Appointment No Show Predictions GH1019736 AI

July 19, 2023

1 "No Show" Appointments Prediction for Patients (Classification Task) M507

2 Submitted by GH1019736

2.1 Table of contents

2.2 1. Problem Statement

Shoukat Khanam is the biggest cancer Hospital in Pakistan. One of the major challenges facing the Hospital is the failure of patients to show up for Medical appointments. The failure of patients to show up for their medical appointments costs the hospital a lot of money because Specialist Consultants are very expensive to book for appointments with patients. In a recent management meeting, the accounts department suggested that if it was possible to predict "No-show" appointments, the hospital will be able to cut down on some expenses associated with patients not showing up for their appointments. The hospital's head of research stated that his department lacked the technical expertise to predict the no-show expertise and perhaps a data scientist will be the best person to help design a model capable of predicting 'No-show' appointments. After series of consultations with various experts, the Research Department recommended to the management deparment that they need to engage a Data Scientist who will develope a machine learning pipeline capable of predicting the no show appointment.

One of my friends who is a Specialist Consultant Doctor with Shoukat Khanam called me and told me about the challenges they are facing with the "No-show" appointments and how . He knows I work in a Data Science Company as a Data Scientist and he would like to engage the services of the company I work with so I referred him to the CEO of the company I work with. After a series of discussion between the Management of the hospital and the Management of the Data Science Company, I was assigned by my Boss to build a Machine Learning Pipeline for prediction fo "No-show" appointments. Application of a "No-show" appointment model will help the hospital reduce Specialist Consultants Financial loss, the hospital's financial loss, and the patients, opportunity loss.

I was provided with a dataset in csv format and the dataset has following features:

- PatientId
- AppointmentID
- Gender
- ScheduledDay
- AppointmentDay
- Age
- Neighbourhood
- Scholarship
- Hipertension
- Diabetes
- Alcoholism
- Handcap
- SMS_received
- No-show

My task is to create a Machine Learning Model using the data provided that can be used to predict the "No-show". Because the target variable is "No Show" this will be supervised learning classification problem.

2.3 2. Importing Libraries and Packages and Data

The first step in this machine learning pipeline is to load all the libraries, and packages utilized within the pipeline. For ease of navigating the notebook, all packages and libraries used in this pipeline are listed in the cell below by category such as basic libraries, libraries for preprocessing, machine learning models and remove warnings.

2.3.1 2.1 Importing Libraries with Packages

[1]:
```
# 1.Basic Libararies
import numpy as np # Linear Algebra
np.random.seed(100)
import pandas as pd # functions for analyzing, cleaning, exploring, and␣
↪manipulating data
import seaborn as sns # for graphical representation
import matplotlib.pyplot as plt #for visualize the data
from matplotlib import pyplot
%matplotlib inline
```

```python
# 2.Libararies for Preprocess
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.metrics import confusion_matrix

# 3.Machine Learning Models
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.linear_model import Perceptron
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

#remove warning
import warnings
warnings.filterwarnings ('ignore')
```

2.3.2 2.2 Loading Datasets

[2]:
```python
# Data link is available on Refrence section
No_show = pd.read_csv("D:/Data Science/MoP and deep learning/Adnan Khalid/
↪archive.zip")
```

Using pandas, I have loaded the dataset and assigned it to a dataframe called No-show.

2.3.3 2.3 Splitting Datasets into Train and Test Dataset

Due to limited computing power, we will only be using 5% of the dataset. This has become necessary as using the entire dataset uses a lot of computing power, and takes a very long time to run successfuly. To this end, the "No-show" dataset is split into "use data" and "not use data".

[3]:
```python
# Part the dataset into train and test set
use_data, not_use_data = train_test_split(No_show, test_size=0.95)
print(f'use data; {use_data.shape}.')
print(f' not_use_data; {not_use_data.shape}.')
```

```
use data; (5526, 14).
 not_use_data; (105001, 14).
```

After splitting the dataset into "use data" and "not use data", the following step will be to devide "use data" into training and test datasets. To follow lines of codes will enable us to achieve that. For the purpose of this pipeline, the training data will be called "train_noshow" while the testing data is called "test_noshow"

[4]:
```python
# Splitting the dataset into training and testing datasets
train_noshow, test_noshow = train_test_split(use_data, test_size=0.2)
```

```
keras_use_data = use_data.copy()
print(f'training no show; {train_noshow.shape}.')
print(f'testing no show; {test_noshow.shape}.')
```

```
training no show; (4420, 14).
testing no show; (1106, 14).
```

The above cell shows that we devided our data into train and test.I just called the Library scikitlearn and use the function train-test-split. Moreover, we will use only training no show to train our model while testing no show will be used to assess ou r Model. In this way, we can compare our actual and predicted values.

2.4 3. Data Exploration

Presently that we have loaded all the libraries, packages and data, we will now able to explore our dataset so that we can familiarise ourselves with the data. The first step is to see what our data looks like

2.4.1 3.1 First View of Dataset

[5]:
```
# lets call our train data in top 3 rows
train_noshow.head(3)
```

[5]:
```
          PatientId  AppointmentID Gender          ScheduledDay  \
11322  7.988424e+12        5746188      M  2016-05-30T13:34:55Z
27684  6.849595e+12        5670140      F  2016-05-06T13:07:07Z
65726  9.943819e+13        5699797      F  2016-05-16T08:50:59Z

             AppointmentDay  Age        Neighbourhood  Scholarship  \
11322  2016-05-30T00:00:00Z   55  ILHA DE SANTA MARIA            0
27684  2016-05-10T00:00:00Z   55           ANDORINHAS            0
65726  2016-05-16T00:00:00Z   70           TABUAZEIRO            0

       Hipertension  Diabetes  Alcoholism  Handcap  SMS_received No-show
11322             0         0           0        0             0      No
27684             1         0           0        0             1      No
65726             1         0           0        0             0      No
```

Above we can see all the features columns our dataset. One important thing to note is the encoding of the last column. “No” means on the calendar patient showed up to appointment and “Yes” it means they did not show up on the calendar.

2.4.2 3.2 Checking Data Types and Missing Values

[6]:
```
# Let us look at few a metadata such as which type of our data,numbers of rows␣
↪and columns, memory usage and others associated with our dataset
train_noshow.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4420 entries, 11322 to 98639
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   PatientId       4420 non-null   float64
 1   AppointmentID   4420 non-null   int64
 2   Gender          4420 non-null   object
 3   ScheduledDay    4420 non-null   object
 4   AppointmentDay  4420 non-null   object
 5   Age             4420 non-null   int64
 6   Neighbourhood   4420 non-null   object
 7   Scholarship     4420 non-null   int64
 8   Hipertension    4420 non-null   int64
 9   Diabetes        4420 non-null   int64
 10  Alcoholism      4420 non-null   int64
 11  Handcap         4420 non-null   int64
 12  SMS_received    4420 non-null   int64
 13  No-show         4420 non-null   object
dtypes: float64(1), int64(8), object(5)
memory usage: 518.0+ KB
```

We can see that 'Gender' and 'Neighbourhood are categoricals columns and other some useless columns. Later we will work on them.

The following cell will reveal all the columns names in the dataset and check for null values in the data. It is compulsory in helping us to determine what type of data preprocessing will need to be carried out. Columns with Object datatype will have to be converted to numbers for easy processing by the machine learning models.

