

Machine Learning Lab

Week2:

Task: DataPre-processing and visualization

Required Python Libraries: pandas, matplotlib, seaborn

Problems:

Import Dataset [car_sales.csv]

Download File::

https://github.com/chandanverma07/DataSets/blob/master/Car_sales.csv

Installation of Seaborn Library

For Python environment :

pip install seaborn

For conda environment :

conda install seaborn

Need Data Pre-Processing?

Raw data often contains:

- Missing values (e.g., empty cells)
- Duplicate entries (same data appearing multiple times)
- Incorrect data types (e.g., text instead of numbers)
- Inconsistent formats (e.g., different date formats)
- Outliers (extremely high or low values affecting results)

By pre-processing the data, we ensure:

- ✓ Data is clean and structured
- ✓ Models perform better
- ✓ Accurate visualizations and insights

Pandas is used for data cleaning and transformation

Matplotlib creates basic plots for visualization

Seaborn helps in advanced and aesthetic statistical visualizations

Data Pre-Processing ensures data quality before visualization

Visualization techniques help understand relationships and detect patterns

Seaborn

- Built on top of Matplotlib
- **Advanced statistical visualizations** (heatmaps, box plots, violin plots)
- **Easier to use** with built-in themes and color schemes

Code:

```
import pandas as pd  
df=pd.read_csv("car_sales.csv")  
print(df.head())
```

- `pd.read_csv("Car_sales.csv")`: Reads the dataset into a Pandas DataFrame.
- `df.head()`: Displays the top five rows of the dataset to understand its structure.

```
print(df.head())
```

- Explicitly prints the DataFrame's first five rows as text.
- Used when running scripts in environments like command-line interfaces (CLI) or Python scripts, where implicit display doesn't happen.

df.head()

- Returns the first five rows of the DataFrame **as an object**.
- In interactive environments like Jupyter Notebook, the DataFrame is automatically displayed with proper formatting (e.g., styled tables).

code:

```
import pandas as pd  
  
df = pd.read_csv("car_sales.csv")  
  
num_attributes = df.shape[1]  
  
print("Number of attributes:", num_attributes)
```

- df.shape returns a tuple (rows, columns).
- df.shape[1] gives the number of columns (attributes).

Code:

```
import pandas as pd  
  
df = pd.read_csv("car_sales.csv")  
  
print("DataFrame Info:")  
  
print(df.info())  
  
df.info()
```

- Provides an overview of the dataset, including the number of **non-null values** and **data types** of columns.

`df.dtypes`

- Lists the **data type** of each column.

Code:

```
import pandas as pd
df = pd.read_csv("car_sales.csv")
num_rows, num_columns = df.shape
print("Number of rows:", num_rows)
print("Number of columns:", num_columns)
```

- `df.shape[0]` → **Number of rows (records)**
- `df.shape[1]` → **Number of columns (attributes)**

Duplicates:

Code:

```
df_cleaned = df.drop_duplicates()
print("Number of rows before removing duplicates:", df.shape[0])
print("Number of rows after removing duplicates:",
      df_cleaned.shape[0])
```

- **df.drop_duplicates()**
- Removes duplicate rows from the DataFrame.
- Returns a **new DataFrame** without modifying the original.
- **Checking the number of rows before and after**
- `df.shape[0]` → Number of rows **before** removing duplicates.

Print Summary Statistics for Numerical Variables

Code:

```
print("Summary Statistics for Numerical Variables:")  
print(df.describe())
```

df.describe() provides statistical details of numerical columns, such as:

- **count** (number of values)
- **mean** (average)
- **std** (standard deviation)
- **min** (minimum value)
- **25%, 50%, 75%** (quartiles)
- **max** (maximum value)

Missing Values in Each Column

- `df.isnull().sum()` counts missing values in each column.

```
print("Number of missing values in each column:")  
print(df.isnull().sum())
```

Drop the Column with Most Missing Values

- `df.isnull().sum().idxmax()` finds the column with the most missing values.

- `df.drop(columns=[column_to_drop])` removes it.

