

✓ Feature Scaling in Machine Learning

Understanding Feature Scaling

Definition

- **Feature Scaling** is a technique to bring all independent features in a dataset to a common scale without distorting differences in the ranges of values.
- **Purpose:**
 - Ensure that no single feature dominates others due to its scale.
 - Improve the performance of certain machine learning algorithms that are sensitive to the scale of data.

Analogy

- **Example:**
 - If you have features like **IQ** (range: 0-200) and **CGPA** (range: 0-10), their scales are different.
 - Scaling brings them to a comparable range, facilitating better model performance.
-

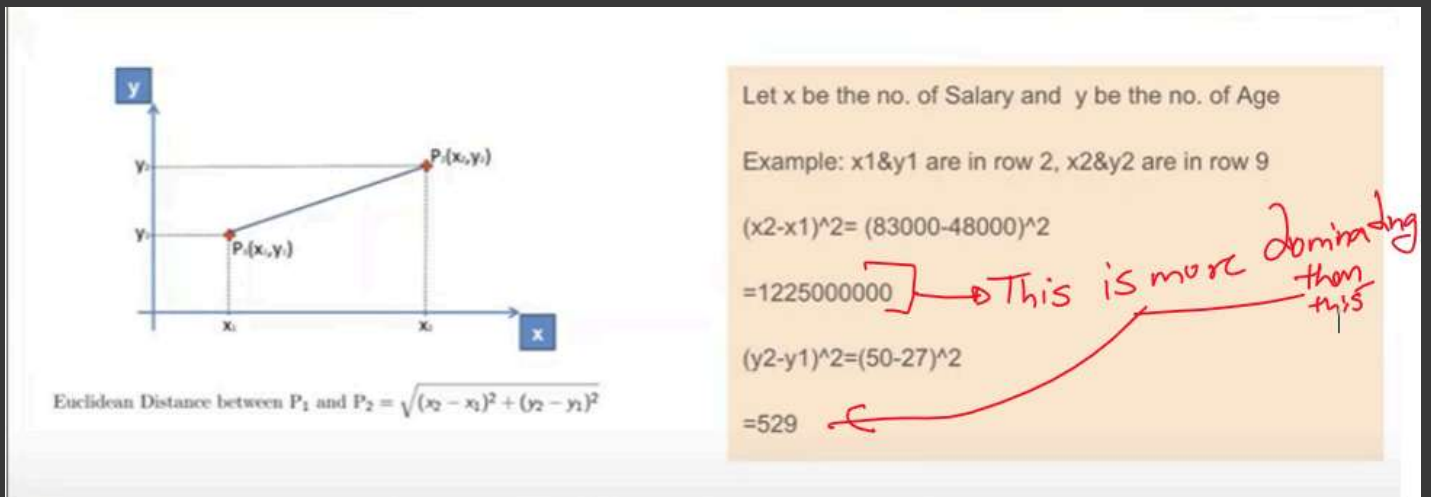
✓ Why Do We Need Feature Scaling?

- **Problem:**
 - Features with different scales can bias machine learning algorithms, especially those relying on distance calculations (e.g., K-Nearest Neighbors).
- **Impact on Algorithms:**
 - Algorithms may perform poorly if features are not scaled properly.
 - Some models assume that all features are centered around zero and have approximately the same variance.

Example

- **Dataset:**
 - Features: **Age** (range: 0-100), **Salary** (range: thousands).
- **Issue:**

- Salary values dominate due to larger numbers.
- Age becomes less significant in distance-based calculations.
- **Solution:**
 - Apply feature scaling to bring both features to a similar scale.



✓ Types of Feature Scaling Techniques

1. Standardization (Z-score Normalization)

- Scales data to have a mean of 0 and a standard deviation of 1.

2. Normalization (Min-Max Scaling)

- Scales data to a fixed range, usually [0, 1].
- **Robust Scaling:** Uses median and quartiles, less sensitive to outliers.
- **Log Transformation:** Useful for skewed distributions.

Standardization

Definition

- **Standardization** transforms data to have a mean (μ) of zero and a standard deviation (σ) of one.
- **Formula:**

$$Z = \frac{X - \mu}{\sigma}$$

- **X:** Original value.

- μ : Mean of the feature values.
- σ : Standard deviation of the feature values.

