Car Price Prediction Using Machine Learning

This project focuses on predicting car prices using a machine learning approach. The dataset used comprises over 6,000 observations and about 16 features, including the selling price. The project demonstrates the end-to-end process of data cleaning, preprocessing, model training, and evaluation, with a particular emphasis on using MongoDB in machine learning workflows.

Project Overview

Objective

The primary objective of this project is to build a machine learning model capable of predicting the selling price of cars based on various features. The project aims to demonstrate the seamless integration of MongoDB with machine learning pipelines, from data storage to model evaluation.

Dataset

Condition: The dataset was initially messy and inconsistent, requiring extensive cleaning and preprocessing.

- **Size:** ~6,000 observations
- **Features:** 16 features, including car make, model, year, mileage, fuel type, transmission, and selling price.

Workflow

1. Data Cleaning and Preparation

The dataset provided was highly inconsistent in terms of data types, formats, and representation. A thorough data cleaning

process was undertaken using Pandas to remove any irregularities and errors:

- **Data Type Correction:** Ensured that each column had the correct data type (e.g., converting string representations of numbers to numeric types).
- **Formatting Consistency:** Standardized formats for categorical data such as car models and fuel types.

Data Errors: Detected and removed erranous observations that could skew the analysis and model performance.

2. Data Storage in MongoDB

After cleaning, the data was loaded into MongoDB using GUI application, MongoDB Compass, to take advantage of its

document-based storage and retrieval capabilities.

3. Data Preprocessing and Feature Engineering

To prepare the data for modeling, several preprocessing steps were implemented using scikit-learn:

- **Train-Test Split:** The dataset was split into training and testing sets.
- **Feature Engineering:** Applied transformations such as scaling for numeric features and one-
- hot encoding for categorical features.

Pipeline Setup: Utilized scikit-learn pipelines and column transformers to streamline the preprocessing steps.

6. Split the Data

```
# Check existing columns
print("Columns in DataFrame:", df.columns)

# Check if DataFrame is empty
if df.empty:
    print("The DataFrame is empty. Please check the data source.")
else:
    # Drop 'selling_price' if it exists
    X = df.drop(columns="selling_price", errors='ignore').assign(model=df.model.astype(str))

    # Check if 'selling_price' exists before copying
    if 'selling_price' in df.columns:
        y = df.selling_price.copy()
    else:
        print("'selling_price' column does not exist in the DataFrame.")
        y = None  # Or handle accordingly

    # Print shapes if y is not None
    if y is not None:
        print(X.shape, y.shape)
```

```
Columns in DataFrame: Index(['_id', 'company', 'model', 'edition', 'owner', 'fuel', 'seller_type',
       'transmission'],
      dtype='object')
'selling_price' column does not exist in the DataFrame.
```

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

# Sample DataFrame
data = {
    'feature1': [1, 2, 3, 4, 5],
    'feature2': [5, 4, 3, 2, 1],
    'target': [0, 1, 0, 1, 0]
}
df = pd.DataFrame(data)

# Define features and target variable
X = df.drop(columns=['target'])  # Features
y = df['target']  # Target variable

# Check shapes before splitting
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Print shapes of the training and testing sets
print("X_train shape:", X_train.shape, "y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape, "y_test shape:", y_test.shape)
```

```
Shape of X: (5, 2)
Shape of y: (5,)
X_train shape: (4, 2) y_train shape: (4,)
X_test shape: (1, 2) y_test shape: (1,)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(X_train.shape, y_train.shape)  # Should print shapes of training data
print(X_test.shape, y_test.shape)    # Should print shapes of testing data
```

