Cook County Assessor's Office Project

Part I executive summary:

1. Overview of the Case and Objectives

The Cook County Assessor's Office (CCAO) is tasked with determining the fair market value of properties, a critical process underpinning property tax collection, which funds public services such as education, law enforcement, and infrastructure. Historically, this process suffered from inaccuracies and a lack of transparency, prompting significant reforms under new leadership in 2018. This project aims to develop machine learning models that can predict property sale prices based on a range of historical and predictive features. The key objectives are:

- Analyze historical property data and identify significant predictors of sale prices.
- Build predictive models capable of estimating sale prices for properties with unknown sale values.
- Evaluate the accuracy of the models.

The primary dataset used for training consists of historical property data (data_historic), while the prediction dataset (data_predict) is used to apply the trained models and generate sale price predictions.

2. Methodology we will use

Data Understanding and Preparation:

Data Exploration: The project leverages two primary datasets, a historical dataset containing sales data for 50,000 properties ("historic_property_data.csv") and a dataset requiring predictions for 10,000 properties ("predict_property_data.csv"). Variable definitions from the "codebook.csv" guide our data analysis.

Data Cleaning: Missing values are addressed, and data is standardized to ensure compatibility across modeling techniques.

Feature Selection: Numeric predictors will be identified based on their correlation with property sales prices. We will use a correlation matrix.

Categorical predictors will be handle using Anova method or Lasso regression with One-hot encoding. Predictors with the most importance will be selected to create a new dataset of selected features to build the models.

Model Development and Validation:

Initial Models: We begin with a simple linear regression to establish baseline predictions, focusing on interpretability and identifying key features. Then, we will perform a multiple linear regression analysis by adding other variables to enhance the accuracy of the prediction.

Advanced Models: During the course of the project, we will explore more complex machine learning techniques such as Random Forests, in order to enhance prediction accuracy. These models capture non-linear relationships and interactions between features.

Validation: Cross-validation is used to evaluate model performance, preventing overfitting and ensuring generalizability. We will calculate the RMSE as well as the R-Squared value to assess the performance of the models, choosing the one with the lowest RMSE.

Prediction and Submission:

The selected model is applied to the "predict_property_data.csv" dataset to generate property value predictions.

Predictions are formatted into a CSV file with two columns: property ID ("pid") and assessed value ("assessed_value"). This file adheres to the specified requirements and ensures compatibility with evaluation protocols.

Explanation of the code

Purpose

Using a dataset 'historic_property_data' to train and test a machine learning model to predict the sale price of houses in a "predict_property_data". The predicted running time of the entire file is about 30 minutes, this is due to the high level of complexity presented in the models.

1. Setting Up the Project

The project is titled Finalproject.

2. Loading Libraries and Data

First, the required libraries, tidyverse and caret, are loaded. Tidyverse is used for data manipulation and visualization, while caret is used for machine learning tasks such as partitioning and training models. The script then loads two datasets: data_historic, which is used for training the model, and data_predict, which is the dataset for predictions.

```
##########################################################################################################
# A full run of this script should take approximately 30 minutes #########################################

# Loading required libraries
library(tidyverse)
library(caret)


# Loading the data
data_historic <- read.csv("historic_property_data.csv")
data_predict <- read.csv("predict_property_data.csv")
```

