UNIVERSITY SCHOOL OF

MANAGEMENT STUDIES

DATA MODELLING WITH PYTHON LAB

(MBA154)

Practical File

MBA ANALYTICS

Year 2023-2025

SUBMITTED BY:
NAME: **Anurag Durga**
ENROLLMENT NO: **00816619823**

SUBMITTED TO:
PROF. SANJAY DHINGRA

Operators and Data Types

In [1]:
```
8+35*25/5-(7|2) #usage of arithmetic and bitwise operators
```
Out[1]:
```
176.0
```

In [2]:
```
# String Data type
s_var = 'String'
print(s_var)
```
```
String
```

In [3]:
```
# List Data Structure
list_a = [1,2,'a','b']
print(list_a)
```
```
[1, 2, 'a', 'b']
```

In [4]:
```
# Indexing
list_a[0]
```
Out[4]:
```
1
```

In [5]:
```
list_a[2]
```
Out[5]:
```
'a'
```

In [7]:
```
# Tuple Data Structure
tuple_a = (3,4,'c','d')
print(tuple_a)
```
```
(3, 4, 'c', 'd')
```

In [8]:
```
tuple_a[1]
```
Out[8]:
```
4
```

In [9]:
```
from array import * # importing all functions from array module
```

In [10]:
```
array_a = array('i',[1,2,3,4]) #Array
```

In [11]:
```
for x in array_a: print(x) # For loop to print elements of an Array
```
```
1
2
3
4
```

In [24]:
```
# Dictionary Data type
dictionary_d = {'first': 1,
                'second': 2,
                'third': 3}
print(dictionary_d)
```
```
{'first': 1, 'second': 2, 'third': 3}
```

In [26]:
```
print(dictionary_d['first'])
```

```
1
```

In [29]:
```
# Set data type
set_a = {"example",5,6}
print(set("example"))
```

```
{'p', 'm', 'x', 'a', 'l', 'e'}
```

In [31]:
```
# range data type
range_r = range(1,12,4)
```

In [32]:
```
for x in range_r: print(x) # for loop to print elements of a range
```

```
1
5
9
```

Slicing and Concatenate

In [33]:
```
string_a = 'python'
```

In [34]:
```
string_a[slice(1,4,2)] # Slicing a string
```

Out[34]:
```
'yh'
```

In [35]:
```
string_a[1:4]
```

Out[35]:
```
'yth'
```

In [36]:
```
string_a[:]
```

Out[36]:
```
'python'
```

In [37]:
```
string_a = 'python' + ' ' + 'learning' # Concatenate string
```

In [38]:
```
print(string_a)
```

```
python learning
```

In [39]:
```
string_a *= 2
```

In [40]:
```
print(string_a)
```

```
python learningpython learning
```

Reading and Viewing the Data (gapminder-FiveYearData)

In [1]:
```
# Importing modules
import pandas as pd # pandas module for data manipulation
import numpy as np # numpy module for mathematical calculation
```

In [2]:
```
data1  =  pd.read_csv("C:/Users/NK/Desktop/Python/data/gapminder-FiveYearData.csv"
```

In [6]:
```
data1.head() # viewing top 5 rows of the data
```

Out[6]:

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8425333.0	Asia	28.801	779.445314
1	Afghanistan	1957	9240934.0	Asia	30.332	820.853030
2	Afghanistan	1962	10267083.0	Asia	31.997	853.100710
3	Afghanistan	1967	11537966.0	Asia	34.020	836.197138
4	Afghanistan	1972	13079460.0	Asia	36.088	739.981106

In [5]:
```
data1.tail() # viewing last 5 rows of the data
```

Out[5]:

	country	year	pop	continent	lifeExp	gdpPercap
1699	Zimbabwe	1987	9216418.0	Africa	62.351	706.157306
1700	Zimbabwe	1992	10704340.0	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948.0	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563.0	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143.0	Africa	43.487	469.709298

In [7]:
```
data1.shape # checking number of rows and columns in the data
```

Out[7]:
```
(1704, 6)
```

In [8]:
```
data1.columns # checking the names of the columns in the data with object data t
```

Out[8]:
```
Index(['country', 'year', 'pop', 'continent', 'lifeExp', 'gdpPercap'], dtype='o
bject')
```

In [9]:
```
data1.dtypes # checking the data type of the columns in the data
```

```
Out[9]: country       object
        year           int64
        pop          float64
        continent     object
        lifeExp      float64
        gdpPercap    float64
        dtype: object
```

```
In [10]: data1.info # viewing the informarmation of the data
```

```
Out[10]: <bound method DataFrame.info of           country  year         pop continent
         lifeExp   gdpPercap
         0     Afghanistan  1952   8425333.0      Asia   28.801  779.445314
         1     Afghanistan  1957   9240934.0      Asia   30.332  820.853030
         2     Afghanistan  1962  10267083.0      Asia   31.997  853.100710
         3     Afghanistan  1967  11537966.0      Asia   34.020  836.197138
         4     Afghanistan  1972  13079460.0      Asia   36.088  739.981106
         ...           ...   ...         ...       ...      ...         ...
         1699     Zimbabwe  1987   9216418.0    Africa   62.351  706.157306
         1700     Zimbabwe  1992  10704340.0    Africa   60.377  693.420786
         1701     Zimbabwe  1997  11404948.0    Africa   46.809  792.449960
         1702     Zimbabwe  2002  11926563.0    Africa   39.989  672.038623
         1703     Zimbabwe  2007  12311143.0    Africa   43.487  469.709298

         [1704 rows x 6 columns]>
```

```
In [11]: country_data1=data1['country']
```

```
In [12]: country_data1.head() # viewing top 5 rows of the country column
```

```
Out[12]: 0    Afghanistan
         1    Afghanistan
         2    Afghanistan
         3    Afghanistan
         4    Afghanistan
         Name: country, dtype: object
```

```
In [13]: country_data1.tail() # viewing bottom 5 rows of the country column
```

```
Out[13]: 1699    Zimbabwe
         1700    Zimbabwe
         1701    Zimbabwe
         1702    Zimbabwe
         1703    Zimbabwe
         Name: country, dtype: object
```

```
In [15]: subset=data1[['country','continent','year']] # assigning a variable to selected
```

```
In [16]: subset.head() # viewing top 5 rows of the selected columns in subset variable
```

Out[16]:

	country	continent	year
0	Afghanistan	Asia	1952
1	Afghanistan	Asia	1957
2	Afghanistan	Asia	1962
3	Afghanistan	Asia	1967
4	Afghanistan	Asia	1972

In [17]:
```
data1.loc[0] # viewing first row of the data
```

Out[17]:
```
country       Afghanistan
year                 1952
pop             8425333.0
continent            Asia
lifeExp            28.801
gdpPercap      779.445314
Name: 0, dtype: object
```

In [18]:
```
data1.loc[99] # viewing 100th row of the data
```

Out[18]:
```
country       Bangladesh
year                1967
pop           62821884.0
continent           Asia
lifeExp           43.453
gdpPercap     721.186086
Name: 99, dtype: object
```

In [19]:
```
data1.loc[[0,99,999]] # vewing first, 100th, 1000th row of the cata
```

Out[19]:

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8425333.0	Asia	28.801	779.445314
99	Bangladesh	1967	62821884.0	Asia	43.453	721.186086
999	Mongolia	1967	1149500.0	Asia	51.253	1226.041130

In [21]:
```
subset_a=data1.iloc[:,[2,5,-2]]
subset_a # viewing all rows of selected columns
```

Out[21]:

	pop	gdpPercap	lifeExp
0	8425333.0	779.445314	28.801
1	9240934.0	820.853030	30.332
2	10267083.0	853.100710	31.997
3	11537966.0	836.197138	34.020
4	13079460.0	739.981106	36.088
...	...	...	...
1699	9216418.0	706.157306	62.351
1700	10704340.0	693.420786	60.377
1701	11404948.0	792.449960	46.809
1702	11926563.0	672.038623	39.989
1703	12311143.0	469.709298	43.487

1704 rows × 3 columns

In [24]:

```
data1.iloc[42,0] # viewing entry in 43rd row and 1st column
```

Out[24]: 'Angola'

In [25]:

```
data1.iloc[[1,99,999],[1,3,5]]
```

Out[25]:

	year	continent	gdpPercap
1	1957	Asia	820.853030
99	1967	Asia	721.186086
999	1967	Asia	1226.041130

In [26]:

```
data1.loc[10:13,['country','lifeExp']]
```

Out[26]:

	country	lifeExp
10	Afghanistan	42.129
11	Afghanistan	43.828
12	Albania	55.230
13	Albania	59.280

Conditional Statements, Loops and Mathematical Operations

In [3]:
```
import numpy as np # importing numpy module for mathematical operations
```

In [6]:
```
# Using Conditional structures to assign a Grade to the number
marks = int(input())
if marks <= 50:
    print('D')
elif marks <= 60:
    print('C')
elif marks <= 70:
    print('B')
elif marks <= 80:
    print('A')
else:
    print('A+')
```

```
74
A
```

In [7]:
```
for i in range(1,11,1): print(i) # for Loop
```

```
1
2
3
4
5
6
7
8
9
10
```

In [8]:
```
# for loop to find the sim of first 50 natural numbers
sum = 0
for i in range (1,51,1):
    sum = sum + i
print(sum)
```

```
1275
```

In [9]:
```
# for loop to find the factorial of a given number
f = int(input())
fact = 1
for i in range (1, f+1):
    fact = fact*i
print(fact)
```

```
5
120
```

In [1]:
```
# Conditional structure and for loop to check if the given number is a prime num
num = int(input())
n=0
if num == 1:
    print(str(num) + " is not a Prime number")
```

```
    elif num>1:
        for i in range (2,num):
            if (num % i ==0 ):
                n=1
        if n==1:
            print(str(num)+" not a Prime number")
        else:
            print(str(num) + " is a Prime number")
```

```
47
47 is a Prime number
```

In [4]:
```
list1 = [1,2,3,4,5,6]
array_a  =  np.array(list1,dtype=int)
print(array_a)
```

```
[1 2 3 4 5 6]
```

In [5]:
```
print(type(array_a))
```

```
<class 'numpy.ndarray'>
```

In [7]:
```
print(len(array_a))
```

```
6
```

In [8]:
```
print(array_a.shape)
```

```
(6,)
```

In [9]:
```
array_a  =  array_a.reshape(3,2)
print(array_a)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

In [10]:
```
list2 = [1,2,3,4,5]
list3 = [2,3,4,5,6]
list4 = [7,8,9,10,11]
```

In [11]:
```
mularray = np.array([list2,list3,list4]) # making an array from multiple lists
print(mularray)
```

```
[[ 1  2  3  4  5]
 [ 2  3  4  5  6]
 [ 7  8  9 10 11]]
```

In [12]:
```
print(mularray.shape)
```

```
(3, 5)
```

In [13]:
```
x = [1,2,3,4]
print(x)
```

```
[1, 2, 3, 4]
```

In [14]:
```
y = [5,6,7,8]
print(y)
```

```
[5, 6, 7, 8]
```

In [15]:
```
print(np.sum(x+y)) # using numpy function to sum 2 lists
```

```
36
```

In [16]:
```
print(np.add(x,y)) # using numpy function to add 2 lists
```

```
[ 6  8 10 12]
```

In [17]:
```
print(np.subtract(x,y)) # using numpy function to subtract 2 lists
```

```
[-4 -4 -4 -4]
```

In [18]:
```
print(np.multiply(x,y)) # using numpy function to multiply 2 lists
```

```
[ 5 12 21 32]
```

In [19]:
```
print(np.divide(x,y)) # using numpy function to divide 2 lists
```

```
[0.2        0.33333333 0.42857143 0.5       ]
```

Analysing and Visualizing the Data (Toyota)

In [1]:
```
# Importing modules
import pandas as pd
import numpy as np
```

In [2]:
```
toyota = pd.read_csv("C:/Users/NK/Desktop/Python/data/Toyota.csv") # reading the
```

In [3]:
```
toyota
```

Out[3]:

	Unnamed: 0	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doo
0	0	13500	23.0	46986	Diesel	90	1.0	0	2000	thr
1	1	13750	23.0	72937	Diesel	90	1.0	0	2000	
2	2	13950	24.0	41711	Diesel	90	NaN	0	2000	
3	3	14950	26.0	48000	Diesel	90	0.0	0	2000	
4	4	13750	30.0	38500	Diesel	90	0.0	0	2000	
...	...	...	...	...	...	...	...	...	...	
1431	1431	7500	NaN	20544	Petrol	86	1.0	0	1300	
1432	1432	10845	72.0	??	Petrol	86	0.0	0	1300	
1433	1433	8500	NaN	17016	Petrol	86	0.0	0	1300	
1434	1434	7250	70.0	??	NaN	86	1.0	0	1300	
1435	1435	6950	76.0	1	Petrol	110	0.0	0	1600	

1436 rows × 11 columns

In [4]:
```
toyota.dtypes # checking the data types of the columns in the data
```

Out[4]:
```
Unnamed: 0      int64
Price           int64
Age           float64
KM             object
FuelType       object
HP             object
MetColor      float64
Automatic       int64
CC              int64
Doors          object
Weight          int64
dtype: object
```

In [5]:
```
toyota.select_dtypes(exclude=[object])
```

Out[5]:

	Unnamed: 0	Price	Age	MetColor	Automatic	CC	Weight
0	0	13500	23.0	1.0	0	2000	1165
1	1	13750	23.0	1.0	0	2000	1165
2	2	13950	24.0	NaN	0	2000	1165
3	3	14950	26.0	0.0	0	2000	1165
4	4	13750	30.0	0.0	0	2000	1170
...	...	...	...	...	...	...	...
1431	1431	7500	NaN	1.0	0	1300	1025
1432	1432	10845	72.0	0.0	0	1300	1015
1433	1433	8500	NaN	0.0	0	1300	1015
1434	1434	7250	70.0	1.0	0	1300	1015
1435	1435	6950	76.0	0.0	0	1600	1114

1436 rows × 7 columns

In [6]:
```
toyota.info() # checking information about the data
```

