

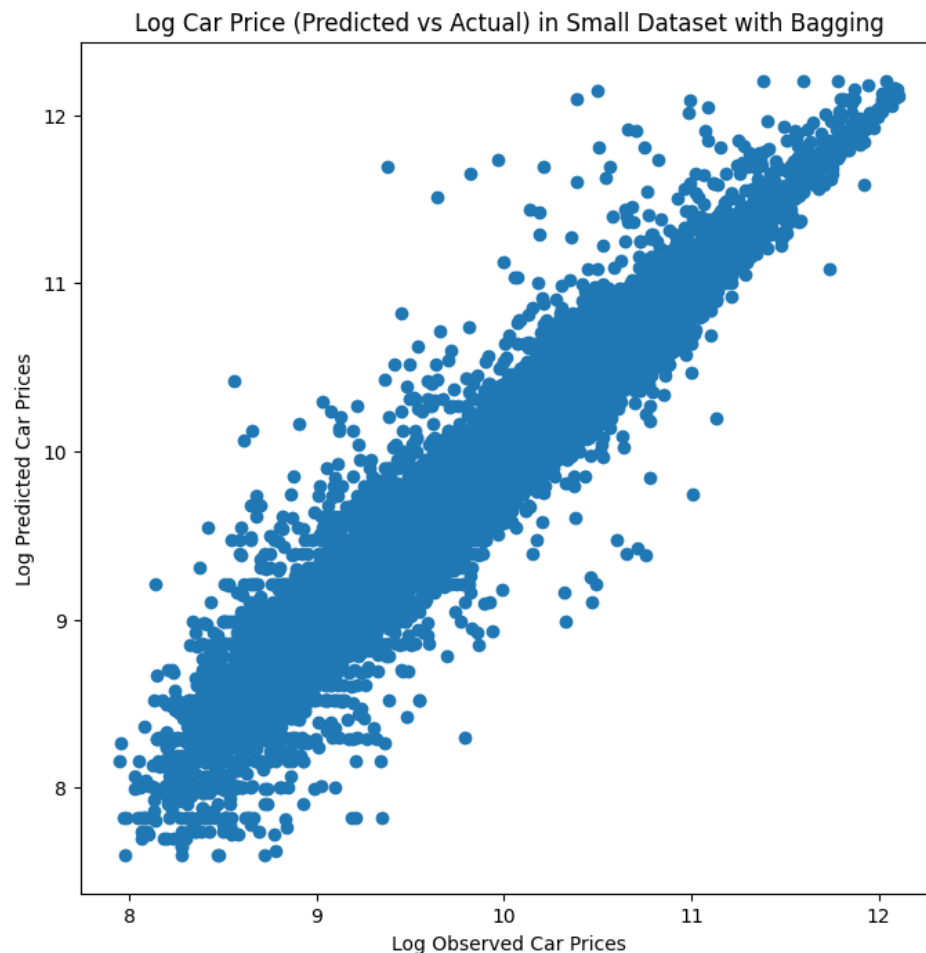
Anonymized Name: TheDebugger

MSE: 0.015556287973209721

$R^2$ : 0.9558079143797128

Algorithm Used: Bagging

## Question 2:



This pattern shows that most of the data follows a very straight line which implies a high accuracy, but the left tail looks slightly wider/spread out which implies this model is slightly worse at predicting or less accurate for cars of lower prices in the region 8-9.5 on the log scale. It also looks slightly below the line (predicted=observed or  $y=x$ ) for the region from the bottom left to (10,10) which implies that it underestimates the car prices when they are lower and conversely, does better for more expensive cars.

## Question 3: ChatGPT Interactions assisting me in finishing the assignment



How do I use ensemble methods to predict the natural log of car prices using bagging, boosting, and random forests? The dataset is given with the following features:

price,back\_legroom,bed,bed\_height,bed\_length,body\_type,cabin,city,city\_fuel\_economy,combi



To predict the natural log of car prices using ensemble methods like bagging, boosting, and random forests, as well as other models like k-nearest neighbors (KNN), and regression trees, you can follow these steps:

### 1. Data Preparation:

- Import the dataset and perform data preprocessing, including handling missing values, encoding categorical variables, and scaling/normalizing numerical features.
- Split the dataset into training and testing sets.

### 2. Model Selection:

- Choose the ensemble methods you want to use: bagging, boosting (e.g., AdaBoost or Gradient Boosting), and random forests. You can also include KNN and regression trees for comparison.

