Overview of the Notebook's Goal

The primary goal of this notebook is to perform **exploratory data analysis (EDA)** on a dataset of used cars from CarDekho. The process involves:

1. **Acquiring the data**: Downloading it from Kaggle.
2. **Cleaning the data**: Handling missing values and removing unnecessary columns.
3. **Analyzing the data**: Examining individual columns (univariate analysis) and relationships between columns (bivariate analysis).
4. **Visualizing the data**: Creating plots to better understand the data's characteristics and draw insights.

Libraries and Modules Used

The script utilizes several powerful Python libraries to accomplish its tasks.

Library	Purpose
qrcode	Used at the beginning to generate a QR code from a given URL. This is separate from the main data analysis task.
os	A standard Python library for interacting with the operating system. Here, it's used to list files in a directory.
kagglehub	A specific library for downloading datasets directly from the Kaggle platform.
pandas	The core library for data manipulation and analysis in Python. It introduces a powerful data structure called a **DataFrame**.
seaborn	A data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative

	statistical graphics.
matplotlib.pyplot	The foundational plotting library in Python. It's used for creating a wide variety of static, animated, and interactive visualizations.
numpy	A fundamental package for scientific computing with Python. It's used here for creating numerical ranges for histogram bins.

Step-by-Step Code Explanation

1. Initial Setup and Data Loading

The notebook begins with some setup and then loads the dataset.

- **Generating a QR Code**:
  ```python
  import qrcode
  img =
  qrcode.make('https://colab.research.google.com/drive/17qdEJHd2jXsfFI1yzdiboK9IFgFxZyfv?usp=sharing')
  img.save('myqr.png')
  ```
  This section is a standalone piece of code. The qrcode.make() function creates a QR code image object from the provided link, and img.save() saves it as a PNG file.
- **Downloading the Dataset from Kaggle**:
  ```python
  import kagglehub
  path = kagglehub.dataset_download("manishkr1754/cardekho-used-car-data")
  ```
  This uses the kagglehub library to download the specified dataset. The dataset_download function returns the local path where the dataset files are stored.
- **Finding and Loading the CSV File**:

```python
import os
import pandas as pd

all_files = os.listdir(path)
file_path = path + '/' + all_files[0]
df = pd.read_csv(file_path)
```

- os.listdir(path) lists all files in the downloaded dataset's directory.
- The code then constructs the full path to the first file.
- pd.read_csv(file_path) is a crucial pandas function that reads a comma-separated values (CSV) file into a **DataFrame** named df. A DataFrame is a 2-dimensional labeled data structure with columns of potentially different types, similar to a spreadsheet or a SQL table.

2. Initial Data Exploration

Once the data is loaded into the df DataFrame, the next step is to understand its basic properties.

- **Viewing the Data**:
  - df.head(): Shows the first 5 rows of the DataFrame.
  - df.tail(): Shows the last 5 rows.
  - df.sample(3): Shows a random sample of 3 rows.
- **Getting Information about the DataFrame**:
  - df.info(): Provides a concise summary of the DataFrame, including the index dtype and columns, non-null values, and memory usage. This is great for quickly seeing if there are missing values.
  - r, c = df.shape: The .shape attribute returns a tuple representing the dimensionality of the DataFrame (rows, columns).
  - df.columns: Returns a list of all column names.
  - df.index: Returns the index (row labels) of the DataFrame.

3. Data Cleaning

Before analysis, the data is cleaned.

- **Dropping an Unnecessary Column**:

```python
df.drop('Unnamed: 0', axis=1, inplace=True)
```

- The drop() function is used to remove rows or columns.
- 'Unnamed: 0' is the column to be removed.
- axis=1 specifies that we are dropping a column (axis=0 would be for a row).
- inplace=True modifies the DataFrame directly, without needing to assign it back to a new variable (e.g., df = df.drop(...)).

- **Checking for Missing Values**:

```python
df.isna().sum()
# or
df.isnull().sum()
```

  - df.isna() (or df.isnull()) returns a DataFrame of the same shape, but with True for missing (NaN) values and False for non-missing values.
  - .sum() is then called on this boolean DataFrame. In this context, True is treated as 1 and False as 0, so the sum gives the total count of missing values in each column.

- **Visualizing Missing Values**:

```python
import seaborn as sns
sns.heatmap(df.isnull())
```

  - sns.heatmap() creates a graphical representation of data where values are depicted by color. When used on df.isnull(), it creates a chart that visually shows the pattern of missing data. A solid color block indicates no missing values.

