University of East London

AIN SHAMS UNIVERSITY
FACULTY OF COMPUTER AND INFORMATION SCIENCES

MACHINE LEARNING PROJECT

Doctor Fee Prediction Project

May 12th, 2024

Definition

Project Overview

In today's healthcare landscape, data-driven approaches are pivotal for enhancing patient outcomes and addressing industry challenges. Leveraging comprehensive datasets, this project endeavors to employ machine learning techniques to analyze various attributes related to medical services. By developing predictive models for regression and classification tasks, we aim to gain insights into factors influencing medical service fees while accurately predicting fee categories. Through this effort, we aim to optimize healthcare services, fostering better decision-making and ultimately improving patient care.

Problem Statement

The objective of this project is to leverage machine learning techniques to enhance the healthcare landscape by predicting medical service fees and catering to different audiences' needs. This project is structured into **two** milestones :

1. (Milestone **1**) **Regression** Task: Predicting Fee Prices - Developing a regression model to estimate the fee prices for medical services based the dataset attributes attributes.
2. (Milestone **2**) **Classification** Task: Predicting Fee Categories - Categorizing Fee Prices into categories and building a classification model to predict medical service belong to which category (e.g., expensive, cheap, affordable).

By tackling these challenges, the project endeavors to enhance transparency, efficiency, and fairness in healthcare pricing, ultimately contributing to better patient experiences and outcomes.

Features Description

- **Doctor Name :** Name of the doctor.
- **City :** The city where the doctor practices.
- **Specialization :** The area of specialization or medical field of the doctor.
- **Doctor Qualification :** The qualifications and degrees held by the doctor.
- **Experience(Years) :** The number of years of experience the doctor has in their field.
- **Total Reviews :** The total number of reviews or ratings received by the doctor.
- **Patient Satisfaction Rate (%age) :** The percentage of patients who reported satisfaction with the doctor's services.
- **Avg Time to Patients (mins) :** The average time, in minutes, taken by the doctor to attend to patients.
- **Wait Time (mins) :**The average wait time, in minutes, for patients before being attended to by the doctor.
- **Hospital Address :** The address of the hospital or clinic where the doctor practices.
- **Doctors Link :** A link to access more information about the doctor.

[MS1] Preprocessing and Regression

Problem Definition

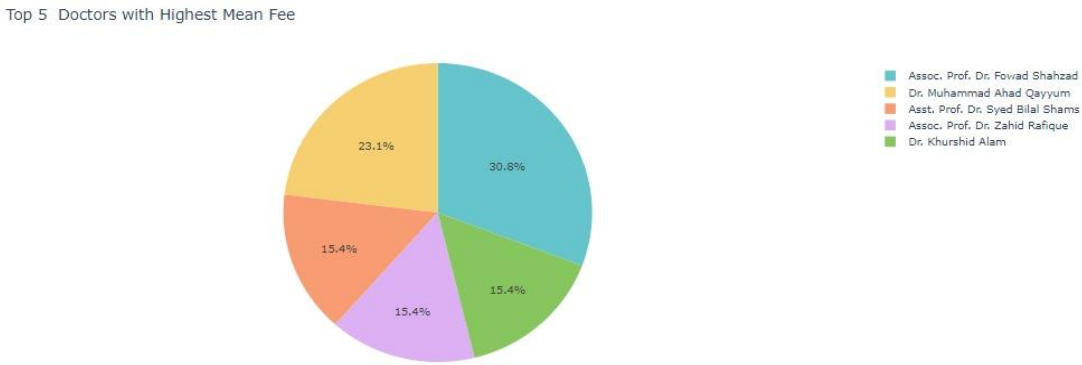
The objective of Milestone 1 is to develop a predictive regression model to estimate fee prices for medical services based on the dataset attributes. This milestone involves the choice of a predictive regression model through extensive model selection and tuning processes. To achieve the objective of estimating fee prices for medical services accurately, Milestone 1 involves a series of essential steps:

1. **Exploratory Data Analysis (EDA):** Conduct a deep analysis of the dataset to understand the distribution and relationships between the features.Observe and handle any errors within the features.Explore the distribution of fee prices and identify potential correlations with other features.
2. **Data Preprocessing:** Handle missing values, outliers, and any inconsistencies in the dataset. Perform data cleaning and normalization to ensure the quality and consistency of the data.
3. **Feature Encoding:** Encode categorical variables into numerical representations suitable for machine learning algorithms. Utilize techniques such as one-hot encoding or label encoding to transform categorical attributes into a format understandable by regression models.
4. **Feature Selection:** Identify and select relevant features that contribute most to predicting fee prices.Using techniques such as correlation analysis or feature importance ranking.
5. **Model Building:** Experimenting with multiple regression algorithms through extensive model selection, tuning, and evaluation processes to identify the best-performing model.
6. **Model Selection:** After tuning and evaluating the models, selecting the best-performing regression model based on its performance on the dataset. Subsequently, training the chosen model and assessing its performance on test data to analyze and gain insights from the results.

Step 1: Exploratory Data Analysis

Performing Data Analysis as the Primary Task to Understand Data, Check the Distributions and Observe any errors.

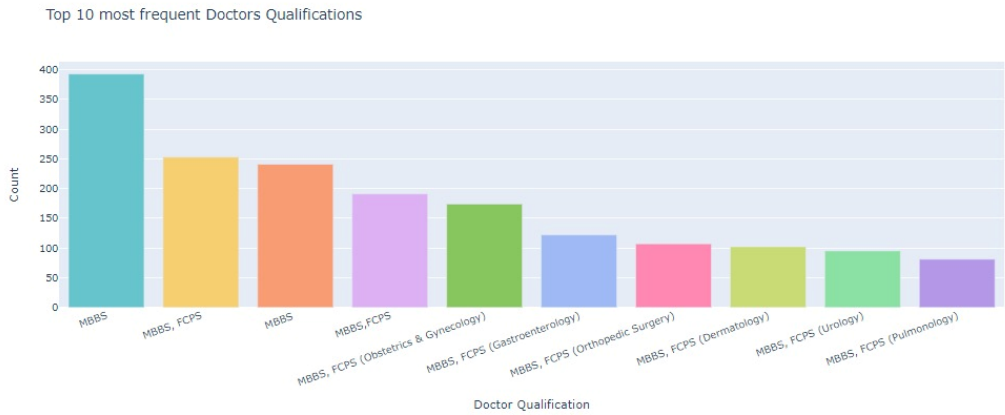
- **Fig. 1** Checking the Percentage of the Top 5 Doctors with the highest Fees



Observations:

- Assoc Prof Dr. Fowad Shahzad has the highest Fees price.
- The second highest fees comes to Dr.Muhamed Ayad Qayum.

- **Fig. 2** Checking the top 10 frequent Doctors qualifications

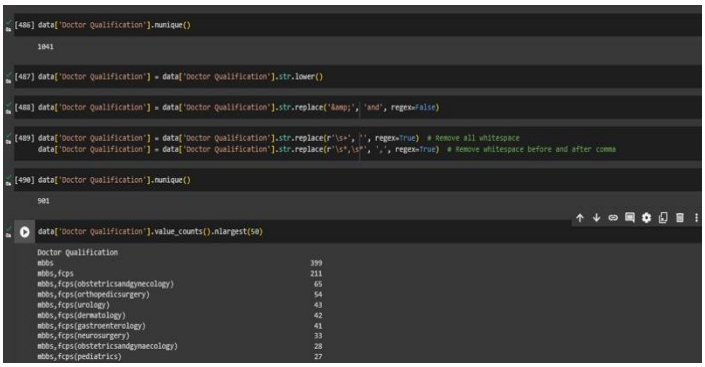


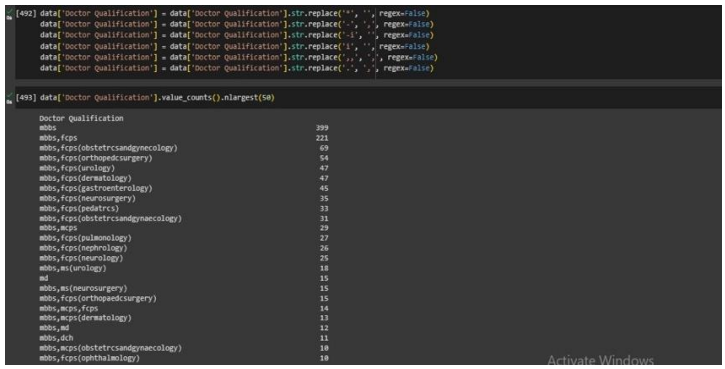
Observations:

- MBBS degree is the most found and frequent degrees among the doctors within the data.
- We identified a potential error and entries in this column, for having 2 bar charts with same values for MBBS and MBBS, FCPS suggesting the existence of some errors and the repetition of this values with different formats.

For handling this problem, we used string manipulation using string functions and regular expressions

- Identifying the unique values present within the columns.
- Converting all the values to lower case for ensuring all entries are within the same format.
- Replacing any entry containing the value (&amp) by and
- Using Regular expressions for removing any identified special characters other than the commas.



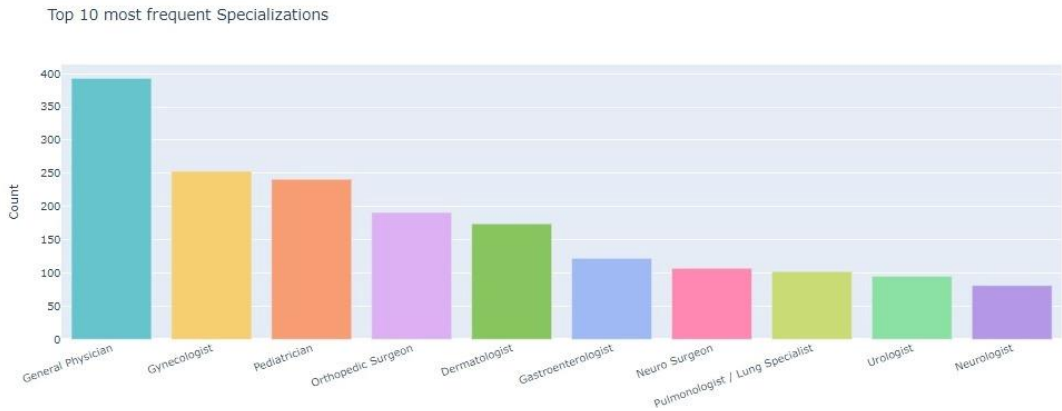


- The unique values have decreased from 1041 to 901.
- All formats have been standardized with all lower case, no extra characters, and no white spaces, but identified that some entries contained the same values with different misspelling like mbbs, fcps(obstetricsandgynecology) and mbbs, fcps(obstetricsandgynaecolog).

So, for handling this we used **string closest matching** to find the closest string to each other and replace them with the same value.

```
[494] data['Doctor Qualification'].nunique()

      859

[495] from difflib import get_close_matches
      unique_values=data['Doctor Qualification'].value_counts().index

      for value in unique_values:
        matches = get_close_matches(value,unique_values)
        data['Doctor Qualification'] = data['Doctor Qualification'].replace(matches, value)
        unique_values=list(filter(lambda x: x not in matches, unique_values))
```

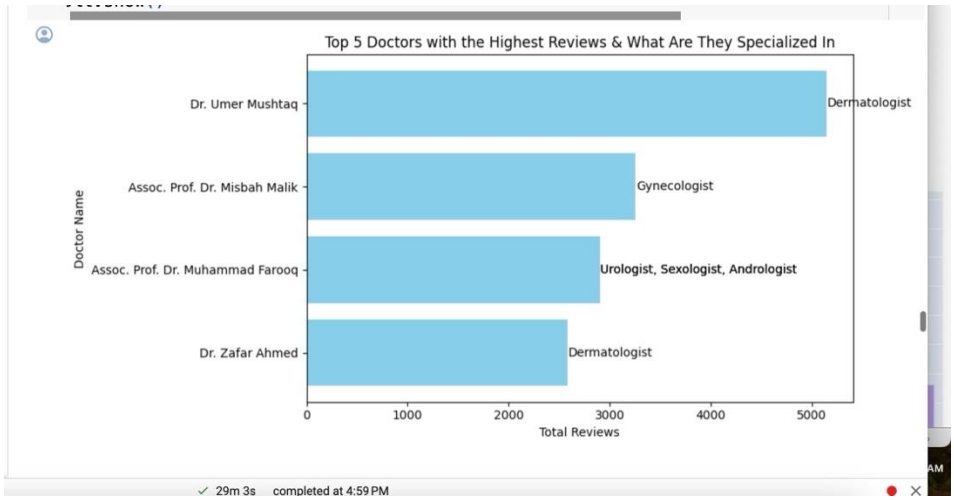
- **Fig. 3** Visualizing the distribution of the top 10 most frequent Specializations



Observations:

- Most of the doctors are General Physicians.
- Consistent values found within the column with no errors identified.

