Code Crunchers : Comprehensive Analysis and Optimization of Car Market Dynamics

Pravallika Mummadi, Tagore charith Chepuri, Manoj Vamanaguntla, Chandra Venkata Vijaya Gopal Raju Kalidindi

Northwest Missouri State University, Maryville MO 64468, USA
S555592@nwmissouri.edu,
S559255@nwmissouri.edu,s559007@nwmissouri.edu,s560460@nwmissouri.edu

1 44517-02

2 Team name: Code Crunchers

3 Team Members

- Pravallika Mummadi
- Tagore charith Chepuri
- Manoj Vamanaguntla
- Chandra Venkata Vijaya Gopal Raju Kalidindi

4 Project Title

Code Crunchers : Comprehensive Analysis and Optimization of Car Market Dynamics

5 Project Idea

Develop a comprehensive system that predicts car prices accurately and provides personalized recommendations to potential buyers based on their preferences and budget constraints.

6 Technology Summary

Data Cleaning: Python (Pandas, NumPy)
PySpark: For handling large-scale data analysis. PySpark provides the ability to work with big data efficiently using the Spark framework.

7 Architecture Diagram

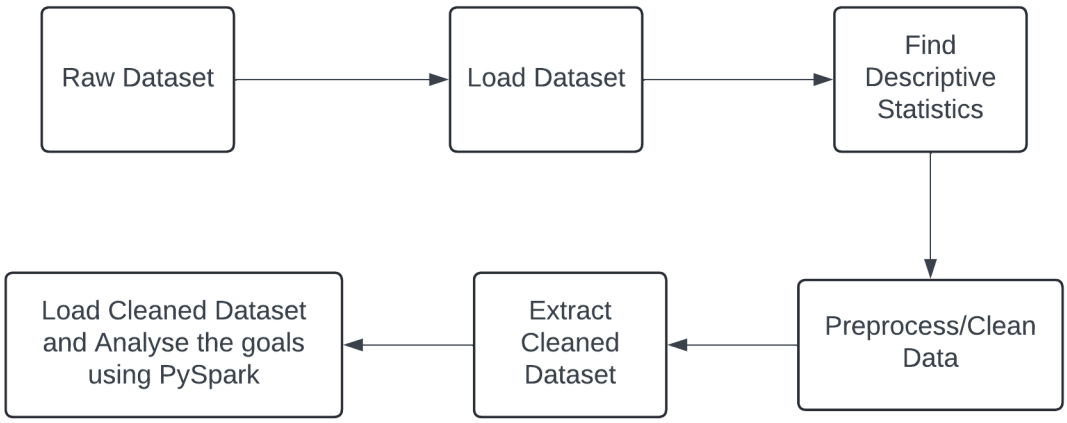

Fig. 1. Architecture Diagram

8 Architecture Summary

8.1 Collect Dataset

This process represents gathering data from various sources or databases.

8.2 Load Dataset

Using the Pandas library in Python, the collected dataset is loaded for further analysis.

8.3 Find Descriptive Statistics

Descriptive statistics and exploratory data analysis are performed on the loaded dataset to understand its characteristics.

8.4 Preprocess/Clean Data

Unwanted columns and data are removed, and necessary data cleaning operations are performed to prepare the data for analysis.

8.5 Extract Cleaned Dataset

The cleaned dataset is extracted after preprocessing, ready for analysis.

8.6 Load Cleaned Dataset

The cleaned dataset is loaded into Pyspark, for creating interactive analysis.

8.7 Analyzing Project Goals

Project goals are analyzed to determine the key metrics and insights to be visualized.

9 Project Goals

9.1 Goal-1

Fuel Efficiency Analysis: Analyze the Fuel Type And Mileage of the vehicles.

9.2 Goal-2

Identifying Under priced Cars helping consumers identify potentially undervalued cars.

9.3 Goal-3

Market Segmentation: Segment the market based on Owner Type.

9.4 Goal-4

Price Elasticity Analysis: Analyze the relationship between Price and other attributes such as Mileage, Engine, and Power to understand the price sensitivity of consumers and adjust pricing strategies accordingly.

9.5 Goal-5

Trend Analysis for New Price: Analyze trends in New Price over time to understand how prices of different car models have evolved.

10 Project Description

This project explores the automotive market using data analysis techniques and tools such as Python, Pandas, NumPy, and PySpark, focusing on five main objectives.
Firstly, it conducts a Fuel Efficiency Analysis to evaluate vehicles by fuel type and mileage, helping consumers make informed decisions about fuel economy.
Secondly, it identifies Underpriced Cars by comparing their market prices to intrinsic values based on their features and performance, aiding consumers in finding great deals.
Thirdly, the project segments the market based on owner type to uncover unique buying patterns, which assists automotive businesses in refining their marketing approaches.
The fourth goal, Price Elasticity Analysis, examines how sensitive consumer demand is to price changes, considering other factors like mileage, engine size, and power.
Lastly, Trend Analysis for New Price tracks and analyzes car price trends over time, providing insights for future pricing strategies and historical price changes.
Together, these analyses leverage statistical and machine learning methods to deliver insights that promote smarter decision-making and strategic planning in the automotive sector.

11 Implementation steps

11.1 Data Cleaning and Processing

- **Import Necessary Libraries:**
  - Import the pandas library as pd, which is commonly used for data manipulation and analysis.
  - Import the numpy library as np, which adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- **Load Data:**
  - Load data from a CSV file named 'used_cars_data.csv' into a DataFrame. This step utilizes pandas' `read_csv` function.
  - Display the first few rows of the DataFrame using the `head()` method to verify correct data loading and to get an initial look at the data structure.
- **Data Cleaning and Preparation:**
  - Identify columns with object data types that may need special handling for missing values. These columns are identified as 'Engine', 'New_Price', 'Mileage', and 'Power'.
  - For each of these columns, replace missing values with the most frequent value (mode) in the column. This is done using pandas' `mode()` method, and filling missing values with `fillna()` method.
- **Handling Missing Values in 'Price' Column:**
  - Calculate the mean of the 'Price' column using the `mean()` method.

