**Predict a Doctor’s Consultation Fee**

Doctors all over the world are given the status next to God. It happens so mostly because they are life savers who work tirelessly for mankind. Moreover, being a doctor is considered one of the most sought after professions.

Doctors have a very noble profession. They are equipped with comprehensive Knowledge and devices that enable them to diagnose and treat their patient with correct procedures. The Medical Scenario in India is renowned all over the world. The Doctors originating from India are reaching new heights globally abroad. Indian Doctors are very much in demand all over the world. India is a reservoir of doctors. Our doctors work everywhere ranging from small village to big metro cities. A Doctor fee varies from country to country, cities to cities and areas to areas. Although the medical field is evolving, there are still immoral practices in the field which makes it tough for patients to get the right treatment. Corruption has not spared this field as well. India suffers from a [high illiteracy rate](https://www.toppr.com/guides/geography/population/population-of-india/) which results in people fooling the citizens for money. There are many wrongs and unethical medical practices prevalent in India which brings a bad name to the country.

Moreover, the greed for money has resulted in various losses of lives of patients. The hospitals diagnose the patients wrongly and give them the wrong treatment. This results in even more worse results. The public is losing its faith in the medical field and its doctors.

As a result, this impacts the reputation of the medical field. Doctors must be more responsible and vigilant with the lives of their patients. The government must provide the public with good medical facilities which can bridge this gap. In addition, we must also come together to help doctors do their job better.

So, this dataset is basically about predicting the doctor consultation fees varying from cities to cities. So, lets’ jump right into the prediction ,This dataset is a smaller data comprising of 5961 rows and 7 unique features.

1. Qualification of the Doctor.

2. Experience of the doctor in years.

3. Ratings that were given by the patients.

4. Place (area, city of doctor’s Location).

5. Profile of the Doctor.

6. Miscl. Info that contain other information about the doctor.

7. Fees charged by the doctor.

The dependant variable is fees charged by the doctor.

* Firstly import all the dependencies and the dataset.

Import numpy as np

Import pandas as pd

Import matplotlib.pyplot as plt

%matplotlib inline

import warnings

import seaborn as sns

warnings.filterwarnings('ignore')

Import unicodedata

Import re

From collections import Counter

From sklearn import metrics

From matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 12, 10

* Next, let’s view the data to see the information given in dataset.

dcf\_train.head()

| **Sl.No.** | **Qualification** | **Experience** | **Rating** | **Place** | **Profile** | **Miscellaneous\_Info** | **Fees** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | BHMS, MD - Homeopathy | 24 years experience | 100% | Kakkanad, Ernakulam | Homeopath | 100% 16 Feedback Kakkanad, Ernakulam | 100 |
| **1** | BAMS, MD - Ayurveda Medicine | 12 years experience | 98% | Whitefield, Bangalore | Ayurveda | 98% 76 Feedback Whitefield, Bangalore | 350 |
| **2** | MBBS, MS - Otorhinolaryngology | 9 years experience | NaN | Mathikere - BEL, Bangalore | ENT Specialist | NaN | 300 |
| **3** | BSc - Zoology, BAMS | 12 years experience | NaN | Bannerghatta Road, Bangalore | Ayurveda | Bannerghatta Road, Bangalore ?250 Available on... | 250 |
| **4** | BAMS | 20 years experience | 100% | Keelkattalai, Chennai | Ayurveda | 100% 4 Feedback Keelkattalai, Chennai | 250 |

* Then, let’s move into looking at the number of missing values in this training dataset.

round(dcf\_train.isnull().sum()/len(dcf\_train) \* 100,2)

Qualification 0.00

Experience 0.00

Rating 55.39

Place 0.42

Profile 0.00

Miscellaneous\_Info 43.95

Fees 0.00

dtype: float64

The image above showed the percentage of missing values for each column.

* The “Experience” column here seems simple enough as it just requires extracting the integer values from the string.

dcf\_train['Experience'] = dcf\_train['Experience'].str.split().str[0]

dcf\_train['Experience'] = dcf\_train['Experience'].astype(int)

* The first line of code split the string into a list while the second extract the first element of the list and convert it into an integer.Next, the “Place” column can be easily processed by separating the City from the area.

dcf\_train.Place.fillna('Unknown,Unknown',inplace=True)

dcf\_train['locality'] = dcf\_train['Place'].str.split(",").str[0]

dcf\_train['city'] = dcf\_train['Place'].str.split(",").str[1]

Before extraction, I replaced all missing values in this column with the string ‘Unknown, Unknown’ to represent them. Side note, sometimes it is a good idea to give missing values a separate class instead of relying on missing values imputation technique like mean/median/mode. For example in this dataset, some regions in Indian might not have listed down their location during data collection but they could have come from the same region. Next, splitting the string at ‘,’ and creating a new column ‘City’ using the last element of the list.