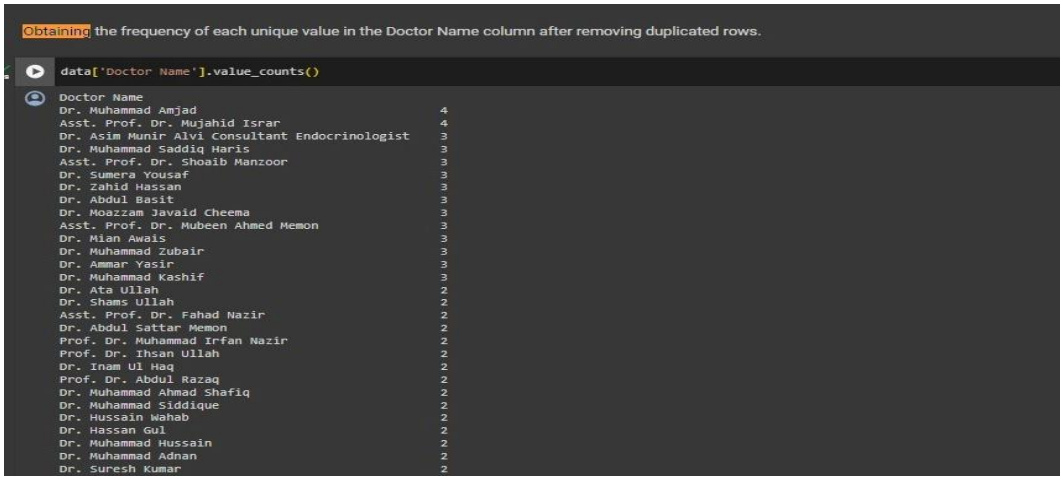
- **Fig. 4** Showing The top 5 Doctors with highest total reviews and what they are specialized in



Observations:

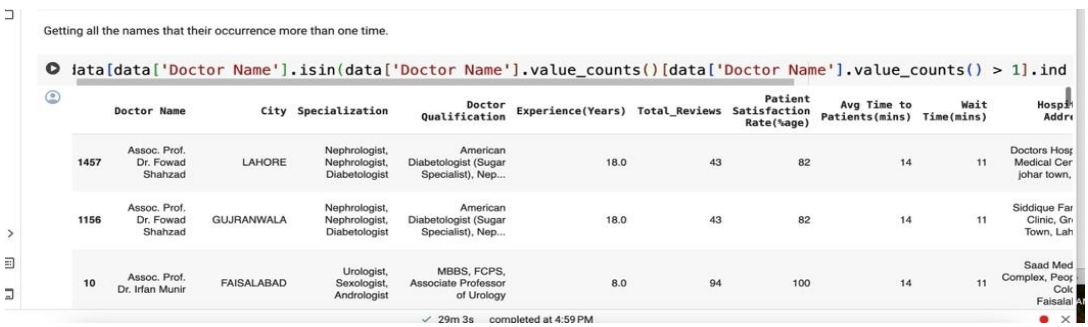
- Dermatologist, Dr.Umar Mushtaq has the highest total reviews value for about 5000 reviews.

❖ **Fig. 5** Checking the Occurrence of each doctor in the data



Observations:

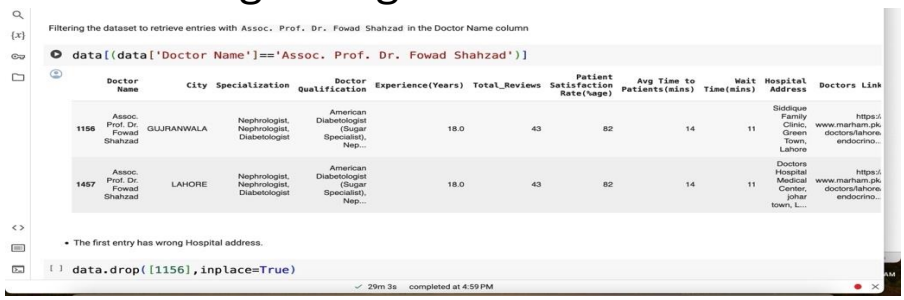
- Some Doctors have been identified to be repeated more than one time, we will analyze the values to check if they represent the same doctor and have consistent values, or different doctor with the same name.



Observations:

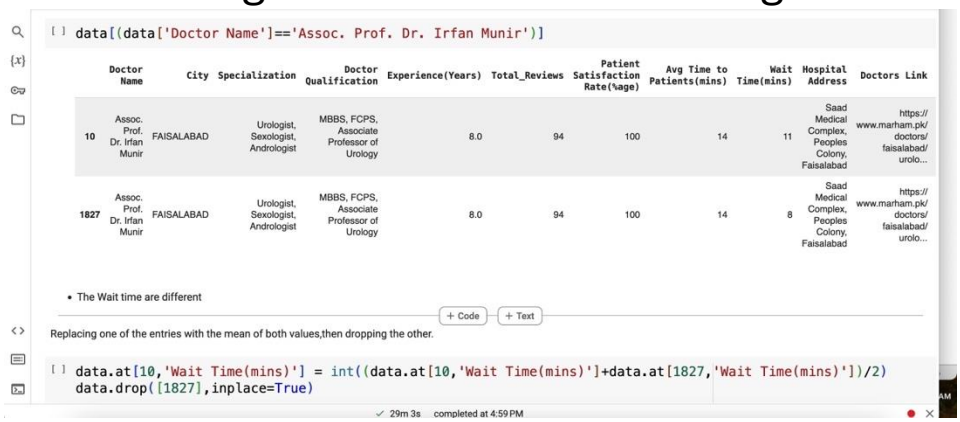
- Some of the repeated doctors have inconsistent values regarding the cities they work in, and the hospital addresses provided by showing inconsistency in values of the records. So, we aimed to analyze each doctor from this filtered data frame to correct the wrong and inconsistent data, and to check if there are different doctors with same name.

- For entries having wrong cities:



Keeping the entry with the correct city and hospital address and dropping the other.

- For entries having different wait time or avg time:



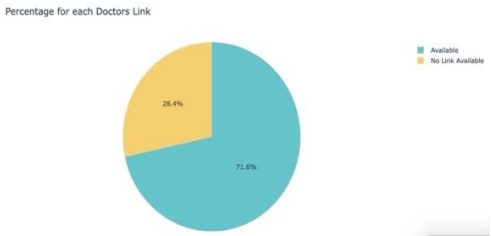
Replacing the wait time with the average of the two records in one row and dropping the other.

Feature Engineering:

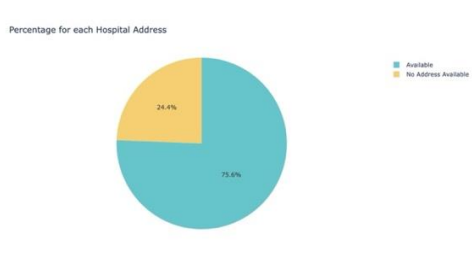
Based on the columns Doctors Link and Hospital Address both columns have entries with No address available or No Link available for certain doctors, so to facilitate preprocessing the columns we will assign the two columns' values representing the availability and the unavailability of the doctor's links and hospital address.

```
[ ] data['Doctors Link'] = np.where(data['Doctors Link']=='No Link Available',data['Doctors Link'],'Available')
    data['Hospital Address'] = np.where(data['Hospital Address']=='No Address Available',data['Hospital Address'],'Available')
```

- **Fig. 6** Checking the percentages of the availability of the Doctor Links and Hospital Addresses after modification



Most of the doctors have links on the website

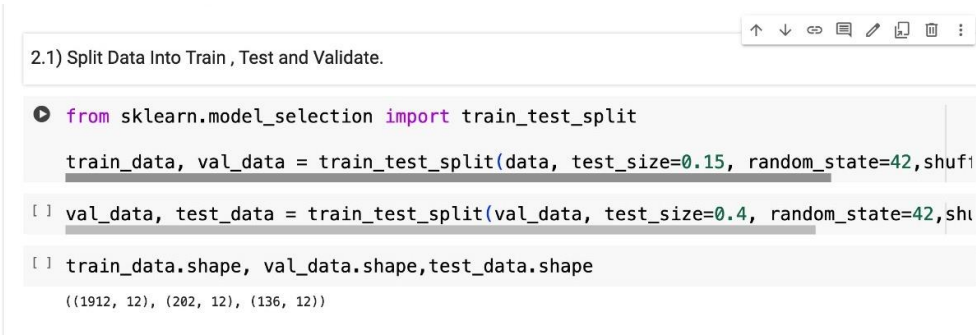


Most of the doctors their workspace is known

Step 2: Preprocessing

Pre-processing step for all features before the modeling to ensure that all features have same scale, numerical values, and no missing values.

- Splitting the data into 3 sets: Training, Validation, and Testing.



2.1) Split Data Into Train , Test and Validate.

```
from sklearn.model_selection import train_test_split

train_data, val_data = train_test_split(data, test_size=0.15, random_state=42,shuf
```

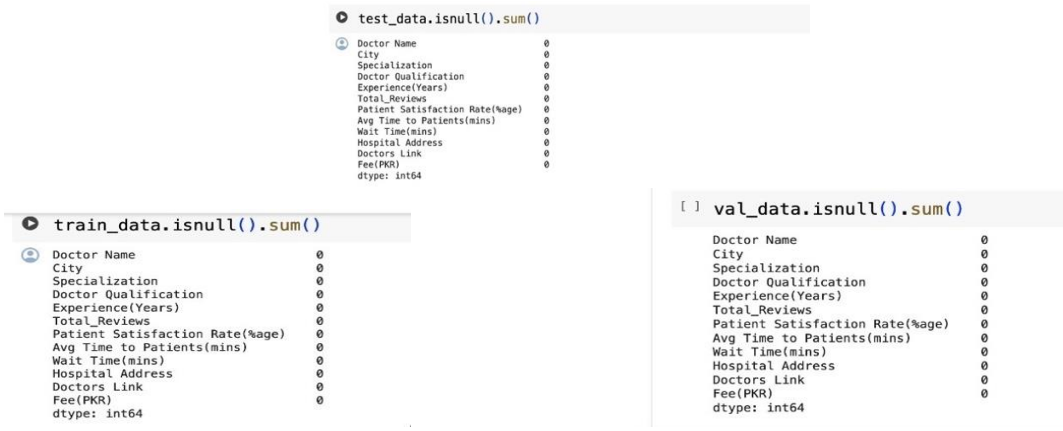
```
val_data, test_data = train_test_split(val_data, test_size=0.4, random_state=42,sh
```

```
train_data.shape, val_data.shape,test_data.shape
```

```
((1912, 12), (202, 12), (136, 12))
```

Training set size: 1912
Validation set size: 202
Test set size:136

- Checking for any missing values in all sets



```
test_data.isnull().sum()
```

```
Doctor Name                       0
City                              0
Specialization                    0
Doctor Qualification              0
Experience(Years)                 0
Total_Reviews                     0
Patient Satisfaction Rate(%age)   0
Avg Time to Patients(mins)        0
Wait Time(mins)                   0
Hospital Address                  0
Doctors Link                      0
Fee(PKR)                          0
dtype: int64
```

```
train_data.isnull().sum()
```

```
Doctor Name                       0
City                              0
Specialization                    0
Doctor Qualification              0
Experience(Years)                 0
Total_Reviews                     0
Patient Satisfaction Rate(%age)   0
Avg Time to Patients(mins)        0
Wait Time(mins)                   0
Hospital Address                  0
Doctors Link                      0
Fee(PKR)                          0
dtype: int64
```

```
val_data.isnull().sum()
```

```
Doctor Name                       0
City                              0
Specialization                    0
Doctor Qualification              0
Experience(Years)                 0
Total_Reviews                     0
Patient Satisfaction Rate(%age)   0
Avg Time to Patients(mins)        0
Wait Time(mins)                   0
Hospital Address                  0
Doctors Link                      0
Fee(PKR)                          0
dtype: int64
```

Observations:

- No missing values present in all sets.

➢ Checking for any duplicates in all sets

```
[ ] train_data.duplicated().sum()
2
```

• 2 duplicated rows has been identfied within the training set,we would go for dropping them.

```
[ ] train_data.drop_duplicates(inplace=True)
[ ] train_data.duplicated().sum()
0
```

Observations:

- 2 duplicates were found in the training set, so we dropped them.

```
val_data.duplicated().sum()
0
```

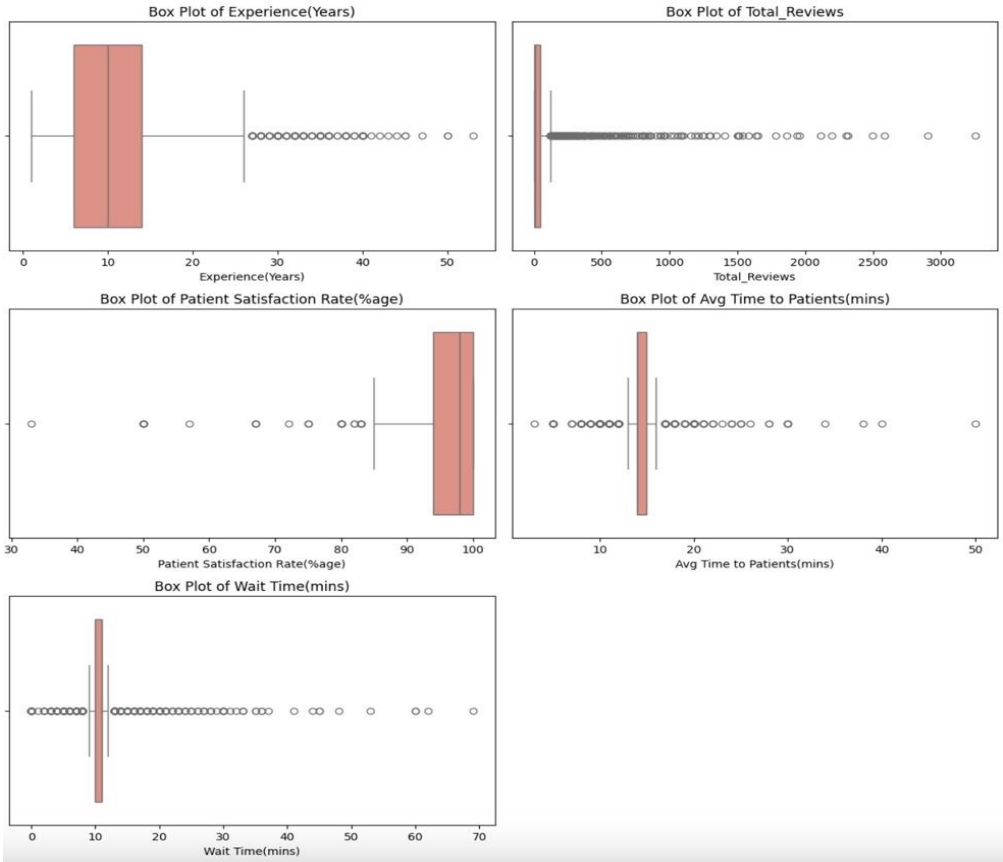
• No duplication were identified in validation set.

```
[ ] test_data.duplicated().sum()
0
```

Observations:

- No other duplicates presented in sets.

