Lec. XI. Multiple L.R

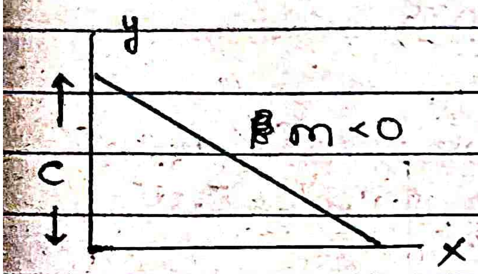
~~Numericals on~~ Recap: In simple L.R, the dependent variable (y) depends on a single independent variable (x).

$y = mx + c$ $\qquad$ m, c → regression coefficient

Slope or weight ~~that~~ specifies the ~~factor~~ By which 'x' has impact on 'y'.

intercept or bias ~~that fixes the~~ offset ~~to~~ a line.

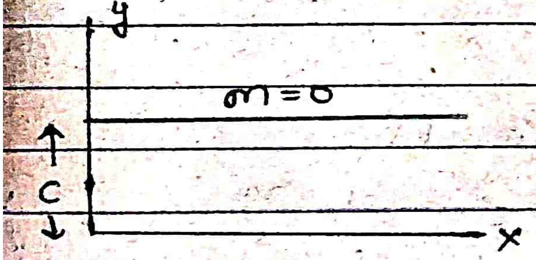
case I: $m < 0$



- variable 'x' has negative impact on 'y'.
- if 'x' increase, 'y' decrease.

case II: $m = 0$

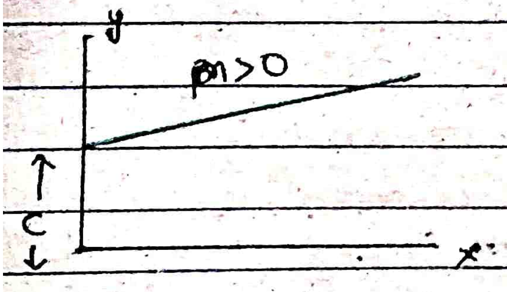
- it indicate variable 'x' has no impact on 'y'.



- if 'x' change, there will be no change in 'y'.

case III: $m > 0$

- it indicate variable 'x' has positive impact on 'y'.



- if 'x' increase, 'y' will increase & vice-versa.

In **Multiple Linear Regression** the dependen variable depends on more than one independ variables-

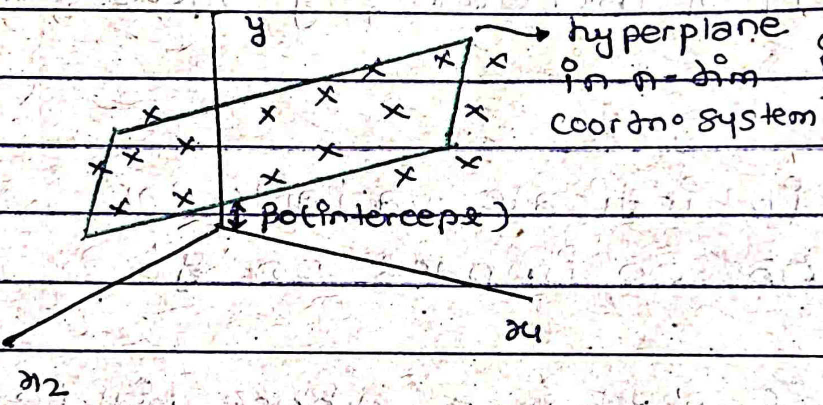
for multiple LR the form of the model is —

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

here, ○ y is a dependent variable

○ $x_1, x_2, \ldots, x_n$ are independent variab

- ○ $\beta_0, \beta_1, \ldots, \beta_n$ are regression coefficient.
- ○ β_j $(1 <= j <= n)$ is the slope or weight that specifies the factor by which x_j has an impact on y



= ie

x_1	x_2	y
cgpa	iq	placement

MLR tries to fit a regression line (or hyperplane) through a multidimensional set of data-points-

Code - Implementation :

```
from sklearn.datasets import make_regression
# The make_regression func. in scikit-learn is used to learn &
# [illegible] generate synthetic dataset.
import pandas as pd
import numpy as np

import plotly.express as px
import plotly.graph_objects as go

from sklearn.metrics import mean_absolute_error,
    mean_squared_error, r2_score.

x, y = make_regression(n_samples =100,
  n_features = 2 , n_informative = 2, n_target = 1,
   noise = 50)

# n_samples = nos of sample in dataset
# n_features = total nos of features (independent
                 variable) in dataset.
# n_informative = nos of features used to ~~generate~~ build
                 the linear model used to generate o/p
# n_target = nos of o/p variables.
# noise = The std. dev. of the gaussian noise
               applied to the o/p.
```