```
<class  'pandas.core.frame.DataFrame'>
RangeIndex: 1436 entries, 0 to 1435
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  1436 non-null   int64
 1   Price       1436 non-null   int64
 2   Age         1336 non-null   float64
 3   KM          1436 non-null   object
 4   FuelType    1336 non-null   object
 5   HP          1436 non-null   object
 6   MetColor    1286 non-null   float64
 7   Automatic   1436 non-null   int64
 8   CC          1436 non-null   int64
 9   Doors       1436 non-null   object
 10  Weight      1436 non-null   int64
dtypes:  float64(2),  int64(5),  object(4)
memory usage: 123.5+ KB
```

In [8]:
```
print(np.unique(toyota['KM'])) # checking unique values in the KM column in the
```

```
['1' '10000' '100123' ... '99865' '99971' '??']
```

In [9]:
```
print(np.unique(toyota['HP'])) # checking unique values in the HP column in the
```

```
['107' '110' '116' '192' '69' '71' '72' '73' '86' '90' '97' '98' '????']
```

In [10]:
```
print(np.unique(toyota['MetColor'])) # checking unique values in the MetColor co
```

```
[ 0.  1. nan]
```

In [11]:
```
toyota  =  pd.read_csv("C:/Users/NK/Desktop/Python/data/Toyota.csv",  index_col=0,n
```

In [12]:
```
toyota.info
```

```
Out[12]: <bound method DataFrame.info of        Price   Age       KM FuelType     HP  Met
         Color  Automatic    CC  Doors  \
         0     13500  23.0  46986.0   Diesel   90.0       1.0          0  2000  three
         1     13750  23.0  72937.0   Diesel   90.0       1.0          0  2000      3
         2     13950  24.0  41711.0   Diesel   90.0       NaN          0  2000      3
         3     14950  26.0  48000.0   Diesel   90.0       0.0          0  2000      3
         4     13750  30.0  38500.0   Diesel   90.0       0.0          0  2000      3
         ...     ...   ...      ...      ...    ...       ...        ...   ...    ...
         1431   7500   NaN  20544.0   Petrol   86.0       1.0          0  1300      3
         1432  10845  72.0      NaN   Petrol   86.0       0.0          0  1300      3
         1433   8500   NaN  17016.0   Petrol   86.0       0.0          0  1300      3
         1434   7250  70.0      NaN      NaN   86.0       1.0          0  1300      3
         1435   6950  76.0      1.0   Petrol  110.0       0.0          0  1600      5

               Weight
         0       1165
         1       1165
         2       1165
         3       1165
         4       1170
         ...      ...
         1431    1025
         1432    1015
         1433    1015
         1434    1015
         1435    1114

         [1436 rows x 10 columns]>
```

```
In [13]: toyota['FuelType'].nbytes
```

```
Out[13]: 11488
```

```
In [16]: toyota['MetColor'] = toyota['MetColor'].astype("object") # changing datatype of
         toyota.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Price      1436 non-null   int64
 1   Age        1336 non-null   float64
 2   KM         1421 non-null   float64
 3   FuelType   1336 non-null   object
 4   HP         1430 non-null   float64
 5   MetColor   1286 non-null   object
 6   Automatic  1436 non-null   int64
 7   CC         1436 non-null   int64
 8   Doors      1436 non-null   object
 9   Weight     1436 non-null   int64
dtypes: float64(3), int64(4), object(3)
memory usage: 123.4+ KB
```

```
In [17]: toyota['Automatic'] = toyota['Automatic'].astype("object") # changing datatype o
         toyota.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Price      1436 non-null   int64  
 1   Age        1336 non-null   float64
 2   KM         1421 non-null   float64
 3   FuelType   1336 non-null   object 
 4   HP         1430 non-null   float64
 5   MetColor   1286 non-null   object 
 6   Automatic  1436 non-null   object 
 7   CC         1436 non-null   int64  
 8   Doors      1436 non-null   object 
 9   Weight     1436 non-null   int64  
dtypes: float64(3), int64(3), object(4)
memory usage: 123.4+ KB
```

In [39]:
```python
toyota['FuelType'].astype('category').nbytes
```

Out[39]:
```
1444
```

In [40]:
```python
toyota['Doors'].replace('three',3,inplace = True) # Replacing values in the Door
toyota['Doors'].replace('four',4,inplace  =  True)
toyota['Doors'].replace('five',5,inplace  =  True)
toyota['Doors'] = toyota['Doors'].astype("int64") # changing datatype of Doors c
toyota.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1436 entries, 0 to 1435
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Price          1436 non-null   int64  
 1   Age            1336 non-null   float64
 2   KM             1421 non-null   float64
 3   FuelType       1436 non-null   int64  
 4   HP             1430 non-null   float64
 5   MetColor       1286 non-null   object 
 6   Automatic      1436 non-null   object 
 7   CC             1436 non-null   int64  
 8   Doors          1436 non-null   int64  
 9   age_converted  1336 non-null   float64
 10  Weight         1436 non-null   int64  
dtypes: float64(4), int64(5), object(2)
memory usage: 134.6+ KB
```

In [22]:
```python
toyota
```

Out[22]:

	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weig
0	13500	23.0	46986.0	Diesel	90.0	1.0	0	2000	3	11
1	13750	23.0	72937.0	Diesel	90.0	1.0	0	2000	3	11
2	13950	24.0	41711.0	Diesel	90.0	NaN	0	2000	3	11
3	14950	26.0	48000.0	Diesel	90.0	0.0	0	2000	3	11
4	13750	30.0	38500.0	Diesel	90.0	0.0	0	2000	3	11
...	...	...	...	...	...	...	...	...	...	
1431	7500	NaN	20544.0	Petrol	86.0	1.0	0	1300	3	10
1432	10845	72.0	NaN	Petrol	86.0	0.0	0	1300	3	10
1433	8500	NaN	17016.0	Petrol	86.0	0.0	0	1300	3	10
1434	7250	70.0	NaN	NaN	86.0	1.0	0	1300	3	10
1435	6950	76.0	1.0	Petrol	110.0	0.0	0	1600	5	11

1436 rows × 10 columns

In [23]:
```
toyota.insert(9,"age_converted",0) # inserting new column in the data
```

In [24]:
```
toyota
```

Out[24]:

	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	age_c
0	13500	23.0	46986.0	Diesel	90.0	1.0	0	2000	3	
1	13750	23.0	72937.0	Diesel	90.0	1.0	0	2000	3	
2	13950	24.0	41711.0	Diesel	90.0	NaN	0	2000	3	
3	14950	26.0	48000.0	Diesel	90.0	0.0	0	2000	3	
4	13750	30.0	38500.0	Diesel	90.0	0.0	0	2000	3	
...	...	...	...	...	...	...	...	...	...	
1431	7500	NaN	20544.0	Petrol	86.0	1.0	0	1300	3	
1432	10845	72.0	NaN	Petrol	86.0	0.0	0	1300	3	
1433	8500	NaN	17016.0	Petrol	86.0	0.0	0	1300	3	
1434	7250	70.0	NaN	NaN	86.0	1.0	0	1300	3	
1435	6950	76.0	1.0	Petrol	110.0	0.0	0	1600	5	

1436 rows × 11 columns

In [25]:
```
def  a_convert(val):
    val_converted = val/12
    return val_converted # funtion to convert the age given in months to age in
```

In [26]:
```
toyota["age_converted"] = a_convert(toyota["Age"])
toyota
```

Out[26]:

	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	age_c
0	13500	23.0	46986.0	Diesel	90.0	1.0	0	2000	3	
1	13750	23.0	72937.0	Diesel	90.0	1.0	0	2000	3	
2	13950	24.0	41711.0	Diesel	90.0	NaN	0	2000	3	
3	14950	26.0	48000.0	Diesel	90.0	0.0	0	2000	3	
4	13750	30.0	38500.0	Diesel	90.0	0.0	0	2000	3	
...	...	...	...	...	...	...	...	...	...	
1431	7500	NaN	20544.0	Petrol	86.0	1.0	0	1300	3	
1432	10845	72.0	NaN	Petrol	86.0	0.0	0	1300	3	
1433	8500	NaN	17016.0	Petrol	86.0	0.0	0	1300	3	
1434	7250	70.0	NaN	NaN	86.0	1.0	0	1300	3	
1435	6950	76.0	1.0	Petrol	110.0	0.0	0	1600	5	

1436 rows × 11 columns

In [27]:
```
toyota["age_converted"] = round(toyota["age_converted"],1)
toyota
```

Out[27]:

	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	age_c
0	13500	23.0	46986.0	Diesel	90.0	1.0	0	2000	3	
1	13750	23.0	72937.0	Diesel	90.0	1.0	0	2000	3	
2	13950	24.0	41711.0	Diesel	90.0	NaN	0	2000	3	
3	14950	26.0	48000.0	Diesel	90.0	0.0	0	2000	3	
4	13750	30.0	38500.0	Diesel	90.0	0.0	0	2000	3	
...	...	...	...	...	...	...	...	...	...	
1431	7500	NaN	20544.0	Petrol	86.0	1.0	0	1300	3	
1432	10845	72.0	NaN	Petrol	86.0	0.0	0	1300	3	
1433	8500	NaN	17016.0	Petrol	86.0	0.0	0	1300	3	
1434	7250	70.0	NaN	NaN	86.0	1.0	0	1300	3	
1435	6950	76.0	1.0	Petrol	110.0	0.0	0	1600	5	

1436 rows × 11 columns

In [28]:
```
toyota1 = toyota.copy() # copying the data
```

In [29]:
```
pd.crosstab(index = toyota1['FuelType'], columns = 'count',dropna=True) # checki
```

Out[29]:

col_0	count
FuelType	
CNG	15
Diesel	144
Petrol	1177

In [30]:
```
pd.crosstab(index = toyota1['Automatic'], columns = toyota1['FuelType'],dropna=T
```

Out[30]:

FuelType	CNG	Diesel	Petrol
Automatic			
0	15	144	1104
1	0	0	73

In [31]:
```
pd.crosstab(index = toyota1['Automatic'], columns = toyota1['FuelType'], margins
```

Out[31]:

FuelType	CNG	Diesel	Petrol	All
Automatic				
0	15	144	1104	1263
1	0	0	73	73
All	15	144	1177	1336

In [32]:
```
pd.crosstab(index = toyota1['Automatic'], columns = toyota1['FuelType'], normali
```

Out[32]:

FuelType	CNG	Diesel	Petrol	All
Automatic				
0	0.011228	0.107784	0.826347	0.945359
1	0.000000	0.000000	0.054641	0.054641
All	0.011228	0.107784	0.880988	1.000000

In [33]:
```
pd.crosstab(index = toyota1['Automatic'], columns = toyota1['FuelType'], normali
```

Out[33]:

FuelType	CNG	Diesel	Petrol
Automatic			
0	0.011228	0.107784	0.826347
1	0.000000	0.000000	0.054641

In [34]:
```
pd.crosstab(index = toyota1['Automatic'], columns = toyota1['FuelType'], normali
```

Out[34]:

FuelType	CNG	Diesel	Petrol
Automatic			
0	0.011876	0.114014	0.874109
1	0.000000	0.000000	1.000000
All	0.011228	0.107784	0.880988

In [36]:
```
pd.crosstab(index = toyota1['Automatic'], columns = toyota1['FuelType'], normali
```

Out[36]:

FuelType	CNG	Diesel	Petrol	All
Automatic				
0	1.0	1.0	0.937978	0.945359
1	0.0	0.0	0.062022	0.054641

In [44]:
```
toyota['MetColor']=toyota['MetColor'].astype("object")
toyota['Automatic']=toyota['Automatic'].astype("object")
numdata = toyota.select_dtypes(exclude = ['object'])
corrl = numdata.corr()
print(corrl) # correlation between the values in the columns selected
```

```
                 Price       Age        KM  FuelType        HP        CC  \
Price         1.000000 -0.878407 -0.574720       NaN  0.309902  0.165067
Age          -0.878407  1.000000  0.512735       NaN -0.157904 -0.120706
KM           -0.574720  0.512735  1.000000       NaN -0.335285  0.299993
FuelType           NaN       NaN       NaN       NaN       NaN       NaN
HP            0.309902 -0.157904 -0.335285       NaN  1.000000  0.053758
CC            0.165067 -0.120706  0.299993       NaN  0.053758  1.000000
Doors         0.185326 -0.157027 -0.036191       NaN  0.097162  0.126768
age_converted -0.878062  0.999826  0.512502       NaN -0.157655 -0.120717
Weight        0.581198 -0.464299 -0.026271       NaN  0.086737  0.651450

