TABLE OF CONTENTS

I. Introduction of the problem

1. Set the problem

House prices are an important measure of the economic situation of a country or territory. Rising house prices are often accompanied by economic growth and increased job opportunities [37], while falling prices can signal economic stagnation or other adverse factors. In recent years, the rapid development of the real estate industry has pushed house prices to unprecedented highs [1], attracting the attention of government agencies, businesses, communities, and individuals.

Amid the real estate boom, the escalating trend of house prices has become the focus of discussion. However, the impressive growth figures are accompanied by growing concerns, as they reflect both macro trends and the micro realities of the economy. While rapid growth may[4] imply bubble risk, soaring house prices threaten basic demand for housing affordability. The factors affecting house prices go far beyond macro regulations. Architectural design, landscaping, construction quality, and even factors such as lighting and layout can all directly affect the value of each property [5]. It is this inherent diversity and volatility that contributes to the dynamics of the real estate market, making navigation both exciting and challenging.

In the context of annual increases in land and housing prices, accurately predicting house prices has become an urgent requirement. Not only do buyers and sellers rely on accurate estimates to make informed decisions [17], but policymakers also need reliable predictions to guide stable and sustainable markets. Predicting house prices has emerged as a core issue in the real estate and economic sectors, requiring in-depth research and the development of advanced methodologies that can contribute to market stability and protect the interests of all stakeholders.

2. Introduction

House prices are always a topic that attracts the attention of many people, especially in the context of the constantly fluctuating real estate market. Predicting accurate sales

prices for each home can bring many benefits to investors, brokers and homebuyers., the goal of the study was to focus on predicting the sales price for each home in Ames, a city in Story County in the state of Iowa. United States. The city has a total area of 62.86 km², of which 62.70 km² is land.

Popular algorithms in predicting house prices such as: Linear Regression, Random Forest, Gradient Boosted Trees, ... The study data is a dataset with 79 explanatory variables describing (almost) every aspect of housing in Ames, Iowa. Some typical data fields are: Sale Price , Electrical , Garage Area , Sale Type , Functional.

The results of this study will help investors, brokers and home buyers make more informed decisions in buying and selling homes in Ames. Although the study focused solely on predicting home prices in Ames, Iowa, USA, these methods and results may also contribute to the development of more effective machine learning models for predicting home prices in other areas.

II. Related studies

In the face of the difficult challenge of changing the real estate market, a series of studies have been produced, each seeking to uncover the secrets hidden in the field of house price prediction. From advanced regression techniques explored in "Predicting House Prices Using Advanced Regression Techniques [1]" to the Dependent Variable-Weighted Regression Method championed in "Dependent Variable-Weighted Regression (CWR): A Case Study for Estimating House Prices [9]", Geo-weighted regression (GWR) is a common tool for modeling spatial heterogeneity in a regression model. However, the current weighting function used in GWR only takes geographic distance into account, while similarities in attributes are completely ignored. In this study, we propose a dependent variable weighting function that combines both geographic distance and attribute distance. Dependent variable distance weighted regression (CWR) is the extension of GWR to include geographic distance and attribute distance. House prices are affected by many factors, such as house age, floor area, and land use. Predictive modeling is used to help understand the characteristics of home

prices by region. CWR is used to understand the relationship between house prices and control factors. CWR can consider geological distances and property distances, and give accurate estimates of house prices, preserving weighting matrices for geological distance functions and property distances. The results show that home properties/conditions and home characteristics, such as floor area and home age, can affect home prices. After factor selection, which only considers the house age and floor area of the building, the RMSE of the CWR model can be improved from 2.9% – 26.3% for skyscrapers compared to GWR. CWR can effectively reduce estimation errors from traditional spatial regression models and provide new and feasible models for spatial estimation.

The researchers also used associative models to predict house prices A blended model based on Python and its xgboost, DF21 and Geatpy packages to predict resale house prices in Singapore [11]. First, high cardinal classification properties are pre-processed using the medium coding method. The researchers then proposed a linear blending method consisting of GA-HL-Xg-Boost, GARandom Forest (GA-RF), deep-Random Forest (DRF), and lightGBM, with Gini impurities to determine the importance of features. Finally, the results suggest that it can achieve an average absolute percentage error (MAPE) of 7.36% in the fixed general trend of the house price trend. This study can provide a strong forecast of home resale prices in different economic environments.

In addition, there are a few works that report the use of machine learning (ML) algorithms to predict house prices. In Brazil, there is this study that analyzes a dataset consisting of 12,223,582 housing ads, collected from Brazilian websites between 2015 and 2018. Each instance includes twenty-four features of five different data types: integers, dates, strings, floats, and images. To predict real estate prices, they combined two different ML architectures, based on[15] Random Forest (RF) and Recurrent Neural Networks (RNN). This study demonstrates that enriching datasets and incorporating different ML methods can be a better alternative to predicting home prices in different areas

III. Methods

3.1. Foundation Engineering

3.1.1. Mean squared error (MSE)

Mean squared error (MSE) is a statistical measure used to evaluate the extent of the difference between the predicted value and the actual value in machine learning models.

