Experiment 1

Python

In [4]:
```
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
#load data into a DataFrame object:
df = pd.DataFrame(data)
print(df)
```

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

In [5]:
```
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)
```

```
      calories  duration
day1       420        50
day2       380        40
day3       390        45
```

In [8]:
```
import pandas as pd

df = pd.read_csv(r"C:\Users\91916\Downloads\archive\Iris.csv")
print(df)
```

```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

In [9]:
```
df.head()
```

Out[9]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [10]:
```
df.tail()
```

Out[10]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

In [11]:
```
df.sample()
```

Out[11]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
110	111	6.5	3.2	5.1	2.0	Iris-virginica

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [13]:
```
df.describe()
```

Out[13]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [14]:
```
df.query("PetalLengthCm>5.0")
```
Out[14]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
83	84	6.0	2.7	5.1	1.6	Iris-versicolor
100	101	6.3	3.3	6.0	2.5	Iris-virginica
101	102	5.8	2.7	5.1	1.9	Iris-virginica
102	103	7.1	3.0	5.9	2.1	Iris-virginica
103	104	6.3	2.9	5.6	1.8	Iris-virginica
104	105	6.5	3.0	5.8	2.2	Iris-virginica
105	106	7.6	3.0	6.6	2.1	Iris-virginica
107	108	7.3	2.9	6.3	1.8	Iris-virginica
108	109	6.7	2.5	5.8	1.8	Iris-virginica
109	110	7.2	3.6	6.1	2.5	Iris-virginica
110	111	6.5	3.2	5.1	2.0	Iris-virginica
111	112	6.4	2.7	5.3	1.9	Iris-virginica
112	113	6.8	3.0	5.5	2.1	Iris-virginica
114	115	5.8	2.8	5.1	2.4	Iris-virginica
115	116	6.4	3.2	5.3	2.3	Iris-virginica
116	117	6.5	3.0	5.5	1.8	Iris-virginica
117	118	7.7	3.8	6.7	2.2	Iris-virginica
118	119	7.7	2.6	6.9	2.3	Iris-virginica
120	121	6.9	3.2	5.7	2.3	Iris-virginica
122	123	7.7	2.8	6.7	2.0	Iris-virginica
124	125	6.7	3.3	5.7	2.1	Iris-virginica
125	126	7.2	3.2	6.0	1.8	Iris-virginica
128	129	6.4	2.8	5.6	2.1	Iris-virginica
129	130	7.2	3.0	5.8	1.6	Iris-virginica
130	131	7.4	2.8	6.1	1.9	Iris-virginica
131	132	7.9	3.8	6.4	2.0	Iris-virginica
132	133	6.4	2.8	5.6	2.2	Iris-virginica

In [15]:
```
df.loc[100,'Species']
```
Out[15]:
```
'Iris-virginica'
```

In [18]:
```
df.iloc[[100, 50]]
```
Out[18]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
100	101	6.3	3.3	6.0	2.5	Iris-virginica
50	51	7.0	3.2	4.7	1.4	Iris-versicolor

In [19]:
```
df["SepalWidthCm"].unique()
```
Out[19]:
```
array([3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4, 2.9, 3.7, 4. , 4.4, 3.8, 3.3,
       4.1, 4.2, 2.3, 2.8, 2.4, 2.7, 2. , 2.2, 2.5, 2.6])
```

In [20]:
```
df["SepalWidthCm"].nunique()
```
Out[20]:
```
23
```

In [21]:
```
df.isnull()
```
Out[21]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False
148	False	False	False	False	False	False
149	False	False	False	False	False	False

150 rows × 6 columns

In [23]:
```
df.sort_values("PetalLengthCm")
```
Out[23]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
22	23	4.6	3.6	1.0	0.2	Iris-setosa
13	14	4.3	3.0	1.1	0.1	Iris-setosa
14	15	5.8	4.0	1.2	0.2	Iris-setosa
35	36	5.0	3.2	1.2	0.2	Iris-setosa
36	37	5.5	3.5	1.3	0.2	Iris-setosa
...	...	...	...	...	...	...
131	132	7.9	3.8	6.4	2.0	Iris-virginica
105	106	7.6	3.0	6.6	2.1	Iris-virginica
117	118	7.7	3.8	6.7	2.2	Iris-virginica
122	123	7.7	2.8	6.7	2.0	Iris-virginica
118	119	7.7	2.6	6.9	2.3	Iris-virginica

150 rows × 6 columns

In [24]:
```
df.value_counts("Species")
```
Out[24]:
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

R

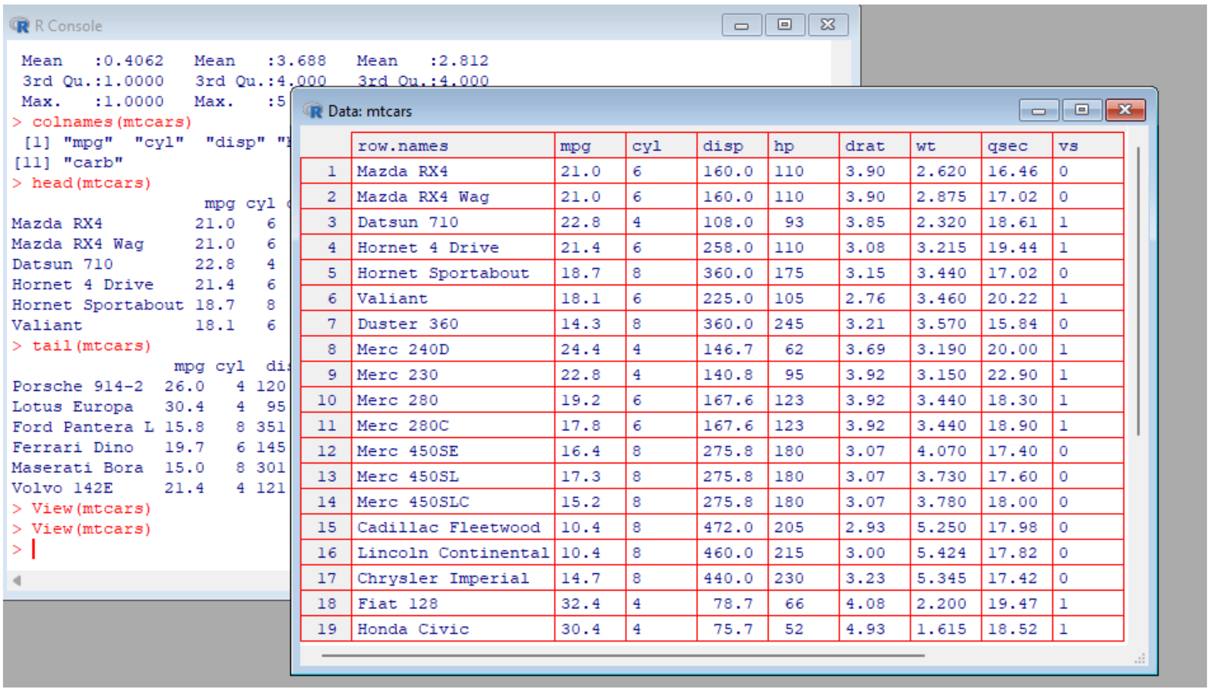
R Console

```
> data(mtcars)
> dim(mtcars)
[1] 32 11
> str(mtcars)
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
> summary(mtcars)
      mpg             cyl             disp             hp
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
 Median :19.20   Median :6.000   Median :196.3   Median :123.0
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
      drat             wt             qsec             vs
```

R Console

```
      drat             wt             qsec             vs
 Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000
 Median :3.695   Median :3.325   Median :17.71   Median :0.0000
 Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
 Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
       am              gear             carb
 Min.   :0.0000   Min.   :3.000   Min.   :1.000
 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
 Median :0.0000   Median :4.000   Median :2.000
 Mean   :0.4062   Mean   :3.688   Mean   :2.812
 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
 Max.   :1.0000   Max.   :5.000   Max.   :8.000
> colnames(mtcars)
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
> tail(mtcars)
                mpg cyl  disp  hp drat    wt qsec vs am gear carb
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```



Experiment 2

Problem Statement

Build a model which predicts sales based on the money spent on different platforms for marketing.

Data:

Use the advertising dataset given in ISLR and analyse the relationship between 'TV advertising' and 'sales' using a simple linear regression model.

In this notebook, we'll build a linear regression model to predict Sales using an appropriate predictor variable.

In [1]:
```
# Supress Warnings

import warnings
warnings.filterwarnings('ignore')

# Import the numpy and pandas package

import numpy as np
import pandas as pd

# Data Visualisation
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
advertising = pd.DataFrame(pd.read_csv("Advertising.csv"))
# advert = pd.read_csv('Advertising.csv')
advertising.head()
```

Out[2]:

	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

Data Inspection ¶

In [3]:
```
advertising.shape
```

Out[3]: (200, 4)