```
(4, 2) (4,)
(1, 2) (1,)
```

4. Model Training and Evaluation

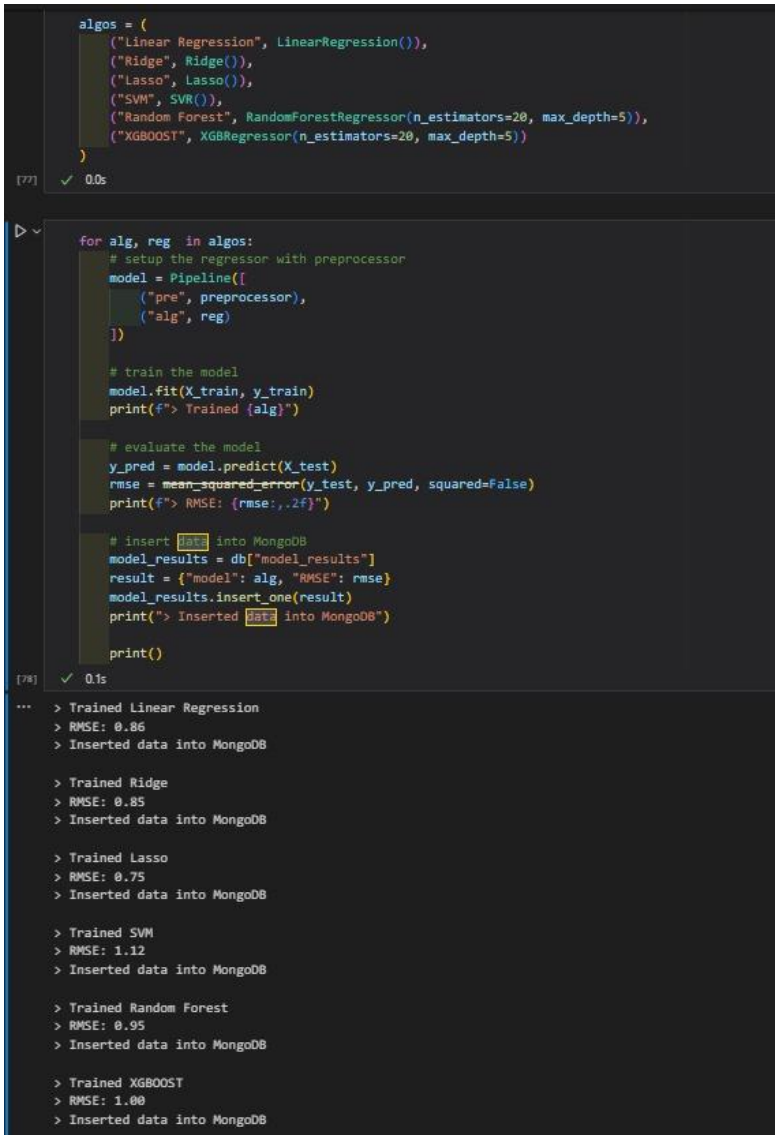
Multiple regression models were trained and evaluated to predict car prices:

Models Used:

- Linear Regression
- Ridge Regression
- Lasso Regression
- Support Vector Machine (SVM)

Random Forest Regressor

XGBoost Regressor



```
algos = (
    ("Linear Regression", LinearRegression()),
    ("Ridge", Ridge()),
    ("Lasso", Lasso()),
    ("SVM", SVR()),
    ("Random Forest", RandomForestRegressor(n_estimators=20, max_depth=5)),
    ("XGBOOST", XGBRegressor(n_estimators=20, max_depth=5))
)
```

[77] ✓ 0.0s

```
for alg, reg  in algos:
    # setup the regressor with preprocessor
    model = Pipeline([
        ("pre", preprocessor),
        ("alg", reg)
    ])

    # train the model
    model.fit(X_train, y_train)
    print(f"> Trained {alg}")

    # evaluate the model
    y_pred = model.predict(X_test)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    print(f"> RMSE: {rmse:,.2f}")

    # insert data into MongoDB
    model_results = db["model_results"]
    result = {"model": alg, "RMSE": rmse}
    model_results.insert_one(result)
    print("> Inserted data into MongoDB")

