Advertising.csv dataset

In [1]:
```
#Basic information (datatypes)
#Column name
#data preprocessing
#Statistical summary
#Check for the null values
#Check for the duplicate values
#Check for coorelation
#Feature scaling
#VIF - Variance Inflation Factor - to check multicollinearity
#Building Linear Regression Model
#OLS model
#Check linearity
#Normality of Residual
#Lasso regularization
#Ridge Regression
#ElasticNet
```

Importing the libraries

In [2]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

import warnings
```

In [3]:
```
df = pd.read_csv('Advertising.csv')
```

In [4]:
```
df.head()
```

Out[4]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

To find imformation about dataset

In [5]:
```
df.shape
```

Out[5]:
```
(200, 4)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   TV         200 non-null    float64
 1   radio      200 non-null    float64
 2   newspaper  200 non-null    float64
 3   sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

In [7]:
```
# In this dataset all are numerical values
```

In [8]:
```
df.describe()
```

Out[8]:

	TV	radio	newspaper	sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	14.022500
std	85.854236	14.846809	21.778621	5.217457
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	10.375000
50%	149.750000	22.900000	25.750000	12.900000
75%	218.825000	36.525000	45.100000	17.400000
max	296.400000	49.600000	114.000000	27.000000

In [9]:
```
df.columns
```

Out[9]:
```
Index(['TV', 'radio', 'newspaper', 'sales'], dtype='object')
```

In [10]:
```
df['TV'].describe()
```

Out[10]:
```
count    200.000000
mean     147.042500
std       85.854236
min        0.700000
25%       74.375000
50%      149.750000
75%      218.825000
max      296.400000
Name: TV, dtype: float64
```

Data Preprocessing

In [11]:
```
# Missing value treatment
```

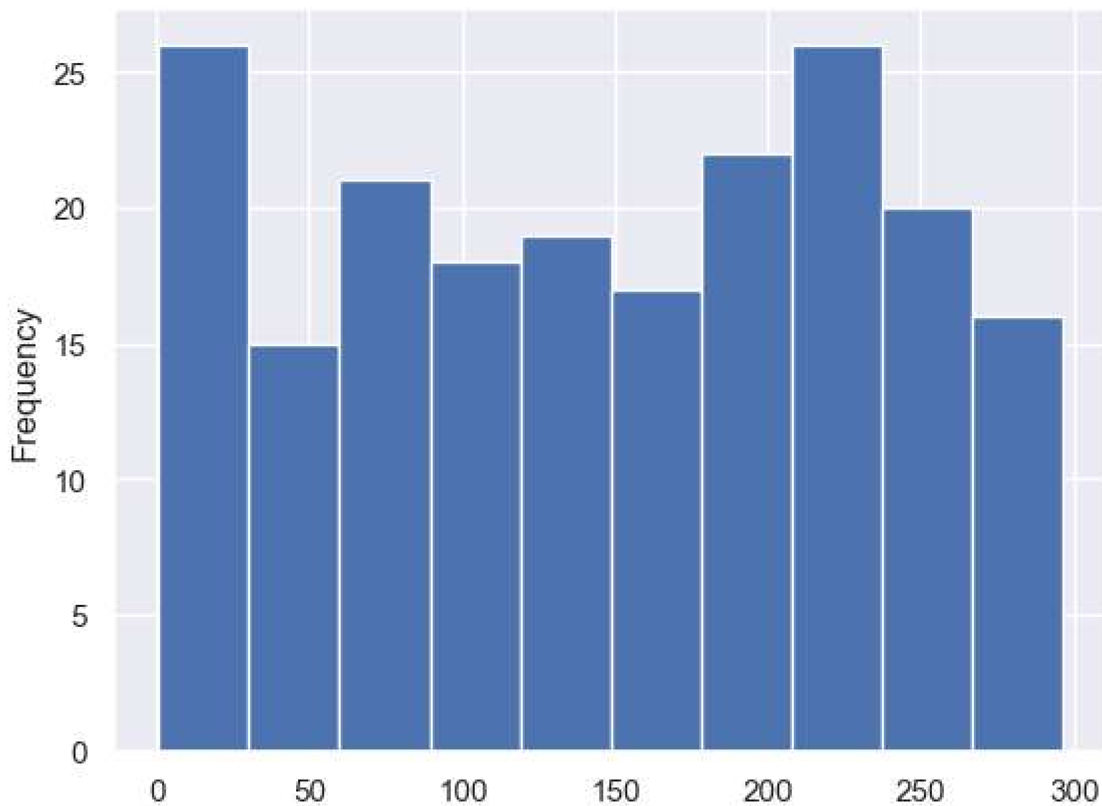
In [12]:
```
df.isnull().sum()
```

Out[12]:
```
TV           0
radio        0
newspaper    0
sales        0
dtype: int64
```