➢ Outliers Handling



Observations:

- Many outliers observed in almost all columns.

Ensuring that these outliers are real ones not random values through analyzing the outliers' values within each column and checking their distributions.

Inspecting the outliers' values within each column to confirm if they are real values or random ones:

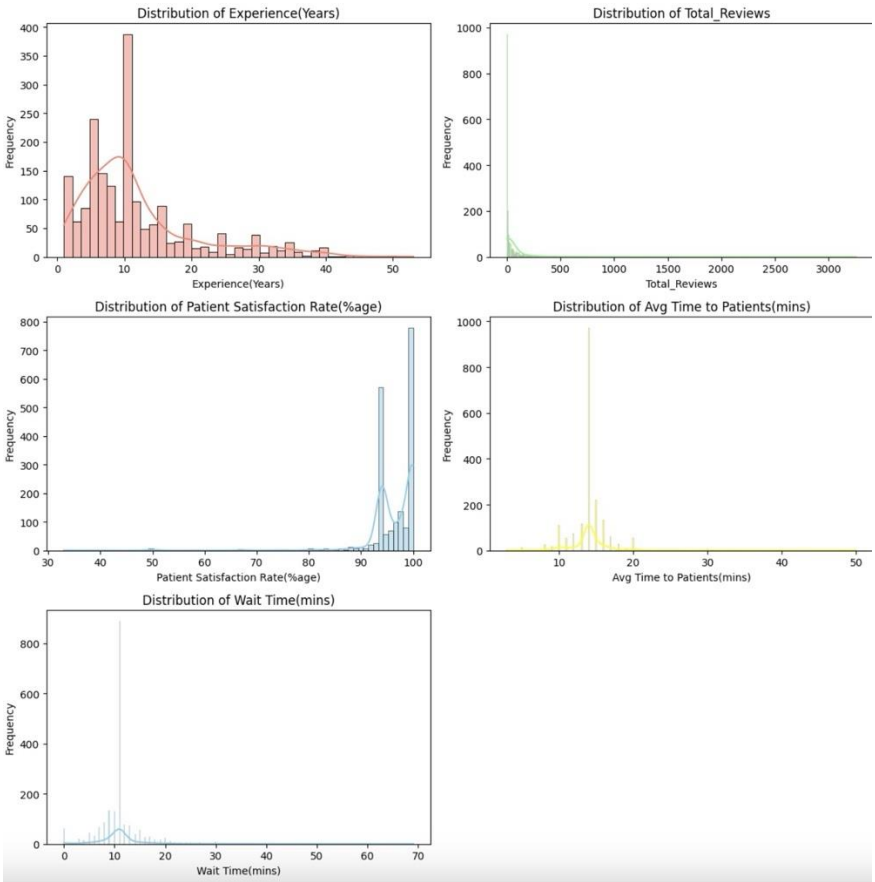
```
for col in numerical_columns:
    inspect_outliers(train_data[col])
```

```
Column: Experience(Years)
Lower Limit: -6.0
Upper Limit: 26.0
10th Percentile: 3.0
90th Percentile: 25.0
Number of outliers: 177
Percentage of outliers: 9.27%
Outliers:
229     35.0
222     39.0
674     28.0
318     32.0
308     38.0
2104    29.0
186     50.0
2084    32.0
1593    28.0
1530    35.0
387     44.0
112     27.0
128     34.0
1980    38.0
437     33.0
```

Observations:

- After analyzing each column values, it is ensured that all the values of outliers are real not random or wrong entries.

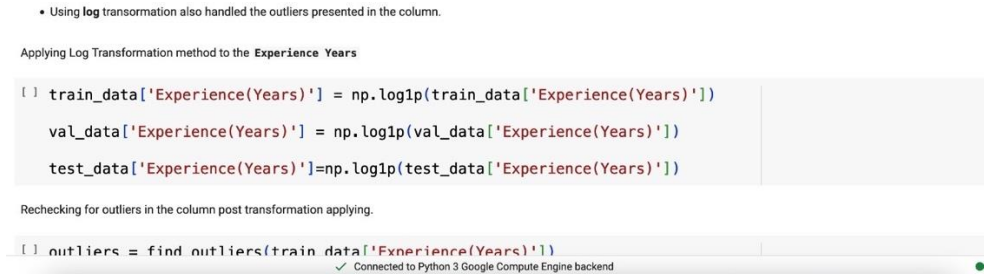
Secondly checking the distributions of the columns



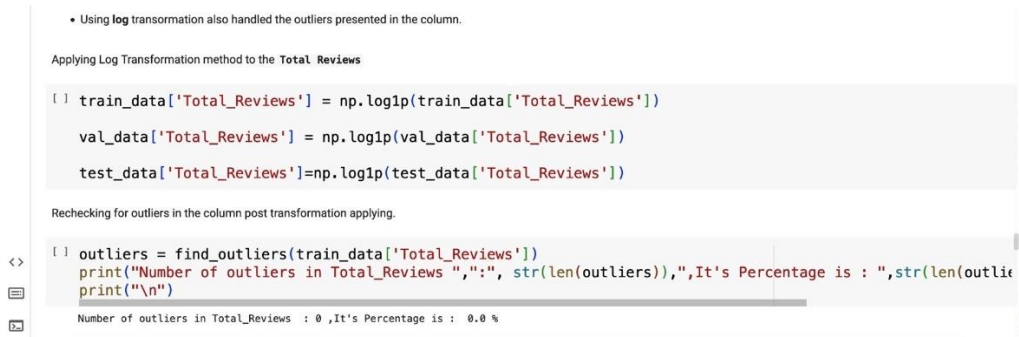
Observations:

- Most of the columns are right skewed proving the presence of the large outlier values in most of the columns. Transforming these columns to be normally distributed will fix the skewed and will remove the outliers found based on the suitable transformations used.

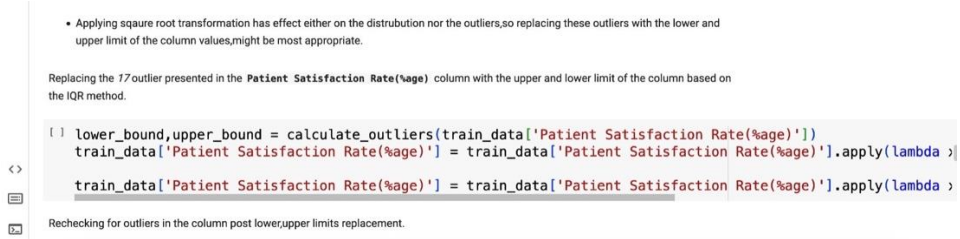
i. The best method for **Experience Years** column was applying **Log** transformation aimed in making the column normally distributed and removal of all outliers.



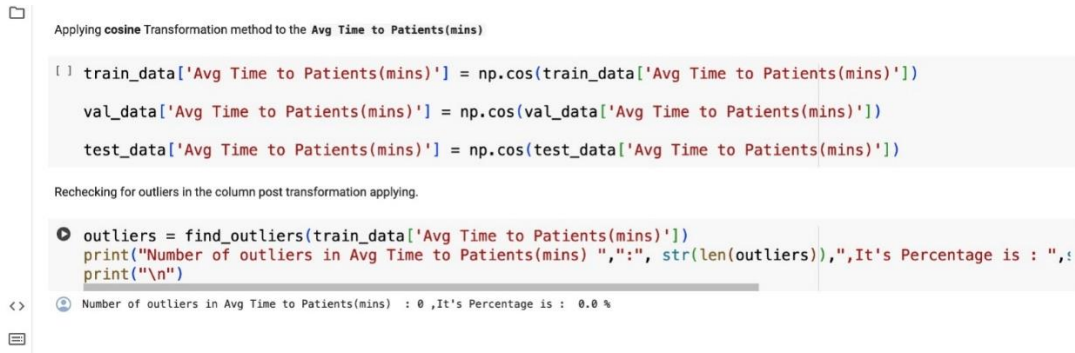
ii. The best method for **Total Reviews** column was applying **Log** transformation aimed in making the column normally distributed and removal of all outliers.



iii. No method fixed column **Patient Satisfaction Rate,** and since the small number of outliers present, we replaced them with the upper and lower using IQR method.



iv. For the column **Avg Wait Time,** common transformations like (cbrt, sqrt, log) did not fix the skewness, and they also increased the outliers of the column, but applying trigonometric transformation like **cos**, was most appropriate making the column normally distributed and outliers removed.

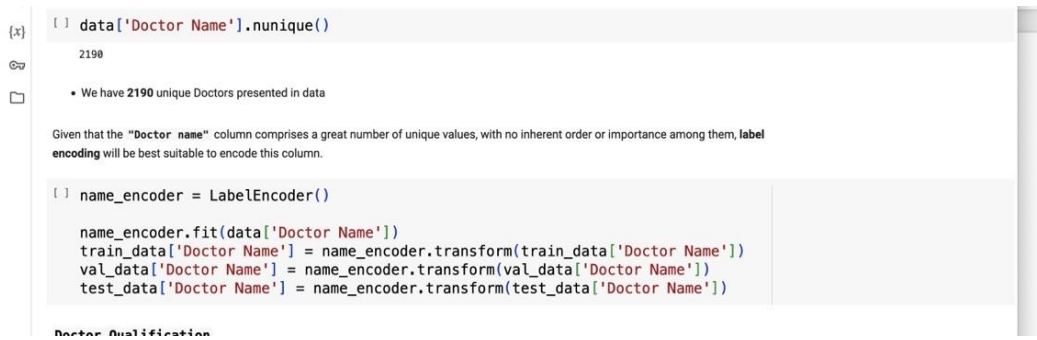


v. For the column **Wait Time** as well, common transformations like (cbrt, sqrt, log) did not fix the skewness, and they also increased the outliers of the column, but applying trigonometric transformation like **sin**, was most appropriate making the column normally distributed and outliers removed.

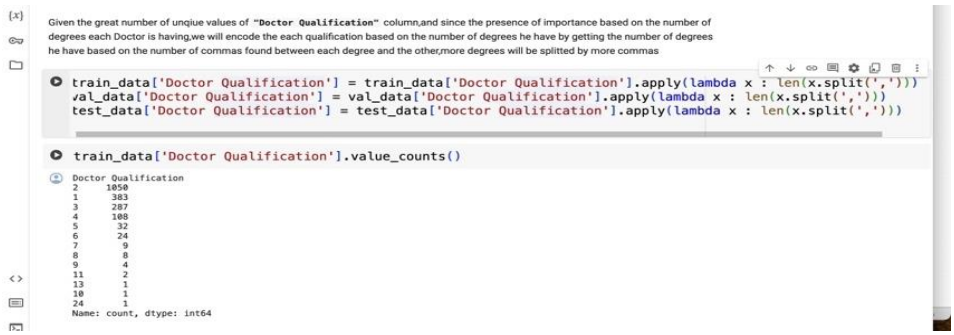
Features Encoding

Encoding the categorical features in the dataset involves transforming them into a suitable numerical format using appropriate techniques based on the nature of each column. This preprocessing step is crucial before training the model.

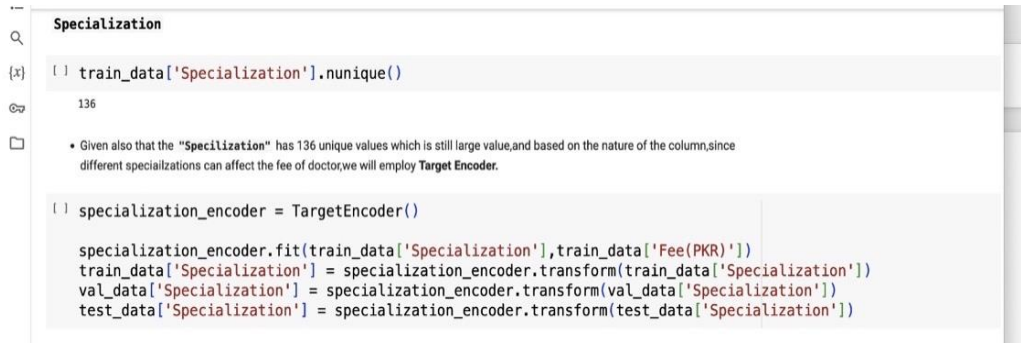
i. **Doctor name** column: we chose **<u>Label Encoding</u>** as it has large number of values to encode, and the values don't have any order or importance between them.



ii. **Doctor Qualification** column: We chose to apply encoding based on the number of degrees each doctor has, by counting the number of commas separating each degree, Since this column has large unique values and to maintain the importance of higher qualifications giving them higher value.



iii. **Specialization** column: best method was to apply Target Encoder fees can change according to the specialization of the doctor.



