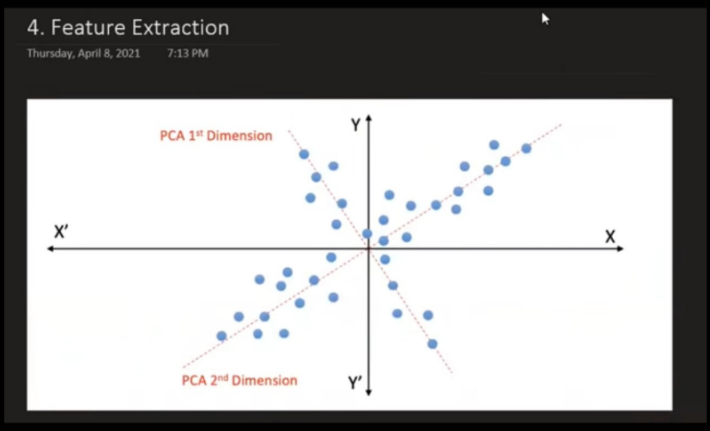
Feature Extraction



In this we build completely new feature from that

This is not feature construction

↙

Lets say No of Rooms

Feature Scaling :→

After doing all Data preprocessing task → Feature Scaling

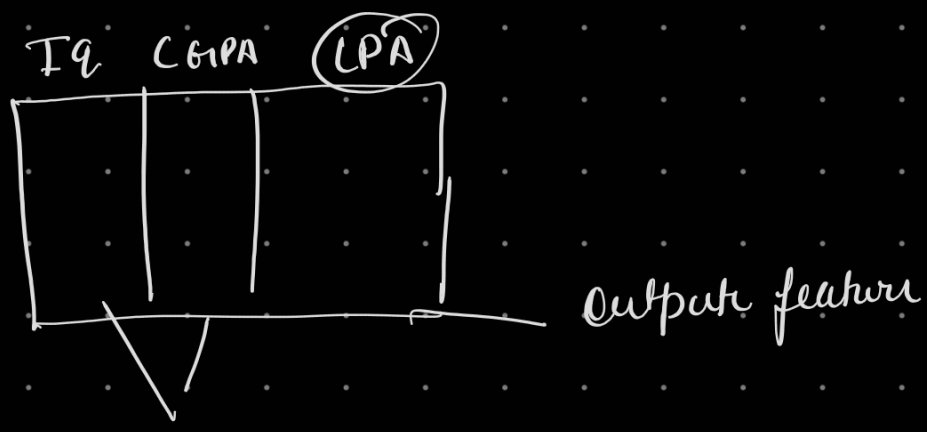
↓

Machine learning ko dena hi phle

* Easist way ↴

To Join

Feature Scaling → is a technique → to standardize the independent feature present in data in a fixed range



[iq, CGPA] unko Ek choti range mai likaru ana → Mat is a feature Scaling

Iq	Age
100	10

age	Salary	Purchase
50	83,000	1
27	48,000	0

→ KNN use karu rhe

↙ Distance find out

↘ Distance calculate karne ke liye

P_1 (x_1, y_1)
P_2 (x_2, y_2) → $\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$

$(x_2 - x_1)^2 = (83000 - 48000)^2$

$= 1\,22\,5000000$ — (1)

$$(y_2 - y_1)^2 = (50 - 27)^2 = \boxed{529} \quad -(2)$$

In this difference is 2 high As of now

in (1) (2)

1 will Dominant

knn ← on (2)

will not able to do that

Types of feature Scaling :→

Standardization

Normalization

→ min max Scaler

→ Robust Scaler

Standardization :→ (Z - Score Normalization)

Let say we have input Column hai

Age	Salary
27	
15	
33	
63	
⋮	⋮

→ Standardize karna hai

So let say

$$x_i = \frac{x_i - \bar{x}}{\sigma}$$

mean

$$x_1 = \frac{27 - (32)}{\sigma}$$

Standard daviation (10)