In [13]:
```
# There is no missing values
```

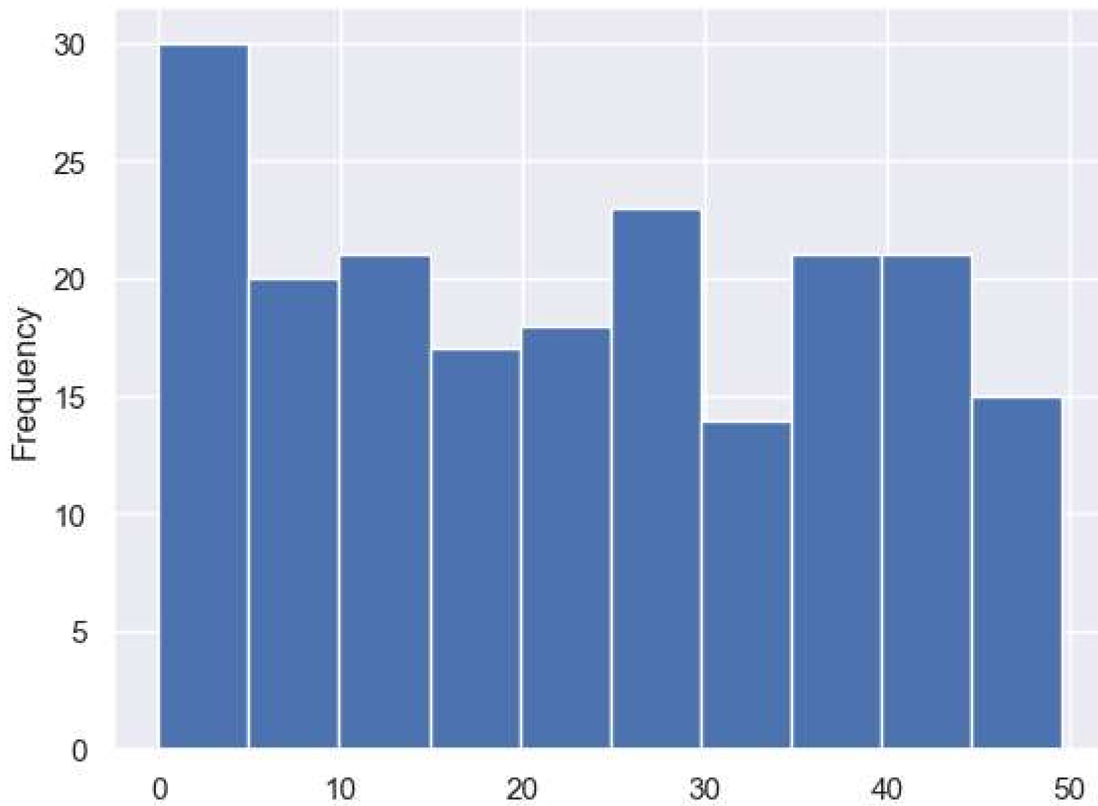
In [14]:
```
df['TV'].plot(kind='hist')
```

Out[14]:
```
<Axes: ylabel='Frequency'>
```



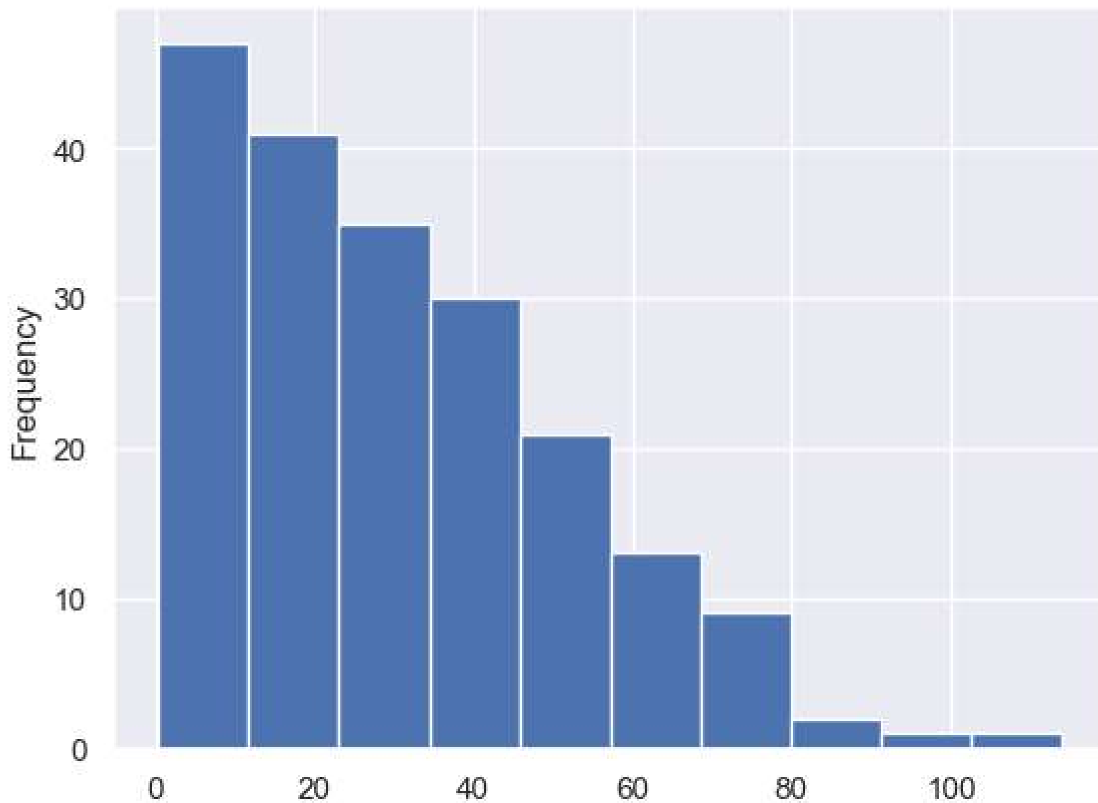
In [15]:
```
df['radio'].plot(kind='hist')
```