Specialization

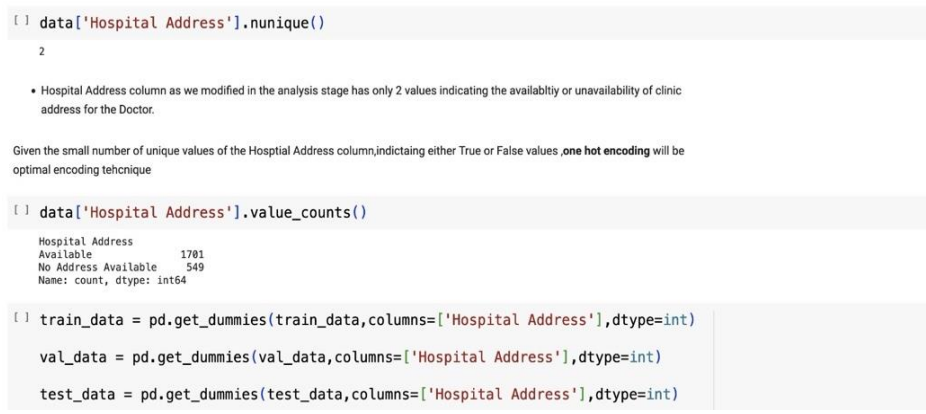
```
[ ] train_data['Specialization'].nunique()
```

```
136
```

- Given also that the **"Specilization"** has 136 unique values which is still large value,and based on the nature of the column,since different speciailzations can affect the fee of doctor,we will employ **Target Encoder.**

```
[ ] specialization_encoder = TargetEncoder()

    specialization_encoder.fit(train_data['Specialization'],train_data['Fee(PKR)'])
    train_data['Specialization'] = specialization_encoder.transform(train_data['Specialization'])
    val_data['Specialization'] = specialization_encoder.transform(val_data['Specialization'])
    test_data['Specialization'] = specialization_encoder.transform(test_data['Specialization'])
```

iv. **Hospital Address** and **Doctors Link** columns: since in the analysis we made it binary values if the doctors have address, link or not available, One Hot Encoding technique was best suitable for this column due to the small number of values (binary) that represent either true or false.



```
[ ] data['Hospital Address'].nunique()
```

```
2
```

- Hospital Address column as we modified in the analysis stage has only 2 values indicating the availabltiy or unavailability of clinic address for the Doctor.

Given the small number of unique values of the Hosptial Address column,indictaing either True or False values ,**one hot encoding** will be optimal encoding tehcnique

```
[ ] data['Hospital Address'].value_counts()
```

```
Hospital Address
Available              1701
No Address Available    549
Name: count, dtype: int64
```

```
[ ] train_data = pd.get_dummies(train_data,columns=['Hospital Address'],dtype=int)

    val_data = pd.get_dummies(val_data,columns=['Hospital Address'],dtype=int)

    test_data = pd.get_dummies(test_data,columns=['Hospital Address'],dtype=int)
```

v. **City** column: Target encoder also was most suitable as fees can get affected according to the city the Doctor working in.

City

```
[ ] data['City'].nunique()
```

115

- We have 115 cities in the data

Given also that the **"City"** has great number of unique values which is and based on the nature of, since it doesnot inherit any order and improtance among cities ,and different citites can affect the fee of doctor,we will employ **Target Encoder.**

```
city_encoder = TargetEncoder()

train_data['City'] = city_encoder.fit_transform(train_data['City'],train_data['Fee(PKR)'])
val_data['City'] = city_encoder.transform(val_data['City'])
test_data['City'] = city_encoder.transform(test_data['City'])
```

Features Scaling

Rescaling all features within a specified range ensures consistency and scalability of values, thereby enhancing the performance of machine learning models.

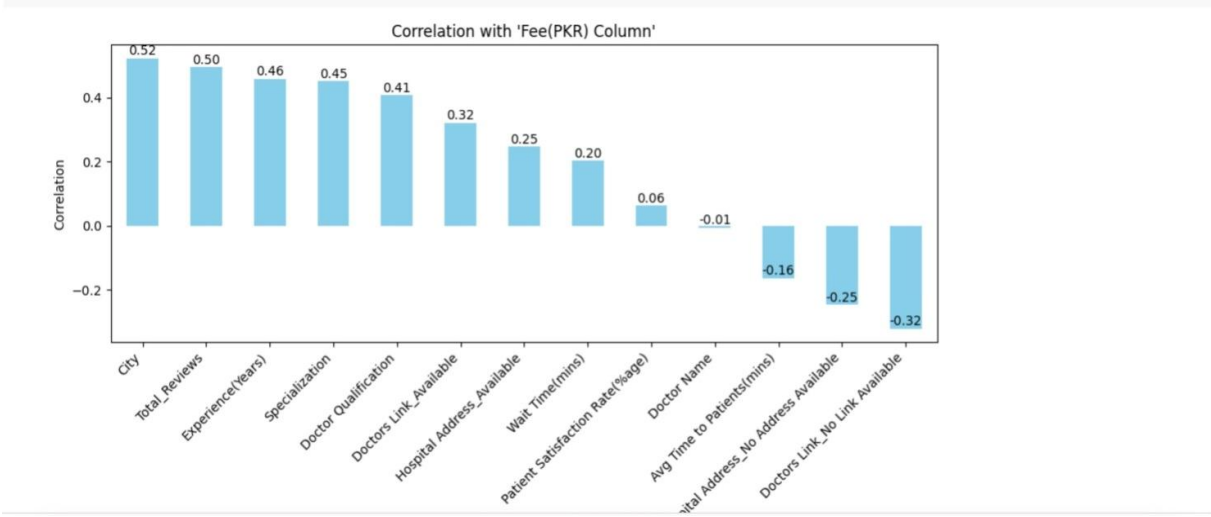
- Chose **Min Max Scaling**, since some columns still skewed and don't inherit Normal Gaussian Distribution.

```
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()

train_data_scaled = min_max_scaler.fit_transform(train_data.iloc[:,:-1])

val_data_scaled = min_max_scaler.transform(val_data.iloc[:,:-1])
test_data_scaled = min_max_scaler.transform(test_data.iloc[:,:-1])

train_data_scaled = pd.DataFrame(train_data_scaled, columns=train_data.iloc[:,:-1].columns)
train_data_scaled['Fee(PKR)'] = train_data['Fee(PKR)'].values

val_data_scaled = pd.DataFrame(val_data_scaled, columns=val_data.iloc[:,:-1].columns)
val_data_scaled['Fee(PKR)'] = val_data['Fee(PKR)'].values

test_data_scaled = pd.DataFrame(test_data_scaled, columns=test_data.iloc[:,:-1].columns)
test_data_scaled['Fee(PKR)'] = test_data['Fee(PKR)'].values
```

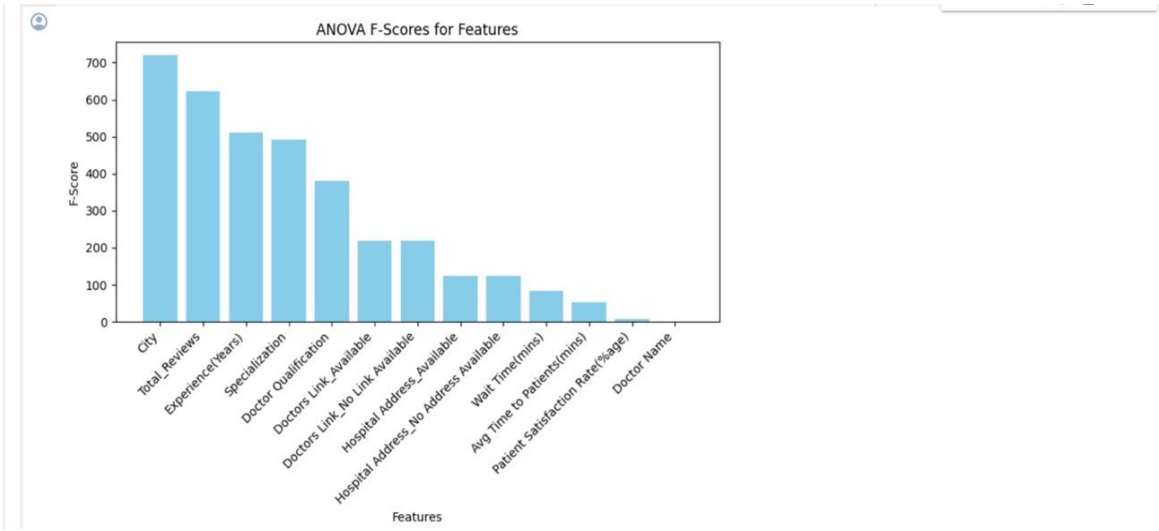
Step 3: Feature Selection

After all preprocessing steps have been done, all features are encoded, and scaled to the range **[0,1],** comes Identifying and selecting the relevant features that contribute most to predicting fee prices and will be used for model's training.

- **Correlation Analysis**: Visualizing the correlation between all features and the target variable, with choosing features highly correlated to target



- **ANOVA F-value (Analysis of Variance):** Visualizing ANOVA Scores between features and target to support choices of correlation.



Based on Correlation and ANOVA F-value analysis, they both align on feature significance. Excluding Patient Satisfaction Rate and Doctor Name due to negligible correlation and ANOVA scores. Retaining remaining features with strong correlations and ANOVA scores.

Final Selected features are:

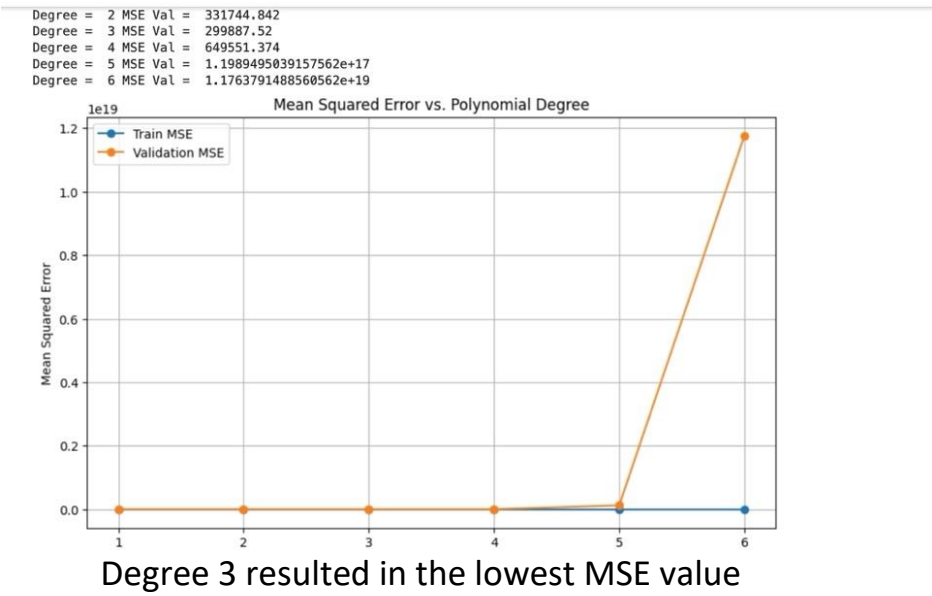
[City, Specialization, Doctor Qualification, Experience (Years), Total_Reviews, Avg Time to Patients(mins), Wait Time(mins), Hospital Address,Doctors Link]

Step 5: Modeling

After feature selection and preprocessing all the features, we experimented with various regression techniques for model selection and evaluation to choose best and reliable model for predicting our target.

1. Polynomial Regression

To choose best degree of polynomial regression we plotted the MSE with range of degrees and training a polynomial model to evaluate which degree resulted in the lowest MSE value.



Degree 3 resulted in the lowest MSE value

Training a polynomial Regression model on transformed features to degree 3 and evaluating its performance

```
print("Evaluating Training Metrics :\n")
print("Training Accuracy of Polynomial Regression  = ",r2_score(y_train,y_train_pred)*100)
print("Training MAE of Polynomial Regression = ",mean_absolute_error(y_train,y_train_pred)*100)
print("Training MSE of Polynomial Regression = ",mean_squared_error(y_train,y_train_pred)*100)
print("Training RMSE of Polynomial Regression = ",np.sqrt(mean_squared_error(y_train,y_train_pred))*100)
```

```
Evaluating Training Metrics :

Training Accuracy of Polynomial Regression  =  54.36237120338991
Training MAE of Polynomial Regression =  39652.34293193717
Training MSE of Polynomial Regression =  28775739.708769634
Training RMSE of Polynomial Regression =  53643.02350610901
```

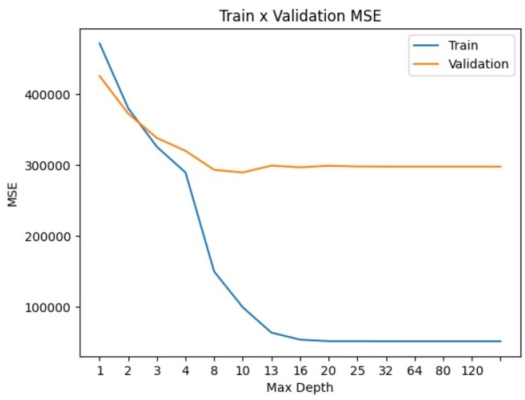
```
print("Evaluating Validation Metrics :\n")

print("Accuracy = ",r2_score(y_val,predictions)*100)
print("MAE = ",mean_absolute_error(y_val,predictions)*100)
print("MSE = ",mean_squared_error(y_val,predictions)*100)
print("RMSE = ",np.sqrt(mean_squared_error(y_val,predictions))*100)

models_eval.loc[len(models_eval)]=['Poly',(r2_score(y_val,predictions)*100),(mean_absolute_error(y_val,pred:
```

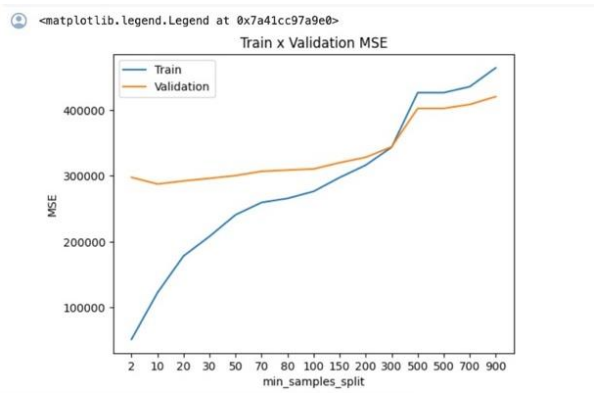
```
Evaluating Validation Metrics :

Accuracy =  43.58731455792122
MAE =  42362.87128712871
MSE =  29988752.04207921
RMSE =  54761.98685409361
```

By evaluating the polynomial regression model, the performance wasn't very good, indicating underfitting issue due to the poor performance on both sets.