✓ Steps

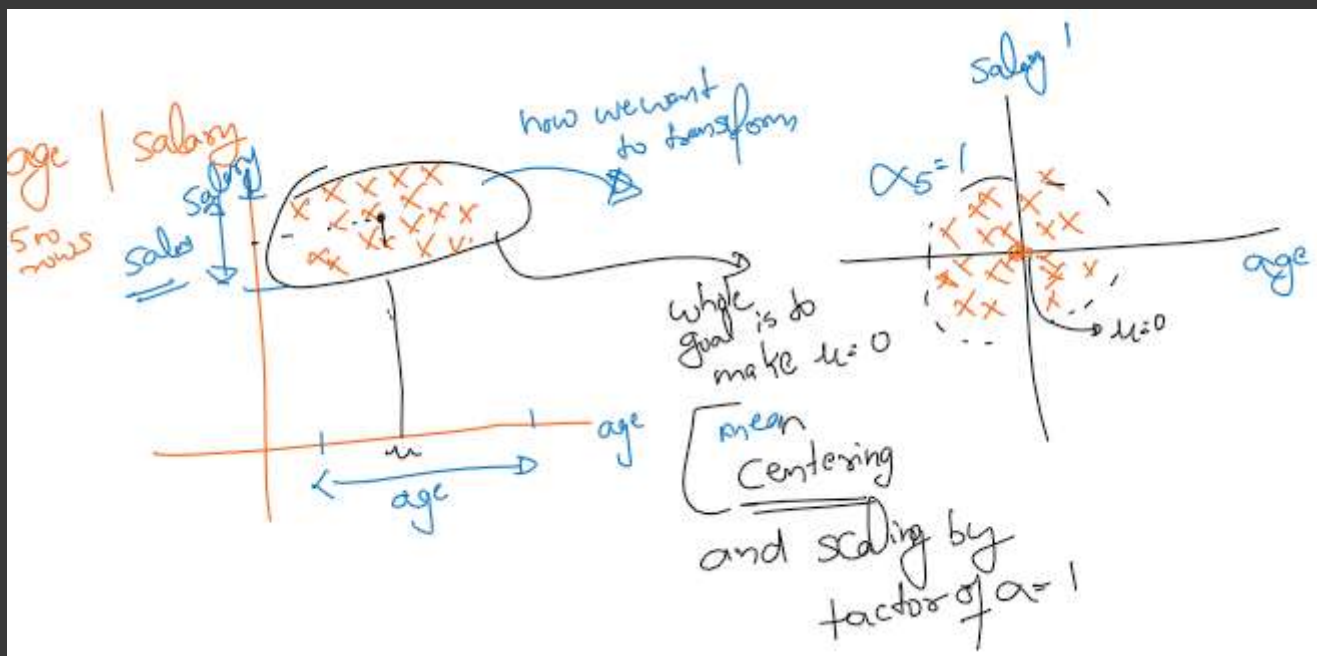
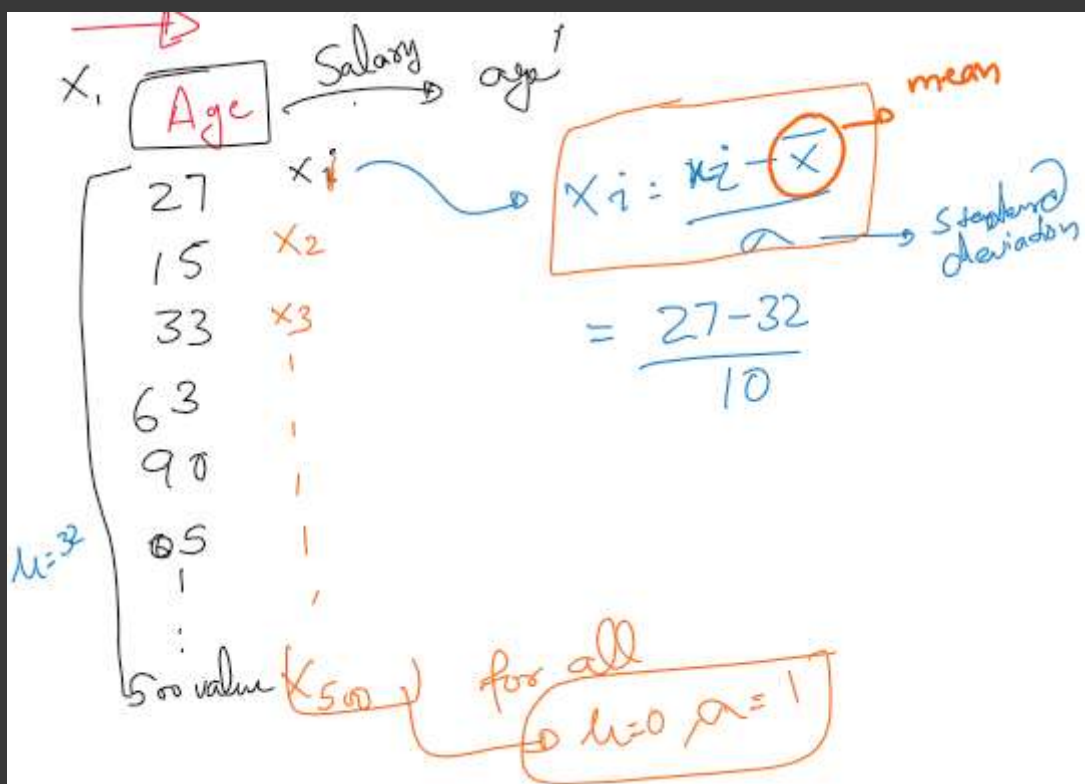
1. **Compute the Mean (μ)** of the feature.
2. **Compute the Standard Deviation (σ)** of the feature.
3. **Subtract the Mean** from each feature value.
4. **Divide** the result by the Standard Deviation.

Properties

- The transformed feature will have:
 - **Mean (μ): 0**
 - **Standard Deviation (σ): 1**

Geometric Intuition

- **Mean Centering:**
 - Shifts the data so that the mean aligns with zero.
- **Scaling Variance:**
 - Adjusts the spread of the data to have a standard deviation of one.
- **Visualization:**
 - Data points are re-scaled but the shape of the distribution remains the same.



✓ Practical Example

Dataset

- **Social Network Ads Dataset:**
 - Features: **Age, Estimated Salary.**

- Target: **Purchased** (Yes/No).

Implementation Steps

1. Import Libraries:

- `pandas`, `numpy`, `matplotlib`, `seaborn`, `sklearn`.

2. Load Dataset:

- Read data into a DataFrame.

3. Data Preprocessing:

- Remove unnecessary columns.

4. Split Dataset:

- Use `train_test_split` to divide data into training and test sets.

5. Apply Standardization:

- Use `StandardScaler` from `sklearn.preprocessing`.
- Fit the scaler on training data.
- Transform both training and test data.

6. Visualize Results:

- Plot histograms or density plots to compare distributions before and after scaling.
- Observe that mean is zero and standard deviation is one after scaling.

Code Snippet

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Observations

• Before Scaling:

- Features have different scales and units.