- Fill missing values in the 'Price' column with this mean value, ensuring that any missing price information is reasonably estimated based on available data.

– **Adjust DataFrame Structure:**
  - Remove unnecessary columns such as 'Location' and 'Seats' using the `drop()` method with the `inplace=True` parameter to modify the original DataFrame.
  - Verify the changes by displaying the first few rows of the updated DataFrame using the `head()` method.
– **Output DataFrame Shape:**
  - Display the shape of the DataFrame, which shows the number of rows and columns, using the `shape` attribute. This provides a summary of the DataFrame's dimensions after cleaning.
– **Save Cleaned Data:**
  - Save the cleaned DataFrame to a new CSV file named 'output.csv'. Ensure that the index is not included in the output file by setting `index=False` in the `to_csv()` method.

11.2 Goal-1 implementation steps:

– **Import Necessary Libraries:**
  - Import SparkSession from pyspark.sql, which is essential for initializing the Spark context and creating the Spark session.
  - Import various functions from pyspark.sql.functions, including col, regexp_extract, and functions for aggregations like max, sum, and min.
  - Import Window from pyspark.sql for potential window functions, although it's not used in the given code snippet.
– **Initialize Spark Session:**
  - Create a Spark session to manage Spark operations. If there is an existing Spark session, it will use that; otherwise, it creates a new one.
– **Read Data:**
  - Read a CSV file named "output.csv" into a DataFrame. This step assumes the CSV file includes a header row and that the data types in the columns are automatically inferred.
– **Data Manipulation - Regular Expression:**
  - Modify the 'Mileage' column to extract numerical values using a regular expression that captures patterns like "18 kmpl". This extracts the mileage as a float number into a new column named 'Mileage_km_per_liter'.
– **Calculate Average Mileage:**
  - Group the data by 'Fuel_Type'.
  - Calculate the average mileage for each fuel type and rename the resulting column to 'Average_Mileage'.
– **Display Results:**
  - Show the DataFrame containing the average mileage per fuel type using the .show() method. This will print the results to the console.
– **Note:**
  - Ensure that your Spark environment is correctly set up before running this script. The DataFrame used is assumed to be named 'cars'.

11.3 Goal-2 implementation steps:

- **Import Necessary Libraries:**
  - Import the necessary modules from PySpark, including SparkSession, various functions for data manipulation, machine learning features like VectorAssembler, StringIndexer, and OneHotEncoder, the RandomForestRegressor for the regression model, the Pipeline to streamline ML workflows, and RegressionEvaluator for model evaluation.
- **Initialize Spark Session:**
  - Create and configure a Spark session with an application name "Identifying Underpriced Cars".
- **Load and Prepare Data:**
  - Load data from a CSV file named 'output.csv', ensuring the file includes headers and inferSchema is set to True to automatically detect column data types.
  - Clean data by removing units from 'Mileage', 'Engine', and 'Power' columns and converting them to appropriate numeric types.
  - Drop rows with missing values in key columns like 'Mileage', 'Engine', 'Power', and 'Price' to prepare for analysis.
- **Feature Engineering:**
  - Use StringIndexer to convert categorical columns ('Fuel_Type', 'Transmission', 'Owner_Type') into numeric indices.
  - Apply OneHotEncoder to these indices to create binary vector columns for each category, suitable for model input.
  - Use VectorAssembler to combine all feature columns (numeric and vectorized categorical) into a single feature vector.
- **Machine Learning Model:**
  - Define a RandomForestRegressor to predict the 'Price' based on assembled features.
  - Create a Pipeline comprising the stages of indexing, encoding, assembling, and regression to streamline the model building process.
- **Model Training and Evaluation:**
  - Split the data into training and test datasets (80% training, 20% testing).
  - Fit the model to the training data and use it to make predictions on the test data.
  - Evaluate the model using the Root Mean Squared Error (RMSE) metric to assess prediction accuracy.
- **Identifying Underpriced Cars:**
  - Calculate the difference between the actual price and the predicted price.
  - Filter out cars where the actual price is greater than the predicted price (indicating underpricing).
  - Select relevant columns ('Name', 'Year', 'Price', 'prediction', 'Price_Difference') for further analysis or display.
- **Output Results:**
  - Save the DataFrame of underpriced cars to a CSV file, ensuring that the results are easy to access and review.

11.4 Goal-3 implementation steps:

- **Import Necessary Libraries:**
  - Import SparkSession from pyspark.sql to manage Spark operations and create the Spark session.
- **Initialize Spark Session:**
  - Initialize a Spark session with an application name "MarketSegmentation". This session acts as the entry point for programming Spark with the Dataset and DataFrame API.
- **Load Data:**
  - Load data from a CSV file named "output.csv" using the Spark session. The 'header' option is set to True to use the first line of the file as names for columns. The 'inferSchema' option is enabled, allowing Spark to automatically deduce the types of the columns.
- **Data Analysis:**
  - Group the dataset by the "Owner_Type" column to analyze the market segmentation based on the type of ownership.
  - Count the number of records in each group to understand the distribution of cars by ownership type.
  - Order the results by "Owner_Type" for systematic presentation and analysis.
- **Display Results:**
  - Use the `show()` method to print the results, which will display the counts of each owner type, thus providing insights into the market segmentation.
- **Further Analysis:**
  - Based on the grouped data, further statistical analysis or visualization can be performed to extract deeper insights into market trends or to inform business strategies.

