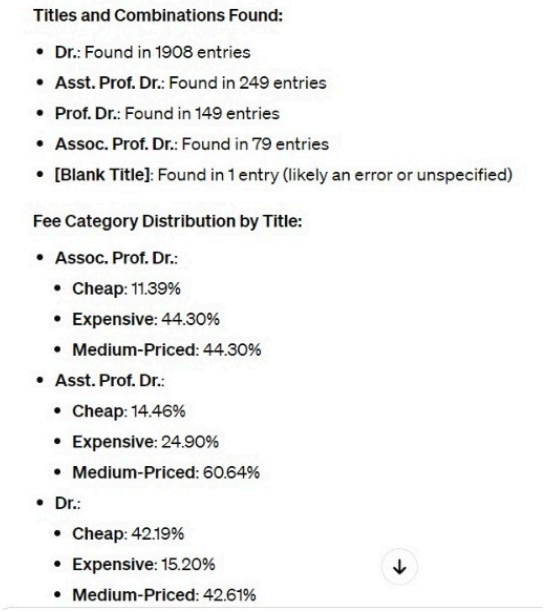
Machine Learning
Milestone 2

Updated Encoding since milestone 1 :

1- Based on our analysis for Doctors Name column & their title :



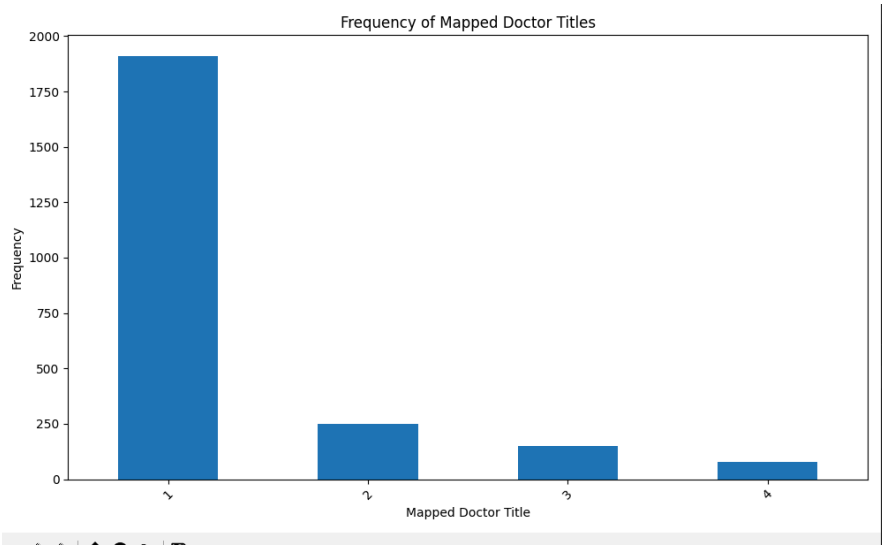
Titles and Combinations Found:

- **Dr.**: Found in 1908 entries
- **Asst. Prof. Dr.**: Found in 249 entries
- **Prof. Dr.**: Found in 149 entries
- **Assoc. Prof. Dr.**: Found in 79 entries
- **[Blank Title]**: Found in 1 entry (likely an error or unspecified)

Fee Category Distribution by Title:

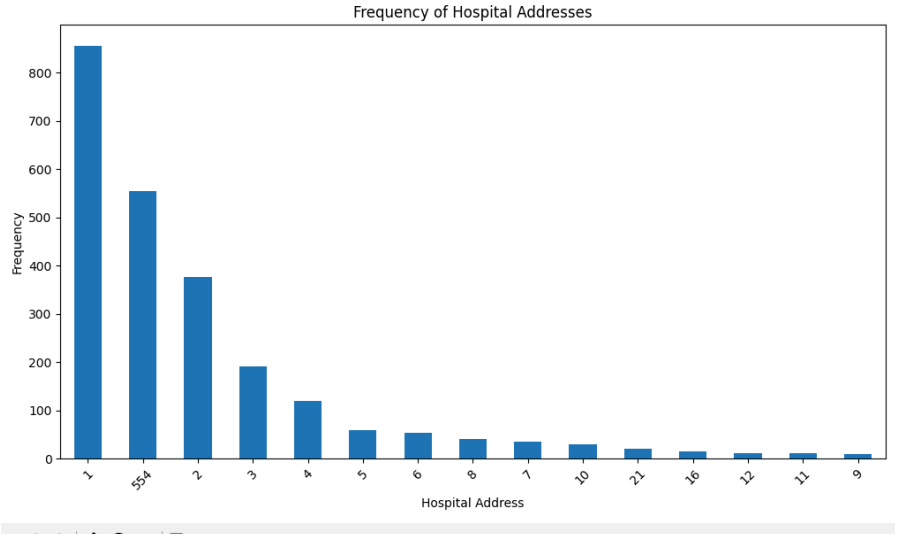
- **Assoc. Prof. Dr.**:
  - **Cheap**: 11.39%
  - **Expensive**: 44.30%
  - **Medium-Priced**: 44.30%
- **Asst. Prof. Dr.**:
  - **Cheap**: 14.46%
  - **Expensive**: 24.90%
  - **Medium-Priced**: 60.64%
- **Dr.**:
  - **Cheap**: 42.19%
  - **Expensive**: 15.20%
  - **Medium-Priced**: 42.61%

We found that we could benefit from this & we did Ordinal encoding in which we encoded every name by their title.



2- Hospital Address :

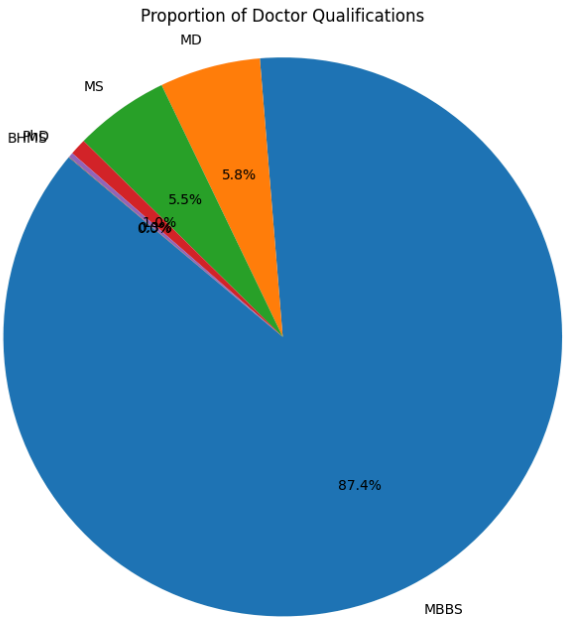
Rather than just encoding it if it has a link encode it to 0 otherwise we did [frequency encoding] which gives each unique value a number then it replaces each category with frequency of it



3-Doctor’s Qualification

With our analysis it showed that one hot encoding results in 1000+ extra columns and most of them with very low frequency.

The approach involves using regular expressions to identify common medical degrees such as MBBS, BDS, BAMS, BHMS, MD, MS, MDS, DNB, and PhD within the "Doctor Qualification" string. These degrees are then extracted and sorted to remove duplicates, and finally joined into a single string.