Impute (Fill) Missing Numerical Values

in our dataset (`car_sales.csv`), some rows have missing values in the set

- Incomplete data entry
- Errors in data collection
- Data corruption

To check missing values in the dataset:

```
print(df.isnull().sum())
```

+ in case occurring missing values in the data set

following this steps to fill the data **or** replaces missing values

```
df[" "].fillna(df[" "].mean(), inplace=True)
```

- `df[" "].mean()` → Calculates the mean (average) **example** of all available values.
- `fillna(df[" "].mean())` → Replaces missing values in the "example" column with this mean.
- `inplace=True` → Modifies the original df without creating a new copy.

Horsepower, Length, and Fuel Efficiency

+ Function Definition (`find_min_max`): The function calculates the min and max values for a given column in the DataFrame and returns the corresponding rows.

 Calling the Function: We call find_min_max with "Horsepower" as the column name to get the min/max horsepower values and their corresponding rows

Code:

```
import pandas as pd

# Assuming df is already loaded

df = pd.read_csv("car_sales.csv")

# Function to find min and max values of any column

def find_min_max(df, column_name):
    min_value = df[column_name].min()
    max_value = df[column_name].max()

    # Find the rows with the min and max values
    min_row = df[df[column_name] == min_value]
    max_row = df[df[column_name] == max_value]

    return min_value, max_value, min_row, max_row
```

```
# Now call the function to find min and max for Horsepower

hp_min, hp_max, hp_min_row, hp_max_row = find_min_max(df,
"Horsepower")

print("\nHorsepower - Min:", hp_min, "\n", hp_min_row)
print("\nHorsepower - Max:", hp_max, "\n", hp_max_row)
```

the Function:

- `df[column_name].min()`: This finds the **minimum value** in the specified column (column_name).
- `df[column_name].max()`: This finds the **maximum value** in the specified column (column_name).
- `df[df[column_name] == min_value]`: This filters the DataFrame and returns the rows where the column value equals the **minimum value**.
- `df[df[column_name] == max_value]`: This filters the DataFrame and returns the rows where the column value equals the **maximum value**.

Horsepower

```
# Get min and max horsepower values
hp_min = df["Horsepower"].min()
hp_max = df["Horsepower"].max()

# Get rows where horsepower is min/max
hp_min_row = df[df["Horsepower"] == hp_min]
hp_max_row = df[df["Horsepower"] == hp_max]

print("\nHorsepower - Min:", hp_min, "\n", hp_min_row)
print("\nHorsepower - Max:", hp_max, "\n", hp_max_row)
```

- `df["Horsepower"] == hp_min` → This filters the DataFrame to show only the rows where "Horsepower" is equal to the **minimum** value.
- `df["Horsepower"] == hp_max` → This filters the DataFrame to show only the rows where "Horsepower" is equal to the **maximum** value.

Why This Works:

- `df[df["Horsepower"] == hp_min]` → Filters out rows where the "Horsepower" column equals the min value (hp_min).
- `df[df["Horsepower"] == hp_max]` → Filters out rows where the "Horsepower" column equals the max value (hp_max).

Probability density distribution of continuous numerical variable-length

code:

```
import pandas as pd
from scipy.stats import gaussian_kde
```

```
# Load dataset
df = pd.read_csv("car_sales.csv")
```

```
df.columns = df.columns.str.strip()
```

```
if "Length" in df.columns:
```

```
df["Length"] = pd.to_numeric(df["Length"], errors="coerce")
```

```
data = df["Length"].dropna()
```

```
kde = gaussian_kde(data)
```

```
density_values = kde.evaluate(data)
```

```
result = pd.DataFrame({'Length': data, 'Density': density_values})
```

```
print(result.head())
```

```
else:
```

```
    print("Error: 'Length' column not found in the dataset.")
```

1. **gaussian_kde** is used to estimate the probability density function (PDF) of the "Length" column.
2. **dropna()** is used to remove any NaN values before estimation.
3. **evaluate(data)** computes the density for the data points.
4. **Outputs a DataFrame** with "Length" and its corresponding **Density**.