[7]:
```
# I am going to check missing values in training data
print( 'Null values in our Training data columns :\n ', train_noshow.isnull().
 ↪sum( ) )
print("-"
*40)
```

```
Null values in our Training data columns :
  PatientId         0
AppointmentID     0
Gender            0
ScheduledDay      0
AppointmentDay    0
Age               0
Neighbourhood     0
Scholarship       0
Hipertension      0
Diabetes          0
Alcoholism        0
Handcap           0
```

```
SMS_received      0
No-show           0
dtype: int64
----------------------------------------
```

You can see in the above we found that there is no missing values in our training data set. This means we would not need to fill in missing values.

2.4.3 3.3 Statistical Summary of Numerical features

Our dataset is comprised of several columns. PatientId and AppointmentID are not really relevant to our analysis so we do not need summary satistics for it. We also do not need summary statistics for categorical data like Gender, Neighbourhood, Scholarship, Hipertension, Diabetes, Alcoholism, Handcap, SMS_received and No-show. ScheduledDay and Appointment are in Datetime data and we would not also be doing summary statistics for it. Only the Age Colum is really numerical so we would be examing the summary statistics for the Age column.

[8]:
```
# Lets check only age statistic summary in dataset
train_noshow['Age'].describe(include='all')
```

[8]:
```
count    4420.000000
mean       37.259729
std        23.195753
min         0.000000
25%        17.750000
50%        37.000000
75%        56.000000
max        97.000000
Name: Age, dtype: float64
```

From the above cell, we can see that there are a total of f4420 values for age, and the mean age of the patients is 37.25. The standard deviation of the patients' age is 23.19. The youngest patient is 0 years old while the oldest patient is 97 years old.

2.4.4 3.4 Correlation Matrix

Here, we will look at the correlation between all the variables with one another.

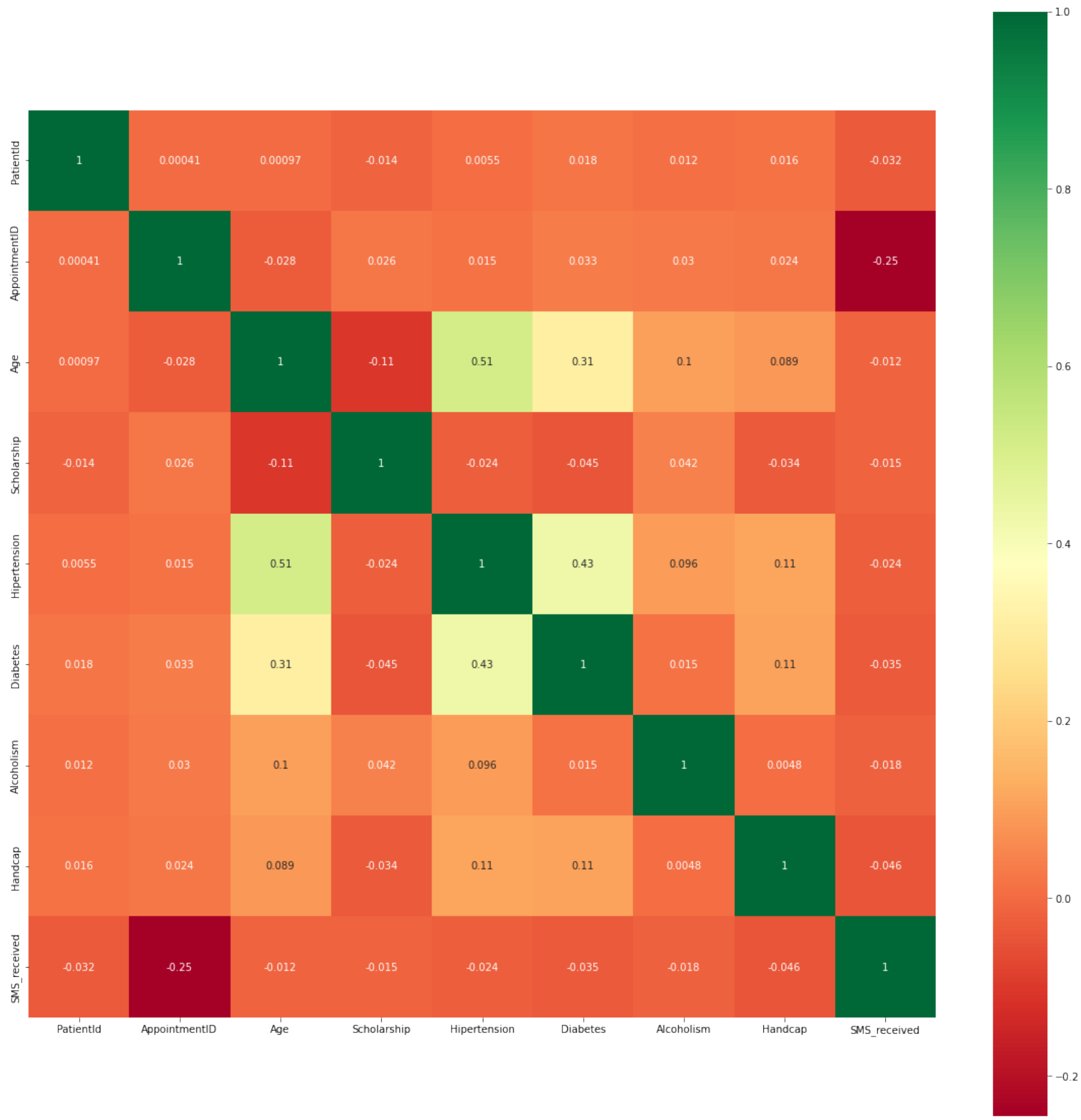
[9]:
```
# We are going to check correlation among all the features
train_noshow_correlation_matrix = train_noshow.corr()
train_noshow_correlation_matrix
```

[9]:
```
               PatientId  AppointmentID       Age  Scholarship  Hipertension  \
PatientId       1.000000       0.000406  0.000973    -0.014002      0.005511
AppointmentID   0.000406       1.000000 -0.027986     0.025914      0.015267
Age             0.000973      -0.027986  1.000000    -0.109016      0.509073
Scholarship    -0.014002       0.025914 -0.109016     1.000000     -0.024170
Hipertension    0.005511       0.015267  0.509073    -0.024170      1.000000
Diabetes        0.017851       0.033206  0.310354    -0.044786      0.433700
```

```
Alcoholism       0.011943       0.030042  0.102568     0.041733      0.096285
Handcap          0.015997       0.024235  0.089281    -0.034048      0.110462
SMS_received    -0.032342      -0.245639 -0.012405    -0.015039     -0.024306

               Diabetes  Alcoholism   Handcap  SMS_received
PatientId      0.017851    0.011943  0.015997     -0.032342
AppointmentID  0.033206    0.030042  0.024235     -0.245639
Age            0.310354    0.102568  0.089281     -0.012405
Scholarship   -0.044786    0.041733 -0.034048     -0.015039
Hipertension   0.433700    0.096285  0.110462     -0.024306
Diabetes       1.000000    0.015341  0.107506     -0.035174
Alcoholism     0.015341    1.000000  0.004830     -0.018189
Handcap        0.107506    0.004830  1.000000     -0.045787
SMS_received  -0.035174   -0.018189 -0.045787      1.000000
```

For easier understanding the following is a heatmap matrix of the table above

[10]:
```python
# visualize heatmap of Correlation matrix
plt.figure(figsize=(20,20)) # fig size
plot = sns.heatmap(train_noshow.corr(), annot=True, cmap='RdYlGn', square=True)
```



