Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

Home

Aniruddha Bhandari — **Published On April 3, 2020 and Last Modified On July 7th, 2023**

Beginner Data Engineering Python Regression Structured Data Technique

Feature engineering is a critical step in building accurate and effective machine learning models. One key aspect of feature engineering is scaling, normalization, and standardization, which involves transforming the data to make it more suitable for modeling. These techniques can help to improve model performance, reduce the impact of outliers, and ensure that the data is on the same scale. In this article, we will explore the concepts of scaling, normalization, and standardization, including why they are important and how to apply them to different types of data. By the end of this article, you'll have a thorough understanding of these essential feature engineering techniques and be able to apply them to your own machine learning projects.

Table of contents

What is Feature Scaling?

Feature scaling is a data preprocessing technique used to transform the values of features or variables in a dataset to a similar scale. The purpose is to ensure that all features contribute equally to the model and to avoid the domination of features with larger values.

Feature scaling becomes necessary when dealing with datasets containing features that have different ranges, units of measurement, or orders of magnitude. In such cases, the variation in feature values can lead to biased model performance or difficulties during the learning process.

There are several common techniques for feature scaling, including standardization, normalization, and min-max scaling. These methods adjust the feature values while preserving their relative relationships and distributions.

Recommended For You

Become a full stack data scientist

Feature Scaling Techniques in Python – A Complete Guide

Understand the Concept of Standardization in Machine Learning

Step by Step process of Feature Engineering for Machine Learning Algorithms in Data Science

Learn About Apache Spark Using Python

Feature Engineering (Feature Improvements – Scaling)

Unraveling Data Anomalies in Machine Learning

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

Why Should We Use Feature Scaling?

Some machine learning algorithms are sensitive to feature scaling, while others are virtually invariant. Let's explore these in more depth:

1. Gradient Descent Based Algorithms

Machine learning algorithms like linear regression, logistic regression, neural network, PCA (principal component analysis), etc., that use gradient descent as an optimization technique require data to be scaled. Take a look at the formula for gradient descent below:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

The presence of feature value X in the formula will affect the step size of the gradient descent. The difference in the ranges of features will cause different step sizes for each feature. To ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features, we scale the data before feeding it to the model.

> Having features on a similar scale can help the gradient descent converge more quickly towards the minima.

2. Distance-Based Algorithms

Distance algorithms like KNN, K-means clustering, and SVM(support vector machines) are most affected by the range of features. This is because, behind the scenes, **they are using distances between data points to determine their similarity.**

For example, let's say we have data containing high school CGPA scores of students (ranging from 0 to 5) and their future incomes (in thousands Rupees):

	Student	CGPA	Salary '000
0	1	3.0	60
1	2	3.0	40
2	3	4.0	40

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

algorithm; obviously, we do not want our algorithm to be biased towards one feature.

> Therefore, we scale our data before employing a distance based algorithm so that all the features contribute equally to the result.

	Student	CGPA	Salary '000
0	1	-1.184341	1.520013
1	2	-1.184341	-1.100699
2	3	0.416120	-1.100699
3	4	1.216350	0.209657
4	5	0.736212	0.471728

The effect of scaling is conspicuous when we compare the Euclidean distance between data points for students A and B, and between B and C, before and after scaling, as shown below:

- Distance AB before scaling =>
$$\sqrt{(40-60)^2+(3-3)^2}=20$$
- Distance BC before scaling =>
$$\sqrt{(40-40)^2+(4-3)^2}=1$$
- Distance AB after scaling =>
$$\sqrt{(1.1+1.5)^2+(1.18-1.18)^2}=2.6$$
- Distance BC after scaling =>
$$\sqrt{(1.1-1.1)^2+(0.41+1.18)^2}=1.59$$

3. Tree-Based Algorithms

Tree-based algorithms, on the other hand, are fairly insensitive to the scale of the features. Think about it, a decision tree only splits a node based on a single feature. The decision tree splits a node on a feature that increases the homogeneity of the node. Other features do not influence this split on a feature.

So, the remaining features have virtually no effect on the split. This is what makes them invariant to the scale of the features!

What Is Normalization?

Normalization is a data preprocessing technique used to adjust the values of features in a dataset to a common scale. This is done to facilitate data analysis and modeling, and to reduce the impact of different scales on the accuracy of machine learning models.

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Here's the formula for normalization:

$$X' = \frac{X - X_{min}}{}$$

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

hence X' is 0

- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator, and thus the value of X' is 1
- If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1

What Is Standardization?

