SEEM2460

Introduction to Data Science

Studying Key Pricing Factors in US Used Car Market

Project Report

Cheng Yu Shing	1155158488
Lo Hoa Tsun	1155158762
Lam Chi Fung	1155159343

Source Code Available at

https://github.com/LilAiluropoda/SEEM2460

Index

Introduction

Ambiguous pricing has been a long-existing problem in the used car market in the United States. Differ from factory new vehicles, the pricing of second-hand vehicles are driven by additional factors like mileages, conditions and modifications. These factors are often subjective and qualitative, resulting in deviation of prices. Therefore, it can cause potential loss of the customers as they may overpaying for the used car. According to online studies, only 21% of consumers find the current pricing mechanisms satisfactory.

With these in mind, the project aims to remove the ambiguity in used car pricing by applying various parametric and non-parametric techniques to identify key pricing factors, including decision tree models, correlation analysis and hypothesis testing. By recognizing important pricing factors and their corresponding significance, the project aims to enhance market efficiency and facilitate fairer transactions. Eventually, the project also targets to deliver a stable decision tree model which can predict the used car price using the aforementioned key factors.

Related Works

Decision Tree

A decision tree is a graphical representation resembling a flowchart-like tree, in which each internal node signifies a test conducted on a specific attribute. The branches of the tree correspond to the possible outcomes of the test, while each leaf node represents the assigned class label. When a tuple X is presented, the attribute values of the tuple are evaluated against the decision tree. By following a path from the root to a leaf node, we can determine the predicted class for the tuple (Sharma Kumar, 2016). A Regression Tree is a type of decision tree where the target variable can take continuous values rather than discrete class labels.

Gradient Boosting Tree

Gradient boosting trees are ensemble techniques that sequentially construct decision tree learners by fitting the gradients of the residuals from previously built tree learners. The tree building process begins with a single node and iteratively introduces branches until a specific criterion is fulfilled. When adding branches to each leaf node, the objective is to maximize the reduction in loss following the split (Anghel, A., 2018). Some common gradient boosting

tree algorithms include XGBoost, LightGBM, and CatBoost. These algorithms are widely used for their performance and efficiency in handling gradient boosting tasks.

Dataset

To explore and understand the relationship between various car key factors and second-hand car market prices in the US, we take the dataset “Vehicle Sales Data” created by Syed Anwar in Kaggle (Syed, A., 2024).

This dataset included car manufacturing from 1982 to 2015, and a total of over 500000 records, providing a comprehensive overview of various car transactions in the past 40 years. The dataset has a total of 16 columns. Columns in the dataset indicate the car manufacturing details, such as car brand, car model, manufacturing year, etc. It also describes the car's condition, such as mileage coverage by the vehicle. As well as, including the sales details such as sale date, selling price, etc. The descriptions and corresponding data type for each columns are described in the following table.

Column Description

Column	Data Type	Description
year	integer	The manufacturing year of the vehicle.
make	string	The brand or manufacturer of the vehicle.
model	string	The specific model of the vehicle.
trim	string	Additional designation (if any) for the vehicle model.
body	string	The body type of the vehicle.
transmission	string	Type of transmission in the vehicle.
vin	string	Vehicle Identification Number for uniquely identifying the vehicle.
state	string	The registration state of the vehicle.
condition	integer	Condition of the vehicle, rated on a scale from 1 to 50.
odometer	integer	The miles traveled by the vehicle.
color	string	The exterior color of the vehicle.
interior	string	The interior color of the vehicle.
seller	string	The seller of the vehicle.
mmr	integer	Manheim Market Report, indicating the average market value of the vehicle.
sellingprice	integer	The final selling price of the vehicle.
saledate	string	The date and time when the vehicle was sold.

Table 1: Dataset’s Column Description

Preprocessing

Prior to model selection and training, a thorough examination of the dataset is performed to identify and eliminate any missing or inconsistent columns. Subsequently, irrelevant columns are eliminated from the dataset, and a test is conducted to identify and address duplicate rows. Last but not least, to facilitate a more efficient training process, some data transformations will be performed on the dataset.

Note that data normalization or standardization is not required for this project as gradient boosting models as the tree-building process in gradient boosting models inherently handles variations in feature scales, and the ensemble nature of the model helps to mitigate the effect of outliers.

Removing Invalid Data

```
# Remove rows with missing values
df.dropna(inplace=True)
# Remove rows with duplicate values
df.drop_duplicates(inplace=True)
```

To commence with, the built-in functions of **pandas** package **dropna** and **drop_duplicates** will be incorporated to remove rows with missing or duplicated values. Note that as the size of dataset is large and there are a lot of columns with string (categorical) datatype, it is suggested that removing rows with missing values is preferred over filling them with mean, as it may cause bias in the data distribution.

Removing Unrelated Data

```
# Remove vin identifier, state
df.drop(columns=["vin", "state"], inplace=True)
```

Secondly, unrelated columns like *vin* and *state* will removed from the dataset as they are the indices for uniquely identifying the used vehicle, which is not helpful in pricing.

Date Time Transformation

```python
def transform_datetime(df: pd.DataFrame, col: str):
    tmp = col + "_utc"
    df[col + "_utc"] = pd.to_datetime(df[col], utc=True)
    df[col + "_year"] = df[tmp].dt.year
    df[col + "_month"] = df[tmp].dt.month
    df[col + "_day"] = df[tmp].dt.day

    # Set date as categorical variable
    df[col + "_year"] = df[col + "_year"].astype("category")
    df[col + "_month"] = df[col + "_month"].astype("category")
    df[col + "_day"] = df[col + "_day"].astype("category")

    # Remove saledate columns
    df.drop(columns=[col, tmp], inplace=True)
    return df
```

As the *saledate* column contains datetime data in string format, it is necessary to transform the datatime data such that it can be better understood by the machine learning models. For instance, the above function will convert datetime in string into datetime data type, then break the data down into 3 columns representing year, month and day of the transaction. As we do not want to introduce any hierarchy or ranking in the datetime data, the 3 columns will be transformed into categorical data type.

Example Input	{"saledate": "Thu Jan 15 2015 04:30:00 GMT-0800 (PST)"}
Example Output	{"saledate_year": 2015, "saledate_month": 1, "saledate_day: 15"}

Categorical Data Encoding

```python
def transform_categorical(df, cols: list[str]):
    mapper = {}
    for col in cols:
        mapped_col, tmp_map = pd.factorize(df[col])
        mapper[col] = tmp_map
        # index starts at 1
        df[col] = pd.Categorical(mapped_col + 1)
    return df, mapper
```

Similarly, we will also convert other columns with string data into numerical format as the machine learning models may not accept string as input. The strings will be encoded and represented as classes with numerical labels. This allows for the proper representation of categorical variables in a numerical form that can be effectively utilized by the models. For this project, label encoding is used for the encoding process.

