the dataset was partitioned into a training and testing dataset. For linear regression and cluster analysis, the No.show variable was changed to 0/1, where 1 was for no-show and 0 to indicate the patient showed up to the appointment. Also, a variable was created using the calculation of days between the schedule day and appointment day variables. The latter two were removed from the dataset along with patient and appointment IDs. Gender was ultimately removed after descriptive statistics indicated that the no show rates mirrored the total patient rates by gender.

# Linear Regression

Linear regression was performed on the training data set (due to restrictions of the CPU) to identify variables with a significant relationship to the likelihood of a patient abandoning their medical appointment. Based on this linear model, all variables in the dataset were significant corollaries to whether or not a patient showed up to a scheduled appointment with the surprising exceptions of the hypertension and handicap variables. In our later analysis these variables turn out to have a strong association to a patient showing up or not.



# Association Rule Mining

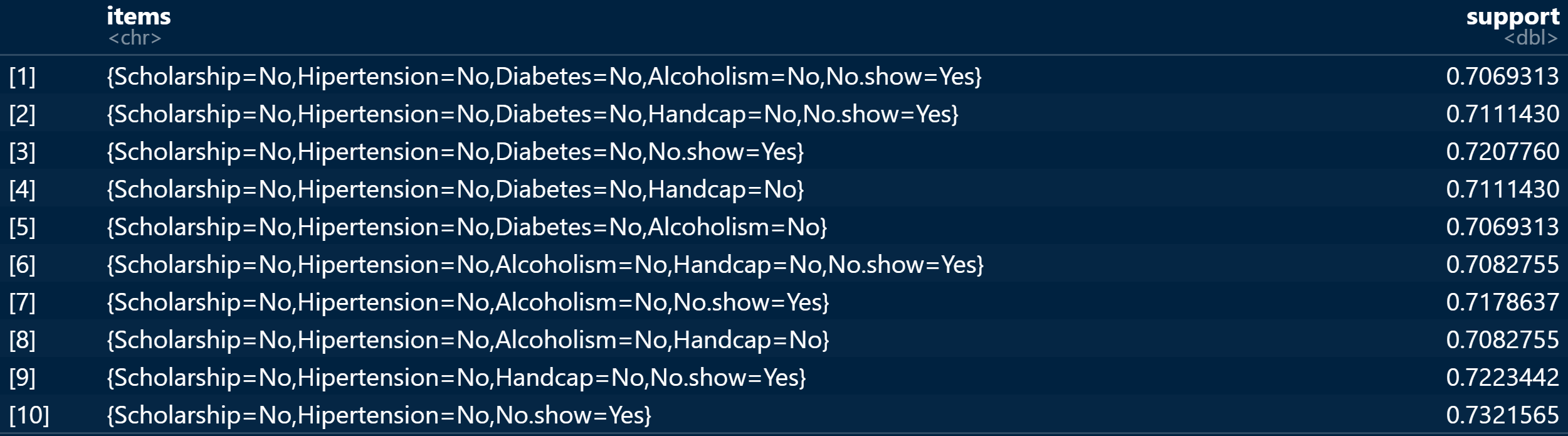
To identify frequently co-occurring associations in the dataset, association rule mining was performed. To preprocess the dataset, the integer attributes were transformed to 1=”yes” and 0=“no”.

Prior to rule mining, variable frequency was investigated using the eclat function. Though the support parameter was changed between runs of the model, the most frequent variables were continually found to be combinations of disease state variables (diabetes, hypertension, handicap, and alcoholism) all of which were “no” observations, meaning it was far more likely to find patients in the records who did not have a combination of these diseases. Additionally, these variables were frequently accompanied by Scholarship=No, essentially stating that being on state-run healthcare plans were not a frequent observation.

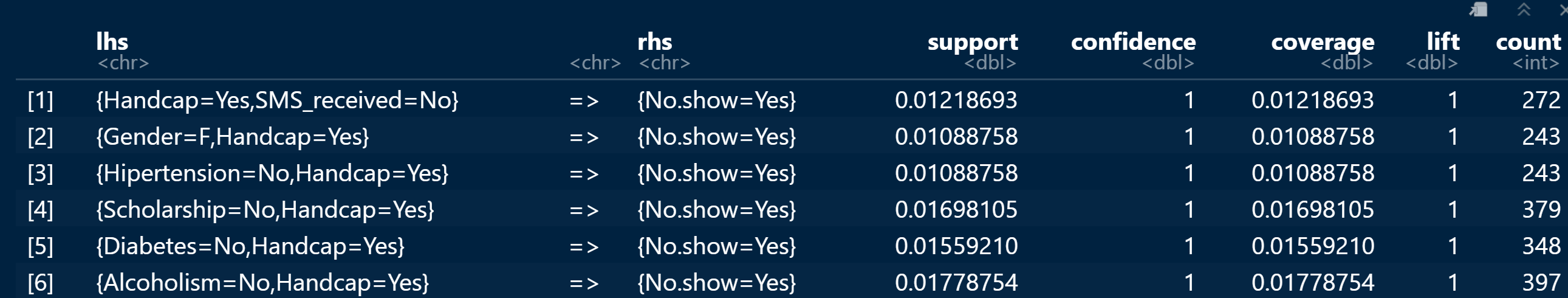
A screenshot of a cell phone

Description automatically generated

A closer look to see how the frequency changed when removing the records of those patients who showed up to their appointment shows that the frequent variables did not change and the no show variable was just added to three of the top 10 existing frequent variable lines. This indicates that people without disease were just as frequently linked to patients not showing up to an appointment as those who showed up.



After the frequent variables were discovered, the apriori algorithm was used to perform association rule mining. As with the frequency model, multiple runs of the apriori algorithm were performed using a variety of combinations of support and confidence. Ultimately the strongest associations were found using supp=0.01 and conf=0.5 and setting rhs= “No.show=Yes”.



Based on the mined rules with the highest confidence and lift, it is evident that a patient with a handicap has a much higher likelihood of not showing up to an appointment than other patients. Not knowing any more detail than the binomial data provided, it is hard to tell if specific handicaps impact this likelihood or if lack of transportation or caregiver support is at play here. Also found in the rules, which seems rather intuitive, patients with a handicap who did not receive a reminder text message about their appointment were prone to miss their appointment.