• After Scaling:

- Features are centered around zero with a unit standard deviation.
- The distribution shape remains the same.

```

1
2 import os
3 import zipfile
4
5 # Ensure Kaggle directory exists
6 os.makedirs(os.path.expanduser("~/kaggle"), exist_ok=True)
7 # Move the Kaggle JSON file to the Kaggle directory
8 !cp kaggle.json ~/.kaggle/
9 !chmod 600 ~/.kaggle/kaggle.json # Secure the Kaggle API token file
10
11 print("Kaggle API configured successfully.")
12
13 # Step 2: Define a function to download and extract datasets from Kaggle
14 def download_kaggle_dataset(dataset_name, download_path='/content'):
15     """
16     Downloads a dataset from Kaggle and extracts it to the specified directory.
17
18     Args:
19     - dataset_name (str): The Kaggle dataset identifier (e.g., 'markmedhat/titanic')
20     - download_path (str): The directory where the dataset should be extracted.
21
22     """
23     # Download the dataset

```



```

cp: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
Kaggle API configured successfully.
Downloading d4rklucif3r/social-network-ads dataset...
Dataset URL: https://www.kaggle.com/datasets/d4rklucif3r/social-network-ads
License(s): Community Data License Agreement - Permissive - Version 1.0
Downloading social-network-ads.zip to /content
  0% 0.00/1.46k [00:00<?, ?B/s]
100% 1.46k/1.46k [00:00<00:00, 2.93MB/s]
Dataset d4rklucif3r/social-network-ads extracted successfully.
Data source import complete.
/content/Social_Network_Ads.csv
/content/.config/default_configs.db
/content/.config/.last_survey_prompt.yaml
/content/.config/.last_update_check.json
/content/.config/.last_opt_in_prompt.yaml
/content/.config/gce
/content/.config/config_sentinel
/content/.config/hidden_gcloud_config_universe_descriptor_data_cache_configs.db
/content/.config/active_config
/content/.config/logs/2024.11.07/20.55.26.763095.log
/content/.config/logs/2024.11.07/20.56.26.532854.log
/content/.config/logs/2024.11.07/20.56.27.350325.log
/content/.config/logs/2024.11.07/20.56.09.372862.log
/content/.config/logs/2024.11.07/20.55.50.913514.log
/content/.config/logs/2024.11.07/20.56.08.122576.log
/content/.config/configurations/config_default
/content/sample_data/anscombe.json
/content/sample_data/README.md
/content/sample_data/mnist_train_small.csv

```

```
/content/sample_data/california_housing_test.csv
/content/sample_data/mnist_test.csv
/content/sample_data/california_housing_train.csv
```

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
```

```
1 df = pd.read_csv("Social_Network_Ads.csv")
```



```
1 df.shape
```

➞ (400, 3)

```
1 df.sample(5)
```

➞

	Age	EstimatedSalary	Purchased
130	31	58000	0
287	48	138000	1
28	29	43000	0
301	48	74000	1
67	23	82000	0

✓ Train Test split

(Recommended to do before the train test split)

```
1 from sklearn.model_selection import train_test_split
2
3 X_train, X_test, y_train, y_test = train_test_split(df.drop('Purchased', axis=1),
4                                                    df['Purchased'],
5                                                    test_size=0.3,
6                                                    random_state=0)
7 X_train.shape, X_test.shape
```

➞ ((280, 2), (120, 2))

✓ Standard Scalar

```

1 from sklearn.preprocessing import StandardScaler
2 scaler = StandardScaler()
3 # fit the scaler to the train set, it will learn the parameter like mean and std de
4 scaler.fit(X_train)
5
6 # Perform standardization by centering and scaling on the traind and test sets
7 # basically applying the formula
8 # and we only learn(fit) from the training data and transform on the both data
9 X_train_scaled = scaler.transform(X_train)
10 X_test_scaled = scaler.transform(X_test)

```

```
1 scaler.mean_
```

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

```

1 print(type(X_train))
2 print(type(X_train_scaled))

```

```

→ <class 'pandas.core.frame.DataFrame'>
   <class 'numpy.ndarray'>

```

```

1 # since both the types are different so we have to modify the X_train_scaled to
2 # pandas dataframe
3
4 X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
5 X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)

```

```
1 X_train_scaled.head(5)
```

```

→
      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

```

Next steps: [View recommended plots](#)

[New interactive sheet](#)

```
1 np.round(X_train.describe(), 1)
```




	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



