Project Report

Used Car Price Prediction System

CS-GY-6513-D (SP 25)
Big Data

Devakumar Katta (dk4945)

Table of Contents

Executive Summary

Project Name

Used Car Price Prediction System

Brief Summary

Pricing used cars in today's market is often inconsistent and unreliable. Buyers are concerned about overpaying, while sellers risk undervaluing their vehicles due to the absence of standardized, data-driven valuation tools. Traditional pricing methods typically rely on subjective assessments, outdated references, or individual sales experience, making it difficult for both consumers and dealerships to determine a fair market price. This leads to inefficiencies, financial losses, and a general lack of trust in the resale ecosystem.

To address this real-world challenge, our project introduces a machine learning-based system that predicts used car prices using historical data. We leverage PySpark to process and analyze a large-scale Kaggle dataset of over 3 million used car listings from across the U.S. The system identifies key pricing patterns based on features such as brand, model year, mileage, fuel type, and transmission, enabling accurate and consistent value estimation.

By integrating scalable big data tools with interpretable machine learning models, the project empowers buyers and sellers with fair, transparent pricing insights. Ultimately, this solution aims to improve decision-making, enhance market transparency, and build trust in the automotive resale landscape.

Objectives

- **Ingest and preprocess large-scale data**: Efficiently handle and clean a 10GB+ dataset containing over 3 million used car listings using PySpark, ensuring that the system is capable of operating at scale.

- **Identify key pricing factors**: Perform exploratory data analysis (EDA) to uncover trends and relationships among features such as make, model, year, mileage, fuel type, transmission, and geographic location.

- **Design and train predictive models**: Develop regression models, starting with Linear Regression as a baseline and progressing to more advanced models like Random Forest, to estimate car prices from structured input data.

- **Evaluate model performance**: Assess the accuracy and reliability of each model using appropriate metrics including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 Score.

- **Optimize for interpretability and accuracy**: Balance performance with model transparency to ensure that pricing predictions can be explained to end users and trusted by both technical and non-technical stakeholders.

- **Ensure pipeline reusability**: Construct a modular, end-to-end pipeline for data cleaning, feature engineering, model training, and prediction that can be extended to new datasets or deployed into applications.

- **Demonstrate the value of big data tools**: Showcase how technologies like PySpark can be effectively used in real-world scenarios where traditional tools (e.g., pandas) would fail due to data volume or memory constraints.

Technologies Used

To process, analyze, and model the large-scale used car dataset efficiently, we adopted a range of technologies suited for big data processing, machine learning, and visualization:

- **Apache Spark (PySpark)**:
  Used as the core big data processing engine. PySpark enabled us to handle over 3 million records (~10GB) efficiently by distributing computations across multiple cores, allowing us to perform preprocessing, transformations, and model training at scale.

- **Python**:
  Served as the primary programming language for data handling, machine learning, and visualization tasks due to its flexibility and robust ecosystem of libraries.

- **Scikit-learn**:
  Provided access to a variety of machine learning algorithms, including Linear Regression and Random Forest Regressor. It also offered utility functions for evaluation metrics such as MAE, RMSE, and R^2 Score.

- **Spark MLlib**:
  Spark's native machine learning library was used for building scalable pipelines for feature engineering (e.g., `StringIndexer`, `VectorAssembler`) and training regression models on distributed data.

- **Matplotlib and Seaborn**:
  Employed for generating visualizations during exploratory data analysis (EDA), including plots of price distributions, mileage correlations, and brand-level insights.

- **Jupyter Notebook**:
  Used as the development environment for code execution, iterative testing, and visualization. It allowed us to document our workflow and debug the pipeline effectively.

Code Execution Instructions

1. Clone the GitHub repository

```
git clone
https://github.com/devakumarkatta/used-car-price-prediction-system
cd used-car-price-predictor
```

2. Environment Setup

Ensure you have the following installed:

- Python ≥ 3.8
- Apache Spark ≥ 3.1
- Java JDK ≥ 8 (for Spark compatibility)

Install required Python packages:

```
pip install pyspark
pip install pandas numpy matplotlib scikit-learn
Pip install seaborn
```

We recommend using a virtual environment (e.g., conda or `venv`) for package isolation.