# Cluster Analysis

Another data mining tool utilized to analyze the dataset to help predict the chances of patients showing up to their appointments is cluster analysis. K-means and Hierarchical Cluster models were developed to identify similarities in patients who either showed up to their appointments or not.

To preprocess the dataset, all attributes were transformed to numeric and only complete records were included.

To find the optimal number of clusters for analysis, an elbow plot was created. It showed a distinct bend at 2 clusters.

A screenshot of a cell phone

Description automatically generated

The algorithm used for the K-means cluster analysis was Hartigan-Wang. On the first run of the algorithm, the number of clusters was set to 2. The yield of between sum of squares/ total sum of squares = 50.5%.

A screenshot of a cell phone

Description automatically generated

Several more runs of the model were performed changing the number of clusters and iterations. In the end, 3 clusters yielded the highest between sum of squares/ total sum of squares = 65.0%. Due to the strength of this model using the training dataset, the algorithm was run again using the testing dataset which yielded between sum of squares/ total sum of squares = 65.1%, thus validating the model as a predictive tool.

**A screenshot of a cell phone

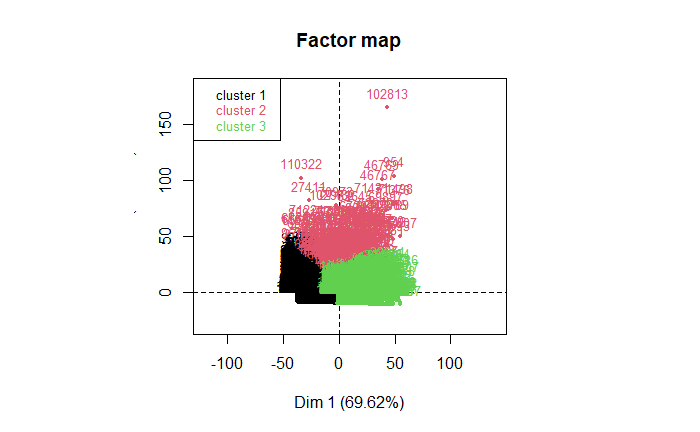
Description automatically generated**

Due to the strength of this model employing the training dataset, the algorithm was run again using the testing dataset which yielded between sum of squares/ total sum of squares = 65.1%, thus validating the model as a predictive tool.

**A screenshot of a cell phone

Description automatically generated**

To look at the clusters from another perspective, hierarchical clustering was endeavored using the euclidean distance method. A factor graph was plotted using the results.



Through the cluster analyses, the attributes of patients by cluster are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Cluster** | **1** | **2** | **3** |
| **Shows up to appointment** | Less Likely to Show Up | Least Likely to Show Up | Most likely to Show Up |
| **Age** | Avg. age is 17 | Avg. age 45 | Avg. age 57 |
| **Scholarship** |  |  |  |
| **Hypertension** |  |  |  |
| **Diabetes** |  |  |  |
| **Alcoholism** |  |  |  |
| **Handicap** |  |  |  |
| **Text Received** |  |  |  |
| **Days between scheduling and appointment** |  |  |  |

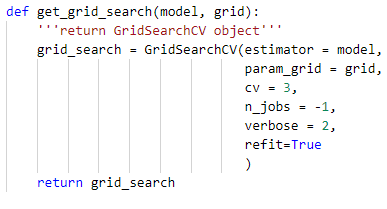
= more likely to have the attribute = less likely to have the attribute

The overlapping attribute, between the results of Association Rule Mining and K-means cluster analysis, that is the strongest predictor that a patient will not show up for an appointment, is not receiving a text message reminder. While other attributes like having a handicap and length of days between scheduling and the appointment are also strong predictors, the simplicity of ensuring an increase in patients showing up by sending out a reminder, cannot be overstated.

# Classification Analysis

## Overview

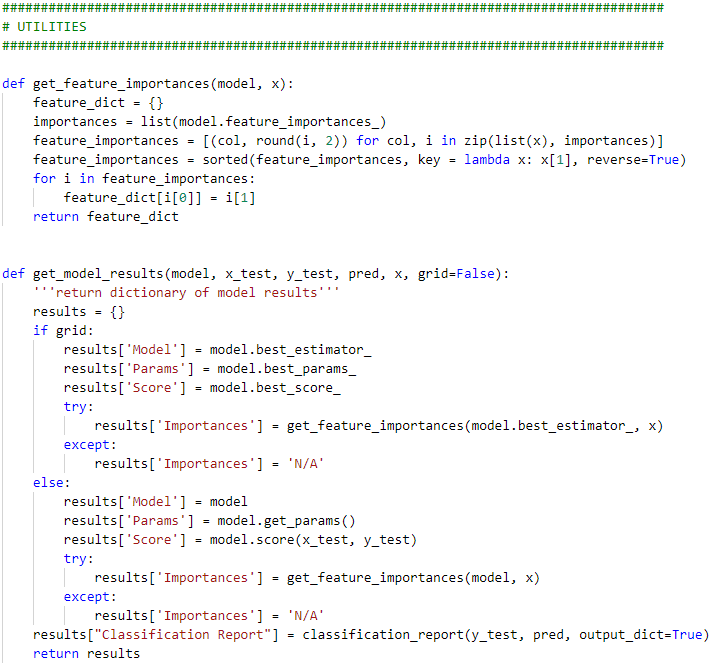
We ran the following classification models: decision tree, random forest, Naïve Bayes (Gaussian and Complement), k-nearest neighbors, and support vector classification (SVC). Every model is executed within its own function. For random forest, kNN, and SVC, we used a grid search to test a range of parameters. The grid is defined under each model function and passed to the get\_grid\_search function, where we specify three-fold cross-validation.



After the data has been preprocessed, we split the data into testing and training sets. Then, we define model\_dict, which is a dictionary which will store the results from each model run. Each model function returns a summary of the model’s performance as the dictionary values, which are then parsed into an output csv.