                 Doors  age_converted    Weight
Price         0.185326      -0.878062  0.581198
Age          -0.157027       0.999826 -0.464299
KM           -0.036191       0.512502 -0.026271
FuelType           NaN            NaN       NaN
HP            0.097162      -0.157655  0.086737
CC            0.126768      -0.120717  0.651450
Doors         1.000000      -0.156914  0.302618
age_converted -0.156914       1.000000 -0.464600
Weight        0.302618      -0.464600  1.000000
```

In [45]:
```
toyota['Price'].corr(toyota['Age'])
```

Out[45]:
```
-0.878407409362202
```

In [46]:
```
import matplotlib.pyplot as plt # importing matplot module for data visualizatio
```

In [66]:
```
toyota1.dropna(axis=0,  inplace=True)
```

In [67]:
```
toyota1
```

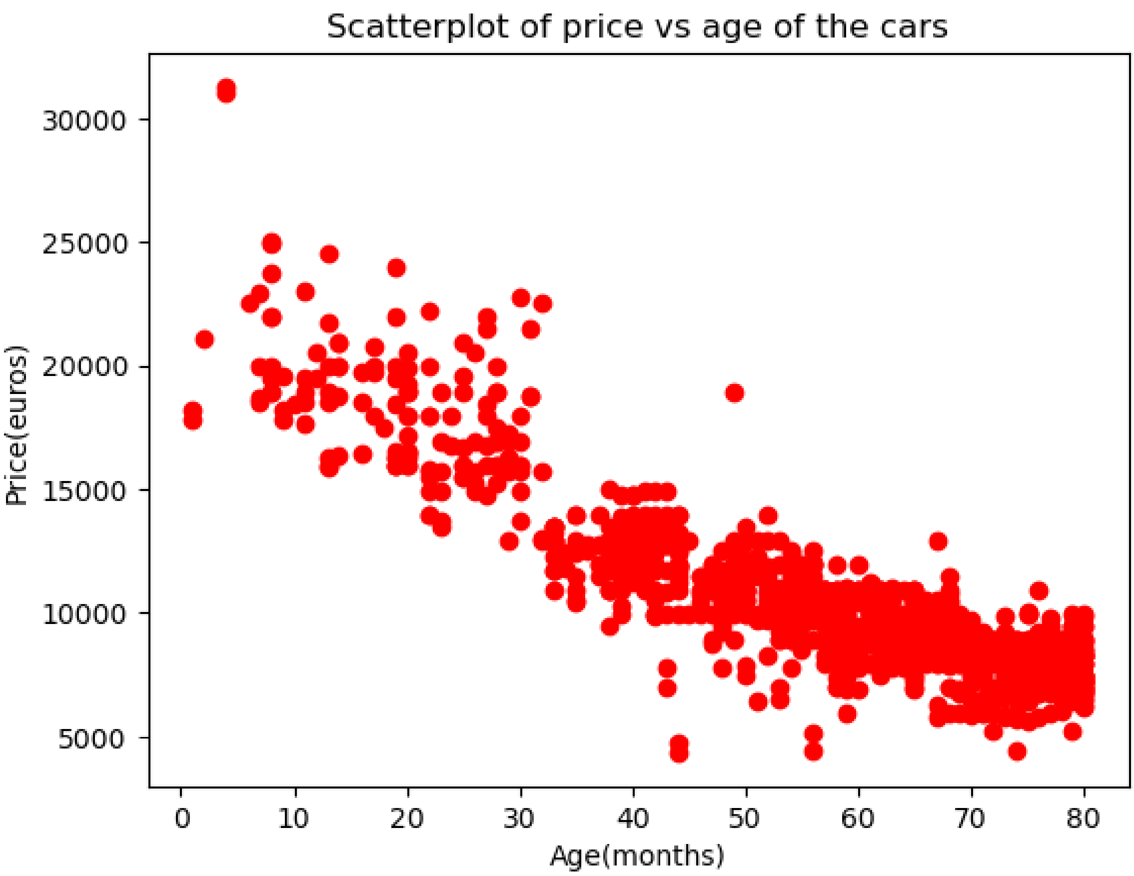
Out[67]:

	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	age_c
0	13500	23.0	46986.0	Diesel	90.0	1.0	0	2000	3	
1	13750	23.0	72937.0	Diesel	90.0	1.0	0	2000	3	
3	14950	26.0	48000.0	Diesel	90.0	0.0	0	2000	3	
4	13750	30.0	38500.0	Diesel	90.0	0.0	0	2000	3	
5	12950	32.0	61000.0	Diesel	90.0	0.0	0	2000	3	
...	...	...	...	...	...	...	...	...	...	
1423	7950	80.0	35821.0	Petrol	86.0	0.0	1	1300	3	
1424	7750	73.0	34717.0	Petrol	86.0	0.0	0	1300	3	
1429	8950	78.0	24000.0	Petrol	86.0	1.0	1	1300	5	
1430	8450	80.0	23000.0	Petrol	86.0	0.0	0	1300	3	
1435	6950	76.0	1.0	Petrol	110.0	0.0	0	1600	5	

1096 rows × 11 columns

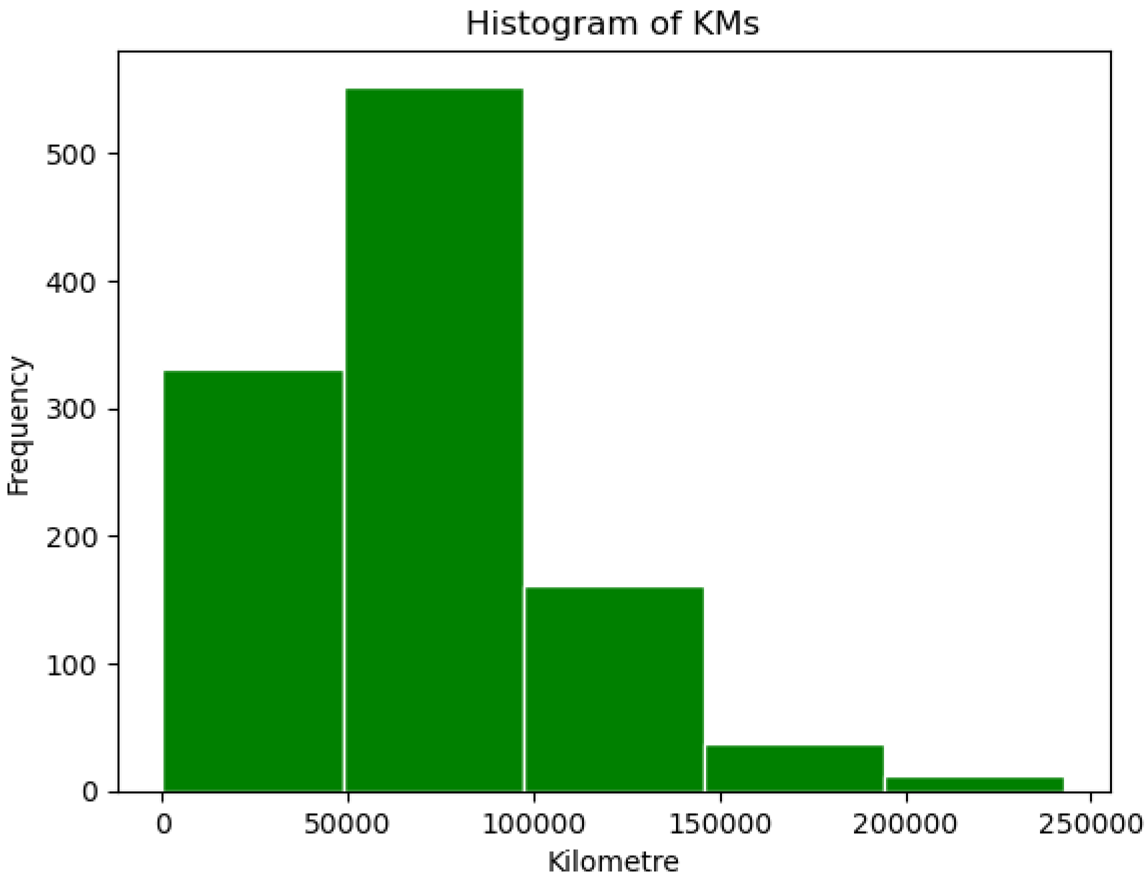
In [68]:

```
plt.scatter(toyota1['Age'], toyota1['Price'], c='red')
plt.title('Scatterplot of price vs age of the cars')
plt.xlabel('Age(months)')
plt.ylabel('Price(euros)')
plt.show() # Scatterplot of age and price column in the data
```



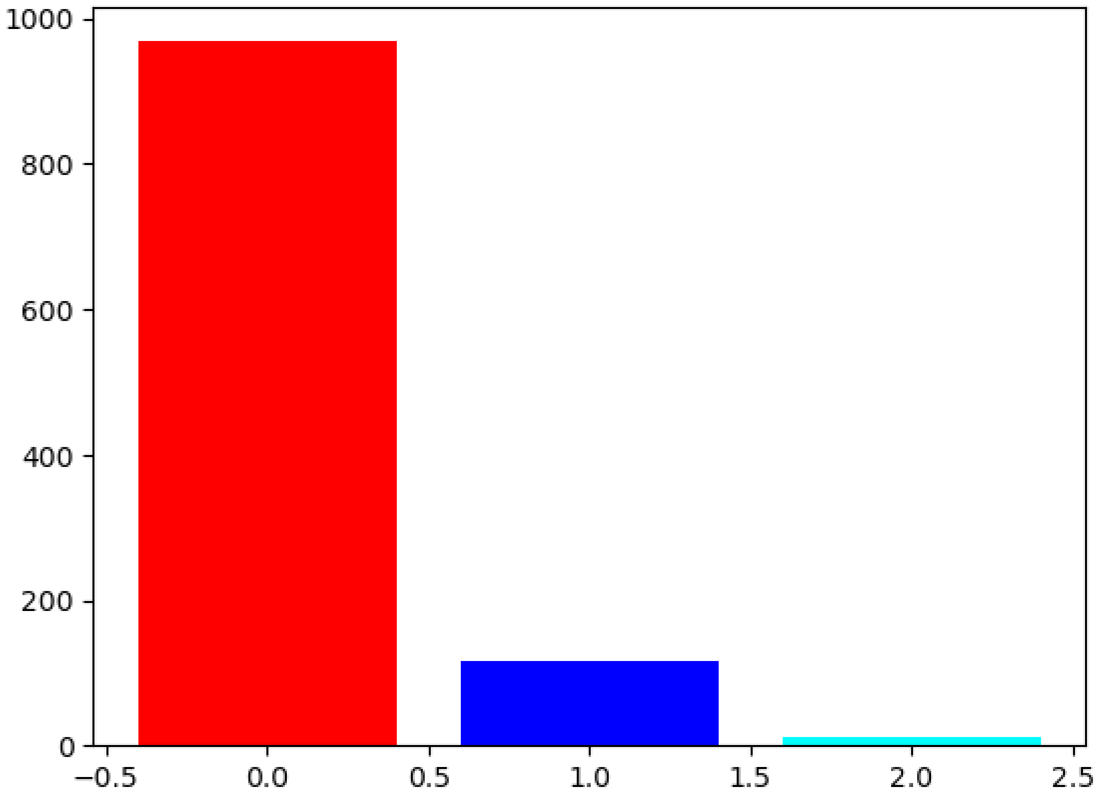
In [69]:

```
plt.hist(toyota1['KM'])
plt.hist(toyota1['KM'], color = 'green', edgecolor = 'white', bins=5)
plt.title('Histogram of KMs')
plt.xlabel('Kilometre')
plt.ylabel('Frequency')
plt.show() # Histogram of the KM column in the data
```



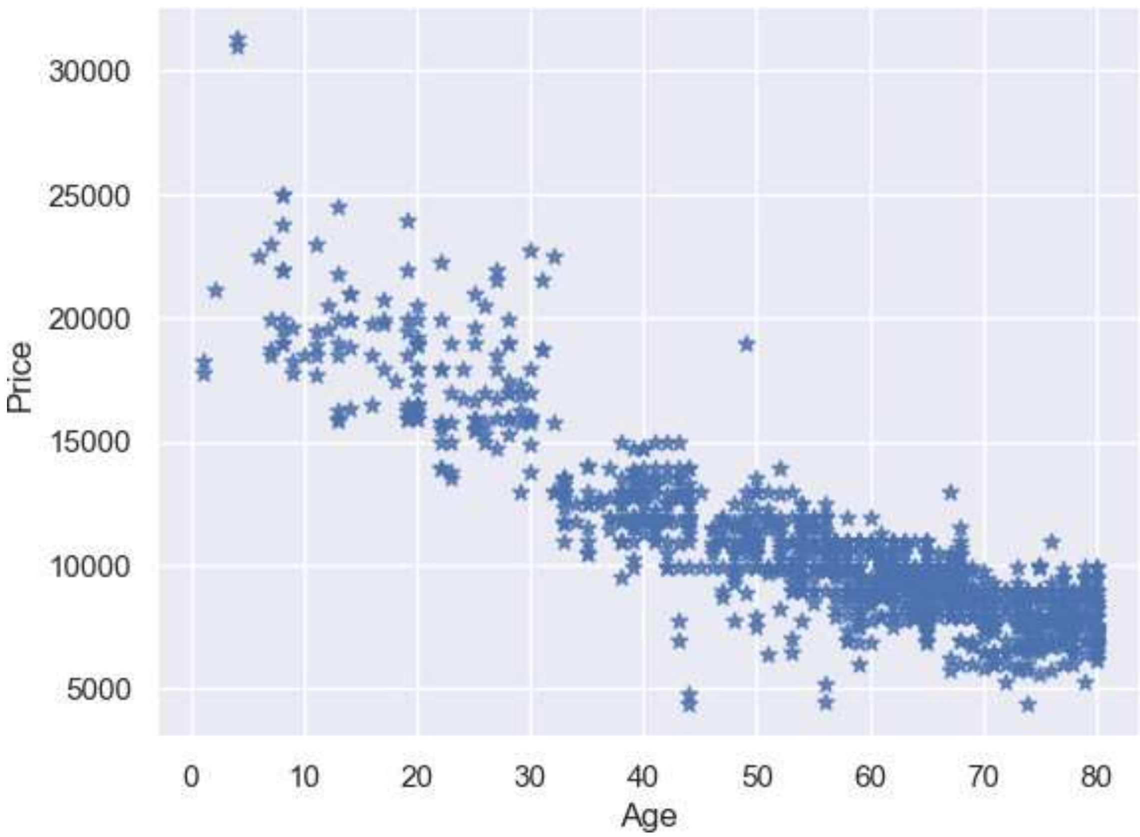
In [70]:

```
counts  =  toyota1['FuelType'].value_counts()
fueltype = ('Petrol','Diesel','CNG')
index  =  np.arange(len(fueltype))
plt.bar(index,  counts,  color  =  ['red','blue','cyan'])
plt.show() # bar graph of the FuelType column in the data
```



In [71]:
```
import seaborn as sns
```

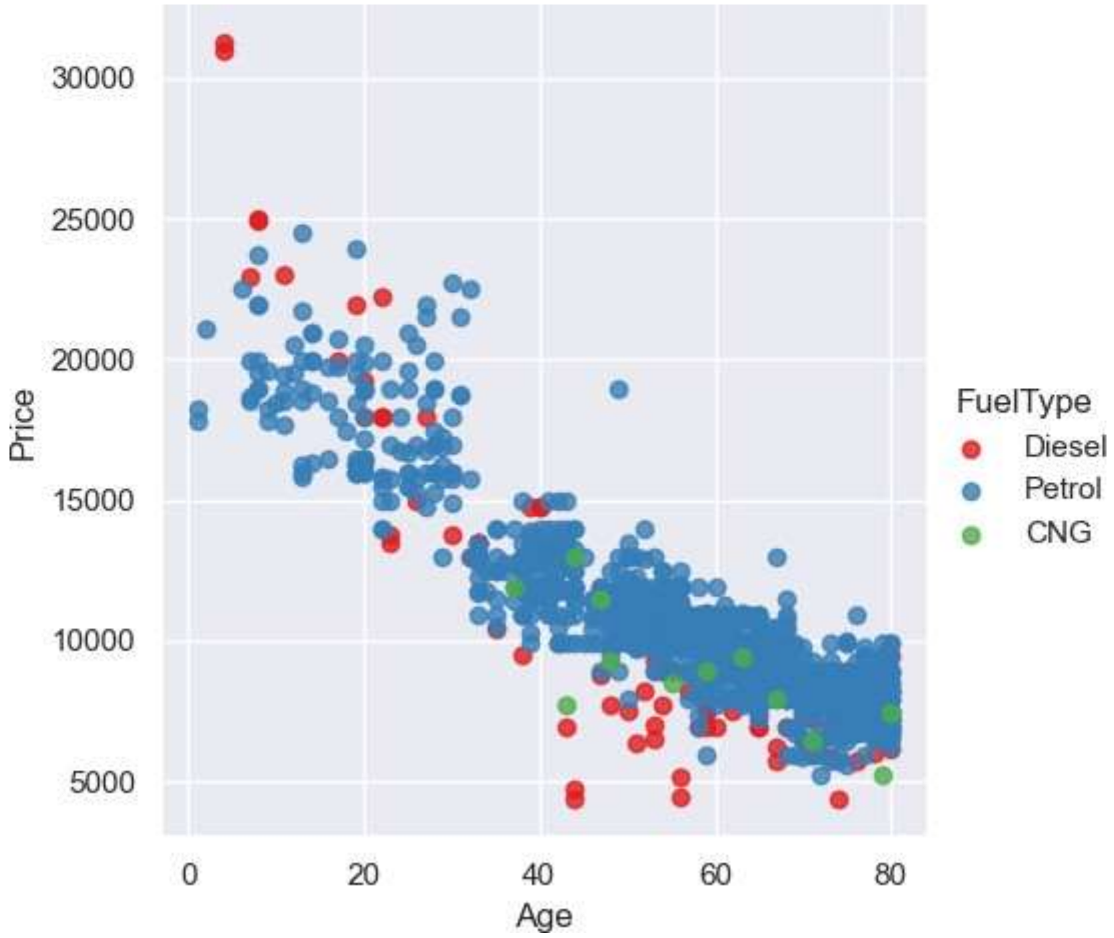
In [72]:
```
sns.set(style = 'darkgrid')
sns.regplot(x = toyota1['Age'], y = toyota1['Price'], fit_reg = False, marker =
plt.show() # Regression plot for the age and price column in the data
```



In [74]:
```
sns.set(style = 'darkgrid')
sns.lmplot(x = 'Age', y = 'Price', data=toyota1, fit_reg= False, hue = "FuelType
```

```
plt.show() # linear model plot for fueltype column in the data
```

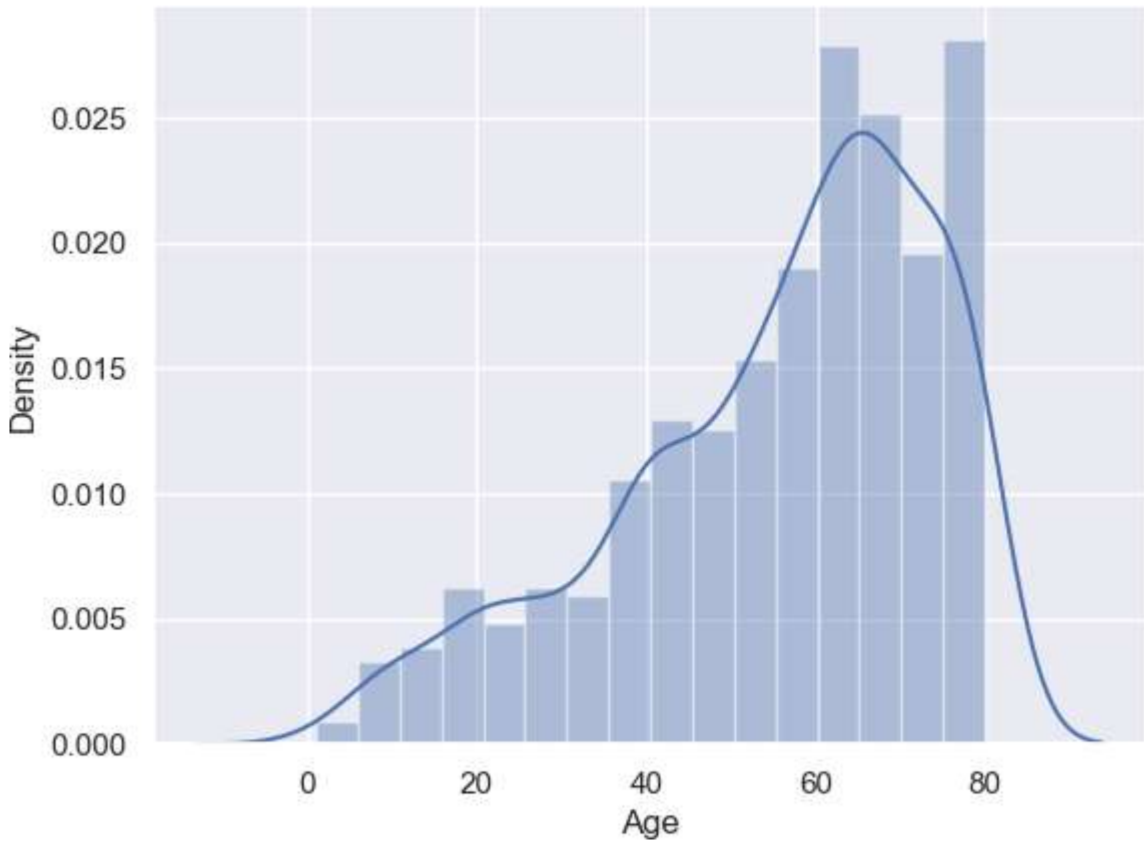
```
D:\Anaconda\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure la
yout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [76]:
```
sns.distplot(toyota1['Age'])
```

```
C:\Users\NK\AppData\Local\Temp\ipykernel_7036\3503570373.py:1:   UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(toyota1['Age'])
```

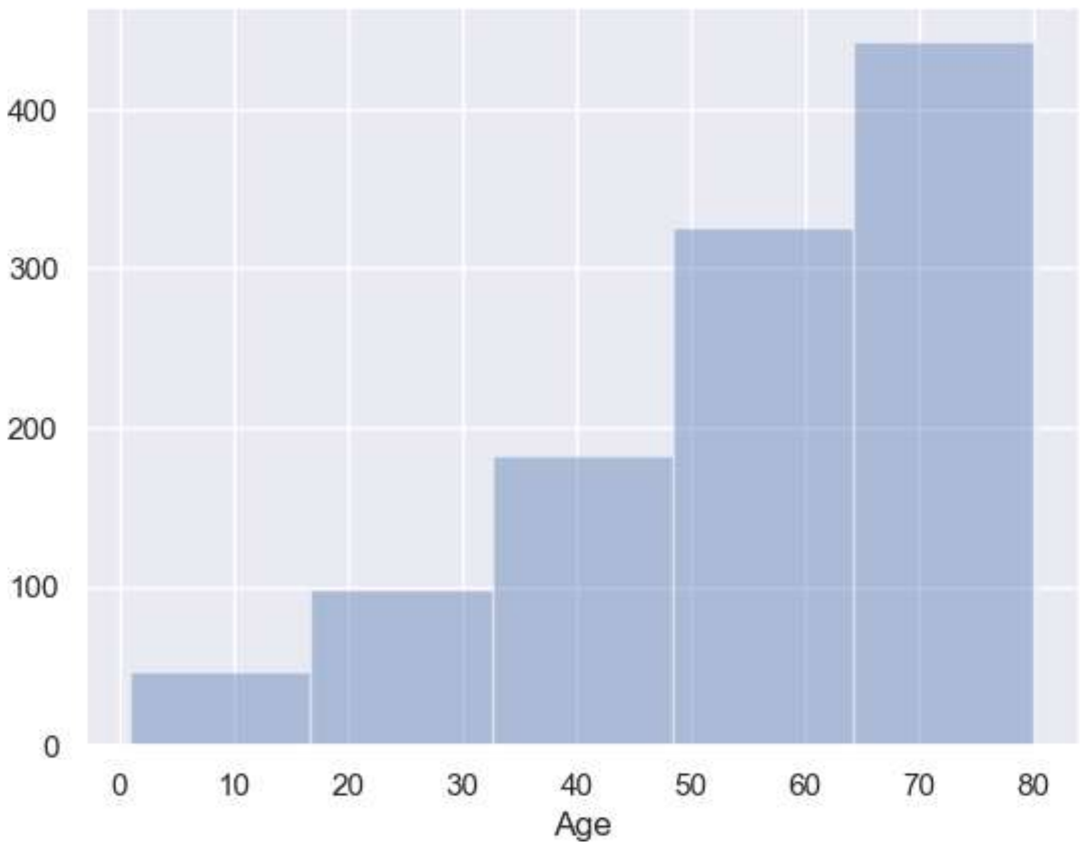
Out[76]:
```
<Axes: xlabel='Age', ylabel='Density'>
```



In [77]:
```
sns.distplot(toyota1['Age'], kde = False, bins = 5)
```

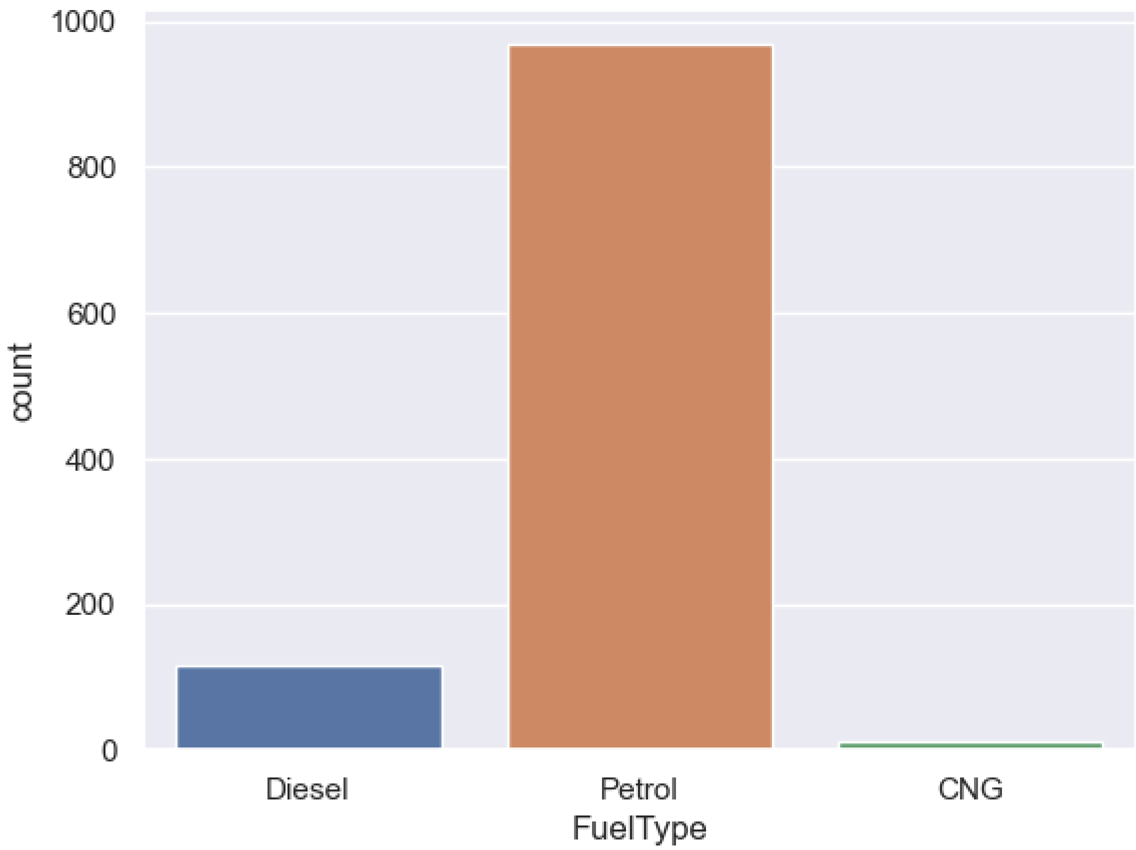
```
C:\Users\NK\AppData\Local\Temp\ipykernel_7036\2183260027.py:1:   UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(toyota1['Age'], kde = False, bins = 5)
```