* Now, Moving on to the ‘Ratings’ column, remember that this column has more than 50% of missing values. We have to deal with the missing values before any other processing.

dcf\_train['Rating'].fillna('-99%',inplace=True)

dcf\_train['Rating'] = dcf\_train['Rating'].str.slice(stop=-1).astype(int)

bins = [-99,0,10,20,30,40,50,60,70,80,90,100]

labels = [i for i in range(11)]

dcf\_train['Rating'] = pd.cut(dcf\_train['Rating'], bins=bins, labels=labels, include\_lowest=True)

* Missing values were replaced with -99% to differentiate them. Then, assuming a rating of 91% has no significant difference as a rating of 99%, I grouped them into bins of size 10. Missing values will fall under class 0 while, 0-9% will be class 1, and 10–19% will be class 2 dcf\_train['Rating'].value\_counts().sort\_index() displayed the distribution.

0 3302

1 1

2 0

3 0

4 4

5 3

6 19

7 32

8 98

9 280

10 2222

Name: Rating, dtype: int64

* For the ‘Qualification’ columns, it consists of various qualification of the doctor without any standardized reporting method. I start off by doing the normal split and try to get an idea of the frequency of the different terms appeared in this column.

dcf\_train["Qualification"]=dcf\_train["Qualification"].str.split(",")

Qualification ={}

for x in dcf\_train["Qualification"].values:

for each in x:

each = each.strip()

if each in Qualification:

Qualification[each]+=1

else:

Qualification[each]=1

* I decided to go with the simplest approach and simply identify the top 10 qualification that occurs the most.

most\_qual = sorted(Qualification.items(),key=lambda x:x[1],reverse=True)[:10]

final\_qual = []

for qual in most\_qual:

final\_qual.append(qual[0])

print(final\_qual)

for qual in final\_qual:

dcf\_train[qual] = 0

for x,y in zip(dcf\_train['Qualification'].values, np.array([i for i in range(len(dcf\_train))])):

for c in x:

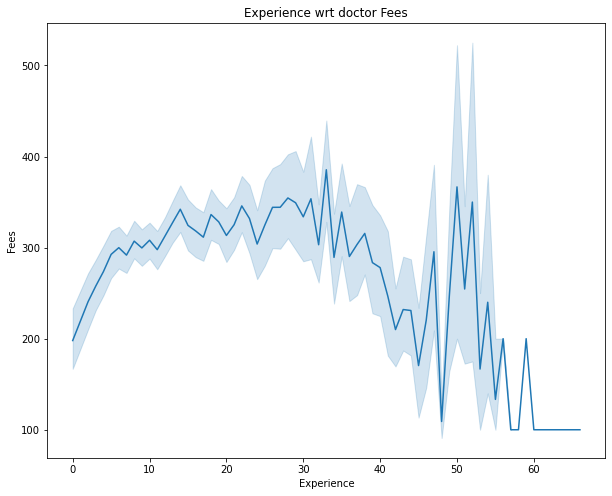
c = c.strip()

if c in final\_qual:

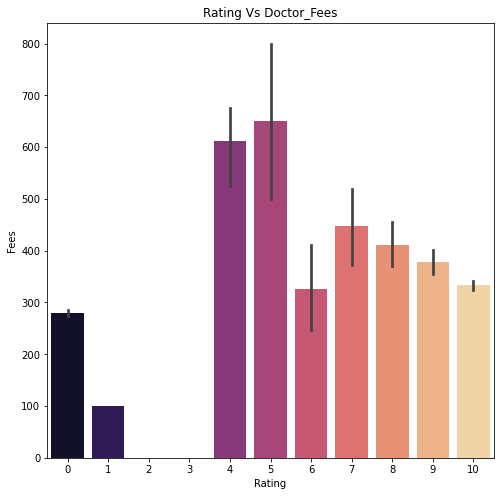
dcf\_train[c][y] = 1

dcf\_train.drop(['Qualification','Qualification\_count'],axis=1,inplace=True)