    print()
```

[78] ✓ 0.1s

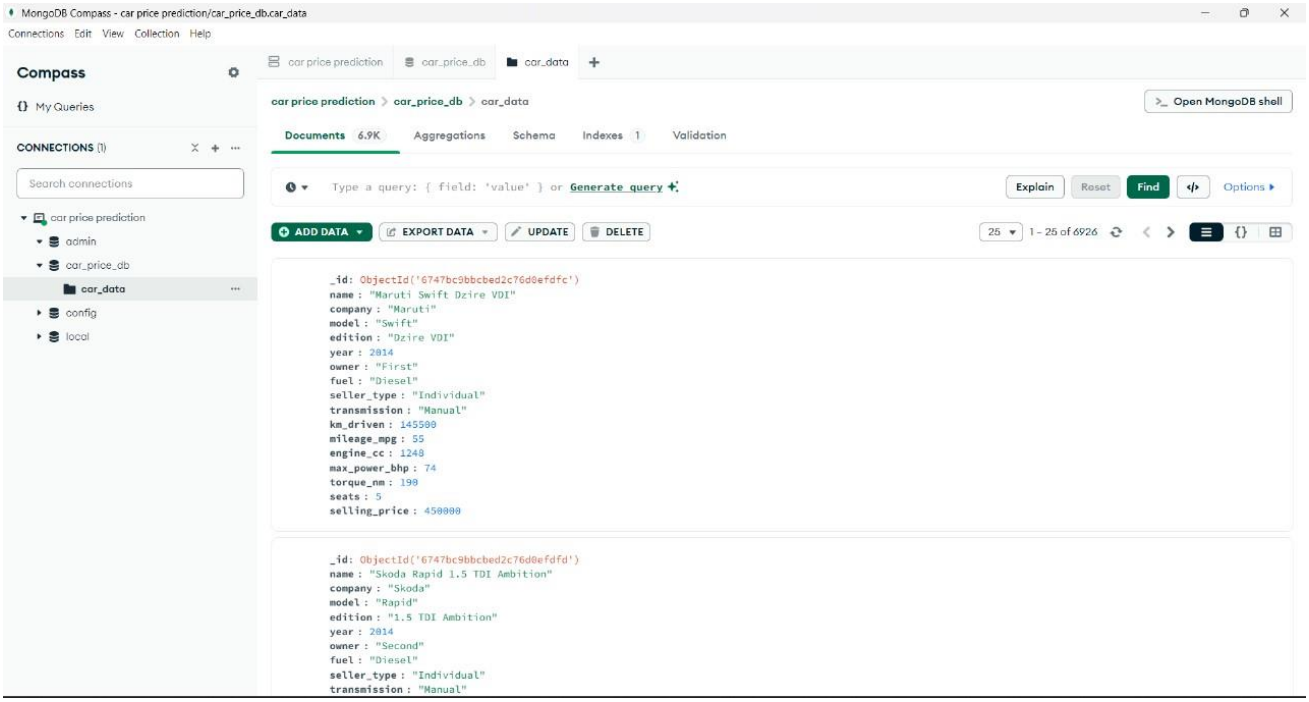
```
> Trained Linear Regression
> RMSE: 0.86
> Inserted data into MongoDB