3. Exploring the Data

The structure of the datasets is visualized using the str() function. This step helps identify data types, detect missing values, and determine which variables need cleaning or transformation before proceeding. We want to know the exact data structure, the data types for each column as well as verifiyng any NA values that could cause issues later.

```
'data.frame':	50000 obs. of  63 variables:
 $ sale_price                : int  23000 95000 234900 110000 640000 195000 520000 160000 60400 367000 ...
 $ meta_class                : int  203 211 205 203 206 203 205 202 202 211 ...
 $ meta_town_code            : int  12 70 12 72 38 22 17 71 31 17 ...
 $ meta_nbhd                 : int  12084 70120 12051 72071 38052 22111 17110 71071 31030 17112 ...
 $ meta_certified_est_bldg   : int  64800 126710 190650 73610 379790 353430 308910 192500 58180 333750 ...
 $ meta_certified_est_land   : int  16800 33480 33750 33020 61870 53120 45370 48760 23450 61420 ...
 $ meta_cdu                  : chr  NA NA "AV" NA ...
 $ meta_deed_type            : chr  "O" "W" "W" "W" ...
 $ char_hd_sf                : int  6720 3720 9000 4128 9900 6250 5500 3612 6700 5850 ...
 $ char_age                  : int  58 96 65 89 75 99 93 103 60 91 ...
 $ char_apts                 : int  NA 2 NA NA NA NA NA NA NA 2 ...
 $ char_ext_wall             : int  2 2 3 2 3 2 1 1 1 2 ...
 $ char_roof_cnst            : int  1 1 1 1 1 1 1 1 1 2 ...
 $ char_rooms                : int  5 10 8 6 8 5 8 4 4 10 ...
 $ char_beds                 : int  3 6 4 3 4 3 3 2 2 5 ...
 $ char_bsmt                 : int  1 1 1 1 1 1 1 1 2 1 ...
 $ char_bsmt_fin             : int  3 3 3 3 3 3 3 3 3 3 ...
 $ char_heat                 : int  1 2 1 2 1 2 1 2 1 1 ...
 $ char_oheat                : int  5 5 5 5 5 5 5 5 5 5 ...
```

We can see, many different variables, with different types. For example, sale_price (target) is n integer while meta_cdu is a char and contains NA values.

4. Data Cleaning

The data cleaning process ensures that all variables are in the correct format for analysis. Logical columns like ind_large_home, ind_garage, and ind_arms_length are converted into factors to make them compatible with machine learning models. Character columns are also converted to factors for standardization. Redundant or irrelevant columns, such as ind_garage, are removed to reduce dimensionality and focus on meaningful predictors. Missing values in numeric columns are replaced with their median, while missing values in categorical columns are replaced with the label Unknown.

```
'data.frame':	50000 obs. of  62 variables:
 $ sale_price                : int  23000 95000 234900 110000 640000 195000 520000 160000 60400 367000 ...
 $ meta_class                : int  203 211 205 203 206 203 205 202 202 211 ...
 $ meta_town_code            : int  12 70 12 72 38 22 17 71 31 17 ...
 $ meta_nbhd                 : int  12084 70120 12051 72071 38052 22111 17110 71071 31030 17112 ...
 $ meta_certified_est_bldg   : int  64800 126710 190650 73610 379790 353430 308910 192500 58180 333750 ...
 $ meta_certified_est_land   : int  16800 33480 33750 33020 61870 53120 45370 48760 23450 61420 ...
 $ meta_cdu                  : Factor w/ 13 levels "AR","AV","AX",..: NA NA 2 NA NA NA NA NA NA NA ...
 $ meta_deed_type            : Factor w/ 3 levels "O","T","W": 1 3 3 3 3 2 3 3 1 2 ...
 $ char_hd_sf                : int  6720 3720 9000 4128 9900 6250 5500 3612 6700 5850 ...
 $ char_age                  : int  58 96 65 89 75 99 93 103 60 91 ...
 $ char_apts                 : int  NA 2 NA NA NA NA NA NA NA 2 ...
 $ char_ext_wall             : int  2 2 3 2 3 2 1 1 1 2 ...
 $ char_roof_cnst            : int  1 1 1 1 1 1 1 1 1 2 ...
 $ char_rooms                : int  5 10 8 6 8 5 8 4 4 10 ...
 $ char_beds                 : int  3 6 4 3 4 3 3 2 2 5 ...
 $ char_bsmt                 : int  1 1 1 1 1 1 1 1 2 1 ...
 $ char_bsmt_fin             : int  3 3 3 3 3 3 3 3 3 3 ...
 $ char_heat                 : int  1 2 1 2 1 2 1 2 1 1 ...
 $ char_oheat                : int  5 5 5 5 5 5 5 5 5 5 ...
 $ char_air                  : int  2 2 1 2 1 2 2 2 2 2 ...
 $ char_frpl                 : int  0 0 0 0 1 0 2 0 0 0 ...
 $ char_attic_type           : int  3 3 2 1 3 3 3 1 2 3 ...
 $ char_fbath                : int  1 2 2 1 2 1 2 1 1 2 ...
 $ char_hbath                : int  0 0 1 0 1 0 0 0 0 0 ...
 $ char_tp_plan              : int  NA 2 2 NA 2 NA 2 NA 2 2 ...
 $ char_tp_dsgn              : int  NA 2 2 NA NA NA 2 NA NA 2 ...
 $ char_cnst_qlty            : int  2 2 2 2 2 2 2 2 2 2 ...
```