### 3. Feature Selection:

- If you have a large number of features, consider performing feature selection or dimensionality reduction to improve model performance.

### 4. Model Training:

- Train each of the selected models on the training data. This includes training individual decision trees, KNN, and ensemble models.

### 5. Cross-Validation:

- Perform cross-validation to estimate the performance of each model. You can use k-fold cross-validation to ensure that the results are not biased by the choice of a single train-test split.

### 6. Model Evaluation:

- Evaluate each model's performance on the validation set using appropriate metrics. Since



## 6. Model Evaluation:

- Evaluate each model's performance on the validation set using appropriate metrics. Since you want to predict the natural log of car prices, you might use metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE).

## 7. Hyperparameter Tuning:

- For ensemble methods, tuning hyperparameters is essential. Grid search or randomized search can help find the best hyperparameters for each model.

## 8. Final Model Selection:

- Based on the cross-validation results and model evaluation, select the best-performing models.

## 9. Model Testing:

- Evaluate the selected models on the test dataset to get an estimate of their real-world performance.

Here's a Python code example of how to implement this process using scikit-learn:

python

 Copy code

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import BaggingRegressor, RandomForestRegressor, AdaBoostRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import numpy as np
```

```
# Load and preprocess your dataset here
```

```
# Split the dataset into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, np.log(y), test_size=0.2)
```

```
# Initialize models
```

```
bagging = BaggingRegressor()
```

```
random_forest = RandomForestRegressor()
```

```
adaboost = AdaBoostRegressor()
```

```
gradient_boosting = GradientBoostingRegressor()
```

```
knn = KNeighborsRegressor()
```

```
decision_tree = DecisionTreeRegressor()
```

```
models = [bagging, random_forest, adaboost, gradient_boosting, knn, decision_tree]
```

```
# Cross-validation and model evaluation
```

```
for model in models:
```

```
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
```

```
    rmse = np.sqrt(-scores)
```

```
    print(f"Model: {model.__class__.__name__}")
```

```
    print(f"Cross-Validation RMSE: {rmse.mean()}")
```



Draw a figure that illustrates how well your model predicts for different values/ranges of the response variable (for example: high, medium, low) in the training set. One way to do this is to draw a figure where values of the response variable are on the horizontal axis and values of predictions are on the vertical axis. You can consider other ways too, please provide only one figure.

Discuss the implications of this pattern in 1-2 sentences.



I can describe how to create a figure and discuss its implications based on the pattern you're interested in. However, I don't have the actual data or the ability to create visual figures. Here's how you can create the figure and interpret the implications:

### Figure Description:

Create a scatter plot where the x-axis represents the actual values of the response variable (e.g., "price") in your training set, and the y-axis represents the corresponding predicted values generated by your model. Each data point in the scatter plot corresponds to an observation in your training set.

### Implications:

The pattern in this scatter plot can reveal how well your model predicts values for different ranges of the response variable. Key implications to look for:

1. **High Accuracy Region:** If the data points cluster closely around the diagonal line ( $y = x$ ), it indicates that your model provides accurate predictions for response values in a certain range. This implies that your model performs well for those observations with values in this range.
2. **Overestimation or Underestimation:** If the data points tend to fall above or below the diagonal line, it suggests that your model systematically overestimates or underestimates the response variable. This could indicate a bias in your model.
3. **Scattered Points:** If the data points are widely scattered and don't exhibit a clear pattern, it may suggest that your model's predictions are inconsistent across different ranges of the response variable. This could imply that your model doesn't capture the underlying relationships well or that there are unmodeled factors.
4. **Heteroscedasticity:** If the spread of data points increases or decreases as you move along the x-axis, it may indicate heteroscedasticity, suggesting that the model's prediction errors vary systematically with the response variable.

## Code

```
In [ ]: import pandas as pd
        from sklearn.preprocessing import LabelEncoder
        import re
        import numpy as np
```

## Data Prep

```
In [ ]: # Read in Data
        data = pd.read_csv("./Econ424_F2023_PC4_training_data_small.csv")
        print(data.head())
```

```
In [ ]: # Find in missing values
        missing = data.isna().sum()
        for x in range(len(missing)):
            print(str(data.columns[x]) + ": " + str(missing[x]))
            # print(missing[x])
        data.shape
        data.info()
```

```
In [ ]: # Large Data
        dataL = pd.read_csv("./Econ424_F2023_PC4_training_data_large.csv")
        print(dataL.head())
```

