SALES PREDICTION USING PYTHON

Description of the dataset

The Advertising dataset contains data on advertising budgets spent on three different media channels (TV, Radio, and Newspaper) and the corresponding sales figures. This dataset provides an excellent opportunity to understand how different advertising channels impact sales and build a predictive model to estimate future sales based on advertising expenditures.

In [40]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [41]:
```
df = pd.read_csv("C:/Users/Akash K Shaji/Downloads/Advertising.csv")
df.head(5)
```

Out[41]:

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

In [42]:
```
df.shape
```

Out[42]:
```
(200, 5)
```

We have 200 data points and 5 variables in the dataset.

In [43]:
```
#Dropping irrelevent column
df = df.drop('Unnamed: 0', axis=1)
df.head()
```

Out[43]:

	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

In [44]:
```
#Checking for missing values
df.isnull().sum()
```

Out[44]:
```
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```

There is no null values

In [45]:
```
#Checking for duplicate values
df.duplicated().sum()
```

Out[45]:
```
0
```

Duplicated values are not present in the dataset.

In [46]:
```
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
```

The dataset contains 4 numerical variables.

Number of non null values in each column is also obtained.

It also provides an estimate of the memory usage of the DataFrame. Here the memory usage is approximately 6.4 KB.

In [47]:
```
df.describe().T
```

Out[47]:

	count	mean	std	min	25%	50%	75%	max
TV	200.0	147.0425	85.854236	0.7	74.375	149.75	218.825	296.4
Radio	200.0	23.2640	14.846809	0.0	9.975	22.90	36.525	49.6