5. Feature Engineering

Categorical variables like meta_class, meta_town_code, and geo_fips are explicitly converted to factors to align with the needs of the machine learning models. Binary variables, such as geo_withinmr100 and geo_withinmr101300, are transformed into labeled factors with levels like "Outside" and "Inside" for better interpretability. These steps ensure consistent data preparation for both the training and prediction datasets. Once every steps is done, we make sure that there are not any remaining NA values using the is.na() function:

```
      sale_price               meta_class          meta_town_code
               0                        0                       0
       meta_nbhd  meta_certified_est_bldg meta_certified_est_land
               0                        0                       0
        meta_cdu           meta_deed_type              char_hd_sf
               0                        0                       0
        char_age                char_apts           char_ext_wall
               0                        0                       0
  char_roof_cnst               char_rooms               char_beds
               0                        0                       0
       char_bsmt            char_bsmt_fin               char_heat
               0                        0                       0
      char_oheat                 char_air               char_frpl
               0                        0                       0
 char_attic_type               char_fbath              char_hbath
               0                        0                       0
    char_tp_plan             char_tp_dsgn           char_cnst_qlty
               0                        0                       0
       char_site           char_gar1_size          char_gar1_cnst
               0                        0                       0
   char_gar1_att           char_gar1_area            char_ot_impr
               0                        0                       0
    char_bldg_sf          char_repair_cnd                char_use
               0                        0                       0
  char_type_resd          char_attic_fnsh         char_renovation
               0                        0                       0
```

As shown above, there are no remaining NA values in the columns.

6. Variable Selection

For numeric variables, a correlation matrix is calculated to assess the relationship between each variable and the target variable, sale_price. Variables with a correlation coefficient greater than or equal to 0.4 are selected as significant predictors.

```
[1] "Significant numeric variables:"
[1] "sale_price"              "meta_certified_est_bldg" "meta_certified_est_land"
[4] "char_frpl"               "char_fbath"              "char_bldg_sf"
[7] "geo_white_perc"          "econ_midincome"
```

The selected numeric variables using 0.4 as a threshold for significance levels are shown above.

For categorical variables, we decided to use ANOVA testing to identify those significantly associated with sale_price. **ANOVA (Analysis of Variance)** is a statistical method used to determine whether there are significant differences between the means of a target variable across the levels of a categorical predictor. In the context of variable selection, ANOVA helps identify categorical variables that have a statistically significant relationship with the target variable (sale_price). This process reduces the number of predictors, minimizing overfitting and improving model performance. We first defined all our categorical variables and then created a function performing aov() on the argument. Using this function, here are the selected variables with aov() method:

```
[1] "Significant categorical variables:"
 [1] "meta_class"               "meta_town_code"           "meta_nbhd"
 [4] "meta_cdu"                 "meta_deed_type"           "char_ext_wall"
 [7] "char_roof_cnst"           "char_heat"                "char_use"
[10] "geo_property_city"        "geo_property_zip"         "geo_municipality"
[13] "geo_fips"                 "geo_school_elem_district" "geo_school_hs_district"
[16] "ind_large_home"           "ind_arms_length"
```

Then, we recreate a new dataset that combines all the selected categorical and numerical variables:

```
# Print the results
print("Significant categorical variables:")
print(significant_categorical_cols)

# Combine selected numeric and categorical variables
final_selected_columns <- c(significant_numeric_cols, significant_categorical_cols)
selected_historic_data <- data_historic[final_selected_columns]
```

selected_historic_data is the final dataset that will be used to train our Machine Learning models.