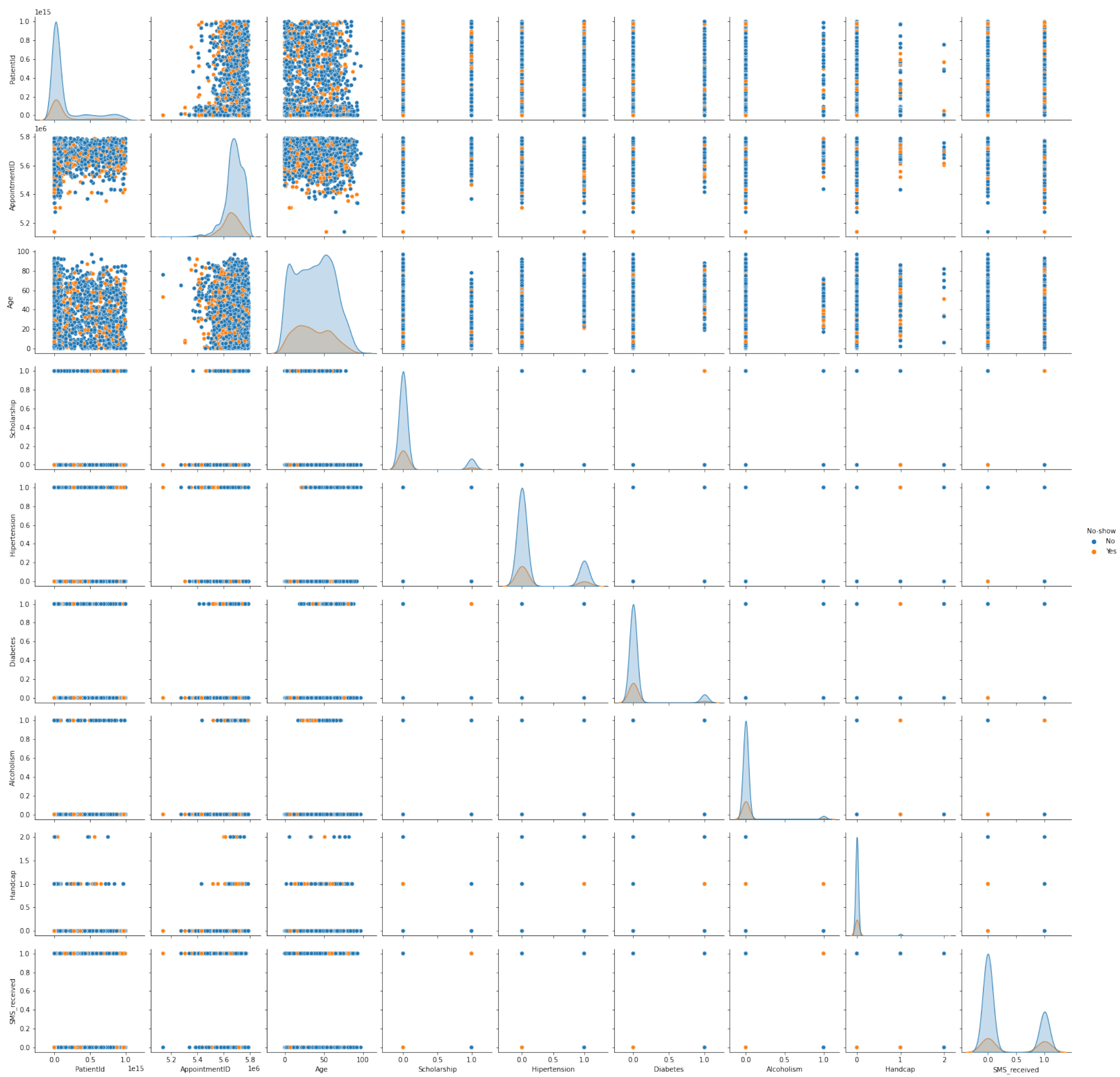
From the Heatmap visualization of the correlarion metrixs, we can see the nature of the relationships between the various variables in the dataset. Some columns such as Gender, ScheduledDay, AppointmentDay, Neighbourhood and No-show did not show up in the correlation matrix because of their datatypes. We are yet to preprocess the data so those columns are not in formats that can be analysed at the moment. However, we can see that there is no autocorrelation in our dataset.

2.4.5 3.5 Pairplots

The pairplots is another way of visualising the relationship between the various variables and the target variables, using scatter plot. Just like the correlation matrix, not all columns are represented in the pairplots because we have not yet preprocessed the dataset.

```
[11]: sns.pairplot(train_noshow, hue= "No-show")
```

```
[11]: <seaborn.axisgrid.PairGrid at 0x9203001fd0>
```



2.5 4. Prepare Our Data

Data preprocessing step in this pipeline will involve creating new features, dropping useless features, encoding categorical variables and selection of our descriptive and target variabes. The purpose of this step is to ensure that the data is in a form that the model can use.

Based on our data exploration we can see that PatientId',‘AppointmentID’, columns are not usefull for our target values. We also drop'ScheduledDay'and ‘AppointmentDay’ columns because we just need their differnce between days to build our Model. Moreover, There are categorical values, so we need to change them in binary numbers by using OneHotEncoding to convert in numericals values.Machine Learning needs numbers to train itself.

2.5.1 4.1 Creating New Features

Since we want to predict No-show appointments, we will create a new feature which will be the difference between ScheduledDay and AppointmentDay. This new feature may help improve the quality of our model.

[12]:
```
# first convert datetime data to datetime format
train_noshow['ScheduledDay'] = train_noshow['ScheduledDay'].
↪astype('datetime64[ns]')
train_noshow['AppointmentDay'] = train_noshow['AppointmentDay'].
↪astype('datetime64[ns]')

# difference in days
# create a new variabe which is the difference between scehduled day and␣
↪appointment day
train_noshow ['diff_days'] = (train_noshow ['ScheduledDay'] - train_noshow␣
↪['AppointmentDay']) / np.timedelta64(1, 'D')
train_noshow ['diff_days']
train_noshow.head(5)
```

[12]:
```
            PatientId  AppointmentID Gender        ScheduledDay AppointmentDay  \
11322    7.988424e+12        5746188      M 2016-05-30 13:34:55     2016-05-30
27684    6.849595e+12        5670140      F 2016-05-06 13:07:07     2016-05-10
65726    9.943819e+13        5699797      F 2016-05-16 08:50:59     2016-05-16
101199   7.386683e+14        5773198      F 2016-06-06 06:54:10     2016-06-06
37881    6.577984e+14        5719427      M 2016-05-19 11:14:42     2016-05-19

        Age        Neighbourhood  Scholarship  Hipertension  Diabetes  \
11322    55  ILHA DE SANTA MARIA            0             0         0
27684    55           ANDORINHAS            0             1         0
65726    70           TABUAZEIRO            0             1         0
101199   62       PARQUE MOSCOSO            0             0         0
37881    78           CONSOLAÇÃO            0             1         0

        Alcoholism  Handcap  SMS_received No-show  diff_days
11322            0        0             0      No   0.565914
27684            0        0             1      No  -3.453391
65726            0        0             0      No   0.368738
101199           0        0             0      No   0.287616
37881            0        0             0      No   0.468542
```

From the above we can see that we have created a new variable in the dataset called "diff_days" which is the differance between scheduled days and appointment days.