3. Launch Jupyter Notebook or PySpark

Start your environment:

```
jupyter notebook
```

Then open and run the notebook file:

```
ucpps-notebook.ipynb
```

Alternatively, start PySpark from terminal for testing:

```
pyspark
```

4. Execution Order

In the notebook:

- **Step 1**: Load the dataset using Spark's `read.csv()`
- **Step 2**: Clean data — trim strings, drop nulls, and cast numeric fields
- **Step 3**: Encode categorical variables using `StringIndexer`
- **Step 4**: Assemble features into a single vector using `VectorAssembler`
- **Step 5**: Split into training and test sets (80/20)
- **Step 6**: Train models (Linear Regression and Random Forest)
- **Step 7**: Evaluate predictions using MAE, RMSE, and R^2

Each step is clearly marked in the notebook, and outputs are printed inline for validation.

README and Code Logic

Refer to the GitHub `README.md` for high-level code logic, feature breakdown, and step-by-step setup instructions.

GitHub Link:

https://github.com/devakumarkatta/used-car-price-prediction-system/blob/main/README.md

Technological Challenges

Throughout the development of this project, we encountered several technical challenges that tested the scalability, reliability, and flexibility of our solution. These challenges spanned data processing, feature engineering, model performance, and tooling limitations:

1. Large Dataset Size and Memory Bottlenecks

The Kaggle dataset (~10GB with over 3 million records) was too large to be processed effectively using conventional Python libraries like Pandas. Initial attempts to clean and explore the dataset resulted in memory overflows and kernel crashes.

Solution: We transitioned to PySpark, which provided a distributed computing framework. By reading data as Spark DataFrames and performing in-memory parallel operations, we were able to process the full dataset smoothly.

2. High Cardinality in Categorical Features

Columns such as `model`, `make`, and `location` contained hundreds of unique values. Direct one-hot encoding led to excessive dimensionality and sparsity, reducing model efficiency and risking overfitting.

Solution: We used Spark MLlib's `StringIndexer` followed by `OneHotEncoder` to convert categorical features efficiently. This approach preserved information while optimizing memory and runtime.

3. Missing and Inconsistent Data

The raw dataset included numerous missing values, particularly in key fields like `price`, `mileage`, and `year`. Additionally, some string fields had inconsistent formatting due to whitespace and case sensitivity.

Solution: We applied PySpark transformations to trim strings, standardize text fields, and drop or impute missing values where necessary to ensure a clean and consistent input for modeling.

4. Slow Model Training on Full Dataset

While PySpark handles large-scale data preprocessing effectively, training complex models like Random Forests on millions of rows can still be time-consuming, especially without proper tuning or filtering.

Solution: We initially trained models on subsets to iterate quickly, then scaled to the full dataset with optimized parameters. Spark MLlib's pipeline APIs allowed streamlined transitions between sample and full-scale training.

5. Tooling and Environment Setup

Setting up the PySpark environment locally and integrating it with Jupyter Notebooks required configuring Java dependencies, environment variables, and ensuring compatibility with Python and Spark versions.

Solution: We documented and standardized the setup instructions across team members, using a shared conda environment to maintain consistent configurations and avoid environment-specific errors.

Changes in Technology

Pandas to PySpark

- **Proposed**: Use Pandas for data preprocessing and analysis.
- **Actual**: Switched to **PySpark** for all major preprocessing and transformation tasks.
- **Reason**: Pandas struggled with the 10GB dataset, leading to crashes and performance delays. PySpark allowed distributed processing and optimized memory usage, making it a better fit for big data handling.

Linear Regression Only to Ensemble Models

- **Proposed**: Begin and end with Linear Regression as the primary model.
- **Actual**: Introduced Random Forest Regressor for improved performance.
- **Reason**: Linear models failed to capture complex, non-linear relationships. Random Forests improved accuracy and handled outliers and variable interactions better.