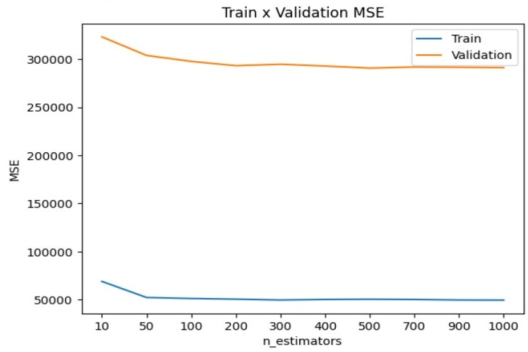
2. Random Forest Regressor

Performing Pre-Pruning Technique for the Random Forest trees to prevent overfitting issues by choosing the values of the hyperparameters before training





<matplotlib.legend.Legend at 0x7a41cc97b3d0>



- Based on the plots observations, we will select the **min_samples_split=30 ,max_depth=8 , n_estimators=500.**

Based on analyzing the values for the hyper parameters **[max_depth, min_samples_split, estimators]** with training and validation MSE, choosing the values **[8,30,500]** respectively shows the most appropriate values for preventing overfitting, and training the Random Forest model using these parameters.

```
print("Training Accuracy of Random Forest  = ",r2_score(y_train,y_train_pred)*100)
print("Training MAE of Random Forest = ",mean_absolute_error(y_train,y_train_pred)*100)
print("Training MSE of Random Forest = ",mean_squared_error(y_train,y_train_pred)*100)
print("Training RMSE of Random Forest = ",np.sqrt(mean_squared_error(y_train,y_train_pred))*100)
```

```
Evaluating Training Metrics :

Training Accuracy of Random Forest  =  64.85456071914896
Training MAE of Random Forest =  33641.01517501157
Training MSE of Random Forest =  22160134.944856256
Training RMSE of Random Forest =  47074.55251498017
```

```
print("Evaluating Validation Metrics :\n")

print("Accuracy = ",r2_score(y_val,predictions)*100)
print("MAE = ",mean_absolute_error(y_val,predictions)*100)
print("MSE = ",mean_squared_error(y_val,predictions)*100)
print("RMSE = ",np.sqrt(mean_squared_error(y_val,predictions))*100)

models_eval.loc[len(models_eval)]=['RandomForest',(r2_score(y_val,predictions)*100),(mean_absolute_error(y_v
```

```
Evaluating Validation Metrics :

Accuracy =  44.292407779277134
MAE =  40957.25989648674
MSE =  29613927.379574824
RMSE =  54418.680046078676
```

By Evaluating the performance of the random forest, It shows good improvement the training data, while a small improvement on the test set, as observed. The performance is getting better but not significant with no signs of overfitting.

3. Gradient Boosting Regressor

Performing Hyperparameter tuning to select the best hyperparameter combinations of the GB model for effectively selecting the values that have the lowest MSE using Bayes Search.

```
%%time
param_grid = {
    'n_estimators': (100, 1000),
    'learning_rate': (0.01, 1.0),
    'max_depth': (1, 30),
    'min_samples_split': (2, 50)
}


gb_regressor = GradientBoostingRegressor(random_state=42)

bayes_search = BayesSearchCV(estimator=gb_regressor, search_spaces=param_grid , n_iter=120, cv=5, scoring='r
bayes_search.fit(X_train, y_train)

best_params = bayes_search.best_params_
best_score = -bayes_search.best_score_

print("Best Parameters:", best_params)
print("Best Score (MSE):", best_score)
```

```
Best Parameters: OrderedDict([('learning_rate', 0.11706307199999215), ('max_depth', 3), ('min_samples_split', 25), ('n_estimators', 100)])
Best Score (MSE): 341872.9032137771
CPU times: user 24min 7s, sys: 4min 27s, total: 28min 34s
Wall time: 25min 10s
```

Based on the hyperparameters selected using the Bayes Search, we train the Gradient Boosting model and evaluates its performance.

```
print("Training Accuracy of Gradient Boosting  = ",r2_score(y_train,y_train_pred)*100)
print("Training MAE of Gradient Boosting = ",mean_absolute_error(y_train,y_train_pred)*100)
print("Training MSE of Gradient Boosting = ",mean_squared_error(y_train,y_train_pred)*100)
print("Training RMSE of Gradient Boosting = ",np.sqrt(mean_squared_error(y_train,y_train_pred))*100)
```

```
Evaluating Training Metrics :

Training Accuracy of Gradient Boosting  =  66.38993142509055
Training MAE of Gradient Boosting =  33525.860107046625
Training MSE of Gradient Boosting =  21192042.847268425
Training RMSE of Gradient Boosting =  46034.8160062234
```

```
print("Evaluating Validation Metrics :\n")

print("Accuracy = ",r2_score(y_val,predictions)*100)
print("MAE = ",mean_absolute_error(y_val,predictions)*100)
print("MSE = ",mean_squared_error(y_val,predictions)*100)
print("RMSE = ",np.sqrt(mean_squared_error(y_val,predictions))*100)

models_eval.loc[len(models_eval)]=['GradientBoosting',(r2_score(y_val,predictions)*100),(mean_absolute_erro
```

```
Evaluating Validation Metrics :

Accuracy =  45.966060799685316
MAE =  40270.40579281365
MSE =  28724220.30322883
RMSE =  53594.98139119821
```

The Gradient Boosting post tuninng shows a slight improvement compared to Random Forest on both sets,but still slight poor performance on the validation set.

4. Support Vector Machine (SVR)

Performing Hyperparameter tuning to select the best hyperparameter combinations of the SVM model for effectively selecting the values that have the lowest MSE using Grid Search.

```
%%time
param_grid = {
    'kernel': ['linear', 'poly', 'rbf'],
    'C': [0.1, 0.6, 4, 2, 10, 20, 100, 1000],
    'gamma': ['scale', 'auto'],
    'degree': [2, 3, 4, 5,6]
}


svm = SVR()

grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', ver
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_score = -grid_search.best_score_
print("Best Parameters:", best_params)
print("Best Score (MSE):", best_score)
```

```
Best Parameters: {'C': 1000, 'degree': 2, 'gamma': 'scale', 'kernel': 'poly'}
Best Score (MSE): 331950.91035481985
CPU times: user 3min 52s, sys: 297 ms, total: 3min 52s
Wall time: 3min 55s
```

Based on the hyperparameters selected using the Grid Search, we train the SVR model and evaluates its performance.

```
print("Training Accuracy of SVM   = ",r2_score(y_train,y_train_pred)*100)
print("Training MAE of SVM  = ",mean_absolute_error(y_train,y_train_pred)*100)
print("Training MSE of SVM = ",mean_squared_error(y_train,y_train_pred)*100)
print("Training RMSE of SVM = ",np.sqrt(mean_squared_error(y_train,y_train_pred))*100)
```

```
Evaluating Training Metrics :

Training Accuracy of SVM   =  49.55577238971289
Training MAE of SVM  =  38171.77609863048
Training MSE of SVM =  31806428.19969151
Training RMSE of SVM =  56397.18805019582
```

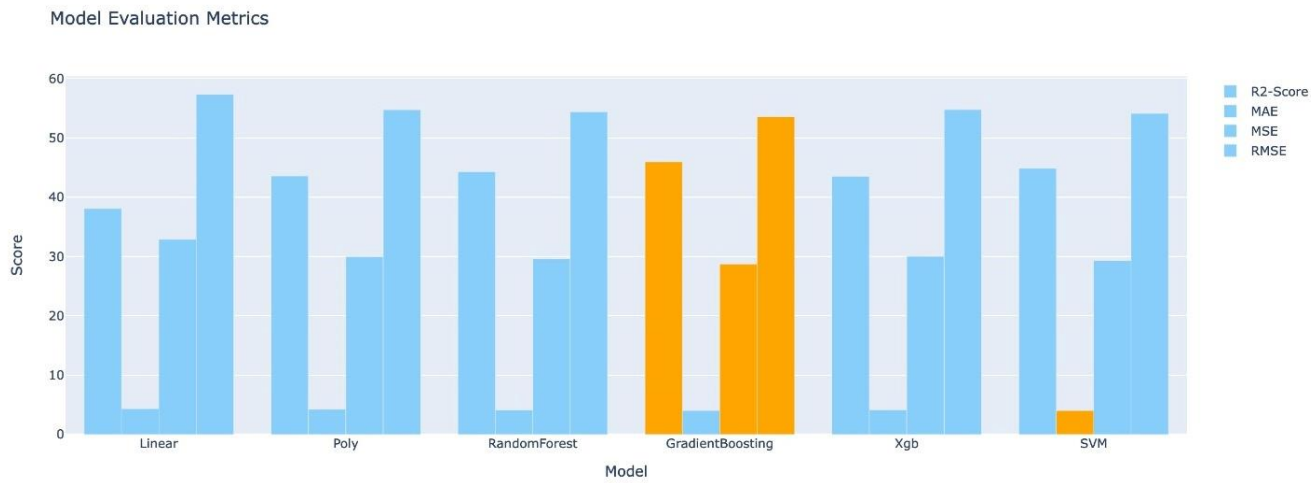
```
print("Evaluating Validation Metrics :\n")

print("Accuracy = ",r2_score(y_val,predictions)*100)
print("MAE = ",mean_absolute_error(y_val,predictions)*100)
print("MSE = ",mean_squared_error(y_val,predictions)*100)
print("RMSE = ",np.sqrt(mean_squared_error(y_val,predictions))*100)

models_eval.loc[len(models_eval)]=['SVM',(r2_score(y_val,predictions)*100),(mean_absolute_error(y_val,predic
```

```
Evaluating Validation Metrics :

Accuracy =  44.875061237913016
MAE =  40235.35651102042
MSE =  29304191.192393452
RMSE =  54133.3457236789
```

By also evaluating the performance of the SVR model on both sets,it shows worse performance on the training compared to Random Forest and Gradient Boosting Regressor,with almost the same validation accuracy.

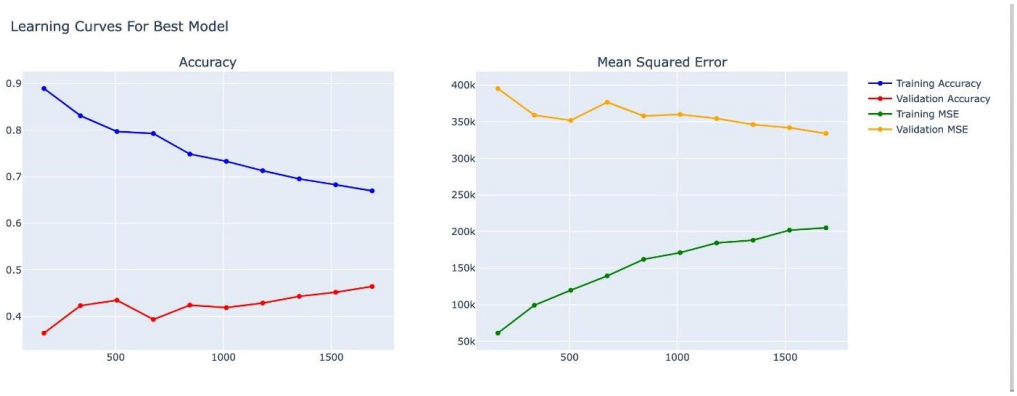
Final Evaluation and Model Selection

Through evaulating all the models used and trained through the model selection process,we find that all models got almost the same validation accuracy,sugessting no improvent can be done on all models and this is the best performance it could be achived.Hence,Comparing all models used metrices and selecting the one showed the lowest MSE,RMSE,highest r2_scores



After Evaluating all the models, we trained, Gradient Boosting showing the best model to be used highlighting the lowest MSE, RMSE values and the highest R2 score

Visualizing The Learning Curve of the Gradient Boosting Regressor model for confirming the final selection of the model.



By analyzing the Learning Curve of the Gradient Boosting model, and since we have small training examples it is shown that most of the models performed badly on the validation set, where it indicates underfitting, the learning curve analysis suggest that having more training examples would result in better performance on both sets for the model. Therefore, we can finally select the model as our final choice and using it for predicting test data.

Conclusion

MILESTONE 1 CONCLUSION:

- The dataset is small and contains a significant number of errors, which we try to fix it through our intensive analysis and preprocessing.

- Providing more data can significantly enhance our model performance, enabling it to better generalize and make accurate predictions on new data.

- After experimenting with multiple models, gradient boosting emerges as the optimal choice.

[MS2] Hyperparameter Tuning and Classification

Problem Definition

The objective of Milestone 2 is to develop a predictive classification model to categorize medical services based on the dataset attributes. This milestone involves the choice of a predictive classification model through extensive model selection and tuning processes. To achieve the objective of categorizing medical services accurately, Milestone 2 involves a series of essential steps.