This will output the calculated **probability density values** for each "Length" value, which you can analyze numerically

Count by category –group by manufacture

To count the number of cars by manufacturer (or any other categorical column) in your dataset, you can use the groupby() function in pandas along with size() or count(). Here's how you can do it:

Code:

```
import pandas as pd  
df = pd.read_csv("car_sales.csv")  
manufacturer_counts =  
df.groupby('Manufacturer').size().reset_index(name='Count')  
print(manufacturer_counts)
```

1. **groupby('Manufacturer')**:

- Groups the dataset by the Manufacturer column.

2. **size()**:

- Counts the number of rows (cars) for each manufacturer.

3. **reset_index(name='Count')**:

- Converts the grouped result into a DataFrame and renames the count column to Count.

Boxplot of other numerical variables w.r.t manufacture

➤ Clean Column Names:

Spaces are trimmed from column names using df.columns.str.strip() to avoid mismatches.

➤ Select Numerical Columns:

We use df.select_dtypes(include=['number']) to pick out only

the numerical columns (e.g., Price, Sales, etc.) excluding Manufacturer.

➤ **Loop Through Numerical Columns:**

For each numerical column (e.g., Price, Horsepower), a boxplot is generated with sns.boxplot().

➤ **X-axis:** Represents different Manufacturers.

➤ **Y-axis:** Represents the numerical variable being analyzed (e.g., Price, Sales).

➤ **Boxplot Visualization:**

A boxplot is generated to visualize the distribution, including the **median**, **quartiles**, and potential **outliers** for each manufacturer.

➤ **Rotation of X-axis Labels:**

The x-axis labels (Manufacturer names) are rotated for better readability using plt.xticks(rotation=45).

Code:

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

```
df = pd.read_csv("car_sales.csv")
```

```
df.columns = df.columns.str.strip()
```

```
numerical_columns = df.select_dtypes(include=['number']).columns  
numerical_columns = [col for col in numerical_columns if col !=  
'Manufacturer']
```

```

if 'Manufacturer' in df.columns:

    # Create boxplots for each numerical column with respect to
    'Manufacturer'

    for col in numerical_columns:

        plt.figure(figsize=(10, 6))

        sns.boxplot(x='Manufacturer', y=col, data=df)

        plt.title(f'Boxplot of {col} by Manufacturer')

        plt.xlabel('Manufacturer')

        plt.ylabel(col)

        plt.xticks(rotation=45)

        plt.show()

else:

    print("Error: 'Manufacturer' column not found in the dataset.")

```

We aim to create **boxplots** for each numerical variable (like Price, Sales, Horsepower, etc.) with respect to the **Manufacturer**.⁴

Divide the data

1. Check Column Names:

- The column names are printed to help you verify the exact name of the sales column.
- A case-insensitive search is performed to find any column containing the word 'sales'.

2. Handle Missing Column:

- If no column related to 'sales' is found, a KeyError is raised with a helpful message.

3. Extract y (Output):

- The first matching column is used as the sales column, and its values are assigned to y.

4. Extract X (Input):

- The sales column is dropped from the dataset, and the remaining columns are assigned to X.

5. Display Shapes and Sample Data:

- The shapes of X and y are printed to confirm the split.
- The first few rows of X and y are displayed for verification.

Code:

```

import pandas as pd
df = pd.read_csv('car_sales.csv')
print("Column names in the dataset:")
print(df.columns)
sales_column = [col for col in df.columns if 'sales' in col.lower()]
if not sales_column:
    raise KeyError("No column related to 'sales' found in the dataset.
Please check the column names.")
sales_column_name = sales_column[0]
print(f"Using column '{sales_column_name}' as the sales column.")

y = df[sales_column_name]
X = df.drop(columns=[sales_column_name])

```

```
print("\nShape of X (input):", X.shape)
print("Shape of y (output):", y.shape)
```

```
print("\nInput (X):")
print(X.head())
print("\nOutput (y):")
print(y.head())
```

Encode other categorical variables using label encoder

What is Label Encoding?