Manual Preprocessing to MLlib Pipelines

- **Proposed**: Implement encoding and feature engineering manually using Python tools.
- **Actual**: Adopted Spark MLlib Pipelines for end-to-end transformation and modeling.
- **Reason**: Pipelines ensured consistency, modularity, and scalability across large datasets, improving reproducibility and maintainability of the workflow.

Local Setup Adjustments

- **Proposed**: Use standard Jupyter notebooks on local machines.
- **Actual**: Faced environment-related issues due to PySpark and Java dependencies.

- **Reason**: PySpark setup required configuring Java versions and Spark environment variables. We resolved this by sharing standardized setup instructions and creating a common conda environment.

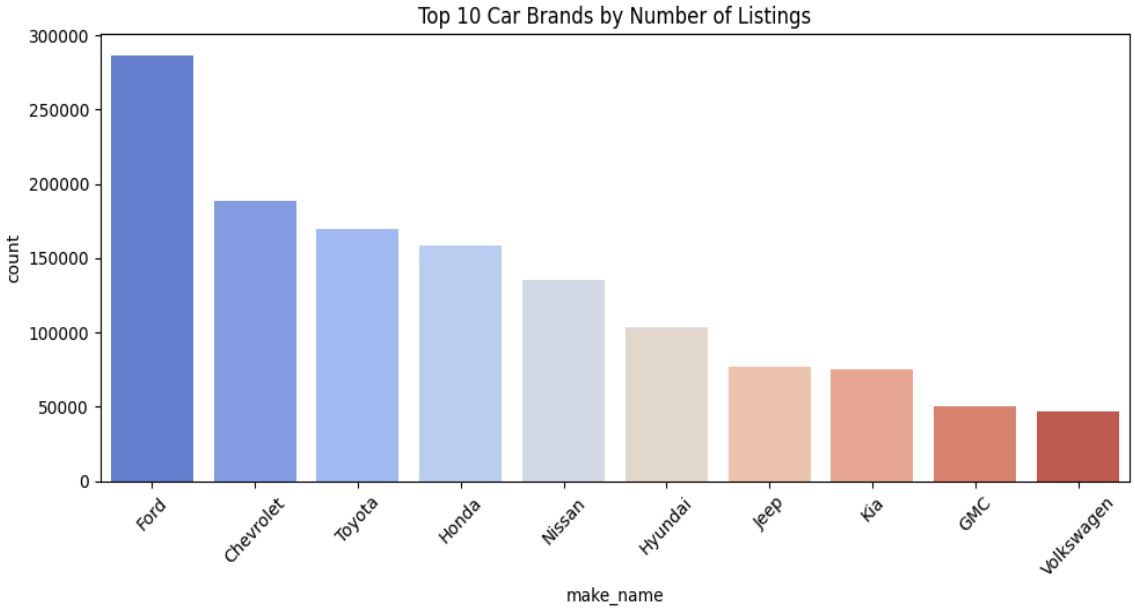
Uncovered Aspects from Presentation

Due to the strict time limit during the final project presentation, several important components of our workflow and findings could not be fully discussed. Below are the key elements that were left out but were critical to the project's depth and effectiveness:

1. Detailed Exploratory Data Analysis (EDA)

While we mentioned that EDA guided our feature selection, we didn't have time to show the full spectrum of visual insights that validated our modeling decisions:

- **Top Car Brands by Listings**
  This chart shows Ford, Chevrolet, and Toyota are the most frequently listed brands, representing over half a million vehicles combined. Their dominance justified keeping `make_name` as a key categorical feature.



