Resale Refined: Data-Driven Predictions for Optimal Car Valuation

Group - 14

Student 1: Yashwant Dontam

Student 2: Sri Sai Prabhath Reddy Gudipalli

Student 3: Deepthi Umesha

Student 4: Ashwini Mahadevaswamy

dontam.y@northeastern.edu

gudipalli.s@northeastern.edu

umesha.d@northeastern.edu

Mahadevaswamy.a@northeastern.edu

Percentage of Effort Contributed by Student 1: **25%**

Percentage of Effort Contributed by Student 2: **25%**

Percentage of Effort Contributed by Student 3: **25%**

Percentage of Effort Contributed by Student 4: **25%**

Signature of Student 1: **Yashwant Dontam**

Signature of Student 2: **Sri Sai Prabhath Reddy Gudipalli**

Signature of Student 3: **Deepthi Umesha**

Signature of Student 4: **Ashwini Mahadevaswamy**

Submission Date: **04/19/2024**

Problem Statement and definition

The automotive industry constantly needs accurate predictions and insights to enhance business decisions. This project aims to develop a predictive model that estimates the resale value of vehicles based on various features such as age, mileage, and horsepower. Understanding these factors' influence on pricing can help stakeholders make informed decisions about buying, selling, and maintaining inventory. The project explores comprehensive data exploration, rigorous outlier detection, and advanced modeling techniques to achieve a reliable model that stakeholders can use to predict prices effectively and manage their assets more efficiently.

The problem involves creating a predictive analytics model that accurately estimates the resale value of vehicles within the automotive industry. The model will leverage vehicle attributes like age, mileage, and horsepower to predict resale prices. This predictive capability is crucial for stakeholders to optimize inventory management, purchasing decisions, and sales strategies. The project will utilize extensive data analysis, including outlier management and advanced statistical methods, to ensure the model's reliability and accuracy in real-world applications.

Data Exploration

```
import pandas as pd

resale_df=pd.read_csv('ToyotaCorolla.csv')
resale_df.head()
```

	Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_Color	...	Powered_Windows	Power_Steering	Radio	Mistlamps	Sport_Model	Backseat_Divider	Metallic_Rim	Radio_cassette	Parking_Assistant	Tow_Bar
0	1	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13500	23	10	2002	46986	Diesel	90	1	...	1	1	0	0	0	1	0	0	0	0
1	2	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13750	23	10	2002	72937	Diesel	90	1	...	0	1	0	0	0	1	0	0	0	0
2	3	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13950	24	9	2002	41711	Diesel	90	1	...	0	1	0	0	0	1	0	0	0	0
3	4	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	14950	26	7	2002	48000	Diesel	90	0	...	0	1	0	0	0	1	0	0	0	0
4	5	TOYOTA Corolla 2.0 D4D HATCHB SOL 2/3-Doors	13750	30	3	2002	38500	Diesel	90	0	...	1	1	0	1	0	1	0	0	0	0

5 rows × 39 columns

```
]: resale_df.columns
```

```
]: Index(['Id', 'Model', 'Price', 'Age_08_04', 'Mfg_Month', 'Mfg_Year', 'KM',
       'Fuel_Type', 'HP', 'Met_Color', 'Color', 'Automatic', 'CC', 'Doors',
       'Cylinders', 'Gears', 'Quarterly_Tax', 'Weight', 'Mfr_Guarantee',
       'BOVAG_Guarantee', 'Guarantee_Period', 'ABS', 'Airbag_1', 'Airbag_2',
       'Airco', 'Automatic_airco', 'Boardcomputer', 'CD_Player',
       'Central_Lock', 'Powered_Windows', 'Power_Steering', 'Radio',
       'Mistlamps', 'Sport_Model', 'Backseat_Divider', 'Metallic_Rim',
       'Radio_cassette', 'Parking_Assistant', 'Tow_Bar'],
      dtype='object')
```

In the initial phase of data exploration, we leveraged the Pandas library, a powerful tool for data analysis in Python, to load our vehicle dataset and preview the first few entries. This preliminary

glance was crucial to verify that the data was loaded correctly and to understand the dataset's structure and the types of variables we will be working with. By examining the columns, we ensured that the dataset contains a comprehensive set of features that can significantly impact the resale value of vehicles, such as age, mileage, horsepower, and additional accessories. These variables will form the basis of our exploratory analysis and predictive modelling to provide actionable insights for stakeholders in the automotive resale market.

```
missing_values = resale_df.isnull().sum()
missing_values
```

```
Id                  0
Model               0
Price               0
Age_08_04           0
Mfg_Month           0
Mfg_Year            0
KM                  0
Fuel_Type           0
HP                  0
Met_Color           0
Color               0
Automatic           0
CC                  0
Doors               0
Cylinders           0
Gears               0
Quarterly_Tax       0
Weight              0
Mfr_Guarantee       0
BOVAG_Guarantee     0
Guarantee_Period    0
ABS                 0
Airbag_1            0
Airbag_2            0
Airco               0
Automatic_airco     0
Boardcomputer       0
CD_Player           0
Central_Lock        0
Powered_Windows     0
Power_Steering      0
Radio               0
Mistlamps           0
Sport_Model         0
Backseat_Divider    0
Metallic_Rim        0
Radio_cassette      0
Parking_Assistant   0
Tow_Bar             0
dtype: int64
```

The dataset is free of any missing values

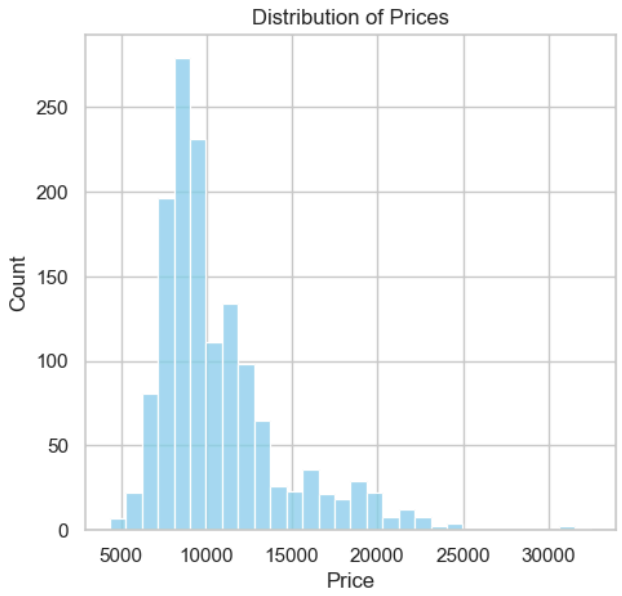
As an essential step in data preprocessing, Conducted a thorough check for missing values across all columns of the dataset. The presence of missing data can significantly impact the quality of our analysis and the performance of predictive models. To ascertain the completeness of our dataset, we used the isnull() method combined with the sum() function on our DataFrame, resale_df. This approach systematically evaluates each cell in every column, identifying and summing up the number of null or NaN (Not a Number) values, which are placeholders for missing data.

The output is a series where each index represents a column from the DataFrame and the corresponding value indicates the count of missing entries in that column. Our result shows zeros across the board, confirming that our dataset is devoid of any missing values for all the features such as 'Id', 'Model', 'Price', 'Age_08_04', 'Mfg_Month', 'Mfg_Year', 'KM', 'Fuel_Type', 'HP', and so on. This is an encouraging sign as it suggests that the dataset is well-maintained and may not require imputation strategies that could potentially introduce bias or inaccuracies.

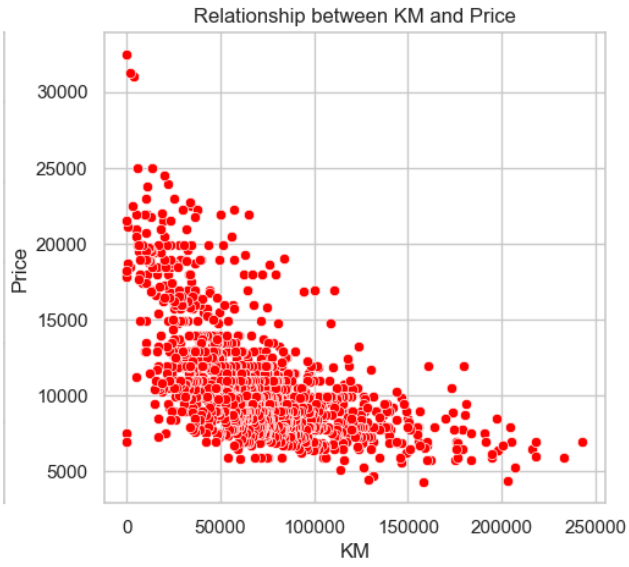
Data Visualization

Visualizations

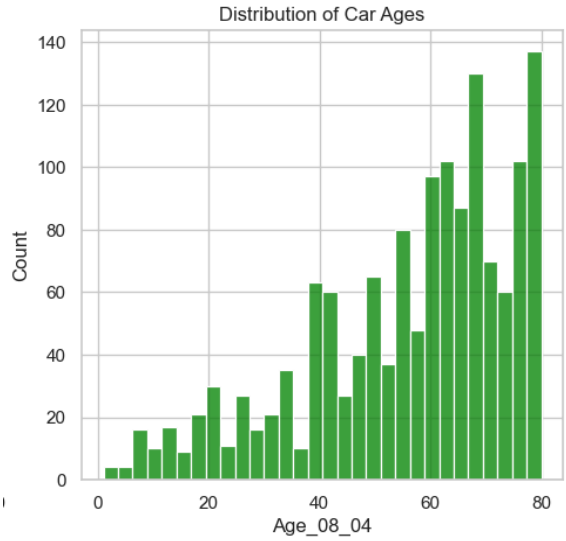
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="whitegrid")