1. **Feature Encoding:** Encode categorical variables into numerical representations suitable for machine learning algorithms. Utilize techniques such as one-hot encoding or label encoding to transform categorical attributes into a format understandable by regression models.
2. **Feature Selection:** Identify and select relevant features that contribute most to predicting fee prices.Using techniques such as correlation analysis or feature importance ranking.
3. **Model Building:** Experimenting with multiple classification algorithms through extensive model selection, tuning, and evaluation processes to identify the best-performing model.
4. **Model Selection:** After tuning and evaluating the models, selecting the best-performing classification model based on its performance on the dataset. Subsequently, training the chosen model and assessing its performance on test data to analyze and gain insights from the results.

Features Encoding

i. **Fee Category** column: mapping the categories in order as they inherit importance, giving Expensive category higher value next medium-priced and cheap the lowest value

Fee Category

```
[ ] data['Fee Category'].unique()
```

```
array(['Expensive', 'Cheap', 'Medium-Priced'], dtype=object)
```

- Encoding the Traget variable using ordinal mapping due to the order it inherits between the categories if fee as [Expensive,Cheap,Medium-Priced] where giving expensive fees higher values.

```
fee_categories_mapping = {'Cheap': 0,
                          'Medium-Priced': 1,
                          'Expensive': 2}

train_data['Fee Category'] = train_data['Fee Category'].map(fee_categories_mapping)
val_data['Fee Category'] = val_data['Fee Category'].map(fee_categories_mapping)
test_data['Fee Category'] = test_data['Fee Category'].map(fee_categories_mapping)
```

ii. **Specialization** column: Considering the nature of the column where different specializations may have different fee categories, with some specializations being more expensive and others being cheaper, we choose to map each category to the **mode** of the fee category present.

Specialization

```
[ ] train_data['Specialization'].nunique()
```

```
127
```

Given that the column "**Specialization**" has 136 unique values, which is still a large number, and considering the nature of the column where different specializations may have different fee categories, with some specializations being more expensive and others being cheaper, we choose to map each category to the **mode** of the fee category present.

```
mode_per_category = train_data.groupby('Specialization')['Fee Category'].agg(lambda x

train_data['Specialization'] = train_data['Specialization'].map(mode_per_category)
val_data['Specialization'] = val_data['Specialization'].map(mode_per_category).fillna
test_data['Specialization'] = test_data['Specialization'].map(mode_per_category).fill
```

```
[ ] train_data['Specialization'].value_counts()
```

```
Specialization
1    983
0    547
2     88
Name: count, dtype: int64
```

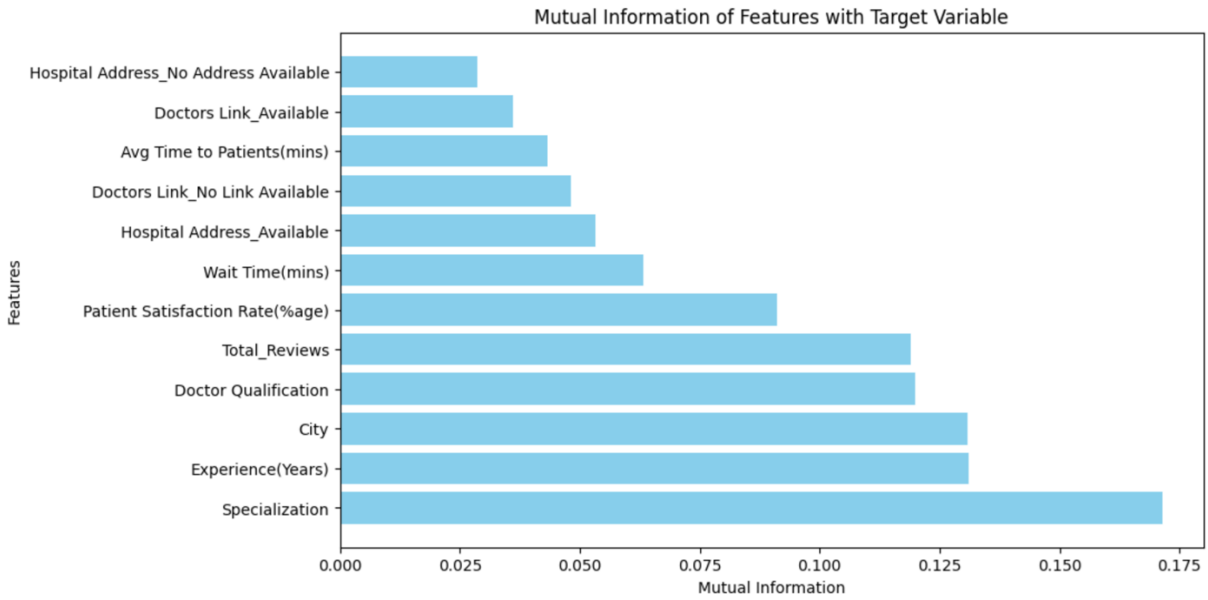
iii. **City** column: considering the nature of the column where cities get affected by fee categories, with some places being more expensive and others being cheaper, we choose to map each category to the mode of the Fee category present.

Features Selection

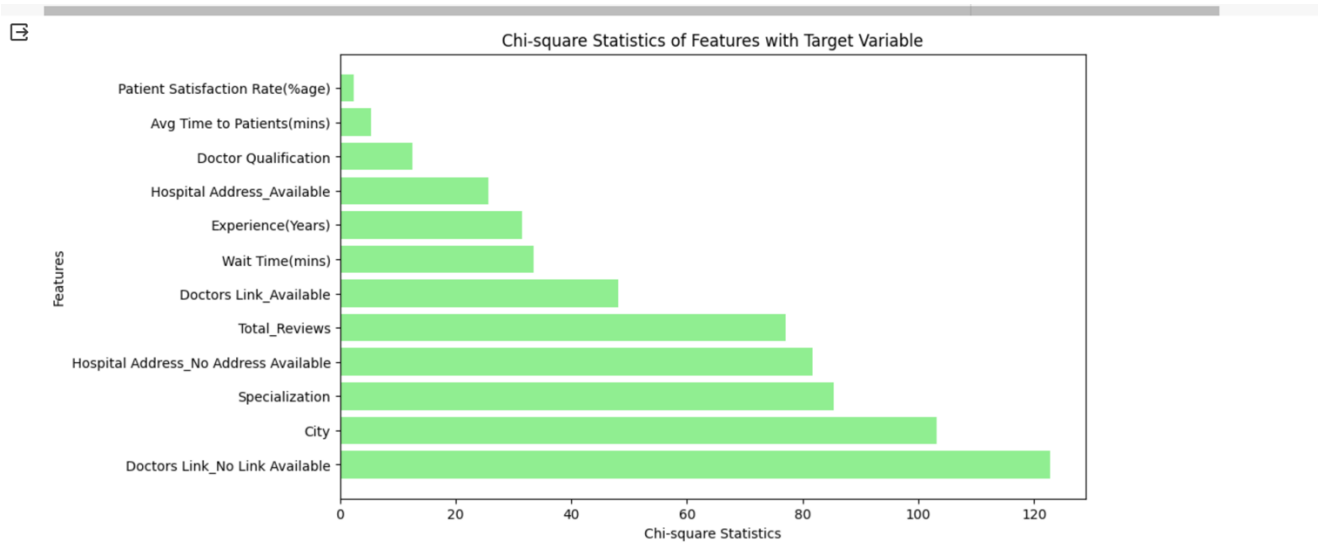
For classification tasks, feature selection techniques changed, we chose to apply

a) CHI Squared statistics.
b) Mutual Information.
c) Fisher Scores

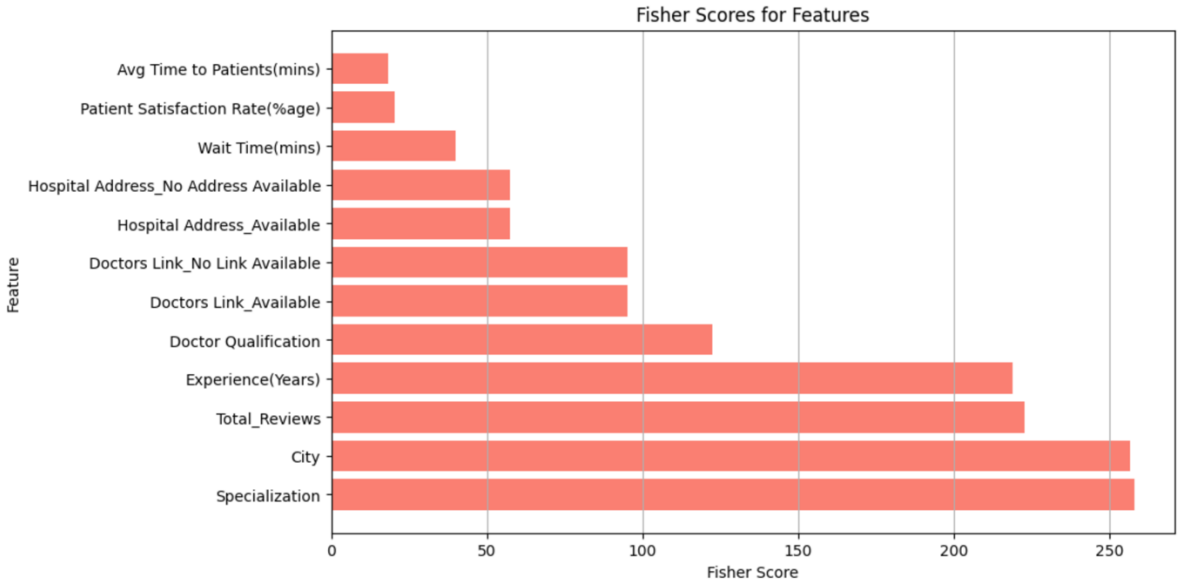
➢ Mutual Information



➢ CHI Squared



➢ Fisher Score



Post analyzing feature selection methods used and identifying the features that contribute mostly to prediction of the target, we selected features that commonly have high scores between at least 2 or 3 of the 3 methods used for selection.

Final Selected features are:

[City, Specialization, Doctor Qualification, Experience (Years), Total_Reviews, Doctors Link]

Modeling

Experimented with various classification tasks and techniques and extensive hyper parameters tuning for models to accurately predict **Fee category**, and choose the model with the best performance on the data

- **Logistic Regression**
- **Tree-based Models (e.g., Random Forest, XGBoost)**
- **KNN**
- **SVM**
- **Ensemble Learning Techniques (e.g., Voting, Stacking)**

1. Logistic Regression

Trained logistic regression model by assigning it **class_weight** parameter to reduce misclassifications encountered for the "**Expensive**" class.

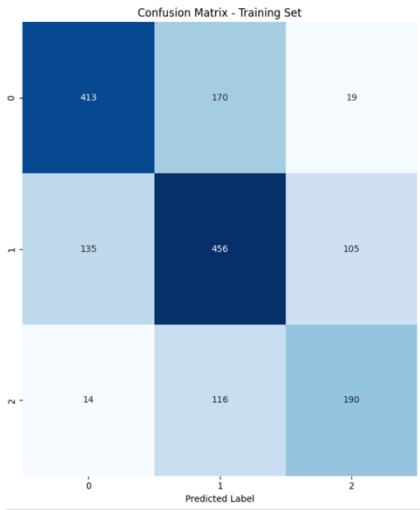
```
print("Evaluating Training Metrics :\n")

print(f"Training Accuracy of Logistic Regression : {accuracy_score(y_train,y_pred_tra
print(f"Training F1 Score of Logistic Regression: {f1_score(y_train,y_pred_train_logi
```

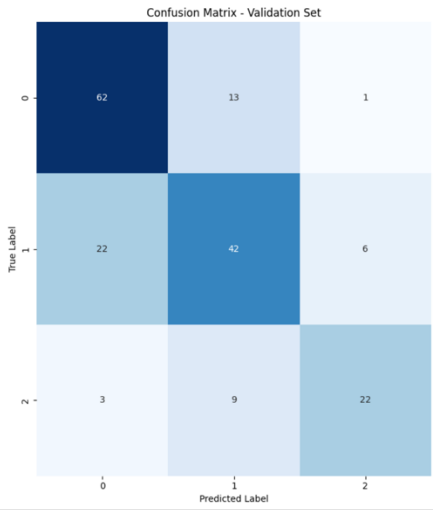
```
Evaluating Training Metrics :

Training Accuracy of Logistic Regression : 65.45
Training F1 Score of Logistic Regression: 65.54
```

```
print("Evaluating Validation Metrics :\n")

print(f"Validation Accuracy of Logistic Regression : {accuracy_score(y_val,y_pred_log
print(f"Validation F1 Score of Logistic Regression: {f1_score(y_val,y_pred_logistic,a

models_eval.loc[len(models_eval)]=['Logistic Regression',(f1_score(y_val,y_pred_logis
```

```
Evaluating Validation Metrics :

Validation Accuracy of Logistic Regression : 70.00
Validation F1 Score of Logistic Regression: 69.69
```

Logistic regression achieving **70%** accuracy, with no signs of overfitting, showing an acceptable performance with **0.69** F1-score, which tells model performed somehow good on training and validation datasets.