11.5 Goal-4 implementation steps:

- **Import Necessary Libraries:**
  - Import SparkSession from pyspark.sql to initialize the Spark context.
  - Import functions such as regexp_replace from pyspark.sql.functions for data cleaning.
  - Import VectorAssembler, StringIndexer, OneHotEncoder from pyspark.ml.feature for feature engineering.
  - Import LinearRegression from pyspark.ml.regression for the prediction model.
  - Import Pipeline from pyspark.ml to build the sequence of data processing and modeling steps.
  - Import RegressionEvaluator from pyspark.ml.evaluation to evaluate the model's performance.
- **Initialize Spark Session:**
  - Create a Spark session with the application name "Price Elasticity Analysis".

- **Load Data:**
  - Load data from a CSV file named 'output.csv', ensuring it includes headers and the schema is inferred.
- **Data Cleaning and Preparation:**
  - Clean the 'Mileage', 'Engine', and 'Power' columns by removing units and converting to appropriate numeric types.
  - Drop rows with any missing values in the 'Mileage', 'Engine', 'Power', or 'Price' columns to prepare clean data for analysis.
- **Feature Engineering:**
  - Apply StringIndexer to convert categorical columns into numeric indices.
  - Use OneHotEncoder to convert these indices into binary vector columns suitable for model input.
  - Assemble all feature columns into a single vector using VectorAssembler.
- **Modeling and Prediction:**
  - Define a Linear Regression model with 'Price' as the label column.
  - Create a Pipeline comprising stages of indexing, encoding, assembling, and regression.
  - Split the data into training (70%) and test (30%) sets.
  - Fit the model on the training data and use it to make predictions on the test data.
- **Output Results:**
  - Select only the 'Price' and 'prediction' columns from the predictions.
  - Save these results to a CSV file, facilitating easy access and further analysis.
- **Model Evaluation:**
  - Evaluate the model using the R2 metric to assess its performance in explaining the variance.
  - Print the R2 score to understand the model's effectiveness.

11.6 Goal-5 implementation steps:

- **Import Necessary Libraries:**
  - Import SparkSession from pyspark.sql to manage Spark operations.
  - Import functions such as avg, col, year, and regexp_replace from pyspark.sql.functions for data manipulation.
- **Initialize Spark Session:**
  - Create a Spark session with the application name "Car Price Trend Analysis".
- **Load Data:**
  - Load data from a CSV file named 'output.csv'. Set 'header' to True to use the first line as column names, and 'inferSchema' to True to automatically detect column types.
- **Data Cleaning:**
  - Convert the 'New_Price' column, which is assumed to be in a format like '1.5 Lakh', into a numeric format by removing the text ' Lakh' and multiplying by 100,000 to convert the amount to a plain number format.

– **Analyzing Trends in Price Over Time:**
  - Group the data by 'Name' and 'Year'.
  - Calculate the average 'Price' for each group to analyze how the prices of cars have trended over the years.
  - Order the results by 'Year' to see the trend chronologically.
– **Output Results:**
  - Save the results of the trend analysis to a CSV file named 'trend_analysis_output.csv' with headers. This will overwrite any existing file with the same name.
  - Coalesce the data into one partition before writing, ensuring that the output is consolidated into a single file for easy review.

12 Results Summary

12.1 Code Snippets

```
[1]: import pandas as pd
     import numpy as np

[2]: df=pd.read_csv("used_cars_data.csv")
     df.head()
```

Fig. 2. Loading actual dataset to pandas and numpy modules

```
[5]: object_columns = ['Engine', 'New_Price','Mileage', 'Power']
     # Replace missing values in these columns with the most frequent value of each column
     for col in object_columns:
         most_frequent = df[col].mode()[0]  # mode() returns a Series, [0] gets the most frequent element
         df[col] = df[col].fillna(most_frequent)  # Direct assignment avoids the FutureWarning

     print(df)
```

```
      S.No.                                          Name    Location  \
0         0                    Maruti Wagon R LXI CNG      Mumbai
1         1                Hyundai Creta 1.6 CRDi SX Option        Pune
2         2                              Honda Jazz V     Chennai
3         3                           Maruti Ertiga VDI     Chennai
4         4                Audi A4 New 2.0 TDI Multitronic  Coimbatore
...     ...                                          ...         ...
7248   7248            Volkswagen Vento Diesel Trendline   Hyderabad
7249   7249                    Volkswagen Polo GT TSI      Mumbai
7250   7250                    Nissan Micra Diesel XV     Kolkata
7251   7251                    Volkswagen Polo GT TSI        Pune
7252   7252  Mercedes-Benz E-Class 2009-2013 E 220 CDI Avan...       Kochi

      Year  Kilometers_Driven Fuel_Type Transmission Owner_Type     Mileage  \
0     2010              72000       CNG       Manual      First  26.6 km/kg
1     2015              41000    Diesel       Manual      First  19.67 kmpl
2     2011              46000    Petrol       Manual      First   18.2 kmpl
3     2012              87000    Diesel       Manual      First  20.77 kmpl
4     2013              40670    Diesel    Automatic     Second   15.2 kmpl
```

Fig. 3. Performing Imputations to the columns where data is null to mean value

```
[7]: # Calculate the mean price
     mean_price = df['Price'].mean()

     # Use recommended approach to avoid chained assignment
     df['Price'] = df['Price'].fillna(mean_price)
```

```
[8]: df.isnull().sum().sort_values(ascending = False)
```

```
[8]: Seats               53
     S.No.                0
     Name                 0
     Location             0
     Year                 0
     Kilometers_Driven    0
     Fuel_Type            0
     Transmission         0
     Owner_Type           0
     Mileage              0
     Engine               0
     Power                0
     New_Price            0
     Price                0
     dtype: int64
```

