Predict Doctor's Consultation Fees

1. Problem Definition

We have all been in situation where we go to a doctor in emergency and find that the consultation fees are too high. As a data scientist we all should do better. What if you have data that records important details about a doctor and you get to build a model to predict the doctor’s consulting fee.?

2. Data Analysis

	Experience	Rating	Profile	Miscellaneous_Info	Fees	locality	city	MBBS	BDS	BAMS	BHMS	MD - Dermatology	MS - ENT	Venereology & Leprosy	MD - General Medicine
0	24	10	Homeopath	100% 16 Feedback Kakkanad, Ernakulam	100	Kakkanad	Ernakulam	0	0	0	1	0	0	0	0
1	12	10	Ayurveda	98% 76 Feedback Whitefield, Bangalore	350	Whitefield	Bangalore	0	0	1	0	0	0	0	0
2	9	0	ENT Specialist	NaN	300	Mathikere - BEL	Bangalore	1	0	0	0	0	0	0	0
3	12	0	Ayurveda	Bannerghatta Road, Bangalore ₹250 Available on...	250	Bannerghatta Road	Bangalore	0	0	1	0	0	0	0	0
4	20	10	Ayurveda	100% 4 Feedback Keelkattalai, Chennai	250	Keelkattalai	Chennai	0	0	1	0	0	0	0	0

Qualification: Qualification and degrees held by the doctor

Experience: Experience of the doctor in number of years

Rating: Rating given by patients

Profile: Type of the doctor

Miscellaneous_Info: Extra information about the doctor

Fees: Fees charged by the doctor (Target Variable)

Place: Area and the city where the doctor is located

Clearly, some data cleaning needs to be done before any modelling process. Let's start by looking at the number of missing values in this training dataset.

```
# Extract years of experience
df["Experience"] = df["Experience"].str.split()
df["Experience"] = df["Experience"].str[0].astype("int")
```

Next, the “Place” column can be easily processed by separating the City from the area.

For the ‘Qualification’ columns, it consists of various qualification of the doctor without any standardized reporting method.

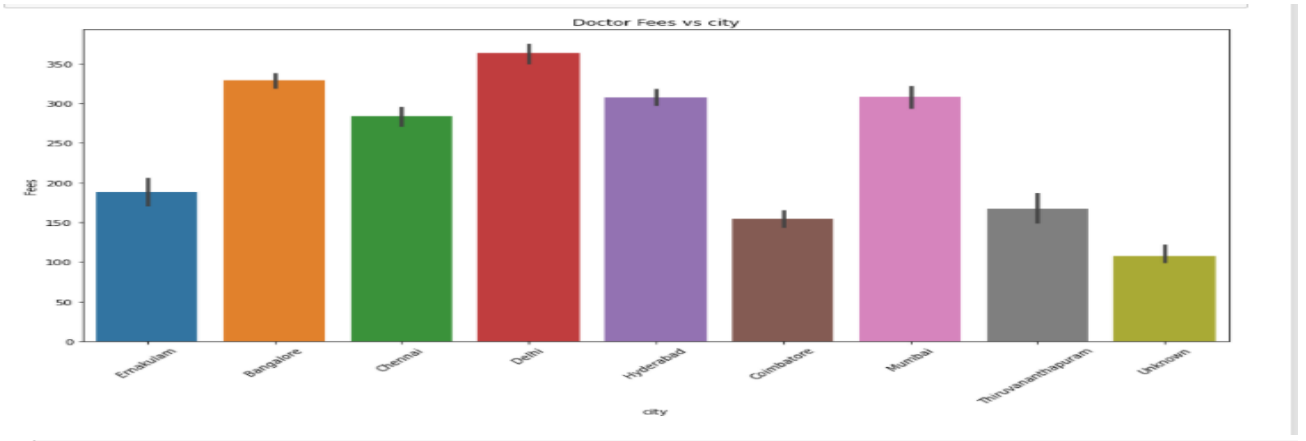
```
# Extract cities
df["Place"].fillna("Unknown,Unknown",inplace=True)
df["Place"] = df["Place"].str.split(",")
df["City"] = df["Place"].str[-1]
df["Place"] = df["Place"].str[0]
```

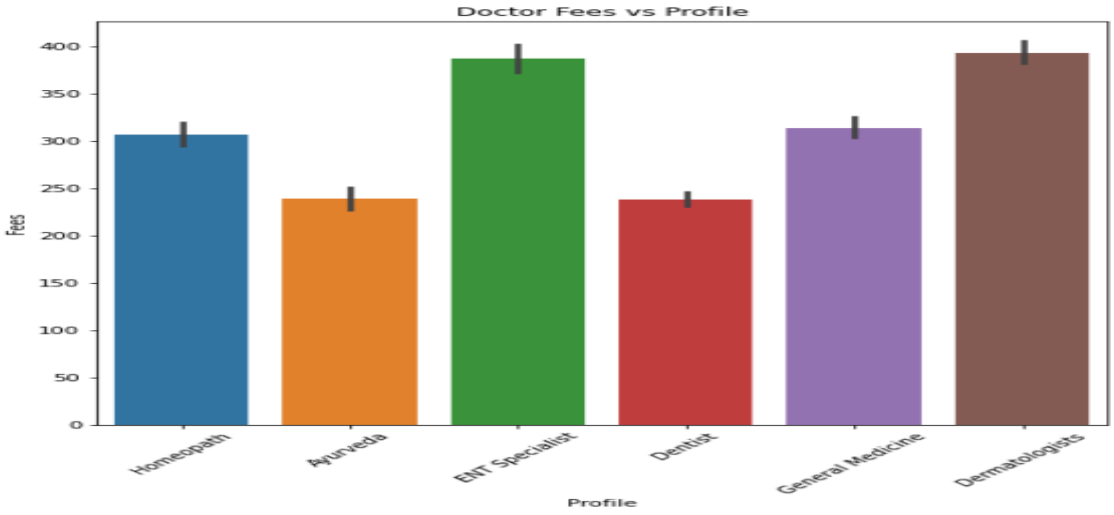
```
# Extract relevant qualification
df["Qualification"]=df["Qualification"].str.split(",")
Qualification ={}
for x in df["Qualification"].values:
    for each in x:
        each = each.strip()
        if each in Qualification:
            Qualification[each]+=1
        else:
            Qualification[each]=1
```