We use two main utility functions for each model run to retrieve the following information: model meta data (including parameters), performance metrics, confusion matrix, and feature importance. For the models that use grid search, we store this data only for the best estimator. The following function returns the model results to the model function, where it is returned to the model\_dict.



## Preprocessing

Preprocessing for classification is done within the preprocess\_df function, which executes the following steps:

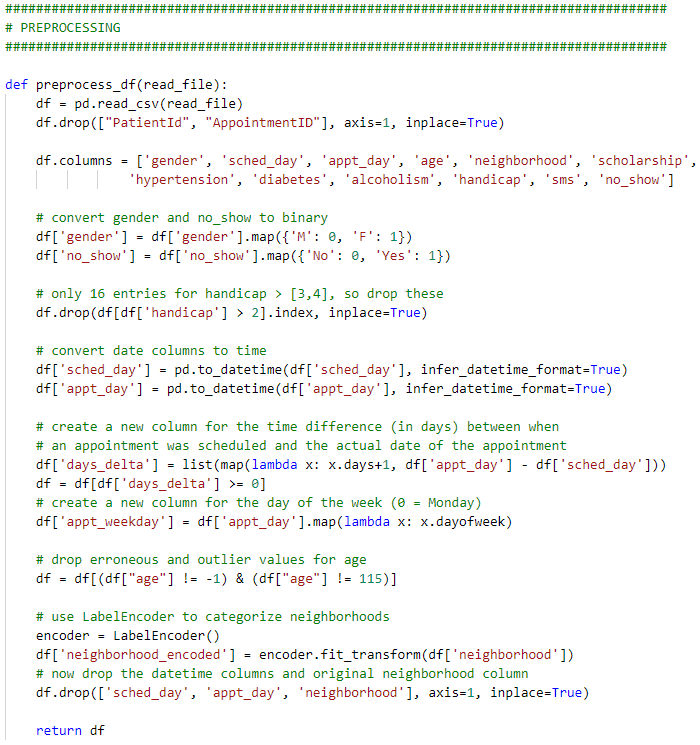
1. Reading the csv file into a pandas data frame
2. Dropping the “PatientId” and “AppointmentID” columns, as these are administrative labels containing data irrelevant for analysis.
3. Fixing misspelled column names and converting the names to snake-case to ensure

Next, the data requires reformatting. This involves:

1. Converting the gender (M/F) and no-show (Yes/No) columns to binary (0/1).
2. Converting the two time-stamp columns from string to data time format.
3. Dropping erroneous values from the age column contained an extreme outlier (age 115) and an erroneous value (-1).
4. Dropping 16 rows from the handicap column that had the value 3 or 4, as these were either erroneous or outliers at best.

Finally, I created three custom columns to improve model accuracy:

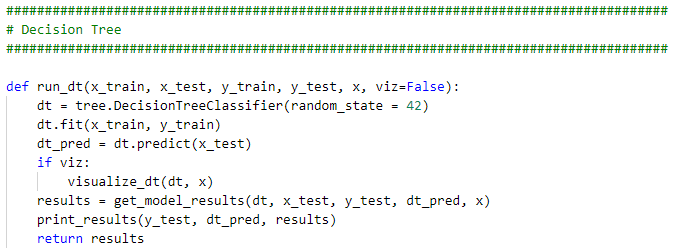
1. days\_delta: This is the floating-point difference in days between the time an appointment was scheduled and the actual day of the appointment, and is derived from the appt\_day and sched\_day columns.
2. appt\_weekday: This is the weekday of the appointment. We include this column because it seems intuitive that no-show probability would vary depending on the day of the week, and could improve the model’s accuracy.
3. neighborhood\_encoded: this converts the neighborhood column into a label for classification. We then drop the original column.