```
1 np.round(X_train_scaled.describe(), 1)
```

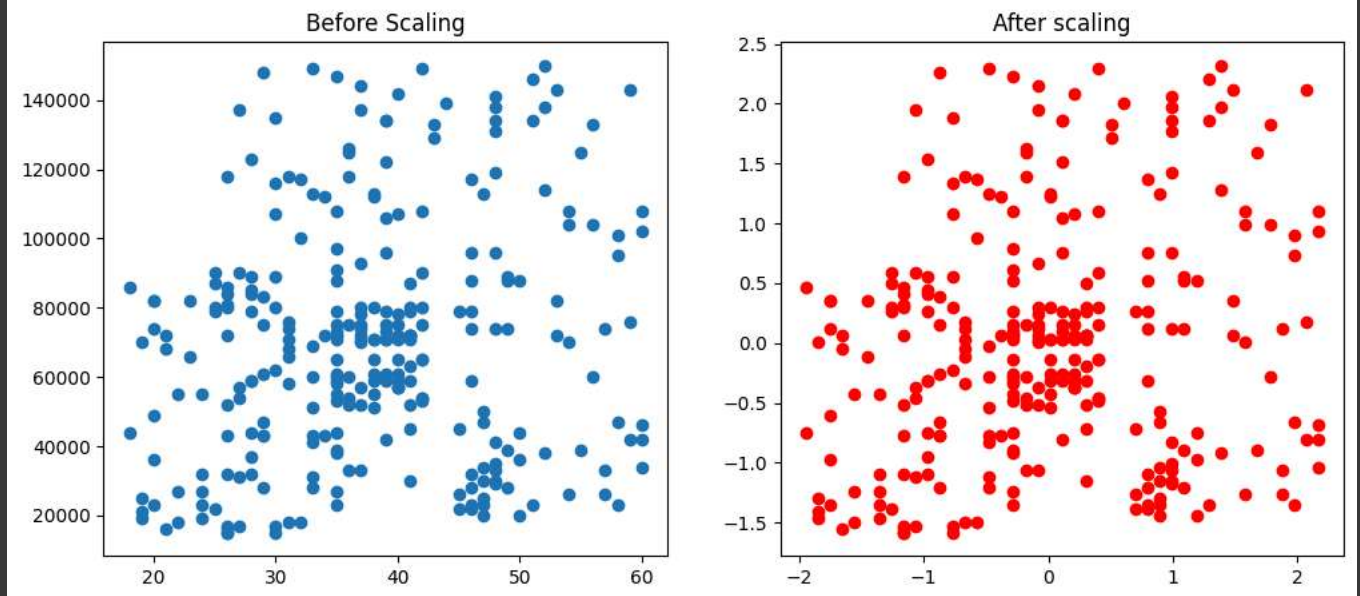


	Age	EstimatedSalary
count	280.0	280.0
mean	0.0	0.0
std	1.0	1.0
min	-1.9	-1.6
25%	-0.8	-0.8
50%	-0.1	0.0
75%	0.8	0.5
max	2.2	2.3

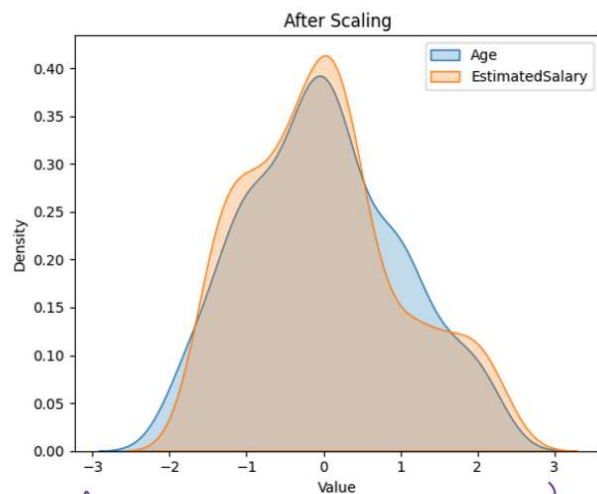
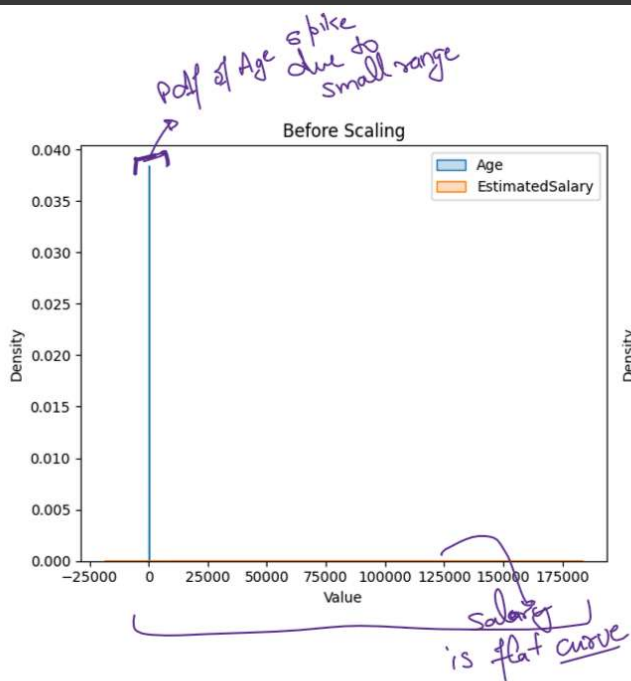
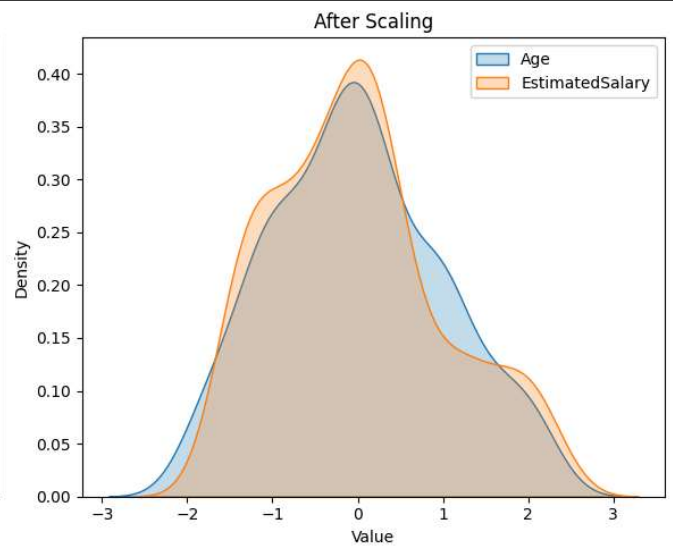
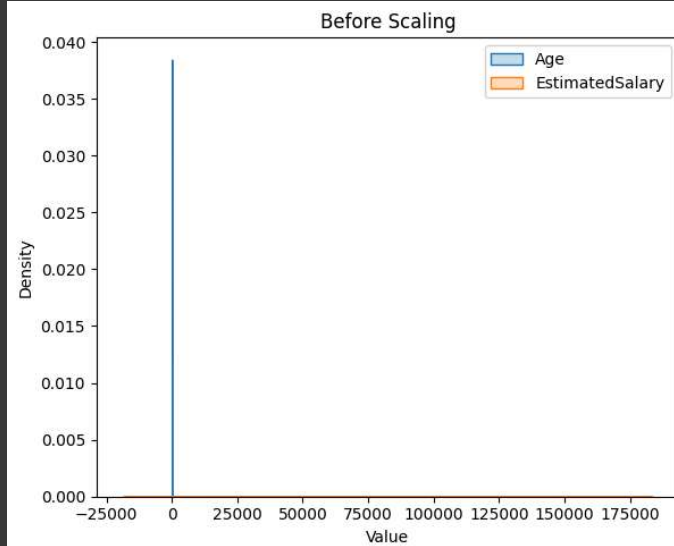


✓ Effect of the **Scaling**