Illustrated as follows: $$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Where: n is the amount of data

Yi is practical value

Y^i is the predictive value

The lower the MSE value, the more accurate the model. The higher the MSE value, the less accurate the model. MSE is widely used to evaluate the effectiveness of machine learning models in many fields, including: Evaluating the accuracy of the model in predicting continuous values (Regression) and Evaluating the accuracy of the model in classifying data patterns (Classification)

MSE is considered easy to understand and simple to calculate, comparable between different models. However, the drawback is sensitivity to externalities, externalities can increase MSE values and affect the evaluation of the model. In predicting home sales prices, MSE is used to evaluate how accurate the Random Forest model is in predicting home sale prices. The lower the MSE value, the more accurately the model predicts the selling price of the home.

3.1.2. Squared Mean Square Root (RMSE)

Square root squared mean of error (RMSE) is a statistical measure used to evaluate the difference between predicted and actual values in machine learning models.

RMSE is the square root of MSE (Mean Squared Error) with the following formula: **RMSE = √(MSE)**

RMSE has the same unit of measurement as the actual value, making it easy to assess the degree of deviation. The lower the RMSE value, the more accurate the model. The higher the RMSE value, the less accurate the model. RMSE is widely used to evaluate the effectiveness of machine learning models in many areas, including: Evaluating the accuracy of the model in predicting continuous values (Regression) and Evaluating the accuracy of the model in classifying data patterns (Classification)

RMSE is considered easy to understand and simple to calculate. There are units of measurement, which makes it easy to compare the degree of deviation between models. However, the drawback is sensitivity to externalities and RMSE is not statistically significant. In predicting home sales prices, RMSE is used to assess how accurate the Random Forest model is in predicting home prices. The lower the RMSE value, the more accurately the model predicts the selling price of a home.

3.1.3. R^2: Deterministic correlation coefficient

R^2 (Deterministic Correlation Coefficient) is a statistical measure used to evaluate the correlation between predicted and actual values in machine learning models. R^2 represents the percentage of variability of the dependent variable explained by the independent variables.

Formula**: R^2 = 1 - (SSR / SST)**

Where: R^2: Deterministic correlation coefficient

SSR: Sum of Squared Residuals.

SST: Sum of Squared Totals

The higher the R^2 value, the better the model's predictive power. The lower the R^2 value, the worse the model's predictive power. The value R^2 is not negative and cannot

be greater than 1. R^2 is widely used to evaluate the effectiveness of machine learning models in many fields, including: Evaluating the accuracy of the model in predicting continuous values (regression) and Evaluating the accuracy of the model in classifying data patterns (Classification)

R^2 is considered easy to understand and simple to calculate, comparable between different models. However, the drawback is that it is not statistically significant, sensitive to extraneous values.

In predicting home sales prices, R^2 is used to evaluate the accuracy of the Random Forest model in predicting home sales prices. The higher the R^2 value, the more accurately the model predicts the selling price of the home.

3.1.4. Adjusted R^2: Correlation coefficient determines adjustment

Adjusted R^2 is an adjusted version of R^2 that compensates for how the predicted number of variables affects the R^2 value. Adjusted R^2 tends to decrease when adding more unimportant predictive variables to the model.

Formula : **Adjusted R^2 = 1 - [(1 - R^2) * (n - 1) / (n - k)]**

Where: Adjusted R^2: Correlation coefficient determines adjustment

R^2: Coefficient of deterministic correlation

n : Quantity of data

k : Number of predicted variables

The higher the Adjusted R^2 value, the better the model's predictive power. The lower the Adjusted R^2 value, the worse the model's predictive power. The Adjusted R^2 value is not negative and cannot be greater than 1. Adjusted R^2 is widely used to evaluate the effectiveness of machine learning models in many areas, including: Evaluating the accuracy of the model in predicting continuous values (Regression) and Evaluating the accuracy of the model in classifying data patterns (Classification)

Adjusted R^2 helps compare the performance between models with different numbers of predicted variables, compensating for the fact that R^2 tends to increase when more predicted variables are added. However, it is not statistically significant and sensitive to externalities. In predicting home sales prices, Adjusted R^2 is used to evaluate how accurate the Random Forest model is in predicting home sales prices. The higher the Adjusted R^2 value, the more accurately the model predicts the selling price of the home. The Adjusted R^2 value of the Random Forest model should be compared with other models to evaluate the effectiveness of the model. Both R^2 and Adjusted R^2 are statistical measures, so they should not be used to make final decisions about model selection.

3.1.5. MAE (Mean Absolute Error) indicator

The MAE (Mean Absolute Error) metric is a statistical measure used to evaluate the average deviation between predicted and actual values in machine learning models. MAE is the average of the absolute values of error.

Formula:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Where: MAE: Mean Absolute Error

n: Quantity of data

Yi: Practical value

Xi: Predictive value

The lower the MAE value, the more accurate the model. The higher the MAE value, the less accurate the model. MAE is widely used to evaluate the effectiveness of machine learning models in many fields, including Evaluating the accuracy of the model in predicting continuous value (Regression) and Evaluating the accuracy of the model in classifying data patterns (Classification).

MAE is considered easy to understand and simple to calculate, has a unit of measurement and is less sensitive to outliers than MSE. However, MAE is not statistically significant and does not penalize errors as large as MSE. In predicting home sales prices, MAE is used to evaluate how accurate the Random Forest model is in predicting home sale prices. The lower the MAE value, the more accurately the model predicts the home sale price. However, it is necessary to compare the MAE value of the Random Forest model with other models to evaluate the effectiveness of the model. Both MAE and MSE are statistical measures, therefore they should not be used to make final decisions about model selection. Other factors such as model complexity, explainability, and generalization should be considered when selecting a model.