Out[15]: <Axes: ylabel='Frequency'>



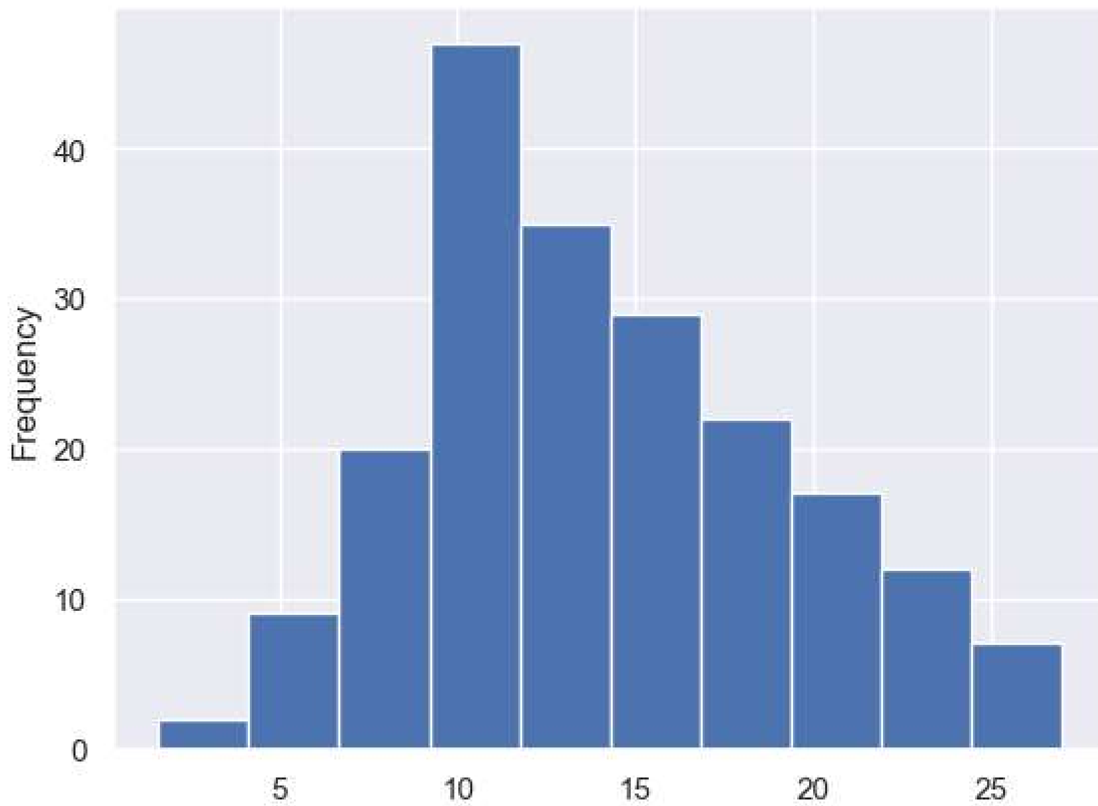
In [16]:
```
df['newspaper'].plot(kind='hist')
```

Out[16]: <Axes: ylabel='Frequency'>



In [17]:
```
df['sales'].plot(kind='hist')
```

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

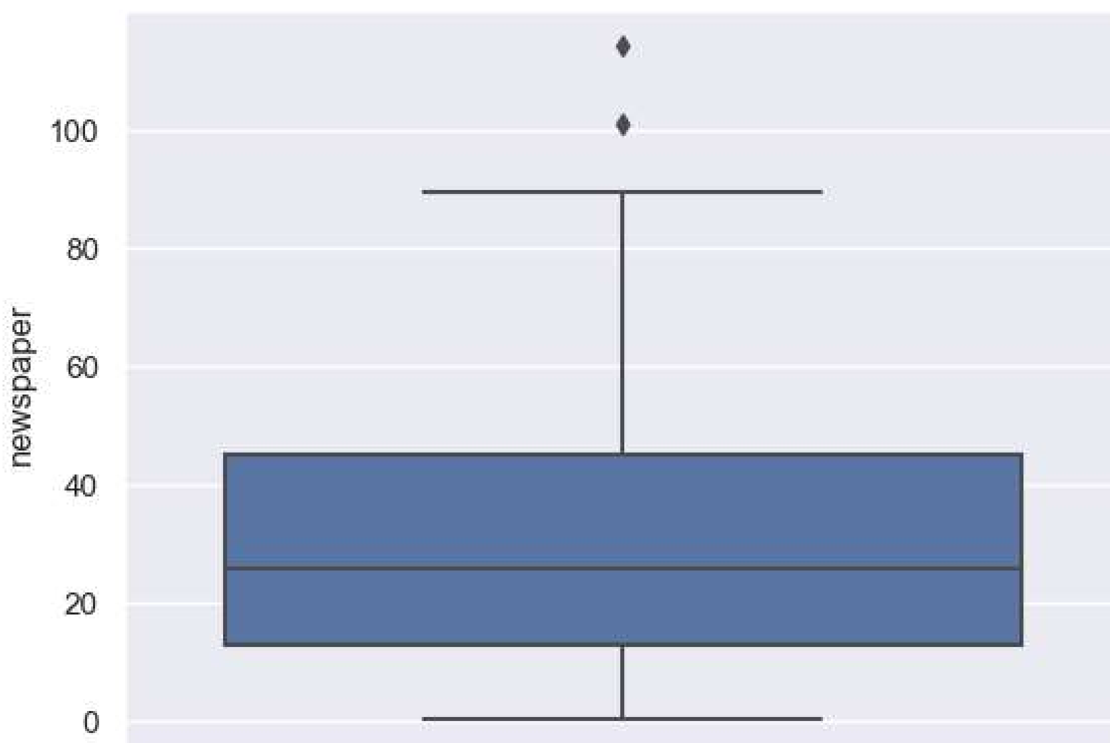


In [18]:
```
sns.boxplot(y = 'TV', data=df)
```

In [ ]:

In [19]:
```
sns.boxplot(y = 'radio', data=df)
```

In [20]:
```
sns.boxplot(y = 'newspaper', data=df)
```



In [21]:
```
df['newspaper'].describe()
```

Out[21]:
```
count    200.000000
mean      30.554000
std       21.778621
min        0.300000
25%       12.750000
50%       25.750000
75%       45.100000
max      114.000000
Name: newspaper, dtype: float64
```