Newspaper	200.0	30.5540	21.778621	0.3	12.750	25.75	45.100	114.0
Sales	200.0	14.0225	5.217457	1.6	10.375	12.90	17.400	27.0

The standard deviation of TV advertising expenditures is relatively high at approximately 85.85. This indicates that the expenditures vary considerably from the mean, suggesting a wide range of spending levels among the observations.

The standard deviation of radio advertising expenditures is about 14.85. This indicates a moderate level of variability around the mean.

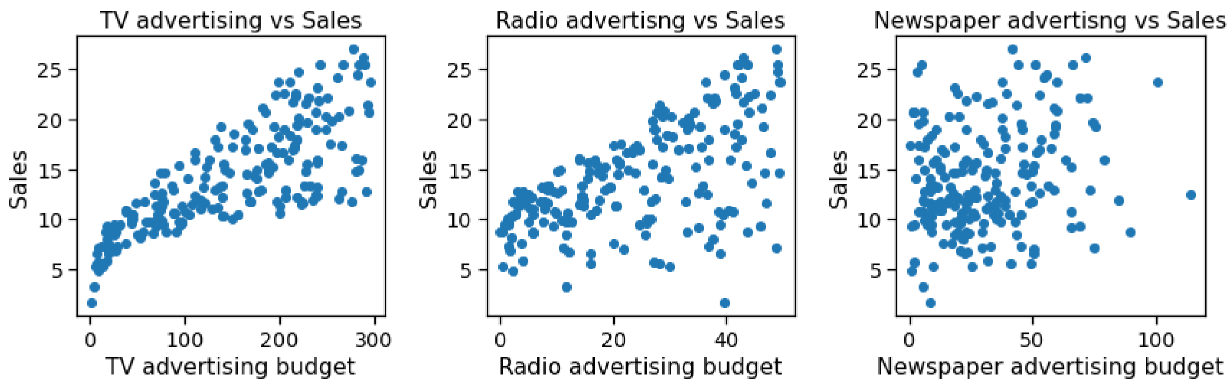
The standard deviation of newspaper advertising expenditures is relatively high at about 21.78. This suggests a wide range of spending levels, similar to TV advertising.

The standard deviation of sales is about 5.22, indicating a moderate level of variability around the mean.

The scales of the features (TV, Radio, Newspaper) are quite different. TV advertising spending has a much larger range compared to Radio and Newspaper spending. In such cases, standardization could be beneficial, as it can help ensure that the model gives equal importance to each feature during analysis.

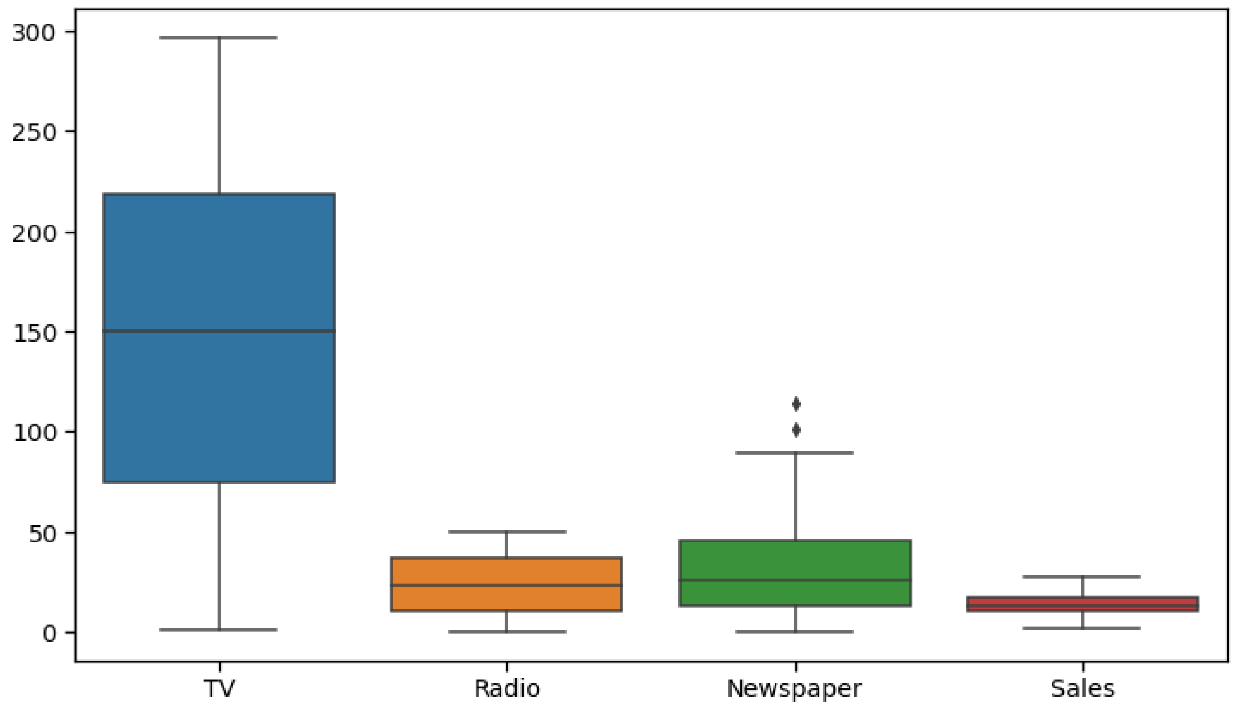
Relation between taget variable and feature variables

In [48]:
```
plt.figure(figsize=(12,4))

# Scatter plot for TV advertising budget vs. Sales
plt.subplot(131)
plt.scatter(df['TV'], df['Sales'])
plt.title('TV advertising vs Sales')
plt.xlabel('TV advertising budget')
plt.ylabel('Sales')

# Scatter plot for Radio advertising budget vs. Sales
plt.subplot(132)
plt.scatter(df['Radio'], df['Sales'])
plt.title('Radio advertisng vs Sales')
plt.xlabel('Radio advertising budget')
plt.ylabel('Sales')