Label Encoding is a method used to convert categorical (text) data into numerical form. It assigns a unique integer to each category in a column.

Need Label Encoding?

1. Machine Learning Requires Numeric Data
 - o ML models like **Linear Regression, Decision Trees, and Neural Networks** can't work with text data.
 - o Label encoding converts categorical data into numbers that models can understand.

2. Reduces Memory Usage

- Storing numbers instead of text is **more efficient** in terms of memory

Code:

```

import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('car_sales.csv')
print("Column names in the dataset:")
print(df.columns)
sales_column = [col for col in df.columns if 'sales' in col.lower()]
if not sales_column:
    raise KeyError("No column related to 'sales' found in the dataset.
Please check the column names.")
sales_column_name = sales_column[0]
print(f"Using column '{sales_column_name}' as the sales column.")
y = df[sales_column_name]
X = df.drop(columns=[sales_column_name])
categorical_columns = X.select_dtypes(include=['object',
'category']).columns
label_encoders = {}
for col in categorical_columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])
    label_encoders[col] = le
print("\nShape of X (input):", X.shape)

```

```

print("Shape of y (output):", y.shape)
print("\nInput (X) with encoded categorical variables:")
print(X.head())
print("\nOutput (y):")
print(y.head())
for col in categorical_columns:
    print(f"\nMapping for {col}:")
    print(dict(zip(label_encoders[col].classes_,
label_encoders[col].transform(label_encoders[col].classes_))))

```

Explanation of the Code:

1. Identify Categorical Columns:

- `X.select_dtypes(include=['object', 'category'])` identifies columns with categorical data (e.g., strings or categories).

2. Encode Categorical Columns:

- A LabelEncoder is applied to each categorical column.
- The `fit_transform` method converts categorical values into numerical labels.
- The encoders are stored in a dictionary (`label_encoders`) for potential inverse transformation later.

Encode categorical variable vehicle type using one-hot Encoder

One-Hot Encoding (OHE) is a technique used to convert categorical data into numerical format by creating separate binary columns for each unique category.

1. Machine Learning Models Need Numeric Input

- Many ML algorithms (**Linear Regression, Neural Networks**) don't understand text data.
- One-Hot Encoding converts text into numbers that models can process.

Code:

```
import pandas as pd  
df = pd.read_csv("car_sales.csv")  
df.columns = df.columns.str.strip()  
print("Available columns:", df.columns)
```

```
if 'Vehicle type' in df.columns:
```

```
    df = pd.get_dummies(df, columns=['Vehicle type'],  
prefix='Vehicle', drop_first=True)
```

```
    print(df.head())
```

```
else:
```

```
    print("Error: 'Vehicle type' column not found in the dataset.")
```

□ **df.columns = df.columns.str.strip()**

- **Removes extra spaces** from all column names, ensuring they match exactly.

□ **print("Available columns:", df.columns)**

- **Displays the column names** so you can check the correct spelling and formatting before using them.
- if 'Vehicle type' in df.columns:
 - **Checks if the column exists** before applying One-Hot Encoding.
- pd.get_dummies(df, columns=['Vehicle type'], prefix='Vehicle', drop_first=True)
 - Converts the categorical column into numerical columns (one column per category).
 - Uses drop_first=True to avoid redundancy (removes one category to prevent multicollinearity).

Split the Dataset into Train (70%) and Test (10%)

- train_test_split(X, y, test_size=0.30, random_state=42)
 - Splits **30%** of data as X_temp, y_temp (this includes both test + unused data).
 - Leaves **70%** as the X_train, y_train.
- train_test_split(X_temp, y_temp, test_size=2/3, random_state=42)
 - Takes **10% of the original dataset** as X_test, y_test.
 - The remaining **20% is ignored**, but you can use it as a validation set if needed.
- **Random State (random_state=42)**
 - Ensures the split is **reproducible** (same results every time).