3.1.6. MAPE (Mean Absolute Percentage Error) indicator

The MAPE (Mean Absolute Percentage Error) metric is a statistical measure used to evaluate the average percentage deviation between the predicted and actual values in machine learning models. MAPE represents the average error in percentages.

Formula:

$$\text{MAPE} = \frac{1}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

Where: Percentage Error (Average Absolute Error Index)

n: Quantity of data

At: Actual value

Ft: Predictive value

The lower the MAPE value, the more accurate the model. The higher the MAPE value, the less accurate the model. MAPE can be directly compared between percentage value prediction models. MAPE is widely used to evaluate the effectiveness of machine learning models in the areas of: Evaluating the accuracy of the model in predicting

percentage values (Regression) and Evaluating the accuracy of the model in classifying data patterns (Classification).

The MAPE model is considered easy to understand and simple to calculate. It has units of measurement (percentages) and allows direct comparisons between models that predict percentage values. However, MAPE is not statistically significant, is sensitive to zero or near-zero values in actual values, and penalizes small errors more than large ones. In predicting home sales prices, MAPE is used to evaluate how accurate the Random Forest model is in predicting home sales prices as a percentage. The lower the MAPE value, the more accurately the model predicts the selling price of the home.

3.2 Method recommendations

TensorFlow Decision Forests (TF-DF) is an open source library developed by Google that provides powerful tools for building and deploying decision forest models on the TensorFlow platform. TF-DF offers many benefits for machine learning model development, including:

- High performance: TF-DF is optimized for high performance, which speeds up training and model prediction.
- Scalability: TF-DF can handle large amounts of data efficiently, suitable for large-scale machine learning applications.
- Easy to use: TF-DF provides a simple and easy-to-use API, making it easy to build and deploy decision forest models.
- Flexibility: TF-DF supports many different types of decision forest models, allowing you to choose the model that best suits your needs.

TF-DF consists of the following main components:

- Model Building Kit: Provides tools for building different types of decision forest models, including Random Forest, Gradient Boosting Trees, and XGBoost.
- Data processor: Provides tools to prepare data for model training, including underhandling, data encoding, and dataset division.

- Model Evaluator: Provides tools for evaluating model performance, including calculating evaluation metrics such as accuracy, F1 degrees, and AUC.
- Model Deployment Kit: Provides tools to deploy models to production, including saving models, exporting models to TensorFlow Lite, and using models in web and mobile applications.

TF-DF is a powerful and easy-to-use library for building and deploying decision forest models on the TensorFlow platform. TF-DF offers many features and benefits that help develop a model that predicts home prices.

Linear regression is a statistical model used to predict the value of a dependent variable based on one or more independent variables. The model uses linear equations to describe relationships between variables.

- Simple and straightforward: Linear regression is a simple and straightforward machine learning model. This model uses linear equations to predict home prices based on independent variables. As a result, model deployment and interpretation become easier than other complex machine learning models.
- Efficiency: Linear regression is an effective machine learning model. This model can learn quickly and predict accurately with small amounts of data. This saves time and costs when deploying the model.
- Explainability: Linear regression provides high explainability. This model allows you to determine the degree of influence of each independent variable on house prices. As a result, users can make more informed investment decisions.
- Scalability: Linear regression can be easily scaled to handle large amounts of data. This makes the model consistent with predicting house prices in major real estate markets.
- Fast processing speed: Linear regression has a faster processing speed than other complex machine learning models. This makes the model consistent with predicting house prices in real time.

Using Linear Regression to predict home prices has many advantages, including simplicity, efficiency, explainability, scalability, fast processing speed. Therefore, Linear regression is a good choice for predicting house prices in practice, especially when the available data is linear.

IV. Experiment

4.1. Data Description

4.1.1. Overview

Data source: Data provided by subject faculty

4.1.2. Data characteristics

Import a library

In [1]:
```
import tensorflow as tf
import tensorflow_decision_forests as tfdf
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Comment this if the data visualisations doesn't work on your side
%matplotlib inline
```

In [2]:
```
print("TensorFlow v" + tf.__version__)
print("TensorFlow Decision Forests v" + tfdf.__version__)
```

```
TensorFlow v2.11.0
TensorFlow Decision Forests v1.2.0
```

Upload data

In [3]:
```
train_file_path = "../input/house-prices-advanced-regression-techniques/train.csv"
dataset_df = pd.read_csv(train_file_path)
print("Full train dataset shape is {}".format(dataset_df.shape))
```

```
Full train dataset shape is (1460, 81)
```

In [6]:
```
dataset_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   MSSubClass     1460 non-null   int64
 1   MSZoning       1460 non-null   object
 2   LotFrontage    1201 non-null   float64
 3   LotArea        1460 non-null   int64
 4   Street         1460 non-null   object
 5   Alley          91 non-null     object
 6   LotShape       1460 non-null   object
 7   LandContour    1460 non-null   object
 8   Utilities      1460 non-null   object
 9   LotConfig      1460 non-null   object
 10  LandSlope      1460 non-null   object
 11  Neighborhood   1460 non-null   object
 12  Condition1     1460 non-null   object

 78  SaleCondition  1460 non-null   object
 79  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(34), object(43)
memory usage: 912.6+ KB
```

