Data Analysis using SQL - Detailed Report

Introduction

This report presents a comprehensive analysis of the "Used Cars Data" using SQL-like operations in Python with pandas. The dataset contains information about various used cars, including their specifications and prices. The goal is to gain insights into the data, perform data preprocessing, and conduct exploratory data analysis (EDA) to understand the relationships between different variables.

1. Data Loading and Overview

The analysis begins by loading the dataset from the "used_cars_data.csv" file and obtaining an overview of the data.

Python code

```
# Importing Libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Loading Data
data = pd.read_csv("used_cars_data.csv")

# Overview of Data
data.info()
data.describe()
```

Output:

- Data Information (e.g., data types, non-null counts)
- Descriptive Statistics (e.g., mean, min, max) for numerical columns

2. Data Preprocessing

Data preprocessing is a crucial step to ensure the data is ready for analysis. In this section, we perform the following preprocessing steps:

2.1. Handling Duplicates

Duplicate rows are removed from the dataset to avoid any repetition and maintain data integrity.

Python code

data.drop_duplicates(inplace=True)

2.2. Handling Missing Values

Missing values in various columns are addressed using different techniques based on the context and significance of the missing data.

2.2.1. Drop the Row

If the number of missing values in a row is insignificant, dropping the row is a viable option.

Python code

data.dropna(axis=0, how='any', inplace=True)

2.2.2. Replace with Statistical Properties

For numerical columns with relatively few missing values, replacing missing values with statistical properties such as mean, median, or mode is a common approach.

Python code

data['Power_bhp'].fillna(data['Power_bhp'].mean(), inplace=True)

2.2.3. Imputation

Imputing missing values using more advanced methods like K-Nearest Neighbors (KNN) imputation can be applied for columns with significant missing data.

Python code

from sklearn.impute import KNNImputer

data_power_engine = data[['Power_bhp', 'engine_cc']]

```
imputer = KNNImputer(n_neighbors=3)
data_power_engine_imputed = imputer.fit_transform(data_power_engine)
data_power_engine_imputed = pd.DataFrame(data_power_engine_imputed,
columns=data_power_engine.columns)
data['Power_bhp'] = data_power_engine_imputed['Power_bhp']
```

2.2.4. Create a New Category Label

For categorical columns with missing values, creating a new category label like "Unknown" can be an appropriate strategy.

Python code

```
data['Seats'].fillna('Unknown', inplace=True)
```

2.3. Creating New Features

Additional features are created based on existing data to provide more context and improve analysis. For example, we create a new feature called 'car_Age' by subtracting the 'Year' column from the current year (2023) to represent the age of the car.

Python code

```
data['car_Age'] = 2023 - data['Year']
```

2.4. Extracting Information from Text

Useful information can be extracted from text data. In this analysis, the 'car_brand' column is derived from the 'Name' column to represent the brand of each car.

Python code

```
data['car_brand'] = data['Name'].apply(lambda x: x.split()[0])
data['car_brand'].replace({'ISUZU': 'Isuzu', 'Land': 'LandRover'}, inplace=True)
```

2.5. One-Hot Encoding

Categorical columns are one-hot encoded to convert them into a numeric representation.

Python code

```
data_encoded = pd.get_dummies(data.select_dtypes(exclude=np.number).drop(columns='model_name'))
```

3. Exploratory Data Analysis (EDA)

EDA is a critical step to understand the data's underlying patterns and relationships. In this section, we perform various EDA tasks:

3.1. Univariate Analysis

Analyzing individual variables in the dataset.

3.1.1. Histograms for Numerical Columns

Python code

```
for col in data.select_dtypes(include=np.number).columns:
    sns.histplot(data[col])
    plt.show()
```

3.1.2. Box Plots for Numerical Columns

Python code

```
for col in data.select_dtypes(include=np.number).columns:
    sns.boxplot(data[col])
    plt.show()
```

3.1.3. Count Plots for Categorical Columns

Python code

```
for col in data.select_dtypes(exclude=np.number).columns:
    plt.figure(figsize=(10, 6))
    sns.countplot(data[col], order=data[col].value_counts().index)
    plt.show()
```

3.2. Bivariate Analysis

Exploring relationships between pairs of variables.

3.2.1. Scatter Plot for 'Price' vs. 'Transmission'

Python code

```
sns.scatterplot(data['Price'], data['Transmission'])
```

3.2.2. Scatter Plot for 'Kilometers_Driven' vs. 'Price'

Python code

```
sns.scatterplot(data['Kilometers_Driven'], data['Price'])
```

3.2.3. Bar Plot for 'Price' vs. 'car_brand'

Python code

```
plt.figure(figsize=(10, 6))
sns.barplot(data['car_brand'], data['Price'])
```

3.3. Multivariate Analysis

Analyzing interactions between multiple variables.

3.3.1. Correlation Heatmap for Numerical Columns

Python code

```
sns.heatmap(data.select_dtypes(include=np.number).corr(), annot=True)
```

Conclusion

This detailed analysis of the "Used Cars Data" provides valuable insights into the dataset. Data preprocessing steps, including handling duplicates, addressing missing values using various techniques, creating new features, and extracting information from text, ensure the data is ready for analysis. Exploratory data analysis (EDA) helps understand the relationships between different variables and discover patterns in the data. The use of SQL-like operations in Python with pandas enables efficient data analysis, making it a powerful tool for data analysts and data scientists. The examples and results presented in this report serve as a comprehensive guide for conducting data analysis using SQL in Python. The included Seaborn graphs visualize the data, making it easier to interpret and draw meaningful conclusions from the analysis.