In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

წრფივი რეგრესია SciKit-Learn-ში ¶

ჩვენ ვნახეთ, თუ როგორ შეიძლება მარტივი წრფივი რეგრესიის აგება.

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

Data

ჩვენ გამოვიყენებთ მონაცემებს წიგნიდან ISLR (Introduction to Statistical Learning: With Applications in R). ეს იქნება კონკრეტული პროდუქტის გაყიდვები (ათასობით ერთეულში) სატელევიზიო, რადიო და გაზეთების სარეკლამო ბიუჯეტის ფუნქციით (ათასობით დოლარში).

In [ ]:
```
# df = pd.read_csv("Advertising.csv")
df=pd.read_csv("/content/drive/MyDrive/Dataset/Advertising.csv")
```

In [ ]:
```
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

ადრე ჩვენ ვეძებდით პასუხს კითხვაზე ** არის თუ არა კავშირი რეკლამის ჯამურ დანახარჯსა და გაყიდვებს შორის?** და ასევე ვცდილობდით გაყიდვების პროგნოზირებას რეკლამის მთლიანი ბიუჯეტის მოცემულ მნიშვნელობებზე დაყრდნობით. ახლა დავსვათ უფრო ფართო კითხვა - რა კავშირია სარეკლამო არხებსა (ტელევიზია, რადიო, გაზეთები) და გაყიდვებს შორის?

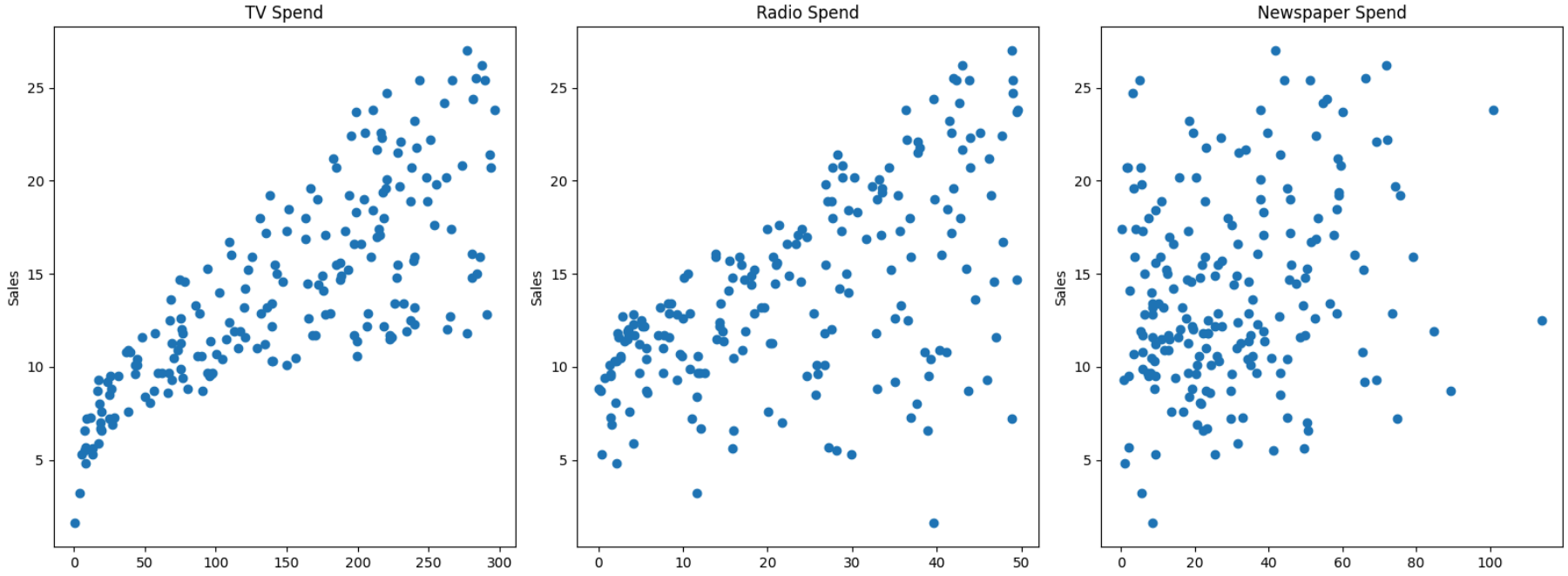
მრავალი ნიშნიანი (N-განზომილებიანი სივრცე)

In [ ]:
```
fig,axes = plt.subplots(nrows=1,ncols=3,figsize=(16,6))