In [4]:
```
advertising.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 [5]:
```
advertising.describe()
```

Out[5]:

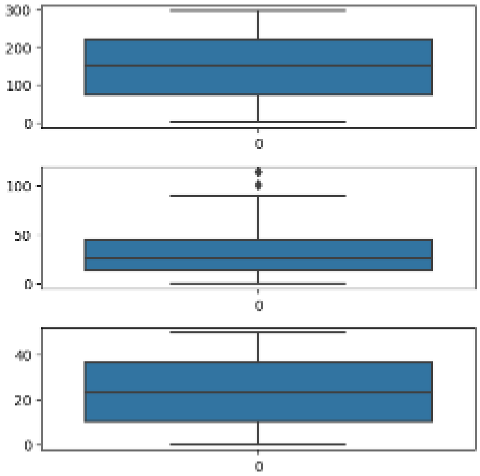
	TV	radio	newspaper	sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	14.022500
std	85.854236	14.846809	21.778621	5.217457
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	10.375000
50%	149.750000	22.900000	25.750000	12.900000
75%	218.825000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

Data Cleaning

In [6]:
```
# Checking Null values
advertising.isnull().sum()*100/advertising.shape[0]
# There are no NULL values in the dataset, hence it is clean.
```

Out[6]:
```
TV           0.0
radio        0.0
newspaper    0.0
sales        0.0
dtype: float64
```

In [8]:
```
# Outlier Analysis
fig, axs = plt.subplots(3, figsize = (5,5))
plt1 = sns.boxplot(advertising['TV'], ax = axs[0])
plt2 = sns.boxplot(advertising['newspaper'], ax = axs[1])
plt3 = sns.boxplot(advertising['radio'], ax = axs[2])
plt.tight_layout()
```



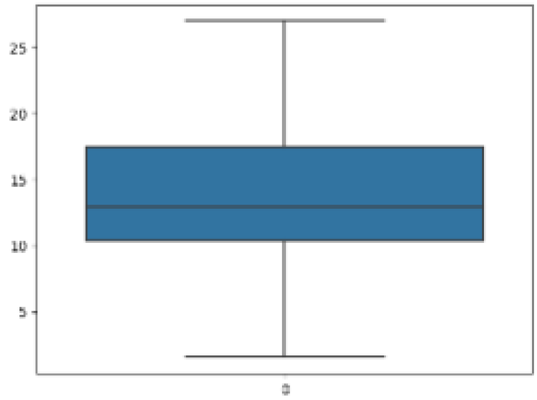
In [ ]:
```
# There are no considerable outliers present in the data.
```

Exploratory Data Analysis

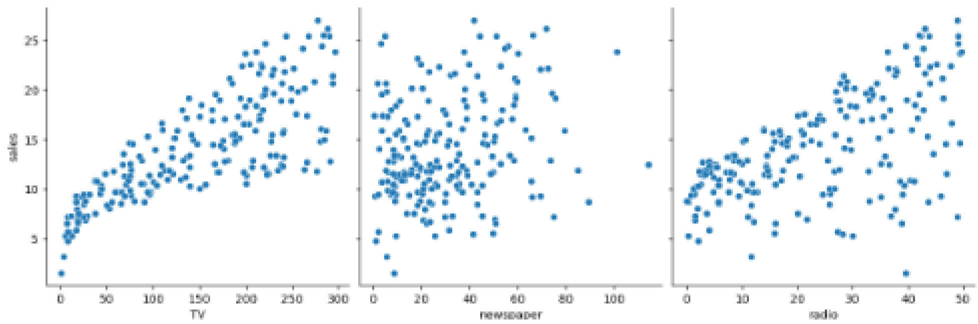
Univariate Analysis

Sales (Target Variable)

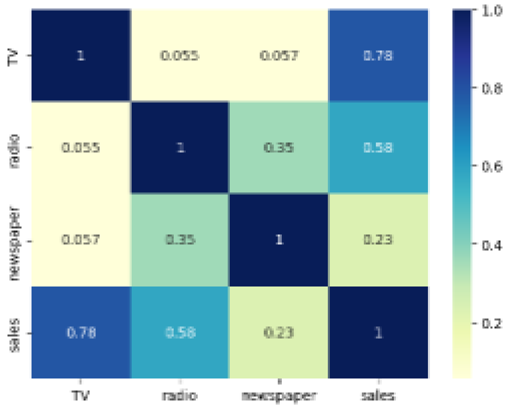
In [9]:
```
sns.boxplot(advertising['sales'])
plt.show()
```



In [10]:
```
# Let's see how Sales are related with other variables using scatter plot.
sns.pairplot(advertising, x_vars=['TV', 'newspaper', 'radio'], y_vars='sales', height=4, aspect=1, kind='scatter')
plt.show()
```



In [11]:
```
# Let's see the correlation between different variables.
sns.heatmap(advertising.corr(), cmap="YlGnBu", annot = True)
plt.show()
```



As is visible from the pairplot and the heatmap, the variable TV seems to be most correlated with Sales. So let's go ahead and perform simple linear regression using TV as our feature variable.

Model Building

Performing Simple Linear Regression Equation of linear regression

Equation of linear regression

y = c + m1x1 + m2x2 + ... + mnxn

y is the response c is the intercept m1 is the coefficient for the first feature mn is the coefficient for the nth feature In our case:

y = c + m1 × TV

The m values are called the model coefficients or model parameters.

Generic Steps in model building using statsmodels We first assign the feature variable, TV, in this case, to the variable X and the response variable, Sales, to the variable y.

In [12]:
```

X = advertising['TV']
y = advertising['sales']
```

Train-Test Split

You now need to split our variable into training and testing sets. You'll perform this by importing train_test_split from the sklearn.model_selection library. It is usually a good practice to keep 70% of the data in your train dataset and the rest 30% in your test dataset

In [13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, test_size = 0.3, random_state = 100)
```

In [14]:
```
# Let's now take a look at the train dataset

X_train.head()
```

Out[14]:
```
74     213.4
3      151.5
185    205.0
26     142.9
90     134.3
Name: TV, dtype: float64
```

In [15]:
```
y_train.head()
```

Out[15]:
```
74     17.0
3      18.5
185    22.6
26     15.0
90     11.2
Name: sales, dtype: float64
```

Building a Linear Model

You first need to import the statsmodel.api library using which you'll perform the linear regression.

In [16]:
```
import statsmodels.api as sm
```

By default, the statsmodels library fits a line on the dataset which passes through the origin. But in order to have an intercept, you need to manually use the add_constant attribute of statsmodels. And once you've added the constant to your X_train dataset, you can go ahead and fit a regression line using the OLS (Ordinary Least Squares) attribute of statsmodels as shown below

In [17]:
```
# Add a constant to get an intercept
X_train_sm = sm.add_constant(X_train)

# Fit the resgression line using 'OLS'
lr = sm.OLS(y_train, X_train_sm).fit()
```

In [18]:
```
# Print the parameters, i.e. the intercept and the slope of the regression line fitted
lr.params
```

Out[18]:
```
const    6.989666
TV       0.046497
dtype: float64
```

In [19]:
```
# Performing a summary operation lists out all the different parameters of the regression line fitted
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  sales   R-squared:                       0.613
Model:                            OLS   Adj. R-squared:                  0.611
Method:                 Least Squares   F-statistic:                     219.0
Date:                Sun, 14 Apr 2024   Prob (F-statistic):           2.84e-30
Time:                        20:09:23   Log-Likelihood:                -370.62
No. Observations:                 140   AIC:                             745.2
Df Residuals:                     138   BIC:                             751.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          6.9897      0.548     12.762      0.000       5.907       8.073
TV             0.0465      0.003     14.798      0.000       0.040       0.053
==============================================================================
Omnibus:                        0.995   Durbin-Watson:                   1.983
Prob(Omnibus):                  0.608   Jarque-Bera (JB):                0.970
Skew:                          -0.008   Prob(JB):                        0.616
Kurtosis:                       2.593   Cond. No.                         328.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Looking at some key statistics from the summary

The values we are concerned with are -

The coefficients and significance (p-values) R-squared F statistic and its significance

1. The coefficient for TV is 0.054, with a very low p value The coefficient is statistically significant. So the association is not purely by chance.
2. R - squared is 0.816 Meaning that 81.6% of the variance in Sales is explained by TV

This is a decent R-squared value.

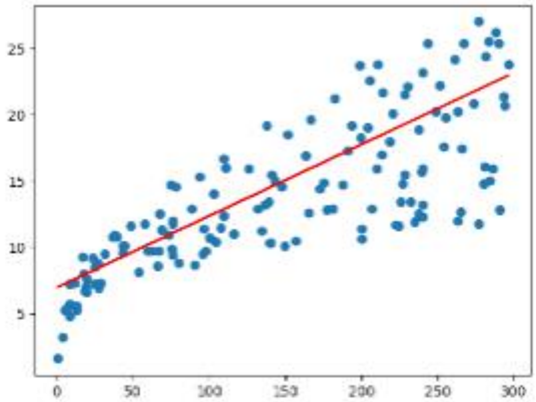
3. F statistic has a very low p value (practically low) Meaning that the model fit is statistically significant, and the explained variance isn't purely by chance.

The fit is significant. Let's visualize how well the model fit the data.

From the parameters that we get, our linear regression equation becomes:

Sales = 6.948 + 0.054 × TV

In [20]:
```
plt.scatter(X_train, y_train)
plt.plot(X_train, 6.948 + 0.054*X_train, 'r')
plt.show()
```



Model Evaluation

Residual analysis

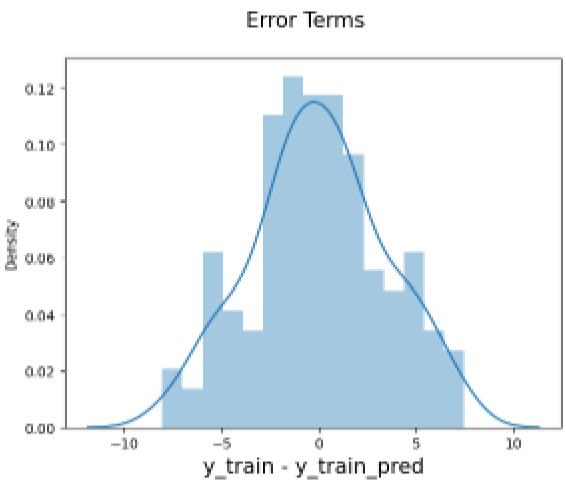
To validate assumptions of the model, and hence the reliability for inference

Distribution of the error terms

We need to check if the error terms are also normally distributed (which is infact, one of the major assumptions of linear regression), let us plot the histogram of the error terms and see what it looks like.

In [21]:
```
y_train_pred = lr.predict(X_train_sm)
res = (y_train - y_train_pred)
```

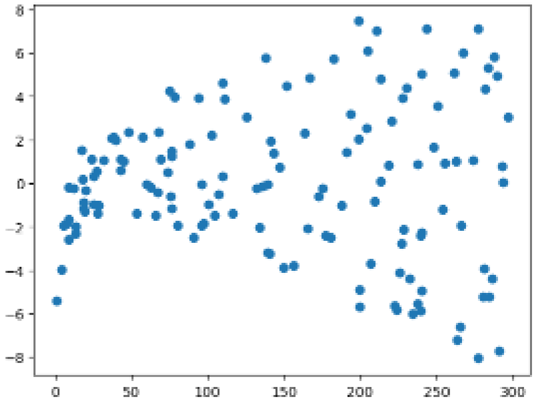
In [22]:
```
fig = plt.figure()
sns.distplot(res, bins = 15)
fig.suptitle('Error Terms', fontsize = 15)                  # Plot heading
plt.xlabel('y_train - y_train_pred', fontsize = 15)         # X-label
plt.show()
```



The residuals are following the normally distributed with a mean 0. All good!

Looking for patterns in the residuals

In [23]:
```
plt.scatter(X_train,res)
plt.show()
```



We are confident that the model fit isn't by chance, and has decent predictive power. The normality of residual terms allows some inference on the coefficients.

Although, the variance of residuals increasing with X indicates that there is significant variation that this model is unable to explain.

As you can see, the regression line is a pretty good fit to the data

Predictions on the Test Set Now that you have fitted a regression line on your train dataset, it's time to make some predictions on the test data. For this, you first need to add a constant to the X_test data like you did for X_train and then you can simply go on and predict the y values corresponding to X_test using the predict attribute of the fitted regression line.

In [24]:
```
# Add a constant to X_test
X_test_sm = sm.add_constant(X_test)

# Predict the y values corresponding to X_test_sm
y_pred = lr.predict(X_test_sm)
```

In [25]:
```
y_pred.head()
```

Out[25]:
```
126     7.352345
104    18.065337
99     13.276109
92     17.112141
111    18.228077
dtype: float64
```

In [26]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

Looking at the RMSE

In [27]:
```
#Returns the mean squared error; we'll take a square root
np.sqrt(mean_squared_error(y_test, y_pred))
```

Out[27]:
```
2.824145628832781
```

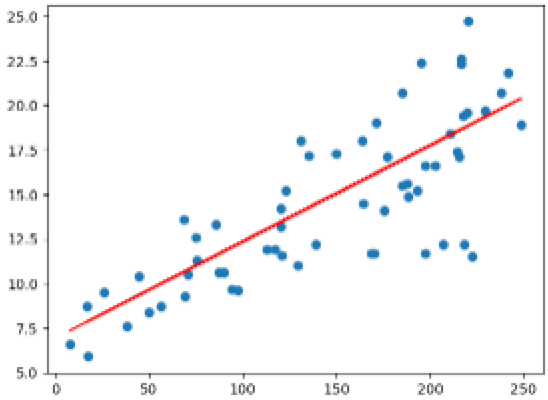
Checking the R-squared on the test set

In [28]:
```
r_squared = r2_score(y_test, y_pred)
r_squared
```

Out[28]:
```
0.59429872677833
```

Visualizing the fit on the test set

In [29]:
```
plt.scatter(X_test, y_test)
plt.plot(X_test, 6.948 + 0.054 * X_test, 'r')
plt.show()
```



Experiment 3

Task 1: Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy.stats import skew
%matplotlib inline
```

In [2]:
```
import matplotlib.pyplot as plt
plt.style.use("ggplot")
plt.rcParams['figure.figsize'] = (12, 8)
```

Task 2: Load the Data

The adverstiting dataset captures sales revenue generated with respect to advertisement spends across multiple channles like radio, tv and newspaper.