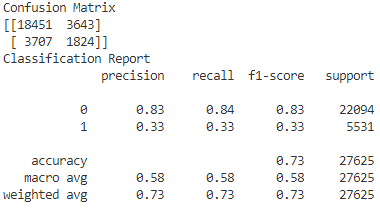
## Model Functions and Performance

### Decision Tree Classifier

#### Model function

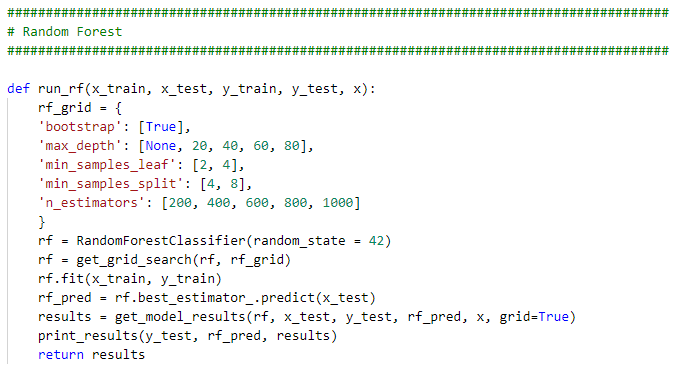


#### Performance

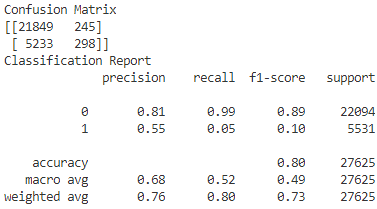


### Random Forest

#### Model Function

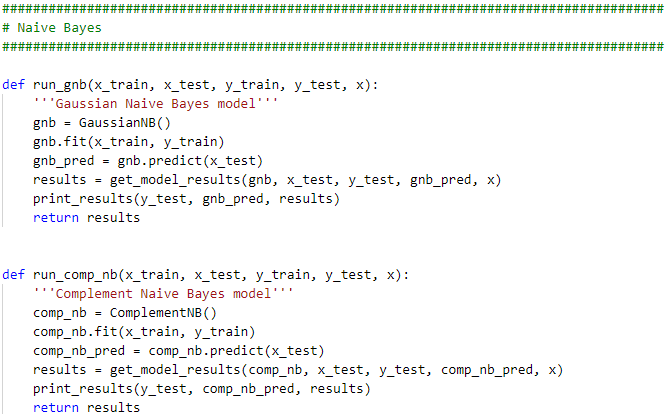


#### Performance



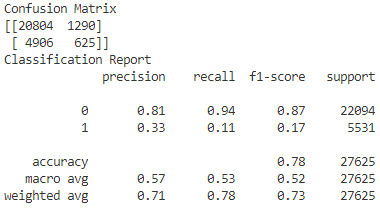
### Naïve Bayes

#### Model Function

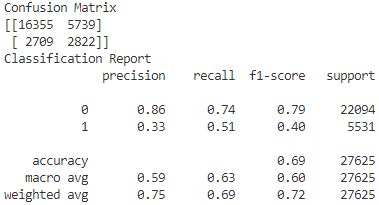


#### Performance

##### Gaussian:

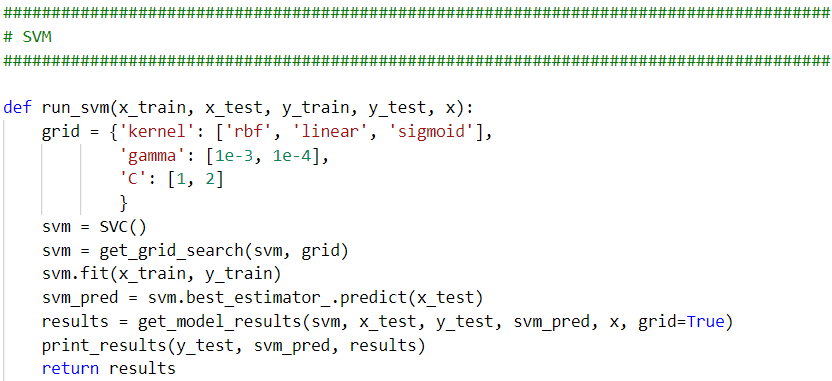


##### Complement:



### SVM

#### Model Function

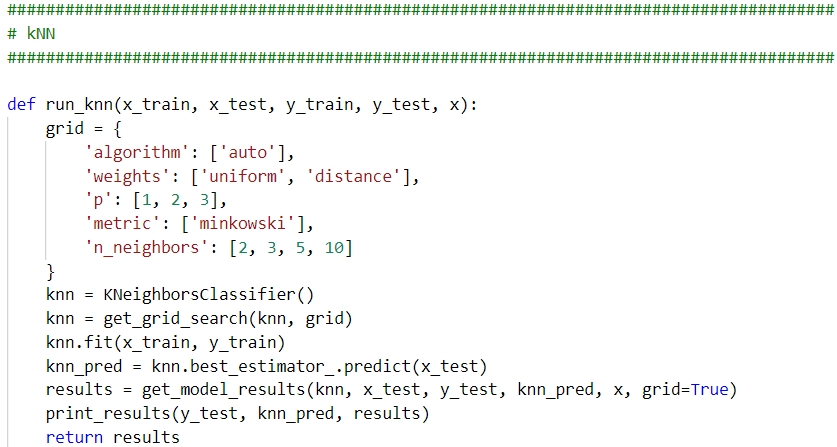


#### Performance

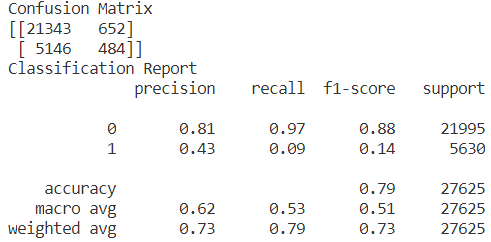
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Accuracy | Precision | Recall | F1-Score |
| SVC(  C=1,  kernel='linear') | 0.798769231 | 0.633932999 | 0.796199095 | 0.705860504 |

### k-Nearest Neighbors

#### Model Function



#### Performance



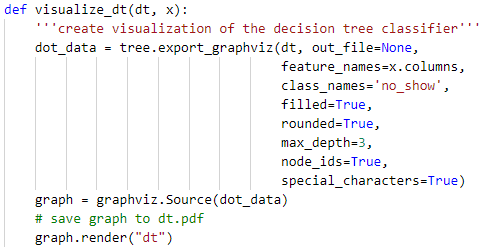
## Classification Summary

For the following table, the precision, recall, and f1-score are all derived from the weighted average metric of the classification report. Based on these scores, Random Forest and kNN were our best-performing models, with comparable scores across all performance metrics.

