Name : Kundan Patil
Email id: patilkundan.1718@gmail.com (mailto:patilkundan.1718@gmail.com)
contact: +91 8485010139
linkedIn: www.linkedin.com/in/kundanpatilds (http://www.linkedin.com/in/kundanpatilds)
Github: https://github.com/patilkundan?tab=repositories (https://github.com/patilkundan?tab=repositories)

Q.5 Write Python Programming for given Dataset “mtcars.csv” file

Expected Output:

1. Need to use required libraries
2. Apply different Data Pre- Processing Techniques using Pandas, NumPy
3. Store data in list

1 Importing required libraries

In [27]:

```
#Let's start with importing required libraries
import sklearn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler
```

In [3]:

```
df = pd. read_csv ("mtcars.csv") # Reading the Data
df.head()
```

Out[3]:

	name	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

1. First and the most important thing is to check the shape of data ...... How Big is our dataset

In [4]:

```
df.shape
```

Out[4]:

```
(32, 12)
```

The given data set is small

2. Lets check the data types ....what are data types present in our dataset

In [5]:

```
df.dtypes # df.info() also this way can be done
```

Out[5]:

```
name     object
mpg     float64
cyl       int64
disp    float64
hp        int64
drat    float64
wt      float64
qsec    float64
vs        int64
am        int64
gear      int64
carb      int64
dtype: object
```

In [6]:

```
df.columns
```

Out[6]:

```
Index(['name', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
       'gear', 'carb'],
      dtype='object')
```

3. Lets Check null values in data set

In [7]:

```
df.isnull().sum()
```

Out[7]:

```
name    0
mpg     0
cyl     0
disp    0
hp      0
drat    0
wt      0
qsec    0
vs      0
am      0
gear    0
carb    0
dtype: int64
```

on the above evidence we can make statement, there is no null present in our data set

4. Checking the duplicate....just to make sure that

In [8]:

```
duplicate = df[df.duplicated()]
print("Duplicate Rows :")
```

```
Duplicate Rows :
```

Great no duplicate found we can move to next

5. Exploratory Data Analysis

1. name

In [10]:

```
df['name'].describe()
```

Out[10]:

```
count            32
unique           32
top       Mazda RX4
freq              1
Name: name, dtype: object
```

In [18]:

```
print('unique categories--->',df['name'].unique())
#counting the uniques
print('value_counts for each unique categories---->',df['name'].value_counts())
print('Null value--->',df['name'].isnull().sum())
```

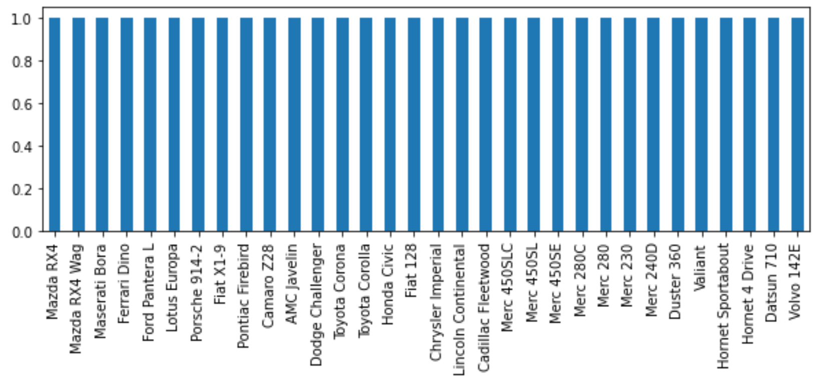
```
unique categories---> ['Mazda RX4' 'Mazda RX4 Wag' 'Datsun 710' 'Hornet 4 Drive'
 'Hornet Sportabout' 'Valiant' 'Duster 360' 'Merc 240D' 'Merc 230'
 'Merc 280' 'Merc 280C' 'Merc 450SE' 'Merc 450SL' 'Merc 450SLC'
 'Cadillac Fleetwood' 'Lincoln Continental' 'Chrysler Imperial' 'Fiat 128'
 'Honda Civic' 'Toyota Corolla' 'Toyota Corona' 'Dodge Challenger'
 'AMC Javelin' 'Camaro Z28' 'Pontiac Firebird' 'Fiat X1-9' 'Porsche 914-2'
 'Lotus Europa' 'Ford Pantera L' 'Ferrari Dino' 'Maserati Bora'
 'Volvo 142E']
value_counts for each unique categories----> Mazda RX4              1
Mazda RX4 Wag          1
Maserati Bora          1
Ferrari Dino           1
Ford Pantera L         1
Lotus Europa           1
Porsche 914-2          1
Fiat X1-9              1
Pontiac Firebird       1
Camaro Z28             1
AMC Javelin            1
Dodge Challenger       1
Toyota Corona          1
Toyota Corolla         1
Honda Civic            1
Fiat 128               1
Chrysler Imperial      1
Lincoln Continental    1
Cadillac Fleetwood     1
Merc 450SLC            1
Merc 450SL             1
Merc 450SE             1
Merc 280C              1
Merc 280               1
Merc 230               1
Merc 240D              1
Duster 360             1
Valiant                1
Hornet Sportabout      1
Hornet 4 Drive         1
Datsun 710             1
Volvo 142E             1
Name: name, dtype: int64
Null value---> 0
```

In [22]:

```
df['name'].value_counts().plot(kind='bar',figsize=(10, 3))
```

Out[22]:

```
<AxesSubplot:>
```



Basically for continoues coloumn we check

- box plot
- distribution plot
- histogram ..many more also ...

2. mpg

In [23]:

```
df['mpg'].describe()
```

Out[23]:

```
count    32.000000
mean     20.090625
std       6.026948
min      10.400000
25%      15.425000
50%      19.200000
75%      22.800000
max      33.900000
Name: mpg, dtype: float64
```

In [29]:

```
df['mpg'].skew()
```

Out[29]:

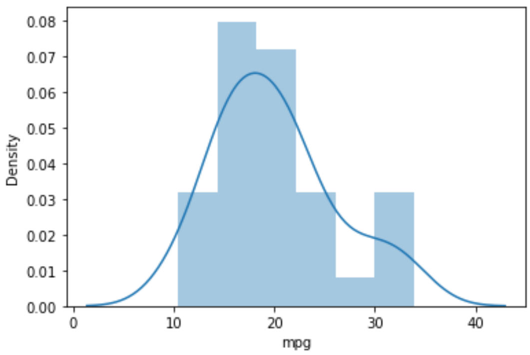
```
0.6723771376290805
```

In [28]:

```
sns.distplot(df['mpg'])
```

Out[28]:

```
<AxesSubplot:xlabel='mpg', ylabel='Density'>
```

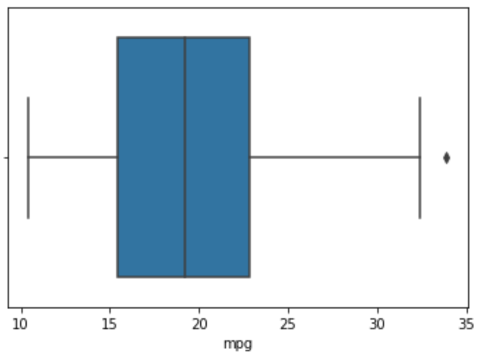


In [30]:

```
sns.boxplot(df['mpg'])
```

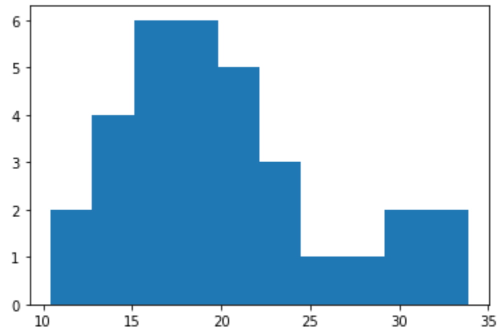
Out[30]:

```
<AxesSubplot:xlabel='mpg'>
```