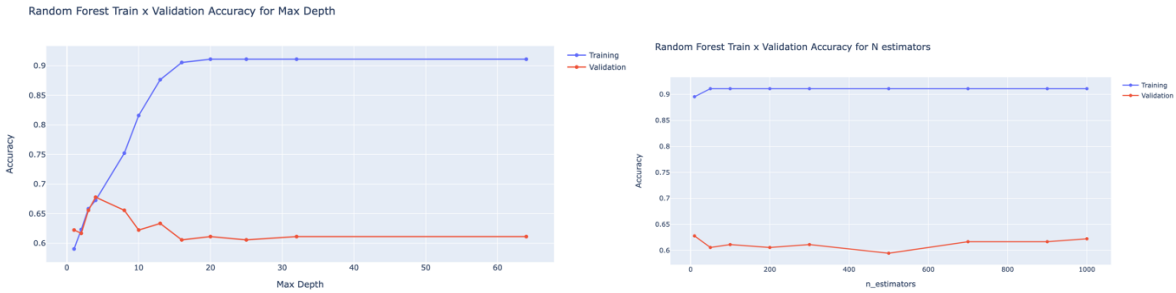
```
Classification Report :
              precision    recall  f1-score   support

           0       0.71      0.82      0.76        76
           1       0.66      0.60      0.63        70
           2       0.76      0.65      0.70        34

    accuracy                           0.70       180
   macro avg       0.71      0.69      0.70       180
weighted avg       0.70      0.70      0.70       180
```

As observed by analyzing the confusion matrix and the classification report for the 3 classes we have, we find that logistic regression finds balance between the 3 classes precision, recall and getting almost a good proportion of each classes examples after experimenting with different class weights values, which suggests that the model has acceptable and somehow reliable performance.

2. Random Forest Classifier

Pre-Pruning and Hyper parameter tuning for the random forest hyperparameters for preventing overfitting and choosing the best values for better accuracy.

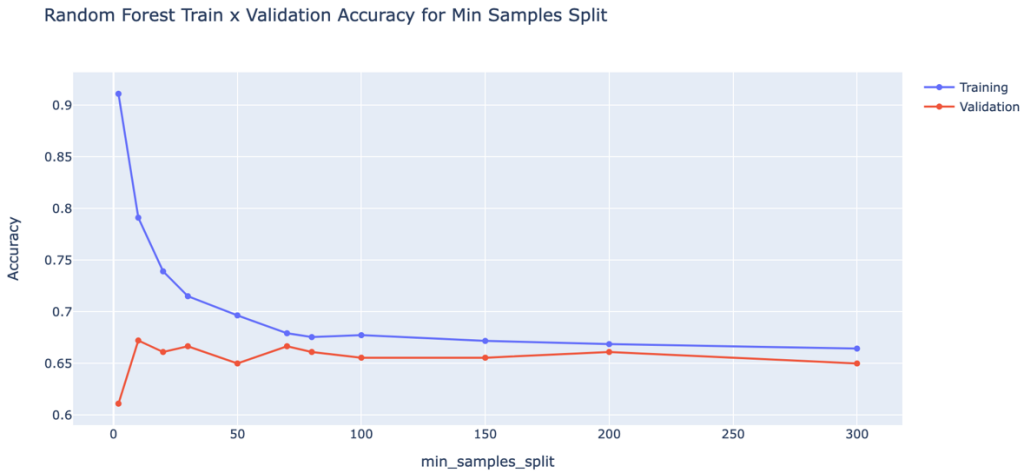
Choosing the values for (**max_depth, n_estimators**), that reduces the gap between training and validation accuracies, and gaining highest validation accuracy:



Choosing (max_depth = 4, n_estimators=700), and experimenting with different values for min_samples_split to choose best accuracy

min_samples_split	Accuracy	F1-score
200	67.22	67.01
300	67.78	67.5
50	67.2	66.9

Choosing the values for (**min_samples_split)** that reduces the gap between training and validation accuracies, and gaining highest validation accuracy:



Choosing (min_samples_split = 200), and experimenting with different values for max_depth to choose best accuracy:

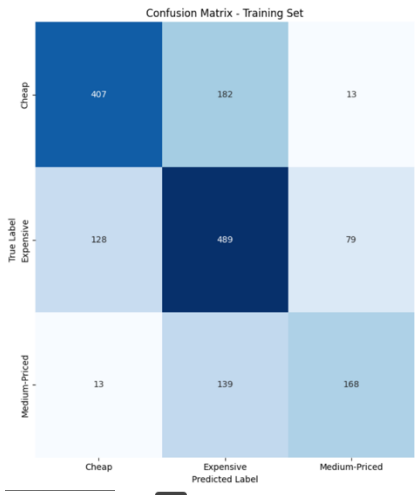
max_depth	Accuracy	F1-score
4	67.22	67.01
8	66.01	65.5
11	66.01	66.02

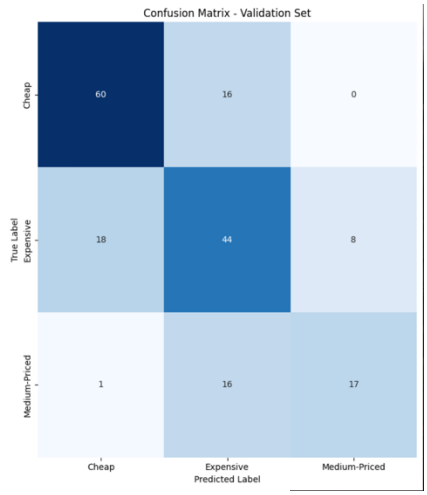
Final selected values:

```
random_forest_model = RandomForestClassifier(n_estimators = 700,
                                              min_samples_split = 200,
                                              max_depth = 4,
                                              class_weight=class_weights,
                                              random_state=42)

random_forest_model.fit(X_train, y_train)
```

```
[ ] print("Evaluating Training Metrics :\n")

    print(f"Training Accuracy of Random Forest : {accuracy_score(y_train,y_pred_rf_train)
    print(f"Training F1 Score of Random Forest : {f1_score(y_train,y_pred_rf_train,averag

    Evaluating Training Metrics :

    Training Accuracy of Random Forest : 65.76
    Training F1 Score of Random Forest : 65.73

[ ] print("Evaluating Validation Metrics :\n")

    print(f"Validation Accuracy of Random Forest : {accuracy_score(y_val,y_pred_rf)*100:.
    print(f"Validation F1 Score of Random Forest : {f1_score(y_val,y_pred_rf,average='wei

    models_eval.loc[len(models_eval)]=['Random Forest',(f1_score(y_val,y_pred_rf,average=

    Evaluating Validation Metrics :

    Validation Accuracy of Random Forest : 67.22
    Validation F1 Score of Random Forest : 67.01
```

Lower validation accuracy compared to logistic regression achieving **67%** accuracy post hyper-parameter tuning, but with no signs of overfitting.





Classification Report for Validation :

	precision	recall	f1-score	support
0	0.76	0.79	0.77	76
1	0.58	0.63	0.60	70
2	0.68	0.50	0.58	34
accuracy			0.67	180
macro avg	0.67	0.64	0.65	180
weighted avg	0.67	0.67	0.67	180

As observed by analyzing the confusion matrix and the classification report for the 3 classes we have, we find that random forest still finds balance between the 3 classes precision, recall and getting almost a good proportion of each classes examples after experimenting with different class weights values and hyper parameter tuning for the rest of parameters, suggesting that the model has also a good and acceptable performance.

3. Support Vector Machine

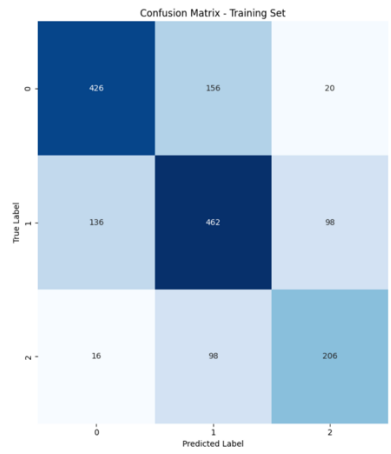
Tuning SVM hyperparameters using grid search to choose best combinations for the values of C, kernel, degree, and class weights that maximizes the accuracy:

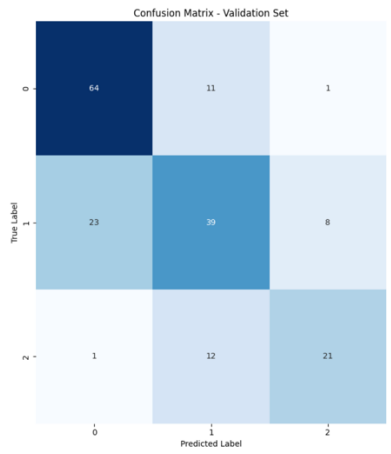
```
param_dist = {
    'C': [0.1, 0.5,0.9 ,1.0,2.5,3,5,25],
    'kernel': ['linear', 'rbf', 'poly','sigmoid'],
    'degree': [1, 3, 5,7],
    'class_weight': [{0: 1, 1: 2,2:3}, {0: 3, 1: 3.5,2:5}, {0: 2.5, 1: 3,2:4.5}, {0:
}
svc = SVC(random_state = 42)
grid_search = GridSearchCV(estimator=svc,
                           param_grid=param_dist,
                           cv=5, scoring='accuracy',
                           verbose=3)

grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score (Accuracy):", best_score)
```

```
Best Parameters: {'C': 0.9, 'class_weight': {0: 3, 1: 3.5, 2: 5}, 'degree': 1, 'kernel': 'rbf'}
Best Score (Accuracy): 0.6594580132247831
```

Choosing the values of the hyperparameters obtained by grid search, and experimenting with different values for C again to enhance the scored accuracy

C	Accuracy	F1-score
0.9	67.78	67.30
3	68.33	67.80
50	69.44	68.75





```
Classification Report for Validation :
                precision    recall  f1-score   support

             0       0.73      0.84      0.78        76
             1       0.63      0.56      0.59        70
             2       0.70      0.62      0.66        34

      accuracy                           0.69       180
     macro avg       0.69      0.67      0.68       180
  weighted avg       0.68      0.69      0.68       180
```

As observed by analyzing the confusion matrix and the classification report for the 3 classes we have, we find that SVM finds a good balance between the 3 classes precision, recall and getting almost a good proportion of each classes examples after experimenting with different class weights values and the C parameter and hyper parameter tuning for the rest of parameters, suggesting that the model has also a good and acceptable performance and very close to logistic regression and random forest model.

Conclusion:

All models, almost got the same performance on both training and validations sets after hyperparameters tuning, with no signs of overfitting with maximum accuracy 70%, we will explore with further techniques like ensemble learning stacking and voting to improve accuracy and predictions and focus on the weaknesses of each model after combining them together

4. Ensemble Learning

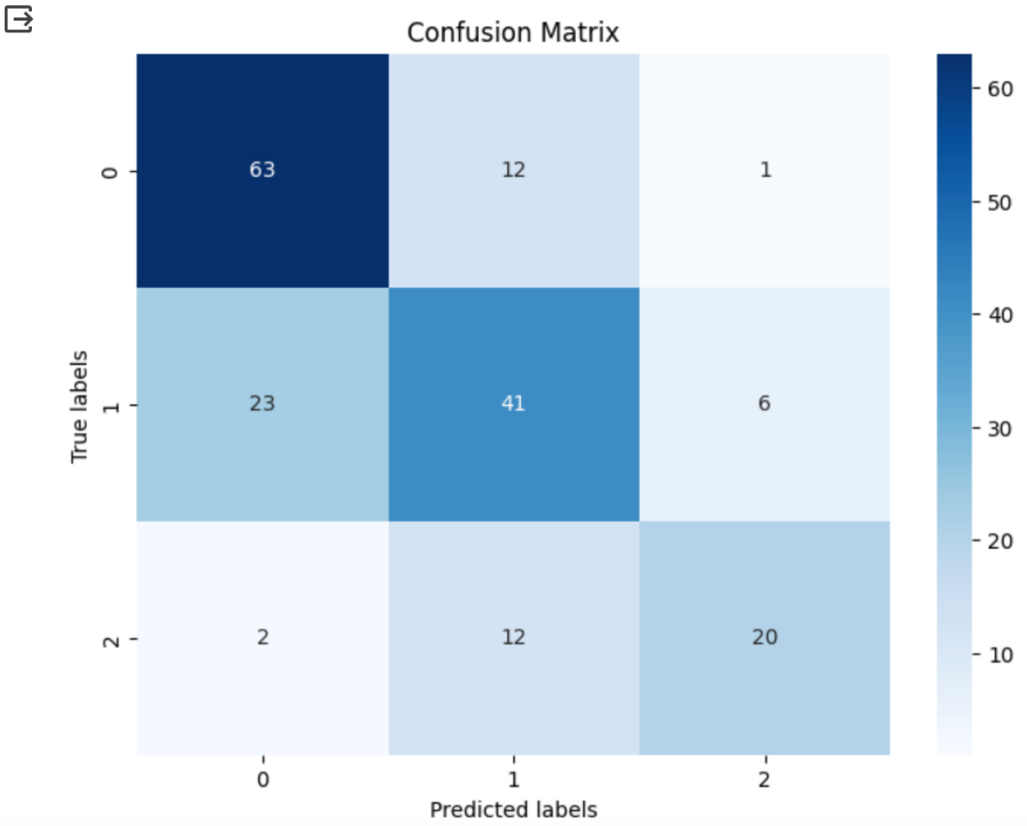
4.1 Voting:

Created a voting classifier between **[Logistic Regression, Random Forest, SVM]** to enhance their accuracy by combining them together and getting the final predictions based on the max voting between them

```
voting_clf.fit(X_train,y_train)
```

```
VotingClassifier
Logistic Regression | Random Forest | SVM
LogisticRegression | RandomForestClassifier | SVC
```

```
Evaluating Validation Metrics :

Validation Accuracy of Voting Classifier : 68.89
Validation F1 Score of Voting Classifier : 68.45
```