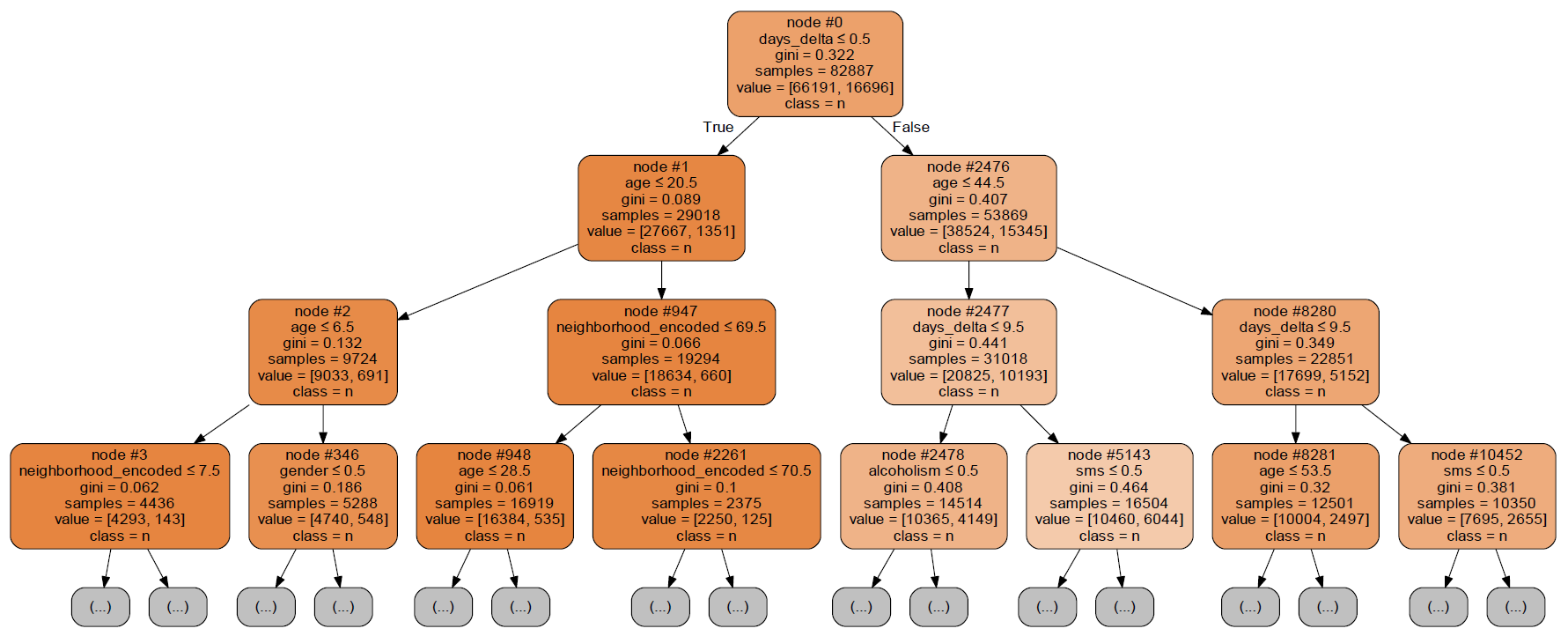
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Accuracy | Precision | Recall | F1-Score |
| DecisionTreeClassifier(  random\_state=42) | 0.72944 | 0.73002 | 0.72944 | 0.72973 |
| GaussianNB() | 0.77107 | 0.71233 | 0.77107 | 0.72942 |
| ComplementNB() | 0.69147 | 0.75024 | 0.69147 | 0.71329 |
| SVC(C=1, kernel='linear') | 0.79876 | 0.63393 | 0.79619 | 0.70586 |
| KNeighborsClassifier(  n\_neighbors=10, p=1) | 0.79338 | 0.72835 | 0.79011 | 0.73014 |
| RandomForestClassifier(  max\_depth=20, min\_samples\_leaf=4, min\_samples\_split=4, n\_estimators=600, random\_state=42) | 0.79873 | 0.74973 | 0.79927 | 0.72173 |

### Decision Tree

The decision tree is particularly conducive to visualization of machine learning problems. We use the graphviz library and the following function to achieve this:



The following graphic shows the first two splits:



The model was run using the default setting, which determines each split based on Gini impurity. For each leaf, the value indicates no-show number of samples in the format [no, yes]. We can see that the first split is performed on days\_delta, and that no-shows are far less likely when the appointment was scheduled less than 24 hours ahead of time. The split at the next two leafs is performed on age. We can see in both cases that younger patients are more likely to be no-shows.

### Naïve Bayes

In the end, the performance of Gaussian Naïve Bayes is not far off from random forest, and has the advantage of a much shorter runtime.

Complement Naïve Bayes did not perform well, though this was to be expected. Complement only excels with highly imbalanced data. In our data, no-show was not highly imbalanced; our no-show variable is split roughly 80/20 between No and Yes.

### SVM

While the accuracy score of SVM was comparable to random forest and kNN, its precision and f1-score were inferior. Additionally, it had by far the longest runtime of all our models.

### kNN

For kNN, the best perameters were as follows:

|  |  |
| --- | --- |
| Parameter | Value |
| algorithm | auto |
| metric | minkowski |
| n\_neighbors | 10 |
| p | 1 |
| weights | uniform |

Uniform weights outperformed “distance” (wherein closer points have greater influence than those further away). P is the power parameters for the Minkowski metric; In this case, p=1 means that Manhattan distance was used. The performance of kNN in our model is comparable to random forest, with the additional benefit of a much faster runtime.

### Random Forest

Our random forest grid included 100 combinations of parameters. With three-fold cross-validation, this results in 300 different models.

For random forest, the parameters and feature importances for the best model are as follows:

|  |  |
| --- | --- |
| Parameter | Value |
| bootstrap | True |
| max\_depth | 20 |
| min\_samples\_leaf | 2 |
| min\_samples\_split | 8 |
| n\_estimators | 800 |

|  |  |
| --- | --- |
| Feature | Importance |
| days\_delta' | 0.4 |
| 'age' | 0.23 |
| 'neighborhood\_encoded' | 0.21 |
| 'appt\_weekday' | 0.06 |
| 'sms' | 0.03 |
| 'gender' | 0.02 |
| 'scholarship' | 0.01 |
| 'hypertension' | 0.01 |
| 'diabetes' | 0.01 |
| 'alcoholism' | 0.01 |
| 'handicap' | 0 |

The top three features were days\_delta, age, then neighborhood. This corroborates the visualization from our decision tree model, which splits first on days\_delta, then on age. The fact that days\_delta, which was a derived feature, was the most important feature shows the importance of deliberate, thoughtful preprocessing and analysis of the dataset before setting out to build a classification model.

In conclusion, our classification modeling supports the use of random forest and kNN as the most accurate models to predict no-shows for this particular dataset.

# References

Gier, J. (2017, April 26). *Missed appointments cost the U.S. healthcare system $150B each year.* Retrieved from Hea!thCare Innovation: https://www.hcinnovationgroup.com/clinical-it

JoniHoppen. (2017). *Medical Appointment No Shows: Why do 30% of patients miss their scheduled appointments?* Retrieved from Kaggle.com: https://www.kaggle.com/joniarroba/noshowappointments

# Appendix A: Classification Code

import numpy as np

import pandas as pd

from sklearn import tree

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix, classification\_report

from sklearn.naive\_bayes import GaussianNB, ComplementNB

from sklearn.model\_selection import GridSearchCV, train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

import csv

import graphviz

######################################################################################

# PREPROCESSING

######################################################################################

def preprocess\_df(read\_file):

    df = pd.read\_csv(read\_file)

    df.drop(["PatientId", "AppointmentID"], axis=1, inplace=True)

    df.columns = ['gender', 'sched\_day', 'appt\_day', 'age', 'neighborhood', 'scholarship',

                'hypertension', 'diabetes', 'alcoholism', 'handicap', 'sms', 'no\_show']