2.5.2 4.2 Dropping useless Data

It is time to drop some useless coulmns which will not help our model. The columns we will be droppign are 'PatientId','AppointmentID', 'ScheduledDay','AppointmentDay'. 'ScheduledDay' and 'AppointmentDay' are no longer relevant because we have already used them in creating a new feature. I have used the drop() funtion from Pandas library to drop the useless features.

```
[13]: # Drop useless columns
train_noshow = train_noshow.drop(['PatientId','AppointmentID',␣
↪'ScheduledDay','AppointmentDay'], axis = 1)
# converting target variable in binary number by using lambda function
train_noshow['No-show'] = train_noshow['No-show'].apply(lambda x: 0 if x.
↪strip()=='No'
                                                        else 1)
train_noshow.head(3)
```

```
[13]:       Gender  Age         Neighbourhood  Scholarship  Hipertension  Diabetes  \
11322       M   55  ILHA DE SANTA MARIA            0             0         0
27684       F   55           ANDORINHAS            0             1         0
65726       F   70           TABUAZEIRO            0             1         0

       Alcoholism  Handcap  SMS_received  No-show  diff_days
11322           0        0             0        0   0.565914
27684           0        0             1        0  -3.453391
65726           0        0             0        0   0.368738
```

2.5.3 4.3 OneHotEncoding for Categorical Data

In this section, we will use OneHotEncoder to convert our Categorical features in binary numbers. I will store my categorical data in new object and call my training data with categoricals columns. After this step, I will create one hot encoding instance and pass it through the encoder by using Pandas. Thus, we will safe this data in new object

```
[14]: # OneHotEncode the Categorical Data
enc_train_noshow = train_noshow[['Gender','Neighbourhood']]

#create an instance of one-hot-encoding
encode = OneHotEncoder(handle_unknown='ignore')

# passsing our data through the encoder
encode_df = pd.DataFrame(encode.fit_transform(enc_train_noshow).toarray())
encode_df
```

```
[14]:      0    1    2    3    4    5    6    7    8    9  …   69   70   71  \
0      0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0
1      1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0
2      1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0
3      1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0
4      0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0
…    …    …   …   …   …   …   …   …   …   …   …   …   …   …   …
4415   1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0
4416   1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0
```

```
4417  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0
4418  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0
4419  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0

       72   73   74   75   76   77   78
0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
1     0.0  0.0  0.0  0.0  0.0  0.0  0.0
2     0.0  0.0  0.0  0.0  1.0  0.0  0.0
3     0.0  0.0  0.0  0.0  0.0  0.0  0.0
4     0.0  0.0  0.0  0.0  0.0  0.0  0.0
…     …    …    …    …    …    …    …
4415  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4416  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4417  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4418  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4419  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[4420 rows x 79 columns]
```

The above cell showed that" our categorical_data has converted to binary numbers (0 and 1) successfully.

In below cell, I will drop 'Gender','Neighbourhood' columns from my orignal training data. After that, binary data will be merged with training data by using pd.concat.

[15]:
```python
#concat with original data
drop_some_columns = train_noshow.drop(['Gender','Neighbourhood'], axis=1)
drop_some_columns.index = encode_df.index
train_merged_data = pd.concat([drop_some_columns, encode_df], axis=1)

train_merged_data
```

[15]:
```
      Age  Scholarship  Hipertension  Diabetes  Alcoholism  Handcap  \
0      55            0             0         0           0        0
1      55            0             1         0           0        0
2      70            0             1         0           0        0
3      62            0             0         0           0        0
4      78            0             1         0           0        0
…     …          …            …           …         …       …
4415   54            0             0         0           0        0
4416   12            0             0         0           0        0
4417   34            0             0         0           0        0
4418    7            0             0         0           0        0
4419   29            0             0         0           0        0

      SMS_received  No-show  diff_days    0  …   69   70   71   72   73  \
0                0        0   0.565914  0.0  …  0.0  0.0  0.0  0.0  0.0
1                1        0  -3.453391  1.0  …  0.0  0.0  0.0  0.0  0.0
```

```
2                  0         0   0.368738  1.0  …  0.0  0.0  0.0  0.0  0.0
3                  0         0   0.287616  1.0  …  0.0  0.0  0.0  0.0  0.0
4                  0         0   0.468542  0.0  …  0.0  0.0  0.0  0.0  0.0
…             …       …         …   …   …   …   …   …   …   …
4415               0         0  -1.337014  1.0  …  0.0  0.0  0.0  0.0  0.0
4416               1         0  -6.603102  1.0  …  0.0  0.0  0.0  0.0  0.0
4417               1         0 -16.367535  1.0  …  0.0  0.0  0.0  0.0  0.0
4418               0         1  -6.397870  0.0  …  0.0  0.0  0.0  0.0  0.0
4419               1         0 -19.313808  1.0  …  0.0  0.0  0.0  0.0  0.0

       74   75   76   77   78
0     0.0  0.0  0.0  0.0  0.0
1     0.0  0.0  0.0  0.0  0.0
2     0.0  0.0  1.0  0.0  0.0
3     0.0  0.0  0.0  0.0  0.0
4     0.0  0.0  0.0  0.0  0.0
…    …   …   …   …   …
4415  0.0  0.0  0.0  0.0  0.0
4416  0.0  0.0  0.0  0.0  0.0
4417  0.0  0.0  0.0  0.0  0.0
4418  0.0  0.0  0.0  0.0  0.0
4419  0.0  0.0  0.0  0.0  0.0

[4420 rows x 88 columns]
```

2.5.4 4,4 Scaling Data

The features in the dataset at this stage are not in the same scale and this can cause some bias in our modelling process so we will need to ensure that they are in thesame scale. Specifically, 'Age' and 'diff_days' need to be brought to thesame scale as the other features in the dataset

[16]:
```python
# Scaling Dataset
normalize_columns = ['Age','diff_days']
train_merged_data[normalize_columns] = StandardScaler().
 ↪fit_transform(train_merged_data[normalize_columns])
train_merged_data.columns =train_merged_data.columns.astype(str)
train_merged_data
```

[16]:
```
           Age  Scholarship  Hipertension  Diabetes  Alcoholism  Handcap  \
0     0.764893            0             0         0           0        0
1     0.764893            0             1         0           0        0
2     1.411637            0             1         0           0        0
3     1.066707            0             0         0           0        0
4     1.756566            0             1         0           0        0
…          …          …            …         …          …       …
4415  0.721777            0             0         0           0        0
4416 -1.089104            0             0         0           0        0
```

```
4417 -0.140547            0             0         0            0        0
4418 -1.304685            0             0         0            0        0
4419 -0.356128            0             0         0            0        0

      SMS_received  No-show  diff_days    0  …   69   70   71   72   73  \
0                0        0   0.685842  0.0  …  0.0  0.0  0.0  0.0  0.0
1                1        0   0.416101  1.0  …  0.0  0.0  0.0  0.0  0.0
2                0        0   0.672610  1.0  …  0.0  0.0  0.0  0.0  0.0
3                0        0   0.667165  1.0  …  0.0  0.0  0.0  0.0  0.0
4                0        0   0.679308  0.0  …  0.0  0.0  0.0  0.0  0.0
…              …      …        …   …   …   …   …   …   …   …
4415             0        0   0.558134  1.0  …  0.0  0.0  0.0  0.0  0.0
4416             1        0   0.204720  1.0  …  0.0  0.0  0.0  0.0  0.0
4417             1        0  -0.450586  1.0  …  0.0  0.0  0.0  0.0  0.0
4418             0        1   0.218493  0.0  …  0.0  0.0  0.0  0.0  0.0
4419             1        0  -0.648314  1.0  …  0.0  0.0  0.0  0.0  0.0

       74   75   76   77   78
0     0.0  0.0  0.0  0.0  0.0
1     0.0  0.0  0.0  0.0  0.0
2     0.0  0.0  1.0  0.0  0.0
3     0.0  0.0  0.0  0.0  0.0
4     0.0  0.0  0.0  0.0  0.0
…    …   …   …   …   …
4415  0.0  0.0  0.0  0.0  0.0
4416  0.0  0.0  0.0  0.0  0.0
4417  0.0  0.0  0.0  0.0  0.0
4418  0.0  0.0  0.0  0.0  0.0
4419  0.0  0.0  0.0  0.0  0.0

[4420 rows x 88 columns]
```