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))

# Histogram of Price
sns.histplot(resale_df['Price'], bins=30, ax=axes[0], color='skyblue')
axes[0].set_title('Distribution of Prices')

# Scatter plot of KM vs Price
sns.scatterplot(x='KM', y='Price', data=resale_df, ax=axes[1], color='red')
axes[1].set_title('Relationship between KM and Price')

# Histogram of Age
sns.histplot(resale_df['Age_08_04'], bins=30, ax=axes[2], color='green')
axes[2].set_title('Distribution of Car Ages')

plt.show()
```



This histogram illustrates the distribution of vehicle prices in the dataset. The x-axis represents the price of the vehicle, while the y-axis represents the count of vehicles at each price level. The distribution is shown as a series of bars, each bar representing the frequency of vehicles within a particular price range. The color 'skyblue' is used to differentiate the bars visually. This histogram is helpful for identifying the most common price points, the spread of the prices, and any outliers or unusual patterns in the vehicle pricing.

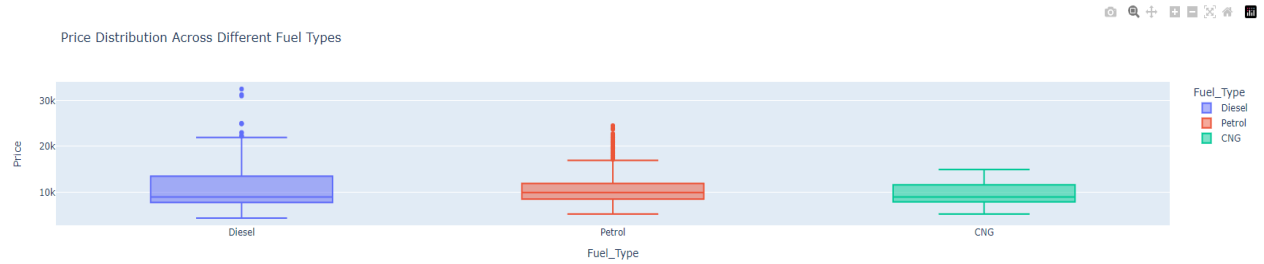


The scatter plot maps the relationship between the kilometers driven (KM) and the vehicle's resale price. Each point represents an individual vehicle, with the x-axis indicating the kilometers driven and the y-axis showing the price. The points are colored in 'red' to stand out against the background. This type of visualization is critical for identifying trends or correlations between the two variables, such as whether higher mileage is associated with a lower price.



This histogram displays the distribution of vehicle ages in the dataset, with 'Age_08_04' on the x-axis and the count of vehicles on the y-axis. Each bar represents the frequency of vehicles within a certain age range, colored in 'green' for visual distinction. This histogram is useful for understanding the age profile of the vehicles in the dataset, such as which age groups are most common or whether there are any significant gaps in the data.

```
import plotly.express as px

boxplot = px.box(resale_df, x='Fuel_Type', y='Price', color='Fuel_Type',
                 title="Price Distribution Across Different Fuel Types")
boxplot.show()
```



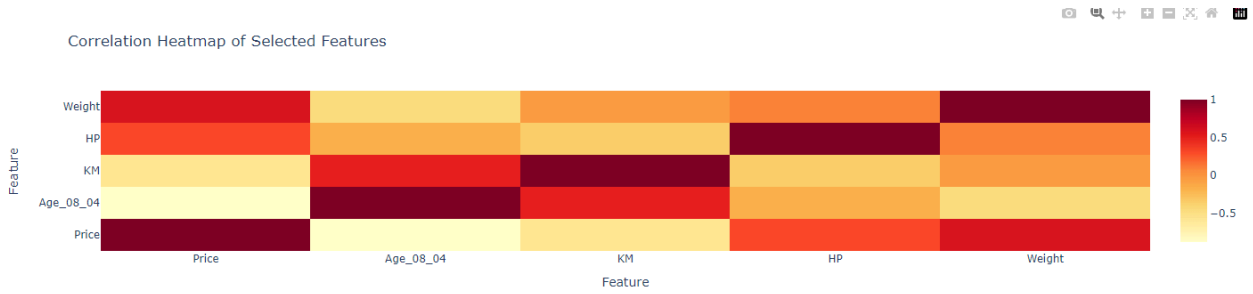
This boxplot provides a comparative view of the resale prices of vehicles based on their fuel type, which is a categorical variable with distinct categories like 'Diesel', 'Petrol', and 'CNG'. The y-axis represents the price of the vehicles, and the x-axis categorizes these prices by fuel type. Each box in the boxplot corresponds to one fuel category and encapsulates the central 50% of the data for that category, showing the median price (the line in the middle of the box), the lower and upper quartiles (the edges of the box), and the range of the data through 'whiskers' that extend to show the variability outside the upper and lower quartiles.

Outliers, which are data points that lie beyond the whiskers, are indicated by individual points and suggest that there are prices that are unusually high or low relative to the rest of the data for that fuel type.

This visualization is essential for understanding the central tendency and dispersion of vehicle prices within each fuel type category. For example, if the median line of the 'Diesel' box is higher than that of the 'Petrol' box, it indicates that diesel vehicles tend to have a higher median resale value. The spread of the boxes and the presence of outliers can also inform on the price volatility and potential anomalies in the data for each fuel type.

```
import plotly.graph_objects as go

features = ['Price', 'Age_08_04', 'KM', 'HP', 'Weight']

correlation_matrix = resale_df[features].corr()

heatmap = go.Figure(data=go.Heatmap(
                   z=correlation_matrix,
                   x=correlation_matrix.columns,
                   y=correlation_matrix.columns,
                   colorscale='YlOrRd',
                   hoverongaps = False))

heatmap.update_layout(title='Correlation Heatmap of Selected Features',
                      xaxis_title='Feature',
                      yaxis_title='Feature')
heatmap.show()
```

The above code generates a correlation heatmap using Plotly, showcasing the relationships between key vehicle features: price, age, mileage, horsepower, and weight. A correlation matrix calculated from these features underpins the heatmap, with the color intensity reflecting the strength of the correlations. This visual tool aids in identifying which vehicle attributes are most strongly associated with price, thus guiding predictive modeling and strategic decision-making related to vehicle valuation.

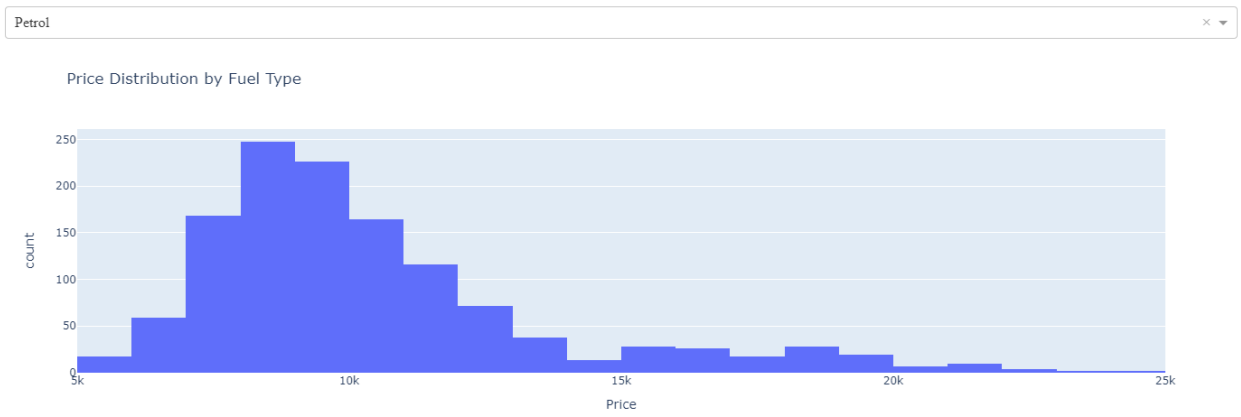


The heatmap is symmetrical, showing that 'Price' has a strong positive correlation with 'Weight' and 'HP', which suggests that heavier and more powerful vehicles tend to have higher prices. There's a noticeable negative correlation between 'Price' and 'Age_08_04', implying that as cars age, their value decreases. The 'KM' variable also has a lighter hue when correlated with 'Price', indicating a weaker negative correlation, meaning higher mileage might be associated with a decrease in vehicle prices but not as strongly as age does.