    # convert gender and no\_show to binary

    df['gender'] = df['gender'].map({'M': 0, 'F': 1})

    df['no\_show'] = df['no\_show'].map({'No': 0, 'Yes': 1})

    # only 16 entries for handicap > [3,4], so drop these

    df.drop(df[df['handicap'] > 2].index, inplace=True)

    # convert date columns to time

    df['sched\_day'] = pd.to\_datetime(df['sched\_day'], infer\_datetime\_format=True)

    df['appt\_day'] = pd.to\_datetime(df['appt\_day'], infer\_datetime\_format=True)

    # create a new column for the time difference (in days) between when

    # an appointment was scheduled and the actual date of the appointment

    df['days\_delta'] = list(map(lambda x: x.days+1, df['appt\_day'] - df['sched\_day']))

    df = df[df['days\_delta'] >= 0]

    # create a new column for the day of the week (0 = Monday)

    df['appt\_weekday'] = df['appt\_day'].map(lambda x: x.dayofweek)

    # drop erroneous and outlier values for age

    df = df[(df["age"] != -1) & (df["age"] != 115)]

    # use LabelEncoder to categorize neighborhoods

    encoder = LabelEncoder()

    df['neighborhood\_encoded'] = encoder.fit\_transform(df['neighborhood'])

    # now drop the datetime columns and original neighborhood column

    df.drop(['sched\_day', 'appt\_day', 'neighborhood'], axis=1, inplace=True)

    return df

######################################################################################

# UTILITIES

######################################################################################

def get\_feature\_importances(model, x):

    feature\_dict = {}

    importances = list(model.feature\_importances\_)

    feature\_importances = [(col, round(i, 2)) for col, i in zip(list(x), importances)]

    feature\_importances = sorted(feature\_importances, key = lambda x: x[1], reverse=True)

    for i in feature\_importances:

        feature\_dict[i[0]] = i[1]

    return feature\_dict

def get\_model\_results(model, x\_test, y\_test, pred, x, grid=False):

    '''return dictionary of model results'''

    results = {}

    if grid:

        results['Model'] = model.best\_estimator\_

        results['Params'] = model.best\_params\_

        results['Score'] = model.best\_score\_

        try:

            results['Importances'] = get\_feature\_importances(model.best\_estimator\_, x)

        except:

            results['Importances'] = 'N/A'

    else:

        results['Model'] = model

        results['Params'] = model.get\_params()

        results['Score'] = model.score(x\_test, y\_test)

        try:

            results['Importances'] = get\_feature\_importances(model, x)

        except:

            results['Importances'] = 'N/A'

    results["Classification Report"] = classification\_report(y\_test, pred, output\_dict=True)

    return results

def print\_results(y\_test, pred, results):

    '''print confusion matrix and classification report'''

    for key in results:

        print(f"{key}:\t{results[key]}")

    print(f"Confusion Matrix\n{confusion\_matrix(y\_test, pred)}")

    print(f"Classification Report\n{classification\_report(y\_test, pred)}")

def visualize\_dt(dt, x):

    '''create visualization of the decision tree classifier'''

    dot\_data = tree.export\_graphviz(dt, out\_file=None,

                                    feature\_names=x.columns,

                                    class\_names='no\_show',

                                    filled=True,

                                    rounded=True,

                                    max\_depth=3,

                                    node\_ids=True,

                                    special\_characters=True)

    graph = graphviz.Source(dot\_data)

    # save graph to dt.pdf

    graph.render("dt")

def get\_grid\_search(model, grid):

    '''return GridSearchCV object'''

    grid\_search = GridSearchCV(estimator = model,

                               param\_grid = grid,

                               cv = 3,

                               n\_jobs = 20,

                               verbose = 2,

                               refit=True

                               )

    return grid\_search

######################################################################################

# MODELS

######################################################################################

######################################################################################

# Decision Tree

######################################################################################

def run\_dt(x\_train, x\_test, y\_train, y\_test, x, viz=False):

    dt = tree.DecisionTreeClassifier(random\_state = 42)

    dt.fit(x\_train, y\_train)

    dt\_pred = dt.predict(x\_test)

    if viz:

        visualize\_dt(dt, x)

    results = get\_model\_results(dt, x\_test, y\_test, dt\_pred, x)

    print\_results(y\_test, dt\_pred, results)

    return results

######################################################################################

# Random Forest

######################################################################################

def run\_rf(x\_train, x\_test, y\_train, y\_test, x):

    rf\_grid = {

    'bootstrap': [True],

    'max\_depth': [None, 20, 40, 60, 80],

    'min\_samples\_leaf': [2, 4],

    'min\_samples\_split': [4, 8],

    'n\_estimators': [200, 400, 600, 800, 1000]

    }

    rf = RandomForestClassifier(random\_state = 42)

    rf = get\_grid\_search(rf, rf\_grid)

    rf.fit(x\_train, y\_train)

    rf\_pred = rf.best\_estimator\_.predict(x\_test)

    results = get\_model\_results(rf, x\_test, y\_test, rf\_pred, x, grid=True)

    print\_results(y\_test, rf\_pred, results)

    return results

######################################################################################

# Naive Bayes

######################################################################################

def run\_gnb(x\_train, x\_test, y\_train, y\_test, x):

    '''Gaussian Naive Bayes model'''

    gnb = GaussianNB()

    gnb.fit(x\_train, y\_train)

    gnb\_pred = gnb.predict(x\_test)

    results = get\_model\_results(gnb, x\_test, y\_test, gnb\_pred, x)

    print\_results(y\_test, gnb\_pred, results)

    return results

def run\_comp\_nb(x\_train, x\_test, y\_train, y\_test, x):

    '''Complement Naive Bayes model'''

    comp\_nb = ComplementNB()

    comp\_nb.fit(x\_train, y\_train)

    comp\_nb\_pred = comp\_nb.predict(x\_test)

    results = get\_model\_results(comp\_nb, x\_test, y\_test, comp\_nb\_pred, x)

    print\_results(y\_test, comp\_nb\_pred, results)

    return results

######################################################################################