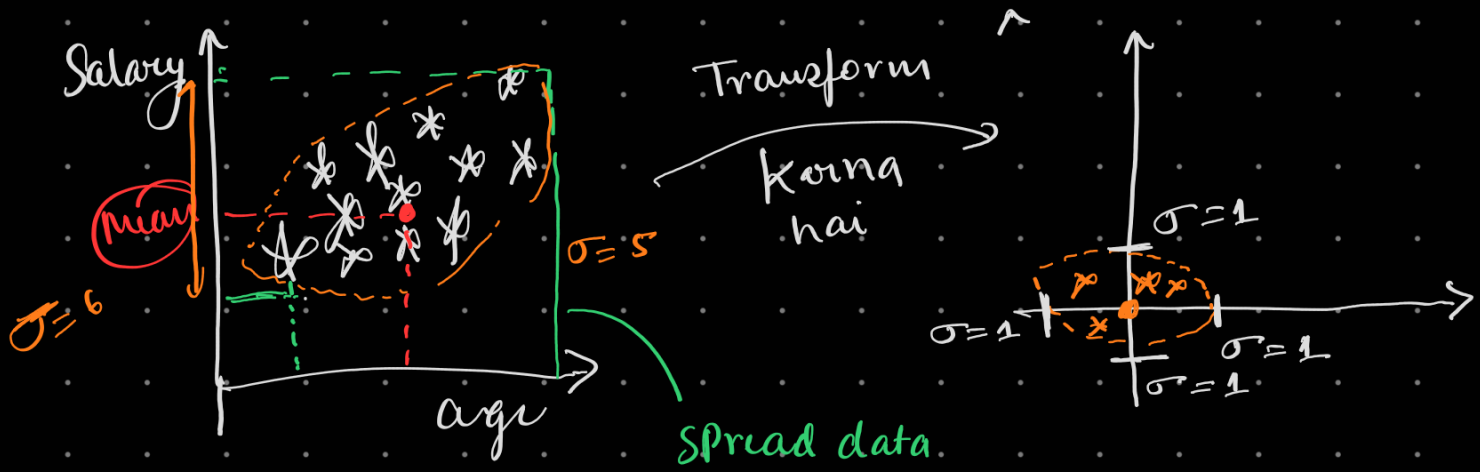
age'
2.3
-1.2
-2.1
⋮

500 values
mean = 0
$\sigma = 1$

$$x_i = \frac{27 - 32}{10}$$

Jab hum age Column ko → Standardize kar diye to → age' column mai → That is for sure uska mean = 0, Standard daviation = 1

Guomatric Show → kya hai



Spread data.

Boundary

When we do Standardization

first Mean Should become 0

It will show in graph like this

Mean of age Salary will become zero

Mean Centric

let say $\sigma = 5$ Age
$\sigma = 6$ Salary

Example & What is use of this Standardization & Impact of outlier

```
import numpy as np # linear algebra
import pandas as pd #data processing
import matplotlib.pyplot as plt
import seaborn as sns
```
[2] ✓ 1.3s

```
df = pd.read_csv('Social_Network_Ads.csv')
```
[3] ✓ 0.0s

```
df=df.iloc[:,2:]
df.sample(5)
```
[4] ✓ 0.0s

	Age	EstimatedSalary	Purchased
164	33	69000	0
276	38	71000	0
196	30	79000	0
384	57	33000	1
165	18	86000	0

Train test not needed but it is good to have this.

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(df.drop('Purchased', axis=1),
                                                    df['Purchased'],test_size=0.3, random_state=0)
X_train.shape, X_test.shape
```
[5] ✓ 0.0s

```
((280, 2), (120, 2))
```

Standard Scaler :

Standar Scalar formula

$$x_i = \frac{x_i - \bar{x}}{\sigma}$$

Scikit learn lib

Train test not needed but it is good to have this.

X_train, X_test, Y_train, Y_test = train_test_split(df.drop('Purchased', axis=1), df['Purchased'], test_size=0.3, random_state=0): This line performs the data split. Let's break it down:

`df.drop('Purchased', axis=1):` This part selects the features (independent variables) for your machine learning model. It assumes that your DataFrame df contains a column named 'Purchased,' and it drops this column using the drop method with axis=1. The result is a DataFrame with the feature columns.

`df['Purchased']:` This part selects the target variable (the dependent variable) that you want to predict. It assumes that the target variable is stored in the 'Purchased' column of your DataFrame.

`test_size=0.3:` This parameter specifies that 30% of the data will be used for testing, and the remaining 70% will be used for training. You can adjust this parameter to change the ratio of the training and testing data.

`random_state=0:` This parameter sets the random seed for reproducibility. Using the same random_state value will ensure that the data split is the same every time you run the code. You can change this value or omit it if you don't need reproducibility.

`X_train, X_test, Y_train, Y_test:` These variables receive the split data. X_train will contain the training feature data, X_test will contain the testing feature data, Y_train will contain the training target data, and Y_test will contain the testing target data.

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(df.drop('Purchased', axis=1),
                                                    df['Purchased'],test_size=0.3, random_state=0)
X_train.shape, X_test.shape
```

✓ 0.0s Python

```
((280, 2), (120, 2))
```

Standard Scaler :

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

#Fit the Scaler to the train set, it will learn the parameters
scaler.fit(X_train)

#Transfrom train adn test sets.
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[24] ✓ 0.0s Python

The code is for standardizing (scaling) your features using the StandardScaler from scikit-learn. Standardization is a common preprocessing step in machine learning to make sure that different features have the same scale and distribution. Here's what each part of the code does:

`**from sklearn.preprocessing import StandardScaler:**` This line imports the StandardScaler class from scikit-learn's preprocessing module.