axes[0].plot(df['TV'],df['sales'],'o')
axes[0].set_ylabel("Sales")
axes[0].set_title("TV Spend")

axes[1].plot(df['radio'],df['sales'],'o')
axes[1].set_title("Radio Spend")
axes[1].set_ylabel("Sales")

axes[2].plot(df['newspaper'],df['sales'],'o')
axes[2].set_title("Newspaper Spend");
axes[2].set_ylabel("Sales")
plt.tight_layout();
```



SciKit Learn-ის შესავალი

In [ ]:
```
X = df.drop('sales',axis=1)
y = df['sales']
X
```

Out[6]:

	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
...	...	...	...
195	38.2	3.7	13.8
196	94.2	4.9	8.1
197	177.0	9.3	6.4
198	283.6	42.0	66.2
199	232.1	8.6	8.7

200 rows × 3 columns

In [ ]:
```
# დაყოფა სასწავლო და სატესტო ნიმუშებად - Train | Test Split
from sklearn.model_selection import train_test_split
```

In [ ]:
```
#help(train_test_split)
```

In [ ]:
```
# random_state: https://stackoverflow.com/questions/28064634/random-state-pseudo-random-number-in-scikit-learn

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

In [ ]:
```
X_train
```

Out[10]:

	TV	radio	newspaper
85	193.2	18.4	65.7
183	287.6	43.0	71.8
127	80.2	0.0	9.2
53	182.6	46.2	58.7
100	222.4	4.3	49.8
...	...	...	...
63	102.7	29.6	8.4
70	199.1	30.6	38.7
81	239.8	4.1	36.9
11	214.7	24.0	4.0
95	163.3	31.6	52.9

140 rows × 3 columns

In [ ]:
```
y_train
```

Out[11]:
```
85     15.2
183    26.2
127     8.8
53     21.2
100    11.7
       ...
63     14.0
70     18.3
81     12.3
11     17.4
95     16.9
Name: sales, Length: 140, dtype: float64
```

In [ ]:
```
X_test
```

Out[12]:

	TV	radio	newspaper
37	74.7	49.4	45.7
109	255.4	26.9	5.5
31	112.9	17.4	38.6
89	109.8	47.8	51.4
66	31.5	24.6	2.2
119	19.4	16.0	22.3
54	262.7	28.8	15.9
74	213.4	24.6	13.1
145	140.3	1.9	9.0
142	220.5	33.2	37.9
148	38.0	40.3	11.9
112	175.7	15.4	2.4
174	222.4	3.4	13.1
55	198.9	49.4	60.0
141	193.7	35.4	75.6
149	44.7	25.8	20.6
25	262.9	3.5	19.5
34	95.7	1.4	7.4
170	50.0	11.6	18.4
39	228.0	37.7	32.0
172	19.6	20.1	17.0
153	171.3	39.7	37.7
175	276.9	48.9	41.8
61	261.3	42.7	54.7
65	69.0	9.3	0.9
50	199.8	3.1	34.6
42	293.6	27.7	1.8
129	59.6	12.0	43.1
179	165.6	10.0	17.6
2	17.2	45.9	69.3
12	23.8	35.1	65.9
133	219.8	33.5	45.1