```
1 fig, (ax1, ax2) = plt.subplots(ncols = 2, figsize=(12, 5))
2
3 ax1.scatter(X_train['Age'], X_train['EstimatedSalary'])
4 ax1.set_title("Before Scaling")
5
6 ax2.scatter(X_train_scaled['Age'], X_train_scaled['EstimatedSalary'], color = "red")
7 ax2.set_title('After scaling')
8 plt.show()
```



```
1
2 fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))
3
4 sns.kdeplot(X_train['Age'], ax=ax1, label='Age', fill=True)
5 sns.kdeplot(X_train['EstimatedSalary'], ax=ax1, label='EstimatedSalary', fill=True)
6 ax1.set_title('Before Scaling')
7 ax1.set_xlabel('Value')
8 ax1.set_ylabel('Density')
9 ax1.legend()
10
11 sns.kdeplot(X_train_scaled['Age'], ax=ax2, label='Age', fill=True)
12 sns.kdeplot(X_train_scaled['EstimatedSalary'], ax=ax2, label='EstimatedSalary', fill=True)
13 ax2.set_title('After Scaling')
14 ax2.set_xlabel('Value')
15 ax2.set_ylabel('Density')
16 ax2.legend()
17
18 plt.tight_layout()
19 plt.show()
```

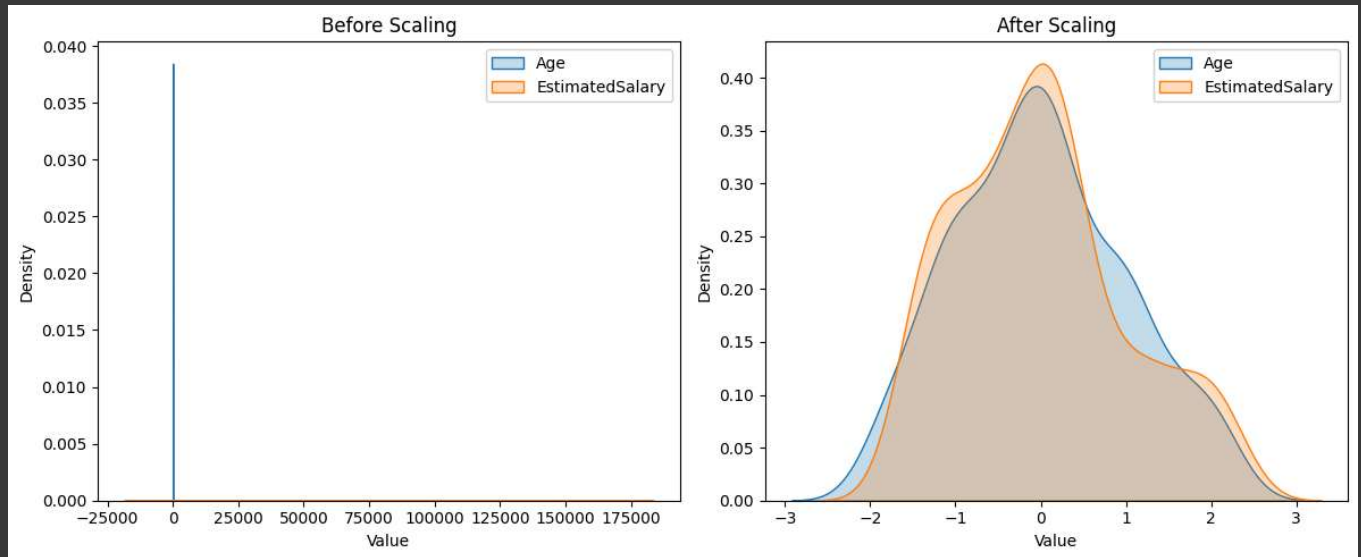


Since both are comparable then model will perform better