`scaler = StandardScaler():` This line creates an instance of the StandardScaler class. This instance will be used to standardize the data.

`scaler.fit(X_train):` This line fits (learns) the scaler on the training data. By fitting the scaler to the training data, it learns the mean and standard deviation of each feature in the training set. This information is necessary for standardization.

`scaler.transform(X_train):` This line transforms (standardizes) the training data. It subtracts the mean and divides by the standard deviation for each feature in X_train, making the training data have a mean of 0 and a standard deviation of 1 for each feature. The result is stored in X_train_scaled.

`scaler.transform(X_test):` This line transforms (standardizes) the testing data in the same way as the training data. It uses the mean and standard deviation learned from the training data to standardize the testing data. The standardized testing data is stored in X_test_scaled.

```
scaler.mean_
```
[6] ✓ 0.0s

```
array([3.78642857e+01, 6.98071429e+04])
```

```
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled,columns=X_test.columns)
```
[8] ✓ 0.0s

```
np.round(X_train.describe(),1)
```
[9] ✓ 0.0s

	Age	EstimatedSalary
count	280.0	280.0
mean	37.9	69807.1
std	10.2	34641.2
min	18.0	15000.0
25%	30.0	43000.0
50%	37.0	70500.0
75%	46.0	88000.0
max	60.0	150000.0

Mean will become zero after scaling and standard deviation beome 1

```
np.round(X_train_scaled.describe(),1)
```
[10] ✓ 0.0s

	Age	EstimatedSalary
count	280.0	280.0
mean	0.0	0.0
std	1.0	1.0
min	-1.9	-1.6
25%	-0.8	-0.8
50%	-0.1	0.0
75%	0.8	0.5
max	2.2	2.3

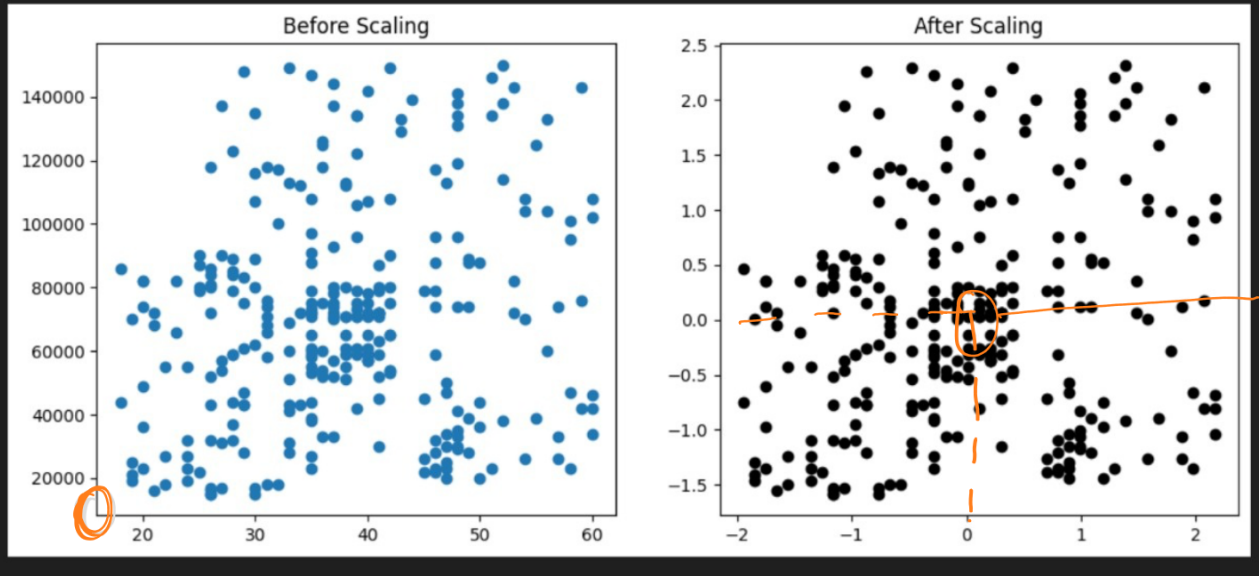
```
X_train_scaled
```
[16] ✓ 0.0s

	Age	EstimatedSalary
0	-1.163172	-1.584970
1	2.170181	0.930987
2	0.013305	1.220177
3	0.209385	1.075582
4	0.405465	-0.486047
...	...	...
275	0.993704	-1.151185
276	-0.869053	-0.775237
277	-0.182774	-0.514966
278	-1.065133	-0.457127
279	-1.163172	1.393691