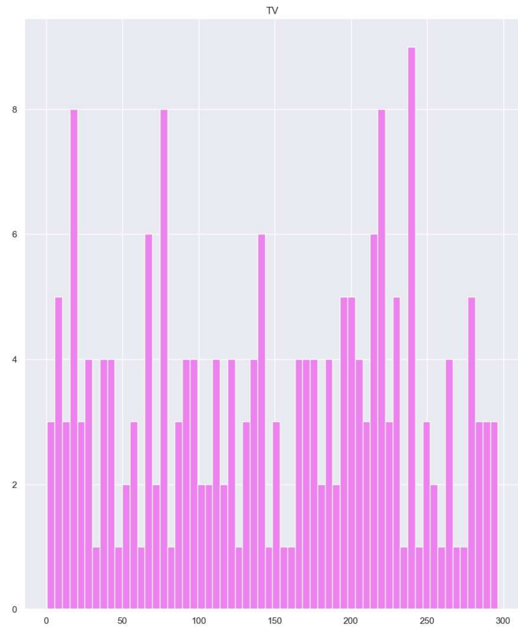
In [22]:
```
df['newspaper'] = df['newspaper'].fillna(df['newspaper'].median())
```

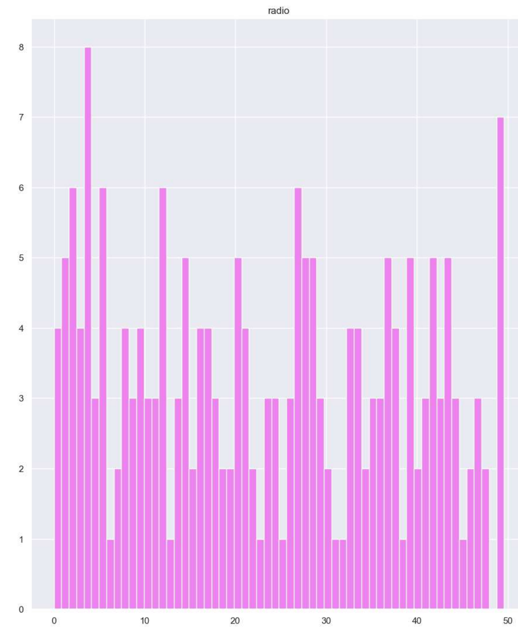
In [ ]:

In [23]:
```
sns.boxplot(y = 'sales', data=df)
```

In [ ]:

In [24]:
```
df.hist(bins=60,figsize=(25,30),color='violet')
```





In [25]:
```
for i in df.columns:
    print("*********************************************************",i,
          "*******************************************************")
    print()
    print(set(df[i].tolist()))
    print()
```

```
******************************************************** TV ****************
**************************************

{0.7, 4.1, 5.4, 7.3, 8.7, 8.6, 7.8, 8.4, 11.7, 13.2, 13.1, 16.9, 17.2, 18.8,
19.4, 17.9, 19.6, 18.7, 23.8, 25.1, 26.8, 27.5, 28.6, 25.0, 25.6, 31.5, 36.9,
38.0, 39.5, 38.2, 43.1, 44.5, 43.0, 44.7, 48.3, 50.0, 53.5, 56.2, 57.5, 215.
4, 59.6, 62.3, 66.1, 67.8, 66.9, 69.2, 70.6, 69.0, 68.4, 73.4, 74.7, 75.3, 7
6.4, 76.3, 78.2, 75.1, 80.2, 75.5, 85.7, 87.2, 88.3, 89.7, 90.4, 93.9, 94.2,
95.7, 96.2, 97.5, 97.2, 100.4, 102.7, 104.6, 107.4, 109.8, 110.7, 112.9, 116.
0, 117.2, 120.5, 120.2, 121.0, 123.1, 125.7, 129.4, 131.1, 131.7, 134.3, 135.
2, 136.2, 137.9, 139.3, 139.2, 141.3, 142.9, 140.3, 139.5, 147.3, 149.8, 149.
7, 151.5, 156.6, 163.3, 163.5, 164.5, 165.6, 166.8, 168.4, 170.2, 171.3, 172.
5, 175.1, 175.7, 177.0, 180.8, 182.6, 184.9, 187.9, 187.8, 188.4, 191.1, 193.
2, 193.7, 195.4, 197.6, 198.9, 199.8, 199.1, 202.5, 204.1, 205.0, 206.9, 206.
8, 209.6, 210.8, 210.7, 213.4, 214.7, 213.5, 216.4, 216.8, 218.4, 217.7, 220.
3, 219.8, 222.4, 220.5, 224.0, 225.8, 218.5, 227.2, 228.3, 228.0, 230.1, 229.
5, 232.1, 234.5, 237.4, 238.2, 239.9, 240.1, 239.3, 239.8, 241.7, 243.2, 248.
8, 248.4, 250.9, 253.8, 255.4, 261.3, 262.9, 262.7, 265.6, 266.9, 265.2, 273.
7, 276.9, 276.7, 280.2, 281.4, 280.7, 283.6, 284.3, 286.0, 287.6, 289.7, 290.
7, 292.9, 293.6, 296.4}

******************************************************** radio *************
*******************************************

{0.8, 1.5, 2.1, 2.6, 3.5, 5.8, 5.1, 7.6, 1.4, 4.1, 10.8, 8.4, 12.6, 9.9, 11.
7, 15.9, 16.9, 16.7, 16.0, 19.6, 20.5, 17.4, 20.0, 23.9, 24.0, 22.3, 26.7, 2
7.7, 27.1, 29.3, 28.3, 25.7, 32.8, 32.9, 33.4, 35.1, 36.6, 37.8, 37.7, 39.3,
39.6, 41.3, 41.5, 43.8, 41.7, 45.9, 46.2, 47.7, 48.9, 49.4, 49.6, 42.7, 43.9,
44.5, 47.8, 46.4, 2.0, 11.0, 49.0, 10.0, 12.0, 14.5, 14.0, 15.5, 17.0, 18.4,
18.1, 20.6, 1.9, 20.9, 20.1, 21.0, 2.4, 2.9, 21.1, 22.5, 3.4, 23.6, 24.6, 25.
5, 25.9, 26.9, 27.5, 28.1, 28.5, 28.9, 29.6, 29.9, 29.5, 4.9, 30.6, 5.4, 31.
6, 0.0, 32.3, 33.0, 33.5, 33.2, 34.3, 34.6, 35.0, 35.4, 35.8, 35.6, 36.5, 36.
3, 36.9, 36.8, 37.6, 38.0, 38.9, 38.6, 13.9, 39.0, 39.7, 40.6, 40.3, 15.4, 2.
3, 41.1, 42.8, 42.3, 4.3, 1.3, 42.0, 43.7, 43.0, 43.5, 45.1, 7.3, 7.8, 46.8,
47.0, 0.4, 8.2, 9.3, 11.8, 14.3, 14.8, 14.7, 15.8, 3.7, 17.2, 5.7, 5.2, 19.2,
7.7, 20.3, 21.7, 21.3, 23.3, 25.8, 0.3, 26.8, 27.2, 28.8, 28.7, 30.2, 7.1, 1.
6, 8.6, 9.6, 3.1, 10.1, 10.6, 11.6, 12.1}

******************************************************** newspaper *********
**********************************************

{1.0, 1.8, 3.6, 4.0, 5.0, 2.2, 7.2, 7.4, 8.5, 9.3, 11.6, 12.6, 8.4, 10.2, 15.
9, 16.6, 9.4, 18.3, 19.1, 19.5, 21.2, 22.9, 23.5, 24.2, 18.5, 26.2, 26.4, 28.
9, 21.4, 27.3, 30.0, 31.6, 32.0, 31.5, 34.6, 35.1, 35.7, 36.8, 38.6, 38.7, 4
0.8, 39.6, 41.4, 43.2, 43.3, 45.1, 46.0, 45.7, 49.6, 49.9, 51.4, 52.9, 53.4,
54.7, 55.8, 11.0, 51.2, 58.5, 58.4, 58.7, 60.0, 59.0, 63.2, 56.5, 65.9, 65.7,
65.6, 59.7, 69.3, 69.2, 71.8, 72.3, 73.4, 74.2, 75.0, 75.6, 79.2, 16.0, 84.8,
17.9, 17.0, 17.6, 89.4, 18.4, 19.4, 19.6, 100.9, 20.5, 20.6, 21.6, 2.4, 22.0,
114.0, 23.1, 23.4, 25.6, 25.9, 5.5, 26.6, 27.4, 5.9, 5.4, 6.0, 6.4, 32.5, 33.
8, 33.0, 34.5, 34.4, 35.6, 35.2, 10.9, 36.9, 11.9, 37.7, 37.9, 12.4, 12.9, 3
7.0, 38.9, 41.8, 43.1, 5.3, 43.0, 5.8, 44.3, 45.9, 45.2, 9.0, 46.2, 9.5, 47.
4, 48.7, 49.8, 49.3, 50.4, 50.6, 50.5, 0.9, 52.7, 57.6, 8.7, 8.3, 9.2, 10.7,
1.7, 12.8, 13.8, 14.2, 14.8, 66.2, 3.2, 3.7, 5.7, 18.2, 19.3, 20.7, 20.3, 22.
3, 23.2, 23.7, 24.3, 0.3, 27.2, 29.7, 30.7, 31.7, 31.3, 8.1, 2.1, 13.1, 15.6}

******************************************************** sales *************
*******************************************
```

```
{1.6, 3.2, 4.8, 5.6, 5.5, 7.2, 8.6, 9.3, 10.4, 11.3, 11.8, 12.9, 13.2, 10.6,
9.2, 9.7, 17.4, 18.5, 19.0, 12.5, 22.1, 22.4, 24.4, 18.0, 18.9, 21.4, 25.4, 2
1.5, 23.2, 22.6, 23.7, 24.2, 27.0, 26.2, 7.0, 8.5, 8.0, 9.5, 10.5, 11.0, 11.
5, 12.0, 14.0, 14.5, 15.5, 15.0, 16.6, 16.0, 16.9, 16.1, 17.1, 17.0, 17.6, 1
8.4, 19.4, 19.6, 20.1, 25.5, 5.9, 6.9, 8.4, 9.4, 9.9, 10.9, 11.9, 11.4, 12.4,
13.4, 14.9, 14.4, 15.9, 5.3, 7.3, 8.8, 8.7, 10.7, 10.8, 10.3, 11.2, 11.7, 12.
8, 12.3, 12.2, 12.7, 13.3, 14.7, 14.8, 14.2, 15.7, 15.2, 15.3, 16.7, 17.2, 1
7.3, 5.7, 18.3, 6.7, 19.2, 19.8, 19.7, 20.7, 20.2, 20.8, 21.2, 21.7, 21.8, 2
2.3, 22.2, 23.8, 24.7, 6.6, 7.6, 8.1, 9.6, 10.1, 11.6, 12.6, 13.6, 14.6, 14.
1, 15.6}
```

Feature scaling

In [26]:
```
# we can only do with independent variable
```

In [27]:
```
x = df.iloc[:,0:-1]
```

In [28]:
```
x.head()
```

Out[28]:

	TV	radio	newspaper
0	230.1	37.8	69.2
1	44.5	39.3	45.1
2	17.2	45.9	69.3
3	151.5	41.3	58.5
4	180.8	10.8	58.4

In [29]:
```
y.head()
```

Out[29]:
```
0    22.1
1    10.4
2     9.3
3    18.5
4    12.9
Name: sales, dtype: float64
```

