Predict A Doctor's Consultation Fee

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.?

Size of training set: 5961 records

Size of test set: 1987 records

Data Analysis

FEATURES:

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

Miscellaeous_Info: Extra information about the doctor

Fees: Fees charged by the doctor

Place: Area and the city where the doctor is located.

https://github.com/dsrscientist/Data-Science-ML-Capstone-Projects

Fees Column will be our target variable in this database.

We have two datasets , one is testing Data and other is the Train Data.

The important libraries to start our project will be pandas and numpy.

Numpy and Pandas are popular and widely used libraries in python

Numpy:

NumPy stands for ‘Numeric Python’ or ‘Numerical Python’. It is an open source module of Python which offers fast mathematical computation on arrays and matrices.

NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation

Pandas:

Pandas is one of the most widely used python libraries in data science. It provides high-performance, easy to use structures and data analysis tools. Pandas 2d table object called Dataframe, as well as Series. It is having structure with column names and row labels. So, pandas is capable of computing columns and plotting graphs.

Reading and Cleaning Data

We can easily import an Excel file into Python using Pandas you’ll need to use *read_excel*.

```
df_doc_train=pd.read_excel("Final_Train.xlsx")

df_doc_train["source"]="train"

df_doc_test=pd.read_excel("Final_Test.xlsx")

df_doc_test["source"]="test"
```

Use concat command to merge this Testing Data and Train Data fr further analysis.

```
df_doc=pd.concat([df_doc_train,df_doc_test])
```

	Qualification	Experience	Rating	Place	Profile	Miscellaneous_Info	Fees	source
0	BHMS, MD - Homeopathy	24 years experience	100%	Kakkanad, Ernakulam	Homeopath	100% 16 Feedback Kakkanad, Ernakulam	100.0	train
1	BAMS, MD - Ayurveda Medicine	12 years experience	98%	Whitefield, Bangalore	Ayurveda	98% 76 Feedback Whitefield, Bangalore	350.0	train
2	MBBS, MS - Otorhinolaryngology	9 years experience	NaN	Mathikere - BEL, Bangalore	ENT Specialist	NaN	300.0	train
3	BSc - Zoology, BAMS	12 years experience	NaN	Bannerghatta Road, Bangalore	Ayurveda	Bannerghatta Road, Bangalore ₹250 Available on...	250.0	train
4	BAMS	20 years experience	100%	Keelkattalai, Chennai	Ayurveda	100% 4 Feedback Keelkattalai, Chennai	250.0	train
...	...	...	...	...	...	...	...	...
1982	BAMS, Doctor of Medicine	10 years experience	NaN	Basavanagudi, Bangalore	Ayurveda	NaN	NaN	test
1983	MD - Dermatology , Venereology & Leprosy, MBBS	4 years experience	NaN	Choolai, Chennai	Dermatologists	NaN	NaN	test
1984	MBBS, MD / MS - Obstetrics & Gynaecology	9 years experience	NaN	Porur, Chennai	General Medicine	NaN	NaN	test
1985	BDS	12 years experience	98%	Arekere, Bangalore	Dentist	RCT - Root Canal Treatment Root Canal Treatmen...	NaN	test
1986	MBBS, MD - Dermatology , Venereology & Leprosy	8 years experience	NaN	Pallikaranai, Chennai	Dermatologists	1 Feedback Pallikaranai, Chennai ₹500	NaN	test

7948 rows × 8 columns

Here we found 7948 Rows and 8 columns.

Data Cleaning is necessary as Experience column has int and string values both, also in Ratings % int and string values are present , we need to refine these columns.

Lets check the null values

df_doc.isnull().sum()

```
In [80]:  #lets chcek the null values
          df_doc.isnull().sum()

Out[80]:  Qualification          0
          Experience             0
          Rating              4392
          Place                 31
          Profile                0
          Miscellaneous_Info  3454
          Fees                1987
          source                 0
          dtype: int64
```

```
In [81]:  df_doc.info()

          <class 'pandas.core.frame.DataFrame'>
          Int64Index: 7948 entries, 0 to 1986
          Data columns (total 8 columns):
           #   Column              Non-Null Count  Dtype
          ---  ------              --------------  -----
           0   Qualification       7948 non-null   object
           1   Experience          7948 non-null   object
           2   Rating              3556 non-null   object
           3   Place               7917 non-null   object
           4   Profile             7948 non-null   object
           5   Miscellaneous_Info  4494 non-null   object
           6   Fees                5961 non-null   float64
           7   source              7948 non-null   object
          dtypes: float64(1), object(7)
          memory usage: 558.8+ KB
```

\

Now we can infer to the observations as below.