The data contains 80 attributes and 1459 records

In [7]:
```
print(dataset_df['SalePrice'].describe())
plt.figure(figsize=(9, 8))
sns.distplot(dataset_df['SalePrice'], color='g', bins=100, hist_kws={'alpha': 0.4});
```

```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

Consider how arithmetic characteristics are distributed. To do this, first list all data types from the dataset and select only arithmetic data types

In [8]:
```
list(set(dataset_df.dtypes.tolist()))
```

Out[8]:
```
[dtype('O'), dtype('int64'), dtype('float64')]
```

There are three types of data in our dataset:

- dtype('O'): Object data, usually strings of characters.
- dtype('int64'): 64-bit integer data.
- dtype('float64'): 64-bit real number data.

We will only select features with arithmetic data types (int64 and float64) for distribution analysis.

In [9]:
```
df_num = dataset_df.select_dtypes(include = ['float64', 'int64'])
df_num.head()
```

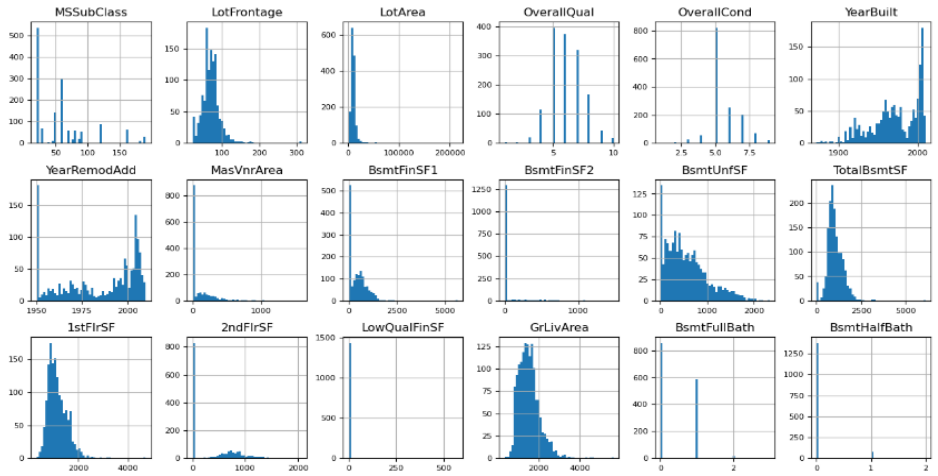
Out[9]:

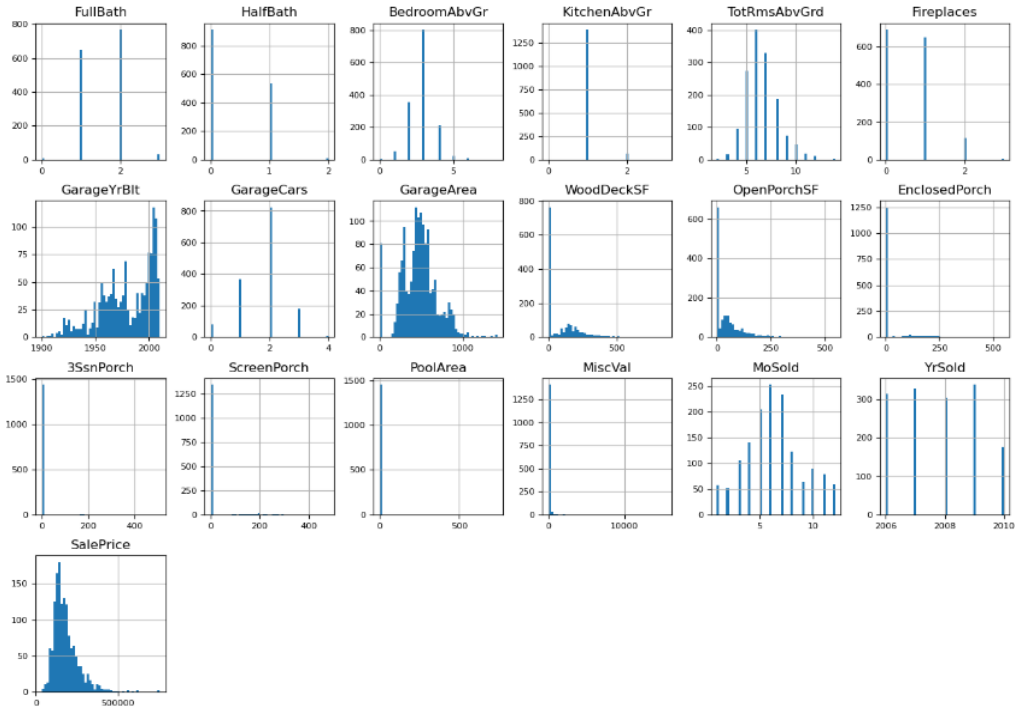
	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	M
0	60	65.0	8450	7	5	2003	2003	1
1	20	80.0	9600	6	8	1976	1976	0
2	60	68.0	11250	7	5	2001	2002	1
3	70	60.0	9550	7	5	1915	1970	0
4	60	84.0	14260	8	5	2000	2000	3

5 rows × 37 columns

Now let's plot the distribution for all the numerical characteristics.

In [10]:
```
df_num.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8);
```