Example Input	{”model”: [“S60”, “M5”, “A4”]}
Example Output	{”model”: [1, 2, 3]}

Analysis

Methodology

The project will be carried out in the following steps. Firstly, the project will apply statistical methods like correlation analysis to identify collinearity within the dataset, as collinearity may impact the model performance and project result. Subsequently, hypothesis testing on feature independence against the target variable will be also performed to see if any features are unrelated to the prediction. After that, two gradient boosting models, namely LightGBM and CatBoost, will be run on the full dataset to obtain feature importance maps. Combining the obtained information above, we will remove any collinearity or unrelated variables from the dataset and re-run the two models on the optimized dataset. We will compare the optimized models and the base models (which use the whole dataset for training) to see if any improvement in model performance. The project outcome is successful if improvement is observed and the feature importance maps generated by the optimized models will be used to identify key pricing factors. If no improvement is observed, the project will try to identify possible causes and suggest corresponding solutions.

Correlation Analysis

Correlation Analysis can be used to detect the existence of collinearity among different pairs of variables. The correlation of each pair of variables in the data will describe the collinearity. It will be in the range of [-1, 1]. A positive correlation value indicate a positive relationship while a negative value implies a negative relationship. The magnitude of the value refers the degree of correlation within the variable pair. The formula of correlation is defined as,

$$r_{xy} = \frac{\sum(x_i - \overline{x})\sum(y_i - \overline{y})}{\sqrt{\sum(x_i - \overline{x})^2 \sum(y_i - \overline{y})^2}}$$

Interpretation

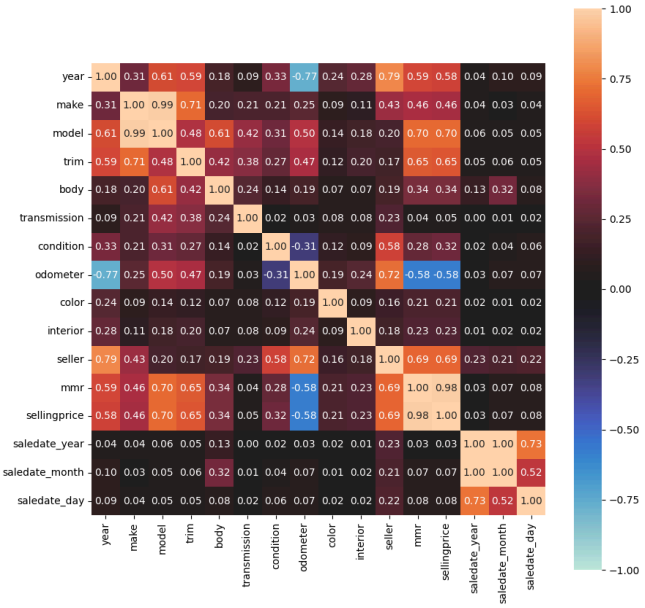


Figure 1: Correlation Matrix of All Features in The Dataset

Figure 1 shows a correlation matrix of all features used in the model training (including the target variable). From the graph, 2 pairs of value are highly related, achieving 0.99 and 1.00 correlation. The first pair is *make* and *model* and the second pair is *saledate_year* and *saledate_month*. Therefore, we should consider eliminating these highly-correlated pairs for improving the performance of model. However, note that correlation matrix assumed every relationship is linear, but in reality the variables of sales car data might be more complex than a pure linear relationship. As such, validation is required before removing the features. For this project, the feature importance maps from the gradient boosting models will be used for validation.

Hypothesis Testing

Hypothesis Testing can be applied to study if the two variables are independent of each other or of the same distribution. For this project, all features are tested against the target variables to study the feature is helpful for predicting target variable. F-Test in this hypothesis testing as the target variable is continuous but not categorical, so Chi-Square test cannot be applied. The test-statistic for F-test is calculated as,

$$F = \frac{s_1^2}{s_2^2}$$

Where s2 is the variance of the variable,

$$s^2 = \frac{\sum(x - \overline{x})^2}{n - 1}$$

As aforementioned, the null hypothesis will be the feature is independent of target variable, where the alternative hypothesis will be the feature is dependent of the target variable.

Interpretation

	column	t-statistic	p-value
11	mmr	13953640.352099	0.000000
0	year	238111.538874	0.000000
7	odometer	236218.763946	0.000000
6	condition	53029.603439	0.000000
9	interior	13985.954032	0.000000
2	model	5259.518110	0.000000
8	color	3699.237541	0.000000
1	make	3642.024082	0.000000
4	body	2190.856329	0.000000
10	seller	1157.036895	0.000000
3	trim	1026.533090	0.000000
5	transmission	1007.108100	0.000000
12	saledate_year	412.305184	0.000000
13	saledate_month	28.243393	0.000000
14	saledate_day	22.553962	0.000002

Figure 2: Result of F-Test

Figure 2 shows the result of hypothesis testing. From the graph, as the p-values of all test are small, we should reject the null hypothesis and all variables have predictive relationship with the target variable. Note that the test-statistic implies the strength of relationship and *mmr* has the strongest relationship with the target variable, while *saledate_day* has the weakest relationship. The test result will be combined with other information (e.g. Correlation Analysis and Feature Importance Maps) for deciding which features to be excluded from the dataset.