```

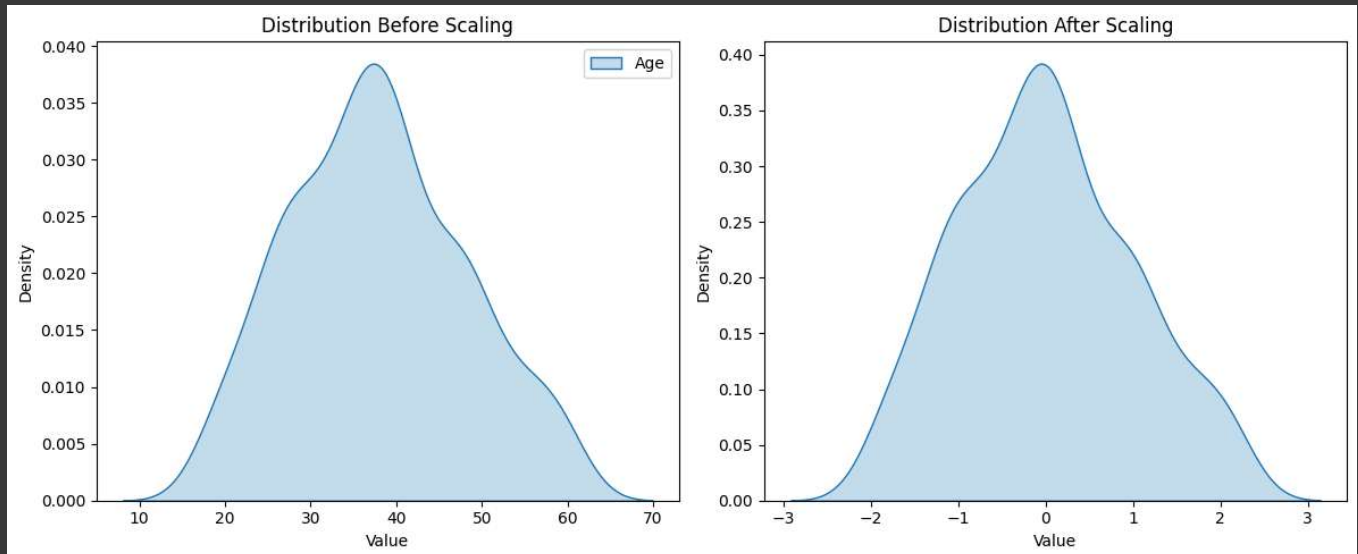
1
2 fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))
3
4 sns.kdeplot(X_train['Age'], ax=ax1, label='Age', fill=True)
5 sns.kdeplot(X_train['EstimatedSalary'], ax=ax1, label='EstimatedSalary', fill=True)
6 ax1.set_title('Before Scaling')
7 ax1.set_xlabel('Value')
8 ax1.set_ylabel('Density')
9 ax1.legend()
10
11 sns.kdeplot(X_train_scaled['Age'], ax=ax2, label='Age', fill=True)
12 sns.kdeplot(X_train_scaled['EstimatedSalary'], ax=ax2, label='EstimatedSalary', fill=True)
13 ax2.set_title('After Scaling')
14 ax2.set_xlabel('Value')
15 ax2.set_ylabel('Density')
16 ax2.legend()
17
18 plt.tight_layout()
19 plt.show()

```



✓ Distribution will not change even after scaling

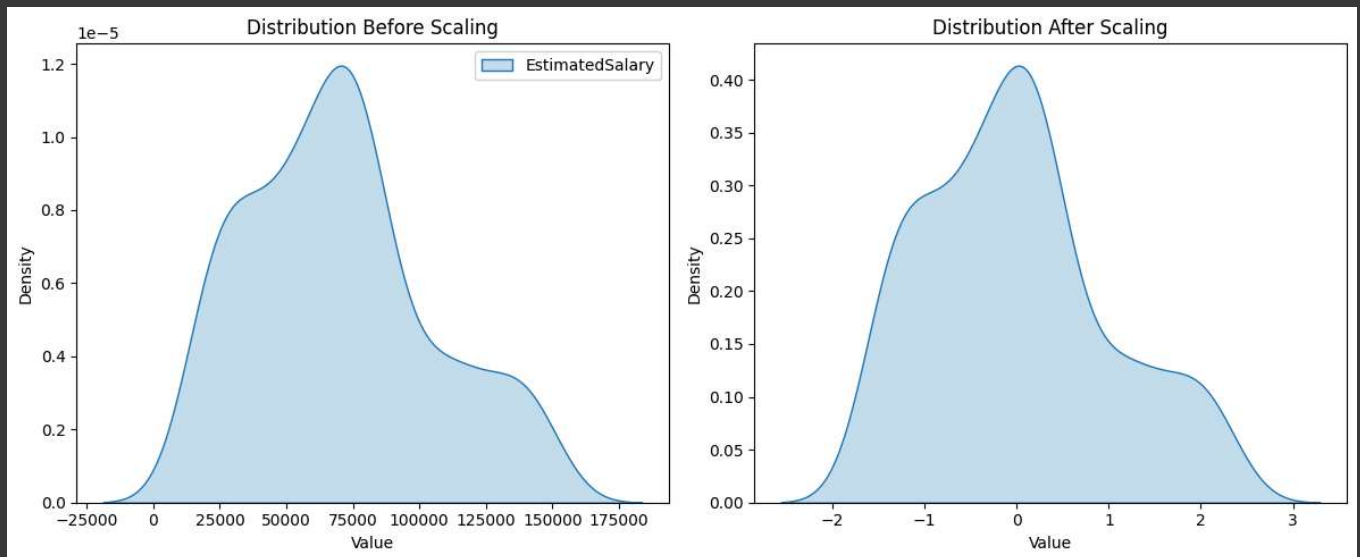
```
1
2 fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))
3
4 sns.kdeplot(X_train['Age'], ax=ax1, label='Age', fill=True)
5
6 ax1.set_title('Distribution Before Scaling')
7 ax1.set_xlabel('Value')
8 ax1.set_ylabel('Density')
9 ax1.legend()
10
11
12 sns.kdeplot(X_train_scaled['Age'], ax=ax2, label='Age', fill=True)
13 ax2.set_title('Distribution After Scaling')
14 ax2.set_xlabel('Value')
15 ax2.set_ylabel('Density')
16
17 plt.tight_layout()
18 plt.show()
```



```

1
2 fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))
3
4 sns.kdeplot(X_train['EstimatedSalary'], ax=ax1, label='EstimatedSalary', fill=True)
5
6 ax1.set_title('Distribution Before Scaling')
7 ax1.set_xlabel('Value')
8 ax1.set_ylabel('Density')
9 ax1.legend()
10
11
12 sns.kdeplot(X_train_scaled['EstimatedSalary'], ax=ax2, label='EstimatedSalary', fill=True)
13 ax2.set_title('Distribution After Scaling')
14 ax2.set_xlabel('Value')
15 ax2.set_ylabel('Density')
16
17 plt.tight_layout()
18 plt.show()

```



✓ Why scaling is important?