Fig. 4. Performing Imputations to the columns where data is null to mean value on Price and checking if there are any null values.

```
[10]: df.drop(columns=['Location','Seats'],inplace=True)
      df.head()
```

[10]:

	S.No.	Name	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	New_Price	Price
0	0	Maruti Wagon R LXI CNG	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	33.36 Lakh	1.75
1	1	Hyundai Creta 1.6 CRDi SX Option	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	33.36 Lakh	12.50
2	2	Honda Jazz V	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	8.61 Lakh	4.50
3	3	Maruti Ertiga VDI	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	33.36 Lakh	6.00
4	4	Audi A4 New 2.0 TDI Multitronic	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	33.36 Lakh	17.74

Fig. 5. Dropping unwanted columns

```
[14]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   S.No.              7253 non-null   int64
 1   Name               7253 non-null   object
 2   Year               7253 non-null   int64
 3   Kilometers_Driven  7253 non-null   int64
 4   Fuel_Type          7253 non-null   object
 5   Transmission       7253 non-null   object
 6   Owner_Type         7253 non-null   object
 7   Mileage            7253 non-null   object
 8   Engine             7253 non-null   object
 9   Power              7253 non-null   object
 10  New_Price          7253 non-null   object
 11  Price              7253 non-null   float64
dtypes: float64(1), int64(3), object(8)
memory usage: 680.1+ KB

[15]: df.to_csv('output.csv', index=False)
```

Fig. 6. Generating new data set from the cleaned dataset

12.2 Goal-1

Fig. 7. Goal-01 source code

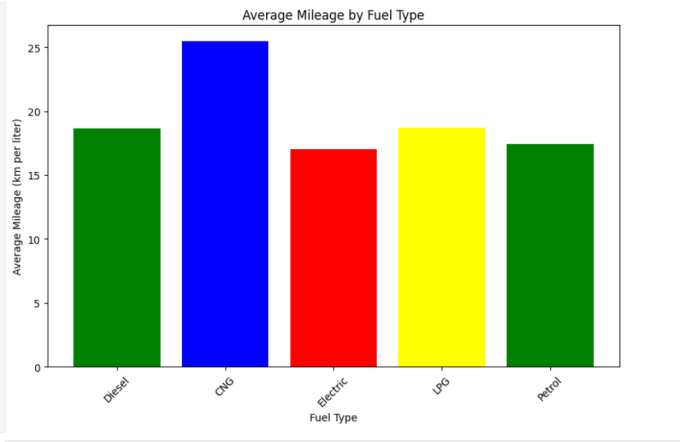

Fig. 8. Goal-01 Output

The bar chart analysis conducted as part of our project's first goal reveals compelling insights into the fuel efficiency of various vehicle types. It is evident from the visual data that vehicles running on CNG offer the highest average mileage, which indicates a potential market advantage in terms of fuel economy. On the contrary, petrol and LPG vehicles exhibit lower average mileage, suggesting a higher running cost for these fuel types. The competitive nature of vehicle fuel options is apparent, with each type catering to different consumer needs and efficiency standards.

During our initial data handling, challenges such as unnamed columns and null values were encountered. These issues were systematically addressed through rigorous data cleaning processes, ensuring a high standard of data quality. The application of the matplotlib library for visualization underscores the veracity of our data, reinforcing the reliability of our conclusions. Furthermore, the execution time of approximately 5 minutes to run the Spark code and visualize the results demonstrates the efficiency and scalability of the PySpark framework. Additionally, the open-source nature of this framework presents a cost-effective solution for our analytical requirements.

In conclusion, the insights garnered from this analysis are not only indicative of the fuel efficiency landscape within the automotive industry but also reflect the robustness of our data processing and visualization pipeline. Such insights are vital for stakeholders looking to understand and optimize fuel consumption patterns, aligning with the broader goals of cost-effectiveness and environmental sustainability.