```
In [ ]: missingL = dataL.isna().sum()
        for x in range(len(missingL)):
            print(str(dataL.columns[x]) + ": " + str(missingL[x]))
```

Missing is\_certified,  
vehicle\_damage\_category,  
combine\_fuel\_economy for all of them

```
In [ ]: # useless categories
        # fleet, is_cpo, is_oemcpo major options, bed, bed height, bed length, cabin
        data.drop(['is_certified', 'vehicle_damage_category', 'combine_fuel_economy',
                    axis='columns', inplace=True)
        data.shape
```

```
In [ ]: data.columns
```

```
In [ ]: for col in data:
        print(col)
        print(data[col].unique())
```

```
In [ ]: # Define a regular expression pattern to extract float values
# pattern = r'(\d+\.\d+)' in'
pattern = r' in'
patternGal = r'(\d+\.\d+) gal'
weirdCols = ["back_legroom", "front_legroom", "height", "length", "wheelbase"]
# Iterate through the columns and extract float components for matching columns
for column in weirdCols:
    if len(data[column].unique()) >= 4 and data[column].dtype == object:
        print(column)
        for i in range(len(data[column])):
            if pd.isna(data[column][i]):
                continue
            elif isinstance(data[column][i], str):
                # print("found string")

                if len(data[column][i]) <= 2:
                    continue
                end = data[column][i][-3:]
                if end == " in":
                    data[column][i] = float(data[column][i][: -3])
                    continue
                if len(data[column][i]) <= 3:
                    continue
                end = data[column][i][-4:]
                if end == " gal":
                    data[column][i] = float(data[column][i][: -4])
                    continue

                if len(data[column][i]) <= 5:
                    continue
                end = data[column][i][-6:]
                if end == " seats":
                    data[column][i] = int(data[column][i][: -6])

data.head
```

```
In [ ]: for col in data.columns:
        print(col + ": " + str(data[col].unique()))
```

```
In [ ]: # Replace all with mean and mode
categorical_columns = ['trimid', 'body_type', 'city', 'dealer_zip', 'engine_type',
                      'listing_color', 'major_options', 'make_name', 'model_name']
bool_columns = ['frame_damaged', 'franchise_dealer', 'has_accidents', 'is_new',

for col in data.columns:
    if col in categorical_columns or col in bool_columns:
        # calculate mode
        average = "-1"
        # Replace "--" with NaN
        data[col] = data[col].replace(np.nan, "--")
        data[col] = data[col].replace("--", pd.NA)
        # Calculate the mode of the valid string values
        mode_value = data[col].mode(dropna=True).iloc[0]

        # Replace NaN with the mode
```

```

data[col].fillna(mode_value, inplace=True)

elif col != "listed_date":
    # calculate mean
    # Convert non-numeric values ("--") to NaN
    data[col] = pd.to_numeric(data[col], errors="coerce")

    # Calculate the mean of the valid numeric values
    mean_value = data[col].dropna().mean()

    # Replace NaN and "--" with the mean
    data[col].fillna(mean_value, inplace=True)

# Mean: back_legroom, city_fuel_economy, engine_displacement, front_legroom,
# highway_fuel_economy, mileage, wheelbase, width

# Mode: maximum_seating, owner_count, seller_rating, trimid

```

```

In [ ]: for col in data.columns:
        print(col + ": " + str(data[col].unique()))

```

```

In [ ]: missing = data.isna().sum()
        for x in range(len(missing)):
            print(str(data.columns[x]) + ": " + str(missing[x]))
            # print(missing[x])
        data.shape
        data.info()
        print(data['listed_date'])
        print(data['year'])

```

```

In [ ]: # Updated csv file
        csv_file = "./updatedSmall.csv"

        # Use numpy.savetxt to save the array as a CSV file
        data.to_csv(csv_file, index=False, encoding="utf-8")

```

## Apply Label Encoder and Standard Scaler

```

In [ ]: data = pd.read_csv("./updatedSmallFinal.csv")

```

```

In [ ]: # setup categorical variables
        # label encoder to encode the different categorical features
        categorical_columns = ['trimid', 'body_type', 'city', 'dealer_zip', 'engine_type',
                               'listing_color', 'major_options', 'make_name', 'model_name',
                               'year']
        bool_columns = ['frame_damaged', 'franchise_dealer', 'has_accidents', 'is_new',
                         'is_used']

        label_encoder = LabelEncoder()
        for category in categorical_columns:
            print("Doing it for category: " + category)

```