Feature Importance Map

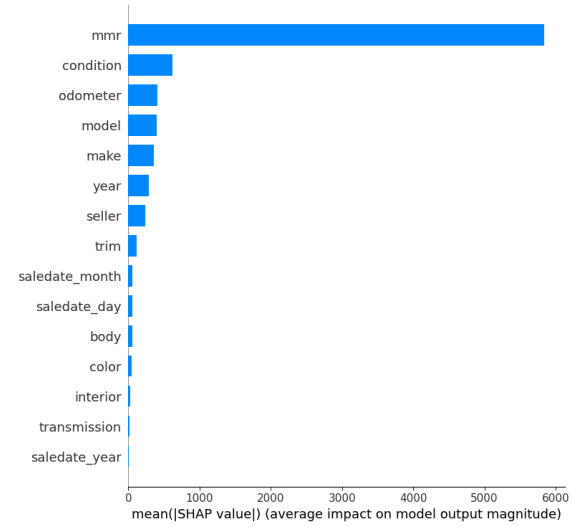


Figure 3: Feature Importance Map of LightBGM Base Model

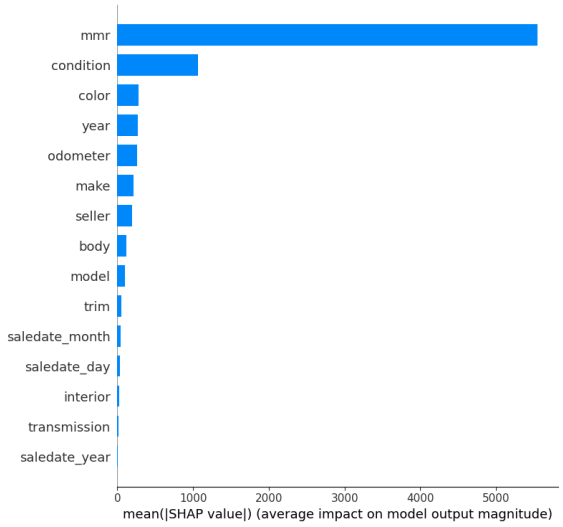


Figure 4: Feature Importance Map of CatBoost Base Model

Figure 3 and 4 show the feature importance maps of the LightGBM model and CatBoost model trained on the full dataset. From the graphs, it is observed that *saledate_year* is not used in the both models, which validates the collinearity claim from correlation analysis. Therefore, *saledate_year* should be removed from the dataset. However, as *make* and *model* both contributed to the prediction of the target variable, they should be kept and it might imply non-linear relationship within the variable pair.

Additionally, it is also observed that *interior* and *transmission* have near zero contribution in the training process. As shown in the F-test result where *interior* and *transmission* have significant smaller test-statistic compared to variables like *mmr* and *condition*, they will be also excluded in the optimized dataset.

As a result, the following table shows the kept features in the optimized dataset.

Features	Keep?	Features	Keep?
year	Y	odometer	Y
make	Y	color	Y
model	Y	interior	
trim	Y	seller	Y
body	Y	mmr	Y
transmission		sellingprice	Y
vin		saledate_year	
state		saledate_month	Y
condition	Y	saledate_day	Y

Table 2: Kept Features in The Optimized Dataset

Model Selection

Gradient Boosting models will be applied for this project because of the following reasons. Firstly, they have been proven to be extremely efficient in processing tabular data by recent studies (Leo Edouard, 2022), outperforming the commonly used deep learning models (Ravid Amital, 2022). Secondly, Gradient Boosting models as one of the decision tree models, are generally more explainable than deep learning models with graphs like result tree graphs and feature importance maps. Last but not least, Gradient Boosting models are much more lightweight than deep learning models, which are suitable for the scope of this project, where the project members have only very limited computational resources. In this project, LightGBM and CatBoost will be used as these implementations of gradient boosting models have outstanding training speed and accuracy.

LightGBM

Algorithm 1: Histogram-based Algorithm

Input: I: training data, d: max depth
Input: m: feature dimension
$nodeSet \leftarrow \{0\}$ ▷ tree nodes in current level
$rowSet \leftarrow \{\{0, 1, 2, ...\}\}$ ▷ data indices in tree nodes
for $i = 1$ ***to*** d **do**
  for $node$ ***in*** $nodeSet$ **do**
    usedRows $\leftarrow rowSet[node]$
    for $k = 1$ ***to*** m **do**
      $H \leftarrow$ new Histogram()
      ▷ Build histogram
      for j ***in*** *usedRows* **do**
        bin $\leftarrow I$.f[k][j].bin
        H[bin].y $\leftarrow H$[bin].y + I.y[j]
        H[bin].n $\leftarrow H$[bin].n + 1
      Find the best split on histogram H.
      ...
  Update $rowSet$ and $nodeSet$ according to the best split points.
  ...

Figure 5: Histogram-Based Algorithm

Algorithm 2: Gradient-based One-Side Sampling

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: $loss$: loss function, L: weak learner
models $\leftarrow$ {}, fact $\leftarrow \frac{1-a}{b}$
topN $\leftarrow$ a $\times$ len(I) , randN $\leftarrow$ b $\times$ len(I)
for $i = 1$ ***to*** d **do**
  preds $\leftarrow$ models.predict(I)
  g $\leftarrow loss(I$, preds), w $\leftarrow$ {1,1,...}
  sorted $\leftarrow$ GetSortedIndices(abs(g))
  topSet $\leftarrow$ sorted[1:topN]
  randSet $\leftarrow$ RandomPick(sorted[topN:len(I)], randN)
  usedSet $\leftarrow$ topSet + randSet
  w[randSet] $\times =$ fact ▷ Assign weight $fact$ to the small gradient data.
  newModel $\leftarrow$ L(I[usedSet], $-$ g[usedSet], w[usedSet])
  models.append(newModel)

Figure 6: Gradient-Based One-Side Sampling

LightGBM is an efficient gradient-boosting framework that is designed to provide high performance and efficiency in training and prediction tasks. In LightGBM, the training dataset is divided into bins using a histogram-based algorithm (Figure 5). This binning process is an important step in the training process and is used for iteration, gain calculation, and data slicing. It allows faster computation and reduced memory usage compared to traditional algorithms that use exact splits. It leverages the precomputed histograms to perform efficient calculations during training, making LightGBM suitable for large-scale datasets.

Additionally, LightGBM also incorporates Gradient-based One-Side Sampling (Figure 6), the data points that have larger gradients will be weighted higher and those data points with smaller gradients will be removed randomly. As a result, the computational time will be greatly reduced.

CatBoost

Algorithm 1: Ordered boosting

input : $\{(\mathbf{x}_k, y_k)\}_{k=1}^{n}$, I;
$\sigma \leftarrow$ random permutation of $[1, n]$;
$M_i \leftarrow 0$ for $i = 1..n$;
for $t \leftarrow 1$ **to** I **do**
  for $i \leftarrow 1$ **to** n **do**
    $r_i \leftarrow y_i - M_{\sigma(i)-1}(\mathbf{x}_i)$;
  for $i \leftarrow 1$ **to** n **do**
    $\Delta M \leftarrow$
     $LearnModel((\mathbf{x}_j, r_j) :$
     $\sigma(j) \leq i)$;
    $M_i \leftarrow M_i + \Delta M$;
return M_n

Figure 7: Algorithm of Ordered Boosting

CatBoost is another high-performing gradient-boosting framework, but it differs from LightGBM in the use of symmetric trees as the structures, which means that each split is on the same attribute, and the lowest loss is selected and applied to all the level's nodes. Another difference is CatBoost uses Ordered Boosting (Figure 7) to reduce the problems of overfitting, which considers the natural ordering of categorical feature values during training instead of labeling, which enables better performance in processing categorical variables.