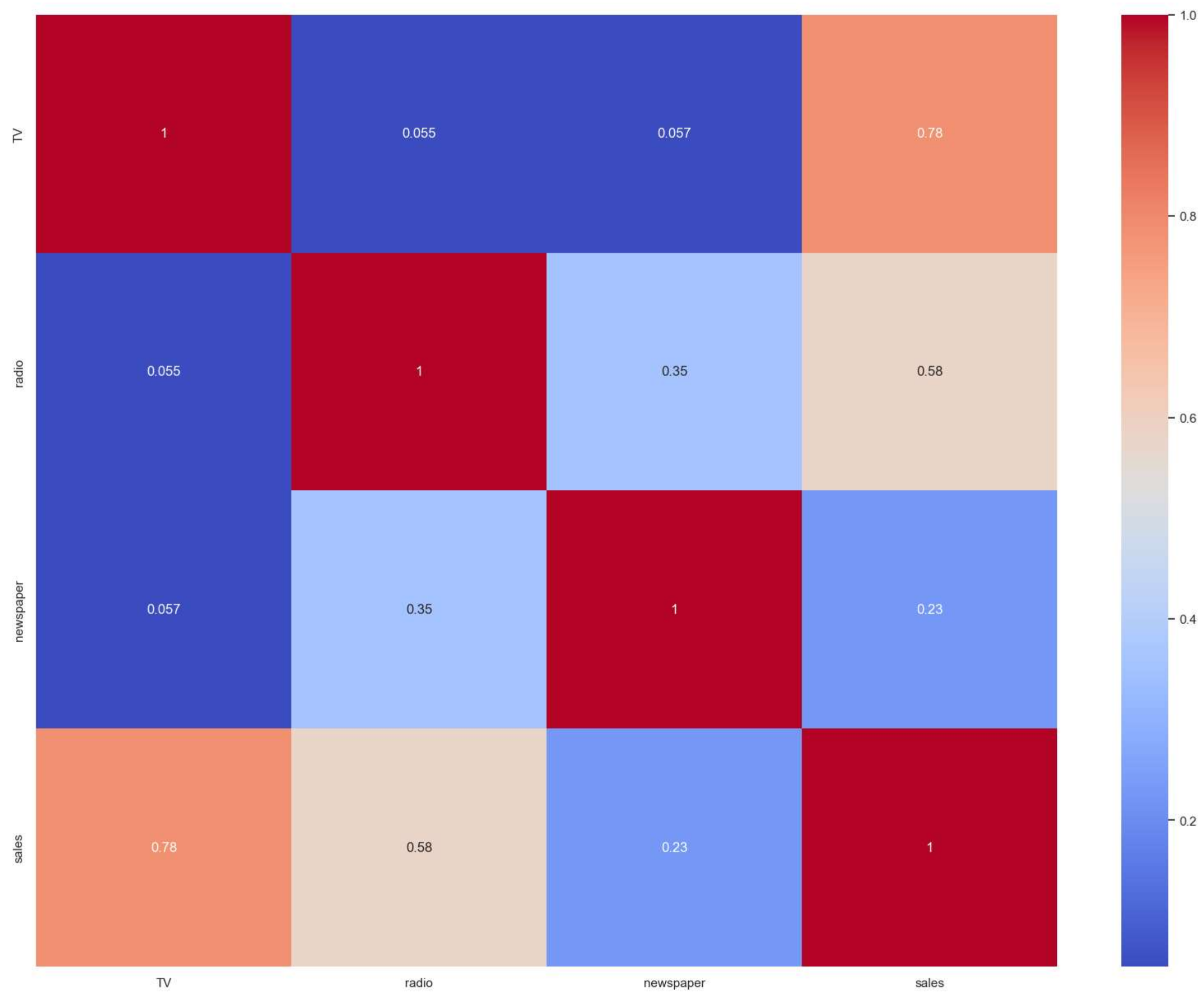
In [30]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc_x = sc.fit_transform(x)
```

Out[30]:

	0	1	2
0	0.969852	0.981522	1.778945
1	-1.197376	1.082808	0.669579
2	-1.516155	1.528463	1.783549
3	0.052050	1.217855	1.286405
4	0.394182	-0.841614	1.281802
...	...	...	...
195	-1.270941	-1.321031	-0.771217
196	-0.617035	-1.240003	-1.033598
197	0.349810	-0.942899	-1.111852
198	1.594565	1.265121	1.640850
199	0.993206	-0.990165	-1.005979

200 rows × 3 columns

In [31]:
```
# Finding correlation
plt.figure(figsize=(20,15))
corr = df.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
```



VIF - Variance Inflation Factor - to check multicollinearity

In [32]:
```
variable = sc_x
```

Out[32]: (200, 3)

In [33]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
variable = sc_x

vif = pd.DataFrame()

vif['Variance Inflation Factor'] = [variance_inflation_factor(variable, i ) for
```

In [34]:
```
vif
```

Out[34]:

	Variance Inflation Factor	Features
0	1.004611	TV
1	1.144952	radio
2	1.145187	newspaper

In [35]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, rando
```

```
(150, 3) (50, 3) (150,) (50,)
```

Building Linear Regression Model

In [36]:
```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
```

Out[36]:
```
▾ LinearRegression
LinearRegression()
```

In [37]:
```
print(lm.intercept_)
print()
```

