Principles of Data Science – 5530

Assignment-2

Name: Nikhil Jagadeesh Sriram

Student ID: 16352573

(A) In this context, I am addressing the identification and handling of missing values across all columns. For categorical columns, the missing values are being imputed with the mode. This approach is chosen to ensure the preservation of the existing data within that categorical field, thereby minimizing the potential for bias in the imputed values. As for numerical columns, missing values are being imputed with the mean. This method is adopted to maintain the distribution and central tendency of the available data in the numerical column, aligning the imputed values with the overall dataset distribution and mitigating the risk of introducing noticeable bias.

```
missing_values = data_given.isnull().sum()
print(missing_values) # Checking for missing values

miss_col = []
for col, value in missing_values.items():
    if value > 0:
        miss_col.append(col)

print("Columns with missing values:", miss_col)
```

```
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
Columns with missing values: ['Mileage', 'Engine', 'Power', 'Seats', 'New_Price']
```

```
for col in miss_col:
    if data_given[col].dtype == 'object':
        data_given[col].fillna(data_given[col].mode()[0], inplace=True)
    else:
        data_given[col].fillna(data_given[col].mean(), inplace=True)
```

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").

```
column_data_types = car_data.dtypes

# Print the data types of columns
print(column_data_types)
```

```
Unnamed: 0             int64
Name                  object
Location              object
Year                   int64
Kilometers_Driven      int64
Fuel_Type             object
Transmission          object
Owner_Type            object
Mileage               object
Engine                object
Power                 object
Seats                float64
New_Price             object
Price                float64
dtype: object
```

```
data_given
```

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	4.78 Lakh	12.50

(B) Strip the units from select attributes, retaining only the numerical values. For instance, eliminate "kmpl" from the "Mileage" column, "CC" from the "Engine" column, "bhp" from the "Power" column, and "lakh" from the "New_price" column.

5842	6014	Maruti Swift VDI	Delhi	2014	27365	Diesel	Manual	First	28.4 kmpl	1248 CC	74 bhp	5.0	7.88 Lakh	4.75
5843	6015	Hyundai Xcent 1.1 CRDi S	Jaipur	2015	100000	Diesel	Manual	First	24.4 kmpl	1120 CC	71 bhp	5.0	4.78 Lakh	4.00
5844	6016	Mahindra Xylo D4 BSIV	Jaipur	2012	55000	Diesel	Manual	Second	14.0 kmpl	2498 CC	112 bhp	8.0	4.78 Lakh	2.90
5845	6017	Maruti Wagon R VXI	Kolkata	2013	46000	Petrol	Manual	First	18.9 kmpl	998 CC	67.1 bhp	5.0	4.78 Lakh	2.65
5846	6018	Chevrolet Beat Diesel	Hyderabad	2011	47000	Diesel	Manual	First	25.44 kmpl	936 CC	57.6 bhp	5.0	4.78 Lakh	2.50

5847 rows × 14 columns

```
data_given['Mileage'] = data_given['Mileage'].str.extract('(\d+.\d+)').astype(float)
data_given['Engine'] = data_given['Engine'].str.replace(' CC', '').astype(int)
data_given['Power'] = data_given['Power'].str.extract('(\d+.\d+)').astype(float)
data_given['New_Price'] = data_given['New_Price'].str.extract('(\d+.\d+)').astype(float)
```

+ Code + Text

```
data_given
```

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67	1582	126.20	5.0	4.78	12.50
1	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	NaN	1199	88.70	5.0	8.61	4.50
2	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77	1248	88.76	7.0	4.78	6.00
3	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.20	1968	140.80	5.0	4.78	17.74
4	6	Nissan Micra Diesel XV	Jaipur	2013	86999	Diesel	Manual	First	23.08	1461	63.10	5.0	4.78	3.50

(C) Convert the categorical variables, namely "Fuel_Type" and "Transmission," into numerical one-hot encoded values.

One-hot encoding is a technique used to convert categorical variables into a binary matrix where each category becomes a separate column, and a binary value (0 or 1) indicates the presence or absence of that category for each observation. In the context of your dataset:

For example, if you have a "Fuel_Type" column with categories 'Petrol' and 'Diesel', after one-hot encoding, you will have two new columns: 'Fuel_Type_Petrol' and 'Fuel_Type_Diesel'. For each row, the column corresponding to the actual fuel type will have a value of 1, while the other column will have a value of 0.