Out[77]:
```
<Axes: xlabel='Age'>
```



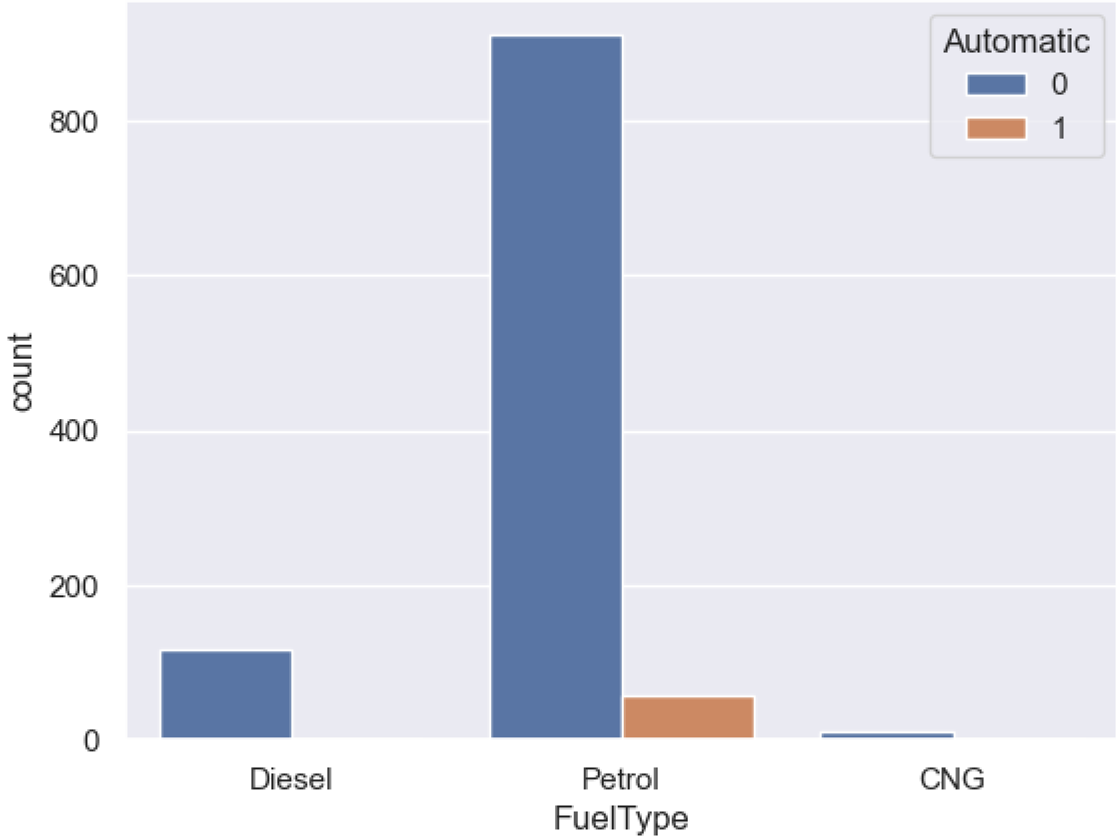
In [78]:
```
sns.countplot(x = 'FuelType', data = toyota1) # Count plot
```

Out[78]: <Axes: xlabel='FuelType', ylabel='count'>



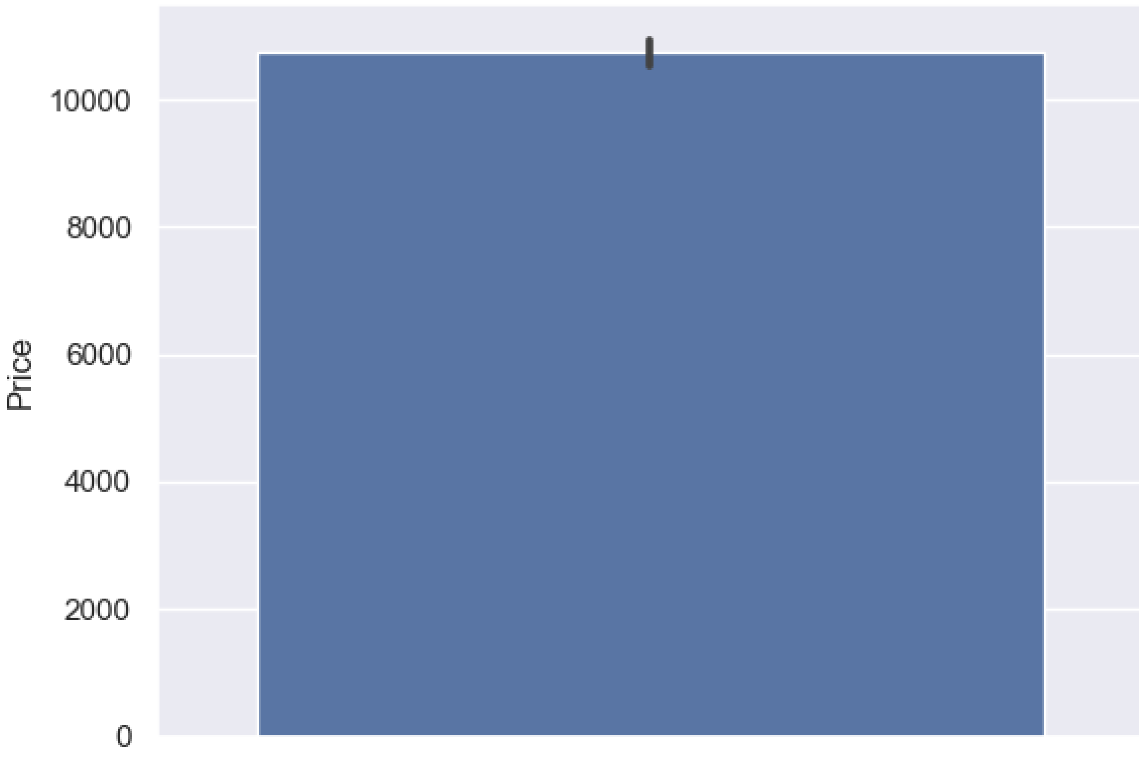
In [79]:
```
sns.countplot(x = 'FuelType', data = toyota1, hue = 'Automatic')
```

```
Out[79]: <Axes: xlabel='FuelType', ylabel='count'>
```



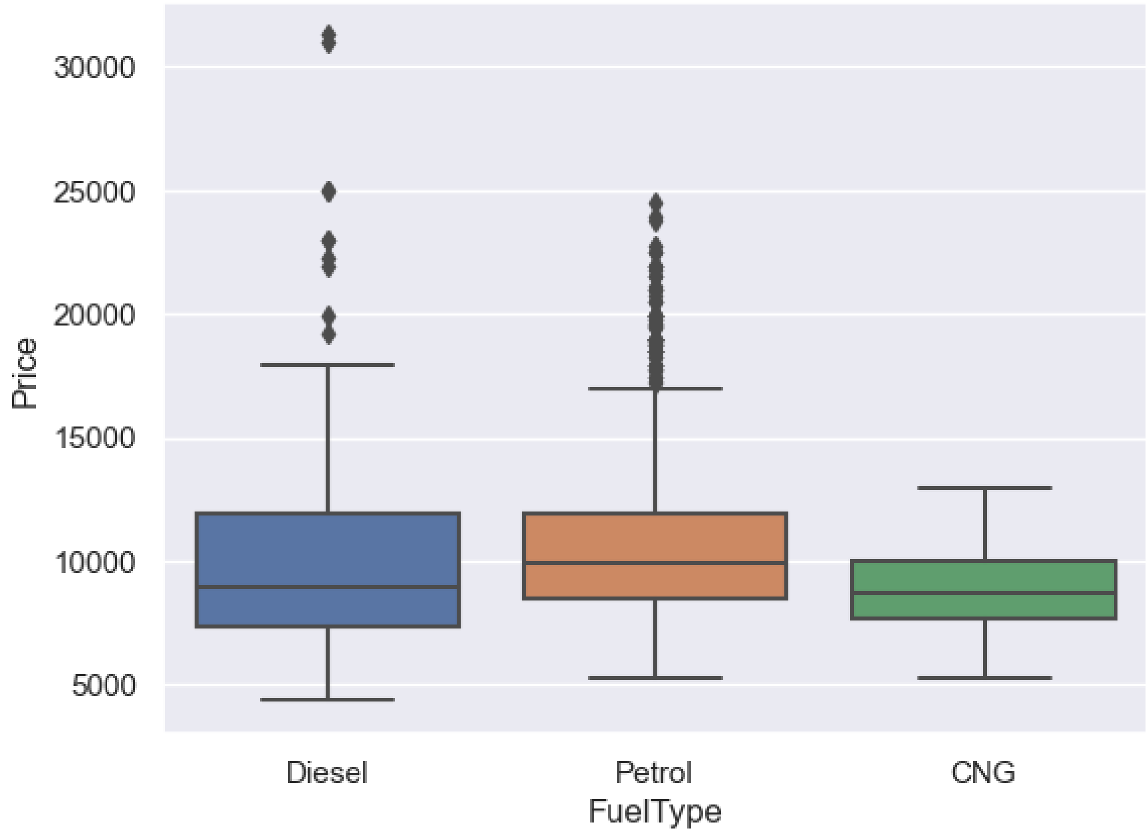

```
In [80]: sns.barplot(y = toyota1['Price']) # Bar plot of price column in the data
```

```
Out[80]: <Axes: ylabel='Price'>
```



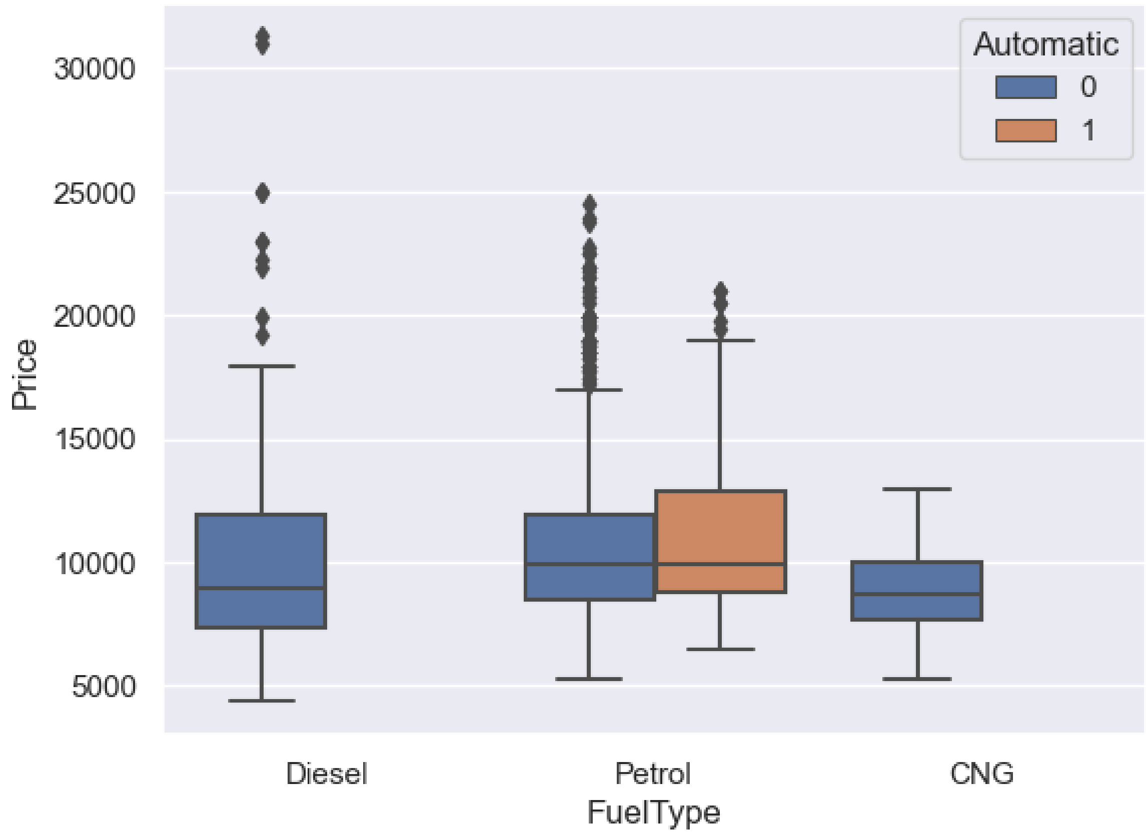

```
In [81]: sns.boxplot(x = toyota1['FuelType'], y = toyota1['Price'])
```

```
Out[81]: <Axes: xlabel='FuelType', ylabel='Price'>
```



In [82]:
```
sns.boxplot(x = 'FuelType', y = 'Price', hue = 'Automatic', data = toyota1)
```

Out[82]:
```
<Axes: xlabel='FuelType', ylabel='Price'>
```



In [86]:
```
toyota2 = pd.read_csv('C:/Users/NK/Desktop/Python/data/Toyota.csv', index_col=0,
```

In [87]:
```
toyota2.isnull().sum()
```

Out[87]:
```
Price          0
Age          100
KM            15
FuelType     100
HP             6
MetColor     150
Automatic      0
CC             0
Doors          0
Weight         0
dtype: int64
```

In [88]:
```
toyota2.describe() # Basic Statistics of the data
```

Out[88]:

	Price	Age	KM	HP	MetColor	Automatic
count	1436.000000	1336.000000	1421.000000	1430.000000	1286.000000	1436.00000C
mean	10730.824513	55.672156	68647.239972	101.478322	0.674961	0.05571C
std	3626.964585	18.589804	37333.023589	14.768255	0.468572	0.229441
min	4350.000000	1.000000	1.000000	69.000000	0.000000	0.00000C
25%	8450.000000	43.000000	43210.000000	90.000000	0.000000	0.00000C
50%	9900.000000	60.000000	63634.000000	110.000000	1.000000	0.00000C
75%	11950.000000	70.000000	87000.000000	110.000000	1.000000	0.00000C
max	32500.000000	80.000000	243000.000000	192.000000	1.000000	1.00000C

In [89]:
```
toyota2['Age'].fillna(toyota2['Age'].mean(),inplace=True) # Filling missing valu
```

In [90]:
```
toyota2['KM'].fillna(toyota2['KM'].median(),inplace=True) # Filling missing valu
```

In [91]:
```
toyota2['FuelType'].value_counts()
```

Out[91]:
```
FuelType
Petrol    1177
Diesel     144
CNG         15
Name: count, dtype: int64
```

In [94]:
```
toyota2['FuelType'].fillna(toyota2['FuelType'].value_counts().index[0],inplace=T
```

In [95]:
```
toyota2['HP'].fillna(toyota2['HP'].median(),inplace=True) # Filling missing valu
```

In [96]:
```
toyota2['MetColor'].fillna(toyota2['MetColor'].median(),inplace=True) # Filling
```

In [97]:
```
toyota2.isnull().sum() # checking null values in the data
```

Out[97]:
```
Price        0
Age          0
KM           0
FuelType     0
HP           0
MetColor     0
Automatic    0
CC           0
Doors        0
Weight       0
dtype: int64
```

In [98]:
```
toyota2['FuelType'].value_counts()
```

Out[98]:
```
FuelType
Petrol    1277
Diesel     144
CNG         15
Name: count, dtype: int64
```

Analysing and Visualizing data (Income)

In [2]:
```
# importing modules
import pandas as pd
import numpy as np
import seaborn as sns # seaborn module for data visualization
```

In [3]:
```
dincome = pd.read_csv("C:/Users/NK/Desktop/Python/data/income.csv") # reading th
```

In [4]:
```
dincome.head() # viewing the data
```

Out[4]:

	age	JobType	EdType	maritalstatus	occupation	relationship	race	gender	capit
0	45	Private	HS-grad	Divorced	Adm-clerical	Not-in-family	White	Female	
1	24	Federal-gov	HS-grad	Never-married	Armed-Forces	Own-child	White	Male	
2	44	Private	Some-college	Married-civ-spouse	Prof-specialty	Husband	White	Male	
3	27	Private	9th	Never-married	Craft-repair	Other-relative	White	Male	
4	20	Private	Some-college	Never-married	Sales	Not-in-family	White	Male	

In [5]:
```
data = dincome.copy() # copying the data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31978 entries, 0 to 31977
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   age            31978 non-null  int64 
 1   JobType        31978 non-null  object
 2   EdType         31978 non-null  object
 3   maritalstatus  31978 non-null  object
 4   occupation     31978 non-null  object
 5   relationship   31978 non-null  object
 6   race           31978 non-null  object
 7   gender         31978 non-null  object
 8   capitalgain    31978 non-null  int64 
 9   capitalloss    31978 non-null  int64 
 10  hoursperweek   31978 non-null  int64 
 11  nativecountry  31978 non-null  object
 12  SalStat        31978 non-null  object
dtypes: int64(4), object(9)
memory usage: 3.2+ MB
```

In [6]:
```
data.isnull().sum() # checking null values in the data
```

Out[6]:
```
age              0
JobType          0
EdType           0
maritalstatus    0
occupation       0
relationship     0
race             0
gender           0
capitalgain      0
capitalloss      0
hoursperweek     0
nativecountry    0
SalStat          0
dtype: int64
```

In [7]:
```
data.describe # basic statistics of the data
```

```
Out[7]: <bound method NDFrame.describe of        age       JobType         EdType
maritalstatus  \
0       45       Private        HS-grad           Divorced
1       24    Federal-gov        HS-grad      Never-married
2       44       Private   Some-college  Married-civ-spouse
3       27       Private            9th      Never-married
4       20       Private   Some-college      Never-married
...    ...           ...            ...                ...
31973   34     Local-gov        HS-grad      Never-married
31974   34     Local-gov   Some-college      Never-married
31975   23       Private   Some-college  Married-civ-spouse
31976   42     Local-gov   Some-college  Married-civ-spouse
31977   29       Private      Bachelors      Never-married

              occupation     relationship    race   gender  capitalgain  \
0           Adm-clerical    Not-in-family   White   Female            0
1           Armed-Forces        Own-child   White     Male            0
2         Prof-specialty          Husband   White     Male            0
3           Craft-repair   Other-relative   White     Male            0
4                  Sales    Not-in-family   White     Male            0
...                  ...              ...     ...      ...          ...
31973    Farming-fishing    Not-in-family   Black     Male          594
31974    Protective-serv    Not-in-family   White   Female            0
31975       Adm-clerical          Husband   White     Male            0
31976       Adm-clerical             Wife   White   Female            0
31977     Prof-specialty    Not-in-family   White     Male            0

       capitalloss  hoursperweek  nativecountry  \
0                0            28  United-States
1                0            40  United-States
2                0            40  United-States
3                0            40         Mexico
4                0            35  United-States
...            ...           ...            ...
31973            0            60  United-States
31974            0            40  United-States
31975            0            40  United-States
31976            0            40  United-States
31977            0            40  United-States

                             SalStat
0       less than or equal to 50,000
1       less than or equal to 50,000
2                greater than 50,000
3       less than or equal to 50,000
4       less than or equal to 50,000
...                              ...
31973   less than or equal to 50,000
31974   less than or equal to 50,000
31975   less than or equal to 50,000
31976   less than or equal to 50,000
31977   less than or equal to 50,000

[31978 rows x 13 columns]>
```

```
In [8]: cat_desc = data.describe(include=['object'])
        cat_desc
```

Out[8]:

	JobType	EdType	maritalstatus	occupation	relationship	race	gender	nativ
count	31978	31978	31978	31978	31978	31978	31978	
unique	9	16	7	15	6	5	2	
top	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	Unit
freq	22286	10368	14692	4038	12947	27430	21370	

In [9]:
```
data['JobType'].value_counts()
```

Out[9]:
```
JobType
 Private             22286
 Self-emp-not-inc     2499
 Local-gov            2067
 ?                    1809
 State-gov            1279
 Self-emp-inc         1074
 Federal-gov           943
 Without-pay            14
 Never-worked            7
Name: count, dtype: int64
```

In [10]:
```
data['occupation'].value_counts()
```

Out[10]:
```
occupation
 Prof-specialty       4038
 Craft-repair         4030
 Exec-managerial      3992
 Adm-clerical         3721
 Sales                3584
 Other-service        3212
 Machine-op-inspct    1966
 ?                    1816
 Transport-moving     1572
 Handlers-cleaners    1350
 Farming-fishing       989
 Tech-support          912
 Protective-serv       644
 Priv-house-serv       143
 Armed-Forces            9
Name: count, dtype: int64
```

In [11]:
```
missing = data[data.isnull().any(axis=1)]
```

In [12]:
```
data2 = data.dropna(axis=0)
```

In [14]:
```
pd.crosstab(index=data2['gender'],columns='count',normalize=True)
```

Out[14]:

col_0	count
gender	
Female	0.331728
Male	0.668272

In [15]:
```
pd.crosstab(index=data['gender'],columns=data2['SalStat'],margins=True,normalize
```

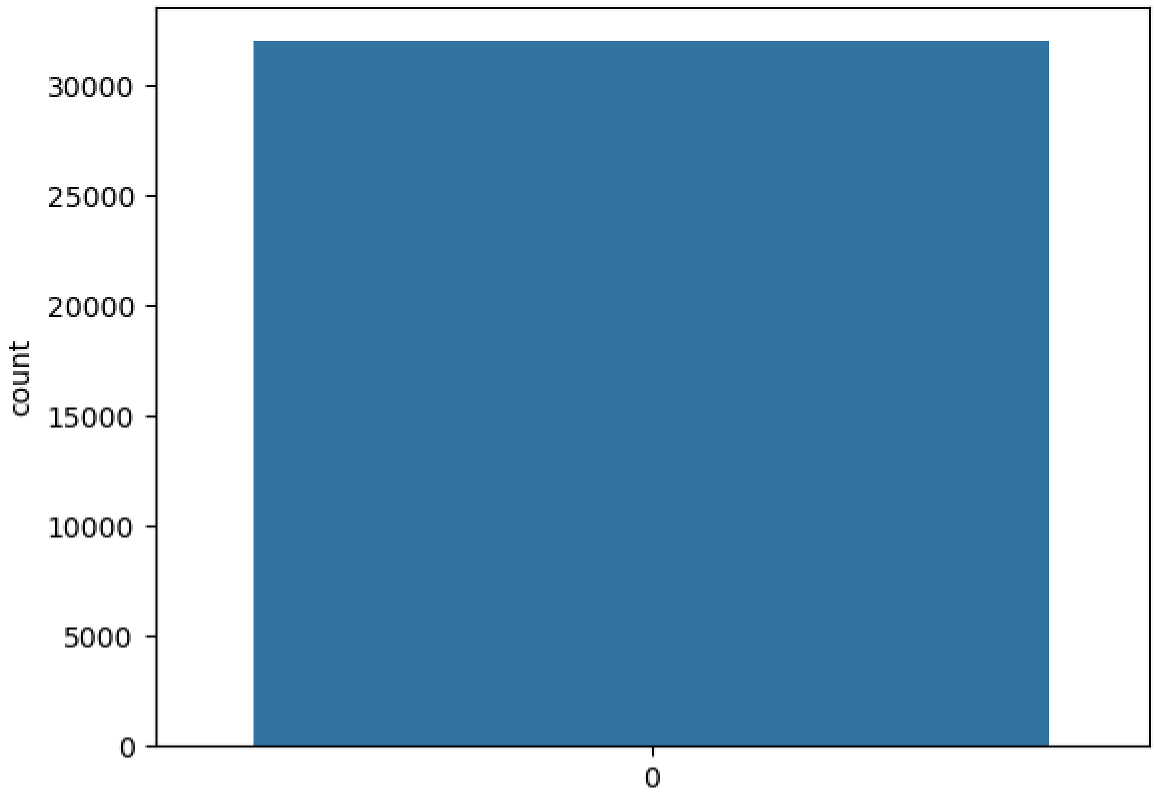
Out[15]:

SalStat	greater than 50,000	less than or equal to 50,000
gender		
Female	0.109540	0.890460
Male	0.305709	0.694291
All	0.240634	0.759366

In [16]:
```
data2['SalStat']=data2['SalStat'].replace({' less than or equal to 50,000':0,' g
```

In [17]:
```
sns.countplot(data2['SalStat']) # count plot of the SalStat column
```