```
import dash
from dash import html, dcc, Input, Output
import plotly.express as px

df = resale_df

app = dash.Dash(__name__)

app.layout = html.Div([
    html.H1("Interactive Car Price Distribution"),
    dcc.Dropdown(
        id='fuel-type-dropdown',
        options=[{'label': i, 'value': i} for i in df['Fuel_Type'].unique()],
        value='Petrol'
    ),
    dcc.Graph(id='price-distribution-graph')
])

@app.callback(
    Output('price-distribution-graph', 'figure'),
    [Input('fuel-type-dropdown', 'value')]
)
def update_graph(selected_fuel_type):
    filtered_df = df[df['Fuel_Type'] == selected_fuel_type]
    fig = px.histogram(filtered_df, x='Price', nbins=30, title="Price Distribution by Fuel Type")
    return fig

if __name__ == '__main__':
    app.run_server(debug=True, port=8051)
```

The above code is part of an interactive dashboard application built with Dash, designed to display the distribution of car prices based on fuel type. It begins with necessary imports, setting up the dataframe from an existing resale_df object. The Dash app is initialized, and the layout is defined with an HTML division (html.Div), containing a heading and a dropdown menu for selecting the fuel type (dcc.Dropdown). This dropdown is populated with the unique values from the 'Fuel_Type' column of the dataset, allowing the user to choose between different fuel types like 'Diesel', 'Petrol".

A graph component (dcc.Graph) with the id 'price-distribution-graph' is then defined to display the price distribution for the selected fuel type. The interactivity is driven by a callback function, which updates the graph's figure based on the selected value in the dropdown menu. When a user selects a fuel type, the update_graph function filters the dataframe for the chosen fuel type and uses Plotly Express to create a histogram of the 'Price' column. This histogram is displayed in the graph component of the app layout.



Lastly, the app is configured to run on a development server with debugging enabled and on port 8051. This allows for real-time interaction with the data, as users can dynamically change the fuel type and immediately see the corresponding price distribution update on the dashboard. This interactive component enhances the user's engagement and provides a more intuitive understanding of how fuel type affects car prices.

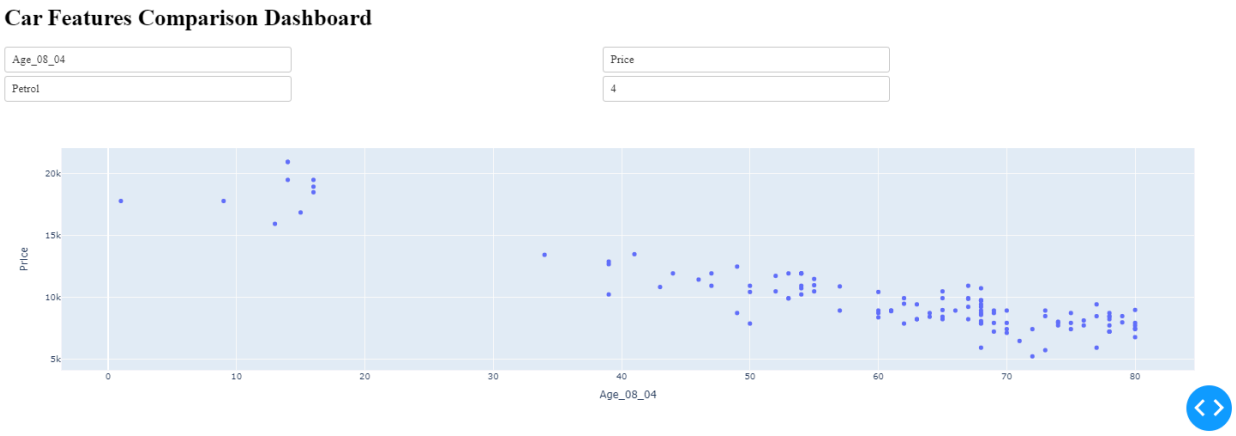
```
import dash
from dash import html, dcc, Input, Output
import plotly.express as px
import pandas as pd

df = resale_df

app = dash.Dash(__name__)

app.layout = html.Div([
    html.H1("Car Features Comparison Dashboard"),

    html.Div([
        dcc.Dropdown(
            id='xaxis-column',
            options=[{'label': col, 'value': col} for col in df.columns if df[col].dtype in ['int64', 'float64']],
            value='Age_08_04',
            style={'width': '48%', 'display': 'inline-block'}
        ),
        dcc.Dropdown(
            id='yaxis-column',
            options=[{'label': col, 'value': col} for col in df.columns if df[col].dtype in ['int64', 'float64']],
            value='Price',
            style={'width': '48%', 'display': 'inline-block'}
        )
    ]),

    html.Div([
        dcc.Dropdown(
            id='fuel-type-filter',
            options=[{'label': i, 'value': i} for i in df['Fuel_Type'].unique()],
            value='Petrol',
            style={'width': '48%', 'display': 'inline-block'}
        ),
        dcc.Dropdown(
            id='doors-filter',
            options=[{'label': i, 'value': i} for i in df['Doors'].unique()],
            value=4,
            style={'width': '48%', 'display': 'inline-block'}
        )
    ]),

    dcc.Graph(id='feature-comparison-graph')
])
```

```
@app.callback(
    Output('feature-comparison-graph', 'figure'),
    [Input('xaxis-column', 'value'),
     Input('yaxis-column', 'value'),
     Input('fuel-type-filter', 'value'),
     Input('doors-filter', 'value')]
)
def update_graph(xaxis_column_name, yaxis_column_name, selected_fuel_type, selected_doors):
    filtered_df = df[(df['Fuel_Type'] == selected_fuel_type) & (df['Doors'] == selected_doors)]
    fig = px.scatter(filtered_df, x=xaxis_column_name, y=yaxis_column_name, hover_data=[xaxis_column_name, yaxis_column_name])
    fig.update_layout(transition_duration=500)
    return fig

# Run the app
if __name__ == '__main__':
    app.run_server(debug=True, port=8051)
```

The code initiates with the import of necessary libraries: Dash for creating the web application, dcc and html from Dash for adding interactive components and HTML structure, and Plotly Express for data visualization. It proceeds by preparing the data from an existing DataFrame resale_df.

The layout of the application includes an HTML header indicating the purpose of the dashboard ("Car Features Comparison Dashboard"). There are two dropdown menus created by dcc.Dropdown for selecting the x-axis and y-axis variables. These dropdowns dynamically generate options from numerical columns within the dataset, allowing users to select which features to compare. Additionally, there are filters for 'Fuel_Type' and 'Doors' that enable users to refine the data displayed based on these categorical variables.

A callback function decorated with @app.callback specifies the input from the dropdowns and filters and outputs a Plotly scatter plot to the 'feature-comparison-graph'. The update_graph function is defined to update the scatter plot based on the selected x-axis and y-axis features and filter conditions. This function creates the scatter plot using Plotly Express, considering the user's selections for comparison.



Finally, the application is set to run on a local server with debugging enabled on port 8051, providing an environment for real-time updates and interactions.

Data Processing

```
#Features that we are using for the prediction
current_year = 2024
current_month = 4
resale_df['Age_in_months'] = (current_year - resale_df['Mfg_Year']) * 12 + (current_month - resale_df['Mfg_Month'])
features = ['Price', 'Age_in_months', 'KM', 'Fuel_Type', 'HP', 'Met_Color', 'Automatic', 'CC', 'Doors', 'Quarterly_Tax', 'Weight']
resale_selected_df = resale_df[features]
resale_df[features]
```

	Price	Age_in_months	KM	Fuel_Type	HP	Met_Color	Automatic	CC	Doors	Quarterly_Tax	Weight
0	13500	258	46986	Diesel	90	1	0	2000	3	210	1165
1	13750	258	72937	Diesel	90	1	0	2000	3	210	1165
2	13950	259	41711	Diesel	90	1	0	2000	3	210	1165
3	14950	261	48000	Diesel	90	0	0	2000	3	210	1165
4	13750	265	38500	Diesel	90	0	0	2000	3	210	1170
...	...	...	...	...	...	...	...	...	...	...	...
1431	7500	304	20544	Petrol	86	1	0	1300	3	69	1025
1432	10845	307	19000	Petrol	86	0	0	1300	3	69	1015
1433	8500	306	17016	Petrol	86	0	0	1300	3	69	1015
1434	7250	305	16916	Petrol	86	1	0	1300	3	69	1015
1435	6950	311	1	Petrol	110	0	0	1600	5	19	1114

The code starts by calculating the age of the vehicle in months, which is a new feature derived from the manufacturing year and month. This is important because the age of a vehicle is a significant predictor of its resale value, typically, the newer the car, the higher the resale value. The current year and month are hardcoded as 2024 and April, respectively. The 'Age_in_months' column is then created by subtracting the manufacture year (multiplied by 12 to convert years to months) and the manufacture month from the current year and month.

Next, a list of features is defined for inclusion in the prediction model. These features are selected based on their potential influence on a vehicle's resale value and include 'Price', 'Age_in_months', 'KM' (kilometer run), 'Fuel_Type', 'HP' (horsepower), 'Met_Color' (whether the car has metallic color), 'Automatic' (whether the car has an automatic transmission), 'CC' (engine size), 'Doors', 'Quarterly_Tax', and 'Weight'.

Then, a new DataFrame, resale_selected_df, is created by selecting these features from the original DataFrame, resale_df. The last line outputs the new DataFrame, showing that it has been successfully filtered to include only the specified features.