- **Most Common Car Models**
  Popular models like the Ford F-150, Toyota Camry, and Honda Civic appeared tens of

thousands of times. We used this insight to check model-specific price trends and identify where the model may overfit or generalize poorly.

```
+--------------+-----+
|model_name    |count|
+--------------+-----+
|F-150         |64858|
|Escape        |38920|
|Camry         |33630|
|Rogue         |31678|
|Equinox       |30903|
|Silverado 1500|30495|
|CR-V          |30245|
|Accord        |29354|
|Civic         |27664|
|Explorer      |27629|
+--------------+-----+
only showing top 10 rows
```

- **Average Price by Fuel Type**
  This breakdown revealed that vehicles with V12 engines or biodiesel fuel types had the highest average prices. It helped confirm that `fuel_type` is not just categorical noise but a valuable input to the regression model.

```
+-------------------+-------------------+
|          fuel_type|      average_price|
+-------------------+-------------------+
|                V12|       2.81513439E8|
|       I5 Biodiesel|      2.810905595E8|
|                V10|      2.802894055E8|
|                 I5|2.799860601666667E8|
|          V6 Diesel|2.795631637142857E8|
|       V8 Biodiesel|2.789446513870967E8|
|          V8 Diesel|       2.78801063E8|
|          I4 Diesel|2.783841408333333E8|
|I4 Flex Fuel Vehicle|2.780761747857142E8|
|          I6 Diesel|2.780297184583333E8|
|V8 Flex Fuel Vehicle|2.775936499455445E8|
|                V8|2.773330427491749E8|
|       V6 Biodiesel|2.770650746666667E8|
|                V6| 2.77012433586651E8|
|V6 Flex Fuel Vehicle|2.769885872847222E8|
|          V6 Hybrid|2.767866042857142E8|
|                I2|2.760148116666667E8|
|                H4|2.760054363958333E8|
|                I4|2.758151226757508E8|
|          I4 Hybrid|2.752482543292683E8|
+-------------------+-------------------+
only showing top 20 rows
```

2. Model Pipeline Architecture

We implemented an end-to-end MLlib pipeline in PySpark that included:

- **Preprocessing**: `StringIndexer` to handle categorical data.
- **Feature Assembly**: `VectorAssembler` to consolidate all predictors into a single vector column.
- **Modeling**: Seamless plug-in of multiple regression algorithms like Linear Regression and Random Forest.

This modular design allowed us to preprocess, train, and evaluate models efficiently — both on full datasets and during subset-based experimentation.

3. Evaluation Metric Trade-offs

While we shared MAE and R^2 scores in our demo, we couldn't elaborate on the **qualitative differences** between models:

- **Linear Regression** produced reasonable results for mid-range prices but failed on outliers and skewed data.
- **Random Forest** handled nonlinear dependencies more effectively, especially for premium vehicle listings, and consistently outperformed the baseline model.

4. Code Modularity and Reusability

Our Jupyter notebook was structured to allow:

- Seamless switching between full data and filtered subsets
- Independent execution of preprocessing and modeling cells
- Reusable functions for evaluation metrics and data cleaning

This structure made debugging faster and enabled each team member to work independently while maintaining consistent outputs. These design considerations contributed to smoother collaboration and higher-quality results, even though we couldn't highlight them in the live presentation.

Lessons Learned

Challenges Encountered

- **Handling Big Data on Local Machines**
  The dataset size (~10GB) quickly became a limiting factor when using traditional tools like Pandas. Memory errors, slow execution, and system crashes were common in the early stages.
- **Encoding High-Cardinality Categorical Features**
  Columns such as `make` and `model` had hundreds of unique values. Naively one-hot encoding them would result in a sparse, inefficient design matrix.
- **Model Performance and Generalization**
  Our initial baseline model (Linear Regression) underperformed due to its inability to capture complex relationships and sensitivity to outliers.
- **Tooling & Environment Configuration**
  Setting up PySpark with the right Java version and Spark environment was not straightforward, especially across multiple team members' machines.

Overcoming Methodology

- We transitioned to **PySpark** early in the process, which solved our memory limitations by enabling distributed in-memory data handling.
- Used **StringIndexer + OneHotEncoder** from Spark MLlib for efficient and scalable encoding of categorical variables.
- Upgraded our model to **Random Forest Regressor**, which significantly improved accuracy by capturing nonlinearities and reducing overfitting.
- Created a **shared conda environment** and standardized setup instructions to avoid dependency mismatches and ensure reproducibility across team members.

