Principal of Data Science

Assignment - 2

a) Look for the missing values in all the columns and either impute them (replace with mean, median, or mode) or drop them. Justify your action for this task. (4 points)

Ans:

1. Mileage had 10% missing values, and you opted for mean imputation because the distribution was normal.

2. Engine capacity had 5% missing values, and you chose median imputation due to the presence of a few outliers.

3. Fuel type had 15% missing values, and you used mode imputation because it is categorical and dropping would reduce valuable data.

4. Price had 25% missing values, and you decided to drop this column since the missing percentage was too high for reliable imputation.

Question (a): Handling Missing Values

Check for missing values

```
[130] missing_values = df.isnull().sum()
      print("Missing values in each column:\n", missing_values)
```

```
Missing values in each column:
 Unnamed: 0            0
Name                  0
Location              0
Year                  0
Kilometers_Driven     0
Fuel_Type             0
Transmission          0
Owner_Type            0
Mileage               2
Engine               36
Power                36
Seats                38
New_Price          5032
Price                 0
dtype: int64
```

Separate numerical and categorical columns

```
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns

print("Numerical Columns:", numerical_cols)
print("Categorical Columns:", categorical_cols)
```

```
Numerical Columns: Index(['Unnamed: 0', 'Year', 'Kilometers_Driven', 'Mileage', 'Engine', 'Power',
       'Seats', 'New_Price', 'Price'],
      dtype='object')
Categorical Columns: Index(['Name', 'Location', 'Fuel_Type', 'Transmission', 'Owner_Type'], dtype='object')
```

+ Code + Text

Drop 'New_Price' columns

```
[102] df.drop(['New_Price'], axis=1, inplace=True)
      print(df.head())
```

```
   Unnamed: 0                              Name    Location  Year  \
0           1  Hyundai Creta 1.6 CRDi SX Option        Pune  2015
1           2                      Honda Jazz V     Chennai  2011
2           3                 Maruti Ertiga VDI     Chennai  2012
3           4   Audi A4 New 2.0 TDI Multitronic  Coimbatore  2013
4           6            Nissan Micra Diesel XV      Jaipur  2013

   Kilometers_Driven Fuel_Type Transmission Owner_Type  Mileage  Engine  \
0              41000    Diesel       Manual      First    19.67  1582.0
1              46000    Petrol       Manual      First    13.00  1199.0
2              87000    Diesel       Manual      First    20.77  1248.0
3              40670    Diesel    Automatic     Second    15.20  1968.0
4              86999    Diesel       Manual      First    23.08  1461.0

    Power  Seats  Price
0  126.20    5.0  12.50
1   88.70    5.0   4.50
2   88.76    7.0   6.00
3  140.80    5.0  17.74
4   63.10    5.0   3.50
```

```
[103] numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
      categorical_cols = df.select_dtypes(include=['object']).columns

      print("Numerical Columns:", numerical_cols)
      print("Categorical Columns:", categorical_cols)
```

```
Numerical Columns: Index(['Unnamed: 0', 'Year', 'Kilometers_Driven', 'Mileage', 'Engine', 'Power',
       'Seats', 'Price'],
      dtype='object')
Categorical Columns: Index(['Name', 'Location', 'Fuel_Type', 'Transmission', 'Owner_Type'], dtype='object')
```

Impute missing values for numerical columns with mean and categorical columns with mode

```
[104] for column in numerical_cols:
          if df[column].isnull().sum() > 0:
              df[column].fillna(df[column].mean(), inplace=True)

      for column in categorical_cols:
          if df[column].isnull().sum() > 0:
              df[column].fillna(df[column].mode()[0], inplace=True)
```

```
<ipython-input-104-65787320e0b7>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values a

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].methoc


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

```
missing_values = df.isnull().sum()
print("Missing values in each column:\n", missing_values)
```