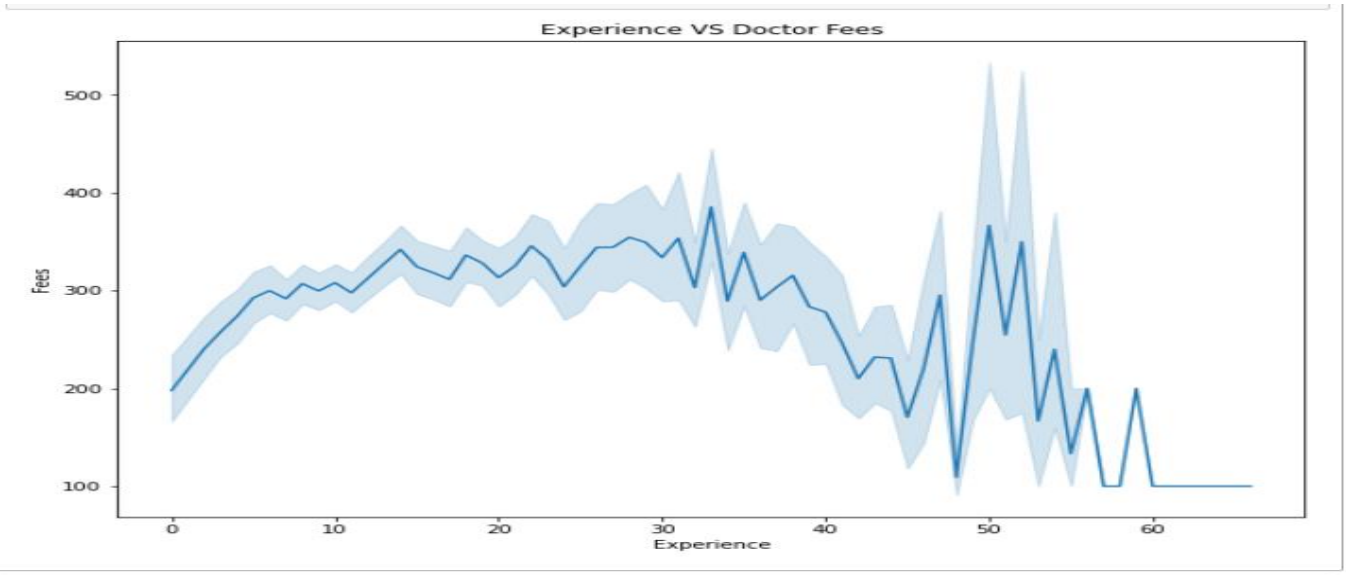
3.EDA Concluding Remark

We will be checking the relation upon the columns mentioned as *Qualification, Experience, Rating , Place,Profile, Miscellaneous_Inf, Fees ,Doctor fees,Ratings,city*.

Below graphs are being shared, where we can get a clear perspective of the versus column.


Doctor Fees vs city
Fees
city
Ernakulam
Bangalore
Chennai
Delhi
Hyderabad
Coimbatore
Mumbai
Thiruvananthapuram
Unknown


Doctor Fees vs Profile
Fees
Profile
Homeopath
Ayurveda
ENT Specialist
Dentist
General Medicine
Dermatologists


Experience VS Doctor Fees
Fees
Experience

4.Test data and train data

We will train &test with the fees column as this is the main model. Finally, we can model our data can do some cool

```
X = df.drop("Fees",axis=1)
y = df["Fees"]

# Encoding
enc = OrdinalEncoder()
X = enc.fit_transform(X)

X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.2)

# feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

machine learning. I had decided to use a support vector machine for this task as it can be used for both linear and non-linear problems. The small amount of data also do not warrant the use of a neural network.

Before implementing the algorithms, we have to encode the categorical variables and scale its features.

5.Model building

We use 4 models to check the efficiency

- KNeighborsRegressor
- SVR
- DecisionTreeRegressor
- RandomForestRegressor

On the basis we got an efficiency of .77,.79,.67,.76

```
In [51]: # Training and testing
         for reg in (knn_reg, svm_reg, dt_reg, rf_reg):
             reg.fit(x_train, y_train)

             y_pred = reg.predict(x_test)

             print(reg, score(y_pred,y_test))
```

```
KNeighborsRegressor() 0.7759692513240842
SVR() 0.7942077126478044
DecisionTreeRegressor() 0.6702978970649067
RandomForestRegressor() 0.7645396171205396
```

6.HyperParameter Tuning

Doing a GridSearchCV is a great way to do hyperparameters tunning, but take note of the computation power needed especially for algorithms like SVM that do not scale well.

We import Gridsearchcv and Randomizedsearchcv for tuning

We can get the best score and best params from the above.

Got a score of .80 and certainly took it forward. For train purpose we are Fitting 5 folds for each of 100 candidates, totalling 500 fits.

Note: - Fees depends upon lot of factors mainly **Cities**(Tier 1,2,3),**Experience**(years)and **Rating**(1-10)

```
In [55]: print("best_score:",svm_random.best_score_)
         print("best_params:\n",svm_random.best_params_)
```

```
best_score: 0.8056521524720275
```