Standardization is another scaling method where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero, and the resultant distribution has a unit standard deviation.

Here's the formula for standardization:

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

μ is the mean of the feature values and σ is the standard deviation of the feature values. Note that, in this case, the values are not restricted to a particular range.

Now, the big question in your mind must be when should we use normalization and when should we use standardization? Let's find out!

The Big Question – Normalize or Standardize?

Normalization	Standardization
Rescales values to a range between 0 and 1	Centers data around the mean and scales to a standard deviation of 1
Useful when the distribution of the data is unknown or not Gaussian	Useful when the distribution of the data is Gaussian or unknown
Sensitive to outliers	Less sensitive to outliers
Retains the shape of the original distribution	Changes the shape of the original distribution
May not preserve the relationships between the data points	Preserves the relationships between the data points
Equation: (x – min)/(max – min)	Equation: (x – mean)/standard deviation

However, at the end of the day, the choice of using normalization or standardization will depend on your problem and the machine learning algorithm you are using. There is no hard and fast rule to tell you when to normalize or standardize your data. **You can always start by fitting your model to raw, normalized, and standardized data and comparing the performance for the best results.**

It is a good practice to fit the scaler on the training data and then use it to transform the testing data. This would avoid any data leakage during the model testing process. Also, the scaling of target values is generally not required.

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

the DataHack platform.

I will skip the preprocessing steps since they are out of the scope of this tutorial. But you can find them neatly explained in this article. Those steps will enable you to reach the top 20 percentile on the hackathon leaderboard, so that's worth checking out!

So, let's first split our data into training and testing sets:

Python Code:

Before moving to the feature scaling part, let's glance at the details of our data using the **pd.describe()** method:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Tier 2	Tier 3	Supermarket Type1	Supermarket Type2
count	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000
mean	12.835420	0.355676	-2.940446	140.486413	15.154884	0.830302	0.326049	0.395571	0.651071	0.111616
std	4.233450	0.478753	0.791551	62.067053	8.383349	0.596352	0.468800	0.489009	0.476667	0.314917
min	4.555000	0.000000	-6.633875	31.290000	4.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	9.300000	0.000000	-3.467944	93.386700	9.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	12.857545	0.000000	-2.862536	142.179900	14.000000	1.000000	0.000000	0.000000	1.000000	0.000000
75%	16.000000	1.000000	-2.331264	184.495000	26.000000	1.000000	1.000000	1.000000	1.000000	0.000000
max	21.350000	1.000000	-1.113550	266.888400	28.000000	2.000000	1.000000	1.000000	1.000000	1.000000

We can see that there is a huge difference in the range of values present in our numerical features: **Item_Visibility**, **Item_Weight, Item_MRP,** and **Outlet_Establishment_Year**. Let's try and fix that using feature scaling!

Note: You will notice negative values in the Item_Visibility feature because I have taken log-transformation to deal with the skewness in the feature.

Normalization Using sklearn (scikit-learn)

To normalize your data, you need to import the MinMaxScaler from the sklearn library and apply it to our dataset. So, let's do that!

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

```
8   X_train_norm = norm.transform(X_train)
9
10  # transform testing dataabs
11  X_test_norm = norm.transform(X_test)
```

NormalizationVsStandarization_2.py hosted with ❤ by GitHub view raw

Let's see how normalization has affected our dataset:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Tier 2	Tier 3	Supermarket Type1	Supermarket Type2
count	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000
mean	0.493029	0.355676	0.595849	0.463485	0.464787	0.416151	0.328049	0.395571	0.651071	0.111616
std	0.252066	0.478753	0.175109	0.263444	0.349656	0.299176	0.469800	0.489009	0.476667	0.314917
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.282525	0.000000	0.479164	0.263566	0.208333	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.494352	0.000000	0.613084	0.470673	0.416667	0.500000	0.000000	0.000000	1.000000	0.000000
75%	0.681453	1.000000	0.730614	0.650280	0.916667	0.500000	1.000000	1.000000	1.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

All the features now have a minimum value of 0 and a maximum value of 1. Perfect!

Try out the above code in the live coding window below!!

Next, let's try to standardize our data.