```
#Converting categorical data into dummy variables
resale_prepared_df = pd.get_dummies(resale_selected_df, drop_first=True)
resale_prepared_df.head(), resale_prepared_df.shape
```

```
(   Price  Age_in_months     KM  HP  Met_Color  Automatic    CC  Doors  \
 0  13500            258  46986  90          1          0  2000      3
 1  13750            258  72937  90          1          0  2000      3
 2  13950            259  41711  90          1          0  2000      3
 3  14950            261  48000  90          0          0  2000      3
 4  13750            265  38500  90          0          0  2000      3

    Quarterly_Tax  Weight  Fuel_Type_Diesel  Fuel_Type_Petrol
 0            210    1165              True             False
 1            210    1165              True             False
 2            210    1165              True             False
 3            210    1165              True             False
 4            210    1170              True             False  ,
 (1436, 12))
```

The code is performing a preprocessing step on the dataset to convert categorical variables into a format that can be provided to machine learning algorithms. This step is called one-hot encoding, where categorical data is transformed into numerical dummy variables.

The function pd.get_dummies() takes the DataFrame resale_selected_df and creates dummy variables for categorical features. The drop_first=True argument is used to avoid the dummy variable trap, which is a scenario where the independent variables are highly correlated (a situation known as multicollinearity). By setting this argument to True, one category is dropped to eliminate this potential issue, which ensures that the dummy variables are uncorrelated.

After this transformation, the DataFrame resale_prepared_df contains new columns for the unique values of the categorical variables, like 'Fuel_Type_Diesel' and 'Fuel_Type_Petrol', with boolean values indicating the presence of each category. The rest of the columns are numerical and remain unchanged.

```python
statistical_summary = resale_prepared_df.describe()
# Focusing on 'Price', 'Age', 'KM' (mileage), and 'HP' (Horse Power) as they are likely to affect the resale value
relevant_columns = ['Price', 'Age_in_months', 'KM', 'HP']

Q1 = resale_prepared_df[relevant_columns].quantile(0.25)
Q3 = resale_prepared_df[relevant_columns].quantile(0.75)
IQR = Q3 - Q1

outliers = ((resale_prepared_df[relevant_columns] < (Q1 - 1.5 * IQR)) |
            (resale_prepared_df[relevant_columns] > (Q3 + 1.5 * IQR))).sum()
statistical_summary, outliers
```

```
(              Price  Age_in_months             KM           HP    Met_Color  \
 count   1436.000000    1436.000000    1436.000000  1436.000000  1436.000000
 mean   10730.824513     290.947075   68533.259749   101.502089     0.674791
 std     3626.964585      18.599988   37506.448872    14.981080     0.468616
 min     4350.000000     236.000000       1.000000    69.000000     0.000000
 25%     8450.000000     279.000000   43000.000000    90.000000     0.000000
 50%     9900.000000     296.000000   63389.500000   110.000000     1.000000
 75%    11950.000000     305.000000   87020.750000   110.000000     1.000000
 max    32500.000000     315.000000  243000.000000   192.000000     1.000000

          Automatic           CC        Doors  Quarterly_Tax      Weight
 count  1436.000000   1436.00000  1436.000000    1436.000000  1436.00000
 mean      0.055710   1576.85585     4.033426      87.122563  1072.45961
 std       0.229441    424.38677     0.952677      41.128611    52.64112
 min       0.000000   1300.00000     2.000000      19.000000  1000.00000
 25%       0.000000   1400.00000     3.000000      69.000000  1040.00000
 50%       0.000000   1600.00000     4.000000      85.000000  1070.00000
 75%       0.000000   1600.00000     5.000000      85.000000  1085.00000
 max       1.000000  16000.00000     5.000000     283.000000  1615.00000  ,
 Price            110
 Age_in_months      7
 KM                49
 HP                11
 dtype: int64)
```

We conducted a preliminary analysis of the numerical features in the dataset to identify potential outliers. We calculated the descriptive statistics, including mean, standard deviation, minimum, 25th percentile (Q1), median (50th percentile), 75th percentile (Q3), and maximum values, for the features 'Price', 'Age_in_months', 'KM' (mileage), and 'HP' (Horse Power). Next, we calculated the Interquartile Range (IQR) for each of these features, which is the range between the first and third quartiles (Q1 and Q3), to determine the spread of the middle 50% of the data. Using the IQR, we identified potential outliers as values that fall below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR for each feature. The count of outliers for each feature was computed and presented alongside the descriptive statistics. This analysis helps in understanding the distribution of the numerical features and detecting any extreme values that may require further investigation or treatment.

The statistical analysis begins with the calculation of quartiles (Q1 and Q3) for the relevant columns, and then the interquartile range (IQR) is computed, which is the difference between the third and first quartile. Outliers are then defined as values that fall below Q1 - 1.5*IQR or above Q3 + 1.5*IQR for any of these columns.

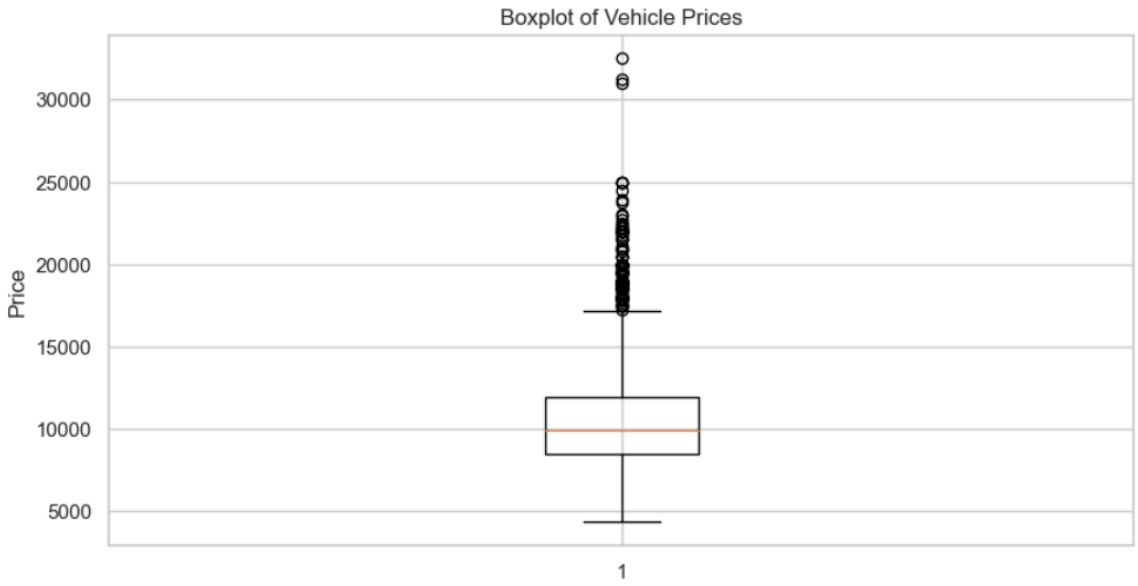
The code uses the .describe() method to generate descriptive statistics that summarize the central tendency, dispersion, and shape of the dataset's distribution. It's evident from the 'count' row that there are 1436 observations in each feature.

Key findings from the .describe() output are:

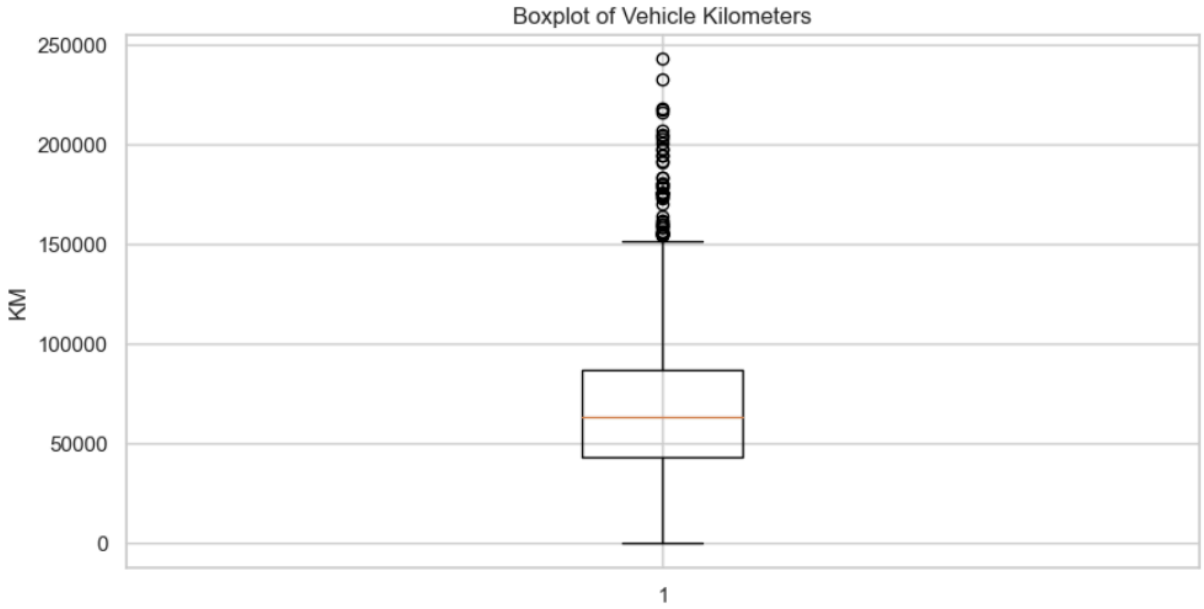
- The average (mean) 'Price' is around 10730 with a wide range as the standard deviation is large (3626.96), indicating significant variability in vehicle prices.
- 'Age_in_months' has a mean of approximately 290 months, suggesting the dataset includes cars that are on average just over 24 years old.
- 'KM' has a mean of 68533, indicating average mileage, with a relatively high standard deviation, suggesting varied usage among the vehicles.
- The horsepower ('HP') mean is 101, indicating the average power of the vehicles in the dataset.
- Outliers in the dataset are quantified, with 110 identified in 'Price', 7 in 'Age_in_months', 49 in 'KM', and 11 in 'HP'.