In [31]:

```
plt.hist(df['mpg'])
plt.show()
```



3. cyl

In [32]:

```
df['cyl'].describe()
```

Out[32]:

```
count    32.000000
mean      6.187500
std       1.785922
min       4.000000
25%       4.000000
50%       6.000000
75%       8.000000
max       8.000000
Name: cyl, dtype: float64
```

In [33]:

```
df['cyl'].skew()
```

Out[33]:

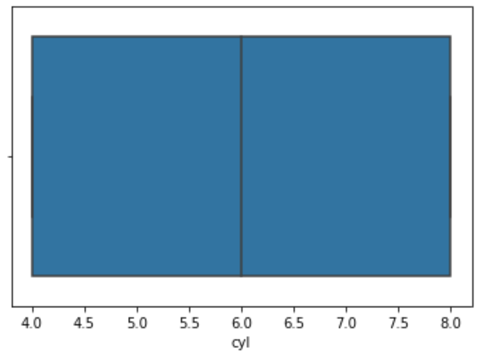
```
-0.19226086006934684
```

In [34]:

```
sns.boxplot(df['cyl'])
```

Out[34]:

```
<AxesSubplot:xlabel='cyl'>
```



In [35]:

```
print('unique categories--->',df['cyl'].unique())
#counting the uniques
print('value_counts for each unique categories---->',df['cyl'].value_counts())
print('Null value--->',df['cyl'].isnull().sum())
```

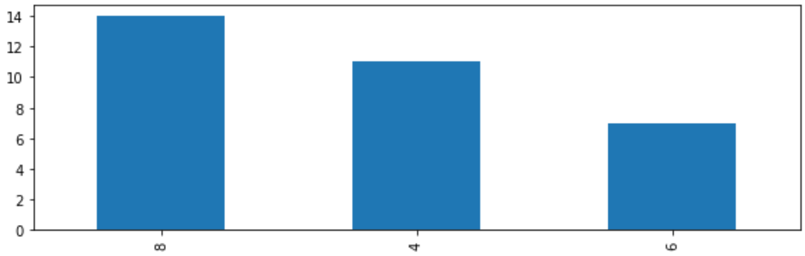
```
unique categories---> [6 4 8]
value_counts for each unique categories----> 8    14
4    11
6     7
Name: cyl, dtype: int64
Null value---> 0
```

In [37]:

```
df['cyl'].value_counts().plot(kind='bar',figsize=(10, 3))
```

Out[37]:

<AxesSubplot:>

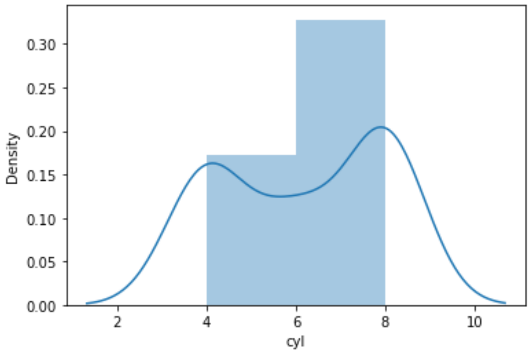


In [38]:

```
sns.distplot(df['cyl'])
```

Out[38]:

<AxesSubplot:xlabel='cyl', ylabel='Density'>



4. disp

In [39]:

```
df['disp'].describe()
```

Out[39]:

```
count     32.000000
mean     230.721875
std      123.938694
min       71.100000
25%      120.825000
50%      196.300000
75%      326.000000
max      472.000000
Name: disp, dtype: float64
```

In [41]:

```
df['disp'].skew()
```

Out[41]:

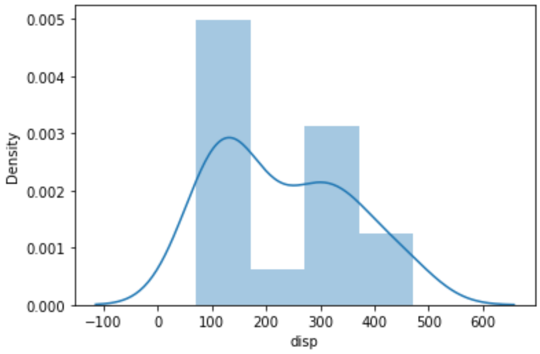
0.42023312147004516

In [42]:

```
sns.distplot(df['disp'])
```

Out[42]:

```
<AxesSubplot:xlabel='disp', ylabel='Density'>
```

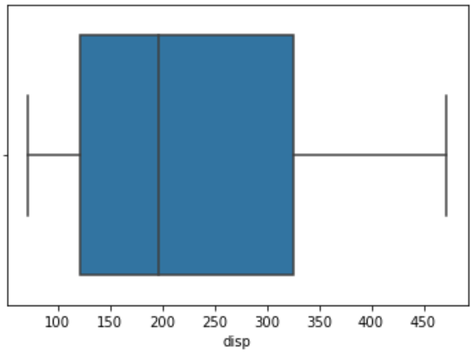


In [43]:

```
sns.boxplot(df['disp'])
```

Out[43]:

```
<AxesSubplot:xlabel='disp'>
```



5.hp

In [44]:

```
df['hp'].describe()
```

Out[44]:

```
count     32.000000
mean     146.687500
std       68.562868
min       52.000000
25%       96.500000
50%      123.000000
75%      180.000000
max      335.000000
Name: hp, dtype: float64
```

In [45]:

```
df['hp'].skew()
```

Out[45]:

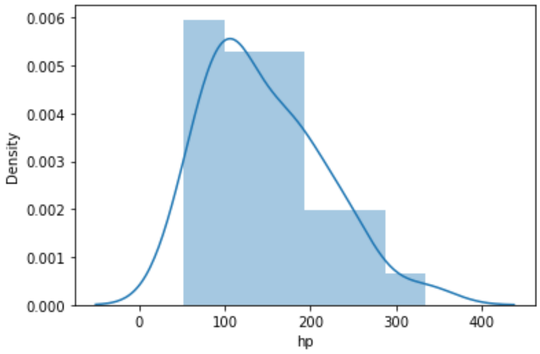
```
0.7994066925956381
```

In [46]:

```
sns.distplot(df['hp'])
```

Out[46]:

<AxesSubplot:xlabel='hp', ylabel='Density'>

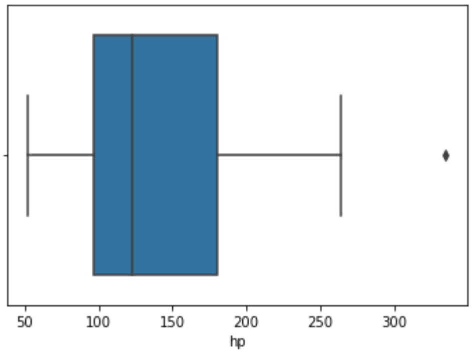


In [47]:

```
sns.boxplot(df['hp'])
```

Out[47]:

<AxesSubplot:xlabel='hp'>



In [48]:

```
print('unique categories--->',df['hp'].unique())
#counting the uniques
print('value_counts for each unique categories---->',df['hp'].value_counts())
print('Null value--->',df['hp'].isnull().sum())
```

```
unique categories---> [110  93 175 105 245  62  95 123 180 205 215 230  66  52  65  97 150  91
 113 264 335 109]
value_counts for each unique categories----> 110    3
175    3
180    3
245    2
123    2
150    2
66     2
65     1
335    1
264    1
113    1
91     1
97     1
230    1
52     1
93     1
215    1
205    1
95     1
62     1
105    1
109    1
Name: hp, dtype: int64
Null value---> 0
```