[17]:
```
scaled_data = train_merged_data
target_name = 'No-show'
training_target = scaled_data[target_name]
training_features = scaled_data.drop([target_name], axis=1)
```

2.5.5 B_Step Test Data Preprocessing and Feature Engineering

It is time to do all the steps for testing data as we did in our training data.So we do not need to explain every cell. In below cells, what we will do:

1) Change date time data into date time formate and take differnace of the days and store in new column.
2) we will drop useless columns
3) OneHotEncode the Categorical Data

4) columns to be normalized in test data

5) scaled the data in test set

[18]:
```
# first convert datetime data to datetime format
test_noshow['ScheduledDay'] = test_noshow['ScheduledDay'].
 ↪astype('datetime64[ns]')
test_noshow['AppointmentDay'] = test_noshow['AppointmentDay'].
 ↪astype('datetime64[ns]')

# difference in days
# create a new variable which is the difference between scehduled day and␣
 ↪appointment day
test_noshow ['diff_days'] = (test_noshow ['ScheduledDay'] - test_noshow␣
 ↪['AppointmentDay']) / np.timedelta64(1, 'D')
test_noshow ['diff_days']
```

[18]:
```
35803    -6.504595
2293     -1.528368
86870    -0.364444
90043    -0.541343
104221    0.389213
            …
34651    -6.426227
48194    -1.682373
89361     0.707442
69151     0.668785
94923     0.605035
Name: diff_days, Length: 1106, dtype: float64
```

[19]:
```
# Drop useless columns
test_noshow = test_noshow.drop(['PatientId','AppointmentID',␣
 ↪'ScheduledDay','AppointmentDay'], axis = 1)

# test data in binary numbers
test_noshow['No-show'] = test_noshow['No-show'].apply(lambda x: 0 if x.
 ↪strip()=='No'
                                                       else 1)
test_noshow.head(3)
```

[19]:
```
      Gender  Age Neighbourhood  Scholarship  Hipertension  Diabetes  \
35803      F    6      DA PENHA            1             0         0
2293       M   49     DO CABRAL            0             1         0
86870      F   64       MARUÍPE            0             1         0

       Alcoholism  Handcap  SMS_received  No-show  diff_days
35803           0        0             0        0  -6.504595
```

```
2293             0        0             0        0  -1.528368
86870            0        0             0        0  -0.364444
```

[20]:
```python
# Here OneHotEncode the Categorical Data

enc_test_noshow = test_noshow[['Gender','Neighbourhood']]

#passsing our data through the encoder
encode_df = pd.DataFrame(encode.transform(enc_test_noshow).toarray())

#concat with original data
testing_drop_some_columns = test_noshow.drop(['Gender','Neighbourhood'], axis=1)
testing_drop_some_columns.index = encode_df.index
testing_merged_data = pd.concat([testing_drop_some_columns, encode_df], axis=1)

testing_merged_data
```

[20]:
```
      Age  Scholarship  Hipertension  Diabetes  Alcoholism  Handcap  \
0       6            1             0         0           0        0
1      49            0             1         0           0        0
2      64            0             1         0           0        0
3      77            0             0         0           0        0
4      58            0             1         0           0        0
…     …            …             …         …         …        …
1101   24            0             0         0           0        0
1102   35            0             0         0           0        0
1103   46            0             0         0           0        0
1104   38            0             0         0           0        0
1105   15            1             0         0           0        0

      SMS_received  No-show  diff_days    0  …   69   70   71   72   73  \
0                0        0  -6.504595  1.0  …  0.0  0.0  0.0  0.0  0.0
1                0        0  -1.528368  0.0  …  0.0  0.0  0.0  0.0  0.0
2                0        0  -0.364444  1.0  …  0.0  0.0  0.0  0.0  0.0
3                0        1  -0.541343  1.0  …  0.0  0.0  0.0  0.0  0.0
4                0        0   0.389213  0.0  …  0.0  0.0  0.0  0.0  0.0
…              …        …        …  …  …  …  …  …  …  …
1101             1        0  -6.426227  1.0  …  0.0  0.0  0.0  0.0  0.0
1102             0        0  -1.682373  1.0  …  0.0  0.0  0.0  0.0  0.0
1103             0        0   0.707442  1.0  …  0.0  0.0  0.0  0.0  0.0
1104             0        0   0.668785  0.0  …  0.0  0.0  0.0  0.0  0.0
1105             0        0   0.605035  1.0  …  0.0  0.0  0.0  1.0  0.0

       74   75   76   77   78
0     0.0  0.0  0.0  0.0  0.0
1     0.0  0.0  0.0  0.0  0.0
2     0.0  0.0  0.0  0.0  0.0
```

```
3     0.0  0.0  0.0  0.0  0.0
4     0.0  0.0  0.0  0.0  0.0
…   …   …   …   …   …
1101  0.0  0.0  0.0  0.0  0.0
1102  0.0  0.0  0.0  0.0  0.0
1103  0.0  0.0  0.0  0.0  1.0
1104  0.0  0.0  0.0  0.0  0.0
1105  0.0  0.0  0.0  0.0  0.0

[1106 rows x 88 columns]
```

[21]:
```
normalize_columns = ['Age','diff_days']
testing_merged_data[normalize_columns] = StandardScaler().
 ↪fit_transform(testing_merged_data[normalize_columns])
testing_merged_data.columns = testing_merged_data.columns.astype(str)

testing_merged_data
```

[21]:
```
           Age  Scholarship  Hipertension  Diabetes  Alcoholism  Handcap  \
0    -1.315682            1             0         0           0        0
1     0.518713            0             1         0           0        0
2     1.158618            0             1         0           0        0
3     1.713202            0             0         0           0        0
4     0.902656            0             1         0           0        0
…          …          …            …        …          …       …
1101 -0.547796            0             0         0           0        0
1102 -0.078532            0             0         0           0        0
1103  0.390732            0             0         0           0        0
1104  0.049449            0             0         0           0        0
1105 -0.931739            1             0         0           0        0

      SMS_received  No-show  diff_days    0  …   69   70   71   72   73  \
0                0        0   0.212166  1.0  …  0.0  0.0  0.0  0.0  0.0
1                0        0   0.532479  0.0  …  0.0  0.0  0.0  0.0  0.0
2                0        0   0.607399  1.0  …  0.0  0.0  0.0  0.0  0.0
3                0        1   0.596013  1.0  …  0.0  0.0  0.0  0.0  0.0
4                0        0   0.655911  0.0  …  0.0  0.0  0.0  0.0  0.0
…             …        …          …  …  …  …  …  …  …  …
1101             1        0   0.217210  1.0  …  0.0  0.0  0.0  0.0  0.0
1102             0        0   0.522566  1.0  …  0.0  0.0  0.0  0.0  0.0
1103             0        0   0.676395  1.0  …  0.0  0.0  0.0  0.0  0.0
1104             0        0   0.673907  0.0  …  0.0  0.0  0.0  0.0  0.0
1105             0        0   0.669804  1.0  …  0.0  0.0  0.0  1.0  0.0

       74   75   76   77   78
0     0.0  0.0  0.0  0.0  0.0
1     0.0  0.0  0.0  0.0  0.0
```

```
2      0.0  0.0  0.0  0.0  0.0
3      0.0  0.0  0.0  0.0  0.0
4      0.0  0.0  0.0  0.0  0.0
…    …   …   …   …   …
1101   0.0  0.0  0.0  0.0  0.0
1102   0.0  0.0  0.0  0.0  0.0
1103   0.0  0.0  0.0  0.0  1.0
1104   0.0  0.0  0.0  0.0  0.0
1105   0.0  0.0  0.0  0.0  0.0

[1106 rows x 88 columns]
```