Hyper-Parameter Tuning

Hyperparameter tuning will be applied to determine the optimal set of hyperparameters for Gradient Boosting models. For this project, Optuna, a popular hyperparameter tuning framework, will be used for searching the best hyperparameters.

Differing from traditional grid search, Optuna employs state-of-the-art algorithms to efficiently sample hyperparameters and prune unpromising trials (Optuna, 2018). The efficiency and automation of Optuna makes it can be used with different machine learning frameworks and algorithms, which provides flexibility in the optimization process.

The following shows the optimal hyperparameters for the base models and optimized models.

LightGBM (Base)

```
{'learning_rate': 0.1972993910502014,
'lambda_l1': 0.3067775578963962,
'lambda_l2': 1.8755878221604316e-07,
'max_depth': 9,
'min_data_in_leaf': 200,
'max_bin': 246,
'num_leaves': 715,
'feature_fraction': 0.6495195538416207,
'bagging_fraction': 0.7610518710482679,
'bagging_freq': 4,
'verbosity': 1,
'num_threads': 8,
'device_type': 'cpu',
'early_stopping_round': 10,
'objective': 'regression',
'metric': 'rmse'}
```

CatBoost (Base)

```
{'learning_rate': 0.095415049020046,
 'depth': 10,
 'subsample': 0.26558875912943886,
 'colsample_bylevel': 0.9627733982178004,
 'min_data_in_leaf': 71,
 'cat_features': ['make', 'model', 'trim', 'body', 'transmission',
 'color', 'interior', 'seller', 'saledate_day', 'saledate_month', 'saledate_year'],
 'verbose': 200,
 'thread_count': 8,
 'task_type': 'CPU',
 'iterations': 200}
```

LightGBM (Optimized)

```
{'learning_rate': 0.2430325495420273,
'lambda_l1': 0.05186940186117274,
'lambda_l2': 0.0002992319399642732,
'max_depth': 8,
'min_data_in_leaf': 200,
'max_bin': 254,
'num_leaves': 1510,
'feature_fraction': 0.8834372471182904,
'bagging_fraction': 0.9312854810099485,
'bagging_freq': 1,
'verbosity': 1,
'num_threads': 8,
'device_type': 'cpu',
'early_stopping_round': 10,
'objective': 'regression',
'metric': 'rmse'}
```

CatBoost (Optimized)

```
{'learning_rate': 0.09512264956219604,
'depth': 10,
'subsample': 0.4329770399887209,
'colsample_bylevel': 0.6864897987921085,
'min_data_in_leaf': 3,
'cat_features': ['make', 'model', 'trim', 'body', 'color', 'seller', 'saledate_month',
 'saledate_day'],
'verbose': 200,
'thread_count': 8,
'task_type': 'CPU',
'iterations': 200}
```

Model Training

The same training setting will be applied to both base models and optimized models for accurate results. The models will train for 200 iterations, while the dataset will be split into training set, validating set and testing set in the ratio of 60%, 20%, and 20% respectively. The data are randomly sampled during the splitting process.

Objective Function

Root Mean Square Error (L2 Loss) will be used as objective function. It measures the average difference between the model's predicted values and the actual values. The error can be calculated by,

$$\frac{1}{N}\sum_{i=1}^{N}\sqrt{(y_{\text{true}}-y_{\text{pred}})^2}$$

Early Stopping

To enhance the efficiency of the training process, early stopping will be applied. Should the model’s performance does not improve in n consecutive training iterations, the training will terminate early. For this project, the terminating threshold is set as 10 iterations.

Project Outcome

Evaluation Metrics

Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) will be used to evaluate the performance of the trained models. Since RMSE has already been introduced in the Model Training section, only MAE will be discussed here. Note that the R-Sqaured score is not used as it is inappropriate for evaluating the regression performance of gradient boosting models since the models are non-linear and therefore violate the assumption of the metric.

Mean Absolute Error (MAE)

Mean Absolute Error (L1 Loss) calculates the average absolute differences between the predicted value and the actual target value. It is defined as,

$$\frac{1}{N}\sum_{i=1}^{N}|y_{\text{true}} - y_{\text{pred}}|$$

Performance Comparison

Model	RMSE	MAE
LightGBM (Base)	1573.1321795062413	888.0662412373841
CatBoost (Base)	1691.2714393609135	938.7538500090541
LightGBM (Optimized)	**1539.2116851569008**	**887.0581880685633**
CatBoost (Optimized)	**1681.6319388968504**	**938.537002836717**

table 3: Performance of The Base Models and Optimized Models

The above table shows the performance of the base models and optimized models. It can be observed that there are improvements in the optimized models. Therefore, the optimized dataset is valid.

Feature Rank

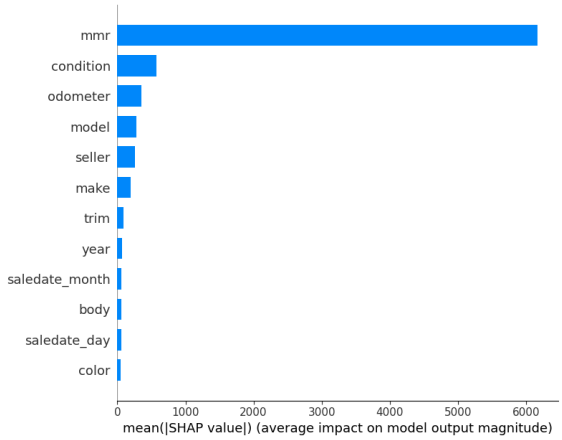


Figure 8: Feature Importance Map of LightBGM Optimized Model

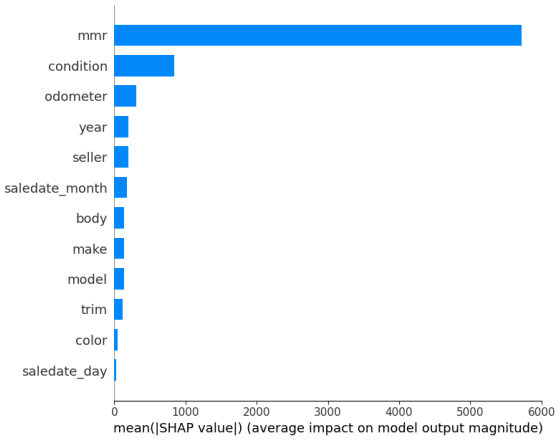


Figure 9: Feature Importance Map of LightBGM Optimized Model

Figure 8 and 9 show the feature importance maps generated by the two optimized models. As the validity of the dataset optimization has been proved in the above section, the output of the graphs is also reliable.