We simplified the data based on most recurring qualifications

4-City Column

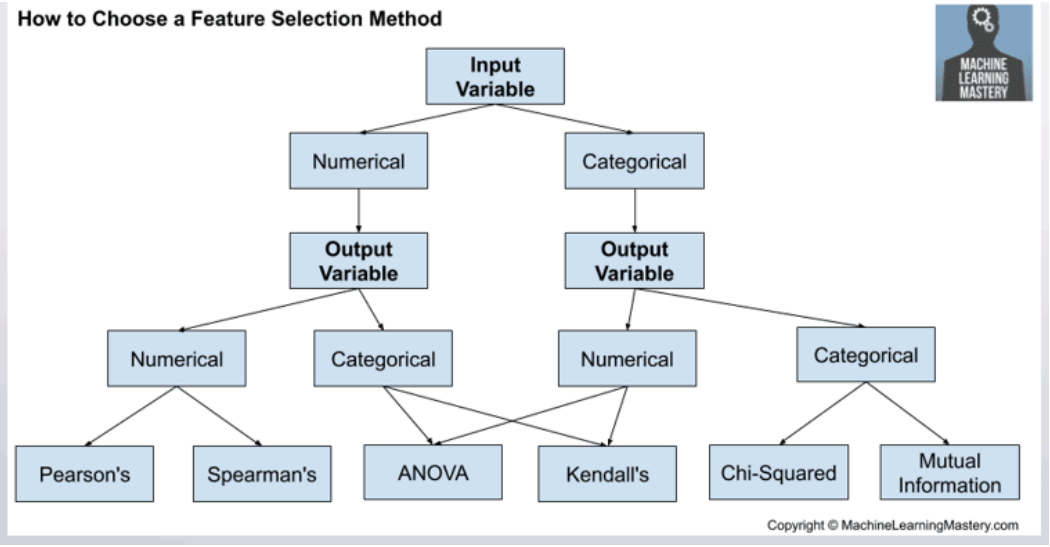
With further analysis we figured out the following:

There are a total of 130 unique cities in the dataset.
Majority in pakistan , 2 cities in turkey , 1 in saudi arabia

So we will use [hash encoding]
Hash encoding can handle new categories that were not present during training, as long as the hashing function remains the same. This makes it suitable for scenarios where the set of categories may change over time.

Any other encoding stayed roughly the same as milestone 1

Feature Selection



We used this as a guideline on how we are going to choose and we also used many other techniques like SelectFromModel , RFE , Lasso & Wrapper Methods.

We Used Pipelining to Organise Feature selection methods.

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline([
            ('imputer', KNNImputer(n_neighbors=5)),
            ('scaler', StandardScaler()),
            ('anova', SelectKBest(score_func=f_classif, k=4))
        ]), ['Experience(Years)', 'Total_Reviews', 'Patient Satisfaction Rate(%age)', 'Avg Time to Patients(mins)',
             'Wait Time(mins)']),
        ('cat', Pipeline([
            ('encoder', OneHotEncoder(handle_unknown='ignore')),
            ('chi2', SelectKBest(score_func=chi2, k=4))
        ]), ['City', 'Specialization', 'Doctor Qualification', 'Hospital Address', 'Doctor Name', 'Doctors Link'])
    ],
    remainder='drop'
)
```

We also used pipelining to ease hyperparameter tuning.

```
models = {
    'RandomForest': {
        'model': RandomForestClassifier(random_state=42),
        'params': {'n_estimators': [100, 200, 300], 'max_depth': [10, 15, 20], 'min_samples_split': [2, 5],
                   'min_samples_leaf': [1, 2]}
    },
    'SVC': {
        'model': SVC(kernel='linear', random_state=42),
        'params': {'C': [0.001, 0.01, 0.1, 1, 10, 100], 'kernel': ['linear', 'rbf'], 'gamma': ['scale', 'auto']}
    }
}
```

```
gb_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('feature_selection', SelectFromModel(estimator=RandomForestClassifier(random_state=42))),
    ('classifier', GridSearchCV(GradientBoostingClassifier(random_state=42),
                                param_grid={'n_estimators': [50, 100, 150],
                                            'learning_rate': [0.01, 0.05, 0.1],
                                            'max_depth': [3, 4]},
                                cv=5,
                                scoring='accuracy'))
])
```

Classification Models

Chosen Models & hyperparameter tuning:

1- Random Forest

Increasing max_depth allows the model to learn more detailed data specifics, which can improve training accuracy but might lead to overfitting, affecting the model's generalisation ability negatively if the depth is too high.

1st variation:max depth = 15