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

Out[3]:

	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

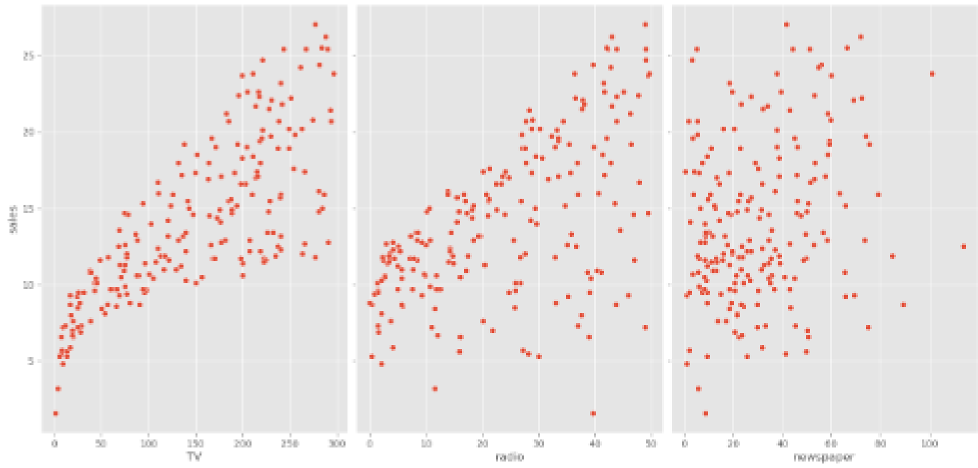
In [4]:
```
advert.info()
```

```
<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
```

Task 3: Relationship between Features and Response

In [5]:
```
sns.pairplot(advert, x_vars=['TV','radio','newspaper'], y_vars='sales', height=7, aspect=0.7);
```

```
C:\Users\rashm\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Task 4: Multiple Linear Regression - Estimating Coefficients

In [6]:
```
from sklearn.linear_model import LinearRegression

# create X and y
feature_cols = ['TV', 'radio', 'newspaper']
X = advert[feature_cols]
y = advert.sales

# instantiate and fit
lm1 = LinearRegression()
lm1.fit(X, y)

# print the coefficients
print(lm1.intercept_)
print(lm1.coef_)
```

```
2.938889369459885
[ 0.04576465  0.18853002 -0.00103749]
```

In [7]:
```python
# pair the feature names with the coefficients
list(zip(feature_cols, lm1.coef_))
```

Out[7]:
```
[('TV', 0.045764645455397615),
 ('radio', 0.18853001691820448),
 ('newspaper', -0.0010374930424763087)]
```

In [8]:
```python
sns.heatmap(advert.corr(), annot=True)
```

Out[8]:
```
<Axes: >
```

Task 5: Feature Selection

In [9]:
```
from sklearn.metrics import r2_score

lm2 = LinearRegression().fit(X[['TV', 'radio']], y)
lm2_preds = lm2.predict(X[['TV', 'radio']])

print("R^2: ", r2_score(y, lm2_preds))
```

```
R^2:  0.8971942610828956
```

In [10]:
```
lm3 = LinearRegression().fit(X[['TV', 'radio', 'newspaper']], y)
lm3_preds = lm3.predict(X[['TV', 'radio', 'newspaper']])

print("R^2: ", r2_score(y, lm3_preds))
```

```
R^2:  0.8972106381789522
```

Task 6: Model Evaluation Using Train/Test Split and Metrics

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Yellowbrick is a Python library that extends the functionality of scikit-learn and other popular machine learning libraries by providing visualizations and visual diagnostic tools to aid in the model selection and evaluation process. It is built on top of matplotlib, scikit-learn, and other libraries, making it easy to integrate with existing machine learning workflows.

Yellowbrick offers a variety of visualizers for different stages of the machine learning pipeline, including data visualization, feature visualization, model selection, evaluation, and tuning. Some of the visualizations provided by Yellowbrick include scatter plots, histograms, confusion matrices, ROC curves, and learning curves, among others.

In the context of the error you encountered, PredictionError and ResidualsPlot are classes within the Yellowbrick library that are specifically designed for visualizing prediction errors and residuals when working with regression models. These visualizations can help you assess the performance of your regression model and identify any patterns or trends in the prediction errors or residuals.

Overall, Yellowbrick is a powerful tool for gaining insights into your machine learning models through visualizations, allowing you to better understand and interpret their behavior.

Let's use train/test split with RMSE to see whether newspaper should be kept in the model:

In [11]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X = advert[['TV', 'radio', 'newspaper']]
y = advert.sales

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1)

lm4 = LinearRegression()
lm4.fit(X_train, y_train)
lm4_preds = lm4.predict(X_test)

print("RMSE :", np.sqrt(mean_squared_error(y_test, lm4_preds)))
print("R^2: ", r2_score(y_test, lm4_preds))
```

```
RMSE : 1.4046514230328953
R^2:  0.9156213613792232
```

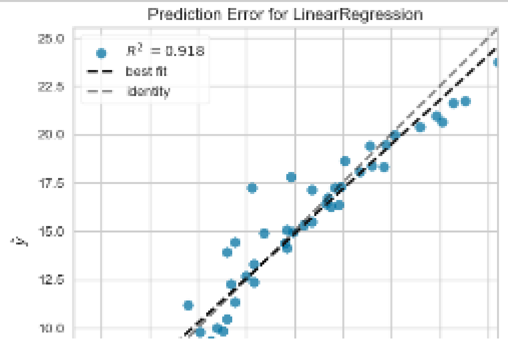
In [12]:
```
X = advert[['TV', 'radio']]
y = advert.sales

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1)

lm5 = LinearRegression()
lm5.fit(X_train, y_train)
lm5_preds = lm5.predict(X_test)

print("RMSE :", np.sqrt(mean_squared_error(y_test, lm5_preds)))
print("R^2: ", r2_score(y_test, lm5_preds))
```

```
RMSE : 1.3879034699382888
R^2:  0.9176214942248908
```

In [13]:
```
pip install yellowbrick
```

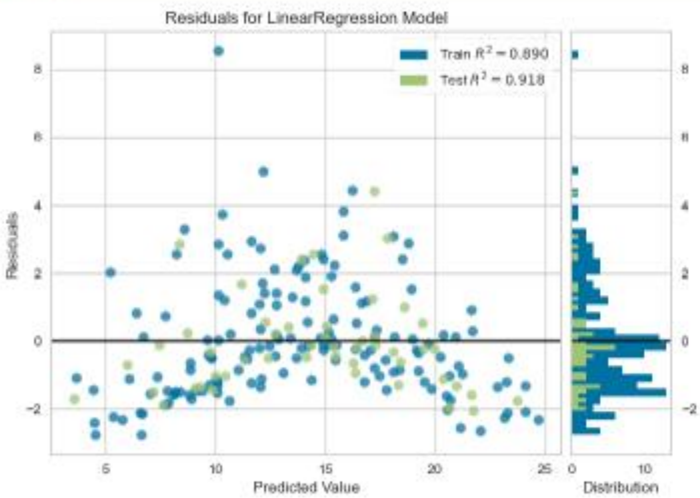
```
Requirement already satisfied: yellowbrick in c:\users\rashm\anaconda3\lib\site-packages (1.5)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in c:\users\rashm\anaconda3\lib\site-packages (from yellowbrick) (3.7.
2)
Requirement already satisfied: scipy>=1.0.0 in c:\users\rashm\anaconda3\lib\site-packages (from yellowbrick) (1.11.1)
Requirement already satisfied: scikit-learn>=1.0.0 in c:\users\rashm\anaconda3\lib\site-packages (from yellowbrick) (1.3.0)
Requirement already satisfied: numpy>=1.16.0 in c:\users\rashm\anaconda3\lib\site-packages (from yellowbrick) (1.24.3)
Requirement already satisfied: cycler>=0.10.0 in c:\users\rashm\anaconda3\lib\site-packages (from yellowbrick) (0.11.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\rashm\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->
yellowbrick) (1.0.5)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\rashm\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2-
>yellowbrick) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\rashm\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2-
>yellowbrick) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\rashm\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->y
ellowbrick) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\rashm\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yel
lowbrick) (9.4.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\rashm\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.
0.2->yellowbrick) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\rashm\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>=2.
0.2->yellowbrick) (2.8.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\rashm\anaconda3\lib\site-packages (from scikit-learn>=1.0.0->yellowbri
ck) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\rashm\anaconda3\lib\site-packages (from scikit-learn>=1.0.0->ye
llowbrick) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\rashm\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib!=
3.0.0,>=2.0.2->yellowbrick) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [14]:
```
from yellowbrick.regressor import PredictionError, ResidualsPlot
visualizer = PredictionError(lm5)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
visualizer.poof()
```



In [15]:
```
visualizer = ResidualsPlot(lm5)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.poof()
```

```
C:\Users\rashm\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, but LinearReg
ression was fitted with feature names
  warnings.warn(
```



Out[15]: `<Axes: title={'center': 'Residuals for LinearRegression Model'}, xlabel='Predicted Value', ylabel='Residuals'>`

Task 7: Interaction Effect (Synergy)

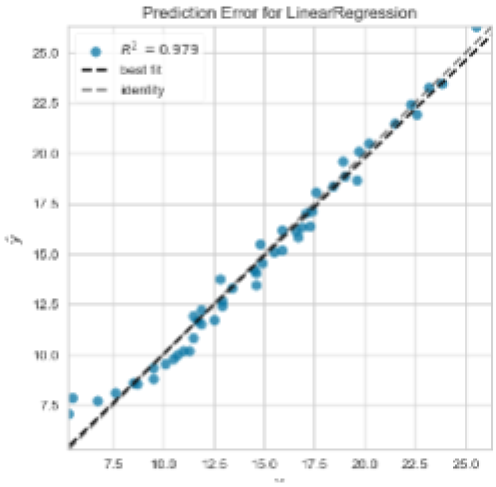
In [16]:
```
advert['interaction'] = advert['TV'] * advert['radio']
```

In [17]:
```
X = advert[['TV', 'radio', 'interaction']]
y = advert.sales