While there may exist some deviations between the two models' output, both models agree that *mmr* is the most influencing factor in the prediction of the selling price, followed by condition, odometer, and seller. *make* and *model* also significantly contributed to prediction, despite having different ranks in the two graphs. Features like *color* and *saledate_day* do not influence a lot in both model's prediction.

Conclusion

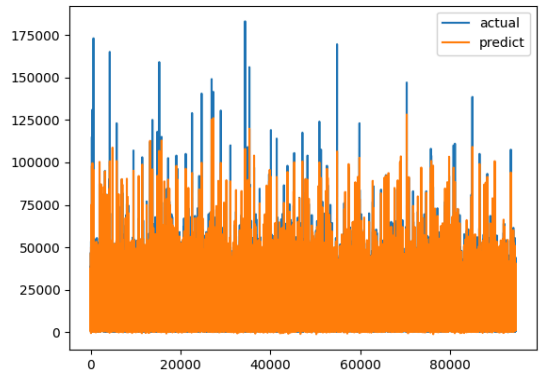


Figure 10: Performance on Testing Set of Optimized LightGBM Model

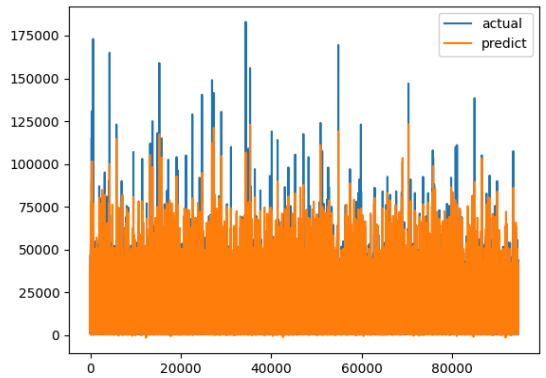


Figure 11: Performance on Testing Set of Optimized CatBoost Model

In this project, we have successfully identified unrelated and highly correlated variables and eliminated them in the optimized dataset. Additionally, key pricing factors like *mmr*, *condition*, and *odometer* have also been successfully identified and sorted according to their importance in deciding the selling price of a used car. Eventually, we have also constructed the two Gradient Boosting models with decent prediction performance which can be used to predict the selling price of a used vehicle.

However, there is room for improvement as the project only points out the significance level of each feature. To construct a clear formula for calculating the car selling price, it is suggested that more advanced models like the recently introduced Kolmogorov-Arnold Network (KAN) can be applied in future work.

Reference

Sharma, H., amp; Kumar, S. (2016). A survey on decision tree algorithms of classification in Data Mining. International Journal of Science and Research (IJSR), 5(4), 2094–2097. https://doi.org/10.21275/v5i4.nov162954

Anghel, A., Papandreou, N., Parnell, T., Palma, A.D., Pozidis, H. (2018). Benchmarking and Optimization of Gradient Boosted Decision Tree Algorithms. ArXiv, abs/1809.04559.

Syed, A. (2024). Vehicle Sales Data. Kaggle, https://www.kaggle.com/datasets/syedanwarafridi/vehicle-sales-data

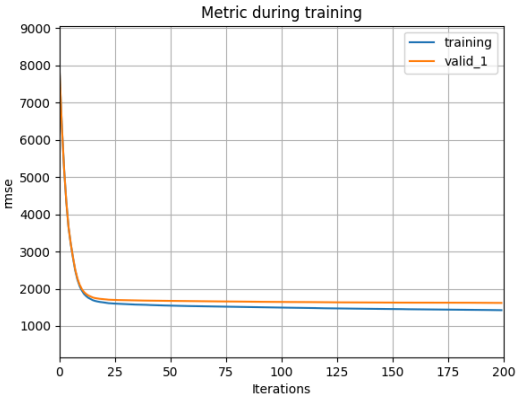
Leo, G., Edouard O., Gael, V. (2022). Why do tree-based models still outperform deep learning on tabular data?. ArXiv, abs/2207.08815

Ravid S., Amital, A. (2022). Tabular data: Deep learning is not all you need. ScienceDirect, https://www.sciencedirect.com/science/article/pii/S1566253521002360?via%3Dihub

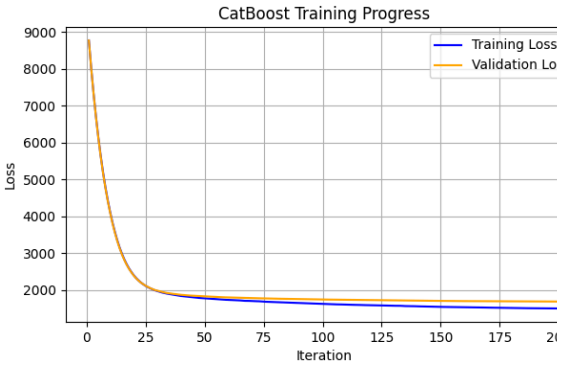
Optuna (2018). Efficient optimization algorithms. https://optuna.readthedocs.io/en/stable/tutorial/10_key_features/003_efficient_optimization_algorithms.html

Appendix

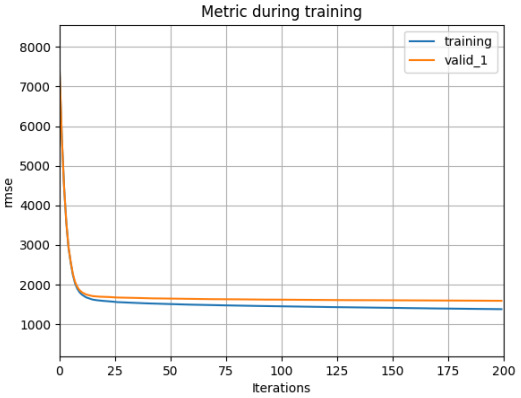
Appendix 1: Other Graphs Generated In Project



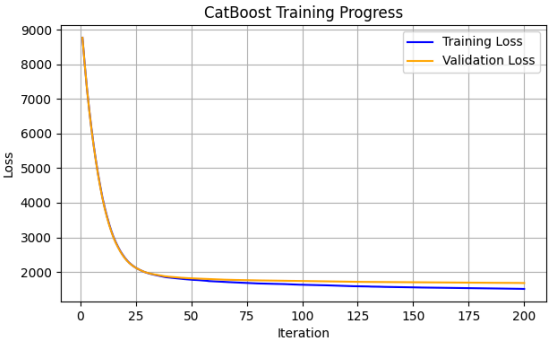
Train / Valid Loss of LightGBM (Base)