```
rf_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(max_depth=15, min_samples_leaf=2, min_samples_split=5, n_estimators=300, random_state=42))
])
```

```
Random Forest - Training Accuracy: 0.7867, Testing Accuracy: 0.5962
```

2nd variation:max depth = 20

```
models = {
    'RandomForest': {
        'model': RandomForestClassifier(max_depth=20, min_samples_leaf=2, min_samples_split=5, n_estimators=300, random_state=42),
```

```
RandomForest - Training Accuracy: 0.7505, Testing Accuracy: 0.6025
```

3rd variation: max depth = 10

```
models = {
    'RandomForest': {
        'model': RandomForestClassifier(max_depth=10, min_samples_leaf=2, min_samples_split=5, n_estimators=300, random_state=42),
```

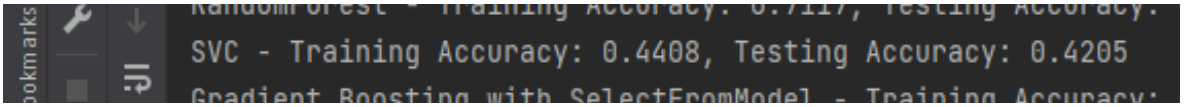
```
C:\Users\HP\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\
RandomForest - Training Accuracy: 0.7117, Testing Accuracy: 0.6004
```

Observation : as we can see we have reduced overfitting

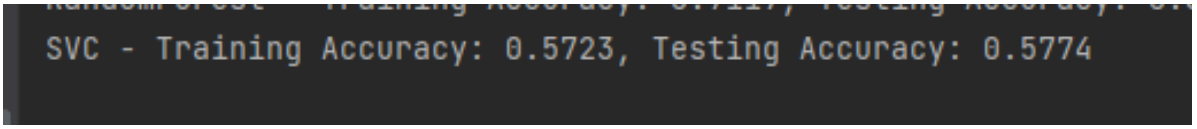
2-SVC

The C parameter trades off correct classification of training examples against maximisation of the decision function's margin. A higher C aims to classify all training examples correctly (low bias but potentially high variance), while a lower C encourages a larger margin and simpler decision boundary (higher bias but lower variance).

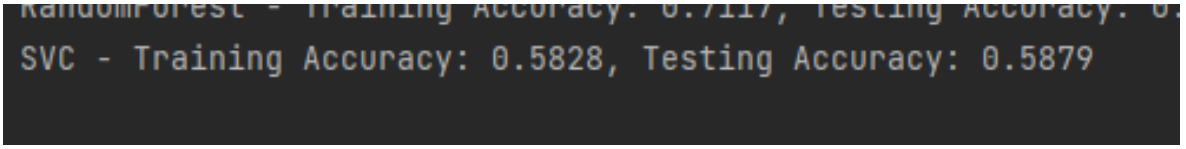
1st variation: C = 0.001



2nd variation: C = 0.01



3rd variation: C = 0.1



Observation: both accuracies increased gradually

3-Gradient Boosting

The learning_rate shrinks the contribution of each tree by the factor specified. A higher learning rate can lead to faster learning, but also to overfitting. A lower rate requires more trees to model all interactions but can generalise better.

1st variation: LearningRate = 0.1

```
# Additional Gradient Boosting Pipeline with Feature Selection
gb_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('feature_selection', SelectFromModel(estimator=RandomForestClassifier(random_state=42))),
    ('classifier', GridSearchCV(GradientBoostingClassifier(random_state=42),
                                param_grid={'n_estimators': [50, 100, 150],
                                            'learning_rate': [0.01],
                                            'max_depth': [3, 4]},
                                cv=5,
                                scoring='accuracy'))
])
```