# Scatter plot for Newspaper advertising budget vs. Sales
plt.subplot(133)
plt.scatter(df['Newspaper'], df['Sales'])
plt.title('Newspaper advertisng vs Sales')
plt.xlabel('Newspaper advertising budget')
plt.ylabel('Sales')

plt.tight_layout()
plt.show()
```



A scatterplot shows the relationship between two quantitative variables. From the scatter plot it is quite clear that variables 'TV' and 'Radio' is having linear relationship with the 'Sales'. They are also exhibiting positive relation.

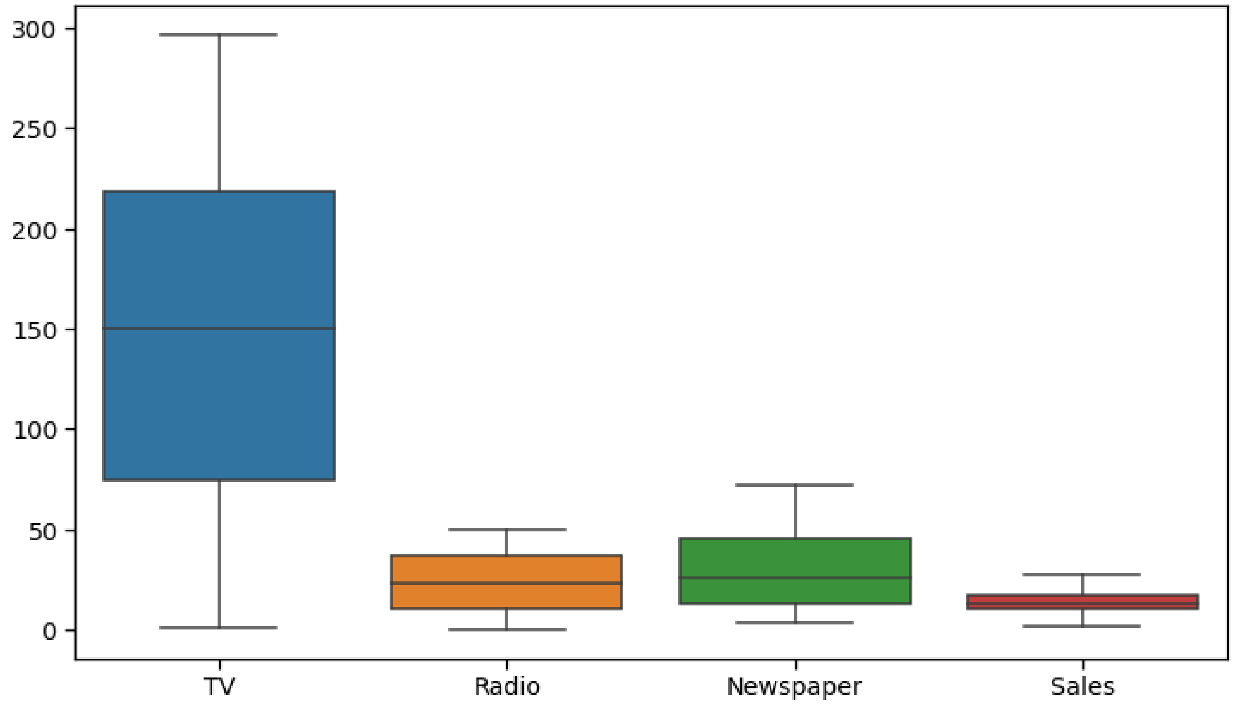
In [49]:
```
#Outlier detection
plt.figure(figsize=(12,7))
sns.boxplot(data = df)
plt.show()
```



Presence of outliers are detected in the variable 'Newspaper'.

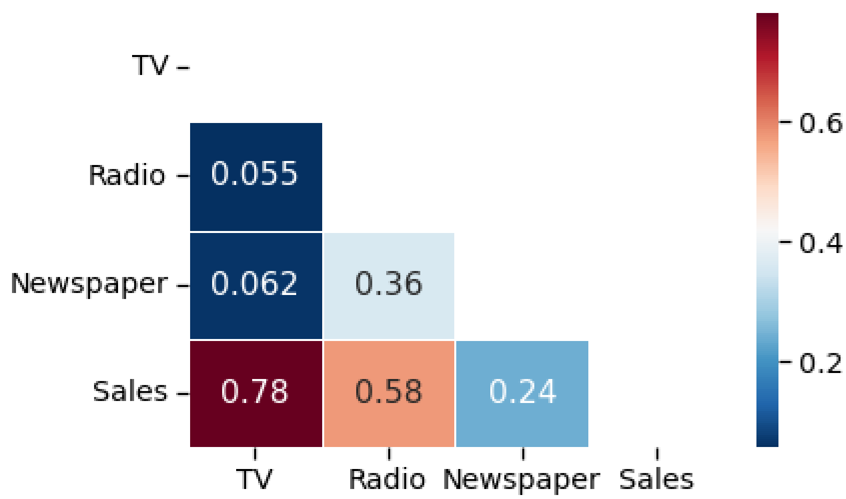
In [50]:
```
from scipy.stats.mstats import winsorize
# handling outliers using winsorize method
column_to_winsorize = 'Newspaper'
df[column_to_winsorize] = winsorize(df[column_to_winsorize], limits=(0.05, 0.05))
```

In [51]:
```
plt.figure(figsize=(12,7))
sns.boxplot(data = df)
plt.show()
```



So the outliers are being treated used winsorization method. Winsorization is a statistical technique used to handle outliers in a dataset by limiting extreme values to be within a specified range.

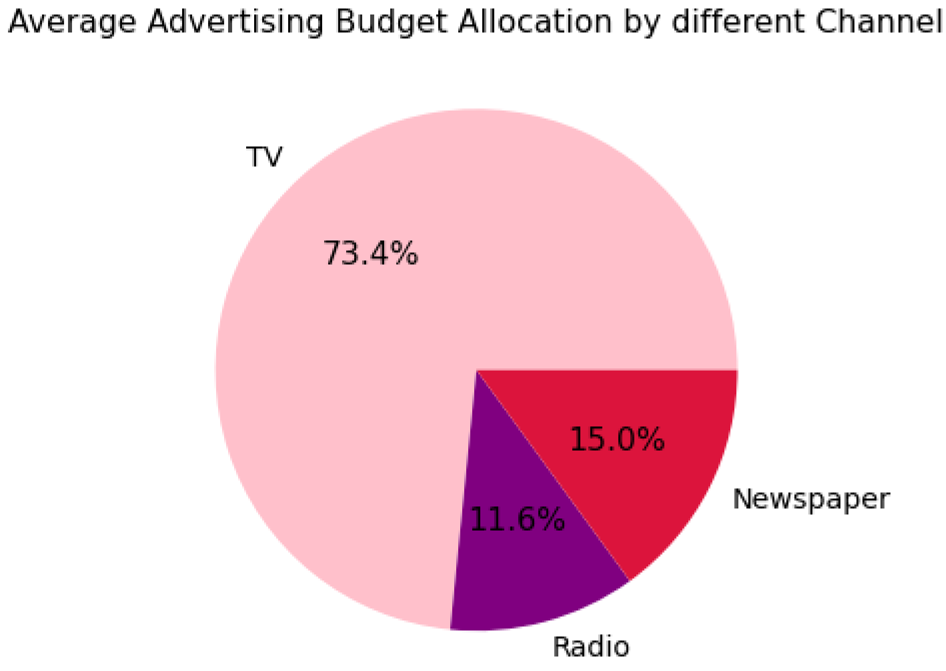
In [52]:
```
sns.set_context('notebook', font_scale=1.3)
mask = np.triu(np.ones_like(df.corr(), dtype=bool))
sns.heatmap(df.corr(), annot=True, cmap='RdBu_r', linewidths=0.5, mask=mask)
plt.show()
```



A correlation matrix is a table that displays the correlation coefficients between many variables. There is positive correlation among the variables. A strong positive correlation can be found between 'TV' and 'Sales'. The variables 'Radio' and 'Sales' is having moderate correlation. Other variables exhibit weak correlation.

Distribution of budget

In [53]:
```
average_budget = df[['TV', 'Radio', 'Newspaper']].mean()
plt.figure(figsize=(8, 6))
plt.pie(average_budget, labels=average_budget.index, autopct='%1.1f%%', colors=['pink'
plt.title('Average Advertising Budget Allocation by different Channel')
plt.show()
```



Among the 3 different channels the more budget is allocated with TV itself. We also find a positive linear relationship between TV and Sales. As the budget for advertising with TV increases the sales also increases.

Determining target variable and feature variable

In [54]:
```
# X is the feature variable
X = df.drop('Sales', axis=1)
# y is the target variable
y = df['Sales']
```

TV, Radio, Newspapers are the independent variables.Sales is the dependent variable.

Splitting the dataset into training and testing sets

In [55]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_stat
```