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1)

lm6 = LinearRegression()
lm6.fit(X_train, y_train)
lm6_preds = lm6.predict(X_test)

print("RMSE :", np.sqrt(mean_squared_error(y_test, lm6_preds)))
print("R^2: ", r2_score(y_test, lm6_preds))
```

```
RMSE : 0.7011871137164336
R^2:  0.978973681468126
```

In [18]:
```
visualizer = PredictionError(lm6)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
visualizer.poof()
```

```
C:\Users\rashm\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, but LinearReg
ression was fitted with feature names
  warnings.warn(
```



Experiment 4

In [1]:
```python
import numpy as np
import pandas as pd


import matplotlib as mpl
import matplotlib.pyplot as plt   # data visualization
import seaborn as sns             # statistical data visualization
```

In [2]:
```python
df = pd.read_csv(r"C:\Users\rashm\Downloads\AirPassengers.csv")

df.head()
```

Out[2]:

	Month	#Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121

In [3]:
```python
df.columns = ['Date','Number of Passengers']

df.head()
```

Out[3]:

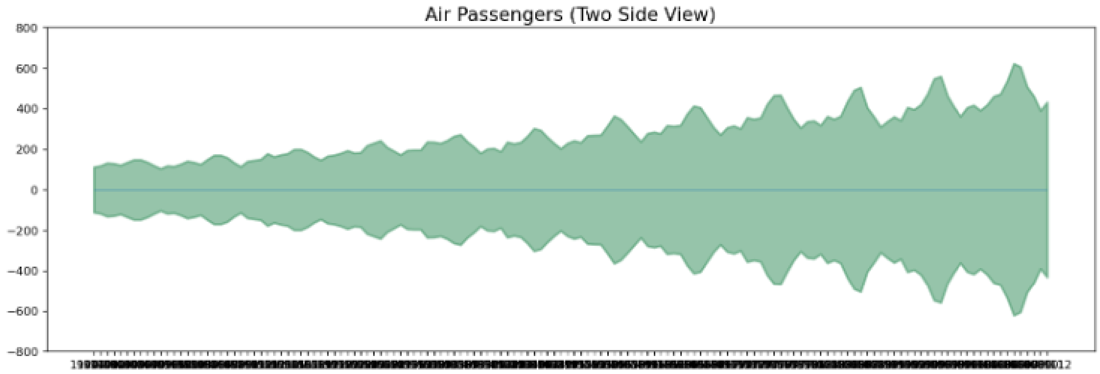
	Date	Number of Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121

Visualize the Time Series

In [4]:
```python
def plot_df(df, x, y, title="", xlabel='Date', ylabel='Number of Passengers', dpi=100):
    plt.figure(figsize=(15,4), dpi=dpi)
    plt.plot(x, y, color='tab:red')
    plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)
    plt.show()


plot_df(df, x=df['Date'], y=df['Number of Passengers'], title='Number of US Airline passengers from 1949 to 1960')
```

Since all the values are positive, we can show this on both sides of the Y axis to emphasize the growth.

```
In [5]: x = df['Date'].values
        y1 = df['Number of Passengers'].values

        # Plot
        fig, ax = plt.subplots(1, 1, figsize=(16,5), dpi= 120)
        plt.fill_between(x, y1=y1, y2=-y1, alpha=0.5, linewidth=2, color='seagreen')
        plt.ylim(-800, 800)
        plt.title('Air Passengers (Two Side View)', fontsize=16)
        plt.hlines(y=0, xmin=np.min(df['Date']), xmax=np.max(df['Date']), linewidth=.5)
        plt.show()
```



It can be seen that its a monthly time series and follows a certain repetitive pattern every year. So, we can plot each year as a separate line in the same plot. This let us compare the year wise patterns side-by-side.

Patterns in a Time Series

Any time series visualization may consist of the following components:

Base Level + Trend + Seasonality + Error.

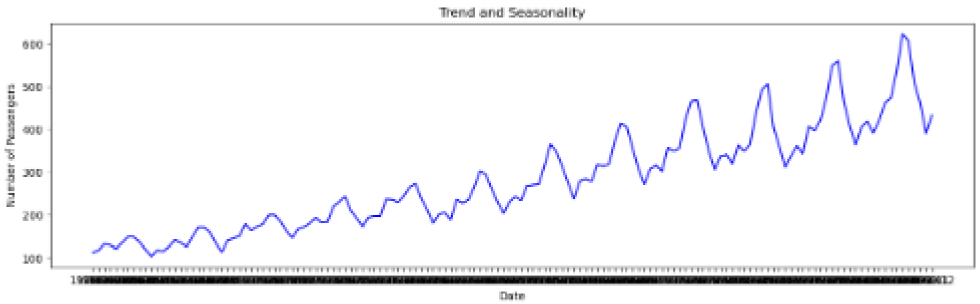
Trend A trend is observed when there is an increasing or decreasing slope observed in the time series.

Seasonality A seasonality is observed when there is a distinct repeated pattern observed between regular intervals due to seasonal factors.

It could be because of the month of the year, the day of the month, weekdays or even time of the day.

However, It is not mandatory that all time series must have a trend and/or seasonality. A time series may not have a distinct trend but have a seasonality and vice-versa.

```
n [6]: def plot_df(df, x, y, title="", xlabel='Date', ylabel='Number of Passengers', dpi=100):
           plt.figure(figsize=(15,4), dpi=dpi)
           plt.plot(x, y, color='blue')
           plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)
           plt.show()


       plot_df(df, x=df['Date'], y=df['Number of Passengers'], title='Trend and Seasonality')
```



Cyclic behaviour

Another important thing to consider is the cyclic behaviour. It happens when the rise and fall pattern in the series does not happen in fixed calendar-based intervals. We should not confuse 'cyclic' effect with 'seasonal' effect.

If the patterns are not of fixed calendar based frequencies, then it is cyclic. Because, unlike the seasonality, cyclic effects are typically influenced by the business and other socio-economic factors.

Additive and Multiplicative Time Series

We may have different combinations of trends and seasonality. Depending on the nature of the trends and seasonality, a time series can be modeled as an additive or multiplicative time series.

Each observation in the series can be expressed as either a sum or a product of the components.

Additive time series:

Value = Base Level + Trend + Seasonality + Error

Multiplicative Time Series:

Value = Base Level x Trend x Seasonality x Error

Decomposition of a Time Series

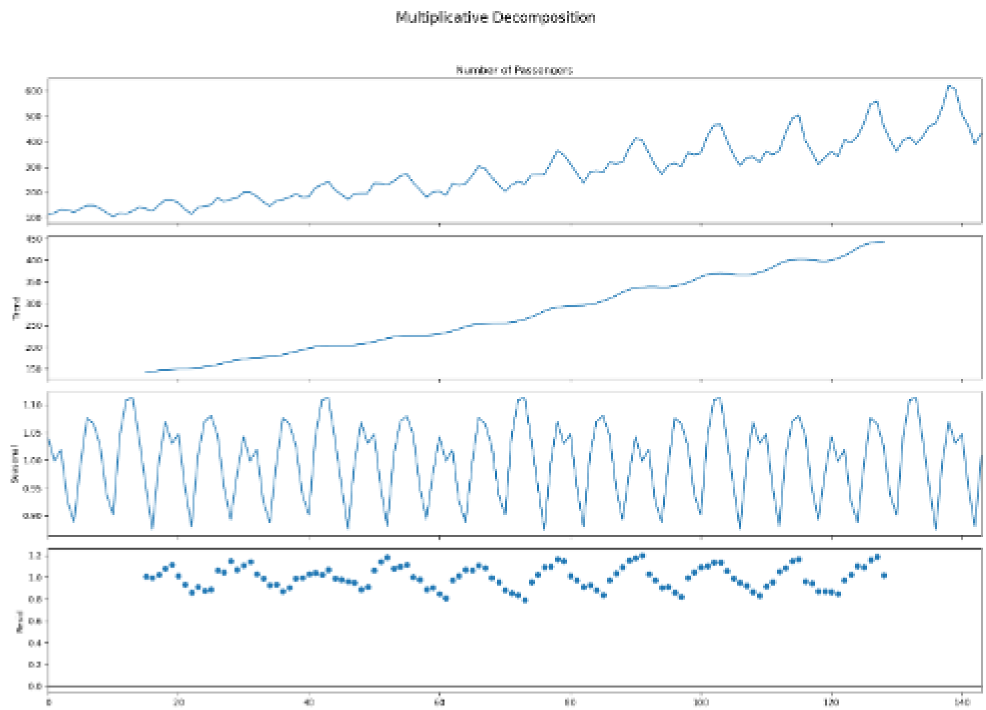
Decomposition of a time series can be performed by considering the series as an additive or multiplicative combination of the base level, trend, seasonal index and the residual term.

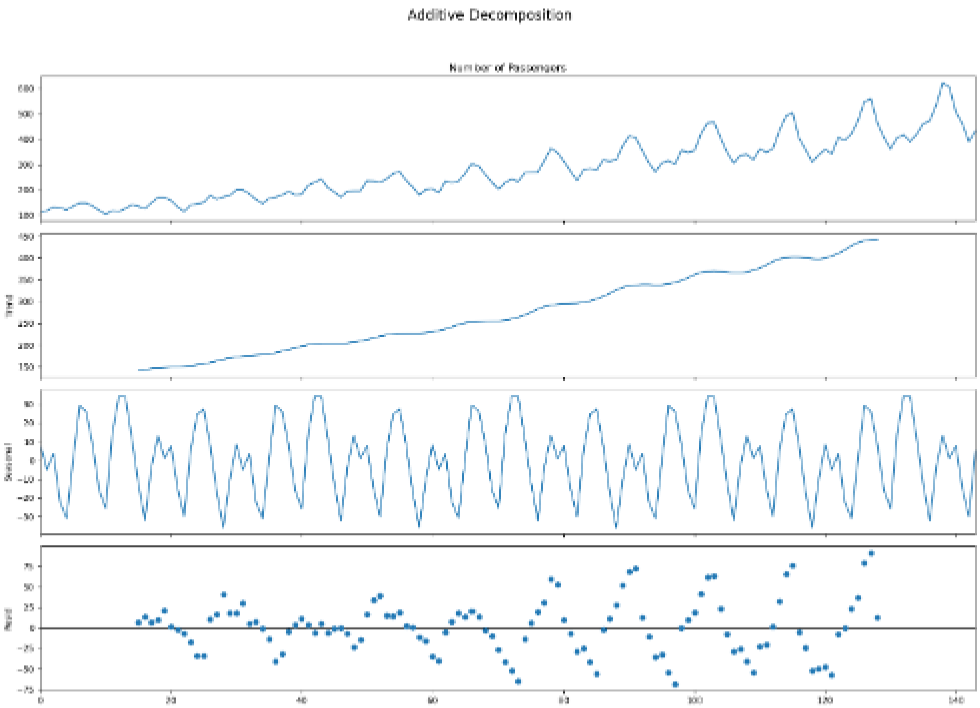
The seasonal_decompose in statsmodels implements this conveniently.

In [7]:

```
from statsmodels.tsa.seasonal import seasonal_decompose
from dateutil.parser import parse


# Multiplicative Decomposition
multiplicative_decomposition = seasonal_decompose(df['Number of Passengers'], model='multiplicative', period=30)

# Additive Decomposition
additive_decomposition = seasonal_decompose(df['Number of Passengers'], model='additive', period=30)

# Plot
plt.rcParams.update({'figure.figsize': (16,12)})
multiplicative_decomposition.plot().suptitle('Multiplicative Decomposition', fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

additive_decomposition.plot().suptitle('Additive Decomposition', fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

plt.show()
```

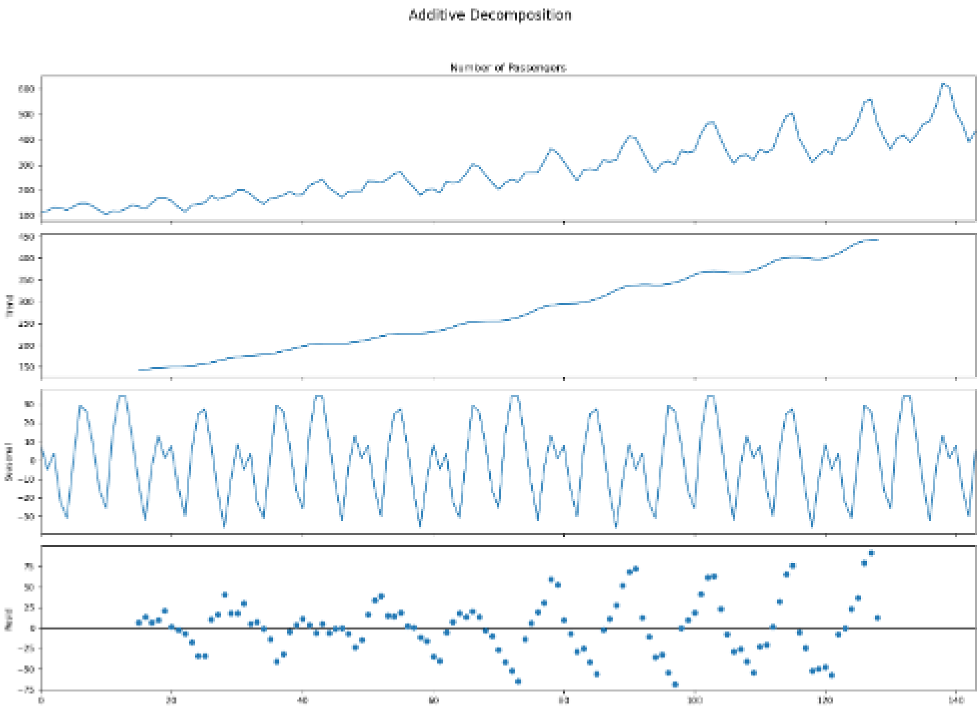
```
C:\Users\rashm\AppData\Local\Temp\ipykernel_22448\4037360255.py:14: UserWarning: The figure layout has changed to tight
  plt.tight_layout(rect=[0, 0.03, 1, 0.95])
C:\Users\rashm\AppData\Local\Temp\ipykernel_22448\4037360255.py:17: UserWarning: The figure layout has changed to tight
  plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```





In [8]:
```
plt.rcParams.update({'figure.figsize': (16,12)})
additive_decomposition.plot().suptitle('Additive Decomposition', fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

plt.show()
```

```
C:\Users\rashm\AppData\Local\Temp\ipykernel_22448\3088285556.py:3: UserWarning: The figure layout has changed to tight
  plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```



If we look at the residuals of the additive decomposition closely, it has some pattern left over.

The multiplicative decomposition, looks quite random which is good. So ideally, multiplicative decomposition should be preferred for this particular series.

Stationary and Non-Stationary Time Series

Now, we wil discuss Stationary and Non-Stationary Time Series.

Stationarity is a property of a time series. A stationary series is one where the values of the series is not a function of time. So, the values are independent of time.

Hence the statistical properties of the series like mean, variance and autocorrelation are constant over time.

Autocorrelation of the series is nothing but the correlation of the series with its previous values.

A stationary time series is independent of seasonal effects as well.

Now, we will plot some examples of stationary and non-stationary time series for clarity.

Experiment 5

Data Loading

We will load our stock price dataset with the "Date" column as index.

In [8]:
```
import pandas as pd

net_df = pd.read_csv(r"C:\Users\rashm\Downloads\NFLX.csv", index_col="Date", parse_dates=True)
net_df.head(3)
```

Out[8]:

Date	Open	High	Low	Close	Adj Close	Volume
2010-01-04	7.931429	7.961429	7.565714	7.640000	7.640000	17239600
2010-01-05	7.652857	7.657143	7.258571	7.358571	7.358571	23753100
2010-01-06	7.361429	7.672857	7.197143	7.617143	7.617143	23290400

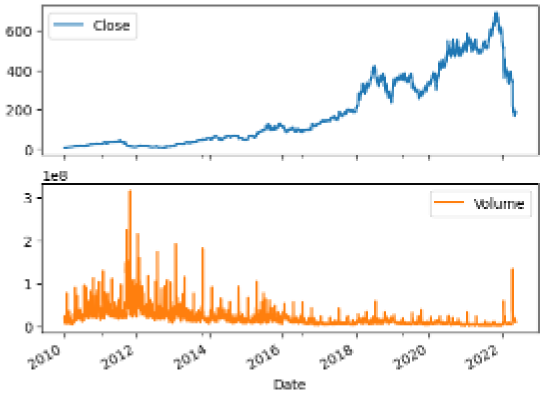
Data Visualization

We can use pandas 'plot' function to visualize the changes in stock price and volume over time.

It's clear that the stock prices are increasing exponentially.

In [ ]:

In [3]:
```
net_df[["Close","Volume"]].plot(subplots=True, layout=(2,1));
```



Rolling Forecast ARIMA Model

Our dataset has been split into training and test sets, and we proceeded to train an ARIMA model. The first prediction was then forecasted.

We received a poor outcome with the generic ARIMA model, as it produced a flat line. Therefore, we have decided to try a rolling forecast method.

In [9]:
```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math


train_data, test_data = net_df[0:int(len(net_df)*0.9)], net_df[int(len(net_df)*0.9):]


train_arima = train_data['Open']
test_arima = test_data['Open']


history = [x for x in train_arima]
y = test_arima
# make first prediction
predictions = list()
model = ARIMA(history, order=(1,1,0))
model_fit = model.fit()
yhat = model_fit.forecast()[0]
predictions.append(yhat)
history.append(y[0])
```

When dealing with time series data, a rolling forecast is often necessary due to the dependence on prior observations.

One way to do this is to re-create the model after each new observation is received.

To keep track of all observations, we can manually maintain a list called history, which initially contains training data and to which new observations are appended each iteration.

This approach can help us get an accurate forecasting model.

In [ ]:

In [10]:
```
# rolling forecasts
for i in range(1, len(y)):
    # predict
    model = ARIMA(history, order=(1,1,0))
    model_fit = model.fit()
    yhat = model_fit.forecast()[0]
    # invert transformed prediction
    predictions.append(yhat)
    # observation
    obs = y[i]
    history.append(obs)
```

Model Evaluation

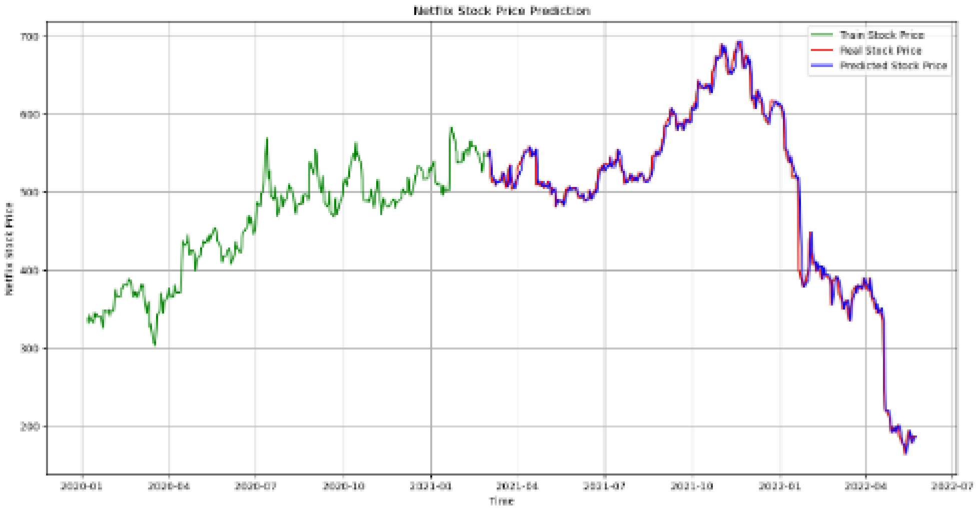
Our rolling forecast ARIMA model showed a 100% improvement over simple implementation, yielding impressive results.

In [11]:
```
# report performance
mse = mean_squared_error(y, predictions)
print('MSE: '+str(mse))
mae = mean_absolute_error(y, predictions)
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(y, predictions))
print('RMSE: '+str(rmse))
```

```
MSE: 170.1585432524581
MAE: 8.012871071857338
RMSE: 13.04448324972891
```

Let's visualize and compare the actual results to the predicted ones . It's clear that our model has made highly accurate predictions.

In [12]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(16,8))
plt.plot(net_df.index[-600:], net_df['Open'].tail(600), color='green', label = 'Train Stock Price')
plt.plot(test_data.index, y, color = 'red', label = 'Real Stock Price')
plt.plot(test_data.index, predictions, color = 'blue', label = 'Predicted Stock Price')
plt.title('Netflix Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Netflix Stock Price')
plt.legend()
plt.grid(True)
plt.savefig('arima_model.pdf')
plt.show()
```



Conclusion

Here, we provided an overview of ARIMA models and how to implement them in Python for time series forecasting.

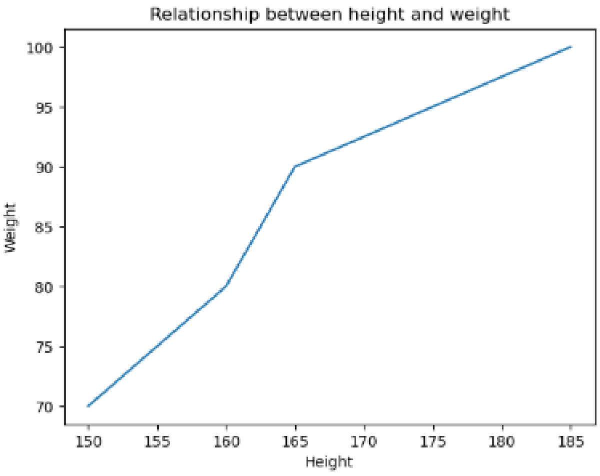
The ARIMA approach provides a flexible and structured way to model time series data that relies on prior observations as well as past prediction errors.