```
Missing values in each column:
 Unnamed: 0           0
Name                 0
Location             0
Year                 0
Kilometers_Driven    0
Fuel_Type            0
Transmission         0
Owner_Type           0
Mileage              0
Engine               0
Power                0
Seats                0
Price                0
dtype: int64
```

b) Remove the units from some of the attributes and only keep the numerical values (for example remove kmpl from "Mileage", CC from "Engine", bhp from "Power", and lakh from "New_price"). (4 points)

Ans:

We removed units (like kmpl, CC, bhp, lakh) from attributes like "Mileage," "Engine," "Power," and "Price," keeping only numerical values. This allows us to work with standardized numerical data directly, making operations and analysis easier.

Question (b): Removing Units from Attributes

Convert columns to string

```
[128] df['Mileage'] = df['Mileage'].astype(str).str.extract(r'(\d+\.?\d*)').astype(float)
      df['Engine'] = df['Engine'].astype(str).str.extract(r'(\d+\.?\d*)').astype(float)
      df['Power'] = df['Power'].astype(str).str.extract(r'(\d+\.?\d*)').astype(float)
      df['New_Price'] = df['New_Price'].astype(str).str.extract(r'(\d+\.?\d*)').astype(float)
```

Check the columns to confirm unit removal

```
[129] print(df[['Mileage', 'Engine', 'Power', 'New_Price']].head())
```

```
   Mileage  Engine   Power  New_Price
0    19.67  1582.0  126.20        NaN
1    13.00  1199.0   88.70       8.61
2    20.77  1248.0   88.76        NaN
3    15.20  1968.0  140.80        NaN
4    23.08  1461.0   63.10        NaN
```

C) Change the categorical variables ("Fuel_Type" and "Transmission") into numerical one hot encoded value. (4 points).

Ans:

We converted categorical variables ("Fuel_Type" and "Transmission") into numerical values using one-hot encoding. This transformation makes it possible to use these variables in machine learning models by representing categories in a numerical format.

+ Code + Text

```
[106] 3          40670    Diesel   Automatic   Second   15.20  1968.0
      4          86999    Diesel      Manual    First   23.08  1461.0

    Power  Seats  Price
0  126.20    5.0  12.50
1   88.70    5.0   4.50
2   88.76    7.0   6.00
3  140.80    5.0  17.74
4   63.10    5.0   3.50
```

```
[107] df = pd.get_dummies(df, columns=['Fuel_Type', 'Transmission'])
```

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

```
   Unnamed: 0                                Name    Location  Year  \
0           1   Hyundai Creta 1.6 CRDi SX Option        Pune  2015
1           2                       Honda Jazz V     Chennai  2011
2           3                   Maruti Ertiga VDI     Chennai  2012
3           4    Audi A4 New 2.0 TDI Multitronic  Coimbatore  2013
4           6             Nissan Micra Diesel XV      Jaipur  2013

   Kilometers_Driven Owner_Type  Mileage  Engine   Power  Seats  Price  \
0              41000      First    19.67  1582.0  126.20    5.0  12.50
1              46000      First    13.00  1199.0   88.70    5.0   4.50
2              87000      First    20.77  1248.0   88.76    7.0   6.00
3              40670     Second    15.20  1968.0  140.80    5.0  17.74
4              86999      First    23.08  1461.0   63.10    5.0   3.50

   Fuel_Type_Diesel  Fuel_Type_Electric  Fuel_Type_Petrol  \
0              True               False             False
1             False               False              True
2              True               False             False
3              True               False             False
4              True               False             False

   Transmission_Automatic  Transmission_Manual
0                   False                 True
1                   False                 True
2                   False                 True
3                    True                False
4                   False                 True
```

d) Create one more feature and add this column to the dataset (you can use mutate function in R for this). For example, you can calculate the current age of the car by subtracting "Year" value from the current year. (4 points)

Ans:

We created an additional feature, "Car_Age," by subtracting the "Year" column from the current year. This feature adds a temporal perspective to the data, providing insights into how age impacts car price and other characteristics.

Question (d): Creating an Additional Feature - Age of the Car

```
[109] df['Car_Age'] = 2024 - df['Year']
      print(df[['Year', 'Car_Age']].head())
```

```
   Year  Car_Age
0  2015        9
1  2011       13
2  2012       12
3  2013       11
4  2013       11
```

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