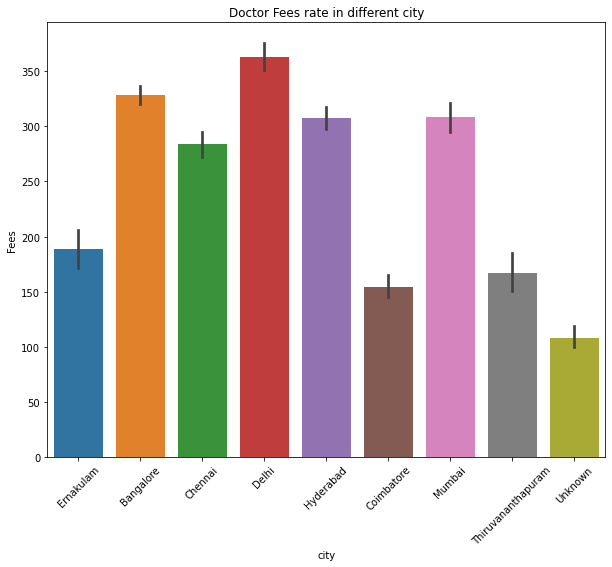
* The final result is dummies variables for the 10 highest frequency qualifications in the dataset.
* Now for the ‘Profile’ column. If you can remember, we do not have any missing value in this column. A quick value\_counts() check yielded this.
* Once we are done with data pre-processing, the next step should naturally be data visualization.
* Most people would have assumed there is some association between the experience of the doctor and the fees they charged. Indeed there is, but it might not what we expect it to be. Average fees increased with experience but peak at approximately 25 years of experience, then, average fees decreased with further increasing number of experience.



* Ratings is one of the most interesting variable to look at. If you remember, we group rating into bins of size 10, inclusive of the smallest value. As you can see, a high rating does not correlate to a higher fee charged (in fact a lower fee might be the reason for high rating!), and the highest average fees charged were actually rated 30–60%. The colour scheme depicts the median experience level in each bin, with dark green representing a higher median experience. Median experience in bins 4 and 5 was 27 years and 31 years respectively while bin 10 only has a median experience of 14 years, justifying the ability of the doctors to charge a higher fee in those bins.



* Fees charged by doctor in different cities. Let’s see how the fees charged by doctor varies from one city to city .Yes the fees will be varied according to the city cost, infrastructure etc.



-In city of Ernakulum the fees rate charged by doctor is around Rs.200.

-In city of Bangalore the fees rate charged by doctor is around Rs.300-350.

-In city of Chennai the fees rate charged by doctor is around Rs.250-300.

-In city of Delhi, the highest fee charged by doctor is around Rs.350.

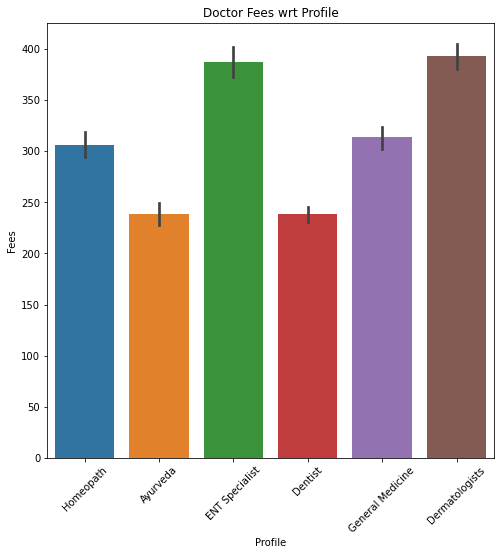
-In city of Hyderabad, the highest fee charged by doctor is around Rs.300.

-In city of Coimbatore, the doctor charges less fee which is around Rs.150.

-In city of Mumbai, the fees rate charged by doctor is around Rs.300.

-In city of Thiruvanthapuram, the fees rate charged by doctor is around Rs.150-200.

* Distribution of Different doctor profile and fees charged by each profile is shown as bellow :



-from this visualisation we can observe that Dermatologist charges the highest fee which is around Rs.400.

* Before implementing the algorithm, we have to encode the variable.

**from** **sklearn.model\_selection** **import** train\_test\_split x\_train, x\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=0)

* SVR implementing...

**from** **sklearn.neighbors** **import** KNeighborsRegressor

**from** **sklearn.svm** **import** SVR

**from** **sklearn.tree** **import** DecisionTreeRegressor

**from** **sklearn.ensemble** **import** RandomForestRegressor

**from** **sklearn.metrics** **import** make\_scorer

* **def** score(y\_pred,y): y\_pred = np.log(y\_pred) y = np.log(y) **return** 1 - ((np.sum((y\_pred-y)\*\*2))/len(y))\*\*1/2 scorer = make\_scorer(score,greater\_is\_better=**True**, needs\_proba=**False**)
* Our prediction on the testing set gave us a score of 0.7759692513240842.If you take a look at the leader board, the winner with the best score is 0.7627072415729066 . Of course, our testing set is not the real testing set used for the leader board and is not comparable. But it gave us a metric to use for further optimization.
* Running reg.best\_params\_ gave the combination of hyper parameters that provide the best score. The best hyper parameters in this case are C=10, and kernel=”rbf”. Note\* define your own scorer to be used in GridSearchCV so it optimized using that scoring matrix.
* Finally.  
  print("best\_params:**\n**",rf\_random.best\_params\_) print("**\n**best\_score:",rf\_random.best\_score\_)

The score obtained here is 0.7950595593000074 .