280 rows × 2 columns

```
fig, (ax1,ax2) = plt.subplots(ncols=2, figsize=(12,5))

ax1.scatter(X_train['Age'], X_train['EstimatedSalary'])
ax1.set_title('Before Scaling')
ax2.scatter(X_train_scaled['Age'], X_train_scaled['EstimatedSalary'], color='black')
ax2.set_title("After Scaling")
plt.show()
```
[19] ✓ 0.5s Python



Origion kyha nai

`plt.subplots(ncols=2, figsize=(12,5))`: This line of code creates a figure with two subplots arranged horizontally (side by side). The `ncols=2` parameter specifies that you want two columns of subplots, and the `figsize=(12,5)` parameter sets the width and height of the figure to be 12 units wide and 5 units tall, respectively.

`(ax1, ax2)`: This part of the code unpacks the two subplot axes that are created as a result of the `plt.subplots` call. ax1 and ax2 are variables that you can use to manipulate the individual subplots.

now data is mean center

Scaling Benifit fund → By this we can get By PDF
↓
Probablity density function

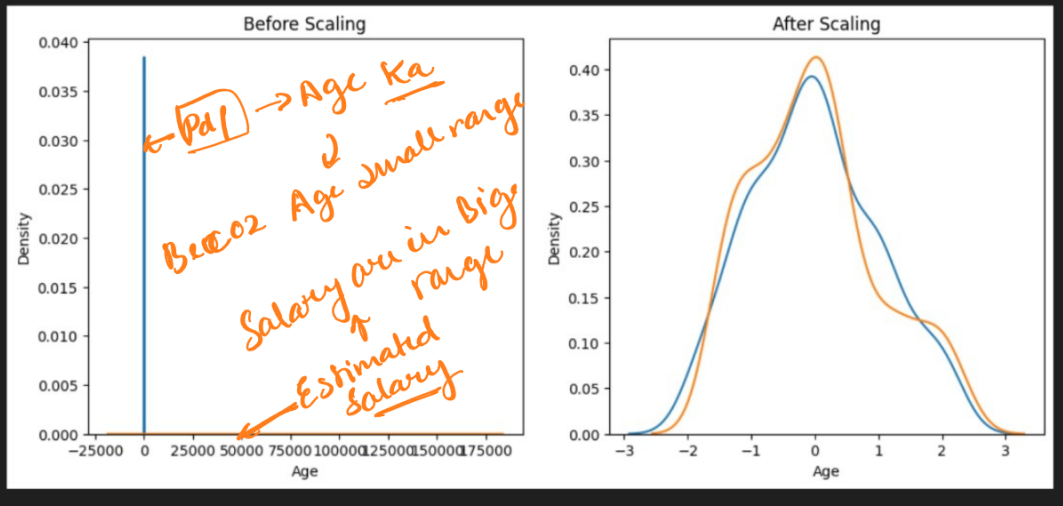
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,5))

#before scaling.
ax1.set_title('Before Scaling')
sns.kdeplot(X_train['Age'], ax=ax1)
sns.kdeplot(X_train['EstimatedSalary'],ax=ax1)

#After Scaling.
ax2.set_title('After Scaling')
sns.kdeplot(X_train_scaled['Age'], ax=ax2)
sns.kdeplot(X_train_scaled['EstimatedSalary'], ax=ax2)
```

[38] ✓ 0.6s Python