1)Data types of Fees is Float, rest is object.

2) Ratings columns, Miscellaneous and Fees columns has some Null values,

So we need to treat this Dataset.

3) We also have found that data in Qualification , Experience, Miscellaneous Fees columns have so many scattered Data.

We can bifurcate the strings and numbers from the columns that we have taken in observations.

For example take Experience column where as an example 24 years experience

Is written We can split the strings and number as “24” and “years experience”. By using

```
df_doc["Experience"]=df_doc["Experience"].str.strip(" years experience")
```

and covert the column to integer by

```
df_doc["Experience"]=df_doc["Experience"].astype(int)
```

This same procedure of Data refining will be done on Ratings column also.

```
df_doc["Rating"]=df_doc["Rating"].str.strip("%")
```

```
df_doc["Rating"]=df_doc["Rating"].astype(float)
```

Now we replace the null values in Rating with 0

```
df_doc["Rating"].fillna(0,inplace=True)
```

Similar procedure on Place table as given below:

```
#And then drop Place column

df_doc["Address"]=df_doc["Place"].str.split(",").str[0]

df_doc["City"]=df_doc["Place"].str.split(",").str[1]

df_doc.drop('Place',axis=1,inplace=True)
```

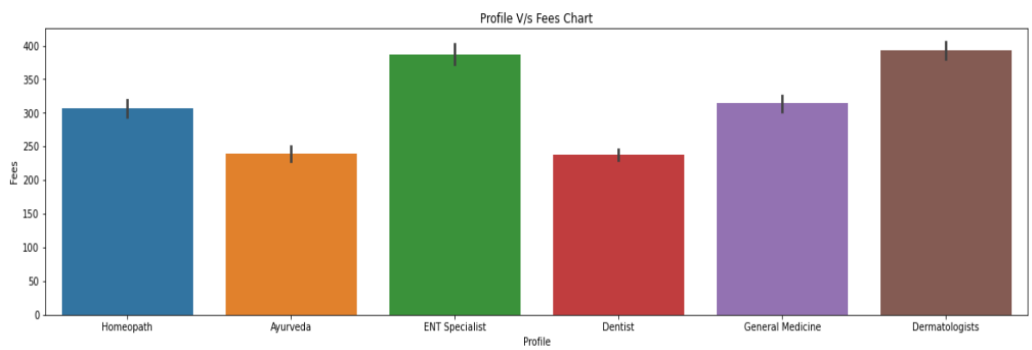
Here we can split this as Address and City name.

EDA Concluding Remarks

As we have taken Fees as our Target Variables. So we can perform Bivariant analysis on various Tables with our target variable(‘column’).

In [99]:

```
#Now we will perform EDA Process
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(20,5))
sns.barplot(x="Profile",y="Fees",data=df_doc)
plt.title("Profile V/s Fees Chart")
plt.show()
```

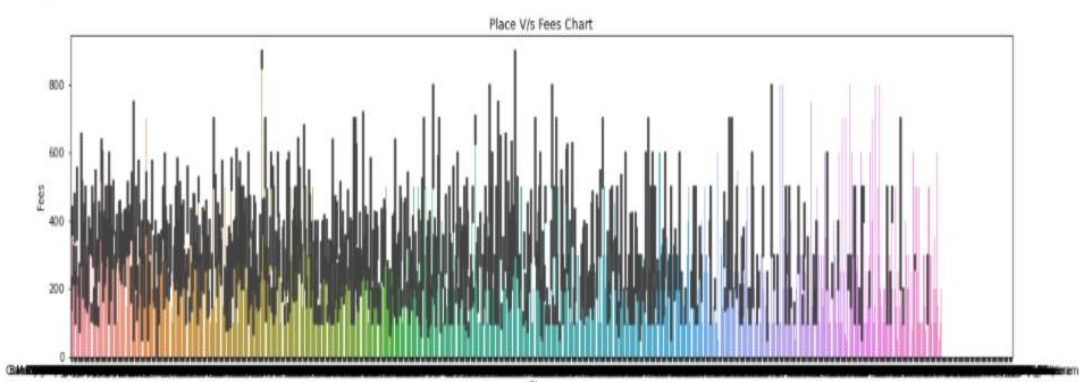


Here we can see that Fees of Dematologist & ENT Soecialist is high amoung the category of specializations.