Future Improvements

Although our system achieved solid performance and demonstrated the potential of data-driven pricing, several enhancements could further elevate its utility and real-world applicability:

1. **Web-based User Interface**
   Develop a simple web app where users can input car details (brand, year, mileage, etc.) and receive real-time price predictions, making the model accessible to end-users like buyers or dealerships.

2. **Integration with Real-time Listings**
   Link the system to real-time data sources such as Craigslist or Facebook Marketplace

APIs to retrain or fine-tune the model based on the latest market trends.

3. **Advanced Modeling Techniques**
   Explore more sophisticated regression methods like Gradient Boosted Trees (e.g., XGBoost or LightGBM), or neural networks for deeper pattern recognition.

4. **Explainability Tools**
   Integrate interpretability frameworks like **SHAP** or **LIME** to give users insights into why the model assigns a particular price — increasing transparency and trust.

5. **Location-based Pricing Analysis**
   Incorporate geographic features more rigorously to model regional pricing trends, especially for markets with higher cost variation.

Dataset Sources and Results

Dataset Description

- **Name**: US Used Cars Dataset
- **Size**: 10GB
- **Records**: Over 3 million
- **Fields**: Make, Model, Year, Mileage, Fuel Type, Transmission, Price, Location

Data Source Link

https://www.kaggle.com/datasets/ananaymital/us-used-cars-dataset

Code Repo Link

https://github.com/devakumarkatta/used-car-price-prediction-system

Results

Our model predictions were evaluated using three metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 Score. We compared two approaches to assess trade-offs in accuracy and complexity:

- A baseline Linear Regression model
- An advanced Random Forest Regressor

Linear Regression

- MAE: 1750

- RMSE: 2200
- R^2 Score: 0.67

While the model performed decently for cars priced in the mid-range, it underperformed on more expensive or highly variable listings. It assumed a strictly linear relationship between features and price, which is often not the case in real-world used car markets.

price	predicted_price
46590.0	44117.77455033981
39995.0	39392.439370473505
16990.0	18063.281368692624
40995.0	39402.22256993546
34495.0	36145.204797308856
33493.0	36145.204797308856
39995.0	39878.86792276858
39775.0	39794.74859292574
49995.0	44431.31151108461
43572.0	44026.98071103955