2.5.6 Defining Feature and Target Variables

[22]:
```
testing_scaled_data = testing_merged_data
target_name = 'No-show'
testing_target = testing_scaled_data[target_name]
testing_features = testing_scaled_data.drop([target_name], axis=1)
```

2.6 5. Model Training and Evaluation

In this section, I will apply some supervised machine learning algorithms to our preprocessed data. The models we will be using for this pipline include random forest classifier, perceptron,Gradient boosting classifier and keras. Because in previous steps, we had prepared our data. So, we have greater understanding with our data. We also need to apply parameters to get high accuracy.However, we will go through all our models to see which model is giving good result. we will check documentation with Scikit_Learn to run our models.

2.6.1 5.1 Random Forest Classifier (GridSearchCV)

[23]:
```
parameters_grid = {
    "criterion":["gini","entropy"],
    "n_estimators": range(50,200,300),
}

rfc_model =  GridSearchCV(RandomForestClassifier(),
                          parameters_grid, scoring="accuracy", cv=5, n_jobs=-1)

rfc_model.fit(training_features, training_target)
```

[23]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'n_estimators': range(50, 200, 300)},
             scoring='accuracy')
```

[24]:
```
rfc_model.best_score_
```

[24]: 0.781447963800905

[25]:
```
rfc_model.best_params_
```

[25]: {'criterion': 'entropy', 'n_estimators': 50}

Result:

Above cell we got 77 percent score from the random forest classifier by applied parameters criterion and n_estimators. We can also see that the ideal parameters for our model was 'criterion': 'gini', and 'n_estimators': 50. One more thing, we have to check the confusion matrix.

[26]:
```
testing_rfc_pred = rfc_model.predict(testing_features)

#visulalizing the confusion matrix
rfc_confusion_matrix = confusion_matrix(testing_target, testing_rfc_pred)
rfc_confusion_matrix
```

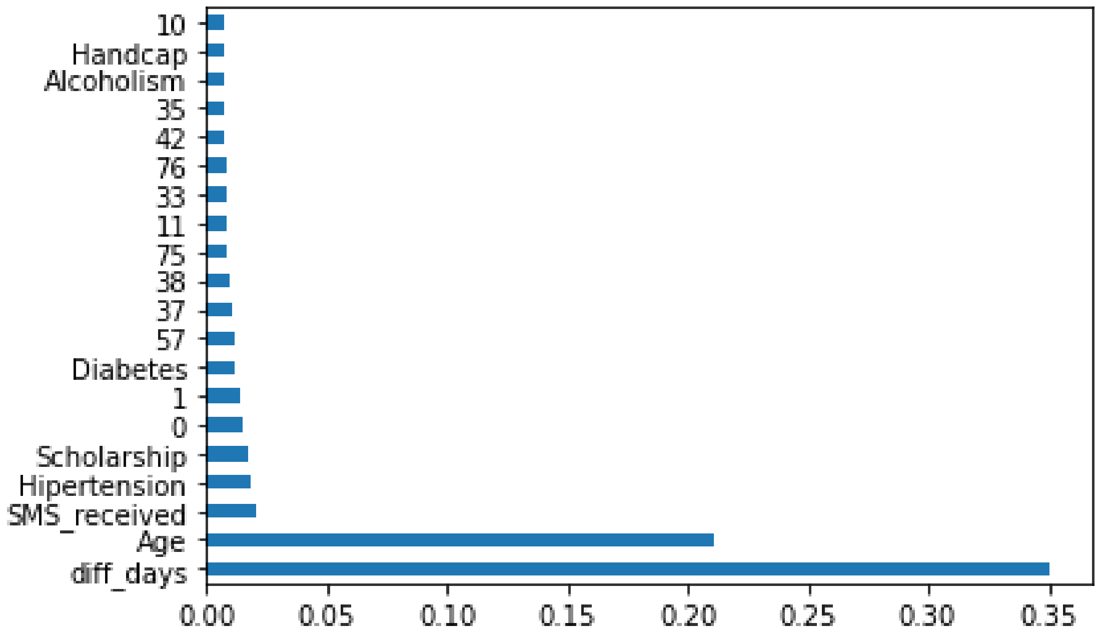
[26]:
```
array([[840,  60],
       [183,  23]], dtype=int64)
```

Confusion matrix shows:

Zero index is true negative values 840 We got correct. 23 is true positive values. it show how many true postive we got correct 60 is actully negative but we predict positive. 183 actully positive but we predict true negative. We can see 183 false negative much higher than true negative. So, we are not really able to predict person no show by 100 percent.

[27]:
```
# checking Random_Forest Features Importance by subplot
important_ft = pd.Series(rfc_model.best_estimator_.feature_importances_,
 ↪index=training_features.columns)
important_ft.nlargest(20).plot(kind='barh')
```

[27]: <AxesSubplot:>



Above features importance plot shows that the diff_days feature which we created turned out to be very important in predicting No-show appointments. The second most important feature under the random forest classifier is Age.

2.6.2 5.2 Perceptron

[28]:
```
parameters_grid = {
    "penalty":['l2','l1','elasticnet'],
}

p_model =  GridSearchCV(Perceptron(),
                         parameters_grid, scoring="accuracy", cv=5, n_jobs=-1)

p_model.fit(training_features, training_target)
```

[28]:
```
GridSearchCV(cv=5, estimator=Perceptron(), n_jobs=-1,
             param_grid={'penalty': ['l2', 'l1', 'elasticnet']},
             scoring='accuracy')
```

[29]:
```
p_model.best_score_
```

[29]:
```
0.7334841628959277
```

[30]:
```
p_model.best_params_
```

[30]:
```
{'penalty': 'l2'}
```

Result:

From Perceptron, we obtained not bad score which had penalty l1 best parameters. Here we still need to check confusion matrix.

[31]:
```
testing_percep_pred = p_model.predict(testing_features)