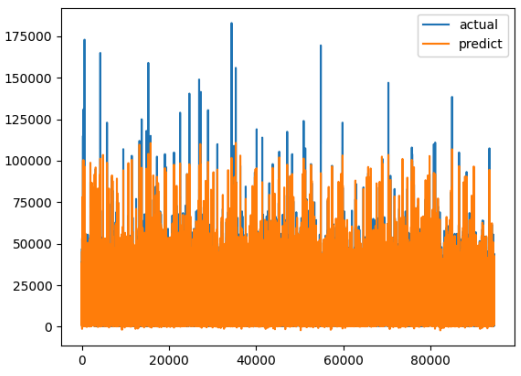
Train / Valid Loss of CatBoost (Base)



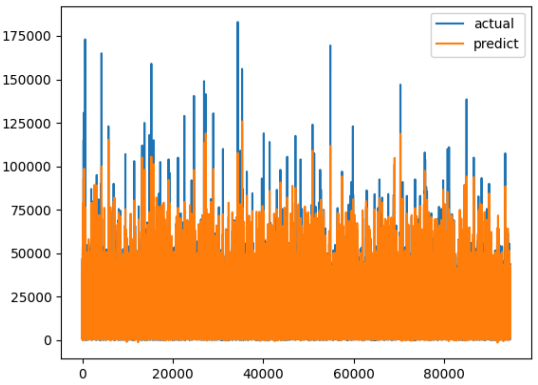
Train / Valid Loss of LightGBM (Optimized)



Train/ Valid Loss of CatBoost (Optimized)



Performance on Testing Set of Base LightGBM Model



Performance on Testing Set of Base CatBoost Model

Appendix 2: Source Code In Main Repository

Appendix 2.1: main.py

```
import preprocessing as pp
import model
import pandas as pd
import sys
import helper
import eda
import matplotlib.pyplot as plt
def main():
    # Preprocessing
    df = pd.read_csv("./data/car_prices.csv")
    helper.message("[INFO] Start preprocessing data...")
    df, mapper = pp.car_preprocessing(df)
    x_train, x_test, x_valid, y_train, y_test, y_valid = pp.generate_dataset(df,
"sellingprice", 0.2)
    # Exploratory Data Analysis
    helper.message("[INFO] Performing EDA...")
    helper.message("[INFO] Start visualizing correlation matrix...")
    eda.corr_matrix_visualize(df, "corr_matrix_visualize")
    helper.message("[INFO] Visualization completed.")
    helper.message("[INFO] Start running hypothesis testing on variable
independence")
    helper.message(eda.f_test(df, "sellingprice").sort_values(by="t-statistic",
ascending=False).info)
    # chi2_summary.style.bar("t-statistic").background_gradient("Blues",
subset="t-statistic")
    # plt.show()
    helper.message("[INFO] Test completed")
    helper.message("[INFO] EDA completed.")
    # Train LightGBM
    helper.message("[INFO] Start training (LightGBM)...")
    lgbm = model.LightGBM()
    lgbm.train(x_train, y_train, x_valid, y_valid)
    helper.message("[INFO] Training completed (LightGBM).")
    # Train CatBoost
    helper.message("[INFO] Start training (CatBoost)...")
    cbt = model.CatBoost()
    cbt.train(x_train, y_train, x_valid, y_valid)
    helper.message("[INFO] Training completed (CatBoost).")
    # Generate training report
    helper.message("[INFO] Generating Training Report (LightGBM)...")
    lgbm.training_report()
    helper.message("[INFO] Generating Training Report (CatBoost)...")
    cbt.training_report()
    # Evaluate LightGBM performance
    lgbm.eval(x_test, y_test)
    # Evaluate CatBoost performance
    cbt.eval(x_test, y_test)
    # Report LightGBM Feature Importance
    helper.message("[INFO] Generating Feature Report (LightGBM)...")
    lgbm.feature_report(x_train, y_train)
    # Report CatBoost Feature Importance
    helper.message("[INFO] Generating Feature Report (CatBoost)...")
    cbt.feature_report(x_train, y_train)
    helper.message("[INFO] Program end, exiting...")
    return 0
if __name__ == "__main__":
```

```
    sys.exit(main())
```

Appendix 2.2: eda.py

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from dython.nominal import identify_nominal_columns, associations
from scipy.stats.contingency import chi2_contingency
from sklearn.feature_selection import chi2, f_regression
from pandas.plotting import table
import dataframe_image as dfi

# Visualize correlation matrix
def corr_matrix_visualize(df: pd.DataFrame, filename: str):
    categorical_features = identify_nominal_columns(df)
    # Debug Message
    # print("categorical features: ", categorical_features)
    complete_correlation = associations(df, filename=f'graphs/{filename}.png',
figsize=(10, 10))
    df_complete_corr = complete_correlation['corr']
    df_complete_corr.dropna(axis=1, how='all').dropna(axis=0,
how='all').style.background_gradient(cmap='coolwarm', axis=None)
    return 0
# Visualize scatterplot matrix (Not Used)
def matrix_scatter_visualize(df: pd.DataFrame):
    sns.set_theme(style="ticks")
    plt.figure(figsize=(2, 2))
    pair_plot = sns.pairplot(df, hue="seller", height=2.5)
    pair_plot.savefig("pair_plot_seller.jpg")
    plt.show()
    return 0

def f_test(df: pd.DataFrame, target: str):
    # Separate Numerical and Categorical Columns
    x = df.drop(columns=[target], axis=1).columns
    # Initialize Result Table
    summary = np.empty((len(x), 3), dtype="object")

    for i, col in enumerate(x, start=len(summary)-len(x)):
        t_stat, pvalue = f_regression(df[[col]], df[target].values.reshape(-1, 1))
        summary[i, :] = [col, t_stat[0], pvalue[0]]
    data = pd.DataFrame(
        data=summary,
        columns=["column", 't-statistic', "p-value"]
    )
    summary = data.sort_values(by="t-statistic", ascending=False)
    df_styled = summary.style.bar("t-statistic").background_gradient("Blues",
subset="t-statistic")
    dfi.export(df_styled, 'graphs/f_test.png', table_conversion="selenium")
    return data
```

Appendix 2.3: helper.py