```
2.9617274966628706

[0.04495318 0.19016471 0.000274  ]
```

In [38]:
```
x.columns
```

Out[38]:
```
Index(['TV', 'radio', 'newspaper'], dtype='object')
```

In [39]:
```
# Predict sales price by using lm model with test dataset

y_pred_price = lm.predict(x_test)
```

In [40]:
```
y_pred_price
```

Out[40]:
```
array([15.7263887 , 19.55970806, 11.35638413, 17.00154366,  9.05640744,
        6.88256478, 20.25202883, 17.23637795,  9.63243796, 19.19775722,
       12.33684688, 13.78919583, 13.60946471, 21.31349216, 18.42170403,
        9.88302868, 15.45083867,  7.53200526,  7.42033885, 20.3890307 ,
        7.66977854, 18.22207646, 24.71977128, 22.843015  ,  7.83227551,
       12.54236433, 21.42803762,  7.93472305, 12.31244402, 12.48247057,
       10.7244511 , 19.22531219,  9.93329519,  6.59231873, 17.28054591,
        7.62464387,  9.13268517,  8.13034377, 10.5171423 , 10.49809833,
       13.00081752,  9.63933072, 10.11131993,  7.94723108, 11.4796586 ,
        9.97587849,  8.89297513, 16.19336555, 13.1590433 , 20.83093062])
```

In [41]:
```
y_test
```

Out[41]:
```
37     14.7
109    19.8
31     11.9
89     16.7
66      9.5
119     6.6
54     20.2
74     17.0
145    10.3
142    20.1
148    10.9
112    14.1
174    11.5
55     23.7
141    19.2
149    10.1
25     12.0
34      9.5
170     8.4
39     21.5
172     7.6
153    19.0
175    27.0
61     24.2
65      9.3
50     11.4
42     20.7
129     9.7
179    12.6
2       9.3
12      9.2
133    19.6
90     11.2
22      5.6
41     17.1
32      9.6
125    10.6
196     9.7
158     7.3
180    10.5
16     12.5
186    10.3
144    11.4
121     7.0
80     11.8
18     11.3
78      5.3
48     14.8
4      12.9
15     22.4
Name: sales, dtype: float64
```

In [42]:
```
# Validate the actual price of the test data and predicted price

from sklearn.metrics import r2_score
```

Out[42]: 0.9246764680774093

In [43]:
```
r2_score(y_train , y_pred_price_train)
```

Out[43]: 0.8865137139252313

OLS method

In [44]:
```
from statsmodels.regression.linear_model import OLS
```

In [45]:
```
reg_model = smf.OLS(endog = y_train, exog=x_train).fit()
```

In [46]:
```
reg_model.summary()
```

Out[46]:

OLS Regression Results

Dep. Variable:	sales	**R-squared (uncentered):**	0.981
Model:	OLS	**Adj. R-squared (uncentered):**	0.981
Method:	Least Squares	**F-statistic:**	2596.
Date:	Mon, 31 Jul 2023	**Prob (F-statistic):**	4.65e-127
Time:	23:07:35	**Log-Likelihood:**	-321.35
No. Observations:	150	**AIC:**	648.7
Df Residuals:	147	**BIC:**	657.7
Df Model:	3		
Covariance Type:	nonrobust		

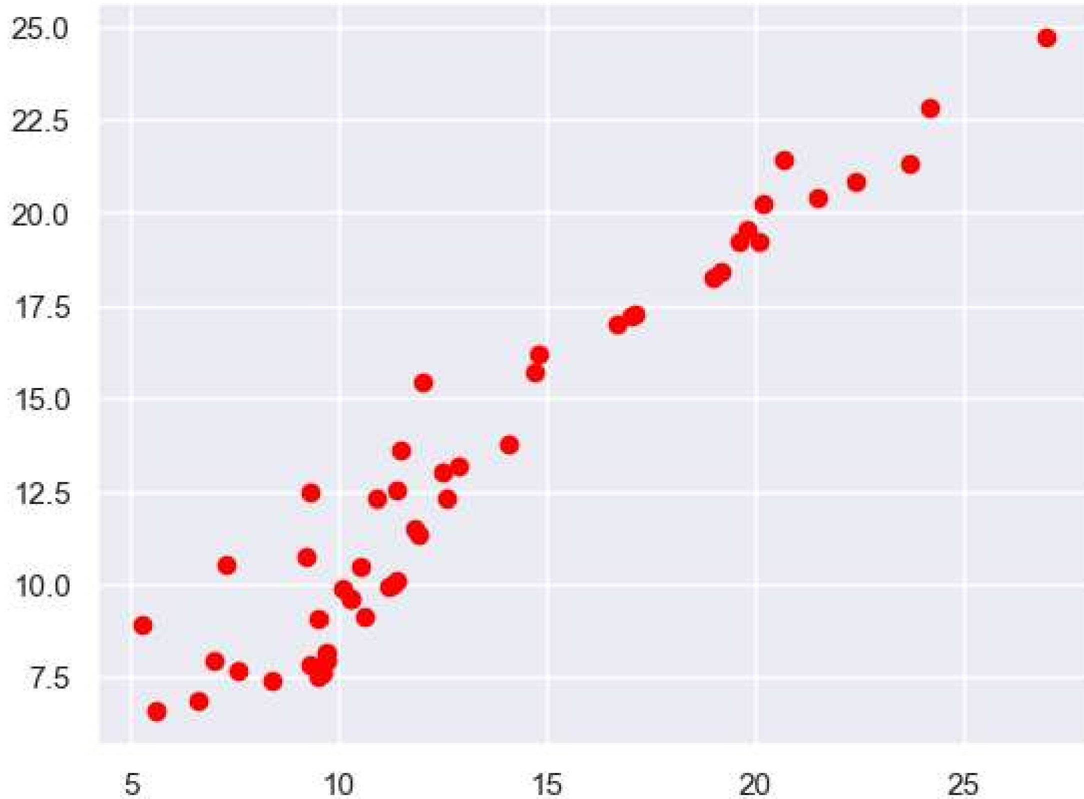
	coef	std err	t	P>|t|	[0.025	0.975]
TV	0.0529	0.002	33.285	0.000	0.050	0.056
radio	0.2252	0.011	20.735	0.000	0.204	0.247
newspaper	0.0174	0.008	2.184	0.031	0.002	0.033

Omnibus:	7.804	**Durbin-Watson:**	2.068
Prob(Omnibus):	0.020	**Jarque-Bera (JB):**	10.718
Skew:	-0.286	**Prob(JB):**	0.00471
Kurtosis:	4.178	**Cond. No.**	12.5

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

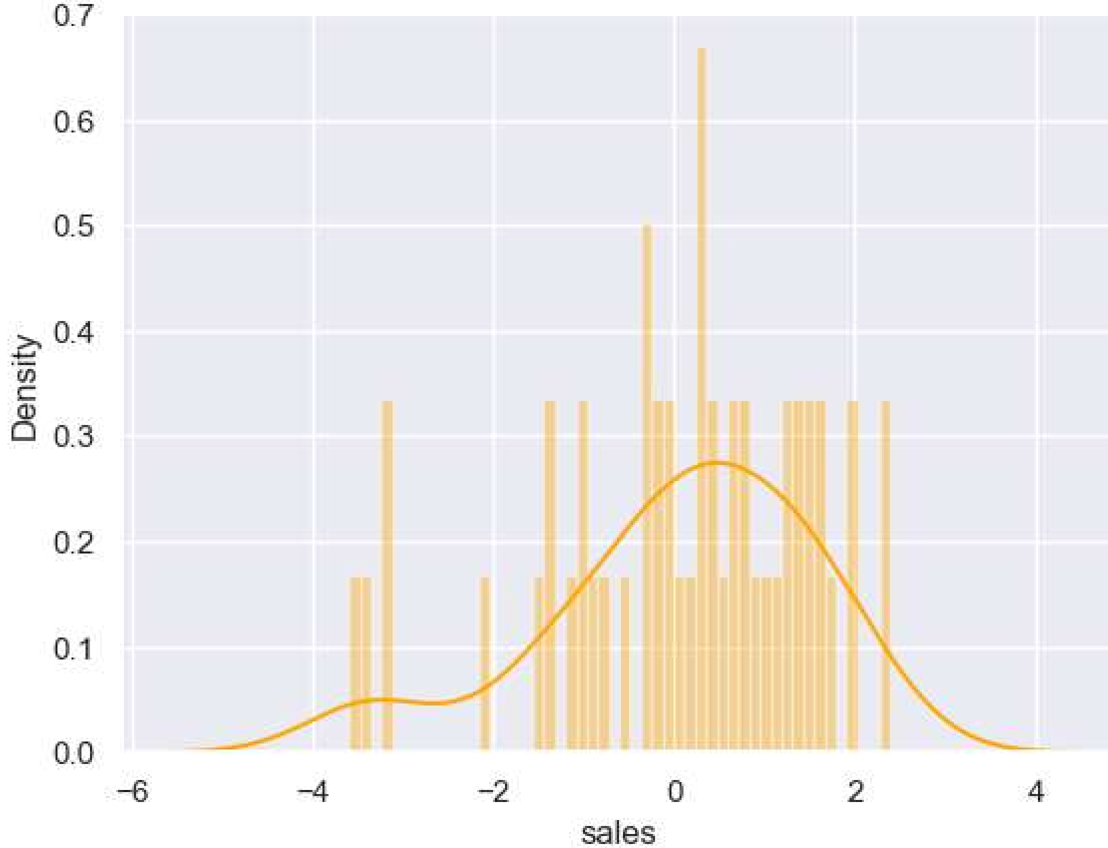
In [47]:
```
# Check linearity
```

Out[47]: <matplotlib.collections.PathCollection at 0x207c0da1390>



In [59]:
```
# Normality of Residual