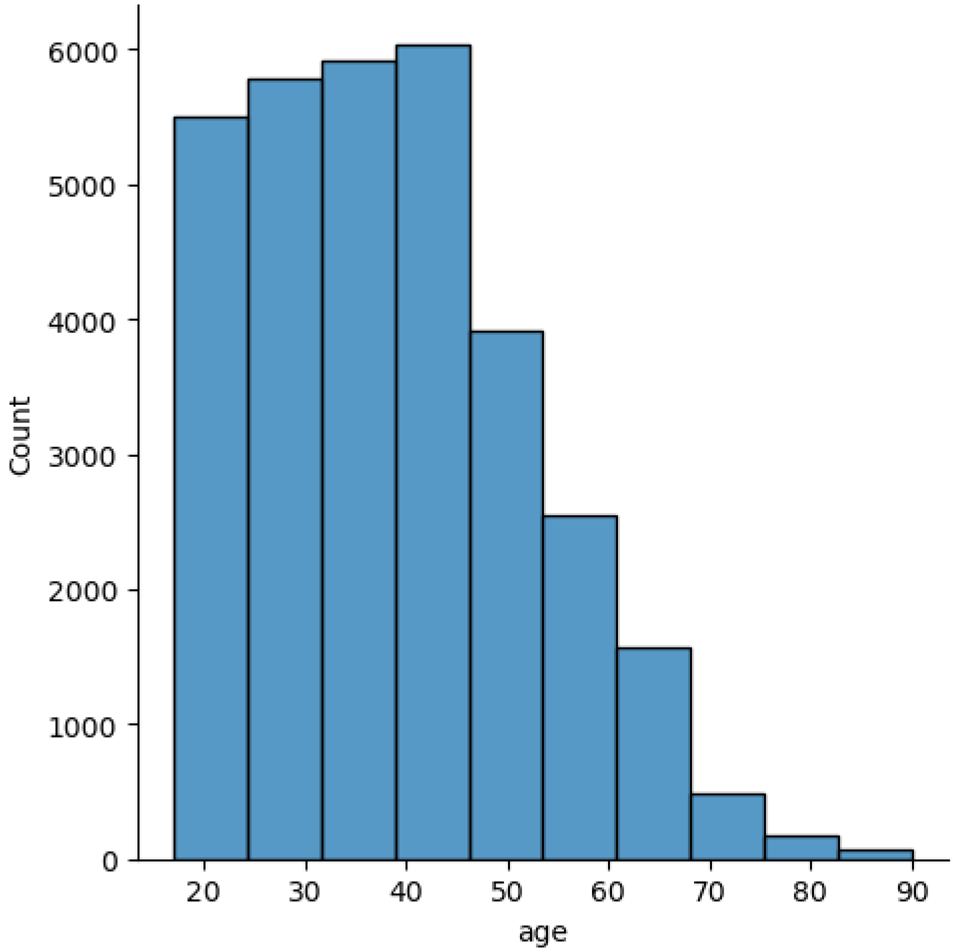
Out[17]: <Axes: ylabel='count'>



In [19]:
```
sns.displot(data2['age'],bins=10, kde= False)
```

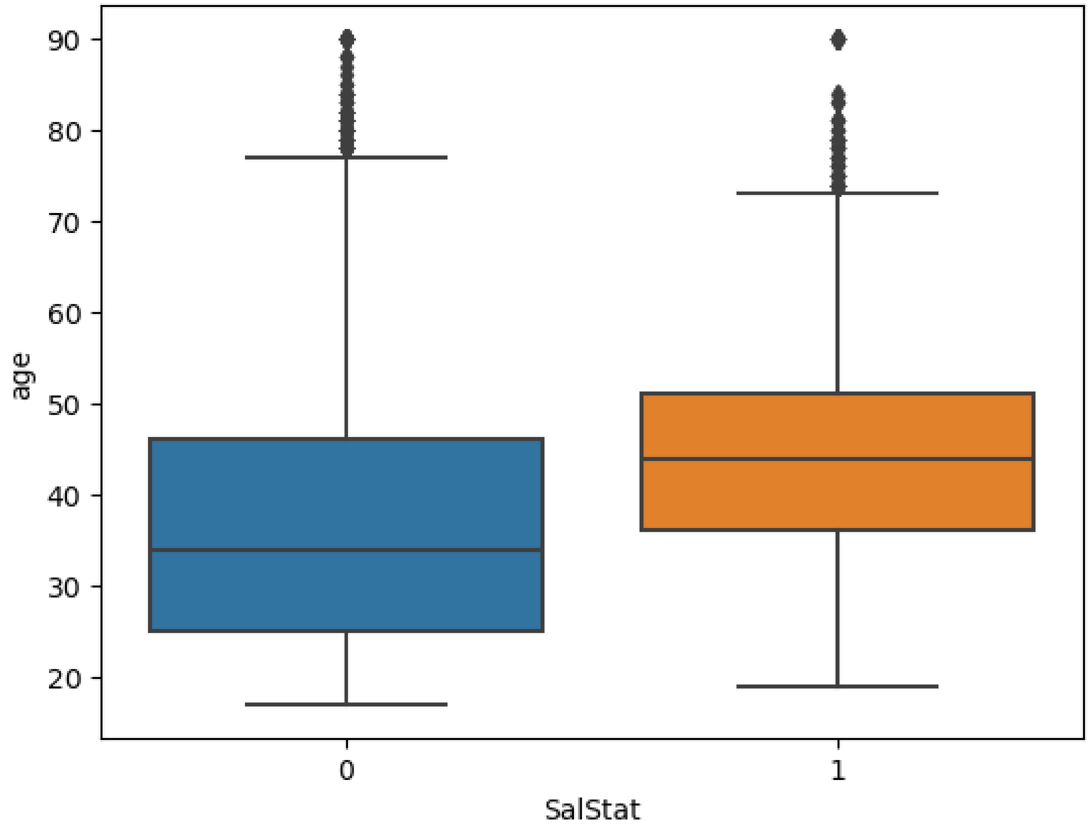
```
D:\Anaconda\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as
_na option is deprecated and will be removed in a future version. Convert inf val
ues to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Out[19]: <seaborn.axisgrid.FacetGrid at 0x22d3b02f690>



In [20]:
```
sns.boxplot(x='SalStat',y='age',data=data2)
```

Out[20]: <Axes: xlabel='SalStat', ylabel='age'>



Logistic Regression, KNN, Confusion Matrix, Accuracy Score

In [21]:
```
from sklearn.model_selection import train_test_split # module for testing models
from sklearn.linear_model import LogisticRegression # module to perform logistic
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [22]:
```
new_data = pd.get_dummies(data2,drop_first=True) # creating dummies
```

In [23]:
```
columns_list = list(new_data.columns)
print(columns_list)
```

```
['age', 'capitalgain', 'capitalloss', 'hoursperweek', 'SalStat', 'JobType_ Federa
l-gov', 'JobType_ Local-gov', 'JobType_ Never-worked', 'JobType_ Private', 'JobTy
pe_ Self-emp-inc', 'JobType_ Self-emp-not-inc', 'JobType_ State-gov', 'JobType_ W
ithout-pay', 'EdType_ 11th', 'EdType_ 12th', 'EdType_ 1st-4th', 'EdType_ 5th-6t
h', 'EdType_ 7th-8th', 'EdType_ 9th', 'EdType_ Assoc-acdm', 'EdType_ Assoc-voc',
'EdType_ Bachelors', 'EdType_ Doctorate', 'EdType_ HS-grad', 'EdType_ Masters',
'EdType_ Preschool', 'EdType_ Prof-school', 'EdType_ Some-college', 'maritalstatu
s_ Married-AF-spouse', 'maritalstatus_ Married-civ-spouse', 'maritalstatus_ Marri
ed-spouse-absent', 'maritalstatus_ Never-married', 'maritalstatus_ Separated', 'm
aritalstatus_ Widowed', 'occupation_ Adm-clerical', 'occupation_ Armed-Forces',
'occupation_ Craft-repair', 'occupation_ Exec-managerial', 'occupation_ Farming-f
ishing', 'occupation_ Handlers-cleaners', 'occupation_ Machine-op-inspct', 'occup
ation_ Other-service', 'occupation_ Priv-house-serv', 'occupation_ Prof-specialt
y', 'occupation_ Protective-serv', 'occupation_ Sales', 'occupation_ Tech-suppor
t', 'occupation_ Transport-moving', 'relationship_ Not-in-family', 'relationship_
Other-relative', 'relationship_ Own-child', 'relationship_ Unmarried', 'relations
hip_ Wife', 'race_ Asian-Pac-Islander', 'race_ Black', 'race_ Other', 'race_ Whit
e', 'gender_ Male', 'nativecountry_ Canada', 'nativecountry_ China', 'nativecount
ry_ Columbia', 'nativecountry_ Cuba', 'nativecountry_ Dominican-Republic', 'nativ
ecountry_ Ecuador', 'nativecountry_ El-Salvador', 'nativecountry_ England', 'nati
vecountry_ France', 'nativecountry_ Germany', 'nativecountry_ Greece', 'nativecou
ntry_ Guatemala', 'nativecountry_ Haiti', 'nativecountry_ Holand-Netherlands', 'n
ativecountry_ Honduras', 'nativecountry_ Hong', 'nativecountry_ Hungary', 'native
country_ India', 'nativecountry_ Iran', 'nativecountry_ Ireland', 'nativecountry_
Italy', 'nativecountry_ Jamaica', 'nativecountry_ Japan', 'nativecountry_ Laos',
'nativecountry_ Mexico', 'nativecountry_ Nicaragua', 'nativecountry_ Outlying-US
(Guam-USVI-etc)', 'nativecountry_ Peru', 'nativecountry_ Philippines', 'nativecou
ntry_ Poland', 'nativecountry_ Portugal', 'nativecountry_ Puerto-Rico', 'nativeco
untry_ Scotland', 'nativecountry_ South', 'nativecountry_ Taiwan', 'nativecountry
_ Thailand', 'nativecountry_ Trinadad&Tobago', 'nativecountry_ United-States', 'n
ativecountry_ Vietnam', 'nativecountry_ Yugoslavia']
```

In [24]:
```
features=list(set(columns_list)-set(['SalStat']))
print(features)
```

```
['EdType_ 7th-8th', 'nativecountry_ United-States', 'relationship_ Not-in-famil
y', 'JobType_ Local-gov', 'nativecountry_ Canada', 'nativecountry_ Poland', 'race
_ White', 'race_ Asian-Pac-Islander', 'nativecountry_ China', 'nativecountry_ Eng
land', 'capitalgain', 'occupation_ Handlers-cleaners', 'EdType_ 11th', 'nativecou
ntry_ Taiwan', 'capitalloss', 'race_ Black', 'nativecountry_ Portugal', 'JobType_
Never-worked', 'occupation_ Farming-fishing', 'occupation_ Transport-moving', 'na
tivecountry_ Germany', 'occupation_ Sales', 'nativecountry_ Honduras', 'nativecou
ntry_ Ecuador', 'occupation_ Exec-managerial', 'nativecountry_ Outlying-US(Guam-U
SVI-etc)', 'maritalstatus_ Married-AF-spouse', 'nativecountry_ Nicaragua', 'nativ
ecountry_ Japan', 'nativecountry_ Dominican-Republic', 'occupation_ Craft-repai
r', 'nativecountry_ Trinadad&Tobago', 'relationship_ Wife', 'nativecountry_ Indi
a', 'relationship_ Other-relative', 'nativecountry_ Puerto-Rico', 'EdType_ 1st-4t
h', 'occupation_ Other-service', 'race_ Other', 'gender_ Male', 'EdType_ 5th-6t
h', 'JobType_ State-gov', 'EdType_ Bachelors', 'nativecountry_ Ireland', 'nativec
ountry_ Vietnam', 'nativecountry_ Scotland', 'EdType_ Prof-school', 'occupation_
Tech-support', 'nativecountry_ Columbia', 'occupation_ Adm-clerical', 'nativecoun
try_ Cuba', 'JobType_ Self-emp-not-inc', 'maritalstatus_ Widowed', 'nativecountry
_ Hungary', 'nativecountry_ Haiti', 'EdType_ Masters', 'EdType_ 12th', 'nativecou
ntry_ Laos', 'EdType_ Assoc-acdm', 'nativecountry_ Philippines', 'nativecountry_
Italy', 'hoursperweek', 'nativecountry_ Jamaica', 'EdType_ 9th', 'age', 'relation
ship_ Unmarried', 'occupation_ Priv-house-serv', 'EdType_ Doctorate', 'nativecoun
try_ Hong', 'nativecountry_ Yugoslavia', 'occupation_ Prof-specialty', 'JobType_
Federal-gov', 'occupation_ Machine-op-inspct', 'nativecountry_ South', 'EdType_ A
ssoc-voc', 'relationship_ Own-child', 'occupation_ Protective-serv', 'nativecount
ry_ Greece', 'EdType_ Preschool', 'nativecountry_ Holand-Netherlands', 'JobType_
Private', 'nativecountry_ Guatemala', 'occupation_ Armed-Forces', 'nativecountry_
Iran', 'EdType_ Some-college', 'nativecountry_ Thailand', 'maritalstatus_ Separat
ed', 'EdType_ HS-grad', 'nativecountry_ El-Salvador', 'maritalstatus_ Never-marri
ed', 'JobType_ Without-pay', 'nativecountry_ France', 'JobType_ Self-emp-inc', 'm
aritalstatus_ Married-spouse-absent', 'maritalstatus_ Married-civ-spouse', 'nativ
ecountry_ Mexico', 'nativecountry_ Peru']
```

In [25]:
```
y=new_data['SalStat'].values # creating variable for regression analysis
```

In [26]:
```
x=new_data[features].values # creating variable for regression analysis
```

In [27]:
```
train_x,test_x,train_y,test_y=train_test_split(x,y,test_size=0.3,random_state=0)
Logistic=LogisticRegression()
Logistic.fit(train_x,train_y)
P=Logistic.predict(test_x)
print(P)
```

```
[0 0 0 ... 0 0 0]
```

```
D:\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceW
arning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [28]:
```
print(data2['SalStat'])
```

```
0        0
1        0
2        1
3        0
4        0
        ..
31973    0
31974    0
31975    0
31976    0
31977    0
Name: SalStat, Length: 31978, dtype: int64
```

In [29]:
```
confusion_mat  =  confusion_matrix(test_y,P)
print(confusion_mat) # Confusion matrix between test_y and predicted_x
```

```
[[6827  509]
 [ 932 1326]]
```

In [30]:
```
acc_score  =  accuracy_score(test_y,P)
print(acc_score) # accuracy of the model
```

```
0.8498019595580572
```

In [31]:
```
from sklearn.neighbors import KNeighborsClassifier # Module for KNN Classifer
```

In [32]:
```
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(train_x,train_y)
prediction=knn.predict(test_x)
print(prediction)
```

```
[0 0 0 ... 0 0 0]
```

In [33]:
```
con_mat  = confusion_matrix(test_y,prediction)
acc_score1  =  accuracy_score(test_y,prediction)
print(acc_score1)
```

```
0.8394830102147175
```

In [35]:
```
for i in range(1,20):
    knn1 = KNeighborsClassifier(n_neighbors=i)
    knn1.fit(train_x,train_y)
    pred=knn1.predict(test_x)
    acc=accuracy_score(test_y,pred)
    print(i)
    print(acc)
#creating a loop to find out at what range of neighbour the accuracy is the high
```

1
0.8154054617469252
2
0.8443819053575151
3
0.8323952470293934
4
0.8431311236189285
5
0.8394830102147175
6
0.8469877006462372
7
0.8427141963727329
8
0.8494892641234104
9
0.848134250573275
10
0.8501146549927038
11
0.848134250573275
12
0.8483427141963727
13
0.849072336877215
14
0.851678132165937
15
0.848134250573275
16
0.8512612049197416
17
0.8512612049197416
18
0.8528246820929748
19
0.8512612049197416

Analysing and Cleaning the data (Cars_sampled)

In [1]:
```
import pandas as pd # pandas module for data manipulation
import numpy as np # numpy module for mathematical calulations
import seaborn as sns # seaborn module for data visualization
```

In [3]:
```
cars_data  =  pd.read_csv('C:/Users/NK/Desktop/Python/data/cars_sampled.csv')  #  re
```

In [4]:
```
cars_data.info()
```