# kNN

######################################################################################

def run\_knn(x\_train, x\_test, y\_train, y\_test, x):

    grid = {

        'algorithm': ['auto'],

        'weights': ['uniform', 'distance'],

        'p': [1, 2, 3],

        'metric': ['minkowski'],

        'n\_neighbors': [2, 3, 5, 10]

    }

    knn = KNeighborsClassifier()

    knn = get\_grid\_search(knn, grid)

    knn.fit(x\_train, y\_train)

    knn\_pred = knn.best\_estimator\_.predict(x\_test)

    results = get\_model\_results(knn, x\_test, y\_test, knn\_pred, x, grid=True)

    print\_results(y\_test, knn\_pred, results)

    return results

######################################################################################

# SVM

######################################################################################

def run\_svm(x\_train, x\_test, y\_train, y\_test, x):

    grid = {'kernel': ['rbf', 'linear', 'sigmoid'],

            'gamma': [1e-3, 1e-4],

            'C': [1, 2]

            }

    svm = SVC()

    svm = get\_grid\_search(svm, grid)

    svm.fit(x\_train, y\_train)

    svm\_pred = svm.best\_estimator\_.predict(x\_test)

    results = get\_model\_results(svm, x\_test, y\_test, svm\_pred, x, grid=True)

    print\_results(y\_test, svm\_pred, results)

    return results

def main():

    read\_file = "KaggleV2-May-2016.csv"

    # fetch data file and preprocess

    df = preprocess\_df(read\_file)

    # create test and train data

    x = df.drop(['no\_show'], axis=1)

    y = df['no\_show']

    x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y)

    model\_dict = {

        'dt': run\_dt(x\_train, x\_test, y\_train, y\_test, x),

        'rf': run\_rf(x\_train, x\_test, y\_train, y\_test, x),

        'gnb': run\_gnb(x\_train, x\_test, y\_train, y\_test, x),

        'comp\_nb': run\_comp\_nb(x\_train, x\_test, y\_train, y\_test, x),

        'knn': run\_knn(x\_train, x\_test, y\_train, y\_test, x),

        'svm': run\_svm(x\_train, x\_test, y\_train, y\_test, x),

    }

    model\_df = pd.DataFrame(model\_dict).T

    model\_df.to\_csv("model\_output.csv", index=False)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# Appendix B

library(readr)

library(tidyverse)

library(ggplot2)

library(stringr)

library(scales)

library(plyr)

library(dplyr)

library(arules)

library(arulesViz)

library(factoextra)

df<-read.csv("C:/Users/katva/Desktop/no\_show.csv",

stringsAsFactors = TRUE,

header = TRUE,

na.strings = c("", " ", "NA"))

##ID number will throw the rules off, as everyone has a unique ID number, so the ID numbers will be ##removed from the dataset

dfNS<-df[,-1]

colnames(dfNS)

dfNS<-dfNS[,-1]

##Discretization of the data and converting numeric data to nominal data

dfNS$Age<- cut(dfNS$Age, breaks = c(0,10,20,30,40,50,60,Inf), labels = c("child","teen","twenties","thirties","forties","fifties","seniors"))

dfNS$Scholarship<- ifelse(dfNS$Scholarship == 0, "No", "Yes")

dfNS$Hipertension<-ifelse(dfNS$Hipertension == 0, "No", "Yes")

dfNS$Diabetes<-ifelse(dfNS$Diabetes == 0, "No", "Yes")

dfNS$Alcoholism<-ifelse(dfNS$Alcoholism == 0, "No", "Yes")

dfNS$Handcap<-ifelse(dfNS$Handcap == 0, "No", "Yes")

dfNS$SMS\_received<-ifelse(dfNS$SMS\_received == 0, "No", "Yes")

##Creating new column to calculate days between scheduling an appointment and the actual appointment date

dfNS$ScheduledDay<-as.Date(dfNS$ScheduledDay)

dfNS$AppointmentDay<-as.Date(dfNS$AppointmentDay)

dfNS$Total\_Days<- difftime(dfNS$AppointmentDay,dfNS$ScheduledDay, units = "days")

dfNS$Total\_Days<- as.numeric(dfNS$Total\_Days)

dfNS$Total\_Days<- cut(dfNS$Total\_Days, breaks = c(0,7,14,21,30,60,90,Inf), labels = c("within\_the\_week","the\_next\_week","two\_weeks\_out","three\_weeks\_out","one\_month","two\_months","three\_months\_or\_longer"))

##Now that we have days between appointment and scheduling, I need to remove the date columns

dfNS<-(dfNS[,-2:-3])

str(dfNS)

##Change characters to factors

dfNS$Scholarship<- as.factor(dfNS$Scholarship)

dfNS$Hipertension<-as.factor(dfNS$Hipertension)

dfNS$Diabetes<-as.factor(dfNS$Diabetes)

dfNS$Alcoholism<- as.factor(dfNS$Alcoholism)

dfNS$Handcap<- as.factor(dfNS$Handcap)

dfNS$SMS\_received<- as.factor(dfNS$SMS\_received)

##0 has been replaced by NA in the Total\_Days col. I need to change it to "same\_day" and correct levels.

library(forcats)

dfNS$Total\_Days<- fct\_explicit\_na(dfNS$Total\_Days, na\_level = "Same\_Day")

##Now looking for frequent variables in the data

frequentVariables <- eclat (dfNS, parameter = list(supp = 0.7, minlen = 2))

inspect(frequentVariables)

frequentVariables <- eclat (dfNS, parameter = list(supp = 0.8, minlen = 2))

inspect(frequentVariables)

##Now running apriori algorithm

noShow\_rules<-apriori(dfNS, parameter = list(supp=0.8, conf=0.5, maxlen = 12 ))

inspect(noShow\_rules)

##Rules by confiedence

rules\_by\_confidence<- sort(noShow\_rules, by="confidence", decreasing = TRUE)

inspect(head(rules\_by\_confidence,10))