12.3 Goal-2

```
*[13]: from pyspark.sql import SparkSession
from pyspark.sql.functions import regexp_replace, col
from pyspark.ml.feature import VectorAssembler, StringIndexer, OneHotEncoder
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator

# Initialize Spark Session
spark = SparkSession.builder.appName("Identifying Underpriced Cars").getOrCreate()
# Load the data
df = spark.read.csv('output.csv', header=True, inferSchema=True)
# Data cleaning and preparation
df = df.withColumn('Mileage', regexp_replace('Mileage', ' kmpl| km/kg', '').cast('float'))
df = df.withColumn('Engine', regexp_replace('Engine', ' CC', '').cast('int'))
df = df.withColumn('Power', regexp_replace('Power', ' bhp', '').cast('float'))
# Handle missing values
df = df.na.drop(subset=['Mileage', 'Engine', 'Power', 'Price'])
# Feature Engineering: Encoding categorical variables
stringIndexer = StringIndexer(inputCols=['Fuel_Type', 'Transmission', 'Owner_Type'],
                              outputCols=['Fuel_Type_Indexed', 'Transmission_Indexed', 'Owner_Type_Indexed'])
oneHotEncoder = OneHotEncoder(inputCols=['Fuel_Type_Indexed', 'Transmission_Indexed', 'Owner_Type_Indexed'],
                              outputCols=['Fuel_Type_Vec', 'Transmission_Vec', 'Owner_Type_Vec'])
# Assembling all features into a single vector
assembler = VectorAssembler(inputCols=['Year', 'Kilometers_Driven', 'Mileage', 'Engine', 'Power',
                                       'Fuel_Type_Vec', 'Transmission_Vec', 'Owner_Type_Vec'],
                            outputCol="features")
# Random Forest Regression Model
rf = RandomForestRegressor(featuresCol="features", labelCol="Price")
# Pipeline
pipeline = Pipeline(stages=[stringIndexer, oneHotEncoder, assembler, rf])
# Split the data into training and test sets
trainData, testData = df.randomSplit([0.8, 0.2])
# Fit the model
model = pipeline.fit(trainData)
# Make predictions
predictions = model.transform(testData)
# Evaluate the model
evaluator = RegressionEvaluator(labelCol="Price", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print(f"Root Mean Squared Error (RMSE) on test data: {rmse}")
# Identifying underpriced cars
underpriced_cars = predictions.withColumn("Price_Difference", col("Price") - col("prediction"))
underpriced_cars = underpriced_cars.filter(col("Price_Difference") > 0).select("Name", "Year", "Price", "prediction", "Price_Difference")
# Save underpriced cars to CSV
underpriced_cars.write.option("header", "true").mode("overwrite").csv("underpriced_cars.csv")
# Save the DataFrame to a CSV file
predictions.withColumn("Price_Difference", predictions["Price"] - predictions["prediction"]) \
    .select("Name", "Year", "Price", "prediction", "Price_Difference") \
    .coalesce(1) \
    .write \
    .option("header", "true") \
    .mode("overwrite") \
    .csv("underpriced_cars_output.csv")

Root Mean Squared Error (RMSE) on test data: 5.454598492067431
```

Fig. 9. Goal-02 source code

	Name	Year	Price	prediction	Price_Difference
1	Nissan Micra Diesel XV	2013	3.5	5.1885886210843495	-1.6885886210843495
2	Vento Diesel Comfortline	2013	5.2	5.9392095288921105	-0.7392095288921103
3	Indica Vista Quadrajet LS	2012	1.95	4.709062798319657	-2.7590627983196567
4	Maruti Ciaz Zeta	2018	9.95	7.186514961562297	2.763485038437702
5	tsubishi Pajero Sport 4X4	2014	15.0	15.73083236186621	-0.7308323618662094
6	C 220 CDI BE Avantgare	2014	28.0	18.076982579455414	9.923017420544586
7	2015 35 TFSI Technology	2015	23.5	18.30271213900163	5.197287860998369
8	da City Corporate Edition	2012	4.25	5.384524974301385	-1.1345249743013852
9	i Alto K10 2010-2014 VXI	2013	2.75	4.176332614178639	-1.426332614178639
10	Honda WRV i-VTEC VX	2018	9.9	6.917403569034924	2.982596430965076
11	Toyota Corolla Altis G	2012	6.75	5.827779448726944	0.9222205512730559
12	Maruti Alto LXi	2008	1.25	3.4386829579066793	-2.1886829579066793
13	Hyundai i10 Asta	2010	2.07	4.232364416805527	-2.1623644168055276
14	ss C 200 CGI Avantgarde	2015	26.7	18.20710846560837	8.492891534391628
15	Tata Indica V2 eLS	2016	2.5	5.856615429076497	-3.3566154290764967
16	ayenne 2009-2014 Turbo	2008	14.5	19.322185854652272	-4.822185854652272
17	Nissan Terrano XV D Pre	2015	6.92	7.8274382555638224	-0.9074382555638225
18	Honda Amaze SX i-VTEC	2016	4.82	6.767077708997489	-1.9470777089974884
19	Maruti Ritz VDi	2013	3.11	5.369145614768823	-2.259145614768823
20	Honda Amaze SX i-VTEC	2016	5.0	6.8537659536551585	-1.8537659536551585
21	ahindra XUV500 W8 2WD	2014	8.1	8.487834538177065	-0.3878345381706537
22	Innova Crysta 2.8 ZX AT	2017	19.25	24.669649763856565	-5.419649763856565
23	2.5 VX (Diesel) 7 Seater	2014	11.11	8.677798384784545	2.432016152154546
24	Maruti Wagon R LXI CNG	2013	3.25	4.485169703469668	-1.2351697034696683
25	Honda CR-V 2.4 MT	2007	5.25	7.17109471770309	-1.92109471770309
26	Honda Amaze SX i-DTEC	2014	3.7	5.886747190189133	-2.1867471901891324