After the variable selection, we have 8 numerical variables and 17 categorical variables with statistical significance, so our **selected_historic_data** set will include 25 variables in total.

```
'data.frame':	50000 obs. of  25 variables:
 $ sale_price              : num  23000 95000 234900 110000 640000 ...
 $ meta_certified_est_bldg : num  64800 126710 190650 73610 379790 ...
 $ meta_certified_est_land : num  16800 33480 33750 33020 61870 ...
 $ char_frpl               : int  0 0 0 0 1 0 2 0 0 0 ...
 $ char_fbath              : num  1 2 2 1 2 1 2 1 1 2 ...
 $ char_bldg_sf            : num  1054 2756 2008 1486 2356 ...
 $ geo_white_perc          : num  0.0768 0.0301 0.3579 0.0424 0.8935 ...
 $ econ_midincome          : num  26167 26508 88225 27423 116667 ...
 $ meta_class              : Factor w/ 14 levels "202","203","204",..: 2 10 4 2 5 2 4 1 1 10 ...
 $ meta_town_code          : Factor w/ 38 levels "10","11","12",..: 3 31 3 33 29 13 8 32 22 8 ...
 $ meta_nbhd               : Factor w/ 843 levels "10011","10012",..: 43 667 30 731 607 241 142 692 417 143 ...
 $ meta_cdu                : Factor w/ 14 levels "AR","AV","AX",..: 14 14 2 14 14 14 14 14 14 14 ...
 $ meta_deed_type          : Factor w/ 4 levels "O","T","W","Unknown": 1 3 3 3 3 2 3 3 1 2 ...
 $ char_ext_wall           : Factor w/ 4 levels "1","2","3","4": 2 2 3 2 3 2 1 1 1 2 ...
 $ char_roof_cnst          : Factor w/ 6 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 2 ...
 $ char_heat               : Factor w/ 4 levels "1","2","3","4": 1 2 1 2 1 2 1 2 1 1 ...
 $ char_use                : Factor w/ 2 levels "1","2": 1 2 1 1 1 1 1 1 1 2 ...
 $ geo_property_city       : Factor w/ 136 levels "ALSIP","ARLINGTON HEIGHTS",..: 21 20 37 20 2 101 35 20 12 35 ...
 $ geo_property_zip        : Factor w/ 172 levels "0-0","60004",..: 62 126 66 135 3 18 53 136 37 53 ...
 $ geo_municipality        : Factor w/ 128 levels "Alsip","Arlington Heights",..: 21 20 36 20 2 94 34 20 12 34 ...
 $ geo_fips                : Factor w/ 127 levels "1010","2154",..: 21 20 36 20 2 94 34 20 12 34 ...
 $ geo_school_elem_district: Factor w/ 477 levels "ADDAMS","AGASSIZ",..: 343 292 372 213 375 413 414 153 398 414 ...
 $ geo_school_hs_district  : Factor w/ 81 levels "AMUNDSEN HS",..: 5 34 35 29 2 50 25 66 61 25 ...
 $ ind_large_home          : Factor w/ 3 levels "FALSE","TRUE",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ ind_arms_length         : Factor w/ 3 levels "FALSE","TRUE",..: 2 2 2 2 2 2 2 2 2 2 ...
```

7. Splitting Data for Training and Testing

The cleaned and filtered dataset is split into training (80%) and testing (20%) subsets using the createDataPartition function. This split ensures that the model can be trained and evaluated on separate portions of the data, providing a realistic assessment of its performance.

```r
###############. Third step: Creating the ML model with the lowest MSE ##########################

# Creating the ML models

library(caret)

# Splitting the dataset into training and tesing datasets, 80% training 20% testing
set.seed(123)  # Set seed for reproducibility
train_index <- createDataPartition(selected_historic_data$sale_price, p = 0.8, list = FALSE)
train_data <- selected_historic_data[train_index, ]
test_data <- selected_historic_data[-train_index, ]
```

8. Model Training

Two machine learning models are trained.

Linear regression model:

First, a linear regression model is built using 5-fold cross-validation, implemented through the caret package.

```{r}
# Linear regression model
set.seed(123)
train_control <- trainControl(method = "cv", number = 5)
lm_model <- train(
  sale_price ~ .,
  data = train_data,         # Training dataset
  method = "lm",             # Linear regression method
  trControl = train_control
)

```

The linear regression model is evaluated using RMSE (Root Mean Square Error) to measure prediction error and R-squared (coefficient of determination) to evaluate the goodness of fit:

```
Residuals:
     Min       1Q   Median       3Q      Max
-1230537   -40578     -326    36493  2231931

Coefficients: (397 not defined because of singularities)
                                                                     Estimate Std. Error t value
(Intercept)                                                        -2.615e+05  2.042e+05  -1.280
meta_certified_est_bldg                                             7.154e-01  6.416e-03 111.494
```

```
Linear Regression Model - RMSE: 124135.7
Linear Regression Model - R²: 0.8378
```

Performance of our linear model:

- RMSE = 124135.7

- R-Squared = 0.8378 : 83% of the variance is explained in the target variable sale_price.

