DATATHON FME 2024 – REPORT CHALLENGE

(restb.ai: AVM on the State of Illinois)

1. Exploratory Data Analysis

In this step, which is the most crucial one when dealing with this type of projects, we will integrate processes such as data cleaning, data integration, data selection and data transformation.

1.1. Dropping Unnecessary Features (in order of appearance)

- Characteristics.LotFeatures: taking into account this variable could lead to data inconsistency due to the difficulty to transform to numerical form given the high amount of NaN values, which is 60%.
- Characteristics.LotSizeSquareFeet: we agreed to drop this feature since 98% of the values are missing. Even though we think it's one of the most interesting aspects to consider for predicting the price of the house, it's mandatory for us to delete.
- Location.Address.PostalCodePlus4: 97% of NaN values, impossible to impute since would lose lots of information.
- Location.Address.UnparsedAddress: very difficult to deal with in terms of transforming to numerical. Better have each of the features in separate columns.
- Location.Address.StreetDirectionPrefix: doesn't make any sense to rely on the prefix or suffix of the street direction of the house. +50% of NaN values.
- Location.Address.StreetDirectionSuffix: same. 99% NaN values.
- Location.Address.StreetSuffix: we dropped it after seeing that it had a very weak correlation with our target feature and concluded it was not relevant.
- Location.Address.StreetNumber: by having the postal code we already have a potential acknowledgement of how the prices are going to go in that area.
- Listing.Dates.CloseDate: completely irrelevant since all the bought dates are just between 10 months, so there won't be a significant increase/decrease in the properties price in such a thin range of time.

- Location.Address.StateOrProvince: we are in the state of Illinois so it is irrelevant since all of the properties have the same value here.
- Location.GIS.Latitude: this is categorical data that is so hard to convert to some valuable numerical information. Furthermore, this is information whose value can be extracted from other existing features as postal code, so we can prescind of this.
- Location.GIS.Longitude: same as above.
- Location.Address.UnitNumber: 77% of Nan values, very difficult to impute.
- Location.Area.SubdivisionName: categorical values difficult to convert and 66% of NaN values.
- Location.School.HighSchoolDistrict: we calculated scores, saw that the model gave less accuracy and that correlation wasn't very relevant.
- Structure.BelowGradeFinishedArea: not enough info and +85% of NaN values.
- Structure.BelowGradeUnfinishedArea: same
- Structure.ParkingFeatures: 87% of NaN values so it gets very difficult to impute values for it.
- Tax.Zoning: 93% NaN values
- UnitTypes.UnitTypeType: 95% of NaN values.
- Location.Address.City: we can get it by the postal code, redundant
- Location.Address.CountyOrParish: same
- ImageData.style.exterior.summary.label: we had 42 different unique values and 25% NaN. Transforming it into numerical values given these data could lead to inconsistency in our data.
- Location.Address.StreetName: we can get it by the suffix.
- Structure.Heating: irrelevant, most of the houses have heating supply and it doesn't make a difference.

*1.2. Removing Biassed Rows

In this step, we deleted the 3. 97% ($\approx$ 4%) of our rows, which is equivalent to deleting 6688 rows. We decided to use a threshold of ⅓ (approx and after already removing the unnecessary features), that is, removing the rows that have 10 or more

NaN values across their features since then it will be very hard to impute the values and gain solid information from there.

*After doing that and delivering the first submission through Discord, we noticed that this was wrong and that we weren't able to delete rows since when running the Restb.ai model it would lead to an error due to row size difference. So we had to go back and leave these changes as they were initially, with the whole 107438 rows.