Fig. 10. Goal-2 Output

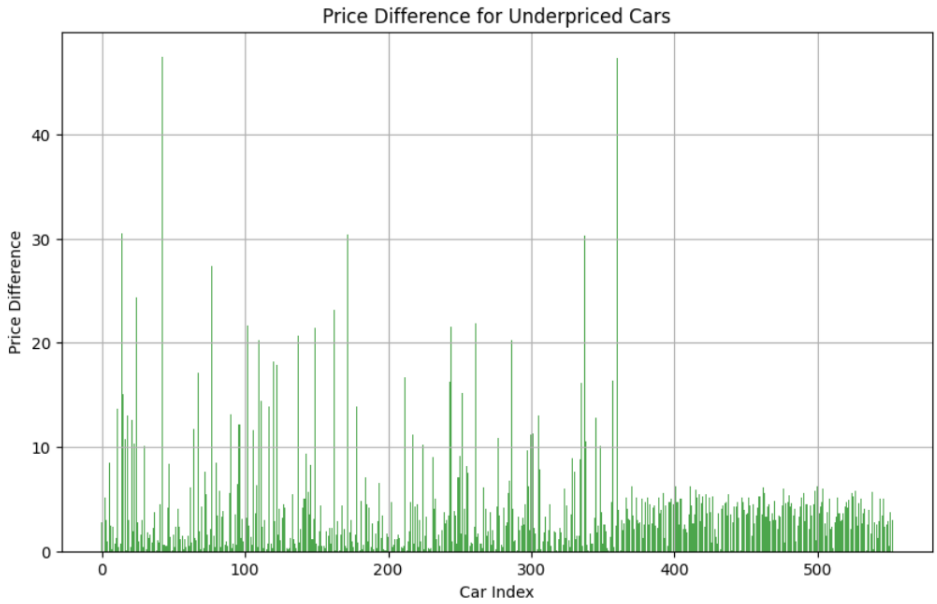

Fig. 11. Goal-2 Output

The histogram underscores the price differentials for cars classified as underpriced within the studied dataset. Each bar represents an individual car's index and the extent to which its selling price is below the model-predicted value. Most cars appear to have a modest undervaluation, but several outliers are significantly underpriced, with differences stretching up to 40 units.

This variance in price difference suggests a range of factors could be influencing the final selling price of these vehicles, from their condition and mileage to market-specific dynamics such as demand and availability. The presence of such underpriced cars in the market could represent potential bargains for cost-conscious consumers or indicate areas where sellers could optimize pricing strategies.

From a data analysis perspective, this graph illustrates the potential of machine learning models to identify pricing anomalies in large datasets, enabling stakeholders to pinpoint and investigate unusual instances efficiently. Such insights could lead to corrective actions in pricing strategies or further market analysis to understand the underlying causes of these price discrepancies.

12.4 Goal-3

[9]:
```
# GOAL 3:
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder.appName("MarketSegmentation").getOrCreate()

# Load data
data = spark.read.csv("output.csv", header=True, inferSchema=True)

# Group by Owner Type and count the number of records in each segment
segment_counts = data.groupBy("Owner_Type").count().orderBy("Owner_Type")

# Display the counts for each segment
segment_counts.show()

# You can perform further analysis or visualization on each segment as needed
```

```
+--------------+-----+
|    Owner_Type|count|
+--------------+-----+
|         First| 5952|
|Fourth & Above|   12|
|        Second| 1152|
|         Third|  137|
+--------------+-----+
```

Fig. 12. Goal-03 source code

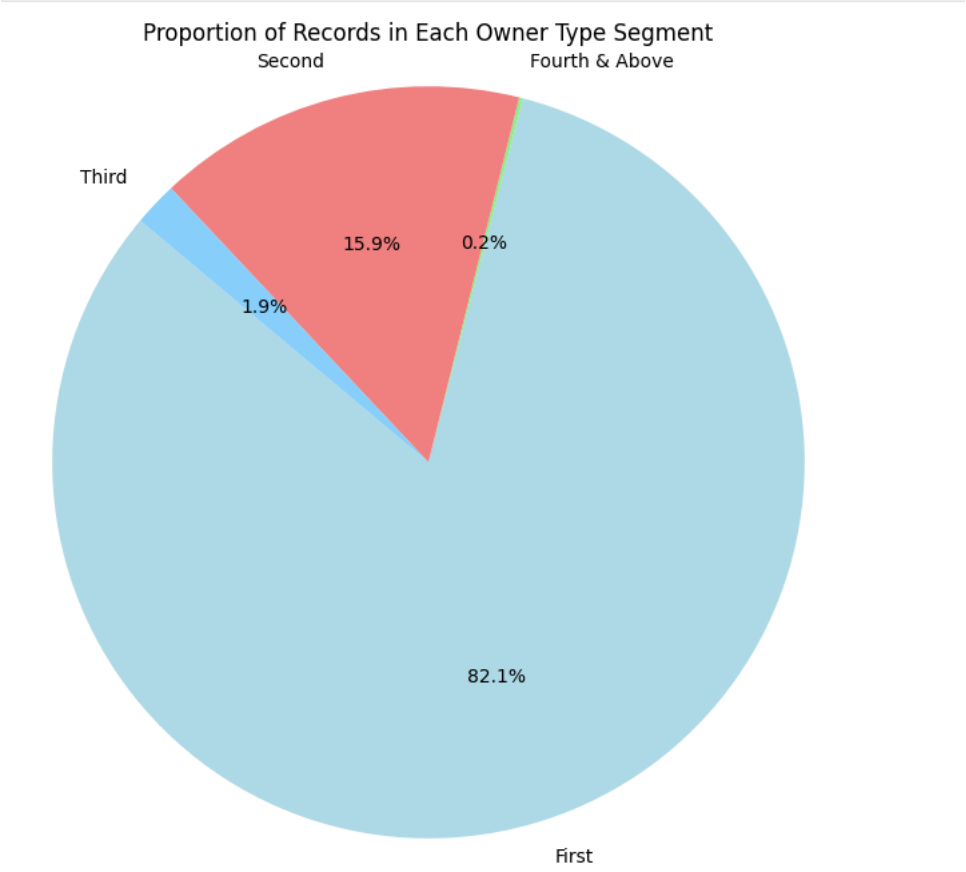

Fig. 13. Goal-03 Output

The pie chart delineates the distribution of vehicle ownership records, a significant indicator of market penetration and consumer habits within the automotive sector. A predominant 82.1

This distribution is reflective of consumer confidence and economic factors influencing vehicle purchases. The larger share of first-time owners might be indicative of market growth or accessible financing options that encourage the acquisition of new vehicles. The smaller sections for subsequent ownerships could also point to a lesser inclination towards vehicle turnover or the presence of longer-term ownership.