```
# Correlation matrix of numerical features
corr_matrix = train_df.select_dtypes(include=['int64', 'float64']).corr()

# Heatmap of correlations
plt.figure(figsize=(15, 10))
sns.heatmap(corr_matrix, cmap='coolwarm', annot=False, linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
```

In [11]:

```
import numpy as np

def split_dataset(dataset, test_ratio=0.30):
  test_indices = np.random.rand(len(dataset)) < test_ratio
  return dataset[~test_indices], dataset[test_indices]

train_ds_pd, valid_ds_pd = split_dataset(dataset_df)
print("{} examples in training, {} examples in testing.".format(
    len(train_ds_pd), len(valid_ds_pd)))
```

```
1010 examples in training, 450 examples in testing.
```

List of TensorFlow Decision Forests models:

- tensorflow_decision_forests.keras.RandomForestModel: Random forest model.
- tensorflow_decision_forests.keras.GradientBoostedTreesModel: Gradient enhancement machine model.
- tensorflow_decision_forests.keras.CartModel: Decision tree model.
- tensorflow_decision_forests.keras.DistributedGradientBoostedTreesModel: Distributed Gradient Strength Enhancement Machine Model.

In [12]:
```
label = 'SalePrice'
train_ds = tfdf.keras.pd_dataframe_to_tf_dataset(train_ds_pd, label=labe
l, task = tfdf.keras.Task.REGRESSION)
valid_ds = tfdf.keras.pd_dataframe_to_tf_dataset(valid_ds_pd, label=labe
l, task = tfdf.keras.Task.REGRESSION)
```

In [13]:
```
tfdf.keras.get_all_models()
```

Out[13]:
```
[tensorflow_decision_forests.keras.RandomForestModel,
 tensorflow_decision_forests.keras.GradientBoostedTreesModel,
 tensorflow_decision_forests.keras.CartModel,
 tensorflow_decision_forests.keras.DistributedGradientBoostedTreesMod
el]
```

4.2. Data preprocessing

```
df.head()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 80 columns):
Id              1459 non-null int64
MSSubClass      1459 non-null int64
MSZoning        1455 non-null object
LotFrontage     1232 non-null float64
```

```
df.describe()
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearI
count	1459.000000	1459.000000	1232.000000	1459.000000	1459.000000	1459.000000	1459.00(
mean	2190.000000	57.378341	68.580357	9819.161069	6.078821	5.553804	1971.35
std	421.321334	42.746880	22.376841	4955.517327	1.436812	1.113740	30.39(
min	1461.000000	20.000000	21.000000	1470.000000	1.000000	1.000000	1879.00(
25%	1825.500000	20.000000	58.000000	7391.000000	5.000000	5.000000	1953.00(
50%	2190.000000	50.000000	67.000000	9399.000000	6.000000	5.000000	1973.00(
75%	2554.500000	70.000000	80.000000	11517.500000	7.000000	6.000000	2001.00(
max	2919.000000	190.000000	200.000000	56600.000000	10.000000	9.000000	2010.00(

The description of the data shows that the data all have positive values (>0), so they can be accurate in terms of values. But there are still many data fields that hold blank or irrelevant data to get rid of.

```
df.isnull().sum()
```

```
Id                0 HalfBath           0 GarageQual        78
MSSubClass        0 BedroomAbvGr       0 GarageCond        78
MSZoning          4 KitchenAbvGr       0 PavedDrive         0
LotFrontage     227 KitchenQual        1 WoodDeckSF         0
LotArea           0 TotRmsAbvGrd       0 OpenPorchSF        0
Street            0 Functional         2 EnclosedPorch      0
Alley          1352 Fireplaces         0 3SsnPorch          0
LotShape          0 FireplaceQu      730 ScreenPorch        0
LandContour       0 GarageType        76 PoolArea           0
Utilities         2 GarageYrBlt       78 PoolQC          1456
LotConfig         0 GarageFinish      78 Fence           1169
LandSlope         0 GarageCars         1 MiscFeature     1408
Neighborhood      0 GarageArea         1 MiscVal            0
```

Because it is a housing data file, each data field has additional specific classification data, of which there is a lot of blank and unnecessary data

```
df = df.dropna(axis=1)
print(df)
```

```
1442       Partial
1443       Partial
1444        Normal
1445        Normal
1446        Normal
1447        Normal
1448        Normal
1449        Normal
1450        Normal
1451        Normal
1452       Abnorml
1453        Normal
1454        Normal
1455       Abnorml
1456       Abnorml
1457        Normal
1458        Normal

[1459 rows x 47 columns]
```

First, in the data file containing data fields with null values, we need to determine whether that data is really needed and remove it from the data file. After deleting columns that contain null values, the data file is left with 47 attributes and 1459 records.

```
50    #Create a Random Forest
51    rf = tfdf.keras.RandomForestModel(task = tfdf.keras.Task.REGRESSION)
52    rf.compile(metrics=["mse"]) # Optional, you can use this to include a list of eval metrics
```

