**Real estate price prediction and place recommendation project**

**EDA**:

Society:

There are many categories which are only one in count (so this is a problem with this column).

Out of 675, 75 societies have 50% of properties.

Sector:

Here the problem is it has high cardinality.

There are 104 unique sectors

Price:

Here the data is right skewed, around 5% of this column could be considered as outlier but we can not remove these many rows.

So applied log-transformation to bring it near to normal distribution so that we could apply linear models.

Price\_per\_sqft:

This column has some outliers on the higher side and also on the lower side, higher side is like 2 lac to 6 lac price per sqft and on lower 4 rupees per\_sqft which is clearly an outlier.

Bathroom and bedroom are sort of similar, because using pie-chart we can see that 2/3 bedrooms and bathrooms are more in number and above that the percentage starts to decrease, could have multi-collinearity issue.

Built\_up\_area 🡪 all the rooms + balconies + thickness of the wall

Carpet\_area 🡪 exclude balconies and thickness of the wall from built\_up\_area

Super\_built\_up\_area 🡪 In flats, the staircase area, lift area (which is common for all)

**Outlier detection**:

There are two methods by which we could identify which technique to use for outlier detection.

If Normal distribution: we can use Standard deviation and remove the points which fall in 2nd, 3rd standard deviation.

If not normal distribution: we can use technique like IQR (box plots) to see which points are less than lower bound and which are bigger than high bound.

In price column there are genuine price, which cannot be removed directly.

In price\_per\_sqft column 🡪 rows below 1000 are in sq\_yard and above are in sq\_ft. So, converted them to sq\_ft.

Removed the values which are above 50000 sq\_ft and improved the distribution of the data.

Area column had some rows where some areas were above 100000 and prices for them were low so considered them as outliers and removed them.

Removed the rows which were having bedrooms above 10.

Some house had area smaller but bedrooms were more than 3/4 which definitely looked like wrong because area and bedroom will have some ratio and these points were not justifying that ratio, ex : 300 sq\_ft/yard having bedrooms.

Points from these:

* Maybe they are not house but are land on which mutli-storey buildings are constructed which have 2/3 floors and have 2 bathrooms on each floor. Maybe the broker labelled them as house by mistake.

**Missing value imputation**

Super\_built\_up\_area, build\_up\_area and carpet\_area is helpful for each other.

Using super\_built\_up\_area and carpet\_area filled some built\_up\_area values, calculated the ration of super\_built\_up and carpet\_area with respect to built\_up\_area.

Floor Number: It had 17 missing values out of which only 2 were flats and rest were houses, generally houses have low floor numbers, as we could have outliers in data, I filled median values to fill the house’s floor number.

Age possession: It does not have missing values, it has undefined values so, imputed the undefined values with mode of sector column and property type column.

Ex: Sector 62 having with respect to age possession having new property as mostly occurring so, it will be filled with this value.

**Feature selection** → we want features which help users find prices based on their preferences and also, we want such features/columns which a user can easily fill.

1] Removed society and price\_per\_sqft features

2] Converted luxury score and floor number features to categorical column

    – As a numeric value user could find it hard to think of a number and enter it, chances are these numbers could be entered randomly.

**Luxury score** converted to 3 categories [Low, medium and high].

Low being score between 0 and 50, medium being between 50 to 100

And above that is high.

Floor number converted to 3 categories [Low, medium and high]

Low being score between 0 and 2, medium being between 3 to 10

And above that is high.

**Model selection** → transformed y\_label as it was right skewed (homoscedasticity)

There are many categorical columns so, I planned to encode them using all some scikit-learn encoders and observe the performance of the model on all of them.

**Ordinal encoding** (generally works good with tree-based models), one hot encoding (generally works good with linear models) and target encoding (generally works good with high dimensional category columns).

Using scikit-learns Column Transformer > scaled the numerical columns and encoded the columns with Ordinal encoding.

Used KFold cross validation of 10 splits.

Used some regressor models and observed the R2\_Score and MAE, here random forest performed well with 82% R2\_Score and MAE of 52 lacs all this with baseline models.

With One Hot Encoding > there are 3 columns which do not have any order so, using one hot encoding on it (assuming it will improve linear models) the downside of OHE is if the column has many categories, then it will increase dimension of our data but we don’t have much data so ignoring this fact at this point. After OHE removing the first column as it could introduce multicollinearity into our data and the inference would be difficult.

As expected, linear regression performance improved.

After fitting multiple models on this data, tree-based models were consistent with extra-trees performing better and also linear models were improved.

To reduce the dimensionality, I used PCA but it did not improve the performance so skipped this approach.

Second approach is Target encoding > what is does > takes all categories one-by-one and finds means value of it against the target encoder.

And it could lead to data leakage and it calculates mean for all value so, use it after splitting the data or use KFold.

With this Linear Regression improved but not much, Extra trees and Random Forest improved to 90% R2\_Score and 44 lac MAE.

**Recommender system**

Doing this on apartments data.

What are recommender systems 🡪 we can call them as ranking systems.

Collaborative filtering: If we have website on which we are taking ratings from the use then we can build a collaborative filtering.

1] User-base filtering: If user 1 and user 2 are similar then we would show them same items

2] Item-based filtering: If user 1 likes item 1 and item 2 is similar to item 1 then we would recommend item 2 to user 1.

Content based: We recommend similar products.

We will make a content-based recommender system in which we would use three columns and based on which we would create three recommender system

**Columns:** Location, price details, top facilities

Approach 🡪 we will make 3 recommender system, as the query will come, we will ask these 3 systems to recommend and we will give each system a weightage based on which we will receive the recommendation.

Why to do this? Let’s say we are told to give recommendation based on price details or should align a little more with price details then we would increase its weightage.

Strategy 🡪 Use text vectorization on top facilities column, now we would use distance measuring method to find how closer vectors are to each other and based on that we will recommend top 5 properties to the user.

For vectorization we will use Tf-idf vectorizer (there are many like BoW, Word2Vec).

For distance we will use cosine similarity because as per research angular distance work good compared to distances such as Euclidean distance.

<https://stats.stackexchange.com/questions/99171/why-is-euclidean-distance-not-a-good-metric-in-high-dimensions>

**Price Details** column has many details and is in nested dictionaries so, we can convert them to string and vectorize them.

Applied one hot encoding on the categories so the data now we have is property having features assigned as 1 and if not then assigned as 0.

Filled the NaN values with 0 and applied scaling to get better results.

**Location advantages**

There are repetition of the landmarks and because of which there is higher dimension which is 1070 landmarks (this problem has not been tackled yet)

Filled NaN values with big values because we cannot fill NaN with 0 because here, we have numeric values which represents distance (KM, meters) so filled with bigger values (because we don’t have any idea about if that place is near) if filled with small number then it would appear as if that place is near to the property.