Student name: Vangari Prashanth -11645119

Assignment: Regression

Part 1: Data Wrangling (40 pts)

You have to write code to answer the questions below 4 pts each subtask except for the first one (importing pandas...)

- Import pandas library
- Read the car_price data stored in your local machine
- Save data to a variable named df
- Show it's information such as column titles, types of the columns

```
import pandas as pd
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
file_path= '/content/drive/My Drive/Machine Learning/Regression Assignment/car_price.csv'

# Read the car_price data stored in your local machine
df = pd.read_csv(file_path)
df.dtypes
```

```
Mounted at /content/drive
Unnamed: 0              int64
car_name               object
car_prices_in_rupee    object
kms_driven             object
fuel_type              object
transmission           object
ownership              object
manufacture             int64
engine                 object
Seats                  object
dtype: object
```

```
# Drop the first column: Unnamed: 0 and show the last 5 rows of the new dataset
df.drop(columns=['Unnamed: 0'], axis=1, inplace=True)
df.tail(5)
```

	car_name	car_prices_in_rupee	kms_driven	fuel_type	transmission	ownership	manufacture	engine	Seats
5507	BMW X1 sDrive 20d xLine	28.90 Lakh	45,000 kms	Diesel	Automatic	1st Owner	2018	2995 cc	7 Seats
5508	BMW M Series M4 Coupe	64.90 Lakh	29,000 kms	Petrol	Automatic	2nd Owner	2015	1968 cc	5 Seats
5509	Jaguar XF 2.2 Litre Luxury	13.75 Lakh	90,000 kms	Diesel	Automatic	2nd Owner	2013	2755 cc	5 Seats
	BMW 7 Series							2967	6

```
# Turn all columns to lowercase and show the new columns
df = df.apply(lambda x: x.astype(str).str.lower() if x.dtype== 'object' else x)
df
```

	car_name	car_prices_in_rupee	kms_driven	fuel_type	transmission	ownership	manufacture	engine	Seats
0	jeep compass 2.0 longitude option bsiv	10.03 lakh	86,226 kms	diesel	manual	1st owner	2017	1956 cc	5 seats
1	renault duster							1330	5

```
'''
Show unique values of columns that satisfy the following:
    The columns are of object type
    The number of unique values is smaller than 7
'''

output = []
for columnName in df.columns:
  if df.dtypes[columnName] == 'object' and len(df[columnName].unique()) < 7:
    output.append((columnName,df[columnName].unique()))
output

for columnName, uniqueValues in output:
  print(f"Column '{columnName}' has following unique values: {uniqueValues}")
```

```
Column 'fuel_type' has following unique values: ['diesel' 'petrol' 'cng' 'electric' 'lpg']
Column 'transmission' has following unique values: ['manual' 'automatic']
Column 'ownership' has following unique values: ['1st owner' '2nd owner' '3rd owner' '4th owner' '5th owner' '0th owner']
Column 'Seats' has following unique values: ['5 seats' '6 seats' '7 seats' '4 seats' '8 seats' '2 seats']
```

```
'''
Some columns have redundant values such as Lakh, kms, Owner, cc, Seats. Remove them
You have to additionally take care of columns
    ownership such that the result does not have any character like the console
    kms_driven and car_prices_in_rupee such that the result does not have commas like the console
'''

import re

for column in ['ownership','kms_driven','car_prices_in_rupee', 'engine', 'Seats' ]:
  if column == 'ownership':
    df[column] = df[column].str.extract('(^\d*)')
  elif df[column].dtypes == 'object':
    df[column] = df[column].str.split(r' ').str.get(0)
    df[column] = df[column].str.replace(',', '', regex=True)
df
```