In the same way we can perform the analysis on the Place Column and Fees column. Our Observations from the table are given below as:

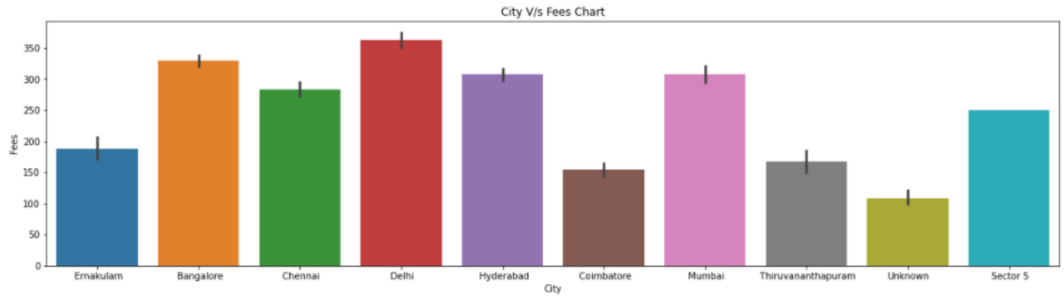
In [101]:

```
plt.figure(figsize=(20,5))
sns.barplot(x="Place",y="Fees",data=df_doc)
plt.title("Place V/s Fees Chart")
plt.show()
```



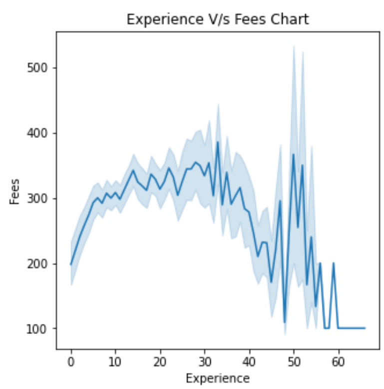
In the same way we can perform the analysis on the Place City and Fees column. Our Observations from the table are given below as:

In [108]:
```
plt.figure(figsize=(20,5))
sns.barplot(x="City",y="Fees",data=df_doc)
plt.title("City V/s Fees Chart")
plt.show()
```



Cities like Delhi , Banglore, hydrabad Mumbai and Chennai has shown highest earning of the Fees by Doctor’s.

In [110]:
```
plt.figure(figsize=(5,5))
sns.lineplot(x="Experience",y="Fees",data=df_doc)
plt.title("Experience V/s Fees Chart")
plt.show()
```



Now we can perfomr some more analysis on Experience vs Fees Charged, It will show us how experience Specialist can be charge the fees.

Observations are as the Experience of 50-60 yrs has charged more as compared to 10 – 30 yrs of exp. Specialist.

Pre-processing Pipeline

We can use numpy, pandas, matplotlib. pyplot,, seaborn, sklearn libraries in analysis.

As we have clean our Data in above processes.

We can now deal with Some messy columns in the database Miscellaneous Column and Qualification table. As these are the columns which cant be ignored and also we have to extract most of information’s from the desired columns.For doing so we can check the length of Qualification Column and use for loop for the same.

```
max_qual_length=-1

for i in range(len(df_doc["Qualification"])):

    temp=len(df_doc["Qualification"].iloc[i].strip().upper().split(","))

    if temp>max_qual_length:

        max_qual_length=temp

q1=list()

q2=list()

q3=list()

q4=list()

q5=list()

q6=list()

q7=list()

q8=list()

q9=list()

q10=list()

q11=list()

q12=list()

q13=list()

q14=list()

q15=list()
```

```
q16=list()
q17=list()
for i in range(len(df_doc["Qualification"])):
    temp=df_doc["Qualification"].iloc[i].split(",")
    try :
        q1.append(temp[0].strip().upper())
    except :
        q1.append('NONE')
    try :
        q2.append(temp[1].strip().upper())
    except :
        q2.append('NONE')
    try :
        q3.append(temp[2].strip().upper())
    except :
        q3.append('NONE')
    try :
        q4.append(temp[3].strip().upper())
    except :
        q4.append('NONE')
    try :
        q5.append(temp[4].strip().upper())
    except :
```

```
        q5.append('NONE')
    try :
        q6.append(temp[5].strip().upper())
    except :
        q6.append('NONE')
    try :
        q7.append(temp[6].strip().upper())
    except :
        q7.append('NONE')
    try :
        q8.append(temp[7].strip().upper())
    except :
        q8.append('NONE')
    try :
        q9.append(temp[8].strip().upper())
    except :
        q9.append('NONE')
    try :
        q10.append(temp[9].strip().upper())
    except :
        q10.append('NONE')
    try :
        q11.append(temp[10].strip().upper())
```

```
    except :
        q11.append('NONE')
    try :
        q12.append(temp[11].strip().upper())
    except :
        q12.append('NONE')
    try :
        q13.append(temp[12].strip().upper())
    except :
        q13.append('NONE')
    try :
        q14.append(temp[13].strip().upper())
    except :
        q14.append('NONE')
    try :
        q15.append(temp[14].strip().upper())
    except :
        q15.append('NONE')
    try :
        q16.append(temp[15].strip().upper())
    except :
        q16.append('NONE')
    try :
```

```
        q17.append(temp[16].strip().upper())
    except :
        q17.append('NONE')
print("Max qualification length is :{}".format(max_qual_length))
```

```
In [122]: df_doc["q1"]=q1
          df_doc["q2"]=q2
          df_doc["q3"]=q3
          df_doc["q4"]=q4
          df_doc["q5"]=q5
          df_doc["q6"]=q6
          df_doc["q7"]=q7
          df_doc["q8"]=q8
          df_doc["q9"]=q9
          df_doc["q10"]=q10
          df_doc["q11"]=q11
          df_doc["q12"]=q12
          df_doc["q13"]=q13
          df_doc["q14"]=q14
          df_doc["q15"]=q15
          df_doc["q16"]=q16
          df_doc["q17"]=q17
```

```
In [123]: df_doc.loc[(df_doc["q8"]!="NONE")|(df_doc["q9"]!="NONE")|(df_doc["q10"]!="NONE")|(df_doc["q11"]!="NONE")|(df_doc["q12"]!="NON
          #We can drop q8 to q17. which is of no use.
```