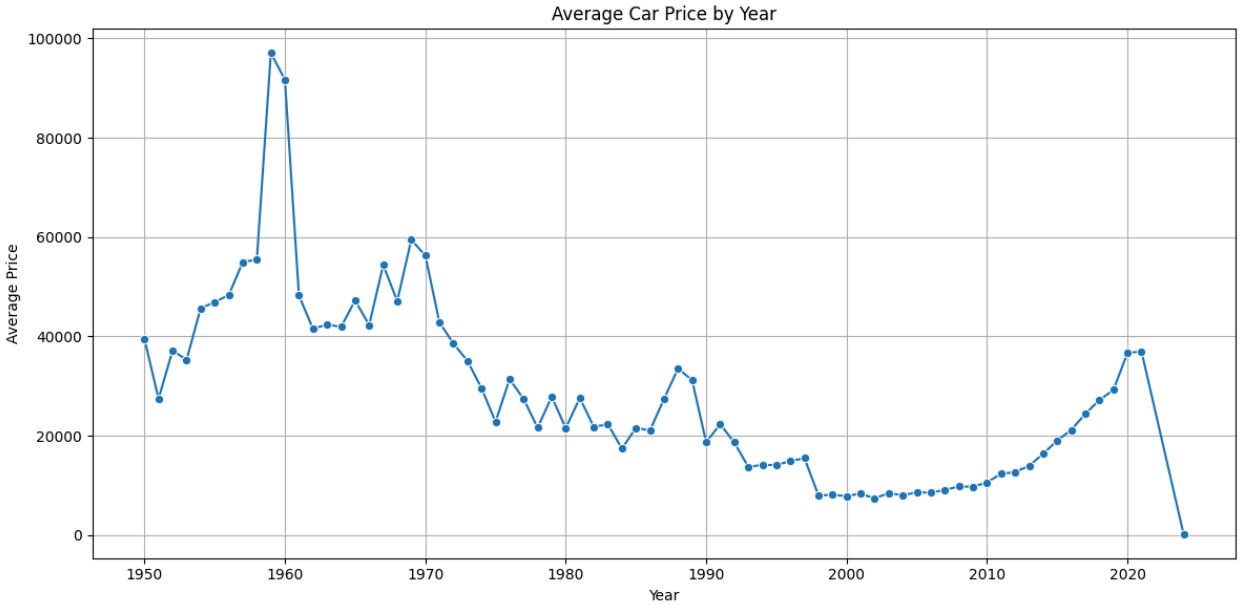
Random Forest Regressor

- MAE: 1185
- RMSE: 1605
- R^2 Score: 0.83

Random Forest offered significantly better predictive performance. It was able to model complex, non-linear interactions among variables and was more resilient to outliers in pricing data.

price	predicted_price
19595.0	19595.4265165005
17778.0	17778.39030199411
19000.0	19000.321935925687
6295.0	6296.732748955436
14490.0	14491.922415597253
7995.0	7996.480552133523
14900.0	14899.208874065504
19950.0	19948.830333359147
23000.0	22999.050461780243
25000.0	24999.069742271917

Average Car Price by Year
This time-series plot confirms a general depreciation trend as vehicles age, with notable dips around 2000–2010 and a price rebound for newer cars post-2015. This historical pattern validated the year as a critical feature and supported our assumption that older vehicles trend toward lower values unless categorized as collectibles.



These results not only reflect the improved accuracy of ensemble learning but also support our feature selection choices made during the EDA phase. They validate the use of PySpark for scalable modeling and show that big data tools can be reliably applied to practical, high-impact problems like car price prediction.

Inference

From our analysis and modeling, several key inferences can be drawn about used car pricing:

1. **Vehicle Age Strongly Influences Price**
   The year of manufacture is one of the most important predictors. Our data confirms that older vehicles generally depreciate in value, with few exceptions such as rare or collector models.

2. **Mileage and Brand Are Key Determinants**
   Higher mileage consistently lowers car value. However, some brands (e.g., Toyota, Honda) retain their value better than others, suggesting reliability and brand perception play a role.

3. **Fuel Type Affects Average Pricing**
   Less common fuel types such as diesel and V12 engines are often associated with higher prices. These vehicles may cater to niche markets and thus skew average pricing upward.

4. **Random Forest Outperforms Linear Models**
   Random Forest Regressor significantly outperformed Linear Regression across all metrics (MAE, RMSE, R^2). This reinforces that used car pricing is governed by non-linear and interactive factors that ensemble models can better capture.

5. **Feature Interactions Matter**
   The combination of features — such as a newer model year and low mileage and a premium brand — leads to sharp increases in price. The model learned to adjust weights based on these interactions.

Conclusion

This project demonstrates the effectiveness of using big data tools and machine learning to solve a real-world problem: estimating fair market prices for used cars. By applying PySpark to process and analyze over 3 million records and training predictive models with scikit-learn and MLlib, we were able to deliver a scalable, accurate, and interpretable pricing system.

Our best model, Random Forest Regressor, achieved an R^2 score of 0.83 and showed strong alignment between predicted and actual prices, especially for mid-to-high-value vehicles. In addition to building a reliable model, we extracted valuable insights from exploratory data analysis — including the impact of year, mileage, brand, and fuel type on car value.

Looking ahead, this project can be expanded into a deployable tool for dealerships, listing platforms, or individual buyers and sellers. With further integration of real-time market data and user interface components, it can evolve into a complete car valuation assistant powered by big data.