Standardization Using sklearn

To standardize your data, you need to import the StandardScaler from the sklearn library and apply it to our dataset. Here's how you can do it:

```
1   # data standardization with  sklearn
2   from sklearn.preprocessing import StandardScaler
3
4   # copy of datasets
5   X_train_stand = X_train.copy()
6   X_test_stand = X_test.copy()
7
8   # numerical features
```

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

```
16
17      # transform the training data column
18      X_train_stand[i] = scale.transform(X_train_stand[[i]])
19
20      # transform the testing data column
21      X_test_stand[i] = scale.transform(X_test_stand[[i]])
```

NormalizationVsStandarization_3.py hosted with ❤ by GitHub view raw

You would have noticed that I only applied standardization to my numerical columns, not the other One-Hot Encoded features. Standardizing the One-Hot encoded features would mean assigning a distribution to categorical features. You don't want to do that!

But why did I not do the same while normalizing the data? Because One-Hot encoded features are already in the range between 0 to 1. So, normalization would not affect their value.

Right, let's have a look at how standardization has transformed our data:

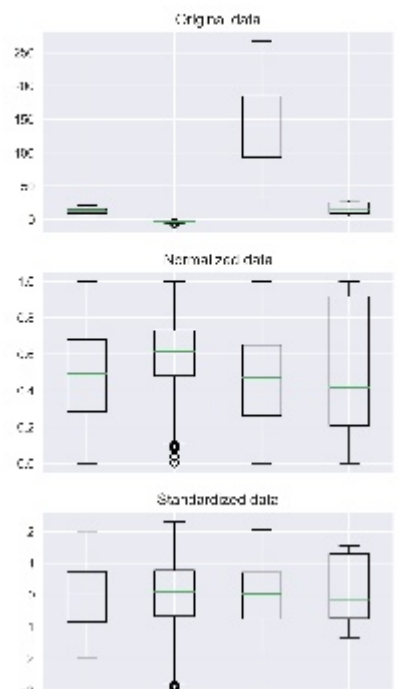
	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Tier 2	Tier 3	Supermarket Type1	Supermar Ty
count	6.818000e+03	6818.000000	6.818000e+03	6.818000e+03	6.818000e+03	6818.000000	6818.000000	6818.000000	6818.000000	6818.000
mean	1.704764e-16	0.365676	2.342737e-16	1.233002e-16	2.051747e-17	0.830302	0.326049	0.395571	0.651071	0.111
std	1.000073e+00	0.478753	1.000073e+00	1.000073e+00	1.000073e+00	0.596352	0.468800	0.489009	0.476667	0.314
min	-1.956094e+00	0.000000	-3.402972e+00	-1.759459e+00	-1.329746e+00	0.000000	0.000000	0.000000	0.000000	0.000
25%	-8.351767e-01	0.000000	-6.664603e-01	-7.589239e-01	-7.337084e-01	0.000000	0.000000	0.000000	0.000000	0.000
50%	5.250371e-03	0.000000	9.843446e-02	2.728679e-02	-1.376709e-01	1.000000	0.000000	0.000000	1.000000	0.000
75%	7.475728e-01	1.000000	7.696596e-01	7.091011e-01	1.292819e+00	1.000000	1.000000	1.000000	1.000000	0.000
max	2.011410e+00	1.000000	2.308161e+00	2.036689e+00	1.531234e+00	2.000000	1.000000	1.000000	1.000000	1.000

The numerical features are now centered on the mean with a unit standard deviation. Awesome!

Comparing Unscaled, Normalized, and Standardized Data

It is always great to visualize your data to understand the distribution present. We can see the comparison between our unscaled and scaled data using boxplots.

You can learn more about data visualization here*.*



Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

Applying Scaling to Machine Learning Algorithms

It's now time to train some machine learning algorithms on our data to compare the effects of different scaling techniques on the algorithm's performance. I want to see the effect of scaling on three algorithms in particular: K-Nearest Neighbors, Support Vector Regressor, and Decision Tree.