	car_name	car_prices_in_rupee	kms_driven	fuel_type	transmission	ownership	manufacture	engine	Seats
0	jeep compass 2.0 longitude option bsiv	10.03	86226	diesel	manual	1	2017	1956	5
1	renault duster rxz turbo cvt	12.83	13248	petrol	automatic	1	2021	1330	5
2	toyota camry 2.5 g	16.40	60343	petrol	automatic	1	2016	2494	5
3	honda jazz vx cvt	7.77	26696	petrol	automatic	1	2018	1199	5
4	volkswagen polo 1.2 mpi highline	5.15	69414	petrol	manual	1	2016	1199	5
...	...	...	...	...	...	...	...	...	...
5507	bmw x1 sdrive 20d xline	28.90	45000	diesel	automatic	1	2018	2995	7
5508	bmw m series m4 coupe	64.90	29000	petrol	automatic	2	2015	1968	5

```
'''
Convert columns car_prices_in_rupee, kms_driven, ownership, engine, and seats to numeric
'''

columns = ['car_prices_in_rupee', 'kms_driven', 'ownership', 'engine', 'Seats']

for col in columns:
    df[col] = pd.to_numeric(df[col])
```

```
df.head(5)
```

	car_name	car_prices_in_rupee	kms_driven	fuel_type	transmission	ownership	manufacture	engine	Seats
0	jeep compass 2.0 longitude option bsiv	10.03	86226	diesel	manual	1	2017	1956	5
1	renault duster rxz turbo cvt	12.83	13248	petrol	automatic	1	2021	1330	5
2	toyota camry 2.5 g	16.40	60343	petrol	automatic	1	2016	2494	5
3	honda jazz vx cvt	7.77	26696	petrol	automatic	1	2018	1199	5

```
# Show average values of all numeric columns by "car_name" in a same DataFrame table. Do not overwrite df.
group =  df.groupby('car_name')
group.mean(numeric_only=True)
```

car_name	car_prices_in_rupee	kms_driven	ownership	manufacture	engine	Seats
audi a3 35 tdi premium	29.500000	11000.000000	1.000000	2017.000000	1086.000000	5.000000
audi a3 35 tdi premium plus	23.430909	44936.454545	1.181818	2019.000000	1667.272727	5.545455
audi a4 1.8 tfsi	14.800000	57500.000000	3.000000	2013.000000	1345.000000	5.000000
audi a4 2.0 tdi	10.478333	77771.666667	2.000000	2012.833333	1740.833333	5.666667
audi a4 2.0 tdi multitronic	13.505000	90000.000000	1.000000	2013.500000	1783.000000	5.000000
...	...	...	...	...	...	...
volvo xc60 d4 summum	15.000000	105241.000000	1.000000	2015.000000	1985.000000	5.000000
volvo xc60 d5 inscription	52.000000	42285.714286	1.000000	2019.000000	1737.000000	5.428571
volvo xc60 d5 summum	15.560000	126553.000000	1.000000	2013.000000	999.000000	7.000000
volvo xc60 inscription d5	60.000000	7500.000000	1.000000	2020.000000	1498.000000	5.500000
volvo xc60 inscription d5 bsiv	56.750000	13000.000000	1.000000	2019.000000	1086.000000	5.000000

1882 rows × 6 columns

```
'''
Show the sum of kms_driven and max of car_prices by "seats" in a same DataFrame table.
Rename the aggregated columns as the console
Do not overwrite df.
'''
output = df.groupby('Seats').agg(kilometers=('kms_driven', 'sum'), price=('car_prices_in_rupee', 'max'))
output
```