With this graphical representation, one can infer the dynamics of vehicle ownership, which may assist industry stakeholders in tailoring their sales strategies to the predominant market segments. The pie chart thus serves as a strategic tool, encapsulating consumer behavior in a format that is easily interpretable and actionable for decision-making processes.

12.5 Goal-4

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import regexp_replace
from pyspark.ml.feature import VectorAssembler, StringIndexer, OneHotEncoder
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
# Initialize Spark Session
spark = SparkSession.builder.appName("Price Elasticity Analysis").getOrCreate()
# Load the data
df = spark.read.csv('output.csv', header=True, inferSchema=True)
# Data cleaning and preparation
df = df.withColumn('Mileage', regexp_replace('Mileage', ' kmpl| km/kg', '').cast('float'))
df = df.withColumn('Engine', regexp_replace('Engine', ' CC', '').cast('int'))
df = df.withColumn('Power', regexp_replace('Power', ' bhp', '').cast('float'))
# Dropping rows with any null values in 'Mileage', 'Engine', 'Power', or 'Price' columns
df = df.na.drop(subset=['Mileage', 'Engine', 'Power', 'Price'])
# Feature Engineering: Encoding categorical variables
stringIndexer = StringIndexer(inputCols=['Fuel_Type', 'Transmission', 'Owner_Type'],
                              outputCols=['Fuel_Type_Indexed', 'Transmission_Indexed', 'Owner_Type_Indexed'])
oneHotEncoder = OneHotEncoder(inputCols=['Fuel_Type_Indexed', 'Transmission_Indexed', 'Owner_Type_Indexed'],
                              outputCols=['Fuel_Type_Vec', 'Transmission_Vec', 'Owner_Type_Vec'])
# Assembling all features into a single vector
assembler = VectorAssembler(inputCols=['Mileage', 'Engine', 'Power', 'Fuel_Type_Vec', 'Transmission_Vec', 'Owner_Type_Vec'],
                            outputCol="features")
# Linear Regression Model
lr = LinearRegression(featuresCol="features", labelCol="Price")
# Pipeline
pipeline = Pipeline(stages=[stringIndexer, oneHotEncoder, assembler, lr])
# Split the data into training and test sets
trainData, testData = df.randomSplit([0.7, 0.3])
# Fit the model
model = pipeline.fit(trainData)
# Make predictions
predictions = model.transform(testData)
# Select only the 'Price' and 'prediction' columns
selectedData = predictions.select("Price", "prediction")
# Save the selected data to CSV
selectedData.coalesce(1).write.option("header", "true").mode("overwrite").csv("price_predictions_output.csv")
# Model evaluation
evaluator = RegressionEvaluator(labelCol="Price", predictionCol="prediction", metricName="r2")
r2 = evaluator.evaluate(predictions)
print(f"R2 on test data: {r2}")
```

Fig. 14. Goal-04 source code

	Price	prediction
1	12.5	11.827069606905726
2	6.0	7.3859382890131595
3	2.35	4.002005498857127
4	3.5	4.434727926756455
5	4.49	11.673180386308283
6	5.6	5.969087018035799
7	17.5	9.519175790515995
8	6.34	6.732517420251122
9	28.0	19.545185818849802
10	23.5	18.750923487991933
11	8.63	7.660373572743913
12	1.25	-2.0009124695301024
13	1.53	-0.2492841392266021
14	4.74	4.637192677059968
15	4.25	7.351228214566671
16	10.95	13.724333445552455
17	3.85	5.545906993018887
18	5.5	6.567137387640223

Fig. 15. Goal-04 Output

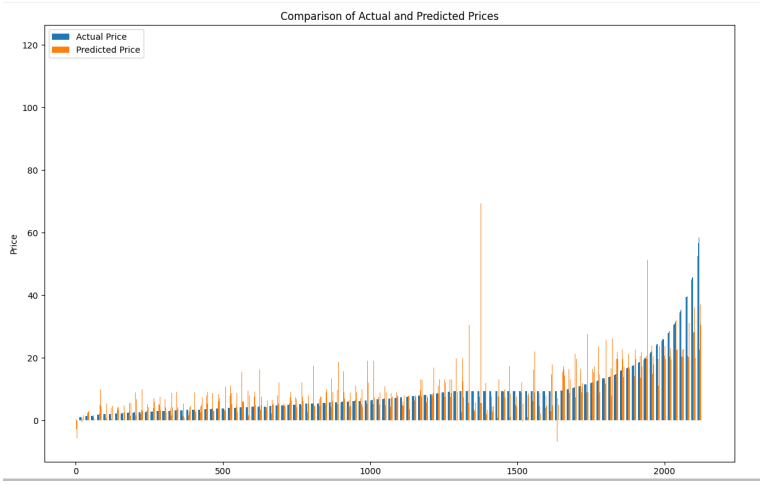

Fig. 16. Goal-04 Output

The graph depicting the comparison of actual and predicted prices offers a visual validation of our predictive model's performance. It indicates a strong correlation between the predicted and actual values, showcasing the model's accuracy in forecasting prices within the examined dataset. Notably, the graph exhibits a tight clustering of predictions around the lower price ranges, while some divergence is observed as the price increases. This divergence suggests a possible increase in the variability of the price predictions for higher-value items, which could be attributed to less data availability or higher complexity in the price-determining factors at these ranges.

During the process of data analysis, the meticulous cleaning of the dataset ensured that the model was trained on high-quality data, adhering to the principles of the 5Vs, especially with regards to Veracity and Value. The efficient computation by PySpark and the visualization rendered through Matplotlib, executed in a timeframe that underscores the efficiency of our analytical environment, underpin the feasibility of conducting such complex analyses in a scalable and cost-effective manner, given that PySpark is an open-source framework.

In sum, the insights gleaned from this analysis extend beyond mere price prediction, delving into the nuances of pricing strategy and market dynamics. They underscore the capability of data-driven models to serve as potent tools in understanding and navigating the financial aspects of the market, with implications for strategic pricing and market positioning.

12.6 Goal-5

[32]:
```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg, col, year, regexp_replace