6. drat

In [49]:

```
df['drat'].describe()
```

Out[49]:

```
count    32.000000
mean      3.596563
std       0.534679
min       2.760000
25%       3.080000
50%       3.695000
75%       3.920000
max       4.930000
Name: drat, dtype: float64
```

In [50]:

```
df['drat'].skew()
```

Out[50]:

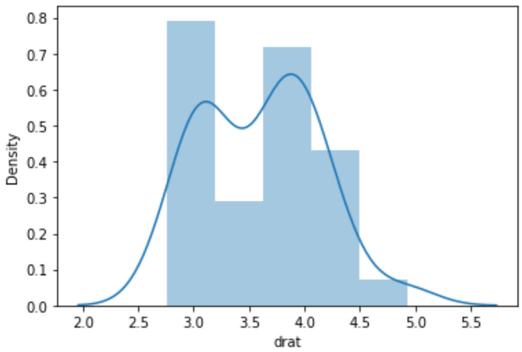
```
0.29278021324083486
```

In [51]:

```
sns.distplot(df['drat'])
```

Out[51]:

```
<AxesSubplot:xlabel='drat', ylabel='Density'>
```

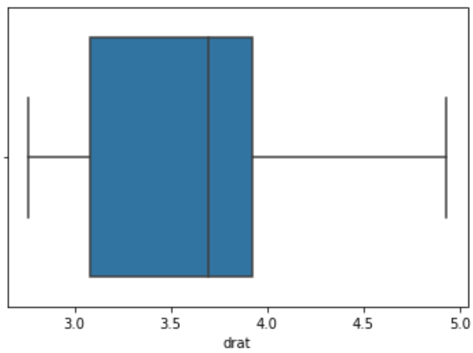


In [52]:

```
sns.boxplot(df['drat'])
```

Out[52]:

```
<AxesSubplot:xlabel='drat'>
```



7. wt

In [53]:

```
1 df['wt'].describe()
```

Out[53]:

```
count    32.000000
mean      3.217250
std       0.978457
min       1.513000
25%       2.581250
50%       3.325000
75%       3.610000
max       5.424000
Name: wt, dtype: float64
```

In [54]:

```
1 df['wt'].skew()
```

Out[54]:

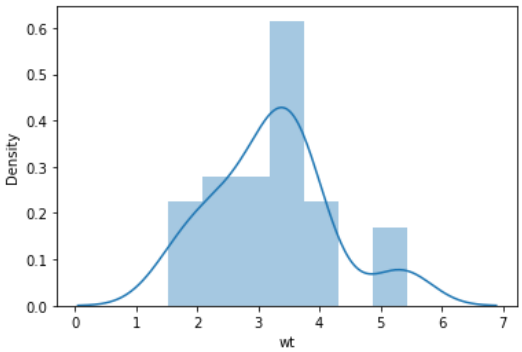
```
0.4659161067929858
```

In [55]:

```
1 sns.distplot(df['wt'])
```

Out[55]:

```
<AxesSubplot:xlabel='wt', ylabel='Density'>
```

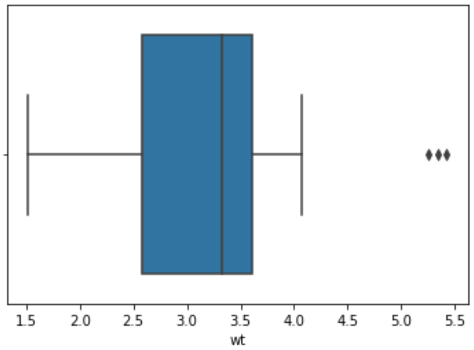


In [56]:

```
1 sns.boxplot(df['wt'])
```

Out[56]:

```
<AxesSubplot:xlabel='wt'>
```



8. qsec

In [57]:

```
1 df['qsec'].describe()
```

Out[57]:

```
count    32.000000
mean     17.848750
std       1.786943
min      14.500000
25%      16.892500
50%      17.710000
75%      18.900000
max      22.900000
Name: qsec, dtype: float64
```

In [59]:

```
1 df['qsec'].skew()
```

Out[59]:

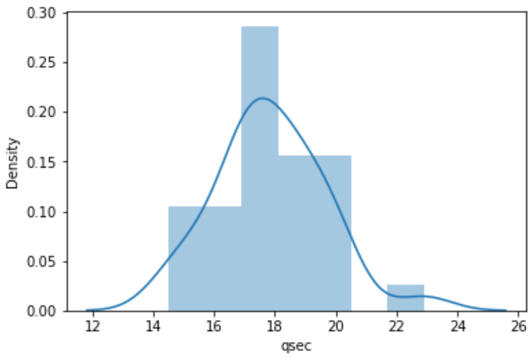
```
0.4063466292404903
```

In [60]:

```
1 sns.distplot(df['qsec'])
```

Out[60]:

<AxesSubplot:xlabel='qsec', ylabel='Density'>

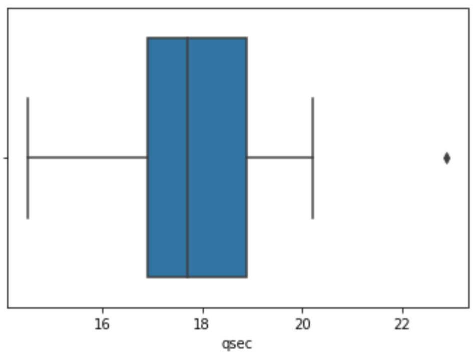


In [61]:

```
1 sns.boxplot(df['qsec'])
```

Out[61]:

<AxesSubplot:xlabel='qsec'>



9. vs

In [62]:

```
df['vs'].describe()
```

Out[62]:

```
count    32.000000
mean      0.437500
std       0.504016
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       1.000000
Name: vs, dtype: float64
```

In [63]:

```
df['vs'].skew()
```

Out[63]:

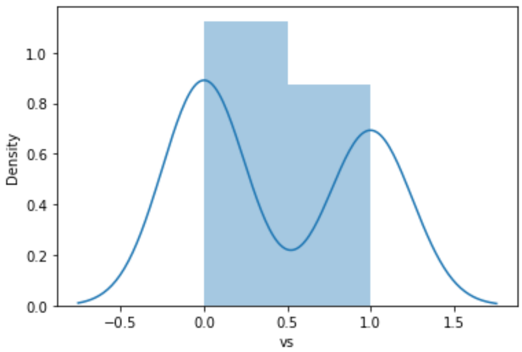
```
0.2645417988063449
```

In [64]:

```
sns.distplot(df['vs'])
```

Out[64]:

```
<AxesSubplot:xlabel='vs', ylabel='Density'>
```

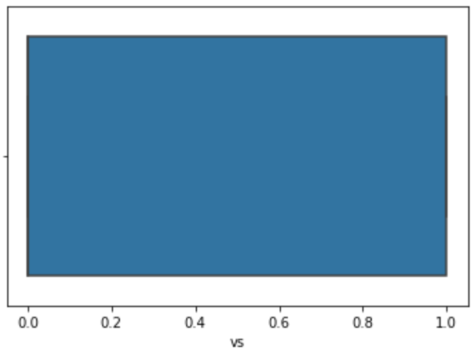


In [65]:

```
sns.boxplot(df['vs'])
```

Out[65]:

```
<AxesSubplot:xlabel='vs'>
```



In [67]:

```
print('unique categories--->',df['vs'].unique())
#counting the uniques
print('value_counts for each unique categories---->',df['vs'].value_counts())
print('Null value--->',df['vs'].isnull().sum())
```