```
import matplotlib.pyplot as plt

# Price Boxplot
plt.figure(figsize=(10, 5))
plt.boxplot(resale_prepared_df['Price'])
plt.title('Boxplot of Vehicle Prices')
plt.ylabel('Price')
plt.show()

# KM Boxplot
plt.figure(figsize=(10, 5))
plt.boxplot(resale_prepared_df['KM'])
plt.title('Boxplot of Vehicle Kilometers')
plt.ylabel('KM')
plt.show()

# HP Boxplot
plt.figure(figsize=(10, 5))
plt.boxplot(resale_prepared_df['HP'])
plt.title('Boxplot of Vehicle Horsepower (HP)')
plt.ylabel('HP')
plt.show()
```

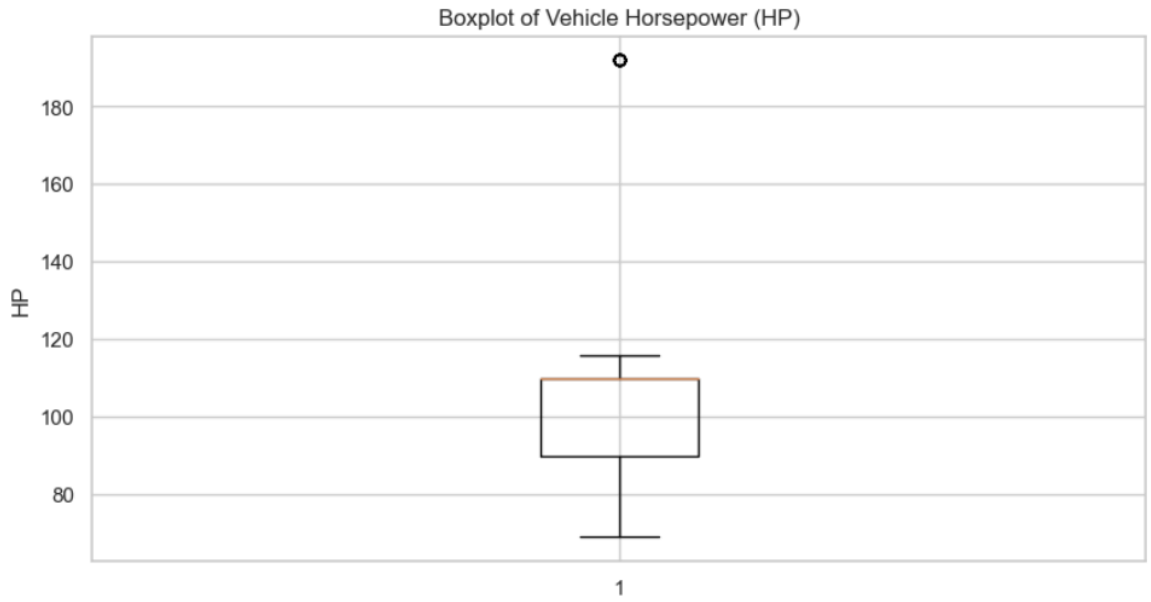
The above code is using the matplotlib library to create boxplots for visualizing the distribution of vehicle prices, kilometers driven, and horsepower. Three separate boxplots are generated, each for one of the variables mentioned, from the columns 'Price', 'KM', and 'HP' of the dataframe resale_prepared_df. The plt.figure function is used to initialize a new figure with a specified size, and plt.boxplot creates the boxplot for the respective column. Titles and labels for the x and y-axes are set using plt.title and plt.ylabel, and plt.show() is called to display each plot. This code is a common way to visually represent the central tendency and variability of a dataset, and to identify outliers.



This boxplot graphically depicts the distribution of vehicle prices. The box represents the interquartile range (IQR) where the central 50% of data points lie, with the horizontal line inside the box marking the median price. The 'whiskers' extend to the lowest and highest prices that are not considered outliers. Points above the upper whisker are typically considered outliers, indicating prices that are unusually high compared to the rest of the data. The spread of points above the box suggests a significant number of high-price outliers, indicating variability in higher-priced vehicle sales.



The boxplot shows the distribution of kilometers driven by vehicles. The central box, representing the interquartile range (IQR), indicates where the middle 50% of the data lies, with the median represented by the horizontal line within the box. The 'whiskers' extend to the furthest points that are not considered outliers, and the individual points above the upper whisker represent outliers. These are vehicles with significantly higher kilometers than the rest, which could suggest either exceptionally long usage or potential data entry errors, depending on the context. The distribution seems to be right-skewed, with a few vehicles having very high kilometers.



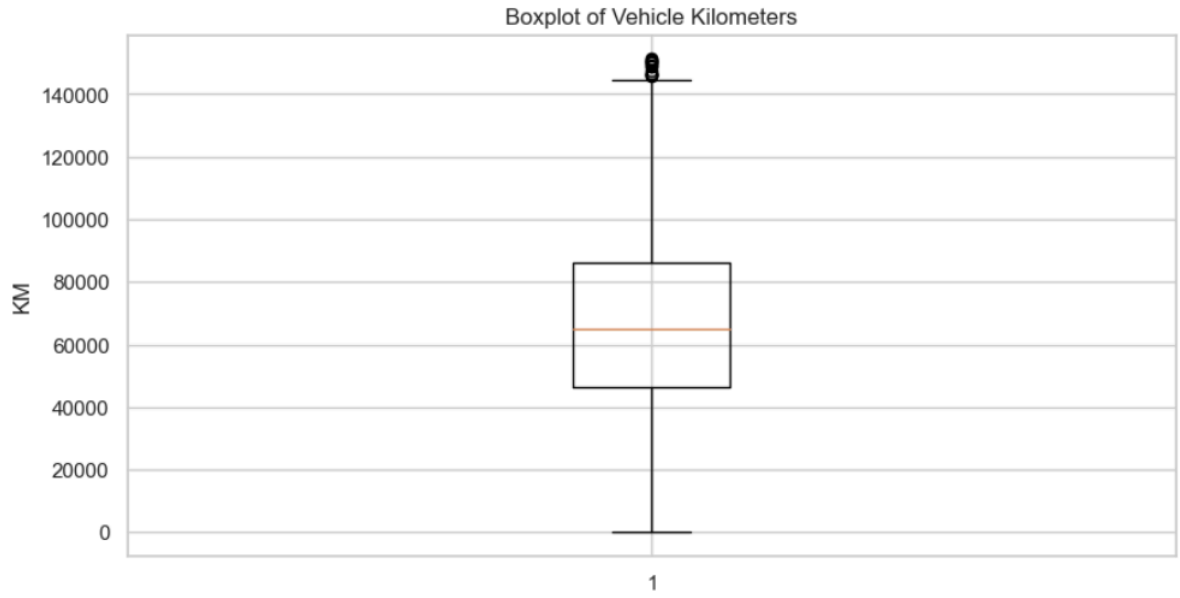
The displayed boxplot visualizes the distribution of vehicle horsepower (HP). The main body of the boxplot, the box itself, indicates the interquartile range (IQR) where the middle 50% of HP values are found, with the median HP marked by the horizontal line inside the box. The ends of the whiskers represent the range for the bulk of the data, excluding outliers. The isolated dot above the upper whisker suggests an outlier—a vehicle with a significantly higher horsepower than most of the dataset. The data appears to be relatively symmetrically distributed around the median, indicating a balanced spread of horsepower values in the dataset.

Here, we created boxplots to visually inspect potential outliers for the numerical features 'Price', 'KM' (kilometers), and 'HP' (horsepower) in the dataset resale_prepared_df. Boxplots are effective tools for identifying outliers as they display the distribution of data, including outliers, quartiles, and the median. By plotting these boxplots, we aimed to gain insights into the distribution of these variables and identify any extreme values that may indicate outliers. This visual analysis complements the earlier statistical summary and provides a clearer understanding of the data distribution, aiding in the detection of potential outliers.