1.3. Categorical → Numerical and Imputing NaN values

- ImageData.c1c6.summary." " & ImageData.q1q6.summary." ": since all the features of this type have a range of NaN values between [10-15%] except for the property summary that only has 1%, we reached the conclusion that we must impute all the blanks with the median (more representative than the mean since the median is not affected by extreme values) of the other rows.
- ImageData.features_reso.results & ImageData.room_type_reso.results: here we compute the length of the list in order to convert it to numerical.
- ImageData.style.stories.summary.label: we extracted numeric values from labels (1, 2, 3, 1.5 and 2.5), imputed missing values with the median, and capped higher values at 3.
- Location.Address.PostalCode: we calculated the mean Listing.Price.ClosePrice by postal code, then normalised these values to a range of 0 to 1 using MinMaxScaler. Then we merged PostalCode_Score back into the dataset, by replacing the original postal code column (dropped). We tried to improve this approach by computing the scores based on a real dataset of incomes based on the postal code, however after some submissions we realised that it didn't really improve accuracy and the model performed better with the other approach, so we kept the initial proposal.
- Location.Address.CensusBlock: We used the same normalised method as before to convert the feature from categorical to numerical and being able to use it in our model.

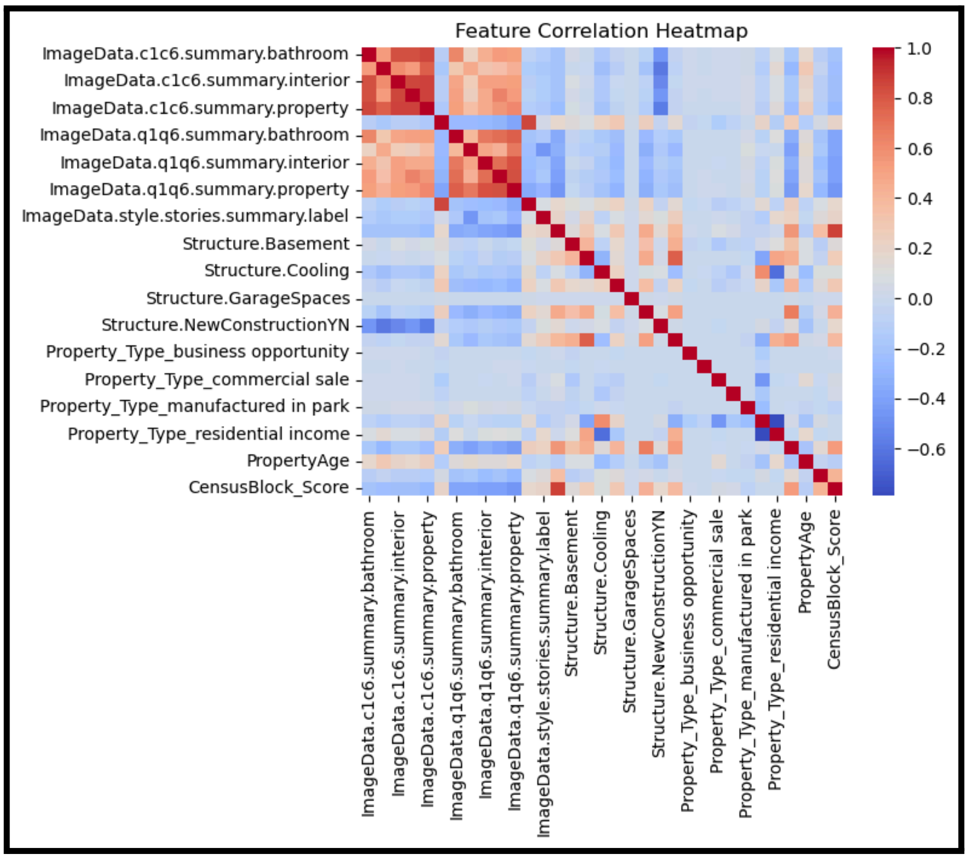
  * Location.Address.CensusTract: we also calculated scores, but we saw that the model gave less accuracy and that correlation wasn't very relevant.