Experiment 6

Matplotlib Inline

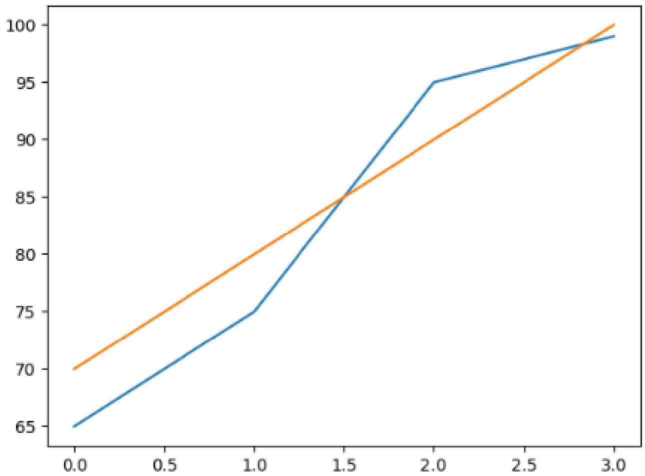
In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
#sample plot
height=[150,160,165,185]
weight=[70,80,90,100]
plt.plot(height,weight)
plt.title("Relationship between height and weight")
plt.xlabel("Height")
plt.ylabel("Weight")
```

Out[1]:
```
Text(0, 0.5, 'Weight')
```



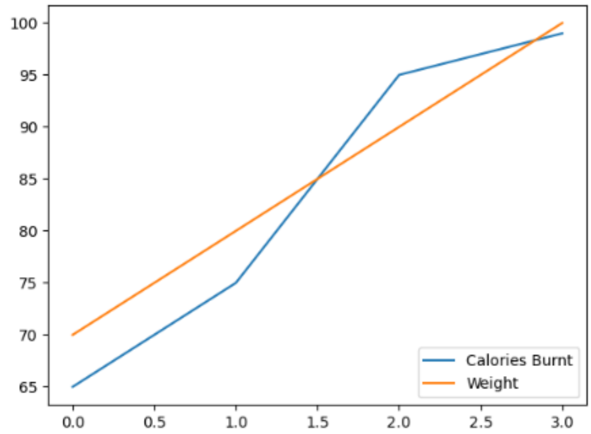
In [2]:
```
calories_burnt=[65,75,95,99]
plt.plot(calories_burnt)
plt.plot(weight)
```

Out[2]:
```
[<matplotlib.lines.Line2D at 0x1ad4e9f56d0>]
```

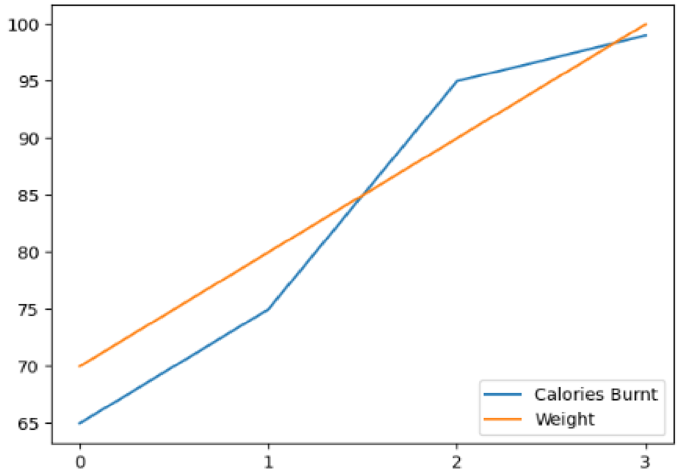


In [4]:
```
plt.plot(calories_burnt)
plt.plot(weight)
plt.legend(labels=['Calories Burnt','Weight'],loc='lower right')
```

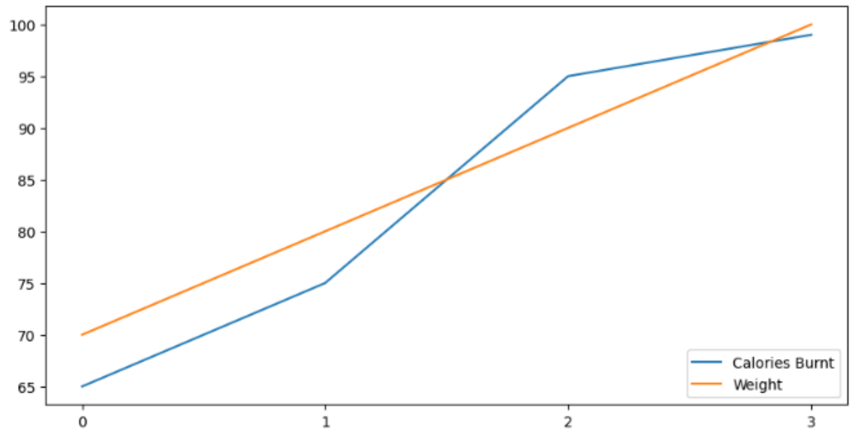
Out[4]: <matplotlib.legend.Legend at 0x1ad4ea8b290>



In [14]:
```
plt.plot(calories_burnt)
plt.plot(weight)
plt.legend(labels=['Calories Burnt','Weight'],loc='lower right')
plt.xticks(ticks=[0,1,2,3],label=['p1','p2','p3','p4']);
```

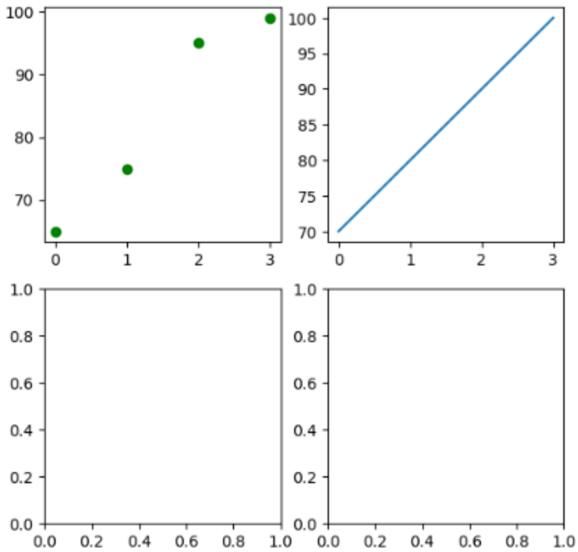


In [13]:
```
plt.figure(figsize=(10,5))
plt.plot(calories_burnt)
plt.plot(weight)
plt.legend(labels=['Calories Burnt','Weight'],loc='lower right')
plt.xticks(ticks=[0,1,2,3],label=['p1','p2','p3','p4']);
```

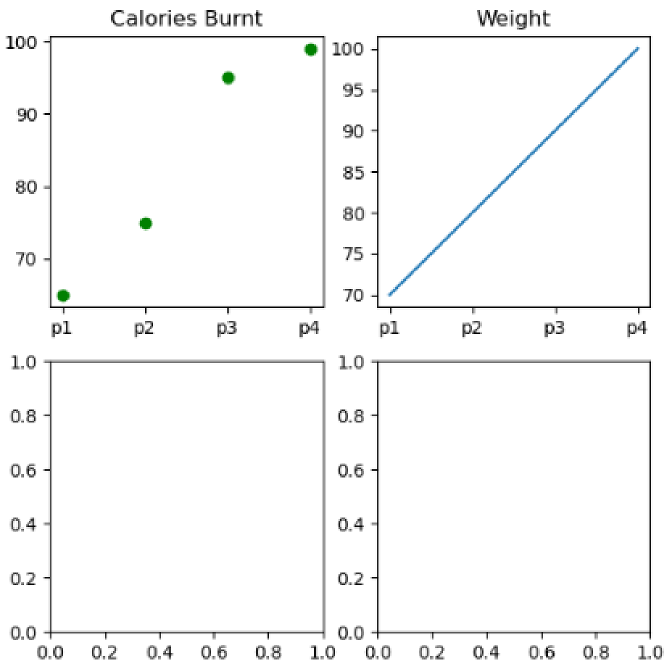


In [15]:
```
fig,ax=plt.subplots(nrows=2,ncols=2,figsize=(6,6))
ax[0,0].plot(calories_burnt,'go')
ax[0,1].plot(weight)
```

Out[15]: [<matplotlib.lines.Line2D at 0x1ad534c7590>]



In [17]:
```
fig,ax=plt.subplots(nrows=2,ncols=2,figsize=(6,6))
ax[0,0].plot(calories_burnt,'go')
ax[0,1].plot(weight)

ax[0,0].set_title("Calories Burnt")
ax[0,1].set_title("Weight")

ax[0,0].set_xticks(ticks=[0,1,2,3]);
ax[0,1].set_xticks(ticks=[0,1,2,3]);

ax[0,0].set_xticklabels(labels=['p1','p2','p3','p4']);
ax[0,1].set_xticklabels(labels=['p1','p2','p3','p4']);
```



Line Chart

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

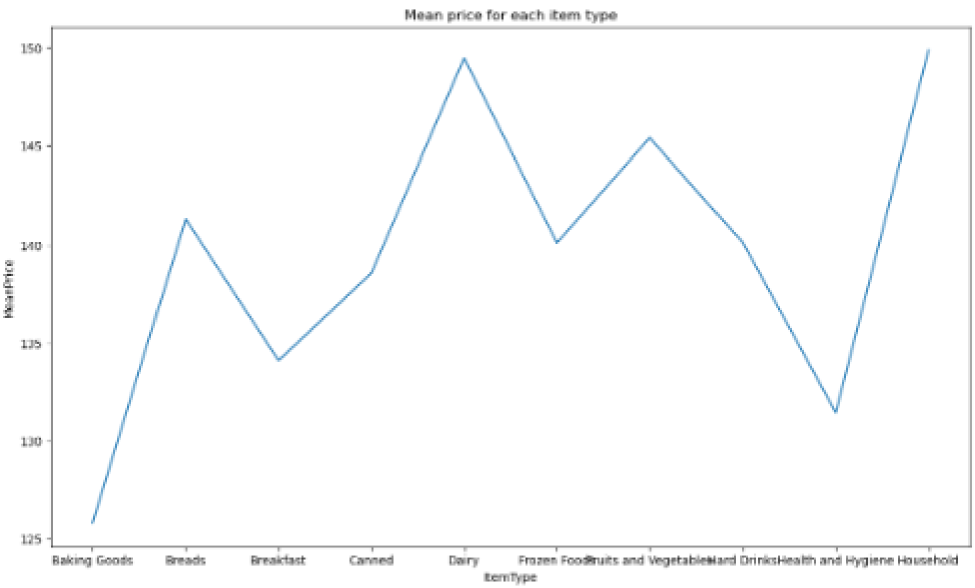
In [5]:
```
data_BM=pd.read_csv(r'C:\Users\HP\Downloads\archive.zip')
data_BM=data_BM.dropna(how="any")
data_BM.head()
```

Out[5]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	
5	FDP36	10.395	Regular	0.000000	Baking Goods	51.4008	OUT018	2009	Medium	

In [25]:
```
price_by_item=data_BM.groupby('Item_Type').Item_MRP.mean()[:10]
x=price_by_item.index.tolist()
y=price_by_item.values.tolist()