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```

df = pd.read_csv("car_sales.csv")
df.columns = df.columns.str.strip()

X = df.drop(columns=['Sales in thousands'])
y = df['Sales in thousands']

X_train, X_temp, y_train, y_temp = train_test_split(X, y,
test_size=0.30, random_state=42)

X_test, _, y_test, _ = train_test_split(X_temp, y_temp,
test_size=2/3, random_state=42)

print("Training set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)

```

Final Data Distribution

Dataset	Percentage
Training Set (x_train, y_train)	70%
Test Set (x_test, y_test)	10%
Unused Data (or Validation)	20%

- ◊ If you also need a validation set, just store the ignored 20% as x_val, y_val instead of discarding it.

Data visualization with Seaborn Pairplot

Data Visualization is the presentation of data in pictorial format. It is extremely important for Data Analysis, primarily because of the fantastic ecosystem of data-centric Python packages. **Seaborn** is one of those packages that can make analyzing data much easier.

- It automatically selects numerical columns from the dataset.
- It creates scatter plots for every pair of numerical columns.
- It plots histograms (or KDE plots) on the diagonal to show individual distributions.

Code:

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

```
df = pd.read_csv("car_sales.csv")  
df.columns = df.columns.str.lower()  
sns.pairplot(df)  
plt.show()
```

- Converts column names to lowercase to avoid case mismatches.
- Uses sns.pairplot(df) to create pairwise scatter plots & histograms for all numerical columns.
- Displays the plot with plt.show().

Plot correlation of price and sales using scatterplot

- Reads the dataset: pd.read_csv("car_sales.csv")
- Handles case differences: Converts all column names to lowercase.
- Finds the correct columns:
 - Searches for a column containing "price".
 - Searches for a column containing "sales" or "units".
- Plots a scatter plot if both columns exist.

- **Handles errors:** If columns are missing, it prints an error message.

Code:

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

```
df = pd.read_csv("car_sales.csv")  
df.columns = df.columns.str.lower()  
price_col = next((col for col in df.columns if "price" in col), None)  
sales_col = next((col for col in df.columns if "sales" in col or "units"  
in col), None)
```

if price_col and sales_col:

```
    plt.figure(figsize=(8, 5))  
    sns.scatterplot(x=df[price_col], y=df[sales_col])  
    plt.xlabel(price_col.capitalize())  
    plt.ylabel(sales_col.capitalize())  
    plt.title("Price vs Sales Scatter Plot")  
    plt.show()
```

else:

```
    print("Error: Could not find 'price' and 'sales' columns.")
```

Boxplot of sales of different manufacturer

- Creates a sample dataset with "manufacturer" and "sales" columns.
- Uses sns.boxplot() to visualize the distribution of sales for each manufacturer.
- Adjusts figure size & labels for better readability

Code:

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
# Sample dataset with 'manufacturer' and 'sales' columns
```

```
data = {
```

```
    "manufacturer": ["Toyota", "Ford", "BMW", "Toyota", "Ford",
"BMW", "Toyota", "Ford", "BMW"],
```

```
    "sales": [500, 450, 520, 600, 490, 580, 620, 470, 590]
```

```
}
```

```
# Create DataFrame
```

```
df = pd.DataFrame(data)
```

```
# Create a boxplot of sales by manufacturer
```

```
plt.figure(figsize=(8, 5))
```

```
sns.boxplot(x=df["manufacturer"], y=df["sales"])
```

```
# Customize labels
```

```
plt.xlabel("Manufacturer")
```

```
plt.ylabel("Sales")
plt.title("Sales Distribution by Manufacturer")
plt.show()
```

- .
- Compares sales distributions across Toyota, Ford, and BMW.
 - Median (central line): Shows typical sales for each manufacturer.
 - Interquartile range (IQR): Box represents the middle 50% of sales.
 - Outliers (dots): Show any extreme values.

correlation coefficient value of price and sales

ex: sample Data:

- Creates a sample dataset with price and sales columns.
- Calculates the correlation coefficient using .corr(), which will give a value between -1 and 1.
- Prints the correlation coefficient between price and sales.