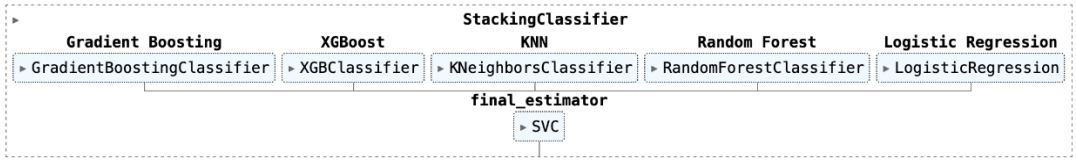


Almost got an average validation accuracy between all the 3 models' accuracies, with almost same predictions of the 3 models, providing no enhancement done when combining them together, with still logistic regression being the one having maximum validation accuracy

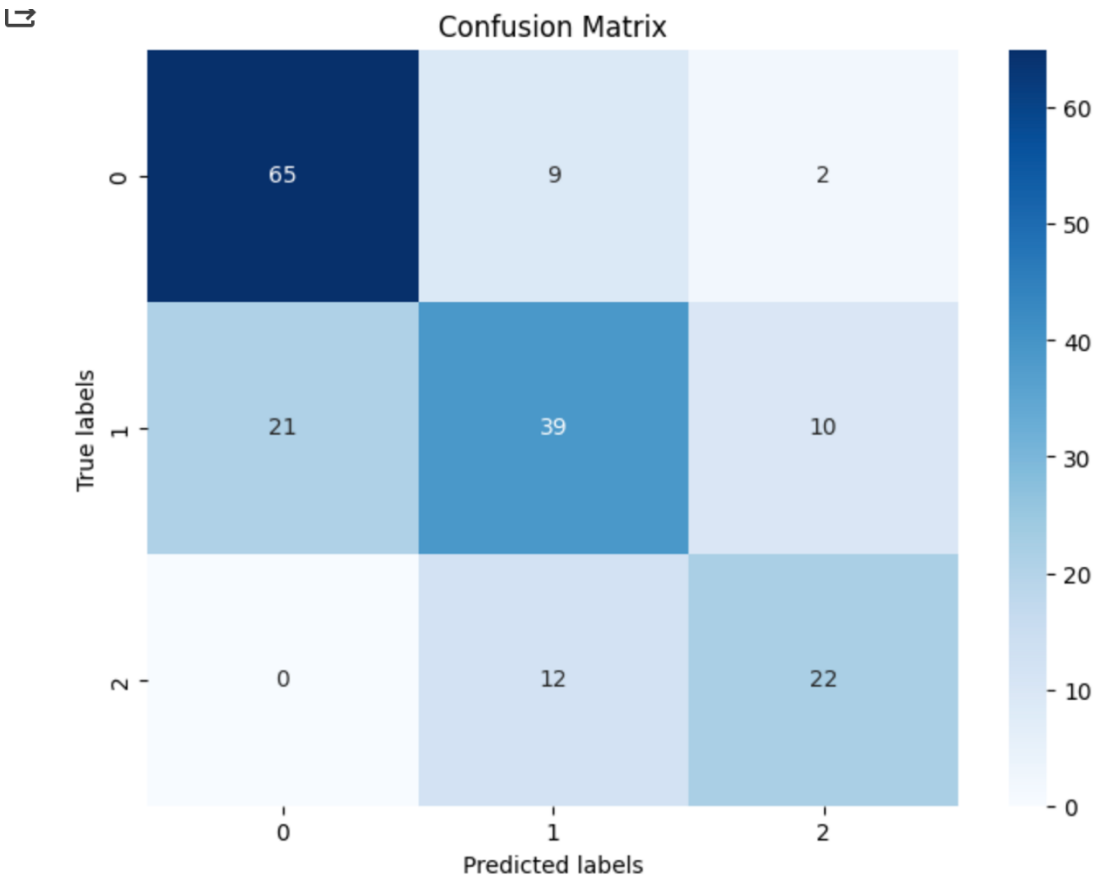
4.2 Stacking:

Created a stacking classifier between most of the trained models **[Logistic Regression, Random Forest, KNN, XGBoost, Gradient Boosting]** to enhance their accuracy by combining them together and getting the final predictions based with the **SVM**

```
clf.fit(X_train,y_train)
```




```
Evaluating Validation Metrics :

Validation Accuracy of Stacking Classifier: 70.00
Validation F1 Score of Stacking Classifier: 69.44
```



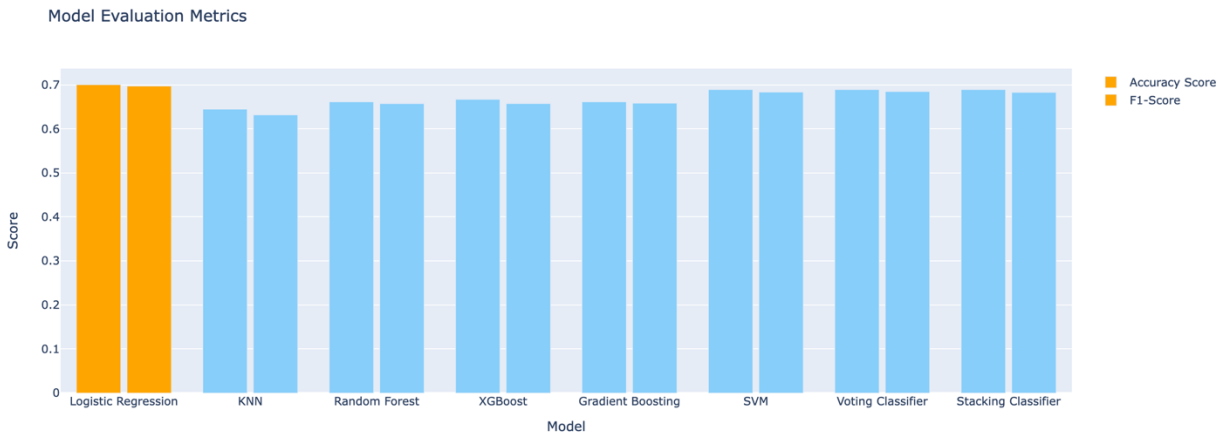
More enhanced accuracy compared to all models, and the voting classifier with the same accuracy of logistic regression and almost same predictions and misclassifications, suggesting no further enhancement can be done on the data for achieving higher accuracy.

Final Evaluation and Model Selection

1. Accuracy Evaluation

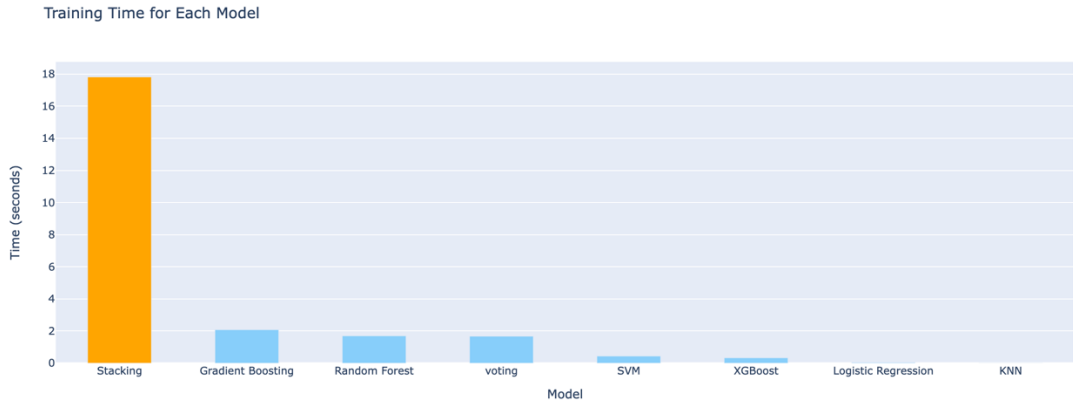
```
models_eval
```

	Model	F1 score	Accuracy
0	Logistic Regression	69.690322	70.000000
1	KNN	63.139891	64.444444
2	Random Forest	65.702349	66.111111
3	XGBoost	65.712366	66.666667
4	Gradient Boosting	65.796437	66.111111
5	SVM	68.329561	68.888889
6	Voting Classifier	68.446580	68.888889
8	Stacking Classifier	68.255860	68.888889



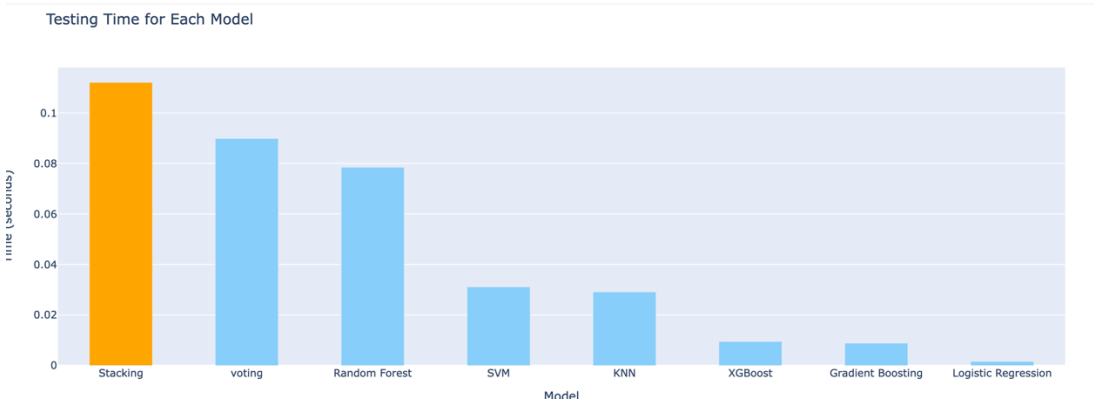
Logistic Regression performs as the best models through the model selection and enhancement process, achieving highest validation accuracy and F1-score, balancing between all the classes metrics with no signs of overfitting and no bias to certain class post adjustment

2. Total Training Time Evaluation



Stacking classifier takes the most time for training and fitting about 18 seconds since it has many estimators' models that need training and fitting the output of the estimators to the final estimator, Logistic regression and KNN almost take no time for training.

3. Total Testing Time Evaluation

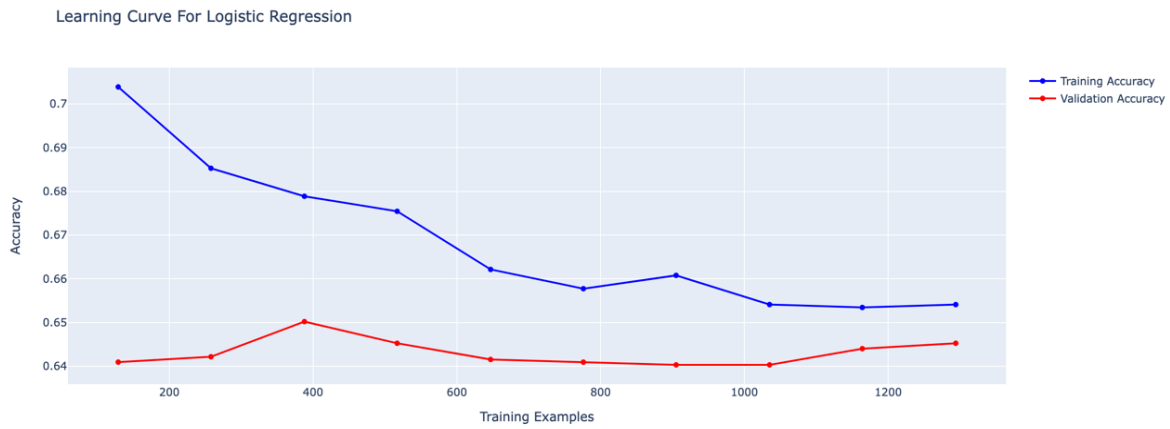


Stacking classifier as well the most time for testing and predicting about 0.2 seconds compared to the other models, which take very small time for testing.

Final selected model:

[Logistic Regression]

Visualizing The Learning Curve of the **Logistic Regression** model



After selecting logistic regression as the final model, due to its highest accuracy compared to other models, the learning curve indicates no signs of overfitting,as the gap between the training and validation curves decreases as more training examples are provided. However, the overall model appears to suffer from underfitting, as there is no significant improvement observed with additional training examples. This suggests that the data itself may not be very informative, and the model may struggle to identify better patterns and this the best performance it can achieve. It is likely that different features are needed to improve model performance, along with the possibility of requiring additional training examples to enhance the model's learning capability.

Finally, training the **Logistic Regression** model on combined training and validation set, and using it for predicting test set

```
[1498] X_train_final = np.concatenate((X_train, X_val), axis=0)

       y_train_final = np.concatenate((y_train, y_val), axis=0)
```

```
[1500] logistic_regression.fit(X_train_final,y_train_final)

       LogisticRegression
       LogisticRegression(class_weight={0: 2, 1: 2, 2: 3})

       predictions = logistic_regression.predict(X_test)

       print("Evaluating Test Metrics :\n")

       print(f"Test Accuracy  : {accuracy_score(y_test,predictions)*100:.2f}")
       print(f"Test F1 Score : {f1_score(y_test,predictions,average='weighted')*100:.2f}")

       Evaluating Test Metrics :

       Test Accuracy  : 57.33
       Test F1 Score : 57.43
```

Logistic Regression achieved **57%** accuracy on the test after training and tuning.

Conclusion

MILESTONE 2 CONCLUSION:

- The dataset is small and contains a significant number of errors, which we try to fix it through our intensive analysis and preprocessing.
- The dataset lacks substantial information, making it challenging for models to identify significant patterns. Alternative features may be required to improve model performance.
- With extensive tuning for all models, none of the classification models were able to attain a validation accuracy exceeding 70%, with no noticeable reduction in misclassifications.

Credits:

- Mohamed samy
- Yomna Mohamed
- Ammar Mohamed
- Nadine Haitham
- Mohamed Ashraf
- Youssef Tamer