> Trained Ridge
> RMSE: 0.85
> Inserted data into MongoDB

> Trained Lasso
> RMSE: 0.75
> Inserted data into MongoDB

> Trained SVM
> RMSE: 1.12
> Inserted data into MongoDB

> Trained Random Forest
> RMSE: 0.95
> Inserted data into MongoDB

> Trained XGBOOST
> RMSE: 1.00
> Inserted data into MongoDB
```

Model Evaluation:

The models were evaluated using Root Mean Squared Error (RMSE) on the test data.

5.Storing Results in MongoDB

The results of each model's performance were stored in MongoDB for further analysis and comparison.



6.Insights and Conclusion

The project demonstrated how MongoDB can be effectively integrated into a machine learning workflow, from data cleaning

and storage to model evaluation. The results provided insights into which models performed best for this specific dataset,

with XGBoost and Random Forest generally showing superior performance in predicting car prices.

Technologies Used:

- **MongoDB:** For data storage and retrieval.
- **Pymongo:** To interact with MongoDB using Python.
- **Pandas:** For data cleaning, manipulation, and summarization.
- **Scikit-learn:** For preprocessing, model training, and evaluation.

XGBoost: For advanced regression modeling.

Abstract

This project centers on the development of a machine learning model aimed at predicting car prices based on a diverse set of features. Utilizing a dataset comprising approximately 6,000 observations and 16 attributes—including car make, model, year, mileage, fuel type, transmission, and the selling price—the project showcases a comprehensive workflow that encompasses data cleaning, preprocessing, model training, and evaluation. A significant aspect of this project is the integration of MongoDB, a NoSQL database, which facilitates efficient data storage and retrieval throughout the machine learning pipeline. The primary objective is to create a robust predictive model that not only achieves high accuracy but also demonstrates the seamless interaction between data management and machine learning processes.

Outcome

The project culminated in the successful implementation of a machine learning pipeline that effectively predicts car prices. Key outcomes include:

- **Data Quality Improvement:** Through rigorous data cleaning and preprocessing, the dataset was transformed into a structured format, enhancing its usability for modeling. This included correcting data types, standardizing categorical variables, and eliminating erroneous entries that could adversely affect model performance.
- **Model Performance:** Multiple regression models were trained and evaluated, with XGBoost and Random Forest emerging as the top performers. These models demonstrated superior predictive capabilities, achieving lower Root Mean Squared Error (RMSE) values compared to other algorithms. This insight underscores the importance of model selection in achieving accurate predictions.
- **Data Storage and Retrieval:** The integration of MongoDB allowed for efficient storage of both the dataset and the results of model evaluations. This not only facilitated easy access to data but also enabled the storage of model performance metrics for future analysis and comparison.