The standard error of the coefficient is very small (<0.05), showing precise estimation.

Random forest model:

One-hot encoding is applied to the training dataset for compatibility with the Random Forest algorithm.

```
# Apply one-hot encoding to training and test datasets
dummies_train <- model.matrix(sale_price ~ ., data = train_data)[, -1]  # Remove intercept
dummies_test <- model.matrix(sale_price ~ ., data = test_data)[, -1]  # Remove intercept
```

To reach better performances, we decided to build a random forest model with an optimal number of trees. A Random Forest model is trained with 35 trees (ntree = 35), and variable importance is assessed.

```r
# Train a Random Forest model with a fixed ntree value (ntree=35)
set.seed(123)
rf_model <- randomForest(
  x = dummies_train,
  y = train_data$sale_price,
  ntree = 35,
  importance = TRUE
)
```

9. Model Evaluation

For the Random Forest model, RMSE is calculated on the test dataset, and variable importance scores are extracted to identify the most impactful predictors. These metrics provide a comprehensive evaluation of the models' performance.

```
Random Forest regression test RMSE (ntree = 10): 122593.5
Random Forest variable importance (ntree = 10):
                                                           %IncMSE IncNodePurity
meta_certified_est_bldg                                11.9339947926  2.015800e+15
meta_certified_est_land                                 5.8758192693  3.736715e+14
char_frpl                                               2.3232387444  1.809676e+13
char_fbath                                              2.6875411642  1.311822e+14
char_bldg_sf                                            5.3338282524  1.874004e+14
```

Performance of our random forest model:

- RMSE = 122593.5

10. Making Predictions

We have better results with a higher accuracy using the random forest model (smaller RMSE/MSE), so we decided to use this one for the predictions. The prediction dataset, data_predict, is prepared by aligning its columns with the training data through one-hot encoding. Missing columns are added with default values of zero, and extra columns are removed to ensure consistency. Predictions for sale_price are generated using the trained Random Forest model and added to the data_predict dataset.

11. Output Results

The predictions are appended to the original data_predict dataset as a new column, predicted_sale_price. These predictions are then saved as a CSV file for external use.

```
# Save predictions to a CSV file
output_file <- "predicted_sale_prices.csv"
write.csv(data_predict, output_file, row.names = FALSE)
cat("Predictions successfully saved to:", output_file, "\n")

# Prepare and format final results for reporting
final_result <- data_predict[, c("pid", "predicted_sale_price")]
colnames(final_result) <- c("pid", "assessed_value")
print("Final formatted results:")
print(head(final_result))
```

Finally, the results are formatted for reporting by extracting the **pid** and **predicted_sale_price** columns and renaming them to pid and assessed_value, respectively. The formatted results are displayed for review.

	pid <dbl>	meta_class <fctr>	meta_town_code <fctr>	meta_nbhd <fctr>	meta_certified_est_bldg <dbl>	meta_certified_est_land <dbl>	meta_cdu <fctr>	meta_deed_type <fctr>
1	1	208	26	26010	434370	123280	Unknown	W
2	2	211	77	77091	66430	37010	Unknown	W
3	3	211	72	72091	34170	19500	Unknown	W
4	4	234	32	32160	97950	29400	Unknown	O
5	5	234	22	22080	225820	53590	Unknown	W
6	6	203	29	29060	179770	37800	Unknown	W

6 rows | 1-9 of 63 columns

pid <dbl>	assessed_value <dbl>
1	568390.64
2	61848.33
3	71476.67
4	89799.91
5	296252.50
6	249819.67
7	96930.19
8	195624.33
9	121394.67
10	304721.27

Conclusions

The Random Forest model was the most accurate model for the predicted value with an accurac of prediction over 88% compared to the actual value. Therefore, we will use this model as final model for the propriety assessment.