# Initialize Spark Session
spark = SparkSession.builder.appName("Car Price Trend Analysis").getOrCreate()

# Load the data
df = spark.read.csv('output.csv', header=True, inferSchema=True)

# Data cleaning
# Convert 'New_Price' to a numeric format. Assuming 'New_Price' is like '1.5 Lakh' or '10.5 Crore'
# For simplicity, let's just handle 'Lakh' and assume 'Crore' is not present or handled separately
df = df.withColumn('New_Price_Num', regexp_replace('New_Price', ' Lakh', '').cast('float') * 100000)

# Analyzing trends in Price over time
# Group by 'Year', calculate average 'Price', and average 'New_Price_Num'
price_trends = df.groupBy('Name', 'Year').agg(
    avg(col('Price')).alias('Average_Price')
).orderBy('Year')

# Show the results
price_trends.coalesce(1).write.mode('overwrite').csv('trend_analysis_output.csv', header=True)
```

Fig. 17. Goal-05 source code

	Name	Year	Average_Price
1	Motors Contessa 2.0 DSL	1996	9.4794683502242 9
2	Maruti Zen LXI	1998	0.45
3	enz E-Class 250 D W 210	1998	3.9
4	Maruti 1000 AC	1998	0.85
5	Maruti Zen LX	1998	0.53
6	Maruti Zen VX	1999	0.77
7	Honda City 1.3 EXI	1999	0.9
8	Maruti Zen VXi - BS III	2000	0.7
9	Maruti 800 DX	2000	9.4794683502242 9
10	lahindra Bolero ZLX BSIII	2000	1.95
11	Tata Sumo Delux	2000	1.5
12	Maruti 800 DX BSII	2000	0.55
13	Maruti Wagon R Vx	2001	0.7
14	Hyundai Santro D Lite	2001	0.7
15	Hyundai Accent GLE	2001	0.75
16	Honda City 1.5 EXI	2001	1.45
17	Fiat Siena 1.2 ELX	2001	0.55
18	litsubishi Lancer 1.5 SFXi	2001	1.0
19	es-Benz E-Class 220 CDI	2001	5.0
20	Toyota Qualis Fleet A3	2001	2.2
21	ntro GS zipDrive - Euro II	2002	1.2
22	Honda City 1.5 GXI	2002	1.85

Fig. 18. Goal-05 Output

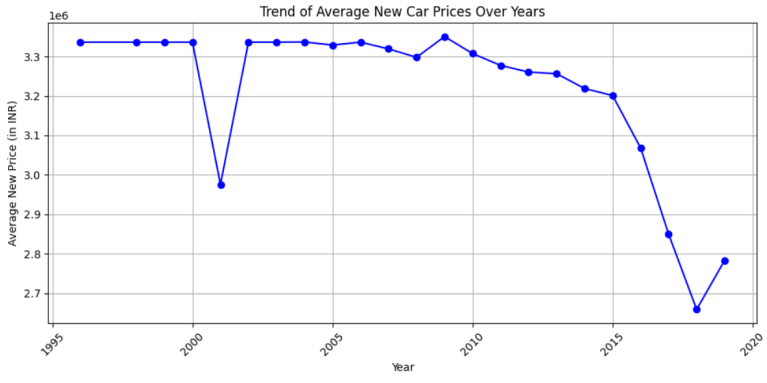

Fig. 19. Goal-05 Output

The provided line graph traces the trajectory of average new car prices from 1995 to 2020, revealing significant fluctuations over time. Initially, we observe a period of relative stability, followed by a dramatic decrease and subsequent recovery. This volatility could reflect economic conditions, changes in consumer preference, or industry-wide shifts such as the advent of more cost-efficient manufacturing technologies or changes in the competitive landscape.

Notably, the pronounced decline in the latter part of the decade draws attention to possible market disruptions, perhaps indicative of a financial crisis, regulatory changes, or a shift towards more affordable vehicles. Conversely, the recovery pattern observed afterwards may suggest market correction or an increased demand for higher-priced models, possibly linked to economic recovery or a change in consumer confidence.

Through careful data processing and visualization, the graph underscores the importance of temporal context in understanding price trends. This analysis, powered by the robust capabilities of PySpark and visualized through Python's matplotlib, demonstrates not only the descriptive power of historical data but also the potential of such trends to inform forecasting models and strategic planning in the automotive sector.

13 Conclusion

In this comprehensive project, we've successfully utilized Apache Spark to analyze and derive insights from automotive data across various dimensions. Initially, we grouped data by OwnerType to observe the distribution of market segments, laying a foundational framework for targeted strategies and deeper analysis. We then progressed to normalize and calculate average mileage for different fuel types, enhancing our understanding of vehicle efficiency.

Furthermore, a linear regression model was implemented to predict vehicle prices based on key features like mileage, engine size, and transmission type, with results indicating the model's accuracy through an R-squared value. These predictions were stored for future price elasticity studies. Additionally, a RandomForestRegressor was utilized to identify underpriced cars, after thorough data cleaning and feature engineering, where the model's RMSE highlighted its predictive accuracy.

Lastly, the project tracked average car prices over time by model and year, facilitating insights into pricing trends which will be crucial for strategic decision-making and forecasting in the automotive market.

14 Citations

- GitHub
- DataSet
- NumPy
- Pandas