```
1 from sklearn.linear_model import LogisticRegression
```

```
1 lr = LogisticRegression()  
2 lr_scaled = LogisticRegression()
```

```
1 lr.fit(X_train, y_train)  
2 lr_scaled.fit(X_train_scaled, y_train)
```



▼ LogisticRegression ⓘ ?
LogisticRegression()

```
1 y_pred = lr.predict(X_test)  
2 y_pred_scaled = lr_scaled.predict(X_test_scaled)
```

```
1 from sklearn.metrics import accuracy_score
```

```
1 print("Actual", accuracy_score(y_test, y_pred))  
2 print("Scaled", accuracy_score(y_test, y_pred_scaled))
```



Actual 0.875
Scaled 0.8666666666666667

```
1 from sklearn.tree import DecisionTreeClassifier
```

```
1 dt = DecisionTreeClassifier()  
2 dt_scaled = DecisionTreeClassifier()
```

```
1 dt.fit(X_train, y_train)  
2 dt_scaled.fit(X_train_scaled, y_train)
```



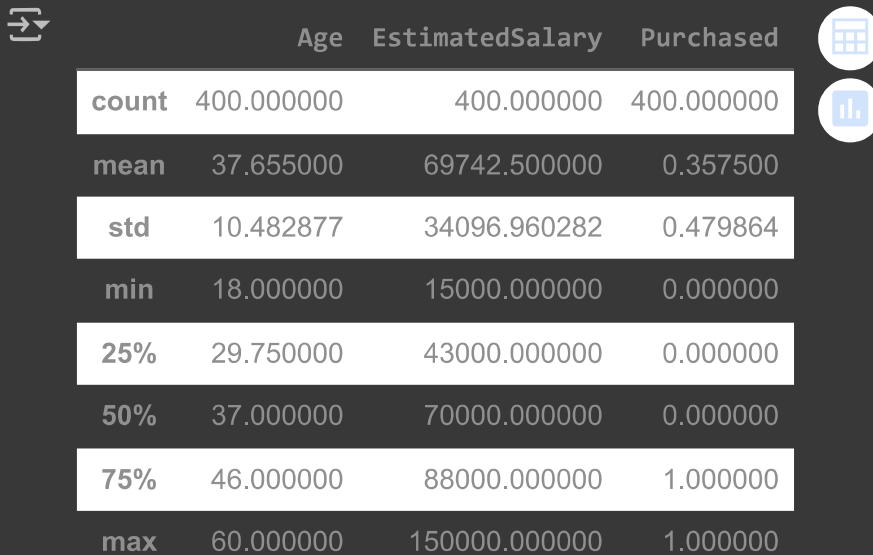
▼ DecisionTreeClassifier ⓘ ?
DecisionTreeClassifier()

```
1 y_pred = dt.predict(X_test)  
2 y_pred_scaled = dt_scaled.predict(X_test_scaled)
```

```
1 print("Actual", accuracy_score(y_test, y_pred))
2 print("Scaled", accuracy_score(y_test, y_pred_scaled))
```

Actual 0.875
Scaled 0.875

```
1 df.describe()
```



The table displays the statistical summary of a dataset with three columns: Age, EstimatedSalary, and Purchased. The first column lists the statistical measures, and the subsequent columns show the corresponding values for each feature. To the right of the table are two circular icons: a grid icon for copying the table and a bar chart icon for visualizing the data.

	Age	EstimatedSalary	Purchased
count	400.000000	400.000000	400.000000
mean	37.655000	69742.500000	0.357500
std	10.482877	34096.960282	0.479864
min	18.000000	15000.000000	0.000000
25%	29.750000	43000.000000	0.000000
50%	37.000000	70000.000000	0.000000
75%	46.000000	88000.000000	1.000000
max	60.000000	150000.000000	1.000000

Effect on Machine Learning Models

Experiment

- **Model Used:** Logistic Regression.
- **Objective:** Compare model performance before and after scaling.
- **Results:**
 - **Without Scaling:**
 - Lower accuracy (e.g., 65%).
 - **With Scaling:**
 - Higher accuracy (e.g., 86%).
- **Conclusion:**
 - Scaling can significantly improve the performance of certain algorithms.

✓ Effect Of Outlier


```
1 df.columns
```

```
➞ Index(['Age', 'EstimatedSalary', 'Purchased'], dtype='object')
```

```
1 df = pd.concat([df, pd.DataFrame({
2     'Age': [5, 90, 95],
3     'EstimatedSalary': [1000, 250000, 350000],
4     'Purchased': [0, 1, 1]
5 })], ignore_index=True)
6
```

```
1 df.shape
```

```
➞ (403, 3)
```

```
1
```

✓ Train Test split

(Recommended to do before the train test split)

```
1 from sklearn.model_selection import train_test_split
2
3 X_train, X_test, y_train, y_test = train_test_split(df.drop('Purchased', axis=1),
4                                                    df['Purchased'],
5                                                    test_size=0.3,
6                                                    random_state=0)
7 X_train.shape, X_test.shape
```

```
➞ ((282, 2), (121, 2))
```

✓ Standard Scaler

```
1 from sklearn.preprocessing import StandardScaler
2 scaler = StandardScaler()
3 # fit the scaler to the train set, it will learn the parameter like mean and std de
4 scaler.fit(X_train)
5
6 # Perform standardization by centering and scaling on the train and test sets
7 # basically applying the formula
8 # and we only learn(fit) from the training data and transform on the both data
```

```
9 X_train_scaled = scaler.transform(X_train)
10 X_test_scaled = scaler.transform(X_test)
```

```
1 scaler.mean_
```

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

```
1 print(type(X_train))
2 print(type(X_train_scaled))
```

```
➦ <class 'pandas.core.frame.DataFrame'>
  <class 'numpy.ndarray'>
```

```
1 # since both the types are different so we have to modify the X_train_scaled to
2 # pandas dataframe
3
4 X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
5 X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

```
1 X_train_scaled.head(5)
```

```
➦
```