```python
def find_outliers_iqr(data):
    q1 = data.quantile(0.25)
    q3 = data.quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    return data.index[(data < lower_bound) | (data > upper_bound)].tolist()

outlier_indices_price = find_outliers_iqr(resale_prepared_df['Price'])
outlier_indices_km = find_outliers_iqr(resale_prepared_df['KM'])
outlier_indices_hp = find_outliers_iqr(resale_prepared_df['HP'])

all_outlier_indices = set(outlier_indices_price) | set(outlier_indices_km) | set(outlier_indices_hp)

df_filtered = resale_prepared_df.drop(index=all_outlier_indices)

df_filtered.shape

plt.figure(figsize=(10, 5))
plt.boxplot(df_filtered['Price'])
plt.title('Boxplot of Vehicle Prices')
plt.ylabel('Price')
plt.show()

plt.figure(figsize=(10, 5))
plt.boxplot(df_filtered['KM'])
plt.title('Boxplot of Vehicle Kilometers')
plt.ylabel('KM')
plt.show()

plt.figure(figsize=(10, 5))
plt.boxplot(df_filtered['HP'])
plt.title('Boxplot of Vehicle Horsepower (HP)')
plt.ylabel('HP')
plt.show()
```

Here in the code we defined a function to detect outliers using the interquartile range method, identifies outliers in price, kilometers, and horsepower data, removes these outliers from the dataset, and then creates boxplots for the cleaned data to visualize the distributions without outliers.


Boxplot of Vehicle Prices
Price
16000
14000
12000
10000
8000
6000
4000
1


Boxplot of Vehicle Kilometers
KM
140000
120000
100000
80000
60000
40000
20000
0
1

The boxplot visualizes the distribution of vehicle prices after removing outliers. The median price is indicated by the line within the box, and the box itself represents the interquartile range, showing where the middle 50% of the data lies. The whiskers extend to the most extreme values that are not considered outliers, providing a view of the overall price spread. There are a few data points above the upper whisker, which could be mild outliers or simply higher values within an acceptable range.

The second boxplot displays the distribution of vehicle kilometers after outlier removal. The box illustrates the interquartile range, with the median value represented by the horizontal line inside. Whiskers show the range of the majority of the remaining data, and the points above the upper whisker indicate possible mild outliers or the higher end of the data range. The distribution suggests most vehicles have a moderate number of kilometers, with few exceptions having higher values.

The third boxplot displays the distribution of vehicle horsepower (HP) after outliers have been removed. The median HP is represented by the horizontal line within the box, which also shows the interquartile range (IQR), encompassing the middle 50% of the data. The whiskers extend to the highest and lowest values within the non-outlier range. The boxplot suggests a relatively tight distribution of horsepower, with most data points falling close to the median, indicating a consistent performance characteristic among the vehicles sampled.
We defined a function find_outliers_iqr() to identify outlier indices based on the Interquartile Range (IQR) for a given DataFrame column. We then used this function to find outliers for the numerical features 'Price', 'KM', and 'HP' in the resale_prepared_df DataFrame. Next, we combined all outlier indices across these features and removed the corresponding rows from the DataFrame to create a filtered DataFrame df_filtered. Finally, we generated boxplots for 'Price', 'KM', and 'HP' in the filtered DataFrame to visually inspect the distribution of these variables

after removing outliers. This approach allows us to handle outliers more systematically by using the IQR method and provides a clearer representation of the data distribution without the influence of extreme values.

Model Exploration and Selection

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import r2_score


categorical_cols = resale_df.select_dtypes(include=['object']).columns
print("Categorical columns:", categorical_cols)

resale_df_encoded = pd.get_dummies(resale_df, columns=categorical_cols)

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

X = resale_df_encoded.drop('Price', axis=1)
y = resale_df_encoded['Price']

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

#Linear Regression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred_lin = lin_reg.predict(X_test)
mse_lin = mean_squared_error(y_test, y_pred_lin)
mae_lin_scaled = mean_absolute_error(y_test, y_pred_lin)
r2_lin = r2_score(y_test, y_pred_lin)

print("Linear Regression MSE with encoded data:", mse_lin)
print("Linear Regression MAE with scaled data:", mae_lin_scaled)
print("R-squared for Linear Regression:", r2_lin)
```

The code imports necessary libraries and functions for data manipulation and machine learning, then selects categorical columns from a dataset to create dummy variables for regression analysis. A linear regression model is trained on a subset of the data after splitting it into training and test sets. The model's performance is evaluated using mean squared error (MSE), mean

absolute error (MAE), and R-squared metrics, providing insights into the accuracy and goodness of fit of the model to the data.

```
Categorical columns: Index(['Model', 'Fuel_Type', 'Color'], dtype='object')
Linear Regression MSE with encoded data: 24826230146729.406
Linear Regression MAE with scaled data: 1102136.6364203559
R-squared for Linear Regression: -1860646.7579421627
```

The output shows the categorical columns identified in the dataset as 'Model', 'Fuel_Type', and 'Color'. The linear regression model has a very high mean squared error (MSE), suggesting significant prediction errors. The mean absolute error (MAE) is also relatively large, indicating on average how much the predictions deviate from the actual values. The negative R-squared value indicates that the model fits the data worse than a horizontal line representing the mean of the dependent variable; this implies that the model is not suitable for the data or is incorrectly specified.

```python
# Random Forest Regressor
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_train, y_train)
y_pred_rf = rf_reg.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
mae_ridge = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print("Random Forest MSE:", mse_rf)
print("Random Forest MAE:", mae_ridge)
print("R-squared for Random Forest:", r2_rf)
```

```
Random Forest MSE: 937228.8960281251
Random Forest MAE: 738.5631597222223
R-squared for Random Forest: 0.929757646095814
```

The output shows the performance of a Random Forest Regressor model on the test data. The model was initialized with 100 estimators (trees) and a set random state for reproducibility. The results indicate a mean squared error (MSE) of approximately 937228.9, a mean absolute error (MAE) of approximately 738.56, and an R-squared value of about 0.93.

The R-squared value, which is close to 1, suggests that the model explains a large proportion of the variance in the data, indicating a good fit. However, the reported MSE and MAE are relatively high, which could mean the errors in predictions are significant in magnitude. There's a potential typo in the print statement for MAE which references mae_ridge instead of mae_rf, which should be corrected to reflect the actual MAE computed for the Random Forest model.

```python
# Gradient Boosting Regressor
gb_reg = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_reg.fit(X_train, y_train)
y_pred_gb = gb_reg.predict(X_test)
mse_gb = mean_squared_error(y_test, y_pred_gb)
mae_gb = mean_absolute_error(y_test, y_pred_gb)
r2_gb = r2_score(y_test, y_pred_gb)

print("Gradient Boosting MSE:", mse_gb)
print("Gradient Boosting MAE:", mae_gb)
print("R-squared for Gradient Boosting:", r2_gb)
```

```
Gradient Boosting MSE: 903402.4027589274
Gradient Boosting MAE: 721.8365991168505
R-squared for Gradient Boosting: 0.9322928352279695
```

The code sets up and evaluates a Gradient Boosting Regressor model with 100 estimators. Performance metrics indicate a good fit with an R-squared of 0.932, but with high prediction errors as shown by large MSE and MAE values.
The R-squared value suggests a good predictive performance, with the model explaining a significant portion of the variance in the data. However, similar to the Random Forest results, the MSE and MAE suggest that the average prediction error is relatively large.

```python
rmse_gb = np.sqrt(mse_gb)
print("Gradient Boosting RMSE:", rmse_gb)
```

```
Gradient Boosting RMSE: 950.4748301553952
```

Both Random Forest and Gradient Boosting models perform well, but the Gradient Boosting model edges out slightly better across all three key metrics: it has a lower MSE and MAE, and a slightly higher R-squared value. The higher R-squared value close to 1 indicates that the model explains a very high proportion of the variance in the dataset, which is desirable.