```
Gradient Boosting with SelectFromModel - Training Accuracy: 0.6012, Testing Accuracy: 0.6025
```

2nd variation: LearningRate = 0.05

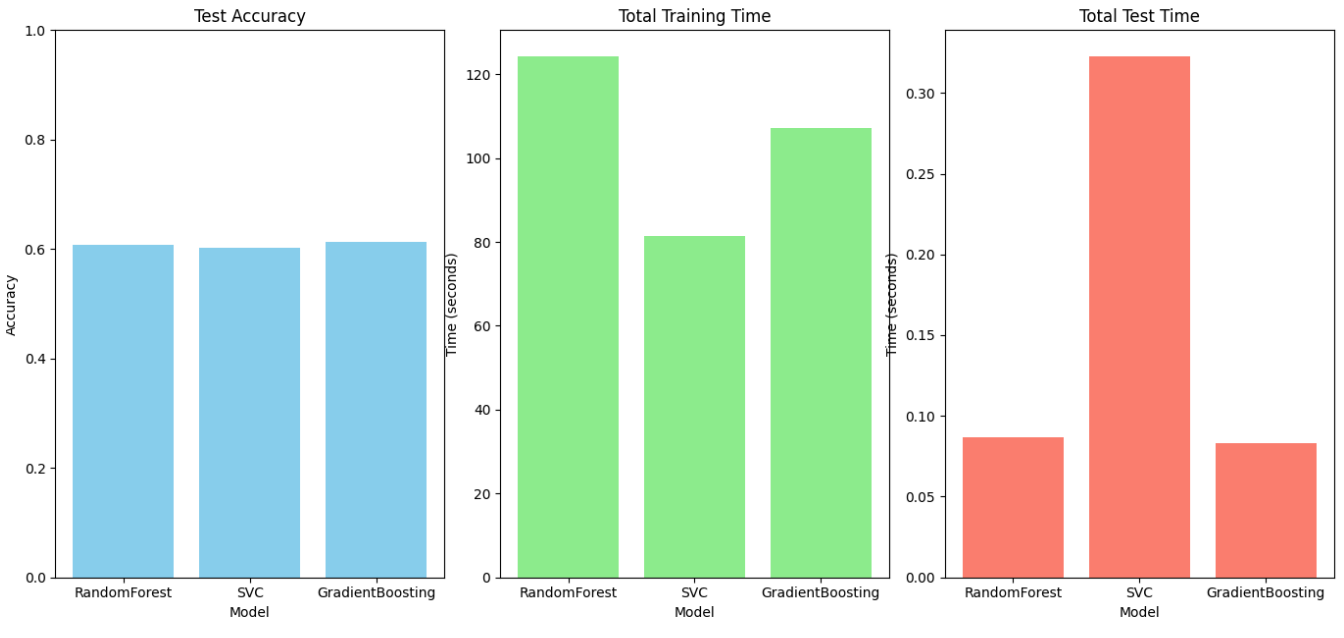
```
Gradient Boosting with SelectFromModel - Training Accuracy: 0.6357, Testing Accuracy: 0.6004
```

3rd variation: LearningRate = 0.01

```
Gradient Boosting with SelectFromModel - Training Accuracy: 0.6378, Testing Accuracy: 0.6004
```

Graphs

The following plot shows bar graphs for comparing test accuracies , training & test time:



Conclusion

In this phase, we successfully applied machine learning algorithms to classify doctor fees into distinct categories. We observed the impact of hyperparameter tuning on model performance and explored the trade-offs between accuracy and computational efficiency. Overall, our intuition about the problem was validated through rigorous experimentation and analysis, demonstrating the effectiveness of machine learning in addressing real-world tasks like doctor fee prediction.we corrected many things we haven't considered in milestone 1 like test split and hyperparameter tuning and the difference between handling data for regression & classification model , and writing test script teaches us many things to generalise data handling , saving & loading models and many other things.