```
def message(text):
    length = 30
    print("=" * length)
```

```
    print(text)
    print("=" * length)
```

Appendix 2.4: preprocessing.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import MinMaxScaler


def transform_categorical(df, cols: list[str]):
    mapper = {}
    for col in cols:
        mapped_col, tmp_map = pd.factorize(df[col])
        mapper[col] = tmp_map
        # index starts at 1
        df[col] = pd.Categorical(mapped_col + 1)
    return df, mapper


def transform_datetime(df: pd.DataFrame, col: str):
    tmp = col + "_utc"
    df[col + "_utc"] = pd.to_datetime(df[col], utc=True)
    df[col + "_year"] = df[tmp].dt.year
    df[col + "_month"] = df[tmp].dt.month
    df[col + "_day"] = df[tmp].dt.day

    # Set date as categorical variable
    df[col + "_year"] = df[col + "_year"].astype("category")
    df[col + "_month"] = df[col + "_month"].astype("category")
    df[col + "_day"] = df[col + "_day"].astype("category")

    # Remove saledate columns
    df.drop(columns=[col, tmp], inplace=True)
    return df

def car_preprocessing(df: pd.DataFrame):
    # Remove rows with missing values
    df.dropna(inplace=True)
    # Remove rows with duplicate values
    df.drop_duplicates(inplace=True)
    # Remove vin identifier, state
    df.drop(columns=["vin", "state"], inplace=True)
    # Split saledate into year, month, and day
    df = transform_datetime(df, "saledate")
    # Transform specified columns into categorical variables
    df, mapper = transform_categorical(df, ["make", "model", "trim", "body",
"transmission", "color", "interior", "seller"])
    df.info()
    return df, mapper

# Not Used: decision tree models do not need data scaling
def transform_scaling(df: pd.DataFrame, cols: list[str]):
    scaler = {}
    for col in cols:
        # Create scaler for target columns
        scaler[col] = MinMaxScaler().fit(df[col].values.reshape(-1, 1))
        # Scale data for target columns
```

```
        df[col] = scaler[col].transform(df[col].values.reshape(-1, 1))
    return df, scaler

def generate_dataset(df: pd.DataFrame, target: str, test_size: float):
    x = df.drop(columns=[target], axis=1)
    y = df[target]
    # Split training set and testing set
    x_tmp, x_test, y_tmp, y_test = train_test_split(
        x,
        y,
        test_size=test_size,
        random_state=314
    )
    # Split training set and validating set
    x_train, x_valid, y_train, y_valid = train_test_split(
        x_tmp,
        y_tmp,
        test_size=test_size,
        random_state=314
    )
    return x_train, x_test, x_valid, y_train, y_test, y_valid
```

Appendix 2.5 model.py

```
import lightgbm as lgb
from catboost import CatBoostRegressor
import matplotlib.pyplot as plt
import shap
import helper
import numpy as np
import lgbm_search
import catboost_search
from sklearn.metrics import mean_absolute_error, root_mean_squared_error,
mean_absolute_percentage_error


class LightGBM:
    model = None
    train_loss = {}

    def get_param(self, x_train, x_valid, y_train, y_valid):
        # Use optuna to find the HyperParameter
        helper.message("[INFO] Tuning hyperparameter (LightGBM) ...")
        param = lgbm_search.getHyperParameter(x_train, x_valid, y_train, y_valid)

        # Add compulsory hyperparameter
        param["verbosity"] = 1
        param["num_threads"] = 8
        param["device_type"] = "cpu"
        param["early_stopping_round"] = 10
        param["objective"] = "regression"
        param["metric"] = "rmse"

        helper.message("[INFO] Tuning completed.")
        helper.message(param)
        return param

    def train(self, x_train, y_train, x_valid, y_valid):
        # Wrapping train and test dataset
```

```
        train_data = lgb.Dataset(x_train, label=y_train,
categorical_feature=["make", "model", "trim", "body", "transmission", "color",
"interior", "seller", "saledate_day", "saledate_month", "saledate_year"])
        test_data = lgb.Dataset(x_valid, label=y_valid,
categorical_feature=["make", "model", "trim", "body", "transmission", "color",
"interior", "seller", "saledate_day", "saledate_month", "saledate_year"])

        # Tune hyperparameter
        param = self.get_param(x_train, x_valid, y_train, y_valid)

        # Create model
        helper.message("[INFO] Training model with tuned hyperparameter ... ")
        self.model = lgb.train(
            param,
            num_boost_round=200,
            train_set=train_data,
            valid_sets=[train_data, test_data],
            callbacks=[lgb.log_evaluation(),
lgb.record_evaluation(eval_result=self.train_loss)]
        )
        return 0

    def feature_report(self, x_train, y_train):
        # initialize JavaScript Visualization Library
        shap.initjs()
        helper.message("[INFO] Training explainer for LightGBM ...")
        tree = shap.TreeExplainer(self.model).shap_values(x_train)
        helper.message("[INFO] Training completed, visualising...")
        shap.summary_plot(tree, x_train, plot_type="bar",show=False)
        plt.savefig("graphs/lgb_feature_report.png")
        plt.show()
        return 0

    def eval(self, x_valid, y_valid):
        y_pred = self.model.predict(x_valid)
        res = ("Evaluation Report (LightGBM)\n\n" +
                "RMSE: " + str(root_mean_squared_error(y_valid, y_pred)) + "\n" +
                "MAE: " + str(mean_absolute_error(y_valid, y_pred)) + "\n")
        helper.message(res)
        return 0

    def training_report(self):
        fig = lgb.plot_metric(self.train_loss)
        plt.savefig("graphs/lgb_training_report.png")
        plt.show()
        return 0

    def eval(self, x_test, y_test):
        y_pred = self.model.predict(x_test)
        plt.plot(list(range(len(y_pred))), y_test, label="actual")
        plt.plot(list(range(len(y_pred))), y_pred, label="predict")
        plt.legend()
        plt.savefig("graphs/lgb_predict_graph.png")
        plt.show()
        res = ("Evaluation Report (LightGBM)\n\n" +
                "RMSE: " + str(root_mean_squared_error(y_test, y_pred)) + "\n" +
                "MAE: " + str(mean_absolute_error(y_test, y_pred)) + "\n")
        helper.message(res)
        return 0
class CatBoost:
    model = None
```

```
    def get_param(self, x_train, x_valid, y_train, y_valid):
        # Use optuna to find the HyperParameter
        helper.message("[INFO] Tuning hyperparameter (CatBoost) ...")
        param = catboost_search.getHyperParameter(x_train, x_valid, y_train,
y_valid)

        # Add compulsory hyperparameter
        param["cat_features"] = ["make", "model", "trim", "body", "transmission",
"color", "interior", "seller", "saledate_day", "saledate_month", "saledate_year"]
        param["verbose"] = 200
        param["thread_count"] = 8
        param["task_type"] = "CPU"
        param["iterations"] = 200

        helper.message("[INFO] Tuning completed.")
        helper.message(param)
        return param

    def train(self, x_train, y_train, x_valid, y_valid):
        # Tune hyperparameter
        param = self.get_param(x_train, x_valid, y_train, y_valid)

        # Create model
        helper.message("[INFO] Training model with tuned hyperparameter ... ")
        self.model = CatBoostRegressor(**param)
        self.model.fit(
            x_train,
            y_train,
            eval_set=(x_valid, y_valid),
            early_stopping_rounds=10,
            use_best_model=True
        )
        return 0

    def feature_report(self, x_train, y_train):
        # initialize JavaScript Visualization Library
        shap.initjs()
        helper.message("[INFO] Training explainer for CatBoost ...")
        tree = shap.TreeExplainer(self.model).shap_values(x_train)
        helper.message("[INFO] Training completed, visualising...")
        shap.summary_plot(tree, x_train, plot_type="bar",show=False)
        plt.savefig("graphs/cbt_feature_report.png")
        plt.show()
        return 0

    def training_report(self):
        evals_result = self.model.get_evals_result()
        train_loss = evals_result["learn"]["RMSE"]
        test_loss = evals_result["validation"]["RMSE"]

        # Plot the training progress
        iterations = np.arange(1, len(train_loss) + 1)

        plt.figure(figsize=(7, 4))
        plt.plot(iterations, train_loss, label="Training Loss", color="blue")
        plt.plot(iterations, test_loss, label="Validation Loss", color="orange")
        plt.xlabel("Iteration")
        plt.ylabel("Loss")
        plt.title("CatBoost Training Progress")
        plt.legend()
```

```
        plt.grid()
        plt.savefig("graphs/cbt_training_report.png")
        plt.show()
        return 0
    def eval(self, x_test, y_test):
        y_pred = self.model.predict(x_test)
        plt.plot(list(range(len(y_pred))), y_test, label="actual")
        plt.plot(list(range(len(y_pred))), y_pred, label="predict")
        plt.legend()
        plt.savefig("graphs/cbt_predict_graph.png")
        plt.show()
        res = (("Evaluation Report (CatBoost)\n\n" +
                "RMSE: " + str(root_mean_squared_error(y_test, y_pred)) + "\n" +
                "MAE: " + str(mean_absolute_error(y_test, y_pred)) + "\n"))
        helper.message(res)
        return 0
```