Standardizing

In [56]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
cols = ['TV', 'Radio', 'Newspaper', 'Sales']
df[cols] = scaler.fit_transform(df[cols])
df.head()
```

Out[56]:

	TV	Radio	Newspaper	Sales
0	0.969852	0.981522	1.955484	1.552053
1	-1.197376	1.082808	0.752217	-0.696046
2	-1.516155	1.528463	1.960477	-0.907406
3	0.052050	1.217855	1.421253	0.860330
4	0.394182	-0.841614	1.416261	-0.215683

Linear Regression

In [57]:

```
from sklearn.linear_model import LinearRegression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
y_pred_linear = linear_model.predict(X_test)
pd.DataFrame({'Actual_y': y_test,'Predicted_y':y_pred_linear })
```

Out[57]:

	Actual_y	Predicted_y
95	16.9	16.572124
15	22.4	21.196014
30	21.4	21.558759
158	7.3	10.890762
128	24.7	22.202849
115	12.6	13.360198
69	22.3	21.199615
170	8.4	7.346498
174	11.5	13.274532
45	14.9	15.126730
66	9.5	9.016875
182	8.7	6.524121
165	11.9	14.224597
78	5.3	8.963789
186	10.3	9.456755
177	11.7	12.007196
56	5.5	8.915936
152	16.6	16.155251
82	11.3	10.295844
68	18.9	18.724134
124	19.7	19.764105
16	12.5	13.492852
148	10.9	12.491424
93	22.2	21.544570
65	9.3	7.620626
60	8.1	5.608752
84	21.7	20.921613
67	13.4	11.802916
125	10.6	9.079493
132	5.7	8.516745
9	10.6	12.176300
18	11.3	9.966005
55	23.7	21.739521
75	8.7	12.664081

	Actual_y	Predicted_y
150	16.1	18.106938
104	20.7	20.074218
135	11.6	14.256694
137	20.8	20.949061
164	11.9	10.834466
76	6.9	4.377896
79	11.0	9.512065
197	12.8	12.401495
38	10.1	10.170372
24	9.7	8.087318
122	11.6	13.163505
195	7.6	5.219152
29	10.5	9.290554
19	14.6	14.092226
143	10.4	8.691113
86	12.0	11.657808
114	14.6	15.719474
173	11.7	11.629335
5	7.2	13.338985
126	6.6	11.155778
117	9.4	6.332246
73	11.0	9.762374
140	10.9	9.415256
98	25.4	24.264752
172	7.6	7.690524
96	11.7	12.150091

Evaluation of Linear Regression model

In [58]:
```
# Calculate the Mean Squared Error (MSE)
from sklearn.metrics import mean_squared_error
MSE = mean_squared_error(y_test, y_pred_linear,squared=False)
MSE
```

Out[58]:
```
1.9395548328970966
```

Mean Squared Error (MSE) calculates the average of the squared differences between predicted values and actual values. Smaller MSE values indicate better model performance.