```
<Axes: title={'center': 'After Scaling'}, xlabel='Age', ylabel='Density'>
```



> ax2.set_title('After Scaling'): Sets the title of the second subplot to "After Scaling".

> sns.kdeplot(X_train_scaled['Age'], ax=ax2): Plots the KDE of the 'Age' feature after scaling on the second subplot (ax2). This subplot shows the distribution of 'Age' after applying feature scaling.

> sns.kdeplot(X_train_scaled['EstimatedSalary'], ax=ax2): Plots the KDE of the 'EstimatedSalary' feature after scaling on the second subplot (ax2). This subplot shows the distribution of 'EstimatedSalary' after applying feature scaling.

SAME FOR BEFORE SCALING

× parameters + Tag markdown

So in Befor Scaling → Age
or
Expected Salary
→ have large Difference
↓

On Second → Both change are in same Range
which increase Performance

Comparison of Distribution :->

ComParison of Distributions

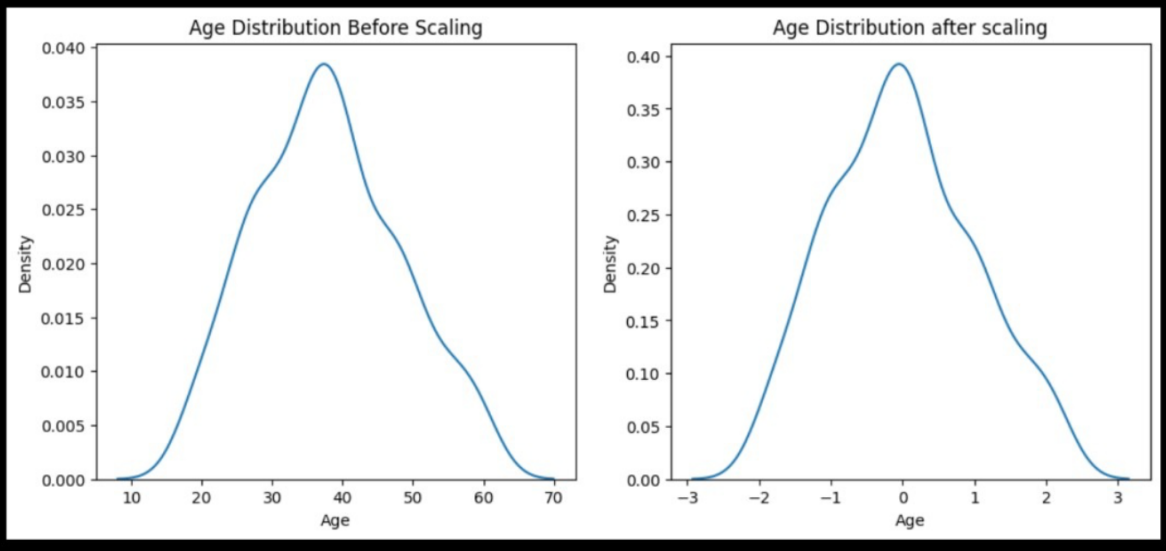
```
fig, (ax1, ax2) = plt.subplots(ncols=2,figsize=(12,5))

#before scaling.
ax1.set_title('Age Distribution Before Scaling')
sns.kdeplot(X_train['Age'], ax=ax1)

#After Scaling.
ax2.set_title('Age Distribution after scaling')
sns.kdeplot(X_train_scaled['Age'], ax=ax2)
```

[40] Python

```
<Axes: title={'center': 'Age Distribution after scaling'}, xlabel='Age', ylabel='Density'>
```



Distribution is Same Even After Scaling

Distribution Retain Always there isn't Any change only Scaling will change.

where mean = 0

σ (Standard deviation = 1)

↳ Outlier Ka impact reduce → Kuch nahi hota after Scaling

Outlier ko handle karna padega Explicitly It behave the way it is

↳ Standardization → will not Impact any thing

But when we need to apply →

Algorithm	Reason of applying feature Scaling
K-Means	→ use the Euclidean distance measure
KNN	→ Measure the distance between Pairs of Sample → and these distances are influenced By measurement units

↳ ye dono → we Calculate Eucladian distance

We are Calculating Distance so Variation Jada hoga to

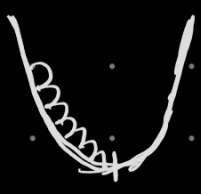
PCA → Principal Component Analysis

↳ feature Extraction Technique

↙

Try to get feature → with Maximum Variance

Artificial Neural Network :→ Apply Gradient Descent

Gradient Descent →  → we have this function and we want

All Deep learning Depend on gradiant Descent

Minimum Value Extraction

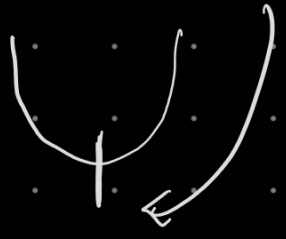
Gradiant Descent → Mai weight Assign Karte hai like

w_1 w_2

If Variable Variation Jada hota hai

So Ek ka weight kam hota Ek ka Jada hojata

Some cannt Converse



Some Algo dont need Standardization

A > B

Pucha hai

Comparision Karte hai

So we dont need

Decision Tree

- Random forest
- XG Boost
- Gradiant Boost