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	manufa
5507	BMW X1 sDrive 20d xLine	28.90 Lakh	45,000 kms	Diesel	Automatic	1st Owner	
5508	BMW M Series M4 Coupe	64.90 Lakh	29,000 kms	Petrol	Automatic	2nd Owner	
5509	Jaguar XF 2.2 Litre Luxury	13.75 Lakh	90,000 kms	Diesel	Automatic	2nd Owner	
5510	BMW 7 Series 730Ld	29.90 Lakh	79,000 kms	Diesel	Automatic	3rd Owner	
5511	BMW 5 Series 520d M Sport	31.90 Lakh	42,000 kms	Diesel	Automatic	2nd Owner	

```
# 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	manufa
0	jeep compass 2.0 longitude option bsiv	10.03 lakh	86,226 kms	diesel	manual	1st owner	

```
'''
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'
Column 'Seats' has following unique values: ['5 seats' '6 seats' '7 seats' '4 seats' '8 seats' '
```

5510	730ld	29.90 lakh	79,000 kms	diesel	automatic	3rd owner

```
'''
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 x, row in df.iterrows():
    for col in df.columns:
        if df.dtypes[col] == 'object':
            df.at[x, col] = row[col].split(' ', 1)[0].replace(',', '')
            if col == 'ownership':
                df.at[x, col] = re.sub('[^0-9.]', '', row[col])

df.head(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
0	jeep	10.03	86226	diesel	manual	1	2017
1	renault	12.83	13248	petrol	automatic	1	2021
2	toyota	16.40	60343	petrol	automatic	1	2016
3	honda	7.77	26696	petrol	automatic	1	2018
4	volkswagen	5.15	69414	petrol	manual	1	2016

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

car_name	car_prices_in_rupee	kms_driven	ownership	manufacture	engine	Seats
audi	22.678675	57651.054217	1.566265	2015.427711	1832.204819	5.301205
bentley	68.900000	44000.000000	1.000000	2013.000000	2967.000000	5.000000
bmw	36.020058	48188.656977	1.348837	2017.197674	1783.976744	5.220930
chevrolet	14366.975244	81264.609756	1.780488	2011.804878	1503.707317	5.219512
datsun	2.572400	49289.080000	1.480000	2016.880000	1327.520000	5.120000
fiat	3531.652941	70394.176471	1.352941	2012.235294	1479.705882	5.294118
force	9.200000	35000.000000	2.000000	2018.000000	1582.000000	7.000000
ford	2932.622857	74064.898810	1.380952	2014.470238	1511.648810	5.273810
honda	5.704960	65342.781124	1.407631	2014.839357	1474.787149	5.198795
hyundai	1494.281564	60672.173998	1.444770	2015.073314	1436.192571	5.223851
isuzu	11.030000	47250.000000	1.500000	2018.750000	1459.750000	5.000000
jaguar	40.401333	35721.500000	1.233333	2017.100000	2017.633333	5.100000
jeep	15.665714	51957.224490	1.183673	2018.510204	1525.734694	5.204082
kia	15.708361	24143.295082	1.065574	2020.409836	1468.918033	5.360656
land	30.031200	56761.520000	1.180000	2017.460000	1822.360000	5.360000
lexus	44.000000	77400.000000	1.000000	2018.200000	1637.000000	5.200000
mahindra	7.846508	75993.476190	1.428571	2015.850794	1589.993651	5.441270

```
'''
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. Expla
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-machi
    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
'''
```

```
'\nExplanation: \nOrdinal encoding is used for categorical columns that have ordering.\nOne-hot
orical columns that have no ordering.\n\nHere, One-hot encoding is used because the columns such
e', '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	car_nam
1	10.03	86226	1	2017	1956	5	0	
2	12.83	13248	1	2021	1330	5	0	
3	16.40	60343	1	2016	2494	5	0	
4	7.77	26696	1	2018	1199	5	0	
5	5.15	69414	1	2016	1199	5	0	

5 rows × 45 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
Seats                     0
car_name_audi             0
```

```
car_name_bentley          0
car_name_bmw              0
car_name_chevrolet        0
car_name_datsun           0
car_name_fiat             0
car_name_force            0
car_name_ford             0
car_name_honda            0
car_name_hyundai          0
car_name_isuzu            0
car_name_jaguar           0
car_name_jeep             0
car_name_kia              0
car_name_land             0
car_name_lexus            0
car_name_mahindra         0
car_name_maruti           0
car_name_maserati         0
car_name_mercedes-benz    0
car_name_mg               0
car_name_mini             0
car_name_mitsubishi       0
car_name_nissan           0
car_name_porsche          0
car_name_premier          0
car_name_renault          0
car_name_skoda            0
car_name_tata             0
car_name_toyota           0
car_name_volkswagen       0
car_name_volvo            0
fuel_type_cng             0
fuel_type_diesel          0
fuel_type_electric        0
fuel_type_lpg             0
fuel_type_petrol          0
transmission_automatic    0
transmission_manual       0
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 calculateMean(self,arr):
        return np.sum(arr)/len(arr)

    def calculateVariance(self,arr, mean):
        return np.sum((arr-mean)**2)

    def calculateCovariance(self,arr_x, mean_x, arr_y, mean_y):
        self.finalArr=0
        finalArr = (arr_x - mean_x)*(arr_y - mean_y)
        return np.sum(finalArr)

    def calculateCoefficients(self,x, y):
        self.m=0
        self.b=0
        self.x_mean = self.calculateMean(x)
        self.y_mean = self.calculateMean(y)
        self.m = self.calculateCovariance(x, self.x_mean, y, self.y_mean)/self.calculateVariance(x, se
        self.b = self.y_mean - self.x_mean*self.m
        return self.m, self.b
    def fit(self,X_train,y_train):
        self.m, self.b = self.calculateCoefficients(X_train, y_train)

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

```
# 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-sklea

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)))
```

```
Mean Squared Error: 121250234.44772844
Mean Absolute Error: 3227.6810986675123
R2 score: 0.002970080656014118
```

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.