In [59]:
```
# Calculate the Root Mean Squared Error (RMSE)
RMSE = np.sqrt(MSE)
RMSE
```

Out[59]:
```
1.3926790128730657
```

Root Mean Squared Error (RMSE) is the square root of MSE.

In [60]:
```
# Calculate the Mean Absolute Error (MAE)
from sklearn.metrics import mean_absolute_error
MAE = mean_absolute_error(y_test, y_pred_linear)
MAE
```

Out[60]:
```
1.5035596259580652
```

Mean Absolute Error (MAE) measures the average absolute difference between the actual values and the predicted values. Lower MAE values indicate better model performance.

In [61]:
```
# Calculate the coefficient of determination (R-squared)
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred_linear)
print("R-squared:", r2)
```

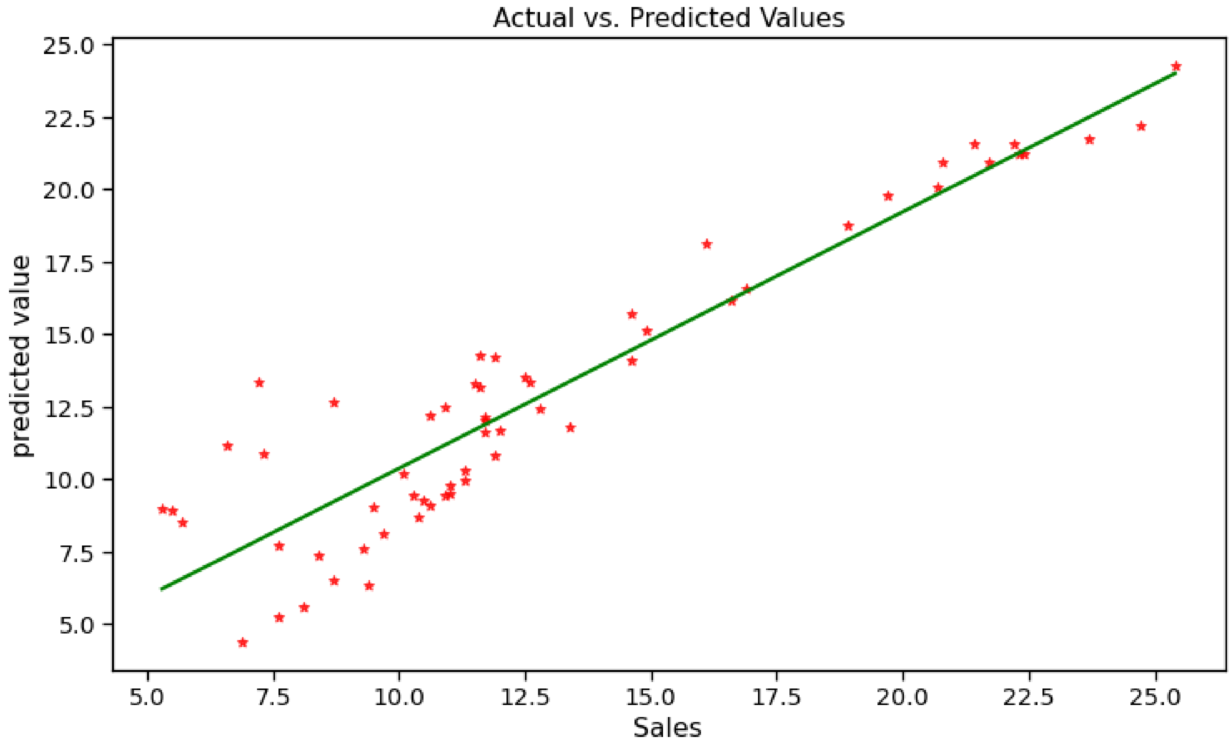
```
R-squared: 0.8622257128214504
```

R-squared measures the proportion of the variance in the dependent variable (target) that is explained by the independent variables (features) in your model. It ranges from 0 to 1, where 0 indicates that the model does not explain any variance, and 1 indicates a perfect fit. A higher R^2 indicates a better fit of the model to the data. Here, 86% of variations in target variable is explained by the independent variables.

Regression Plot

In [62]:
```
plt.figure(figsize=(12,7))
plt.xlabel("Actual value")
plt.ylabel("predicted value")
plt.title('Actual vs. Predicted Values')
sns.regplot(x=y_test, y=y_pred_linear, ci=None, color='red', marker="*",line_kws={"col
plt.show
```

Out[62]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



The green line in the plot is the regression line. It represents the linear relationship between the actual and predicted values. In a perfect model, all points would fall exactly on this line.

The red points on the plot are the actual vs. predicted values. They are used to visualize how well the model's predictions match the actual values. Points above the line indicate that the model overestimated the actual values, while points below the line indicate underestimation.