Seats	kilometers	price
2	203700	47.0
4	5598853	90000.0
5	294601603	99999.0
6	3584287	60.0
7	40263535	90000.0
8	4171949	90000.0

```
'''
Encode the categorical columns to numeric. There are two types of encoding: ordinal and one-hot. Explain why you choose the encoding techniqu
Reference (you may need incognito mode to browse the pages):
    https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html
    https://towardsdatascience.com/guide-to-encoding-categorical-features-using-scikit-learn-for-machine-learning-5048997a5c79
    https://stackoverflow.com/questions/56502864/using-ordinalencoder-to-transform-categorical-values
    https://stackoverflow.com/questions/37292872/how-can-i-one-hot-encode-in-python
    https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html
'''
```

```
import sklearn.preprocessing
oneHotEncoding = pd.get_dummies(df[['car_name','fuel_type', 'transmission']])
df = df.drop(['car_name','fuel_type', 'transmission'],axis = 1)
df = df.join(oneHotEncoding)
df.head(5)

'''
Explanation:
Ordinal encoding is used for categorical columns that have ordering.
One-hot encoding is used for categorical columns that have no ordering.

Here, One-hot encoding is used because the columns such as 'car_name','fuel_type', 'transmission' does not require any order.
'''
```

```
'\nExplanation:\nOrdinal encoding is used for categorical columns that have ordering.\nOne-hot encoding is used for catego
rical columns that have no ordering.\n\nHere, One-hot encoding is used because the columns such as 'car_name','fuel_type',
'transmission' does not require any order.\n'
```

```
# Reset the index such that it starts from 1 (instead of 0) and print the first five rows
df.reset_index(drop = True)
df.index = df.index+1
df.head(5)
```

	car_prices_in_rupee	kms_driven	ownership	manufacture	engine	Seats	car_name_audi a3 35 tdi premium	car_name_audi a3 35 tdi premium plus	car_name_audi a4 1.8 tfsi	car_ a
1	10.03	86226	1	2017	1956	5	0	0	0	
2	12.83	13248	1	2021	1330	5	0	0	0	
3	16.40	60343	1	2016	2494	5	0	0	0	
4	7.77	26696	1	2018	1199	5	0	0	0	
5	5.15	69414	1	2016	1199	5	0	0	0	

5 rows × 1895 columns

```
# Return boolean values indicating the number of missing rows of each column. Do not overwrite df.
df.isna().sum()
```

```
car_prices_in_rupee       0
kms_driven                0
ownership                 0
manufacture               0
engine                    0
                         ..
fuel_type_electric        0
fuel_type_lpg             0
fuel_type_petrol          0
transmission_automatic    0
transmission_manual       0
Length: 1895, dtype: int64
```

Part 2: Regression (50 pts)

Assign X to be the whole df without column price and y to be the column price. Split X and y into X_train, X_test, y_train, and y_test with **random_state=1** and test_size=0.2.

Reference: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
from sklearn.model_selection import train_test_split
import numpy as np

correlationMatrix = df.corr()
X = df.drop('car_prices_in_rupee', axis=1)
X = df[['kms_driven']]
X = X.values

y = df[['car_prices_in_rupee']]
y = y.values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Write a class My_LinearR that implements LinearRegression algorithm. You are required to have the following attributes

- Method:
  - fit
  - predict

Reference: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Using a pre-built library yields no credit. You have to write everything from scratch.

```
import numpy as np

class My_LinearR:
    def calculate_coefficients(self, x, y):
        x_mean = np.mean(x)
        y_mean = np.mean(y)
        covariance = np.sum((x - x_mean) * (y - y_mean))
        variance = np.sum((x - x_mean) ** 2)
        m = covariance / variance
        b = y_mean - m * x_mean
        return m, b

    def fit(self, X_train, y_train):
        self.coefficients = self.calculate_coefficients(X_train, y_train)

    def predict(self, X_test):
        m, b = self.coefficients
        return [(m * int(x) + b) for x in X_test]



# Run the code
reg = My_LinearR()
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
```

Part 3: Metric (10 pts)

```
# Use three of regression metrics in https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics to compute errors between y_

from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
print("Mean Squared Error: "+str(mean_squared_error(y_test, y_pred)))
print("Mean Absolute Error: "+str(mean_absolute_error(y_test, y_pred)))
print("R2 score: "+str(r2_score(y_test, y_pred)))
```

Which one do you think is the best metric of the three? Explain.

Answer:

The best regression metrics is "Mean squared error" becuase Mean Square Errors penalizes large errors more heavily than small errors.