```

data[category] = data[category].astype(str)
print(all)

data[category] = label_encoder.fit_transform(data[category])
# data['target'] = np.log(data['price'])
for category in bool_columns:
    print("Doing it for category: " + category)

data[category] = data[category].astype(str)
print(all)

data[category] = label_encoder.fit_transform(data[category])

```

```

In [ ]: data.head()
data.info()

```

```

In [ ]: missing = data.isna().sum()

for x in range(len(missing)):
    print(str(data.columns[x]) + ": " + str(missing[x]))
    # print(missing[x])

```

```

In [ ]: data['target'] = np.log(data['price'])

```

```

In [ ]: print(data['target'])

```

## Create Different Models

```

In [ ]: # Imports
from matplotlib.pyplot import subplots
from statsmodels.datasets import get_rdataset
import sklearn.model_selection as skm
from ISLP import load_data , confusion_table
from ISLP.models import ModelSpec as MS
from sklearn.tree import (DecisionTreeClassifier as DTC, DecisionTreeRegressor)
from sklearn.metrics import (accuracy_score , log_loss)
from sklearn.ensemble import (RandomForestRegressor as RF, GradientBoostingRegressor)

from matplotlib.pyplot import subplots
import statsmodels.api as sm

from sklearn.model_selection import train_test_split
from matplotlib.pyplot import subplots
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

from sklearn.preprocessing import LabelEncoder

```

```

In [ ]: # Set up Data
Y = data['target']

```



```
X = data.drop(columns=['price', 'target', 'listed_date'])
```

```
In [ ]: # Split data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, ran
```

## Bagging

```
In [ ]: bag = RF(max_features=X_train.shape[1], random_state=0)
bag.fit(X_train, y_train)
```

```
In [ ]: ax = subplots(figsize=(8,8))[1]
y_hat_bag = bag.predict(X_test)
ax.scatter(y_hat_bag, y_test)
np.mean((y_test - y_hat_bag)**2)
```

```
In [ ]: feature_imp = pd.DataFrame( {'importance':bag.feature_importances_}, index=X
feature_imp.sort_values(by='importance', ascending=False)
```

```
In [ ]: data_bag = RF(max_features=X_train.shape[1], n_estimators=500, random_state=
y_hat_bag = data_bag.predict(X_test)
np.mean((y_test - y_hat_bag)**2)
```

```
In [ ]: ax = subplots(figsize=(8,8))[1]
ax.scatter(y_hat_bag, y_test)
ax.title.set_text('Log Car Price (Predicted vs Actual) in Small Dataset with
ax.set_xlabel("Log Observed Car Prices")
ax.set_ylabel("Log Predicted Car Prices")
ax.axline((7.5,7.5), slope=1)
```

```
In [ ]: feature_imp = pd.DataFrame( {'importance':data_bag.feature_importances_}, ir
feature_imp.sort_values(by='importance', ascending=False)
```

```
In [ ]: r2_score(y_test, y_hat_bag)
```

## Random Forests

```
In [ ]: rf = RF(max_features=int(np.sqrt(X_train.shape[1])), random_state=0)
rf.fit(X_train, y_train)
```

```
In [ ]: ax = subplots(figsize=(8,8))[1]
y_hat_rf = rf.predict(X_test)
ax.scatter(y_hat_rf, y_test)
np.mean((y_test - y_hat_rf)**2)
```

```
In [ ]: r2_score(y_test, y_hat_rf)
```

```
In [ ]: feature_imp = pd.DataFrame( {'importance':rf.feature_importances_}, index=X
feature_imp.sort_values(by='importance', ascending=False)
```

## Boosting

```
In [ ]: data_boost = GBR(n_estimators=5000, learning_rate=0.2, max_depth=3, random_s
data_boost.fit(X_train, y_train)
```

```
In [ ]: test_error = np.zeros_like(data_boost.train_score_)
for idx, y_ in enumerate(data_boost.staged_predict(X_test)):
    test_error[idx] = np.mean((y_test - y_)**2)
plot_idx = np.arange(data_boost.train_score_.shape[0])
ax = subplots(figsize=(8,8))[1]
ax.plot(plot_idx,data_boost.train_score_, 'b',label='Training')
ax.plot(plot_idx, test_error , 'r',label='Test')
ax.legend();
```