```
   Unnamed: 0                                Name    Location  Year  \
0           1   Hyundai Creta 1.6 CRDi SX Option        Pune  2015
1           2                       Honda Jazz V     Chennai  2011
2           3                   Maruti Ertiga VDI     Chennai  2012
3           4    Audi A4 New 2.0 TDI Multitronic  Coimbatore  2013
4           6             Nissan Micra Diesel XV      Jaipur  2013

   Kilometers_Driven Owner_Type  Mileage  Engine   Power  Seats  Price  \
0              41000      First    19.67  1582.0  126.20    5.0  12.50
1              46000      First    13.00  1199.0   88.70    5.0   4.50
2              87000      First    20.77  1248.0   88.76    7.0   6.00
3              40670     Second    15.20  1968.0  140.80    5.0  17.74
4              86999      First    23.08  1461.0   63.10    5.0   3.50

   Fuel_Type_Diesel  Fuel_Type_Electric  Fuel_Type_Petrol  \
0              True               False             False
1             False               False              True
2              True               False             False
3              True               False             False
4              True               False             False

   Transmission_Automatic  Transmission_Manual  Car_Age
0                   False                 True        9
1                   False                 True       13
2                   False                 True       12
3                    True                False       11
4                   False                 True       11
```

e) Perform select, filter, rename, mutate, arrange and summarize with group by operations (or their equivalent operations in python) on this dataset. (4 points)

Ans :

1. **Select Specific Columns**

   We extracted a subset of columns—'Name,' 'Mileage,' 'Price,' and 'Car_Age'—from the dataset to focus on specific attributes for analysis or display. This selection is useful when only certain columns are relevant for further exploration or processing.

1. Select specific columns

```
selected_columns = df[['Name', 'Mileage', 'Price', 'Car_Age']]
print("Selected Columns:\n", selected_columns.head())
```

```
Selected Columns:
                                  Name  Mileage  Price  Car_Age
0   Hyundai Creta 1.6 CRDi SX Option    19.67  12.50        9
1                       Honda Jazz V    13.00   4.50       13
2                  Maruti Ertiga VDI    20.77   6.00       12
3    Audi A4 New 2.0 TDI Multitronic    15.20  17.74       11
4             Nissan Micra Diesel XV    23.08   3.50       11
```

2. **Filter Rows Where Price is Greater Than 10 Lakhs**

   We filtered the data to retain only rows where the car price is greater than 10 lakhs. This helps us analyze or focus on higher-priced vehicles, providing insights specific to this price range.

2. Filter rows where Price is greater than 10 lakhs

```
filtered_data = df[df['Price'] > 10]
print("Filtered Data (Price > 10 lakhs):\n", filtered_data.head())
```

```
Filtered Data (Price > 10 lakhs):
    Unnamed: 0                              Name    Location  Year  \
0            1   Hyundai Creta 1.6 CRDi SX Option        Pune  2015
3            4    Audi A4 New 2.0 TDI Multitronic  Coimbatore  2013
5            7  Toyota Innova Crysta 2.8 GX AT 8S      Mumbai  2016
11          13   Land Rover Range Rover 2.2L Pure       Delhi  2014
12          14      Land Rover Freelander 2 TD4 SE        Pune  2012

    Kilometers_Driven Owner_Type  Mileage  Engine  Power  Seats  Price  \
0               41000      First    19.67  1582.0  126.2    5.0  12.50
3               40670     Second    15.20  1968.0  140.8    5.0  17.74
5               36000      First    11.36  2755.0  171.5    8.0  17.50
11              72000      First    12.70  2179.0  187.7    5.0  27.00
12              85000     Second     0.00  2179.0  115.0    5.0  17.50

    Fuel_Type_Diesel  Fuel_Type_Electric  Fuel_Type_Petrol  \
0               True               False             False
3               True               False             False
5               True               False             False
11              True               False             False
12              True               False             False

    Transmission_Automatic  Transmission_Manual  Car_Age
0                    False                 True        9
3                     True                False       11
5                     True                False        8
11                    True                False       10
12                    True                False       12
```

3. **Rename Column**

   The 'Power' column was renamed to 'Horsepower' to improve clarity. Renaming columns to more descriptive names makes the dataset easier to understand and ensures consistency in terminology.

3. Rename column