Now, we use label Encodinig for preprocessing our Data.

In machine learning, we usually deal with datasets which contains multiple labels in one or more than one columns. These labels can be in the form of words or numbers. To make the data understandable or in human readable form, the training data is often labeled in words.

Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

```
In [130]: from sklearn.preprocessing import LabelEncoder
          for col in df_doc.columns:
              if df_doc[col].dtype=="object":
                  df_doc[col]=LabelEncoder().fit_transform(df_doc[col])
```

```
In [131]: df_doc.columns
```

```
Out[131]: Index(['Experience', 'Rating', 'Profile', 'Fees', 'source', 'Address', 'City',
                 'q1', 'q2', 'q3', 'q4', 'q5', 'q6', 'q7'],
                dtype='object')
```

```
In [132]: df_doc
```

Out[132]:

	Experience	Rating	Profile	Fees	source	Address	City	q1	q2	q3	q4	q5	q6	q7
0	24	100.0	5	100.0	1	324	4	8	302	406	225	104	31	13
1	12	98.0	0	350.0	1	926	0	5	292	406	225	104	31	13
2	9	0.0	3	300.0	1	494	0	102	396	406	225	104	31	13
3	12	0.0	0	250.0	1	74	0	14	10	406	225	104	31	13
4	20	100.0	0	250.0	1	367	1	5	411	406	225	104	31	13
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1982	10	0.0	0	NaN	0	76	0	5	172	406	225	104	31	13
1983	4	0.0	2	NaN	0	135	1	111	493	316	225	104	31	13
1984	9	0.0	4	NaN	0	658	1	102	332	406	225	104	31	13
1985	12	98.0	1	NaN	0	34	0	6	411	406	225	104	31	13
1986	8	0.0	2	NaN	0	607	1	102	296	476	225	104	31	13

7948 rows × 14 columns

Building Machine Learning Models

We then separate the features and the dependent variable into variables x and y respectively. Because our data is all numbers and there's no text in it:

```
In [133]: #Lets seprate test and train data
          df_train=df_doc.loc[df_doc["source"]==1]
          df_test=df_doc.loc[df_doc["source"]==0]
```

```
In [134]: #Resetting the index
          df_test.reset_index(drop=True,inplace=True)
```

```
In [135]: #Dropping the source column
          df_train.drop(columns=["source"],inplace=True)
          df_test.drop(columns=["source"],inplace=True)
```

```
In [136]: #Lets seprate the input and output from train dataset
          df_x=df_train.drop(columns=["Fees"])
          y=df_train[["Fees"]]
```

```
In [137]: #Lets bring every column to common scale
          from sklearn.preprocessing import StandardScaler
          sc = StandardScaler()
          x = sc.fit_transform(df_x)
          x=pd.DataFrame(x,columns=df_x.columns)
```

```
In [138]: #Train Terst Split
          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)
```