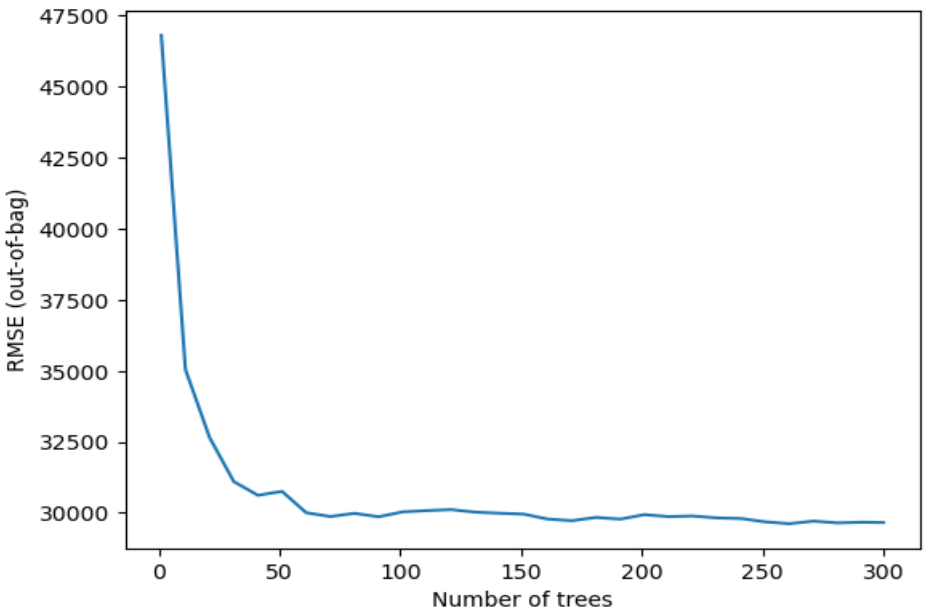
4.3. Evaluation of the performance scale

In this case, using the REGRESSION job indicates that the problem model will be used to predict the continuous value (regression). After successful model creation, during the "compile" process will specify training parameters and measurements to evaluate the performance of the model. The common measure used in regression problems to measure the difference between the predicted value and the actual value is "mse" (Mean Squared Error)

```
#Train the model
rf.fit(x=train_ds)
```

Once the model has been created, it is trained using "train_ds" input. This training data will be used to adjust the parameters of the Random Forest model so that the model can correctly predict the outputs corresponding to the inputs.

```
logs = rf.make_inspector().training_logs()
plt.plot([log.num_trees for log in logs], [log.evaluation.rmse for log in logs])
plt.xlabel("Number of trees")
plt.ylabel("RMSE (out-of-bag)")
plt.show()
inspector = rf.make_inspector()
inspector.evaluation()
evaluation = rf.evaluate(x=valid_ds,return_dict=True)
```



The chart above shows the change in RMSE (Root Mean Square Error) values when adding or removing the number of trees in the model. Adding more trees can help add performance but can also cause overfitting on the training dataset

RMSE is a measurement that evaluates the degree of difference between predicted values and actual values. In this case, RMSE was evaluated from the training log of the Random Forest model. The lower the RMSE score, the more accurate the prediction model. If there is a specific point of the number of trees where the RMSE value reaches its lowest or stable level, we can determine the best number of trees for our model

```
Out[18]:
        Evaluation(num_examples=1010, accuracy=None, loss=None, rmse=29660.363022492173, ndcg=None, auc
        s=None, auuc=None, qini=None)
```

The number of examples in the test dataset used to evaluate the model is 1010. With the accuracy index of the model is None, it means that the model is not evaluated based on accuracy. Similarly, the loss value is also None.

The value of the model's Root Mean Square Error (RMSE) on the test dataset is RMSE=29660.363022492173. In addition, the sub-indicators all include None results, which means that the model is not evaluated based on those indicators. RMSE value =

29660.363022492173 shows that the Decision Forests model has relatively low accuracy in predicting house prices.

```
inspector.variable_importances()["NUM_AS_ROOT"]
```

```
[("OverallQual" (1; #62), 121.0),
 ("GarageCars" (1; #32), 49.0),
 ("ExterQual" (4; #22), 40.0),
 ("Neighborhood" (4; #59), 35.0),
 ("GrLivArea" (1; #38), 21.0),
 ("GarageArea" (1; #31), 15.0),
 ("BsmtQual" (4; #14), 7.0),
 ("YearBuilt" (1; #76), 5.0),
 ("KitchenQual" (4; #44), 4.0),
 ("TotalBsmtSF" (1; #73), 3.0)]
```

This is the result of a variable in order of importance, e.g. OverallQual is position 1 in the list of variables. The number 62 in parentheses is the index of this variable in the dataset or model. The importance of this variable is measured by the value 121.0.

OverallQual: This is often one of the most important variables when predicting home prices. The high level of importance proposes that the model considers Total Quality to be an important factor in predicting home prices. The model can use this variable as a starting point to divide the data into smaller groups based on the overall quality of the home.

GarageCars: This is also an important factor, as the number of cars can affect home prices. A garage can be used to store vehicles, and the number of vehicles it can hold can affect the value of the home.

ExterQual (Exterior Quality): The exterior quality of a home also has a significant effect on its value. The model can use this information to classify homes into groups of different exterior qualities, which in turn helps predict their value.