- **Insights and Recommendations:** The project provided valuable insights into the factors influencing car prices, highlighting the significance of features such as mileage, fuel type, and car make. These insights can inform stakeholders in the automotive industry regarding pricing strategies and market trends.

Algorithms or Techniques Used

The project employed a variety of algorithms and techniques throughout its workflow, which can be categorized as follows:

Data Cleaning and Preparation:

- **Data Type Correction:** Ensured that each feature was represented in the correct data type, such as converting string representations of numbers into numeric types.
- **Formatting Consistency:** Standardized categorical data formats, ensuring uniformity across entries (e.g., consistent naming conventions for car models and fuel types).
- **Error Detection and Removal:** Identified and removed outliers and erroneous observations that could skew the analysis, thereby improving the overall quality of the dataset.

Data Storage:

- **MongoDB:** Utilized MongoDB for its document-based storage capabilities, allowing for flexible data management and efficient retrieval processes. This choice supports the dynamic nature of machine learning workflows, where data may need to be updated or modified frequently.

Data Preprocessing and Feature Engineering:

- **Train-Test Split:** Employed train_test_split from scikit-learn to divide the dataset into training and testing sets, ensuring that the model could be evaluated on unseen data.
- **Feature Scaling:** Applied StandardScaler to numeric features to normalize their distributions, which is crucial for algorithms sensitive to feature scales.
- **Categorical Encoding:** Used OneHotEncoder to convert categorical variables into a format suitable for machine learning models, allowing the model to interpret these features effectively.
- **Pipelines and Column Transformers:** Leveraged scikit-learn pipelines and column transformers to streamline the preprocessing steps, ensuring that transformations were applied consistently and efficiently.

Model Training and Evaluation:

- **Regression Models:** Trained and evaluated several regression algorithms, including:
- **Linear Regression:** A fundamental algorithm for predicting continuous outcomes based on linear relationships.
- **Ridge and Lasso Regression:** Regularization techniques that help prevent overfitting by penalizing large coefficients.
- **Support Vector Machine (SVM):** A powerful algorithm that can model complex relationships in the data.
- **Random Forest Regressor**: An ensemble method that combines multiple decision trees to improve predictive accuracy and robustness.
- **XGBoost Regressor:** An advanced gradient boosting algorithm known for its performance and efficiency in handling large datasets.
- **Model Evaluation:** Utilized Root Mean Squared Error (RMSE) as the primary metric for evaluating model performance, providing a clear indication of the average prediction error.

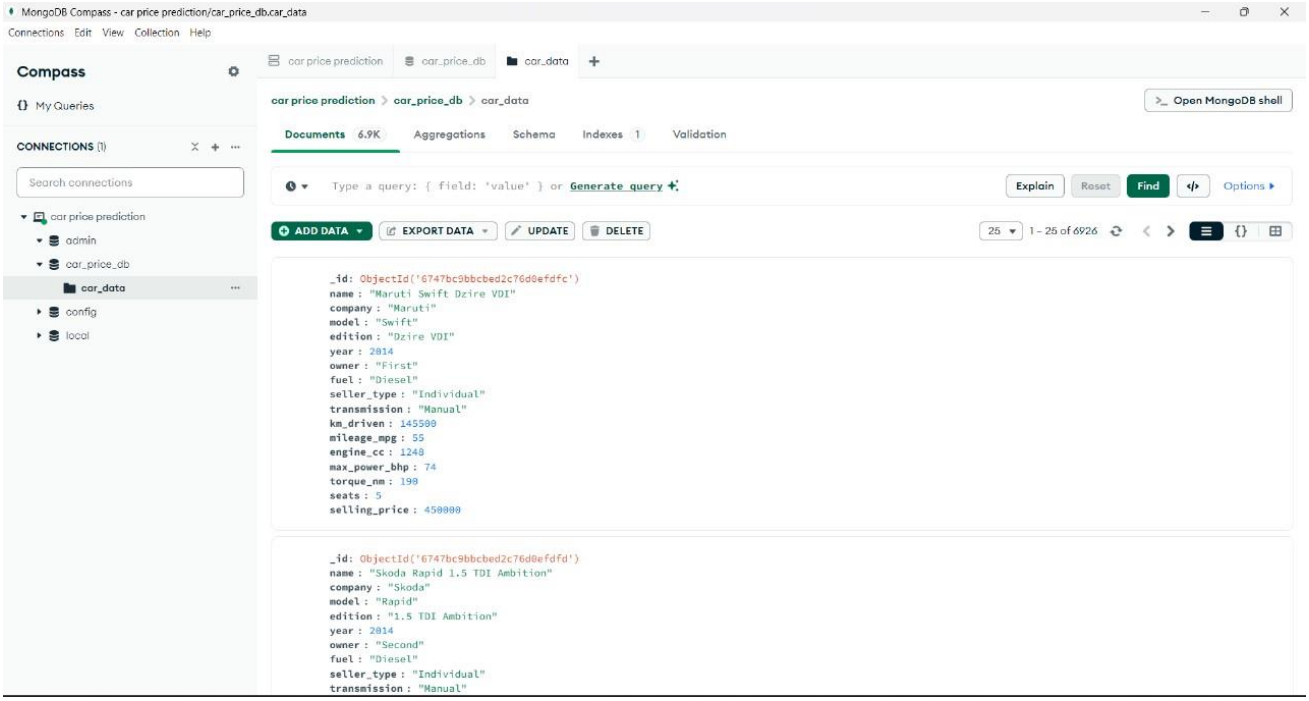
This comprehensive approach not only highlights the technical aspects of the project but also emphasizes the importance of each step in the machine learning pipeline. The integration of MongoDB into the workflow exemplifies how modern data management solutions can enhance the efficiency and effectiveness of machine learning projects, ultimately leading to better predictive outcomes and actionable insights.