	TV	radio	newspaper
90	134.3	4.9	9.3
22	13.2	15.9	49.6
41	177.0	33.4	38.7
32	97.2	1.5	30.0
125	87.2	11.8	25.9
196	94.2	4.9	8.1
158	11.7	36.9	45.2
180	156.6	2.6	8.3
16	67.8	36.6	114.0
186	139.5	2.1	26.6
144	96.2	14.8	38.9
121	18.8	21.7	50.4
80	76.4	26.7	22.3
18	69.2	20.5	18.3
78	5.4	29.9	9.4
48	227.2	15.8	49.9
4	180.8	10.8	58.4
15	195.4	47.7	52.9
1	44.5	39.3	45.1
43	206.9	8.4	26.4
102	280.2	10.1	21.4
164	117.2	14.7	5.4
9	199.8	2.6	21.2
155	4.1	11.6	5.7
36	266.9	43.8	5.0
190	39.5	41.1	5.8
33	265.6	20.0	0.3
45	175.1	22.5	31.5

In [ ]:
```
y_test
```

Out[13]:

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

```
1      10.4
43     12.9
102    14.8
164    11.9
9      10.6
155     3.2
36     25.4
190    10.8
33     17.4
45     14.9
Name: sales, dtype: float64
```

In [ ]:
```
len(df)
```

Out[14]: 200

In [ ]:
```
len(X_train)
```

Out[15]: 140

In [ ]:
```
len(X_test)
```

Out[16]: 60

შევქმნათ მოდელი (Scikit-Learn-ის თვალსაზრისით, ეს არის მოდელი Estimator (შემფასებელი))

მოდელის კლასის იმპორტირება

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

In [ ]:
```
# პარამეტრები და ჰიპერპარამეტრები
# help (LinearRegression)
```

In [ ]:
```
model = LinearRegression()
```

მოდელის დასწავლა (Fit/Train) სასწავლო მონაცემებზე

ტრენინგი არ უნდა ჩატარდეს ყველა მონაცემზე, მხოლოდ სასწავლო მონაცემებზე; მაშინ გვექნება შესაძლებლობა შევაფასოთ მოდელის მოქმედება ტესტის მონაცემებზე, რათა შევამოწმოთ მომავალში უცნობ მონაცემებთან მუშაობა.