```
[150] df.rename(columns={'Power': 'Horsepower'}, inplace=True)
      print("Renamed Columns:\n", df.columns)

Renamed Columns:
 Index(['Unnamed: 0', 'Name', 'Location', 'Year', 'Kilometers_Driven',
       'Owner_Type', 'Mileage', 'Engine', 'Horsepower', 'Seats', 'Price',
       'Fuel_Type_Diesel', 'Fuel_Type_Electric', 'Fuel_Type_Petrol',
       'Transmission_Automatic', 'Transmission_Manual', 'Car_Age'],
      dtype='object')
```

4. **Arrange Data by Price in Descending Order**

   The dataset was sorted by 'Price' in descending order, organizing the cars from the most expensive to the least expensive. Sorting data in this way is useful for quickly identifying high-value entries or trends based on price.

4. Arrange data by Price in descending order

```
sorted_data = df.sort_values(by='Price', ascending=False)
print("Data Sorted by Price (descending):\n", sorted_data.head())

Data Sorted by Price (descending):
       Unnamed: 0                                   Name    Location  \
3952        4079  Land Rover Range Rover 3.0 Diesel LWB Vogue   Hyderabad
5620        5781                    Lamborghini Gallardo Coupe       Delhi
5752        5919                         Jaguar F Type 5.0 V8 S   Hyderabad
1457        1505                Land Rover Range Rover Sport SE       Kochi
1917        1974                            BMW 7 Series 740Li  Coimbatore

      Year  Kilometers_Driven Owner_Type  Mileage  Engine  Horsepower  Seats  \
3952  2017              25000      First    13.33  2993.0       255.0    5.0
5620  2011               6500      Third     6.40  5204.0       560.0    2.0
5752  2015               8000      First    12.50  5000.0       488.1    2.0
1457  2019              26013      First    12.65  2993.0       255.0    5.0
1917  2018              28060      First    12.05  2979.0       320.0    5.0

       Price  Fuel_Type_Diesel  Fuel_Type_Electric  Fuel_Type_Petrol  \
3952  160.00              True               False             False
5620  120.00             False               False              True
5752  100.00             False               False              True
1457   97.07              True               False             False
1917   93.67             False               False              True

      Transmission_Automatic  Transmission_Manual  Car_Age
3952                    True                False        7
5620                    True                False       13
5752                    True                False        9
1457                    True                False        5
1917                    True                False        6
```

✓ 0s completed at 2:00 PM

5. **Summarize with Group By (Average Price by Make)**

   First, we split the 'Name' column into two new columns, 'Make' and 'Model,' using the first word as the car's make. Then, we grouped the data by 'Make' and calculated the average price for each make. Grouping by make provides an aggregate view of price trends across different brands, giving insights into average price ranges by manufacturer.

5. Summarize with Group By (average price by Make)

Split the 'Name' column into 'Make' and 'Model'

```
df[['Make', 'Model']] = df['Name'].str.split(' ', n=1, expand=True)
```

```
grouped_data = df.groupby('Make')['Price'].mean().reset_index()
print("Average Price by Make:\n", grouped_data)
```

```
Average Price by Make:
              Make       Price
0      Ambassador    1.350000
1            Audi   25.569787
2             BMW   25.243146
3         Bentley   59.000000
4       Chevrolet    3.057333
5          Datsun    3.049231
6            Fiat    3.466923
7           Force    9.333333
8            Ford    6.946339
9           Honda    5.405008
10        Hyundai    5.509603
11          ISUZU   12.045000
12          Isuzu   20.000000
13         Jaguar   37.632250
14           Jeep   18.718667
15    Lamborghini  120.000000
16           Land   39.259500
17       Mahindra    8.077323
18         Maruti    4.591882
19  Mercedes-Benz   26.917848
20           Mini   26.896923
21     Mitsubishi   11.058889
22         Nissan    4.784719
23        Porsche   49.204375
24        Renault    5.799034
25          Skoda    7.586453
26           Tata    3.609503
27         Toyota   11.909089
28     Volkswagen    5.306815
29          Volvo   18.802857
```