```
<class  'pandas.core.frame.DataFrame'>
RangeIndex: 50001 entries, 0 to 50000
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   dateCrawled          50001 non-null  object
 1   name                 50001 non-null  object
 2   seller               50001 non-null  object
 3   offerType            50001 non-null  object
 4   price                50001 non-null  int64 
 5   abtest               50001 non-null  object
 6   vehicleType          44813 non-null  object
 7   yearOfRegistration   50001 non-null  int64 
 8   gearbox              47177 non-null  object
 9   powerPS              50001 non-null  int64 
 10  model                47243 non-null  object
 11  kilometer            50001 non-null  int64 
 12  monthOfRegistration  50001 non-null  int64 
 13  fuelType             45498 non-null  object
 14  brand                50001 non-null  object
 15  notRepairedDamage    40285 non-null  object
 16  dateCreated          50001 non-null  object
 17  postalCode           50001 non-null  int64 
 18  lastSeen             50001 non-null  object
dtypes: int64(6), object(13)
memory usage: 7.2+ MB
```

In [5]:
```
col=['name','dateCrawled','dateCreated','postalCode','lastSeen']
cars=cars_data.drop(columns=col,axis=1) # droping columns
cars_data.drop_duplicates(keep='first',inplace=True)  #  droping  duplicates
```

In [6]:
```
cars.isnull().sum() # checking number of null values
```

```
Out[6]: seller                 0
        offerType              0
        price                  0
        abtest                 0
        vehicleType         5188
        yearOfRegistration     0
        gearbox             2824
        powerPS                0
        model               2758
        kilometer              0
        monthOfRegistration    0
        fuelType            4503
        brand                  0
        notRepairedDamage   9716
        dtype: int64
```

```
In [8]: yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
        yearwise_count # year wise counting on the cars on the base of year of registrat
```

```
Out[8]: yearOfRegistration
        1000     6
        1255     1
        1500     2
        1910    15
        1928     1
                ..
        7500     1
        7800     1
        8500     1
        8888     2
        9999     7
        Name: count, Length: 97, dtype: int64
```

```
In [9]: yearwise_count=cars['price'].value_counts().sort_index()
        yearwise_count #year wise counting on the cars on the base of price
```

```
Out[9]: price
        0           1451
        1            172
        2              1
        3              1
        5              4
                    ...
        1250000        1
        2795000        1
        9999999        1
        10010011       1
        12345678       1
        Name: count, Length: 2393, dtype: int64
```

```
In [10]: yearwise_count=cars['powerPS'].value_counts().sort_index()
         yearwise_count #year wise counting on the cars on the base of powerPS
```

Out[10]:
```
powerPS
0        5605
1           3
2           2
3           2
4           4
         ...
15033       1
16011       1
16312       1
19211       1
19312       1
Name: count, Length: 460, dtype: int64
```

In [11]:
```
sum(cars['yearOfRegistration']>2018)
```

Out[11]: 26

In [12]:
```
sum(cars['yearOfRegistration']<1950)
```

Out[12]: 39

In [13]:
```
cars=cars[(cars.yearOfRegistration<=2018)
         &(cars.yearOfRegistration>=1950)
         &(cars.price>100)
         &(cars.price<=150000)
         &(cars.powerPS>=10)
         &(cars.powerPS<=500)] # removing outliers
cars
```

Out[13]:

	seller	offerType	price	abtest	vehicleType	yearOfRegistration	gearbo
0	private	offer	4450	test	limousine	2003	manua
1	private	offer	13299	control	suv	2005	manua
2	private	offer	3200	test	bus	2003	manua
3	private	offer	4500	control	small car	2006	manua
4	private	offer	18750	test	suv	2008	automati
...	...	...	...	...	...	...	.
49991	private	offer	10900	test	limousine	2004	manua
49992	private	offer	790	test	limousine	1998	manua
49993	private	offer	830	test	small car	1999	manua
49995	private	offer	2290	test	station wagon	2001	manua
50000	commercial	offer	1100	test	small car	2006	manua

43070 rows × 14 columns

In [14]:
```
cars_omit=cars.dropna(axis=0) # dropping na values
cars_omit=pd.get_dummies(cars_omit,drop_first=True) # creating dummies
```

```
cars_omit.head()
```

Out[14]:

	price	yearOfRegistration	powerPS	kilometer	monthOfRegistration	seller_private
1	13299	2005	163	150000	6	True
3	4500	2006	86	60000	12	True
4	18750	2008	185	150000	11	True
5	988	1995	90	150000	2	True
7	1399	1997	136	150000	11	True

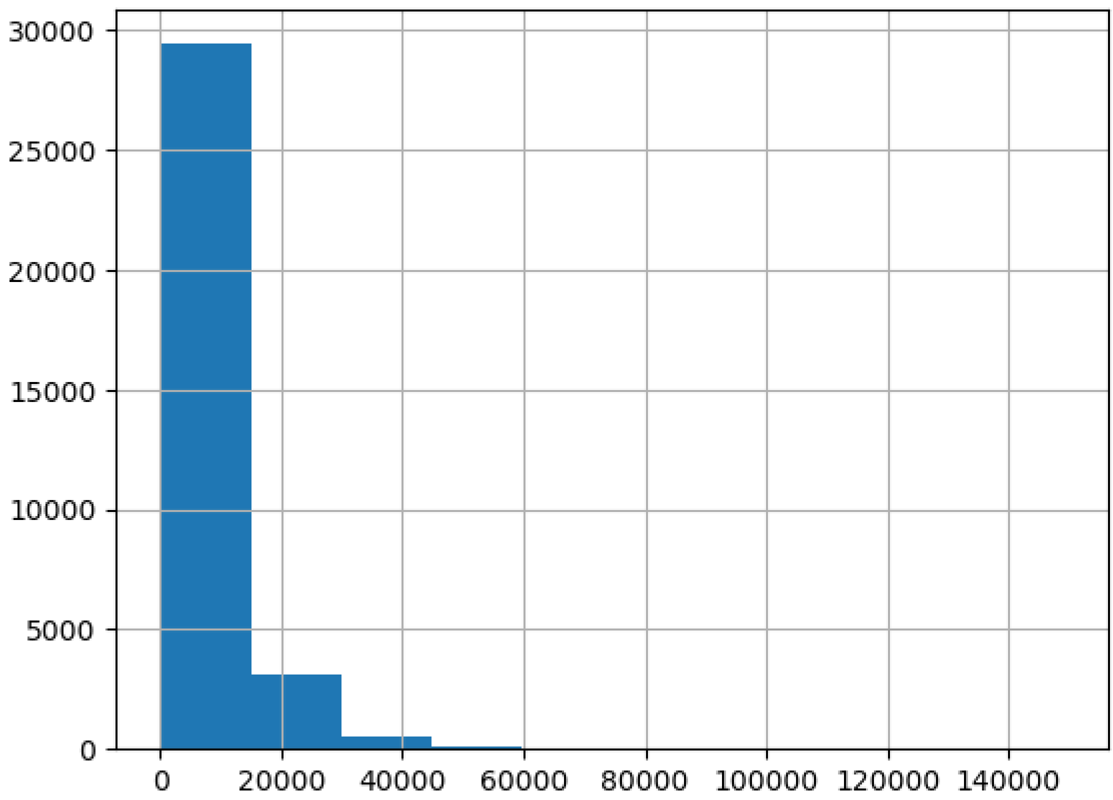
5 rows × 304 columns

Performing Linear Regression

In [15]:
```
from sklearn.model_selection import train_test_split # module for model testing
from sklearn.linear_model import LinearRegression # module for linear regression
x1=cars_omit.drop(['price'],axis='columns',inplace=False)
y1=cars_omit['price']
```

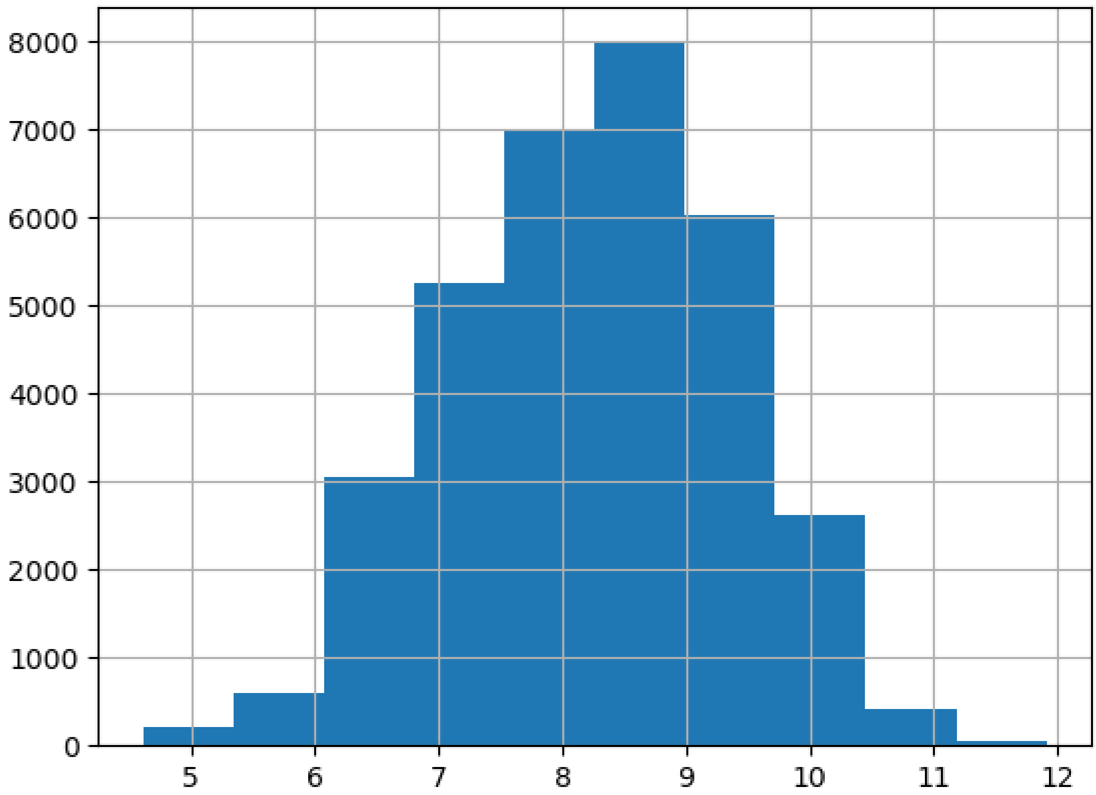
In [16]:
```
y1.hist()
```

Out[16]: <Axes: >



In [17]:
```
y1=np.log(y1)
y1.hist()
```

Out[17]: <Axes: >



In [18]:
```
x_train,x_test,y_train,y_test=train_test_split(x1,y1,test_size=0.30,random_state
lgr=LinearRegression(fit_intercept=True)
model_lin=lgr.fit(x_train,y_train)
pred_y=lgr.predict(x_test)
print(pred_y) # Finding predicted values of Y on X
```

```
[8.84211302 6.90450852 8.65586627 ... 8.15872583 9.75525483 9.29969386]
```

In [19]:
```
r2_test=model_lin.score(x_test,y_test)
r2_train=model_lin.score(x_train,y_train)
print(r2_test,r2_train)
```

```
0.7677908838656784 0.7816222211308383
```

Statistical Test (t-test)

In [2]:
```
import pandas as pd # pandas module for data manipulation
import numpy as np # numpy module for mathimatical calulations
from scipy import stats # scipy module for statistics
```

In [3]:
```
marks = pd.read_excel('C:/Users/NK/Desktop/Python/data/Marks.xlsx') # reading th
```

In [4]:
```
marks.head() # vewing the data
```

Out[4]:

	B	A	C
0	13	3	16
1	10	20	3
2	12	19	6
3	3	11	18
4	18	13	19

In [5]:
```
np.mean(marks['A'])  # checking mean of column A
```

Out[5]:
```
11.033333333333333
```

In [6]:
```
np.mean(marks['B'])  # checking mean of column B
```

Out[6]:
```
10.55
```

In [7]:
```
np.mean(marks['C'])  # checking mean of column C
```

Out[7]:
```
9.433333333333334
```

In [8]:
```
np.median(marks['A']) # checking median of column A
```

Out[8]:
```
10.0
```

In [9]:
```
np.median(marks['B']) # checking median of column B
```

Out[9]:
```
11.0
```

In [10]:
```
np.median(marks['C']) # checking median of column C
```

Out[10]:
```
8.5
```

In [11]:
```
stats.mode(marks['A']) # checking mode of column A
```

Out[11]:
```
ModeResult(mode=20,  count=7)
```

In [12]:
```
stats.mode(marks['B']) # checking mode of column B
```

Out[12]:
```
ModeResult(mode=10,  count=6)
```

In [13]:
```
stats.mode(marks['C']) # checking mode of column C
```
Out[13]:
```
ModeResult(mode=6, count=9)
```
In [14]:
```
np.percentile(marks['A'], 50) # checking 50th percentile of column A
```
Out[14]:
```
10.0
```
In [15]:
```
np.percentile(marks['B'], 50) # checking 50th percentile of column B
```
Out[15]:
```
11.0
```
In [16]:
```
np.percentile(marks['C'], 50) # checking 50th percentile of column C
```
Out[16]:
```
8.5
```
In [17]:
```
np.var(marks['A']) # checking variance of column A
```
Out[17]:
```
33.93222222222222
```
In [18]:
```
np.var(marks['B']) # checking variance of column B
```
Out[18]:
```
30.180833333333343
```
In [19]:
```
np.var(marks['C']) # checking variance of column C
```
Out[19]:
```
29.91222222222222
```
In [20]:
```
stats.ttest_ind(marks['A'], marks['C']) # performing t-test on column A and C
```
Out[20]:
```
TtestResult(statistic=1.5380995049141182, pvalue=0.1267015430917605, df=118.0)
```
In [21]:
```
stats.ttest_rel(marks['B'], marks['A']) # performing t-test on column B and C
```
Out[21]:
```
TtestResult(statistic=-0.3929056110076333, pvalue=0.695805248962132, df=59)
```
In [ ]:
```
# P-value is greater than 0.05, Thus the data is insignificant.
```