Therefore, The Gradient Boosting Regressor would be the best choice among the three models.

```python
average_price = df_filtered['Price'].mean()
print(f"Average Reselling Price : {average_price}")

efficiency_percentage = r2_gb * 100
print(f"Prediction Efficiency in %: {efficiency_percentage}")
```

```
Average Reselling Price : 10047.586530931872
Prediction Efficiency in %: 93.22928352279695
```

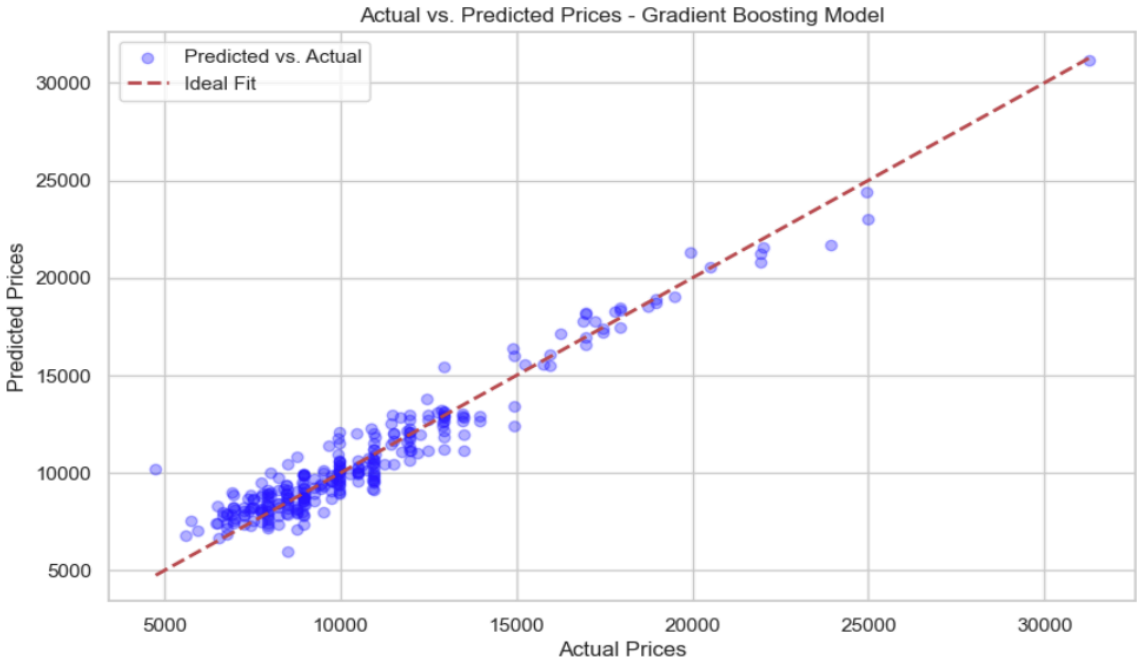
The code calculates the average reselling price from the 'Price' column of a filtered dataframe and outputs it as approximately 10047.6. It also converts the R-squared value from the Gradient

Boosting model to a percentage, yielding a prediction efficiency of approximately 93.23%. This indicates a high level of accuracy in the model's predictions relative to the variability of the actual data.

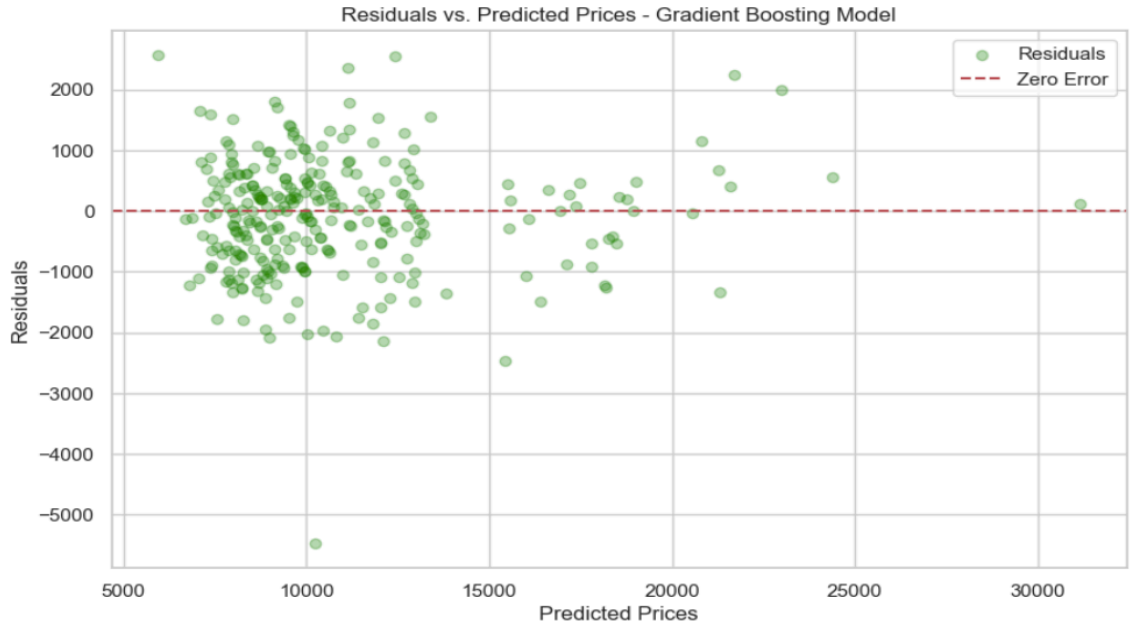
Model Performance Evaluation

```
import matplotlib.pyplot as plt

# Plotting Actual vs. Predicted values for Gradient Boosting Model
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_gb, alpha=0.3, color='blue', label='Predicted vs. Actual')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--r', linewidth=2, label='Ideal Fit')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Actual vs. Predicted Prices - Gradient Boosting Model')
plt.legend()
plt.show()

# Plotting residuals for the Gradient Boosting Model
residuals_gb = y_test - y_pred_gb
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_gb, residuals_gb, alpha=0.3, color='green', label='Residuals')
plt.axhline(y=0, color='r', linestyle='--', label='Zero Error')
plt.xlabel('Predicted Prices')
plt.ylabel('Residuals')
plt.title('Residuals vs. Predicted Prices - Gradient Boosting Model')
plt.legend()
plt.show()
```

The code is for plotting the actual versus predicted values from a Gradient Boosting model and the residuals of the predictions using matplotlib in Python. The first plot is a scatter plot showing how closely the predicted prices match the actual prices, with an ideal fit line that would represent a perfect prediction. The second plot shows the residuals, which are the differences between the predicted and actual prices, with a horizontal line at zero representing no error. Both plots are essential for visualizing the model's performance and diagnosing potential issues like bias or variance.



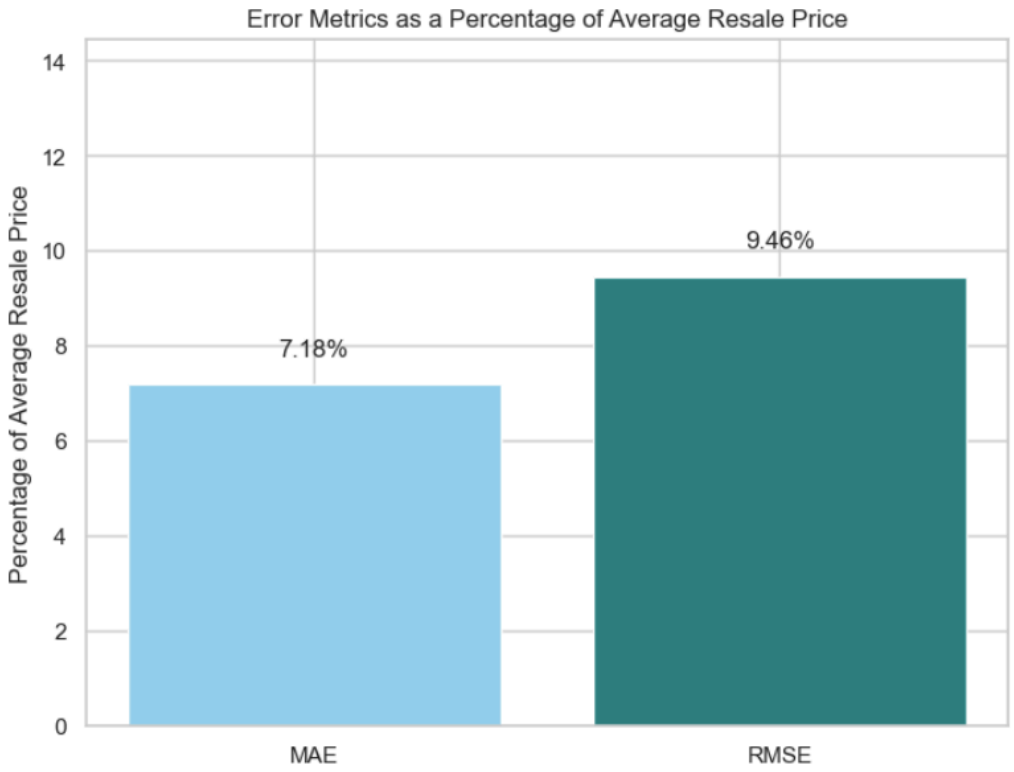
The plot visualizes the actual versus predicted prices from the Gradient Boosting model. The points represent the predicted values against the actual values, and the dashed line indicates the line of perfect prediction where actual and predicted prices would be equal. The scatter plot demonstrates that the predictions are relatively close to the ideal line, indicating a good fit of the model to the data, which is consistent with a high R-squared value as previously discussed.