From sklearn liberary we can use mean_absolute_error :- Mean Absolute Percentage Error (MAPE) is a statistical measure to define the accuracy of a machine learning algorithm on a particular dataset. MAPE can be considered as a loss function to define the error termed by the model evaluation.

mean_squared_error of an estimator measures the average of error squares i.e. the average squared difference between the estimated values and true value. It is a risk function, corresponding to the expected value of the squared error loss. It is always non – negative and values close to zero are better. The MSE is the second moment of the error (about the origin) and thus incorporates both the variance of the estimator and its bias.

r2_score: Coefficient of determination also called as R2 score is used to evaluate the performance of a linear regression model. It is the amount of the variation in the output dependent attribute which is predictable from the input independent variable(s). It is used to check how well-observed results are reproduced by the model, depending on the ratio of total deviation of results described by the model. and mean_squared_log_error.

```
In [139]:
#to find random stat which gives least
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_log_error
from sklearn.model_selection import train_test_split
def maxr2_score(regr,df_x,y):
    min_rmsle_score=100
    for r_state in range(42,52):
        x_train, x_test, y_train, y_test = train_test_split(df_x, y,random_state = r_state,test_size=0.20)
        regr.fit(x_train,y_train)
        y_pred = regr.predict(x_test)
        rmsle_scr=np.sqrt(mean_squared_log_error(y_test,y_pred))
        print("RMSLE corresponding to ",r_state," is ",rmsle_scr)
        if rmsle_scr<min_rmsle_score:
            min_rmsle_score=rmsle_scr
            final_r_state=r_state
    print("min RMSLE corresponding to ",final_r_state," is ",min_rmsle_score)
    return final_r_state
```

Here we are using Light GBM for better accuracy. LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel, distributed, and GPU learning.
- Capable of handling large-scale data.

```
import lightgbm as lgb

from sklearn.model_selection import GridSearchCV

lg = lgb.LGBMRegressor(silent=False)

param_dist = {"max_depth": [25,50, 75],

              "learning_rate" : [0.01,0.05,0.1],

              "num_leaves": [300,900,1200],

              "n_estimators": [200]
```

}

grid_search = GridSearchCV(lg, n_jobs=-1, param_grid=param_dist, cv = 5, scoring="r2")

grid_search.fit(df_x,y)

grid_search.best_params_

```
In [ ]: #LGBM also give provision to define which all columns should be treated as categorical
        #So we will use above provision and see what is the result
        import warnings
        warnings.filterwarnings("ignore")
        params={'learning_rate': 0.01,'max_depth': 25,'n_estimators': 200,'num_leaves': 300}
        cate_features_name=['Profile', 'Rating', 'Address', 'q1', 'q2', 'q3','q4', 'q5', 'q6', "q7"]
        max_r_score=0
        for r_state in range(42,100):
            x_train, x_test, y_train, y_test = train_test_split(df_x, y,random_state = r_state,test_size=0.20)
            d_train = lgb.Dataset(x_train, label=y_train)
            lgb_model = lgb.train(params, d_train, categorical_feature = cate_features_name)
            y_pred = lgb_model.predict(x_test)
            r2_scr=r2_score(y_test,y_pred)
            print("r2 score corresponding to ",r_state," is ",r2_scr)
            if r2_scr>max_r_score:
                max_r_score=r2_scr
                final_r_state=r_state
        print("max r2 score corresponding to ",final_r_state," is ",max_r_score)
```

```
In [ ]: # Lets make lgbm as our final model
        x_train, x_test, y_train, y_test = train_test_split(df_x, y,random_state = 90,test_size=0.20)
        d_train = lgb.Dataset(x_train, label=y_train)
        lgb_model = lgb.train(params, d_train, categorical_feature = cate_features_name)
        y_pred = lgb_model.predict(x_test)
```

```
In [ ]: rmsle_scr=np.sqrt(mean_squared_log_error(y_test,y_pred))
        print("RMSLE correspondingis ",rmsle_scr)
        print("r2 score is ",r2_score(y_test,y_pred))
```

```
In [ ]: #Applying model on the test set
        doc_fee_pred=lgb_model.predict(df_test,predict_disable_shape_check=True)
```

```
In [ ]: #storing predictions as dataFrame
        doc_fee_pred=pd.DataFrame(doc_fee_pred,columns=["fees"])
```

```
In [ ]: #Storing results as csv
        doc_fee_pred.to_csv("doctor_fees_predictions.csv",index=False)
```

Concluding Remarks

By performing EDA and building a better model for use case we can save the file in csv format where the predicted fees for the Doctor is stored.