Code:

```
import pandas as pd
```

```
data = {
    "price": [20000, 25000, 30000, 35000, 40000, 45000, 50000],
    "sales": [250, 200, 150, 130, 120, 100, 90]
}
df = pd.DataFrame(data)
```

```
correlation = df[["price"]].corr(df[["sales"]])  
print(f"The correlation coefficient between price and sales is:  
{correlation}")
```

The correlation coefficient will indicate the **relationship** between price and sales.

- A **negative correlation** (near -1) means that as price increases, sales decrease.
- A **positive correlation** (near 1) means that as price increases, sales also increase.
- A **value near 0** suggests no linear correlation.

Sort and Find Most & Least Expensive Cars

- **Sort the data** by price in **ascending** order using `df.sort_values(by="price")`.
- **Find the least expensive car** by accessing the **first row** (`iloc[0]`) after sorting.
- **Find the most expensive car** by accessing the **last row** (`iloc[-1]`).
- **Print the details** of both the least and most expensive cars.

Code:

```
import pandas as pd
```

```
data = {  
    "car": ["Toyota", "Ford", "BMW", "Honda", "Mercedes", "Audi",  
    "Nissan"],  
    "price": [20000, 25000, 30000, 35000, 40000, 45000, 50000],  
}
```

```
"sales": [250, 200, 150, 130, 120, 100, 90]  
}
```

```
df = pd.DataFrame(data)  
df_sorted = df.sort_values(by="price")  
  
least_expensive_car = df_sorted.iloc[0]  
most_expensive_car = df_sorted.iloc[-1]  
print("Least Expensive Car:")  
print(least_expensive_car)  
  
print("\nMost Expensive Car:")  
print(most_expensive_car)
```

For the sample dataset:

- The **least expensive car** will be the one with the lowest price.
- The **most expensive car** will be the one with the highest price.

find min and max values of any column

□ Handle Non-Numeric Data:

- We convert the column data to numeric using `pd.to_numeric(df[column_name], errors='coerce')`. This will automatically convert any non-numeric values to NaN.

□ Remove Missing Values (NaN):

- The min() and max() functions will now ignore NaN values when calculating the minimum and maximum.

□ **Error Handling:**

- If min_value or max_value is NaN, an error message is returned stating that the column contains invalid or missing values.

Code:

```
import pandas as pd
```

```
def find_min_max(df, column_name):
```

```
    if column_name in df.columns:
        # Clean the column: Convert to numeric and ignore errors
        # (non-numeric values will be set as NaN)
```

```
        df[column_name] = pd.to_numeric(df[column_name],
                                         errors='coerce')
```

```
        min_value = df[column_name].min()
```

```
        max_value = df[column_name].max()
```

```
        if pd.isna(min_value) or pd.isna(max_value):
```

```
            return None, f"Error: Column '{column_name}' contains
            invalid or missing values."
```

```
        return min_value, max_value
    else:
        return None, f"Error: Column '{column_name}' not found in
the DataFrame."
```

```
df = pd.read_csv("car_sales.csv")
```

```
column_name = "Sales in thousands"
min_value, max_value = find_min_max(df, column_name)
```

```
if min_value is None:
```

```
    print(max_value)
```

```
else:
```

```
    print(f"Min Value of '{column_name}': {min_value}")
```

```
    print(f"Max Value of '{column_name}': {max_value}")
```

- If the "Fuel efficiency" column contains valid numeric data, the minimum and maximum values will be printed.
- If there are invalid or missing values in the column, you will see an error message indicating that.

Plot Histograms for Multiple Continuous Variables

Code:

Load the Data:

- We load the car_sales.csv dataset using pd.read_csv.

List of Columns:

- The columns **price**, **sales**, **horsepower**, and **fuelefficiency** are stored in a list columns for easy iteration.

Subplots:

- We set up a 2×2 grid of subplots using plt.subplots(2, 2, figsize=(12, 10)), where each plot will be placed in a specific position.

Plotting Histograms:

- For each variable, we use sns.histplot() to plot a histogram. The kde=True option adds a smooth **Kernel Density Estimate** curve to the histogram, which gives a better sense of the distribution.

Customize Titles and Labels:

- The set_title(), set_xlabel(), and set_ylabel() functions are used to customize the plot's title and labels.