```
In [ ]: ax = subplots(figsize=(8,8))[1]
y_hat_boost = rf.predict(X_test)
ax.scatter(y_hat_boost, y_test)
np.mean((y_test - y_hat_boost)**2)
```

```
In [ ]: y_hat_boost = data_boost.predict(X_test)
np.mean((y_test - y_hat_boost)**2)
```

```
In [ ]: r2_score(y_test, y_hat_boost)
```

## Create log charts for each model

```
In [ ]: # choose best one
bestModel = data_bag
```

## Make Predictions

```
In [ ]: # Preprocess prediction data
# Read in Data
dataPred = pd.read_csv("./Econ424_F2023_PC4_test_data_without_response_var.c
print(dataPred.head())

dataPred.drop(['is_certified', 'vehicle_damage_category', 'combine_fuel_econco
axis='columns', inplace=True)
dataPred.shape

weirdCols = ["back_legroom", "front_legroom", "height", "length", "wheelbase
# Iterate through the columns and extract float components for matching colu
for column in weirdCols:
    if len(dataPred[column].unique()) >= 4 and dataPred[column].dtype == obj
        print(column)
        for i in range(len(dataPred[column])):
            if pd.isna(dataPred[column][i]):
                continue
```

```

elif isinstance(dataPred[column][i], str):
    # print("found string")

    if len(dataPred[column][i]) <= 2:
        continue
    end = dataPred[column][i][-3:]
    if end == " in":
        dataPred[column][i] = float(dataPred[column][i][: -3])
        continue
    if len(dataPred[column][i]) <= 3:
        continue
    end = dataPred[column][i][-4:]
    if end == " gal":
        dataPred[column][i] = float(dataPred[column][i][: -4])
        continue

    if len(dataPred[column][i]) <= 5:
        continue
    end = dataPred[column][i][-6:]
    if end == " seats":
        dataPred[column][i] = int(dataPred[column][i][: -6])

```

```

In [ ]: # Replace all with mean and mode
categorical_columns = ['trimid', 'body_type', 'city', 'dealer_zip', 'engine_type',
                      'listing_color', 'major_options', 'make_name', 'model_r
bool_columns = ['frame_damaged', 'franchise_dealer', 'has_accidents', 'is_new',

for col in dataPred.columns:
    if col in categorical_columns or col in bool_columns:
        # calculate mode
        average = "-1"
        # Replace "--" with NaN
        dataPred[col] = dataPred[col].replace(np.nan, "--")
        dataPred[col] = dataPred[col].replace("--", pd.NA)
        # Calculate the mode of the valid string values
        mode_value = dataPred[col].mode(dropna=True).iloc[0]

        # Replace NaN with the mode
        dataPred[col].fillna(mode_value, inplace=True)

    elif col != "listed_date":
        # calculate mean
        # Convert non-numeric values ("--") to NaN
        dataPred[col] = pd.to_numeric(dataPred[col], errors="coerce")

        # Calculate the mean of the valid numeric values
        mean_value = dataPred[col].dropna().mean()

        # Replace NaN and "--" with the mean
        dataPred[col].fillna(mean_value, inplace=True)

```

```

In [ ]: categorical_columns = ['trimid', 'body_type', 'city', 'dealer_zip', 'engine_type',
                              'listing_color', 'major_options', 'make_name', 'model_r
bool_columns = ['frame_damaged', 'franchise_dealer', 'has_accidents', 'is_new',

```

```

label_encoder = LabelEncoder()
for category in categorical_columns:
    print("Doing it for category: " + category)
    dataPred[category] = dataPred[category].astype(str)
    print(all)

    dataPred[category] = label_encoder.fit_transform(dataPred[category])
for category in bool_columns:
    print("Doing it for category: " + category)
    dataPred[category] = dataPred[category].astype(str)
    print(all)

    dataPred[category] = label_encoder.fit_transform(dataPred[category])

missing = dataPred.isna().sum()

for x in range(len(missing)):
    print(str(dataPred.columns[x]) + ": " + str(missing[x]))
    # print(missing[x])

```

```

In [ ]: # apply prediction
dataPred.drop(["price"],axis="columns",inplace=True)
Y_test = bestModel.predict(dataPred)

```

```

In [ ]: len(Y_test)

```

```

In [ ]: # output to csv file
csv_file_out = "./output.csv"

# Save the DataFrame to a CSV file
np.savetxt(csv_file_out, Y_test, delimiter="\n", fmt="%1.6f")

```

```

In [ ]:

```