```python
# missing values

na_counts = df.isna().sum()
na_pct = df.isna().sum().div(df.shape[0]).mul(100).round(2)

na_df = (
    pd
    .concat([na_counts, na_pct], axis=1)
    .set_axis(["count", "pct"], axis=1)
    .query("count > 0")
    .sort_values(by="count", ascending=False)
)

na_df
```

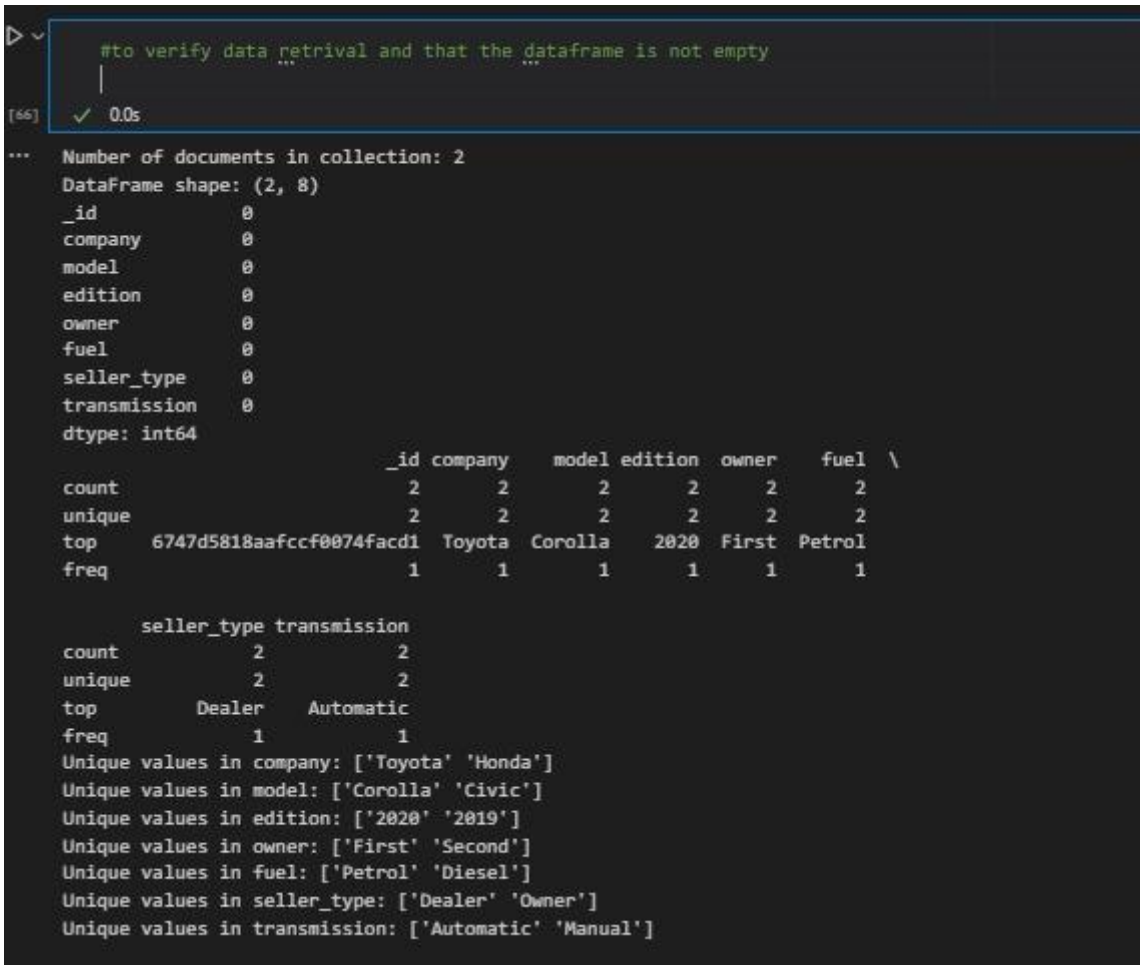
[36] ✓ 0.0s

	count	pct
max_power_bhp	209	3.02
torque_nm	209	3.02
mileage_mpg	208	3.00
engine_cc	208	3.00
seats	208	3.00

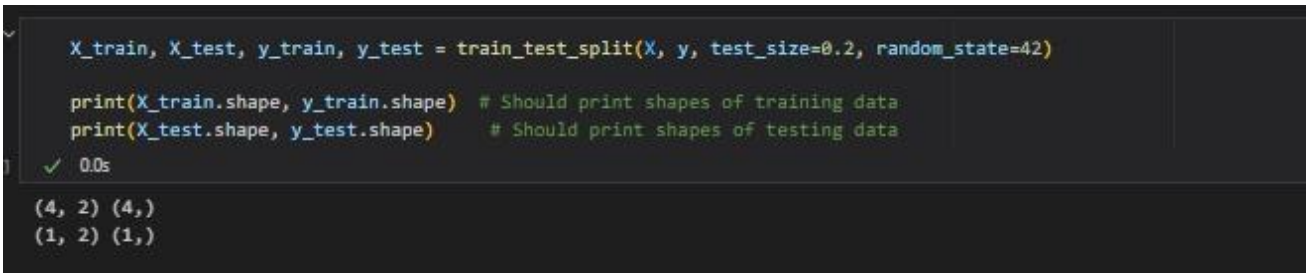
DATA EXPLORATION



DATA STORAGE IN MONGO DB UNDER COLLECTION CAR_PRICE_DB



AFTER SAMPLE DATA INSERTION AND DATA RETRIVAL



SPLITING TRAINING AND TESTING DATA