plt.figure(figsize=(14,8))

plt.title('Mean price for each item type')

plt.xlabel('ItemType')
plt.ylabel('MeanPrice')

plt.xticks(labels=x,ticks=(np.arange(len(x))))
plt.plot(x,y)
```

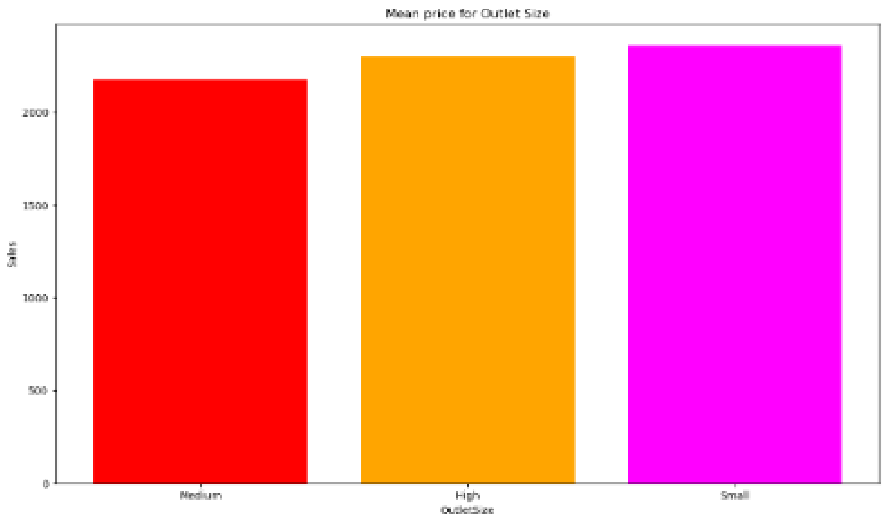
Out[25]:
```
[<matplotlib.lines.Line2D at 0x1d78a95ea10>]
```



Bar Chart

In [28]:
```
sales_by_outlet_size=data_BM.groupby('Outlet_Size').Item_Outlet_Sales.mean()
sales_by_outlet_size.sort_values(inplace=True)

x=sales_by_outlet_size.index.tolist()
y=sales_by_outlet_size.values.tolist()

plt.figure(figsize=(14,8))

plt.title('Mean price for Outlet Size')

plt.xlabel('OutletSize')
plt.ylabel('Sales')

plt.xticks(labels=x,ticks=(np.arange(len(x))))
plt.bar(x,y,color=['red','orange','magenta'])
```

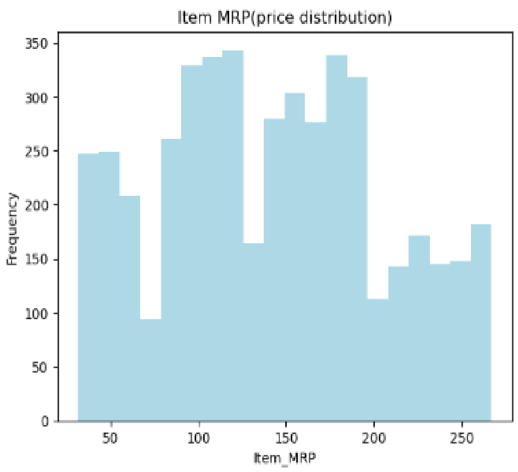
Out[28]:
```
<BarContainer object of 3 artists>
```



Histogram

In [29]:
```
plt.title('Item MRP(price distribution)')

plt.xlabel('Item_MRP')
plt.ylabel('Frequency')

plt.hist(data_BM['Item_MRP'],bins=20,color='lightblue')
```

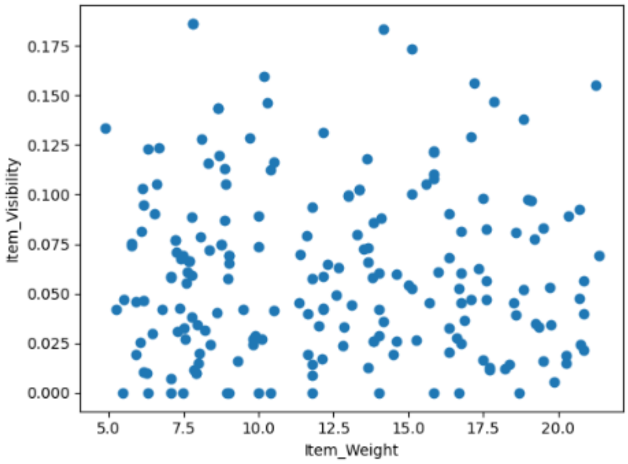
Out[29]:
```
(array([247., 249., 208.,  94., 261., 329., 337., 343., 164., 279., 304.,
        276., 339., 318., 113., 143., 171., 145., 148., 182.]),
 array([ 31.49    ,  43.25992,  55.02984,  66.79976,  78.56968,  90.3396 ,
        102.10952, 113.87944, 125.64936, 137.41928, 149.1892 , 160.95912,
        172.72904, 184.49896, 196.26888, 208.0388 , 219.80872, 231.57864,
        243.34856, 255.11848, 266.8884 ]),
 <BarContainer object of 20 artists>)
```



Scatter Plot

In [36]:
```
plt.xlabel('Item_Weight')
plt.ylabel('Item_Visibility')