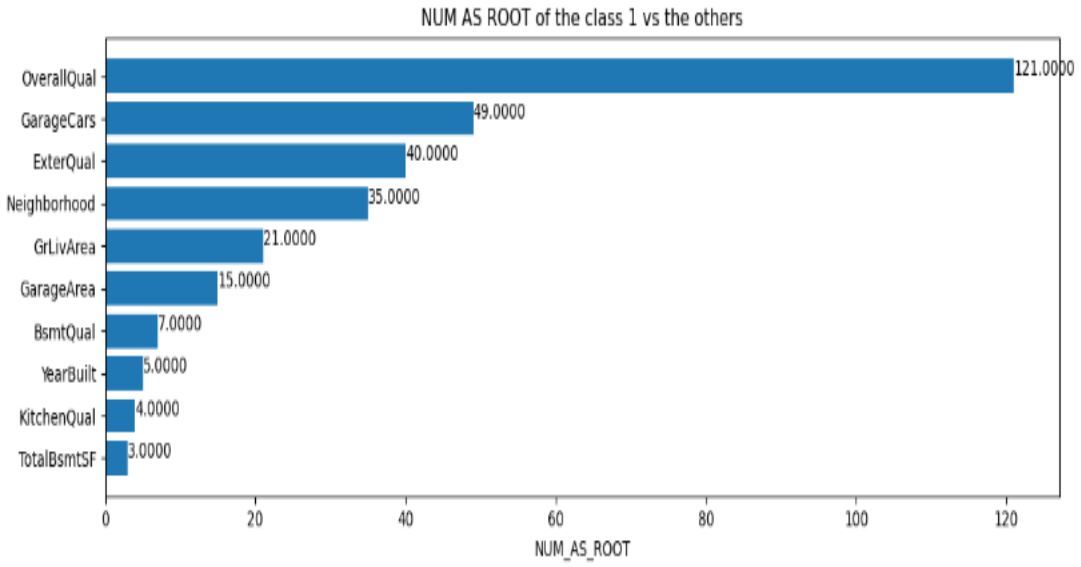
Neighborhood: The area that the house is located in can greatly affect the price of the house. The model can use this information to identify areas of higher and lower value, thereby predicting the value of homes in that area.

And other variables such as GrLivArea, GarageArea, BsmtQual, YearBuilt, KitchenQual, and TotalBsmtSF also have an effect on home prices and are considered important by the model in the prediction process.

```
for importance, patch in zip(feature_importances, bar.patches):
  plt.text(patch.get_x() + patch.get_width(), patch.get_y(), f"{importance:.4f}", va="top")

plt.xlabel(variable_importance_metric)
plt.title("NUM AS ROOT of the class 1 vs the others")
plt.tight_layout()
plt.show()
```

We create a graph, where the vertical axis is the important variables in the prediction process and the horizontal axis is their importance scale. As can be seen, the overall quality of the house (OverallQual) is the most important variable; garage, exterior quality or neighbors are the next most important group of variables; The land area, garage area, the quality of the foundation of the house, the time to build the house, the quality of the kitchen or the total underground area are not so important.



* Download test data:

```
test_file_path = "D:/QTDL with Spark/test.csv"
test_data = pd.read_csv(test_file_path)
ids = test_data.pop('Id')

test_ds = tfdf.keras.pd_dataframe_to_tf_dataset(
    test_data,
    task = tfdf.keras.Task.REGRESSION)

preds = rf.predict(test_ds)
output = pd.DataFrame({'Id': ids,
                       'SalePrice': preds.squeeze()})

output.head()
```

The RandomForest (rf) model is used to predict home prices on test data. A new dataframe is created with 2 columns, 'Id' which are the values removed from the test data and 'SalePrice' which are the predictive values calculated from the RandomForest model. After more than 1 second, the first 5 lines of that dataframe are shown, showing the predicted price of each house with different variables.

	Id	SalePrice
0	1461	123554.718750
1	1462	153939.062500
2	1463	176793.765625
3	1464	183828.296875
4	1465	193644.484375

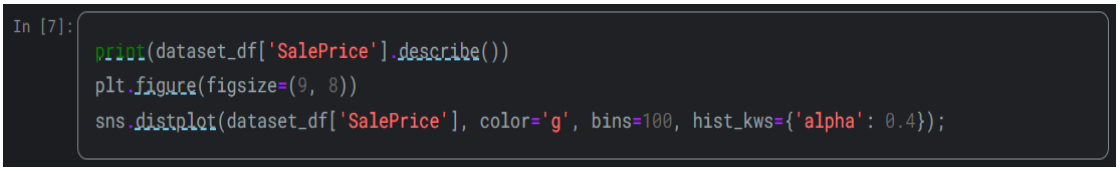
4.4. Installation parameters and environment