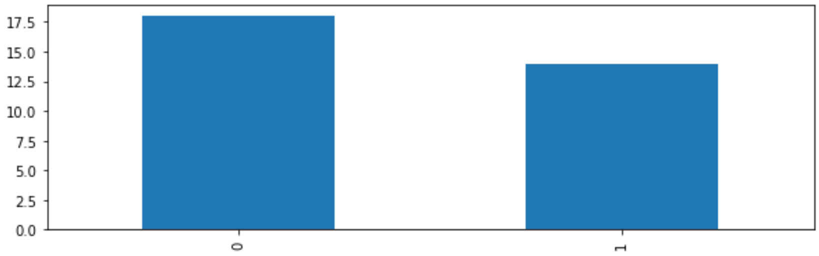
```
unique categories---> [0 1]
value_counts for each unique categories----> 0    18
1    14
Name: vs, dtype: int64
Null value---> 0
```

In [68]:

```
1  df['vs'].value_counts().plot(kind='bar',figsize=(10, 3))
```

Out[68]:

```
<AxesSubplot:>
```



10. am

In [69]:

```
1  df['am'].describe()
```

Out[69]:

```
count    32.000000
mean      0.406250
std       0.498991
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       1.000000
Name: am, dtype: float64
```

In [70]:

```
1  df['am'].skew()
```

Out[70]:

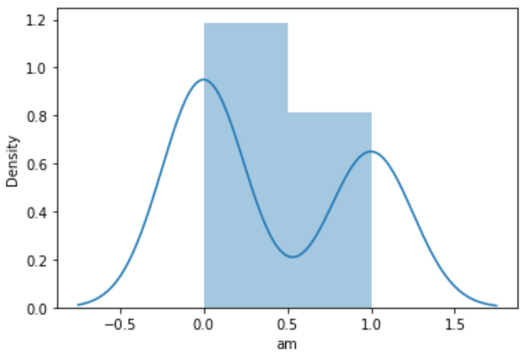
```
0.40080889870280095
```

In [71]:

```
1  sns.distplot(df['am'])
```

Out[71]:

```
<AxesSubplot:xlabel='am', ylabel='Density'>
```

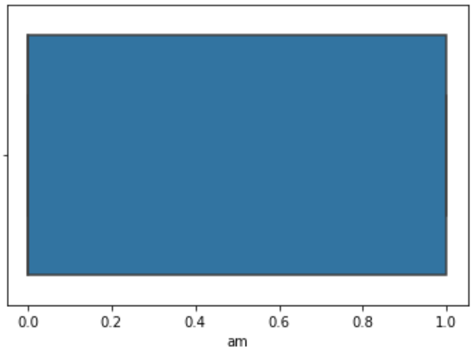


In [72]:

```
sns.boxplot(df['am'])
```

Out[72]:

```
<AxesSubplot:xlabel='am'>
```



In [73]:

```
print('unique categories--->',df['am'].unique())
#counting the uniques
print('value_counts for each unique categories---->',df['am'].value_counts())
print('Null value--->',df['am'].isnull().sum())
```

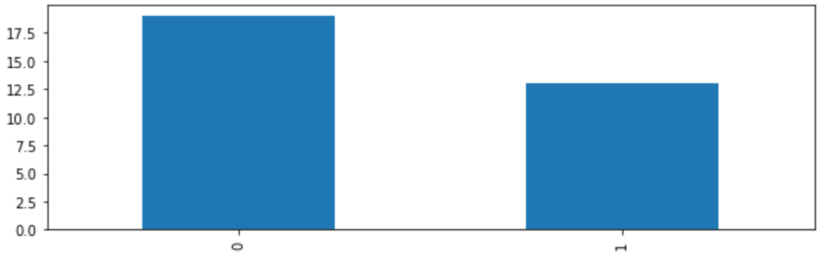
```
unique categories---> [1 0]
value_counts for each unique categories----> 0    19
1    13
Name: am, dtype: int64
Null value---> 0
```

In [74]:

```
df['am'].value_counts().plot(kind='bar',figsize=(10, 3))
```

Out[74]:

```
<AxesSubplot:>
```



11. gear

In [75]:

```
df['gear'].describe()
```

Out[75]:

```
count    32.000000
mean      3.687500
std       0.737804
min       3.000000
25%       3.000000
50%       4.000000
75%       4.000000
max       5.000000
Name: gear, dtype: float64
```

In [76]:

```
df['gear'].skew()
```

Out[76]:

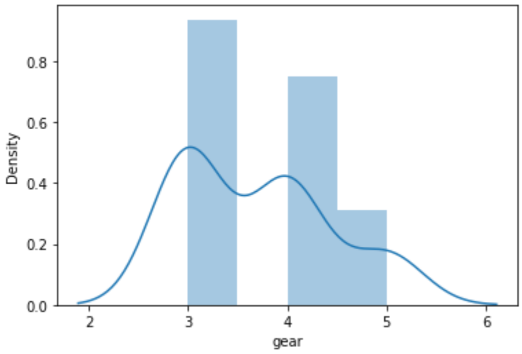
```
0.5823085692540974
```

In [77]:

```
sns.distplot(df['gear'])
```

Out[77]:

```
<AxesSubplot:xlabel='gear', ylabel='Density'>
```

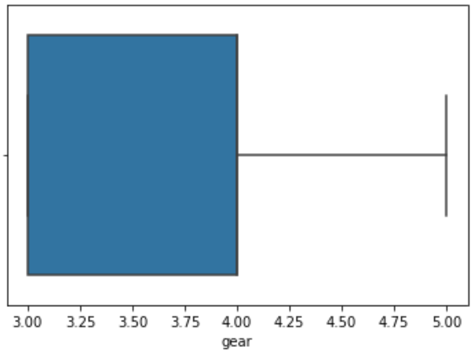


In [78]:

```
sns.boxplot(df['gear'])
```

Out[78]:

```
<AxesSubplot:xlabel='gear'>
```



In [79]:

```
print('unique categories--->',df['gear'].unique())
#counting the uniques
print('value_counts for each unique categories---->',df['gear'].value_counts())
print('Null value--->',df['gear'].isnull().sum())
```

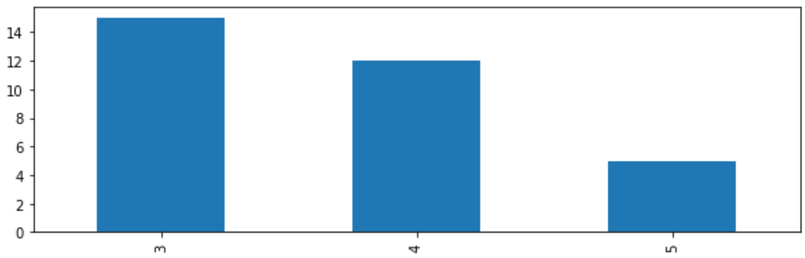
```
unique categories---> [4 3 5]
value_counts for each unique categories----> 3    15
4    12
5     5
Name: gear, dtype: int64
Null value---> 0
```

In [80]:

```
df['gear'].value_counts().plot(kind='bar',figsize=(10, 3))
```

Out[80]:

```
<AxesSubplot:>
```



12. carb

In [81]:

```
1 df['carb'].describe()
```

Out[81]:

```
count    32.0000
mean      2.8125
std       1.6152
min       1.0000
25%       2.0000
50%       2.0000
75%       4.0000
max       8.0000
Name: carb, dtype: float64
```

In [82]:

```
1 df['carb'].skew()
```

Out[82]:

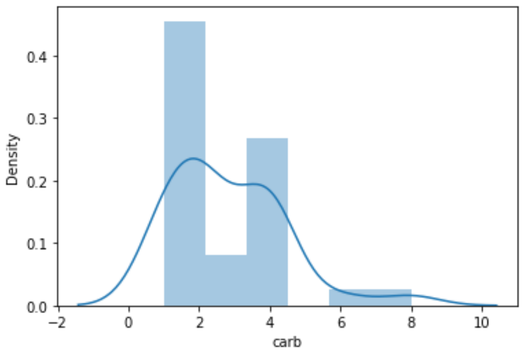
```
1.1570911127381494
```

In [83]:

```
1 sns.distplot(df['carb'])
```

Out[83]:

```
<AxesSubplot:xlabel='carb', ylabel='Density'>
```

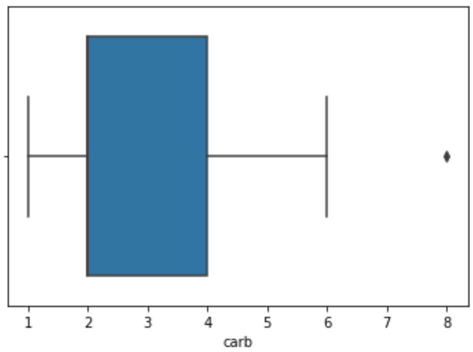


In [85]:

```
1 sns.boxplot(df['carb'])
```

Out[85]:

```
<AxesSubplot:xlabel='carb'>
```



In [87]:

```
print('unique categories--->',df['carb'].unique())
#counting the uniques
print('value_counts for each unique categories---->',df['carb'].value_counts())
print('Null value--->',df['carb'].isnull().sum())
```

```
unique categories---> [4 1 2 3 6 8]
value_counts for each unique categories----> 4    10
2    10
1     7
3     3
6     1
8     1
Name: carb, dtype: int64
Null value---> 0
```

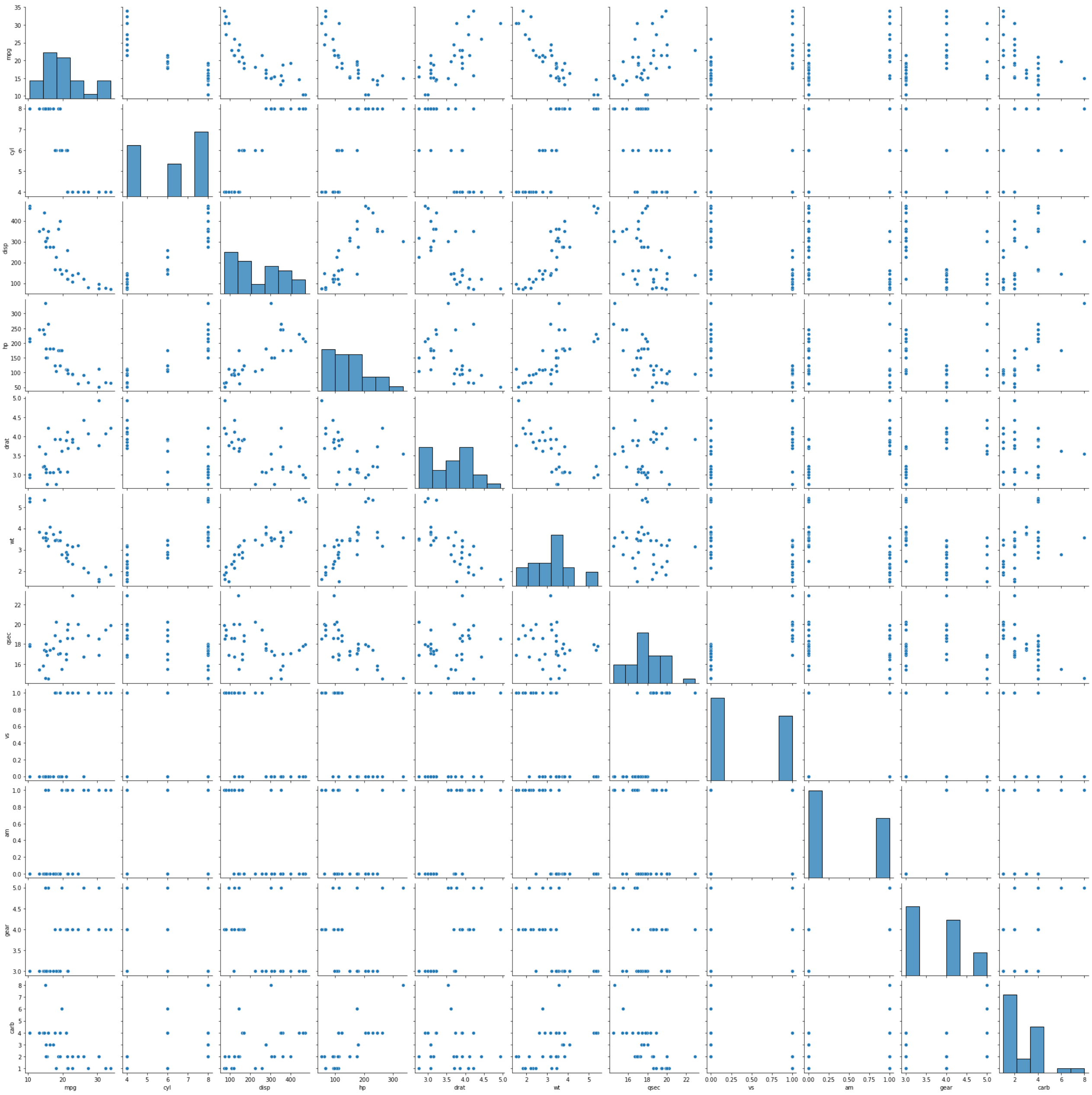
6. Checking the corr with each other

In [88]:

```
sns.pairplot(df)
```

Out[88]:

```
<seaborn.axisgrid.PairGrid at 0x263394521f0>
```



In [90]:

```
df.corr()
```

Out[90]:

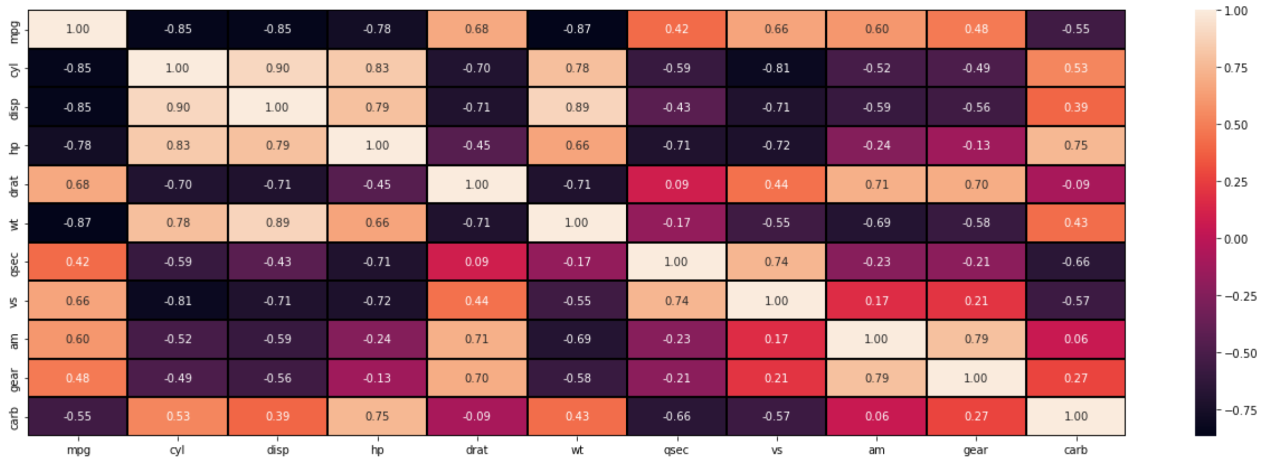
	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
mpg	1.000000	-0.852162	-0.847551	-0.776168	0.681172	-0.867659	0.418684	0.664039	0.599832	0.480285	-0.550925
cyl	-0.852162	1.000000	0.902033	0.832447	-0.699938	0.782496	-0.591242	-0.810812	-0.522607	-0.492687	0.526988
disp	-0.847551	0.902033	1.000000	0.790949	-0.710214	0.887980	-0.433698	-0.710416	-0.591227	-0.555569	0.394977
hp	-0.776168	0.832447	0.790949	1.000000	-0.448759	0.658748	-0.708223	-0.723097	-0.243204	-0.125704	0.749812
drat	0.681172	-0.699938	-0.710214	-0.448759	1.000000	-0.712441	0.091205	0.440278	0.712711	0.699610	-0.090790
wt	-0.867659	0.782496	0.887980	0.658748	-0.712441	1.000000	-0.174716	-0.554916	-0.692495	-0.583287	0.427606
qsec	0.418684	-0.591242	-0.433698	-0.708223	0.091205	-0.174716	1.000000	0.744535	-0.229861	-0.212682	-0.656249
vs	0.664039	-0.810812	-0.710416	-0.723097	0.440278	-0.554916	0.744535	1.000000	0.168345	0.206023	-0.569607
am	0.599832	-0.522607	-0.591227	-0.243204	0.712711	-0.692495	-0.229861	0.168345	1.000000	0.794059	0.057534
gear	0.480285	-0.492687	-0.555569	-0.125704	0.699610	-0.583287	-0.212682	0.206023	0.794059	1.000000	0.274073
carb	-0.550925	0.526988	0.394977	0.749812	-0.090790	0.427606	-0.656249	-0.569607	0.057534	0.274073	1.000000

In [95]:

```
plt.figure(figsize=(22,7)) # ploting the heat map
sns.heatmap(df.corr(),annot=True,linewidths=0.1,linecolor="black",fmt="0.2f")
```

Out[95]:

```
<AxesSubplot:>
```



with the following function we can select highly correlated features

In [96]:

```
# with the following function we can select highly correlated features
# it will remove the first feature that is correlated with anything other feature

def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

In [101]:

```
corr_features = correlation(df, 0.8)
len(set(corr_features))
```

Out[101]:

5

In [103]:

```
corr_features # 80% highly correlated features name
```

Out[103]:

```
{'cyl', 'disp', 'hp', 'vs', 'wt'}
```

all the observations are listed below with respect to what we have seen up to

Column Name	dtypes	type of variable	Observations		Remarks
name	object	Nominal	Model of Vehicle		
mpg	float64	continoues	Miles/US Gallon		few outlires
cyl	int64	quantitative	Number of cylinders		4 , 6 ,8
disp	float64	continoues	Displacement		No outlires
hp	int64	quantitative	horsepower	Slightly right skew	few outlires
drat	float64	continoues	Rear axle ratio		No outlires
wt	float64	continoues	The overall weight of the vehicle		few outlires
qsec	float64	continoues	A performance measure		few outlires
vs	int64	Nominal / Binary	Yes or No (0 , 1)		
am	int64	Nominal / Binary	Transmission Type		
gear	int64	quantitative	Number of forward gears	3 , 4, 5	
carb	int64	quantitative	Number of carburetors		few outlires

7. Outliers treatment

There are several ways to treat outliers when working with data.

- 1) delete the outlires

```
Z-score method
IQR method
```

- 2) Transforming the outlires

```
log or the square root, can help reduce the impact of outliers
```

Note:- we have small data set so are not goint to delete any data so we will go with transformation technique

In [105]:

```
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer (method='box-cox')
df['mpg'] = scaler.fit_transform(df[['mpg']].values)
```

In [106]:

```
df['mpg'].skew()
```

Out[106]:

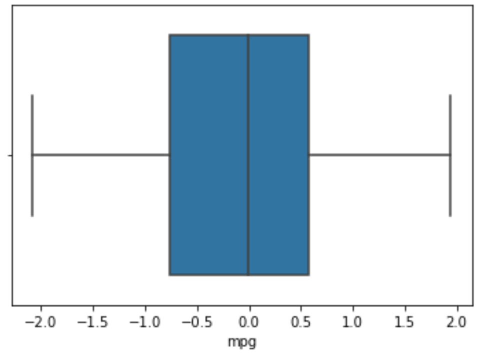
```
-0.0010867846440279492
```

In [107]:

```
sns.boxplot(df['mpg'])
```

Out[107]:

<AxesSubplot:xlabel='mpg'>

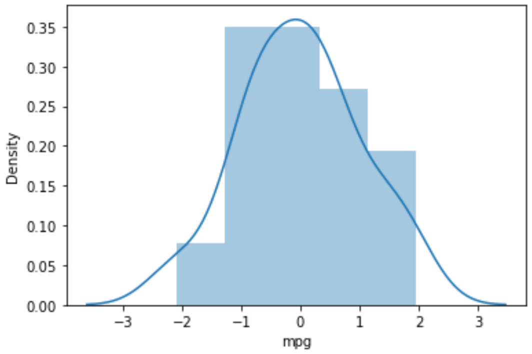


In [108]:

```
sns.distplot(df['mpg'])
```

Out[108]:

<AxesSubplot:xlabel='mpg', ylabel='Density'>



In [110]:

```
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer (method='box-cox')
df['hp'] = scaler.fit_transform(df[['hp']].values)
```

In [111]:

```
df['hp'].skew()
```

Out[111]:

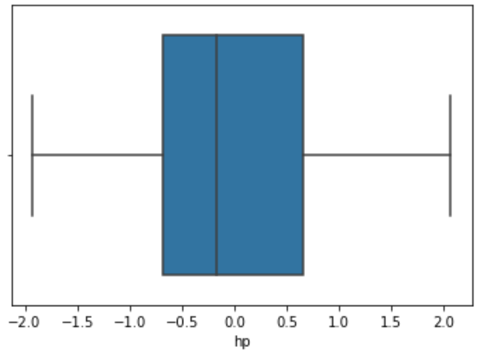
-0.012452153864081352

In [112]:

```
sns.boxplot(df['hp'])
```

Out[112]:

<AxesSubplot:xlabel='hp'>

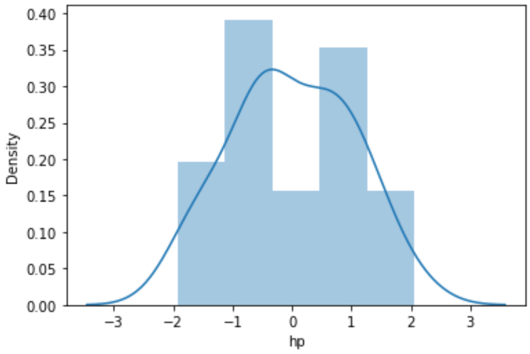


In [113]:

```
sns.distplot(df['hp'])
```

Out[113]:

<AxesSubplot:xlabel='hp', ylabel='Density'>



In [114]:

```
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer (method='box-cox')
df['wt'] = scaler.fit_transform(df[['wt']].values)
```

In [115]:

```
df['wt'].skew()
```

Out[115]:

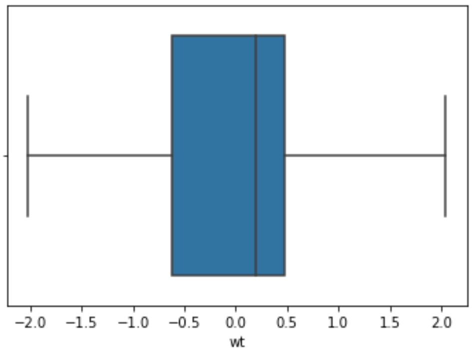
-0.010671914628154918

In [116]:

```
sns.boxplot(df['wt'])
```

Out[116]:

<AxesSubplot:xlabel='wt'>

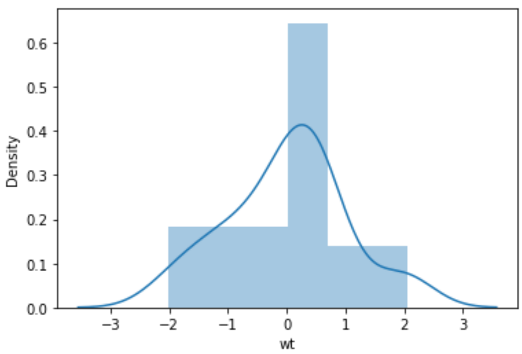


In [117]:

```
sns.distplot(df['wt'])
```

Out[117]:

<AxesSubplot:xlabel='wt', ylabel='Density'>



In [120]:

```
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer (method='box-cox')
df['qsec'] = scaler.fit_transform(df[['qsec']].values)
```

In [121]:

```
df['qsec'].skew()
```

Out[121]:

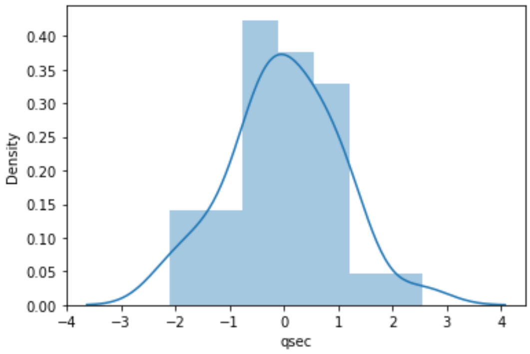
```
-0.0008853731632667843
```

In [123]:

```
sns.distplot(df['qsec'])
```

Out[123]:

```
<AxesSubplot:xlabel='qsec', ylabel='Density'>
```



In [125]:

```
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer (method='box-cox')
df['carb'] = scaler.fit_transform(df[['carb']].values)
```

In [126]:

```
df['carb'].skew()
```

Out[126]:

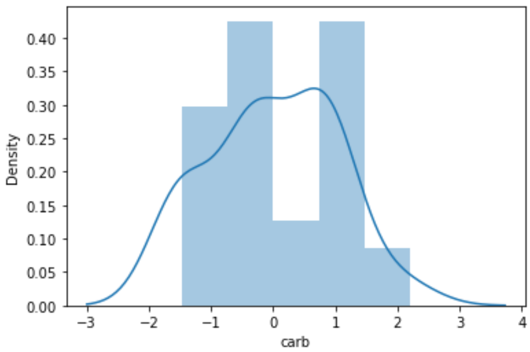
```
-0.01981389351395861
```

In [127]:

```
sns.distplot(df['carb'])
```

Out[127]:

```
<AxesSubplot:xlabel='carb', ylabel='Density'>
```

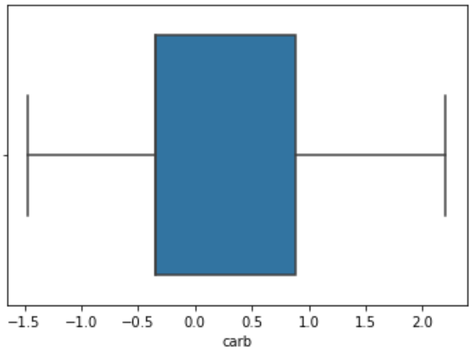


In [128]:

```
sns.boxplot(df['carb'])
```

Out[128]:

```
<AxesSubplot:xlabel='carb'>
```



8. It is time to convert categorical_columns into numrical value using get_dummies encoding technique

In [130]:

```
df_get_dummies = pd.get_dummies(df['name'], drop_first=True)

```

In [131]:

```
df_get_dummies
```

Out[131]:

	Cadillac Fleetwood	Camaro Z28	Chrysler Imperial	Datsun 710	Dodge Challenger	Duster 360	Ferrari Dino	Fiat 128	Fiat X1-9	Ford Pantera L	...	Merc 280C	Merc 450SE	Merc 450SL	Merc 450SLC	Pontiac Firebird	Porsche 914-2	Toy Corc
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
2	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
6	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0	0	0	
13	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	
14	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
15	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
16	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
17	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	
18	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
19	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
21	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	
22	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
23	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
24	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	
25	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	
26	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1	
27	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
28	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	
29	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	
30	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
31	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

32 rows × 31 columns

Concatenation

In [134]:

```
final = dfcat2 = pd.concat([df,df_get_dummies],axis=1).drop (columns = ['name'])
```

In [135]:

```
final
```

Out[135]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	...	Merc 280C	Merc 450SE	Merc 450SL	Merc 450SLC	Pontiac Firebird	Porsche 914-2	Toyota Corolla	Toyota Corona
0	0.292935	6	160.0	-0.406998	3.90	-0.569745	-0.774404	0	1	4	...	0	0	0	0	0	0	0	0
1	0.292935	6	160.0	-0.406998	3.90	-0.283616	-0.429244	0	1	4	...	0	0	0	0	0	0	0	0
2	0.574610	4	108.0	-0.757411	3.85	-0.925952	0.482194	1	1	4	...	0	0	0	0	0	0	0	0
3	0.357501	6	258.0	-0.406998	3.08	0.077733	0.922273	1	0	3	...	0	0	0	0	0	0	0	0
4	-0.103212	8	360.0	0.591461	3.15	0.305833	-0.429244	0	0	3	...	0	0	0	0	0	0	0	0
5	-0.214339	6	225.0	-0.504653	2.76	0.325721	1.316135	1	0	3	...	0	0	0	0	0	0	0	0
6	-1.014170	8	360.0	1.342775	3.21	0.434035	-1.173048	0	0	3	...	0	0	0	0	0	0	0	0
7	0.807426	4	146.7	-1.581244	3.69	0.051874	1.206881	1	0	4	...	0	0	0	0	0	0	0	0
8	0.574610	4	140.8	-0.713305	3.92	0.010275	2.544327	1	0	4	...	0	0	0	0	0	0	0	0
9	-0.013218	6	167.6	-0.170756	3.92	0.305833	0.311865	1	0	4	...	0	0	0	0	0	0	0	0
10	-0.271250	6	167.6	-0.170756	3.92	0.305833	0.638538	1	0	4	...	1	0	0	0	0	0	0	0
11	-0.549781	8	275.8	0.653456	3.07	0.905418	-0.202555	0	0	3	...	0	1	0	0	0	0	0	0
12	-0.368203	8	275.8	0.653456	3.07	0.588476	-0.085573	0	0	3	...	0	0	1	0	0	0	0	0
13	-0.807542	8	275.8	0.653456	3.07	0.636017	0.143764	0	0	3	...	0	0	0	1	0	0	0	0
14	-2.086223	8	472.0	0.941800	2.93	1.908410	0.132440	0	0	3	...	0	0	0	0	0	0	0	0
15	-2.086223	8	460.0	1.048283	3.00	2.045770	0.041312	0	0	3	...	0	0	0	0	0	0	0	0
16	-0.920821	8	440.0	1.199882	3.23	1.983699	-0.190785	0	0	3	...	0	0	0	0	0	0	0	0
17	1.785932	4	78.7	-1.456255	4.08	-1.075234	0.937764	1	1	4	...	0	0	0	0	0	0	0	0
18	1.565359	4	75.7	-1.928963	4.93	-1.873787	0.433090	1	1	4	...	0	0	0	0	0	0	0	0
19	1.942855	4	71.1	-1.486845	4.22	-1.557899	1.156751	1	1	4	...	0	0	0	0	0	0	1	0
20	0.373460	4	120.1	-0.670033	3.70	-0.750921	1.211877	1	0	3	...	0	0	0	0	0	0	0	1
21	-0.741297	8	318.0	0.255121	2.76	0.385024	-0.520372	0	0	3	...	0	0	0	0	0	0	0	0
22	-0.807542	8	304.0	0.255121	3.15	0.300851	-0.261641	0	0	3	...	0	0	0	0	0	0	0	0
23	-1.259109	8	350.0	1.342775	3.73	0.692629	-1.460472	0	0	3	...	0	0	0	0	0	0	0	0
24	-0.013218	8	400.0	0.591461	3.08	0.697326	-0.411132	0	0	3	...	0	0	0	0	1	0	0	0
25	1.193960	4	79.0	-1.456255	4.08	-1.420961	0.638538	1	1	4	...	0	0	0	0	0	0	0	0
26	1.025873	4	120.3	-0.802385	4.43	-1.151492	-0.624811	0	1	5	...	0	0	0	0	0	1	0	0
27	1.565359	4	95.1	-0.350319	3.77	-2.027975	-0.502070	1	1	5	...	0	0	0	0	0	0	0	0
28	-0.676286	8	351.0	1.512805	4.22	0.031109	-2.101151	0	1	5	...	0	0	0	0	0	0	0	0
29	0.074530	6	145.0	0.591461	3.62	-0.399738	-1.399539	0	1	5	...	0	0	0	0	0	0	0	0
30	-0.852413	8	301.0	2.063061	3.54	0.434035	-2.028463	0	1	5	...	0	0	0	0	0	0	0	0
31	0.357501	4	121.0	-0.426203	4.11	-0.388580	0.476752	1	1	4	...	0	0	0	0	0	0	0	0

32 rows × 42 columns

3. Store data in list

In [136]:

```
for name in final.columns:
    print(final[name].tolist())
```

```
[0.2929351253577551, 0.2929351253577551, 0.5746098318820518, 0.35750127255724595, -0.10321166088799767, -0.214338727960
8435, -1.014170056151736, 0.8074256905772834, 0.5746098318820518, -0.013217840429793145, -0.27124998378159987, -0.54978
05347822231, -0.3682034429291773, -0.80754167721729, -2.086223021908872, -2.086223021908872, -0.9208211071659261, 1.785
9321821150813, 1.5653589251128024, 1.9428553375743811, 0.37345978504453786, -0.7412973556616481, -0.80754167721729, -1.
2591093694122153, -0.013217840429793145, 1.1939601781377287, 1.0258730096771351, 1.5653589251128024, -0.676285776275955
8, 0.0745298045207287, -0.8524132033453758, 0.35750127255724595]
[6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, 4, 4, 4, 8, 8, 8, 8, 4, 4, 4, 8, 6, 8, 4]
[160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.7, 140.8, 167.6, 167.6, 275.8, 275.8, 275.8, 472.0, 460.0, 440.0,
78.7, 75.7, 71.1, 120.1, 318.0, 304.0, 350.0, 400.0, 79.0, 120.3, 95.1, 351.0, 145.0, 301.0, 121.0]
[-0.4069982196873767, -0.4069982196873767, -0.7574112195799817, -0.4069982196873767, 0.5914611426867292, -0.50465320914
5498, 1.3427752932915298, -1.5812441398208652, -0.7133052340895865, -0.1707557854491838, -0.1707557854491838, 0.6534559
301645825, 0.6534559301645825, 0.6534559301645825, 0.941800104226914, 1.0482833737349977, 1.1998822453770968, -1.456254
5020663271, -1.9289629940436823, -1.486844864609322, -0.6700325399545136, 0.25512110995404846, 0.25512110995404846, 1.3
427752932915298, 0.5914611426867292, -1.4562545020663271, -0.8023849908567442, -0.3503185533326177, 1.5128046931260561,
0.5914611426867292, 2.063061438348054, -0.4262029003322603]
[3.9, 3.9, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.92, 3.92, 3.07, 3.07, 3.07, 2.93, 3.0, 3.23, 4.08, 4.93, 4.22,
3.7, 2.76, 3.15, 3.73, 3.08, 4.08, 4.43, 3.77, 4.22, 3.62, 3.54, 4.11]
[-0.5697447297958302, -0.28361648965001346, -0.9259516732409199, 0.0777330769902653, 0.30583289443254214, 0.32572116284
960473, 0.4340349426557202, 0.0518738030439501, 0.010274885537164151, 0.30583289443254214, 0.30583289443254214, 0.90541
78281155205, 0.5884761355651972, 0.6360172602690534, 1.9084099802973915, 2.045769861869233, 1.9836988409392011, -1.0752
339663030668, -1.873786636966715, -1.5578993662716913, -0.7509213258165657, 0.38502362485089253, 0.300851272768759, 0.6
926294340421865, 0.697325917750486, -1.4209606601790352, -1.151492395752291, -2.027975135207327, 0.03110906703296682, -
0.3997384173173148, 0.4340349426557202, -0.3885799240301686]
[-0.774403763817452, -0.4292436944400186, 0.48219444773593517, 0.9222734017067431, -0.4292436944400186, 1.3161345843735
24, -1.1730475304385872, 1.2068807686380938, 2.5443265646517497, 0.3118646586998877, 0.6385383403065052, -0.20255470268
187153, -0.0855732304622551 8, 0.14376389486210928, 0.1324400810915003, 0.04131236963504837, -0.19078549635506675, 0.937
7636703522756, 0.4330899614260483, 1.156751256834018, 1.2118774860494612, -0.5203718655612768, -0.26164057281824266, -
1.4604724070032207, -0.4111315062714526, 0.6385383403065052, -0.6248106469130387, -0.5020701139349922, -2.101151324143
1, -1.3995386753102377, -2.028462815611455, 0.47675221353281394]
[0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1]
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
[4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, 4, 3, 3, 3, 3, 3, 4, 5, 5, 5, 5, 5, 4]
[0.8796730039159774, 0.8796730039159774, -1.4709363368619581, -1.4709363368619581, -0.34269361785973795, -1.47093633686
19581, 0.8796730039159774, -0.34269361785973795, -0.34269361785973795, 0.8796730039159774, 0.8796730039159774, 0.360429
94940033024, 0.36042994940033024, 0.36042994940033024, 0.8796730039159774, 0.8796730039159774, 0.8796730039159774, -1.4
709363368619581, -0.34269361785973795, -1.4709363368619581, -1.4709363368619581, -0.34269361785973795, -0.3426936178597
3795, 0.8796730039159774, -0.34269361785973795, -1.4709363368619581, -0.34269361785973795, -0.34269361785973795, 0.8796
730039159774, 1.6414548130311926, 2.2040158362391304, -0.34269361785973795]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
```

In [138]:

```
store_in_list=[final[name].tolist() for name in final.columns]
```

In [139]:

```
store_in_list
```

Out[139]:

```
[[0.2929351253577551,
  0.2929351253577551,
  0.5746098318820518,
  0.35750127255724595,
  -0.10321166088799767,
  -0.2143387279608435,
  -1.014170056151736,
  0.8074256905772834,
  0.5746098318820518,
  -0.013217840429793145,
  -0.27124998378159987,
  -0.5497805347822231,
  -0.3682034429291773,
  -0.80754167721729,
  -2.086223021908872,
  -2.086223021908872,
  -0.9208211071659261,
  1.7859321821150813,
```

In [ ]:

```

```