#visulalizing the confusion matrix
percep_confusion_matrix = confusion_matrix(testing_target, testing_percep_pred)
percep_confusion_matrix
```

[31]:
```
array([[816,  84],
       [188,  18]], dtype=int64)
```

Above clearly shows: TN values 816 We got correct. TP values is 18. How many true postive got it correctly. Actully negative is 84 but we predict positive. 188 actully positive but we predict true negative. It can be observed that 188 false negative value much higher than true negative. So,100 percent is not possible from this prediction.

2.6.3 5.3 Gradient Boosting Classifier

[32]:
```
parameters_grid = {
    "learning_rate":[0.05,0.1],
    "n_estimators": range(50,200,300),
}

gbc_model =  GridSearchCV(GradientBoostingClassifier(),
                          parameters_grid, scoring="accuracy", cv=5, n_jobs=-1)

gbc_model.fit(training_features, training_target)
```

[32]:
```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid={'learning_rate': [0.05, 0.1],
                         'n_estimators': range(50, 200, 300)},
             scoring='accuracy')
```

[33]:
```
gbc_model.best_score_
```

[33]:
```
0.7954751131221719
```

[34]:
```
gbc_model.best_params_
```

[34]:
```
{'learning_rate': 0.05, 'n_estimators': 50}
```

Result:

This model given best score as compared to others. The best parameters of gradient boosting classifier are 0.1 fpr learning_rate and n_estimators are 50. We will visulize confusion matrix as we did previously.

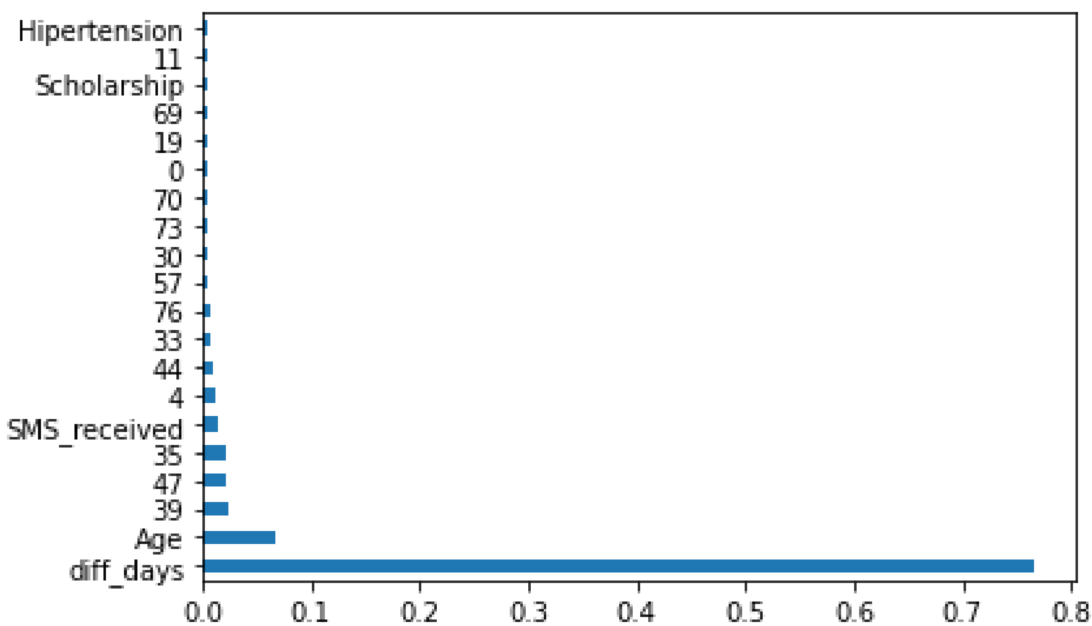
[35]:
```
testing_gbc_pred = gbc_model.predict(testing_features)

#visulalizing the confusion matrix
gbc_confusion_matrix = confusion_matrix(testing_target, testing_gbc_pred)
gbc_confusion_matrix
```

[35]:
```
array([[898,   2],
       [205,   1]], dtype=int64)
```

Mentioned above without considering these numbers.

[36]:
```
# Gradient Boosting Feature Importance
feat_importances = pd.Series(gbc_model.best_estimator_.feature_importances_,␣
 ↪index=training_features.columns)
feat_importances.nlargest(20).plot(kind='barh')
```

[36]:
```
<AxesSubplot:>
```



Once more we can see that the diff_days feature is very important in the Gradient Boosting Classifier.

3 6. Keras (Deep Learning)

Keras is a deep learning process and Keras work with tensorflow. So, I need to upload Library keras. My PC do not have install tensorflow packages. Firstly, i need to install it in my computer(From Anaconda navigator or pip install).

What we are going to do?

a) We will import Library to work with keras

b) Data prepration by drop some columns

c) Feature Preprocessing step

d) Building Keras Model

3.0.1 6.1 Importing Library

Before Keras Model building, some Libraries must to be uploaded.

[37]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

3.0.2 6.2 Preparing Data

In this keras modelling, we will be carrying out similar data preprocessing steps as we did in the earlier models but this time around, using libraries that are under keras. In this section, useless features will be dropped as we did in previous cells. After that, the data will be split into two sets, one for training and one for validation. Moreover, we will creat an object name for each data frame

[38]:
```
# first convert datetime data to datetime format
keras_use_data['ScheduledDay'] = keras_use_data['ScheduledDay'].
 ↪astype('datetime64[ns]')
keras_use_data['AppointmentDay'] = keras_use_data['AppointmentDay'].
 ↪astype('datetime64[ns]')

# difference in days
# create a new variabe which is the difference between scehduled day and␣
 ↪appointment day
keras_use_data ['diff_days'] = (keras_use_data ['ScheduledDay'] -␣
 ↪keras_use_data ['AppointmentDay']) / np.timedelta64(1, 'D')
keras_use_data ['diff_days']

# Drop useless columns
keras_use_data = keras_use_data.drop(['PatientId','AppointmentID',␣
 ↪'ScheduledDay','AppointmentDay'], axis = 1)
```

[39]:
```
# We convert the target variable to binary numbers
keras_use_data['No-show'] = keras_use_data['No-show'].apply(lambda x: 0 if x.
 ↪strip()=='No'
                                                      else 1)
```

[40]:
```
# Splitting Data into validation data and training data
validation_data = keras_use_data.sample(frac=0.2, random_state=100)
training_data = keras_use_data.drop(validation_data.index)
```

```
print(
    "Using %d samples for training and %d for validation"
    % (len(training_data), len(validation_data))
)
```

```
Using 4421 samples for training and 1105 for validation
```

In upper cell, we devided our data into Training(4421 samples) and validation set(1105 samples)

[41]:
```
# giving name to each data frame
def use_data_to_dataset(keras_use_data):
    keras_use_data = keras_use_data.copy()
    labels = keras_use_data.pop("No-show")
    ds = tf.data.Dataset.from_tensor_slices((dict(keras_use_data), labels))
    ds = ds.shuffle(buffer_size=len(keras_use_data))
    return ds


train_noshow = use_data_to_dataset(training_data)
vald_noshow = use_data_to_dataset(validation_data)
```

[42]:
```
# assign the batch to training and validation data
train_noshow = train_noshow.batch(30)
vald_noshow = vald_noshow.batch(30)
```

3.0.3 6.3 Featuring Preprocess Data

In data Preprocessing step, I need to upload Libraries are required from keras for one_hot_encoding.

[43]:
```
# importing Libararies
from tensorflow.keras.layers import IntegerLookup
from tensorflow.keras.layers import Normalization
from tensorflow.keras.layers import StringLookup


def encode_numerical_feature(feature, name, dataset):
    #  For our feature creating a Normalize layer
    normalizer = Normalization()

    # Plan a dataset that as yeild include
    feature_ds = dataset.map(lambda x, y: x[name])
    feature_ds = feature_ds.map(lambda x: tf.expand_dims(x, -1))