4. Descriptive Statistics

This step involves summarizing the data to extract key insights.

- **Numerical Summary**:

```python
df.describe().round(2)
```

  - df.describe() generates descriptive statistics for the **numerical columns** by default. This includes count, mean, standard deviation, min, max, and quartile values.
  - .round(2) rounds the results to two decimal places.

- **Categorical Summary**:

```python
```

```
df.describe(include=['O'])
```

- By specifying include=['O'] (for 'Object' datatype), describe() provides a summary for the **categorical columns**. This includes the count, the number of unique categories, the most frequent category (top), and its frequency (freq).

5. Univariate Analysis (Analyzing Single Columns)

Here, we dive deeper into individual columns.

- **Separating Column Types**:
  ```python
  cat_col = list(df.describe(include=['O']).columns)
  num_col = list(df.describe().columns)
  ```
  This code cleverly uses the output of describe() to get lists of categorical and numerical column names.
- **Analyzing Categorical Columns**:
  ```python
  df['car_name'].value_counts().head(10)
  ```
  - df['car_name'] selects a single column (a pandas **Series**).
  - .value_counts() returns a Series containing counts of unique values, sorted in descending order. This is perfect for finding the most common items.
  - The code iterates through each categorical column (for i in cat_col:) and displays the top 10 most frequent values.
- **Visualizing Categorical Data**:
  ```python
  def graph_plot(col_name):
     # ... (code to create a bar plot) ...
     plt.bar(x, y)
     # ...

  for i in cat_col:
   graph_plot(i)
  ```
  - A function graph_plot is defined to avoid repeating plotting code.
  - It takes a column name, gets the top 10 value counts, and creates a **bar chart** using matplotlib.pyplot.bar() to visualize the frequencies.
  - plt.xticks(rotation=45) rotates the x-axis labels to prevent them from overlapping.

- **Visualizing Numerical Data**:
  Python
  ```
  def plot_hist(col_name, bin_size=100):
   # ... (code to create a histogram) ...
   plt.hist(df[col_name], bins=...)
   # ...

  for i in num_col:
   plot_hist(i)
  ```
  - Similarly, a function plot_hist is created to plot **histograms** for numerical columns using plt.hist().
  - A histogram groups numbers into ranges (bins) and shows how many values fall into each range. It's excellent for understanding the distribution of a variable (e.g., is it skewed?).

6. Filtering and Querying Data (Masking)

This section demonstrates how to select specific rows from the DataFrame based on conditions. This is known as **masking**.

- **The Concept of Masking**:
  - A condition like df['mileage'] == df['mileage'].max() produces a boolean Series (True for rows that meet the condition, False otherwise).
  - When this series is used to index the DataFrame df[...], it returns only the rows where the condition is True.
- **Examples from the Notebook**:
  - df[df['mileage'] == df['mileage'].max()]: Finds the car(s) with the highest mileage.
  - df[df['selling_price'] == df['selling_price'].min()]: Finds the car(s) with the lowest selling price.
- **Sorting to Find Top/Bottom Values**:
  Python
  ```
  df.sort_values(by='selling_price', ascending=False).head(10)
  ```
  - df.sort_values() sorts the DataFrame by one or more columns.
  - by='selling_price' specifies the column to sort by.
  - ascending=False sorts in descending order (highest to lowest).
  - .head(10) then selects the top 10 rows from the sorted DataFrame.
- **Finding the Nth Highest/Lowest Value**:
  Python

```python
second_max_price = df['selling_price'].sort_values(ascending=False).values[1]
```

- .values converts the pandas Series to a NumPy array.
- [1] selects the element at index 1 (the second element), which corresponds to the second-highest price.

7. Bivariate Analysis and Grouping

This is the analysis of two or more variables together to find relationships.

- **The groupby() Function**: This is one of the most powerful features of pandas. It follows a "split-apply-combine" strategy:
  1. **Split**: The data is split into groups based on some criteria (e.g., car brand).
  2. **Apply**: A function is applied to each group independently (e.g., calculate the mean of the selling_price).
  3. **Combine**: The results are combined into a new data structure.
- **Examples from the Notebook**:

  Python
  ```python
  df.groupby('brand')['selling_price'].mean().round(2).sort_values(ascending=0)
  ```

  This line of code calculates the average selling_price for each brand.

  Python
  ```python
  df.groupby('seller_type')['selling_price'].agg(['min', 'max', 'mean']).round()
  ```

  - .agg() (aggregate) allows you to apply multiple functions at once.
  - This calculates the minimum, maximum, and mean selling_price for each seller_type.