In [ ]:
```
model.fit(X_train, y_train)
```

Out[20]:
```
LinearRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

მოდელის მუშაობის დეტალები

მოდელის მუშაობის შეფასება ტესტის მონაცემებზე

მეტრიკა

რეგრესიის პრობლემებისთვის ყველაზე ხშირად გამოიყენება შემდეგი სამი მეტრიკა:

** საშუალო აბსოლუტური შეცდომა ** **Mean Absolute Error (MAE)** - საშუალო აბსოლუტური შეცდომის მნიშვნელობებს:

** საშუალო კვადრატული შეცდომა ** **Mean Squared Error (MSE)** - საშუალო კვადრატულ შეცდომებს:

საშუალო კვადრატული გადახრა Root Mean Squared Error (RMSE) - საშუალო კვადრატული შეცდომის კვადრატული ფესვი:

მეტრიკული შედარება: MAE ყველაზე მარტივი გასაგებია - ეს მხოლოდ საშუალო შეცდომაა. MSE უფრო პოპულარულია ვიდრე MAE, რადგან MSE უფრო "სჯის" დიდ შეცდომებს და ჩვეულებრივ უფრო გამოყენებადია აპლიკაციებში. RMSE კიდევ უფრო პოპულარულია ვიდრე MSE, რადგან RMSE იზომება იმავე ერთეულებში, როგორც"y". ყველა ეს მეტრიკა დაკარგვის ფუნქციებია (loss functions), რადგან ჩვენ გვინდა მათი შემცირება.

გამოთვალეთ მოდელის მუშაობა ტესტის მონაცემებზე

მოდელის მუშაობის სწორად შესაფასებლად, ჩვენ ვიანგარიშებთ მეტრიკას ტესტის მონაცემთა ნაკრების შესახებ (ეს არის ის მონაცემები, რომლებიც მოდელს არასოდეს უნახავს ტრენინგის პროცესში).

In [ ]:
```
X_test
```

Out[21]:

	TV	radio	newspaper
37	74.7	49.4	45.7
109	255.4	26.9	5.5
31	112.9	17.4	38.6
89	109.8	47.8	51.4
66	31.5	24.6	2.2
119	19.4	16.0	22.3
54	262.7	28.8	15.9
74	213.4	24.6	13.1
145	140.3	1.9	9.0
142	220.5	33.2	37.9
148	38.0	40.3	11.9
112	175.7	15.4	2.4
174	222.4	3.4	13.1
55	198.9	49.4	60.0
141	193.7	35.4	75.6
149	44.7	25.8	20.6
25	262.9	3.5	19.5
34	95.7	1.4	7.4
170	50.0	11.6	18.4
39	228.0	37.7	32.0
172	19.6	20.1	17.0
153	171.3	39.7	37.7
175	276.9	48.9	41.8
61	261.3	42.7	54.7
65	69.0	9.3	0.9
50	199.8	3.1	34.6
42	293.6	27.7	1.8
129	59.6	12.0	43.1
179	165.6	10.0	17.6
2	17.2	45.9	69.3
12	23.8	35.1	65.9
133	219.8	33.5	45.1

	TV	radio	newspaper
90	134.3	4.9	9.3
22	13.2	15.9	49.6
41	177.0	33.4	38.7
32	97.2	1.5	30.0
125	87.2	11.8	25.9
196	94.2	4.9	8.1
158	11.7	36.9	45.2
180	156.6	2.6	8.3
16	67.8	36.6	114.0
186	139.5	2.1	26.6
144	96.2	14.8	38.9
121	18.8	21.7	50.4
80	76.4	26.7	22.3
18	69.2	20.5	18.3
78	5.4	29.9	9.4
48	227.2	15.8	49.9
4	180.8	10.8	58.4
15	195.4	47.7	52.9
1	44.5	39.3	45.1
43	206.9	8.4	26.4
102	280.2	10.1	21.4
164	117.2	14.7	5.4
9	199.8	2.6	21.2
155	4.1	11.6	5.7
36	266.9	43.8	5.0
190	39.5	41.1	5.8
33	265.6	20.0	0.3
45	175.1	22.5	31.5

In [ ]:
```
# მხოლოდ ტესტის მონაცემებს ვაგზავნით
# მოდელი აკეთებს პროგნოზებს - y მნიშვნელობები ქუდით
# ახლა ჩვენ შეგვიძლია შევადაროთ პროგნოზირებული მნიშვნელობები y-ის ცნობილ ნამდვილ მნიშვნელობებს.
test_predictions = model.predict(X_test)
test_predictions
```

Out[22]:
```
array([15.74131332, 19.61062568, 11.44888935, 17.00819787,  9.17285676,
        7.01248287, 20.28992463, 17.29953992,  9.77584467, 19.22194224,
       12.40503154, 13.89234998, 13.72541098, 21.28794031, 18.42456638,
        9.98198406, 15.55228966,  7.68913693,  7.55614992, 20.40311209,
        7.79215204, 18.24214098, 24.68631904, 22.82199068,  7.97962085,
       12.65207264, 21.46925937,  8.05228573, 12.42315981, 12.50719678,
       10.77757812, 19.24460093, 10.070269  ,  6.70779999, 17.31492147,
        7.76764327,  9.25393336,  8.27834697, 10.58105585, 10.63591128,
       13.01002595,  9.77192057, 10.21469861,  8.04572042, 11.5671075 ,
       10.08368001,  8.99806574, 16.25388914, 13.23942315, 20.81493419,
       12.49727439, 13.96615898, 17.56285075, 11.14537013, 12.56261468,
        5.50870279, 23.29465134, 12.62409688, 18.77399978, 15.18785675])
```

In [ ]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error
```

In [ ]:
```
MAE = mean_absolute_error(y_test,test_predictions)
MSE = mean_squared_error(y_test,test_predictions)
RMSE = np.sqrt(MSE)
```

In [ ]:
```
MAE
```

Out[25]:
```
1.213745773614481
```

In [ ]:
```
MSE
```

Out[26]:
```
2.2987166978863796
```

In [ ]:
```
RMSE
```

Out[27]:
```
1.5161519375993884
```