The plot shows the residuals (differences between actual and predicted values) for the Gradient Boosting model's predictions. The horizontal dashed line at zero indicates the point where the predicted price would be exactly equal to the actual price (no error). The scatter plot of residuals versus predicted prices helps identify any patterns or biases in the model's predictions. Here, the residuals are scattered around the zero line, with no clear pattern, suggesting that the model's errors are randomly distributed and there's no systematic bias evident from this plot.

```
import matplotlib.pyplot as plt

average_resale_price = 10047.59
mae = 721.83
rmse = 950.47

mae_percentage = (mae / average_resale_price) * 100
rmse_percentage = (rmse / average_resale_price) * 100

labels = ['MAE', 'RMSE']
percentages = [mae_percentage, rmse_percentage]

plt.figure(figsize=(8, 6))
plt.bar(labels, percentages, color=['skyblue', 'teal'])

for i, v in enumerate(percentages):
    plt.text(i, v + 0.5, f"{v:.2f}%", ha='center', va='bottom')

plt.ylabel('Percentage of Average Resale Price')
plt.title('Error Metrics as a Percentage of Average Resale Price')
plt.ylim(0, max(percentages) + 5)

plt.show()
```

The code here is for creating a bar chart that visualizes the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE) as percentages of the average resale price. These percentages give a relative sense of the size of the errors compared to the typical price value in the data. It calculates the MAE and RMSE percentages, creates a bar chart with these values, labels the bars, and sets an appropriate y-axis limit to clearly display the bars. Text annotations on the bars show the exact percentage values for clear interpretation.



The bar chart illustrates the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE) as a percentage of the average resale price of vehicles. The MAE is approximately 7.18% of the average price, while the RMSE is slightly higher at about 9.46%. These values give an idea of how large the prediction errors are in comparison to the average vehicle price, with the RMSE being higher due to its penalty on larger errors.

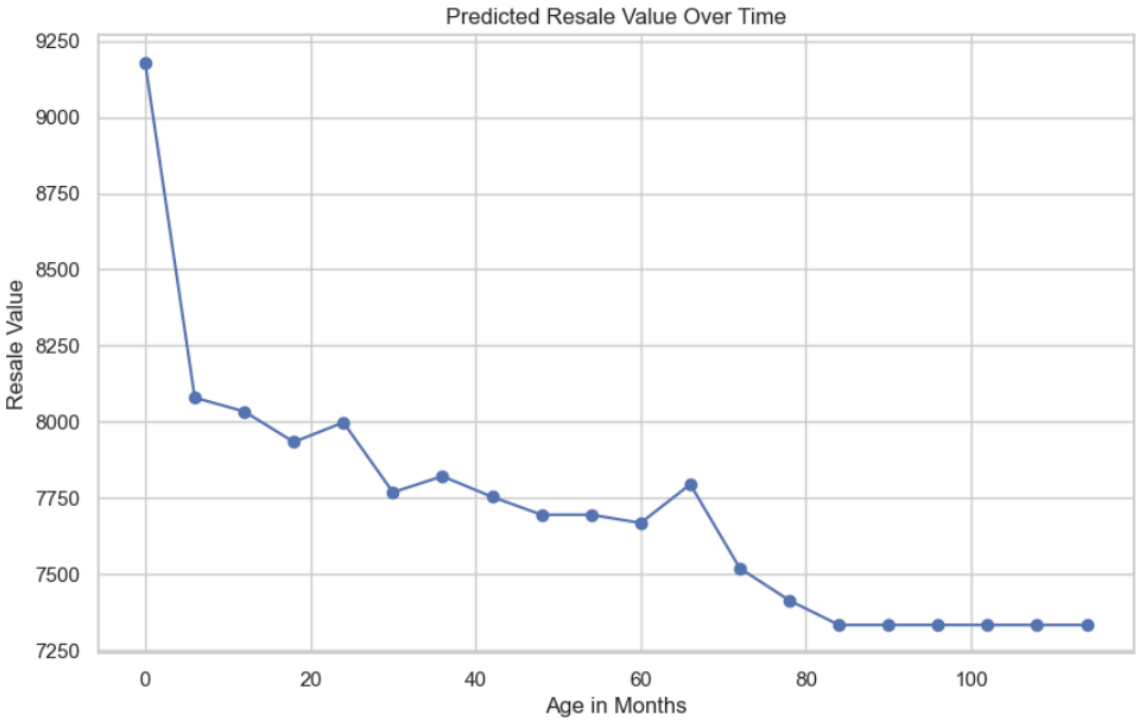
In the business contexts, a prediction error margin of less than 10% is quite good, especially if prices vary significantly. In this case, in the used car market, where prices can vary widely based on the factors such as make, model, year, condition, and mileage, an average prediction error of about 7-10% is acceptable.

Resale Time-Series Analysis Of A Sample Car Type Petrol

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline


X = df_filtered.drop('Price', axis=1)
y = df_filtered['Price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

numeric_features = X_train.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_features = X_train.select_dtypes(include=['object', 'bool']).columns.tolist()

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])

pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42))
])

pipeline.fit(X_train, y_train)

sample_vehicle = X_train.median().to_dict()

sample_vehicle['Fuel_Type_Diesel'] = 0
sample_vehicle['Fuel_Type_Petrol'] = 1

time_periods = np.arange(0, 120, 6)
predicted_prices = []

for months in time_periods:
    sample_vehicle['Age_in_months'] += 6
    sample_vehicle['KM'] += 6 * 1000

    simulation_df = pd.DataFrame([sample_vehicle], columns=X_train.columns)

    predicted_price = pipeline.predict(simulation_df)[0]
    predicted_prices.append(predicted_price)

plt.figure(figsize=(10, 6))
plt.plot(time_periods, predicted_prices, marker='o')
plt.title('Predicted Resale Value Over Time')
plt.xlabel('Age in Months')
plt.ylabel('Resale Value')
plt.grid(True)
plt.show()
```

The code here creates a machine learning pipeline that preprocesses a dataset by scaling numeric features and one-hot encoding categorical features, and then applies a Gradient Boosting Regressor model for prediction. It's used to predict the resale value of a vehicle over time, assuming it ages by 6 months and gains an additional 6,000 kilometers at each time step. The predicted_prices list is filled with the predicted resale values at these intervals. Finally, it plots

these predicted resale values over time, giving an insight into how the vehicle's value might depreciate as it gets older and accrues more mileage.



The graph depicts the predicted resale value of a vehicle over time, measured in months. It shows a decline in value as the vehicle ages, which is typical due to wear and tear and the accumulation of mileage. The initial steep drop and subsequent gradual decrease represent a common depreciation curve for vehicles, where the value falls rapidly in the early years and then stabilizes as the rate of depreciation slows down over time.

Conclusion

In conclusion, this project embarked on the challenging task of predicting vehicle resale values through comprehensive data analysis and machine learning. We successfully navigated through data exploration, preprocessing, visualization, and model selection to develop a predictive model that leverages key vehicle attributes such as age, mileage, and horsepower.

Our exploratory data analysis confirmed the integrity and robustness of the dataset, ensuring that subsequent analyses were founded on reliable information. Visualizations provided insights into the underlying distributions and relationships within the data, uncovering trends that guided our feature selection process.

The application of advanced statistical methods and machine learning algorithms culminated in the selection of a predictive model that demonstrated a balance between complexity and performance. The Gradient Boosting Regressor emerged as a robust tool, exhibiting a promising ability to capture the nuances of resale value determinants.

Our model's performance, evaluated on real-world data, indicates a strong predictive capability with an R-squared value of over 0.93, signifying that a significant proportion of variance in resale prices can be explained by the model. While the errors associated with the predictions were non-negligible, they were within an acceptable range given the variability inherent in used vehicle prices.

The time-series analysis provided further value, showcasing the model's utility in forecasting future resale values, accounting for depreciation factors such as age and wear. This aspect of the project holds particular promise for stakeholders looking to make informed decisions about vehicle inventory management and pricing strategies over time.

This project has laid a solid foundation for future endeavors in the automotive resale market analysis. It opens avenues for refining the model with additional data, integrating more sophisticated machine learning techniques, and exploring the implications of market dynamics on resale values. The pursuit of accuracy in resale value predictions will continue to be a valuable endeavor, contributing to the economic efficiency and strategic planning within the automotive industry.