- here, n_feature = 2 , (n_informative =2) means both features play a role in determining the target value

- noise
  - higher → more scattered distribution of target values, around the linear relationship.
  - lower → make reln. b/w feature & target more apparent.

x → 2D-array
y → 2D-array

? → array([[x_1, y_1

x_2, y_2

- - - - -

x_n, y_n]])

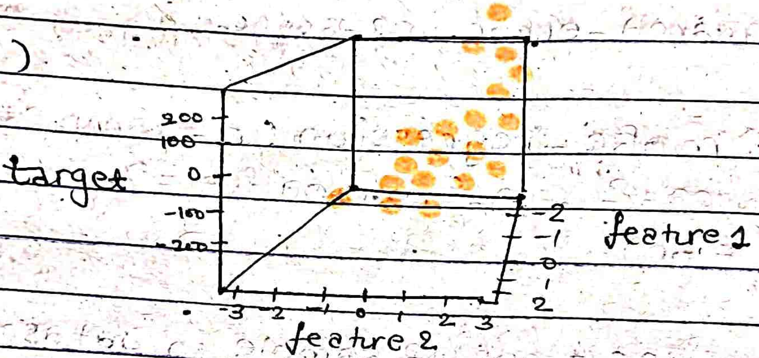
```
df = pd.Dataframe({'feature 1': x[:, 0],
            'feature 2': x[:, 1], 'target': y})

df.shape → (100, 3)

df.head()
  O/P →
```

	feature 1	feature 2	target
0	0.750203	0.505091	10.123456
1	0.156785	1.609091	84.090764
2	-1.220283	1.709091	39.219221
3	-1.023591	0.609991	-69.829168
4	0.837509	0.550550	8.12349

```
fig = px.scatter_3d(df, x='feature 1',
        y='feature 2', z='target')
fig.show()
```



```
from sklearn.model_selection import
  train_test_split

x_train, x_test, y_train, y_test =
train_test_split(x, y, test_size = 0.2, random_state=2

from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(x_train, y_train)

y_pred = model.predict(x_test)
  ↳ array([142.466  _  _  _
          -30.304  _  _  _  ])
```

```
x = np.linspace(-5, 5, 10)
y = np.linspace(-5, 5, 10)
xGrid, yGrid = np.meshgrid(y, x)
```
→(x & y)

→ o/p → 2D grid of coordinates based on the cartesian product of x & y.

```
final = np.vstack((xGrid.ravel().reshape(1,100),
        yGrid.ravel().reshape(1,100))).T
```

xGrid & yGrid are 2D arrays created by np.meshgrid based on x-array & y-array.

.ravel() is used to flatten them into 1D array

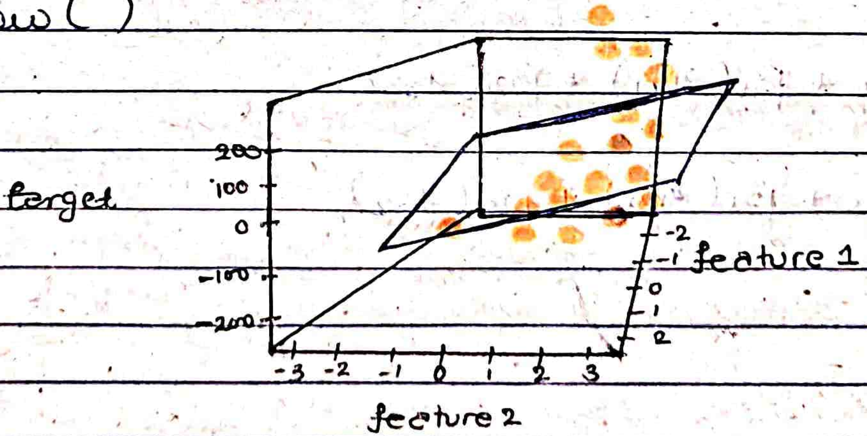
v.stack vertically stack two row vectors
(1,100) + (1,100) → (2,100)
row column

the first row contain flattened and reshaped x-coordinates & similar for y.

```
fig = px.scatter-3d(df, x = 'feature 1',
             y = 'feature 2', z = 'target')

fig.add-trace(go.Surface(x = x, y = y, z = z))

fig.show()
```


target
feature 1
feature 2