When the red points are scattered or deviate significantly from the line, it suggests that the model's predictions may not be accurate.

Ridge Regression

In [63]:
```
from sklearn.linear_model import Ridge
ridge_reg = Ridge(alpha=1, solver="cholesky")
ridge_reg.fit(X_train, y_train)
y_pred_ridge = ridge_reg.predict(X_test)
pd.DataFrame({'Actual_y': y_test,'Predicted_y':y_pred_ridge })
```

Out[63]:

	Actual_y	Predicted_y
95	16.9	16.572104
15	22.4	21.195873
30	21.4	21.558750
158	7.3	10.890683
128	24.7	22.202612
115	12.6	13.360148
69	22.3	21.199458
170	8.4	7.346563
174	11.5	13.274655
45	14.9	15.126740
66	9.5	9.016815
182	8.7	6.524251
165	11.9	14.224825
78	5.3	8.963698
186	10.3	9.456910
177	11.7	12.007324
56	5.5	8.915916
152	16.6	16.155225
82	11.3	10.295870
68	18.9	18.724073
124	19.7	19.764115
16	12.5	13.492824
148	10.9	12.491261
93	22.2	21.544549
65	9.3	7.620683
60	8.1	5.608895
84	21.7	20.921474
67	13.4	11.802948
125	10.6	9.079571
132	5.7	8.516665
9	10.6	12.176443
18	11.3	9.966003
55	23.7	21.739380
75	8.7	12.663998

	Actual_y	Predicted_y
150	16.1	18.107026
104	20.7	20.074095
135	11.6	14.256475
137	20.8	20.949077
164	11.9	10.834487
76	6.9	4.378040
79	11.0	9.512170
197	12.8	12.401561
38	10.1	10.170353
24	9.7	8.087376
122	11.6	13.163641
195	7.6	5.219268
29	10.5	9.290627
19	14.6	14.092203
143	10.4	8.691253
86	12.0	11.657750
114	14.6	15.719303
173	11.7	11.629428
5	7.2	13.338863
126	6.6	11.155693
117	9.4	6.332387
73	11.0	9.762510
140	10.9	9.415272
98	25.4	24.264652
172	7.6	7.690522
96	11.7	12.150200

In [64]:

```
# Calculate the Mean Squared Error (MSE)
MSE_r = mean_squared_error(y_test, y_pred_ridge,squared=False)
MSE_r
```

Out[64]:

```
1.9395254163097309
```

In [65]:

```
# Calculate the Root Mean Squared Error (RMSE)
RMSE_r = np.sqrt(MSE_r)
RMSE_r
```

Out[65]:

```
1.392684516817815
```

In [66]:
```
# Calculate the Mean Absolute Error (MAE)
MAE_r = mean_absolute_error(y_test, y_pred_ridge)
MAE_r
```

Out[66]:
```
1.5035477558529506
```

In [67]:
```
# Calculate the coefficient of determination (R-squared)
from sklearn.metrics import r2_score
r2_r = r2_score(y_test, y_pred_ridge)
print("R-squared:", r2_r)
```

```
R-squared: 0.8622298919439508
```