sns.distplot((y_test - y_pred_price), bins=50,color='orange')
```



Lasso regularization

In [49]:
```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.1)
lasso.fit(x_train, y_train)
```

```
Lasso Model : [4.49462323e-02 1.89763674e-01 1.39174407e-04]
```

In [50]:
```
y_pred_train_lasso = lasso.predict(x_train)
```

In [51]:
```
print("Training Accuracy :", r2_score(y_train, y_pred_train_lasso))
print()
print("Test Accuracy :", r2_score(y_test, y_pred_test_lasso))
```

```
Training Accuracy : 0.8865116745214318

Test Accuracy : 0.9247422099742402
```

Ridge Regression

In [52]:
```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=0.3)
ridge.fit(x_train, y_train)
```
```
Ridge Model : [0.04495318 0.19016278 0.0002744 ]
```

In [53]:
```
y_pred_train_ridge = ridge.predict(x_train)
```

In [54]:
```
print("Training Accuracy :", r2_score(y_train, y_pred_train_ridge))
print()
```
```
Training Accuracy : 0.8865137138976493

Test Accuracy : 0.924676625698444
```

ElasticNet

In [55]:
```
from sklearn.linear_model import ElasticNet
elastic = ElasticNet(alpha=0.3, l1_ratio=0.1)
```

Out[55]:
```
▾ ElasticNet
ElasticNet(alpha=0.3, l1_ratio=0.1)
```

In [56]:
```
y_pred_train_elastic = elastic.predict(x_train)
```

In [57]:
```
print("Training Accuracy :", r2_score(y_train, y_pred_train_elastic))
print()
```
```
Training Accuracy : 0.886512572536935

Test Accuracy : 0.9247167726191866
```

In [58]:
```
# Conclude this model
# Data Preprocessing -
# EDA
# By using sklearn linear model
# training accuracy : 92.4%
# test accuracy = 88.6%
# Split the data into train and test
#Adj. R-squared (uncentered):    0.981
# 1) Linearity - Satisfied
```