    # understanding our data with statistics approch
    normalizer.adapt(feature_ds)
```

```
    # Input feature Normalize
    encod_feature = normalizer(feature)
    return encod_feature


def encode_categorical_feature(feature, name, dataset, is_string):
    lookup_class = StringLookup if is_string else IntegerLookup
    # To turn string into number list, call lookup layer
    lookup = lookup_class(output_mode="binary")

    # dataset to be palned for only yeilds features
    feature_ds = dataset.map(lambda x, y: x[name])
    feature_ds = feature_ds.map(lambda x: tf.expand_dims(x, -1))

    # check the range of potential string possiblites and give each one a fixed␣
↪integer index
    lookup.adapt(feature_ds)

    # integer indices turn by input string
    encod_feature = lookup(feature)
    return encod_feature
```

3.0.4 6.4 Building Keras Model

In this we will convert our categorical features are encoded as integers listed below: Scholarship,Hipertension,Diabetes,Alcoholism,Handcap,SMS_received

After this step, we will encoded string features like Gender,Neighbourhood

Then numerical feature which is age.

[44]:
```
# Categorical features encoded as integers
Scholarship = keras.Input(shape=(1,), name="Scholarship", dtype="int64")
Hipertension = keras.Input(shape=(1,), name="Hipertension", dtype="int64")
Diabetes = keras.Input(shape=(1,), name="Diabetes", dtype="int64")
Alcoholism = keras.Input(shape=(1,), name="Alcoholism", dtype="int64")
Handcap = keras.Input(shape=(1,), name="Handcap", dtype="int64")
SMS_received = keras.Input(shape=(1,), name="SMS_received", dtype="int64")


# Encoded as string which are categorical_columns
Gender = keras.Input(shape=(1,), name="Gender", dtype="string")
Neighbourhood = keras.Input(shape=(1,), name="Neighbourhood", dtype="string")



# Numerical features
Age = keras.Input(shape=(1,), name="Age")
```

```
diff_days = keras.Input(shape=(1,), name="diff_days")

all_inputs = [
    Scholarship,
    Hipertension,
    Diabetes,
    Alcoholism,
    Handcap,
    SMS_received,
    Gender,
    Neighbourhood,
    Age,
    diff_days,

]

# categorical_columns_integer
Scholarship_encoded = encode_categorical_feature(Scholarship, "Scholarship",␣
↪train_noshow, False)
Hipertension_encoded = encode_categorical_feature(Hipertension, "Hipertension",␣
↪train_noshow, False)
Diabetes_encoded = encode_categorical_feature(Diabetes, "Diabetes",␣
↪train_noshow, False)
Alcoholism_encoded = encode_categorical_feature(Alcoholism, "Alcoholism",␣
↪train_noshow, False)
Handcap_encoded = encode_categorical_feature(Handcap, "Handcap", train_noshow,␣
↪False)
SMS_received_encoded = encode_categorical_feature(SMS_received, "SMS_received",␣
↪train_noshow, False)

# Encoded categorical_columns_string
Gender_encoded = encode_categorical_feature(Gender, "Gender", train_noshow,␣
↪True)
Neighbourhood_encoded = encode_categorical_feature(Neighbourhood,␣
↪"Neighbourhood", train_noshow, True)


# Lets encoded our Numerical_features
Age_encoded = encode_numerical_feature(Age, "Age", train_noshow)
diff_days_encoded = encode_numerical_feature(diff_days, "diff_days",␣
↪train_noshow)

all_features = layers.concatenate(
    [
        Scholarship_encoded,
        Hipertension_encoded,
```

```
        Diabetes_encoded,
        Alcoholism_encoded,
        Handcap_encoded,
        SMS_received_encoded,
        Gender_encoded,
        Neighbourhood_encoded,
        Age_encoded,
        diff_days_encoded,

    ]
)
# here we will create hidden layers with 64 activation each function relu
x = layers.Dense(64, activation="relu")(all_features)
x = layers.Dropout(0.5)(x)
output = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(all_inputs, output)
model.compile("adam", "binary_crossentropy", metrics=["accuracy"])
```

[45]:
```
model.fit(train_noshow, epochs=10, validation_data=vald_noshow)
```

```
Epoch 1/10
148/148 [==============================] - 6s 18ms/step - loss: 0.5168 -
accuracy: 0.7829 - val_loss: 0.4910 - val_accuracy: 0.7873
Epoch 2/10
148/148 [==============================] - 2s 10ms/step - loss: 0.4849 -
accuracy: 0.7998 - val_loss: 0.4902 - val_accuracy: 0.7873
Epoch 3/10
148/148 [==============================] - 2s 11ms/step - loss: 0.4736 -
accuracy: 0.8021 - val_loss: 0.4894 - val_accuracy: 0.7873
Epoch 4/10
148/148 [==============================] - 2s 10ms/step - loss: 0.4749 -
accuracy: 0.8019 - val_loss: 0.4885 - val_accuracy: 0.7873
Epoch 5/10
148/148 [==============================] - 2s 10ms/step - loss: 0.4686 -
accuracy: 0.8039 - val_loss: 0.4851 - val_accuracy: 0.7873
Epoch 6/10
148/148 [==============================] - 2s 10ms/step - loss: 0.4657 -
accuracy: 0.8028 - val_loss: 0.4865 - val_accuracy: 0.7873
Epoch 7/10
148/148 [==============================] - 2s 10ms/step - loss: 0.4625 -
accuracy: 0.8041 - val_loss: 0.4838 - val_accuracy: 0.7873
Epoch 8/10
148/148 [==============================] - 2s 11ms/step - loss: 0.4589 -
accuracy: 0.8034 - val_loss: 0.4846 - val_accuracy: 0.7864
Epoch 9/10
148/148 [==============================] - 2s 10ms/step - loss: 0.4562 -
accuracy: 0.8034 - val_loss: 0.4852 - val_accuracy: 0.7864
```

```
Epoch 10/10
148/148 [==============================] - 2s 10ms/step - loss: 0.4545 -
accuracy: 0.8037 - val_loss: 0.4870 - val_accuracy: 0.7864
```

[45]:
```
<keras.callbacks.History at 0x921283bac0>
```

Conclusion:

From Keras, I called only 10 Epoch due to my Pc capacity. Epoch 1 given the 79 percent accuracy while others showed almost 80% accuracy. Keras given us best result from above all.

3.1 7. Final Discussion and Conclusion

To summarize all above, we concluded that our dataset did not have missing values. But we noticed some categorical features and we worked on them. During the Model building we got some result listed below:

1) Random Forest Classifier (GridSearchCV)
2) Perceptron
3) Gradient Boosting Classifier
4) Keras (deep learning model)

It was observed from the RandomForest Classifier and Gradient Boosting Classifier that the feature we generated from the difference between ScheduledDay and AppointmentDay (diff_days) was the most important feature in predicting No-show appointments. and From all the models evaluated, we were able to achieve the highest level of accuracy through keras.

3.2 8. Refrences

Mohamed, G.A. (2022) noshowappointments-kagglev2-may-2016.csv. Available at: https://www.kaggle.com/datasets/muhammetgamal5/noshowappointmentskagglev2may2016csv (Accessed: 22 June 2022).

François, C. (2020) Structured data classification from scratch. Available at: https://keras.io/examples/structured_data/structured_data_classification_from_scratch/ (Accessed: 23 June 2022).

scikit-learn (2022) scikit-learn. Available at: https://scikit-learn.org/stable/ (Accessed: 01 June 2022).