Layout Adjustment:

- plt.tight_layout() is used to adjust the layout to avoid any overlap between plots.

Code:

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

```

df = pd.read_csv("car_sales.csv")
columns = ["Price in thousands", "Sales in thousands",
"Horsepower", "Fuel efficiency"]
fig, axes = plt.subplots(2, 2, figsize=(12, 10)) # 2 rows and 2
columns

for i, column in enumerate(columns):
    # Determine row and column index for the subplot
    ax = axes[i // 2, i % 2]

    # Plot histogram
    sns.histplot(df[column], kde=True, ax=ax)

    ax.set_title(f"Histogram of {column.capitalize()}")
    ax.set_xlabel(column.capitalize())
    ax.set_ylabel("Frequency")

plt.tight_layout()
plt.show()

```

This code will produce 4 histograms:

1. Histogram of price
2. Histogram of sales

3. Histogram of horsepower
4. Histogram of fuel efficiency

Each histogram will show the frequency distribution of these continuous variables.

Apply feature scaling on numerical variables

- We use `pd.to_numeric(df[columns], errors='coerce')` to convert all values in the specified columns to numeric. Any non-numeric values (like . or strings) will be converted to NaN.
- Handle Missing Values:**
 - We provide two options:
 - **Option 1:** Drop rows with NaN values using `df.dropna(subset=columns)`.
 - **Option 2:** Alternatively, you can fill missing values with the mean or any other strategy using `df.fillna(df.mean())`.
- Feature Scaling:**
 - The cleaned DataFrame (`df_cleaned`) is used for standardization (Z-score) and normalization (Min-Max scaling).

Code:

```
import pandas as pd  
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

```
df = pd.read_csv("car_sales.csv")  
columns = ["Price in thousands", "Sales in thousands",  
"Horsepower", "Fuel efficiency"]
```

```

df[columns] = df[columns].apply(pd.to_numeric, errors='coerce')
df_cleaned = df.dropna(subset=columns)
scaler = StandardScaler()
df_standardized = df_cleaned.copy()
df_standardized[columns] =
scaler.fit_transform(df_cleaned[columns])

normalizer = MinMaxScaler()

df_normalized = df_cleaned.copy()
df_normalized[columns] =
normalizer.fit_transform(df_cleaned[columns])

print("Standardized Data:")
print(df_standardized[columns].head())

print("\nNormalized Data:")
print(df_normalized[columns].head())

```

- **Data Cleaning:** Any invalid numeric data (like .) is converted to NaN and handled.
- **Standardized Data:** The numerical columns will be standardized to have a **mean of 0** and a **standard deviation of 1**.
- **Normalized Data:** The numerical columns will be scaled between **0** and **1**.

Exercise Questions:

1. Read top five values
2. Print dataframe info_data types of each column
3. Print number of rows and columns
4. Drop duplicate rows_if any
5. Print number of rows and columns after dropping duplicates
6. Print summary statistics for numerical variables
7. Print number of missing values in each column
8. Drop the column with most missing values
9. Drop the rows with categorical missing values
10. Import the rows with numerical missing values
11. Sort the data w.r.t price_find the details of the most and the least expensive cars
12. Write a function to find min and max values of any column
13. Call the above function to find min max of horse power, length, fuel efficiency
14. Plot histogram of continuous numerical variable :
price,sales,hoursepower,fuelefficiency
15. Probability density distribution of continuous numerical variable-length
16. Count by category –group by manufacture
17. Select all numerical variables
18. Print correlation coefficient value of price and sales
19. Plot correlation of price and sales using scatterplot
- 20.Pair plot

21. Boxplot of sales of different manufacturer
22. Boxplot of other numerical variables w.r.t manufacture
23. Divide the data into input and output $y = \text{sales}$ in thousand, $x = \text{all other variables}$
24. Encode other categorical variables using label encoder
25. Encode categorical variable vehicle type using one-hot encoder
26. Split the data set into train and test set 70% train set
10% test set
27. Apply feature scaling on numerical variables