	Age	EstimatedSalary
0	-0.652473	-0.995561
1	1.903047	2.092697
2	-0.104862	1.922701
3	-0.652473	0.562734
4	-0.013593	-0.287245

Next steps:

 [View recommended plots](#)

[New interactive sheet](#)

```
1 np.round(X_train.describe(), 1)
```



	Age	EstimatedSalary
count	282.0	282.0
mean	38.1	69138.3
std	11.0	35357.7
min	5.0	1000.0
25%	30.0	43000.0
50%	37.0	68000.0
75%	46.0	86750.0
max	90.0	250000.0



```
1 np.round(X_train_scaled.describe(), 1)
```



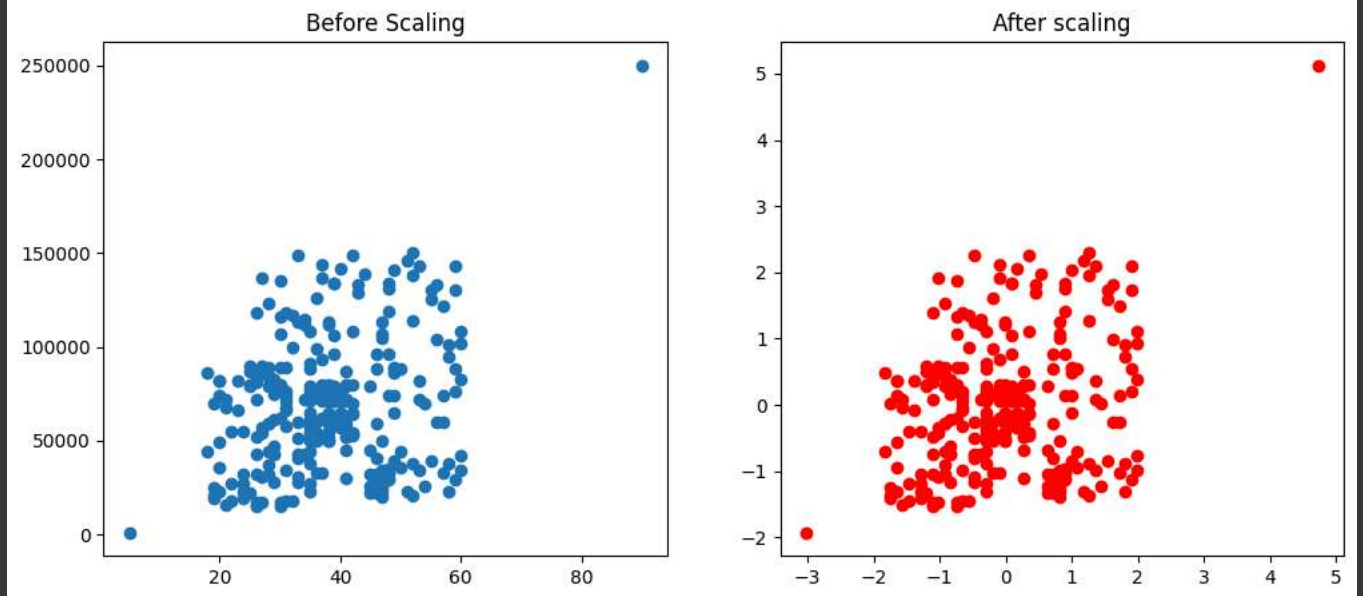
	Age	EstimatedSalary
count	282.0	282.0
mean	0.0	0.0
std	1.0	1.0
min	-3.0	-1.9
25%	-0.7	-0.7
50%	-0.1	-0.0
75%	0.7	0.5
max	4.7	5.1



✓ Effect of the **Scaling** after adding the outliers

outliers are still behaving like outliers

```
1 fig, (ax1, ax2) = plt.subplots(ncols = 2, figsize=(12, 5))
2
3 ax1.scatter(X_train['Age'], X_train['EstimatedSalary'])
4 ax1.set_title("Before Scaling")
5
6 ax2.scatter(X_train_scaled['Age'], X_train_scaled['EstimatedSalary'], color = "red")
7 ax2.set_title('After scaling')
8 plt.show()
```



Handling Outliers with Standardization

- **Issue:**
 - Standardization is sensitive to outliers since it uses mean and standard deviation.
 - **Observation:**
 - Outliers can still exist after scaling.
 - Scaling does not mitigate the effect of outliers.
 - **Recommendation:**
 - Detect and handle outliers before applying standardization.
-

When to Use Standardization

Algorithm(s)	Reason of applying feature scaling
1. K-Means	Use the Euclidean distance measure.
2. K-Nearest-Neighbours	Measure the distances between pairs of samples and these distances are influenced by the measurement units
3. Principal Component Analysis (PCA)	Try to get the feature with maximum variance
4. Artificial Neural Network	Apply Gradient Descent
5. Gradient Descent	Theta calculation becomes faster after feature scaling and the learning rate in the update equation of Stochastic Gradient Descent is the same for every parameter

Algorithms Sensitive to Feature Scale

- **Distance-Based Algorithms:**
 - K-Nearest Neighbors (KNN)
 - K-Means Clustering
 - Support Vector Machines (SVM)
- **Gradient Descent-Based Algorithms:**
 - Linear Regression
 - Logistic Regression
 - Neural Networks
- **Principal Component Analysis (PCA):**
 - Requires data to be scaled to capture maximum variance.

Algorithms Less Sensitive to Feature Scale

- **Tree-Based Models:**
 - Decision Trees
 - Random Forests
 - Gradient Boosting Machines (e.g., XGBoost)
- **Rationale:**
 - These models are based on splits and not distance calculations.
 - Scaling has minimal impact on their performance.