- Property.PropertyType: here we applied one–hot encoding and we added the encoded columns to the dataset, of course removing the original column then.
- Structure.Basement: we transformed into binary (1 if basement exists, 0 otherwise), using both Structure.Basement and ImageData.room_type_reso.results.
- Structure.BathroomsFull & Structure.BathroomsHalf: here we created a new feature TotalBathrooms as a weighted sum of both features and then dropped the original columns.
- Structure.BedroomsTotal: imputed null values by computing the median of the others.
- Structure.Cooling: transformed into binary (1 if cooling exists, 0 otherwise). Use ImageData.features_reso.results to look for cooling instances to determine if some null value had an instance of cooling to set to 1. Otherwise we set the nulls to 0.
- Structure.FireplacesTotal: we applied the same methods as in the Structure.Cooling column.
- Structure.GarageSpaces: we imputed missing values within groups (Property.PropertyType) using the median; for groups with no data, imputed using the global median.
- Structure.LivingArea: imputed missing values by computing the mean of the others.
- Structure.NewConstructionYN: imputed by using the construction year (True if built after 2020).
- Structure.Rooms.RoomsTotal: filled the missing values based on the sum of bedrooms, bathrooms and an additional 2 rooms (kitchen, living room).
- Structure.YearBuilt: filled missing values with the median, calculated PropertyAge (CurrentYear – YearBuilt), and then dropped the original column.
- Listing.Price.ClosePrice: of course, leave as it is since it is our target label.

*We tend to use the median since it is not affected by extreme values (outliers), unlike the mean, which can be skewed by very high or very low values.

2. Modelling our Data

We start here by computing the correlation regarding our features towards the price the house was sold. We obtain the following headmap:



From that, and from the correlation values between each feature and the Listing.Price.ClosePrice, we determined whether there were some irrelevant features that we could drop to avoid redundancy in our model. We decided that we would drop the columns Structure.GarageSpaces and Structure.Cooling and all Property_Type one hot encoded columns. Besides, we obviously needed to drop the price from the training dataset, as it does not make sense to train the data with the label. We also dropped the ListingId before training as it is clear that it doesn't have any value for the correlation, it is just an identifier which we kept for the testing.

3. Training our model

We decided to train a random forest. From the few models we know how they work, which are linear regression, neural networks and random forests, we made some basic tests for each of them and compared accuracy and efficiency. Clearly the random forest yielded greater accuracy while having a reasonable execution time, so we went for this model.

After making that decision, we tried to tune the hyperparameters of the model to achieve the maximum accuracy possible. In our efforts to increase accuracy, we also revised some of our transformations in the data preprocessing part and made some adjustments to enhance the precision of the model; most of them resulted inefficient, as exposed in previous points.

Finally, we were left with a model that achieved the following results on our training dataset (which we splitted 80% on train data and 20% on test data):

- **Random Forest - Validation MSE:** 23,656,678,391.25
- **Random Forest - Validation MAE:** 50,740.45
- **Random Forest Accuracy:** 82.86%

Note that we had to split our data in train and test data even though we had a testing dataset, because that one was meant to be only for the submission to be evaluated, we didn't have the price values for that data, and so we had to split the train dataset to be able to test its performance.

4. References

[1] Internal Revenue Service. (n.d.). *SOI tax stats - Individual income tax statistics: 2021 ZIP code data (SOI)*. U.S. Department of the Treasury. Retrieved November 17, 2024, from
https://www.irs.gov/statistics/soi-tax-stats-individual-income-tax-statistics-2021-zip-code-data-soi-0

Description: From this official website of the US Government, the well-known IRS, we selected the state of Illinois (data 2021) explicitly in order to obtain the desired data for our project. After that, we cleaned the whole dataset (which was huge and difficult to deal with) in order to obtain two columns: one containing the ZIP Code (Postal Code) and the other one with the respective Total Income per this Zip Code (for all existing Postal Codes in Illinois).

*For the last submission, we didn’t use this dataset implementation since it led to more MSE than our best submission handed-in previously.