The Sparks Foundation Task-1

Prediction using Supervised ML

Linear Regression with Python

In this section we will see how the Python library for machine learning can be used to implement regression functions. We will start with simple linear regression involving two variables.

Simple Linear Regression

In this regression task we will predict the percentage of marks that a student is expected to score based upon the number of hours they studied. This is a simple linear regression task as it involves just two variables.

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
df=pd.read_csv("Scores.csv")
df
```

Out[2]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25
10	7.7	85
11	5.9	62
12	4.5	41
13	3.3	42
14	1.1	17
15	8.9	95
16	2.5	30
17	1.9	24
18	6.1	67
19	7.4	69
20	2.7	30
21	4.8	54
22	3.8	35
23	6.9	76
24	7.8	86

In [3]:

```
# The shape of dataset
df.shape
```

Out[3]:

```
(25, 2)
```

In [4]:

```
# Check the info of data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Hours   25 non-null     float64
 1   Scores  25 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 528.0 bytes
```

In [5]:

```
# Check the discription of data
df.describe()
```

Out[5]:

	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000

In [6]:

```
def getgrade(mark):
    if mark >= 75:
        return "A"
    elif mark>=60 and mark<75:
        return "B"
    elif mark>=35 and mark<60:
        return "C"
    else:
        return "F"
```

In [7]:

```
df
```

Out[7]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25
10	7.7	85
11	5.9	62
12	4.5	41
13	3.3	42
14	1.1	17
15	8.9	95
16	2.5	30
17	1.9	24
18	6.1	67
19	7.4	69
20	2.7	30
21	4.8	54
22	3.8	35
23	6.9	76
24	7.8	86

In [8]:

```
df["Grade"]=df["Scores"].apply(getgrade)
```

In [9]:

```
df
```

Out[9]:

	Hours	Scores	Grade
0	2.5	21	F
1	5.1	47	C
2	3.2	27	F
3	8.5	75	A
4	3.5	30	F
5	1.5	20	F
6	9.2	88	A
7	5.5	60	B
8	8.3	81	A
9	2.7	25	F
10	7.7	85	A
11	5.9	62	B
12	4.5	41	C
13	3.3	42	C
14	1.1	17	F
15	8.9	95	A
16	2.5	30	F
17	1.9	24	F
18	6.1	67	B
19	7.4	69	B
20	2.7	30	F
21	4.8	54	C
22	3.8	35	C
23	6.9	76	A
24	7.8	86	A

In [10]:

```
df["Grade"].value_counts()
```

Out[10]:

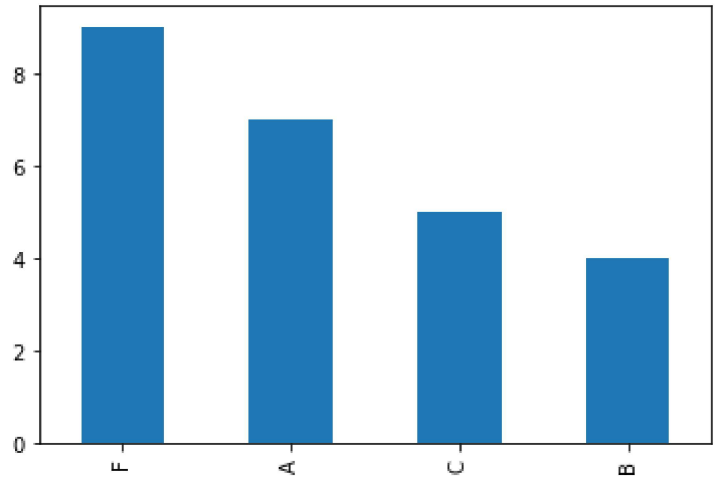
```
F    9
A    7
C    5
B    4
Name: Grade, dtype: int64
```

In [11]:

```
df["Grade"].value_counts().plot(kind="bar")
```

Out[11]:

<AxesSubplot:>

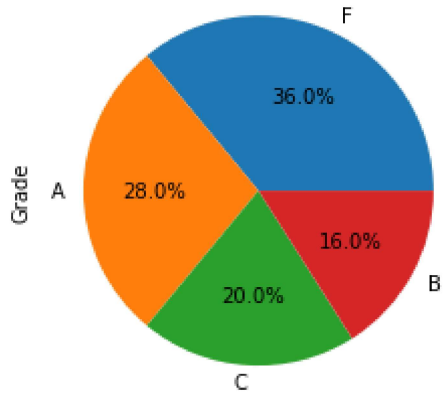


In [12]:

```
df["Grade"].value_counts().plot(kind="pie",autopct="%1.1f%%")
```

Out[12]:

<AxesSubplot:ylabel='Grade'>



In [13]:

```
df
```

Out[13]:

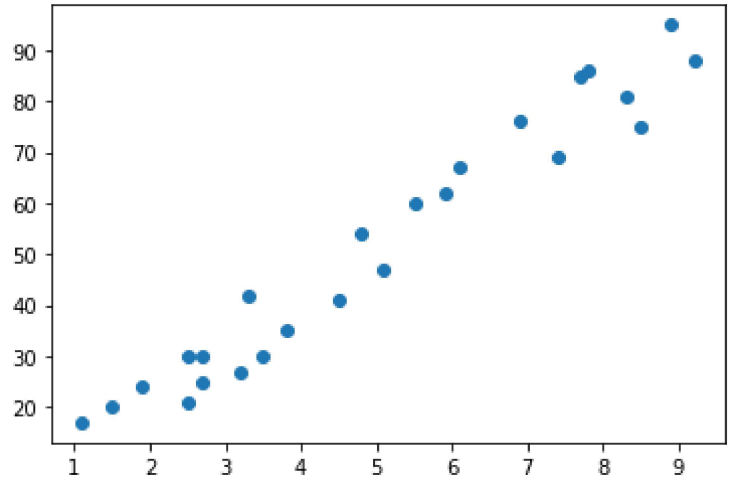
	Hours	Scores	Grade
0	2.5	21	F
1	5.1	47	C
2	3.2	27	F
3	8.5	75	A
4	3.5	30	F
5	1.5	20	F
6	9.2	88	A
7	5.5	60	B
8	8.3	81	A
9	2.7	25	F
10	7.7	85	A
11	5.9	62	B
12	4.5	41	C
13	3.3	42	C
14	1.1	17	F
15	8.9	95	A
16	2.5	30	F
17	1.9	24	F
18	6.1	67	B
19	7.4	69	B
20	2.7	30	F
21	4.8	54	C
22	3.8	35	C
23	6.9	76	A
24	7.8	86	A

In [14]:

```
plt.scatter(df["Hours"],df["Scores"])
```

Out[14]:

```
<matplotlib.collections.PathCollection at 0x1e659ab5b20>
```



From the graph above, we can clearly see that there is a positive linear relation between the number of hours studied and percentage of score.

In [15]:

```
# Check the correlation
df.corr().style.background_gradient()
```

Out[15]:

	Hours	Scores
Hours	1.000000	0.976191
Scores	0.976191	1.000000

We Split The Data Set Into X and Y

In [16]:

```
x = df.iloc[:,:-2]  #2d
y = df.iloc[:,-2]   #1d
```

In [17]:

```
x
```

Out[17]:

	Hours
0	2.5
1	5.1
2	3.2
3	8.5
4	3.5
5	1.5
6	9.2
7	5.5
8	8.3
9	2.7
10	7.7
11	5.9
12	4.5
13	3.3
14	1.1
15	8.9
16	2.5
17	1.9
18	6.1
19	7.4
20	2.7
21	4.8
22	3.8
23	6.9
24	7.8

In [18]:

```
y
```

Out[18]:

```
0     21
1     47
2     27
3     75
4     30
5     20
6     88
7     60
8     81
9     25
10    85
11    62
12    41
13    42
14    17
15    95
16    30
17    24
18    67
19    69
20    30
21    54
22    35
23    76
24    86
Name: Scores, dtype: int64
```

train_test_split

In [19]:

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=1)
```

In [20]:

```
xtrain
```

Out[20]:

	Hours
4	3.5
2	3.2
20	2.7
6	9.2
7	5.5
22	3.8
1	5.1
16	2.5
0	2.5
15	8.9
24	7.8
23	6.9
9	2.7
8	8.3
12	4.5
11	5.9
5	1.5

In [21]:

```
xtest
```

Out[21]:

	Hours
14	1.1
13	3.3
17	1.9
3	8.5
21	4.8
10	7.7
18	6.1
19	7.4

In [22]:

```
#Step1:- Import the Model
from sklearn.linear_model import LinearRegression

#Step2:- Create the Object of the Model
linreg = LinearRegression()

#Step3:- Train the Model m & c
linreg.fit(xtrain,ytrain)

#Step4:- Make Prediction
ypred = linreg.predict(xtest)
```

In [23]:

```
linreg.coef_
```

Out[23]:

```
array([10.41075981])
```

In [27]:

```
linreg.intercept_
```

Out[27]:

```
-1.5123061161277889
```

Evaluating the model

The final step is to evaluate the performance of algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For simplicity here, we have chosen the mae and mse that means mean absolute error mean square error. There are many such metrics.

In [33]:

```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [34]:

```
mae = mean_absolute_error(ytest,ypred)
mse = mean_squared_error(ytest,ypred)

rmse = np.sqrt(mse)

r2 = r2_score(ytest,ypred)

print(f"MAE :- {mae}\n MSE :- {mse}\n RMSE :- {rmse}\n ACCURACY:- {r2}")
```

```
MAE :- 7.169048271425507
 MSE :- 56.092330905646705
 RMSE :- 7.489481350911204
 ACCURACY:- 0.8933827573294114
```

In [35]:

```
newob = 5

linreg.predict([[newob]])[0]
```

Out[35]:

```
50.541492935598384
```

In [36]:

```
def makeprediction():
    newob = float(input("Enter No of Hours of Study:- "))
    yp = linreg.predict([[newob]])[0]
    print(f"If You Study of {newob} hrs, You will score around {yp:.1f} marks")
    return print(f"{yp:.2f}")
```

In [37]:

```
makeprediction()
```

```
Enter No of Hours of Study:- 9.25
If You Study of 9.25 hrs, You will score around 94.8 marks
94.79
```

Ans :- If a student study of 9.25 hrs hi will score around 94.8 marks

In [ ]:

```

```