Appendix 2.6 lgbm_search.py

```
import numpy as np
import optuna
import lightgbm as lgb
from sklearn.metrics import mean_absolute_error,root_mean_squared_error,
mean_absolute_percentage_error
import sklearn.metrics

def objective(trial, x_train, x_test, y_train, y_test):
    train_data = lgb.Dataset(x_train, label=y_train, categorical_feature=["make",
"model", "trim", "body", "transmission", "color", "interior", "seller",
"saledate_day", "saledate_month", "saledate_year"])
    test_data = lgb.Dataset(x_test, label=y_test, categorical_feature=["make",
"model", "trim", "body", "transmission", "color", "interior", "seller",
"saledate_day", "saledate_month", "saledate_year"])
    param = {
        "objective": "regression",
        "metric": "rmse",
        "verbosity": -1,
        "boosting_type": "gbdt",
        "device_type": "cpu",
        "num_threads": 8,
        "early_stopping_round": 10,
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3),
        "lambda_l1": trial.suggest_float("lambda_l1", 1e-8, 10.0, log=True),
        "lambda_l2": trial.suggest_float("lambda_l2", 1e-8, 10.0, log=True),
        "max_depth": trial.suggest_int("max_depth", 3, 12),
        "min_data_in_leaf": trial.suggest_int("min_data_in_leaf", 200, 5000,
step=100),
        "max_bin": trial.suggest_int("max_bin", 200, 300),
        "num_leaves": trial.suggest_int("num_leaves", 2, 4096),
        "feature_fraction": trial.suggest_float("feature_fraction", 0.4, 1.0),
        "bagging_fraction": trial.suggest_float("bagging_fraction", 0.4, 1.0),
        "bagging_freq": trial.suggest_int("bagging_freq", 1, 7),
    }
    gbm = lgb.train(param, train_data, valid_sets=[test_data], num_boost_round=200)
    y_pred = gbm.predict(x_test)
    score = root_mean_squared_error(y_test, y_pred)
    return score
def getHyperParameter(x_train,x_test,y_train,y_test):
    study = optuna.create_study(direction="minimize")
```

```
    study.optimize(lambda trial: objective(trial, x_train, x_test, y_train,
y_test), n_trials=30)
    print("Number of finished trials: {}".format(len(study.trials)))
    trial = study.best_trial
    return trial.params
```

Appendix 2.7 catboost_search.py

```
import numpy as np
import optuna
import catboost as cb
from sklearn.metrics import mean_absolute_error,root_mean_squared_error,
mean_absolute_percentage_error
def objective(trial, x_train, x_test, y_train, y_test):
    params = {
        "cat_features": ["make", "model", "trim", "body", "transmission", "color",
"interior", "seller", "saledate_day", "saledate_month", "saledate_year"],
        "verbose": 0,
        "iterations": 200,
        "learning_rate": trial.suggest_float("learning_rate", 1e-3, 0.1, log=True),
        "depth": trial.suggest_int("depth", 1, 10),
        "subsample": trial.suggest_float("subsample", 0.05, 1.0),
        "colsample_bylevel": trial.suggest_float("colsample_bylevel", 0.05, 1.0),
        "min_data_in_leaf": trial.suggest_int("min_data_in_leaf", 1, 100),
        "task_type": "CPU",
        "thread_count": 8,
    }
    cbt = cb.CatBoostRegressor(**params)
    cbt.fit(
        x_train,
        y_train,
        eval_set=(x_test, y_test),
        use_best_model=True,
        early_stopping_rounds=10,
        verbose=False
    )
    y_pred = cbt.predict(x_test)
    score = root_mean_squared_error(y_test, y_pred)
    return score

def getHyperParameter(x_train, x_test, y_train, y_test):
    study = optuna.create_study(direction="minimize")
    study.optimize(lambda trial: objective(trial, x_train, x_test, y_train,
y_test), n_trials=30)
    print("Number of finished trials: {}".format(len(study.trials)))
    trial = study.best_trial
    return trial.params
```