plt.scatter(data_BM['Item_Weight'][:200],data_BM['Item_Visibility'][:200])
```

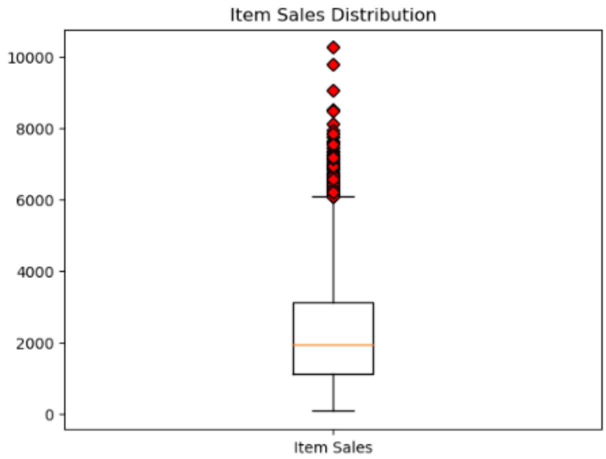
Out[36]: <matplotlib.collections.PathCollection at 0x1d78fd009d0>



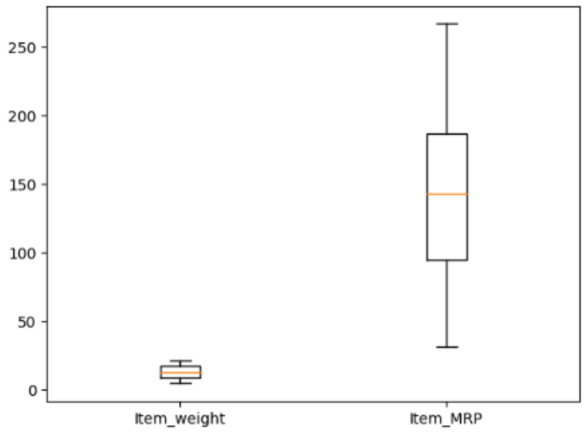
Experiment 7

Box plot

In [31]:
```
data=data_BM[['Item_Outlet_Sales']]
red_diamond=dict(markerfacecolor='r',marker='D')
plt.title('Item Sales Distribution')
plt.boxplot(data.values,labels=['Item Sales'],flierprops=red_diamond);
```



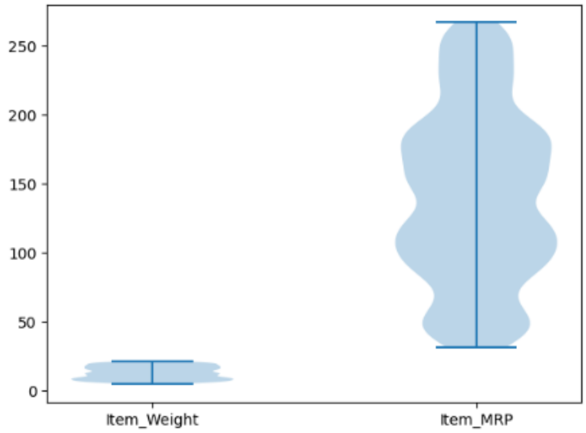
In [33]:
```
data=data_BM[['Item_Weight','Item_MRP']]
red_diamond=dict(markerfacecolor='r',marker='D')
fig,ax=plt.subplots()
plt.boxplot(data.values,labels=['Item_weight','Item_MRP']);
```



Violin plot

In [35]:
```
data=data_BM[['Item_Weight','Item_MRP']]
fig,ax=plt.subplots()
plt.xticks(ticks=[1,2],labels=['Item_Weight','Item_MRP'])
plt.violinplot(data.values)
```

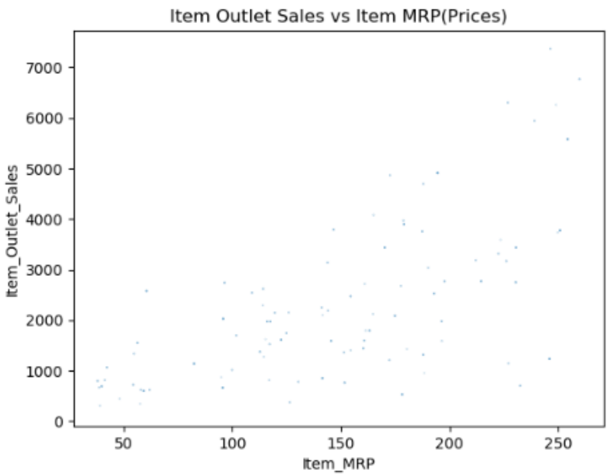
Out[35]:
```
{'bodies': [<matplotlib.collections.PolyCollection at 0x1d78fe52490>,
  <matplotlib.collections.PolyCollection at 0x1d78fe6ba90>],
 'cmaxes': <matplotlib.collections.LineCollection at 0x1d786022110>,
 'cmins': <matplotlib.collections.LineCollection at 0x1d78fe763d0>,
 'cbars': <matplotlib.collections.LineCollection at 0x1d78b0ce010>}
```



Bubble plot

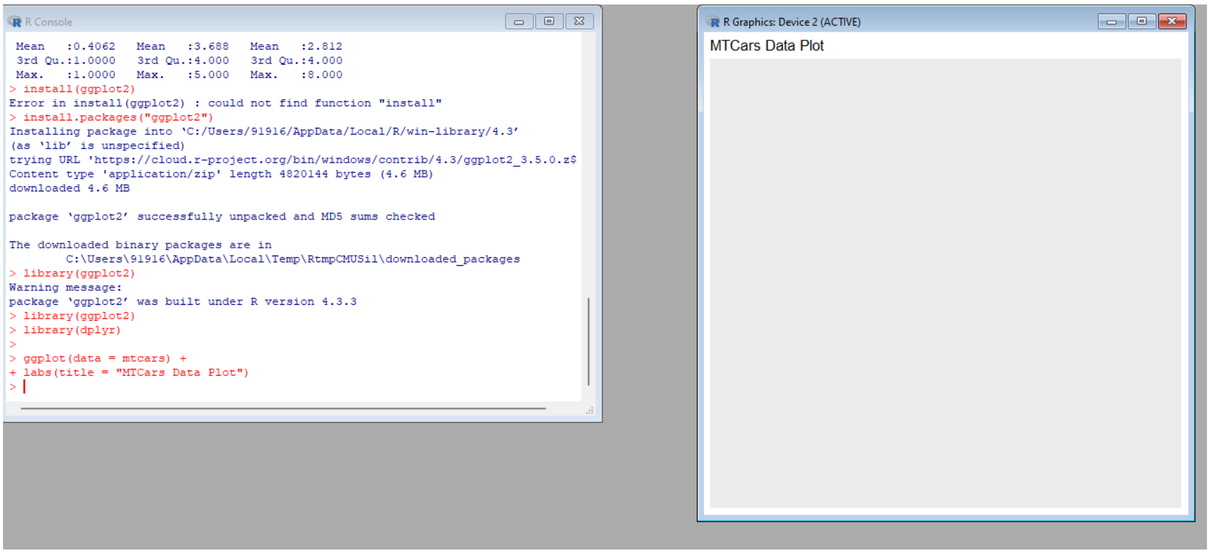
In [39]:
```
plt.xlabel('Item_MRP')
plt.ylabel('Item_Outlet_Sales')
plt.title('Item Outlet Sales vs Item MRP(Prices)')
plt.scatter(data_BM['Item_MRP'][:100],data_BM['Item_Outlet_Sales'][:100],data_BM['Item_Visibility'][:100])
```

Out[39]:
```
<matplotlib.collections.PathCollection at 0x1d790e30610>
```



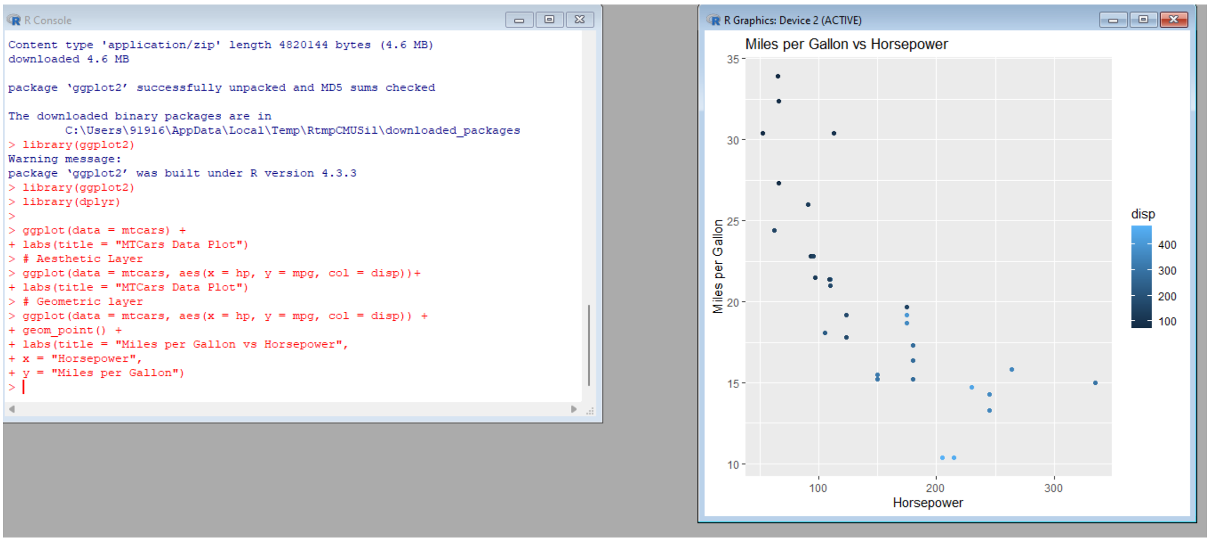
Experiment 8

Data Layer

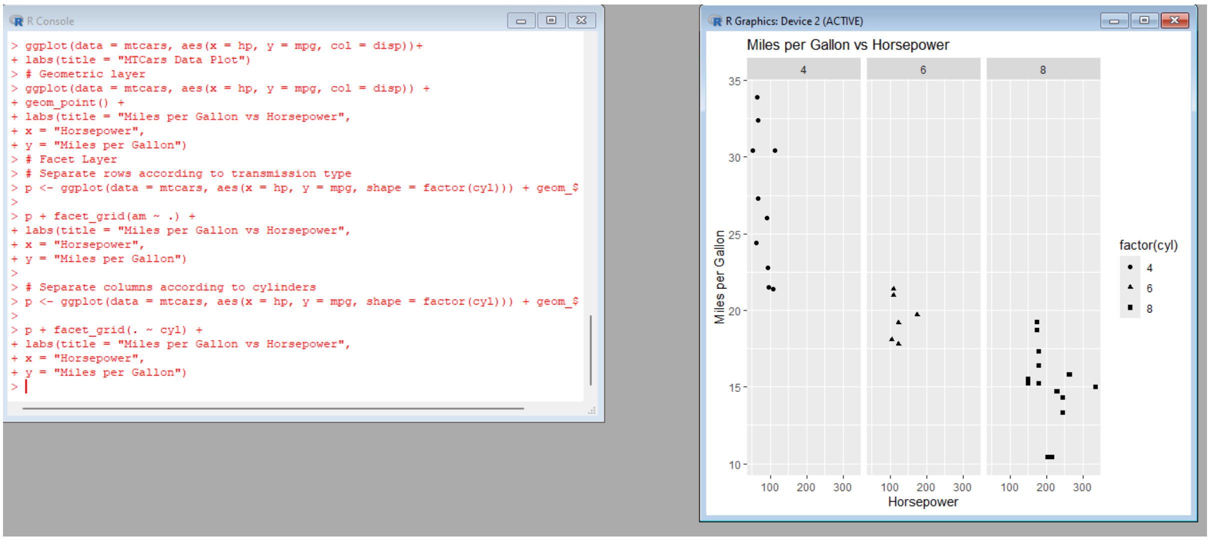


Aesthetic Layer

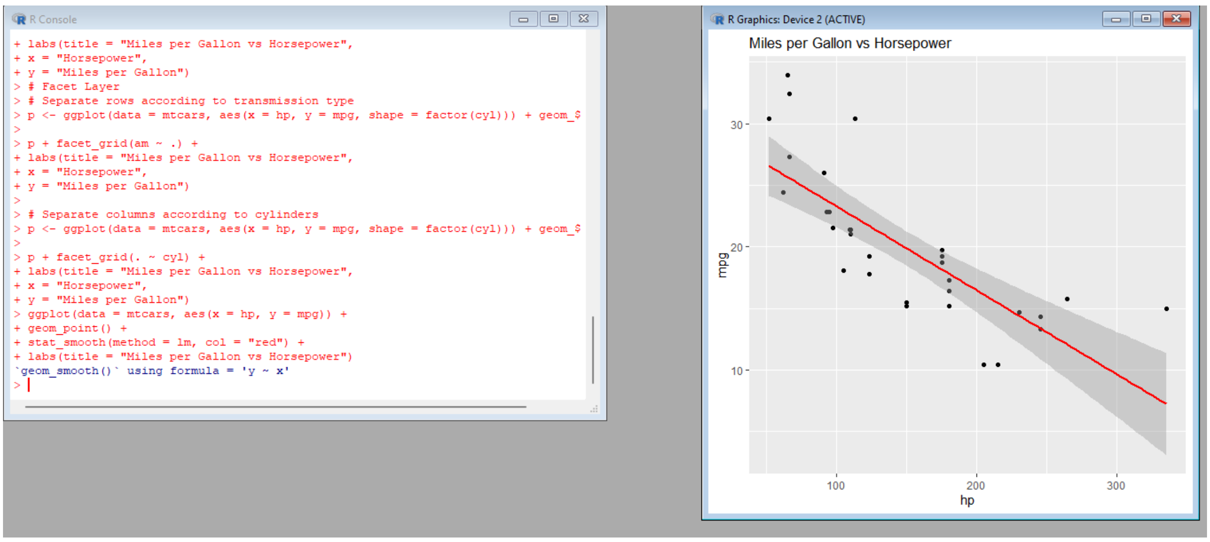
Geometric Layer



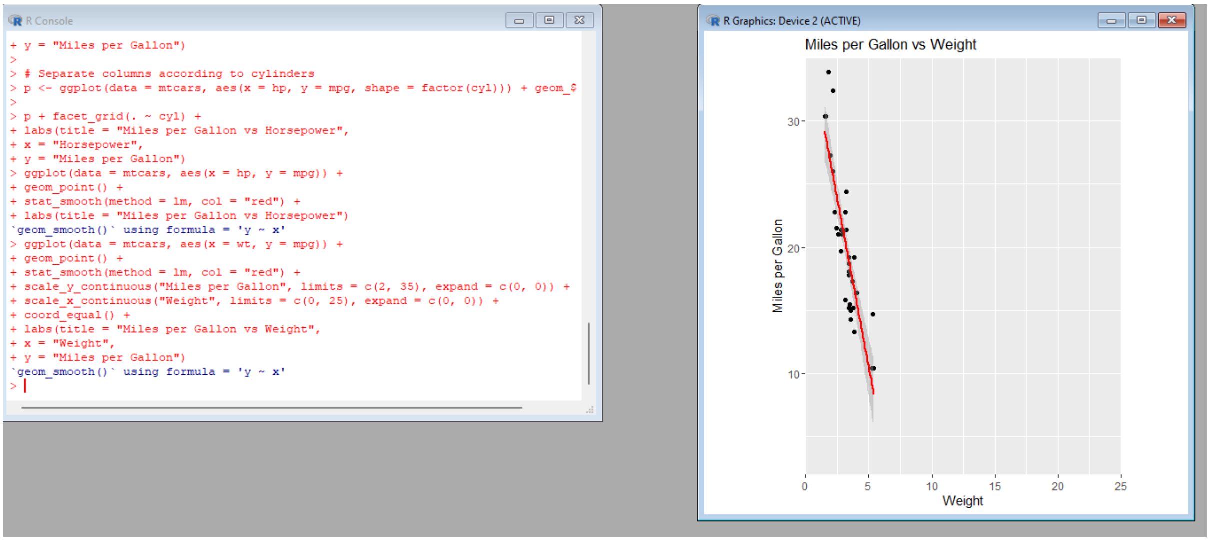
Facet Layer



Statistics Layer



Coordinates Layer



Theme Layer