##Run Apriori again with new parameters

noShow\_rules<-apriori (data=dfNS, parameter=list (supp=0.01,conf = 0.5), appearance = list(default="lhs", rhs="No.show=Yes" ), control = list (verbose=FALSE))

inspect(sort(noShow\_rules, by="lift", decreasing = TRUE))

noShow\_rules<-apriori (data=dfNS\_Yes, parameter=list (supp=0.01,conf = 0.5, minlen=3), appearance = list(default="lhs", rhs="No.show=Yes" ), control = list (verbose=FALSE))

inspect(head(sort(noShow\_rules, by="lift", decreasing = TRUE)))

##I want to look at it without neighborhood

dfNS\_Yes2<-(dfNS\_Yes[,-3])

noShow\_rules<-apriori (data=dfNS\_Yes2, parameter=list (supp=0.01,conf = 0.5, minlen=3), appearance = list(default="lhs", rhs="No.show=Yes" ), control = list (verbose=FALSE))

inspect(head(sort(noShow\_rules, by="lift", decreasing = TRUE)))

dfNS\_Yes<- dfNS %>% filter(No.show == "Yes")

str(dfNS\_Yes)

frequentVariables <- eclat (dfNS\_Yes, parameter = list(supp = 0.7, minlen = 2))

inspect(frequentVariables)

##Descriptive Statistics Look at the Data

age\_counts <- table(dfNS\_Yes2$Age)

barplot(age\_counts, main="No Shows by Age",xlab="age group", col = "green")

totAge<- table(dfNS$Age)

barplot(totAge, main= "total age counts", xlab = "age group", col = "green"

age\_counts <- table(dfNS$Neighbourhood)

barplot(age\_counts, main="Total Patients by Neighborhood",xlab="neighborhood", col = "light blue")

gender\_counts<- table(dfNS\_Yes2$Gender)

barplot(gender\_counts, main="no show by gender", xlab = "gender", col = "purple")

totGen<- table(dfNS$Gender)

barplot(totGen, main= "total gender counts", xlab = "gender", col = "purple")

Neigh\_counts <- table(dfNS$Neighbourhood)

barplot(Neigh\_counts, main="Total Patients by Neighborhood",xlab="neighborhood", col = "light blue"

Neigh\_counts <- table(dfNS\_Yes$Neighbourhood)

barplot(Neigh\_counts, main="Total Neighborhood Counts",xlab="neighborhood", col = "light blue")

##Linear Regression

lmModel<- lm(formula = No.show ~.,data=dfNS)

summary(fawn.lm)

lmModel<- lm(formula = No.show ~.,data=trainingData)

summary(lmModel)

lmModel2<- lm(formula = No.show ~ Age+Scholarship+Diabetes+Alcoholism+SMS\_received+Total\_Days,data=trainingData)

summary(lmModel2)

##ClusterAnalysis

library(readr)

library(tidyverse)

library(ggplot2)

library(stringr)

library(scales)

library(plyr)

library(dplyr)

library(arules)

library(arulesViz)

library(factoextra)

df<-read.csv("C:/Users/katva/Desktop/no\_show.csv",

stringsAsFactors = TRUE,

header = TRUE,

na.strings = c("", " ", "NA"))

str(df)

colnames(df)

##Prepare the dataset

dfNS<-df[,-1:-3]

colnames(dfNS)

dfNS$ScheduledDay<-as.Date(dfNS$ScheduledDay)

dfNS$AppointmentDay<-as.Date(dfNS$AppointmentDay)

dfNS$Total\_Days<- difftime(dfNS$AppointmentDay,dfNS$ScheduledDay, units = "days")

dfNS$Total\_Days<- as.numeric(dfNS$Total\_Days)

dfNS<-(dfNS[,-1:-2])

str(dfNS)

dfK<-(dfNS[,-2])

str(dfK)

dfK$Scholarship<- as.numeric(dfK$Scholarship)

dfK$Hipertension<- as.numeric(dfK$Hipertension)

dfK$Diabetes<- as.numeric(dfK$Diabetes)

dfK$Alcoholism<- as.numeric(dfK$Alcoholism)

dfK$Handcap<- as.numeric(dfK$Handcap)

dfK$SMS\_received<- as.numeric(dfK$SMS\_received)

dfK$Age<- as.numeric(dfK$Age)

dfK$Total\_Days<-as.numeric(dfK$Total\_Days)

dfK$No.show<-as.numeric(dfK$No.show)

str(dfK)

dfk<-na.omit(dfK)

dfk<-dfk[,-1]

str(dfk)

dim(dfK)

dfK[1:7000,]

randIndex<-sample(1:dim(dfK)[1]) #creating an index of random samples from the DF

head(randIndex)

length(randIndex)

#Create a 2/3 split

cutpoint2\_3<- floor(2\*dim(dfK)[1]/3)

cutpoint2\_3

# Now that data has been split, need to create training data set

trainingData<- dfK[randIndex[1:cutpoint2\_3],]

dim(trainingData)

head(trainingData)

# Now creating the test data set

testData<- dfK[randIndex[(cutpoint2\_3 + 1):dim(dfK)[1]],]

dim(testData)

head(testData)

##Elbow Plot to find number of Clusters

set.seed(123)

wss<-function(k){

return(kmeans(trainingData, k, nstart = 25)$tot.withinss)

}

k\_values<-1:5

wss\_values<- purrr::map\_dbl(k\_values, wss)

plot(x=k\_values, y=wss\_values, type = "b", frame = F, xlab = "Number of Clusters K", ylab = "Total Within-Clusters SUM of Squares")

##Running first round of Kmeans

result<-kmeans(trainingData, centers=2, nstart= 25, iter.max = 100, algorithm = "Hartigan-Wong")

result

fviz\_cluster(result, data = trainingData)

##Second run with 3 Clusters

result<-kmeans(trainingData, centers=3, nstart= 25, iter.max = 100, algorithm = "Hartigan-Wong")

result

##running last model using testing data

result<-kmeans(testData, centers=3, nstart= 25, iter.max = 100, algorithm = "Hartigan-Wong")

result

##Hierarchical Clustering required a much smaller Gb of data, so I shortened the training data set to meet the Gb requirements.

trainingDataShort<- trainingData[1:3000,]

library(FactoMineR)

HCPC<- HCPC(trainingDataShort, nb.clust = 3, min = 3, max = NULL, graph = TRUE)