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

```
label_encoder = LabelEncoder()
data_given['Fuel_Type_Label'] = label_encoder.fit_transform(data_given['Fuel_Type'])
data_given['Transmission_Label'] = label_encoder.fit_transform(data_given['Transmission'])

onehot_encoder = OneHotEncoder(sparse=False)

fuel_type_encoded = onehot_encoder.fit_transform(data_given[['Fuel_Type_Label']])
transmission_encoded = onehot_encoder.fit_transform(data_given[['Transmission_Label']])

fuel_type_encoded_df = pd.DataFrame(fuel_type_encoded, columns=['Fuel_Type_' + str(i) for i in range(fuel_type_encoded.shape[1])])
transmission_encoded_df = pd.DataFrame(transmission_encoded, columns=['Transmission_' + str(i) for i in range(transmission_encoded.shape[1])])

data_given = pd.concat([data_given, fuel_type_encoded_df, transmission_encoded_df], axis=1)

data_given.drop(['Fuel_Type', 'Transmission', 'Fuel_Type_Label', 'Transmission_Label'], axis=1, inplace=True)
```

(D) Introduce a new feature by calculating the current age of each car and add this column to the dataset. One way to achieve this is by subtracting the "Year" value from the current year.

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.

```
current_year = datetime.now().year
data_given['Current_Age'] = current_year - data_given['Year']
data_given
```

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price	Fuel_Type_0	Fuel_Type_1	Fuel_Type_2	Transmission_0	Transm
0	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	First	19.67	1582	126.20	5.0	4.78	12.50	1.0	0.0	0.0	0.0	
1	2	Honda Jazz V	Chennai	2011	46000	First	NaN	1199	88.70	5.0	8.61	4.50	0.0	0.0	1.0	0.0	
2	3	Maruti Ertiga VDI	Chennai	2012	87000	First	20.77	1248	88.76	7.0	4.78	6.00	1.0	0.0	0.0	0.0	
3	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Second	15.20	1968	140.80	5.0	4.78	17.74	1.0	0.0	0.0	1.0	
4	6	Nissan Micra	Jaipur	2013	86999	First	23.08	1461	63.10	5.0	4.78	3.50	1.0	0.0	0.0	0.0	

(E) Perform select, filter, rename, mutate, arrange and summarize with group by operations on this dataset.

In data manipulation, the **select** operation involves choosing specific columns, **filter** is used to subset rows based on conditions, **rename** is employed to change column names, **mutate** creates new columns, **arrange** sort rows based on specified criteria, and **summarize** with **group by** aggregates data by groups, calculating summary statistics for each group.

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

```
selected_data = data_given[['Name', 'Year','Kilometers_Driven']]#select
print(selected_data)
```

```
                             Name  Year  Kilometers_Driven
0     Hyundai Creta 1.6 CRDi SX Option  2015              41000
1                      Honda Jazz V  2011              46000
2                 Maruti Ertiga VDI  2012              87000
3      Audi A4 New 2.0 TDI Multitronic  2013              40670
4             Nissan Micra Diesel XV  2013              86999
...                             ...   ...                ...
5842              Maruti Swift VDI  2014              27365
5843        Hyundai Xcent 1.1 CRDi S  2015             100000
5844          Mahindra Xylo D4 BSIV  2012              55000
5845             Maruti Wagon R VXI  2013              46000
5846          Chevrolet Beat Diesel  2011              47000

[5847 rows x 3 columns]
```

```
filtered_data = data_given[data_given['Location'] == 'Jaipur']#filtered Data
print(filtered_data)
```

```
      Unnamed: 0                  Name Location  Year  Kilometers_Driven  \
4              6  Nissan Micra Diesel XV   Jaipur  2013              86999
10            12   Maruti Swift VDI BSIV   Jaipur  2015              64424
15            17   Maruti Swift DDiS VDI   Jaipur  2017              25000
```

```
[403 rows x 18 columns]
```

```
data_given = data_given.rename(columns={'New_Price': 'NewPriceCAR'})#rename
data_given
```

Unnamed: 0	Name	Location	Year	Kilometers_Driven	Owner_Type	Mileage	Engine	Power	Seats	N
	Hyundai									

Diesel

5847 rows × 18 columns

```
data_given['Increseprice'] =  data_given['NewPriceCAR']- data_given['Price']
data_given
```

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Owner_Type	Mileage	Engine	Power	Seats	NewPriceCAR	Price	Fuel_Ty
0	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	First	19.67	1582	126.20	5.0	4.78	12.50	

5847 rows × 19 columns

```
sorted_car_data = car_data.sort_values(by='Price', ascending=True)
sorted_car_data#arrange
```

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Owner_Type	Mileage	Engine	Po
1660	1713	Tata Nano Lx	Pune	2011	65000	Second	26.00	624	

5847 rows × 19 columns

```
average_price_by_fuel_type = data_given.groupby('Kilometers_Driven')['Mileage'].mean()#summarize
average_price_by_fuel_type
```

```
Kilometers_Driven
171        24.700000
600        21.500000
1000       18.493333
1001       22.415000
1011       20.370000
             ...
480000     17.180000
620000     20.360000
720000     20.540000
775000     19.300000
6500000    15.970000
Name: Mileage, Length: 3019, dtype: float64
```

```
data_given.to_csv('clean_data.csv', index=False)
```