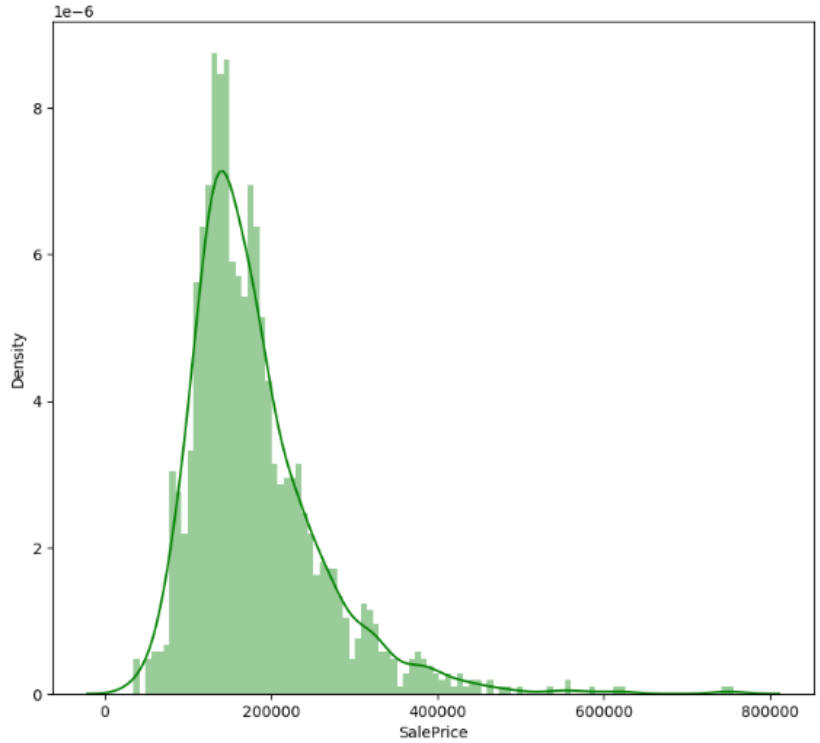
The installation environment for the "House price prediction" problem typically includes common tools and libraries used for data processing and associative rule mining from the procurement dataset. Here are the components needed for this problem:

Programming language: The team uses Python because it provides powerful libraries for data processing and analysis.

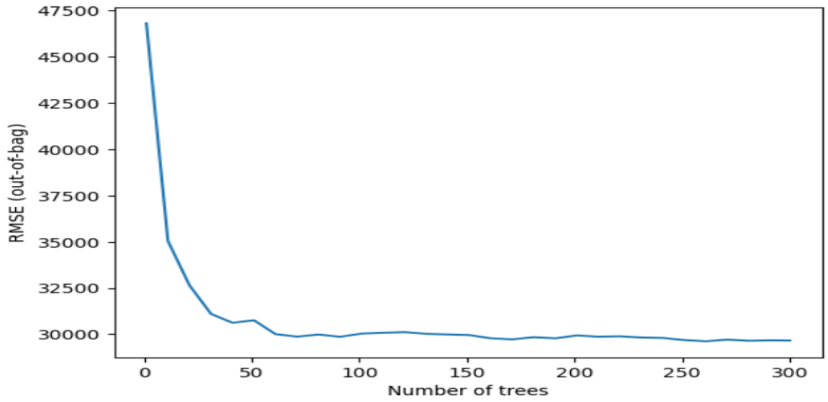
The team loads the libraries needed for data analysis and data visualization using charts such as: pandas, seaborn, matplotlib, tensorflow

The Seaborn Library is a Python library for data visualization based on Matplotlib. It provides a high-level interface to create statistical charts and visualize data easily and efficiently, including line charts, column charts, pie charts,... In this article, the seaborn library is provided for plotting displot charts.





The Matplotlib library is provided for creating line charts:



Parameters used in the "House price prediction" problem

Dataset: This is a data file consisting of 80 different data fields, each field represents a parameter of the house such as the quality of the house, the area of the house, the street, the number of rooms ,...

Important variable: Expresses different factors with varying degrees of importance, the more important the variable, the greater the impact on its model.

4.5. Base method

The code uses Random Forest, a machine learning algorithm to predict house prices. Random Forest works based on combining many simple decision trees.

Each decision tree is trained on a subset of data and makes independent predictions. The final prediction of the model is the aggregation of the predictions from the component decision trees.

Random Forest can be considered an ensemble learning method because it combines many individual weak models to create a stronger model.

Some techniques to improve the efficiency of the Random Forest model:

- Use feature selection techniques to select the most important features for the model.
- Use data augmentation techniques to increase dataset size and minimize the risk of overfitting.
- Use ensemble techniques to combine multiple Random Forest models together to create a more robust model.

4.6. Analysis and comparison of results

Compare the results of two Linear Regression models and TensorFlow Decision Forests

Train Time:

- Linear regression: Often trains faster than TensorFlow Decision Forests due to its simplicity.
- TensorFlow Decision Forests: Often trains slower due to their complexity and need to process more data.

Accuracy:

- Linear regression: Lower accuracy than TensorFlow Decision Forests when the relationships between variables are non-linear.
- TensorFlow Decision Forests: Capable of learning complex non-linear relationships, thus being able to achieve greater accuracy.

IV. Conclusion

Depending on the size of the data file and the variables to be processed, you can choose between the two models

Model selection:

- Linear regression: Suitable for small data sets, can be used for variables that have a linear relationship with home sales price.
- TensorFlow Decision Forests: Suitable for large data sets, which can be used for variables that have a non-linear relationship to home sales price, as well as for handling complex interactions between variables.

It is recommended to use the TensorFlow Decision Forests model to predict house prices with the above dataset

Reason:

- There are many variables that have a non-linear relationship to the selling price of a home.
- TensorFlow Decision Forests are capable of handling complex interactions between variables.