K-Nearest Neighbors

As we saw before, KNN is a distance-based algorithm that is affected by the range of features. Let's see how it performs on our data before and after scaling:

```python
# training a KNN model
from sklearn.neighbors import KNeighborsRegressor
# measuring RMSE score
from sklearn.metrics import mean_squared_error

# knn
knn = KNeighborsRegressor(n_neighbors=7)

rmse = []

# raw, normalized and standardized training and testing data
trainX = [X_train, X_train_norm, X_train_stand]
testX = [X_test, X_test_norm, X_test_stand]

# model fitting and measuring RMSE
for i in range(len(trainX)):

    # fit
    knn.fit(trainX[i],y_train)
    # predict
    pred = knn.predict(testX[i])
    # RMSE
    rmse.append(np.sqrt(mean_squared_error(y_test,pred)))

# visualizing the result
df_knn = pd.DataFrame({'RMSE':rmse},index=['Original','Normalized','Standardized'])
df_knn
```

NormalizationVsStandarization_4.py hosted with ❤ by GitHub view raw

	RMSE
Original	1319.283626
Normalized	1174.205859
Standardized	1183.448734

You can see that scaling the features has brought down the RMSE score of our KNN model. Specifically, the normalized data performs a tad bit better than the standardized data.

Note: I am measuring the RMSE here because this competition evaluates the RMSE.

Support Vector Regressor

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

```
# measuring RMSE score
from sklearn.metrics import mean_squared_error

# SVR
svr = SVR(kernel='rbf',C=5)

rmse = []

# raw, normalized and standardized training and testing data
trainX = [X_train, X_train_norm, X_train_stand]
testX = [X_test, X_test_norm, X_test_stand]

# model fitting and measuring RMSE
for i in range(len(trainX)):

    # fit
    svr.fit(trainX[i],y_train)
    # predict
    pred = svr.predict(testX[i])
    # RMSE
    rmse.append(np.sqrt(mean_squared_error(y_test,pred)))

# visualizing the result
df_svr = pd.DataFrame({'RMSE':rmse},index=['Original','Normalized','Standardized'])
df_svr
```

NormalizationVsStandarization_5.py hosted with ❤ by GitHub view raw

	RMSE
Original	1742.379686
Normalized	1655.643047
Standardized	1453.442942

We can see that scaling the features does bring down the RMSE score. And the standardized data has performed better than the normalized data. Why do you think that's the case?

The sklearn documentation states that SVM, with RBF kernel, assumes that all the features are centered around zero and variance is of the same order. This is because a feature with a variance greater than that of others prevents the estimator from learning from all the features. Great!

Decision Tree

We already know that a Decision tree is invariant to feature scaling. But I wanted to show a practical example of how it performs on the data:

```
# training a Decision Tree model
from sklearn.tree import DecisionTreeRegressor
# measuring RMSE score
from sklearn.metrics import mean_squared_error

# Decision tree
dt = DecisionTreeRegressor(max_depth=10,random_state=27)

rmse = []
```

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

```
17
18      # fit
19      dt.fit(trainX[i],y_train)
20      # predict
21      pred = dt.predict(testX[i])
22      # RMSE
23      rmse.append(np.sqrt(mean_squared_error(y_test,pred)))
24
25  # visualizing the result
26  df_dt = pd.DataFrame({'RMSE':rmse},index=['Original','Normalized','Standardized'])
27  df_dt
```

NormalizationVsStandarization_6.py hosted with ❤ by GitHub view raw

	RMSE
Original	1245.37439
Normalized	1245.37439
Standardized	1245.37439

You can see that the RMSE score has not moved an inch on scaling the features. So rest assured when you are using tree-based algorithms on your data!

Build Effective Machine Learning Models

This tutorial covered the relevance of using feature scaling on your data and how normalization and standardization have varying effects on the working of machine learning algorithms. Remember that there is no correct answer to when to use normalization over standardization and vice-versa. It all depends on your data and the algorithm you are using.

To enhance your skills in feature engineering and other key data science techniques, consider enrolling in our Data Science Black Belt program. Our comprehensive curriculum covers all aspects of data science, including advanced topics such as feature engineering, machine learning, and deep learning. With hands-on projects and mentorship, you'll gain practical experience and the skills you need to succeed in this exciting field. Enroll today and take your data science skills to the next level!

Frequently Asked Questions

Q1. How is Standardization different from Normalization feature scaling?

A. Standardization involves transforming the features such that they have a mean of zero and a standard deviation of one. This is done by subtracting the mean and dividing by the standard deviation of each feature.

On the other hand, normalization scales the features to a fixed range, usually [0, 1]. This is done by subtracting the minimum value of each feature and dividing by the difference between the maximum value and the minimum value.

Q2. Why is Standardization used in machine learning?

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

A. Normalization helps in scaling the input features to a fixed range, typically [0, 1], to ensure that no single feature disproportionately impacts the results. It preserves the relationship between the minimum and maximum values of each feature, which can be important for some algorithms. It also improves the convergence and stability of some machine learning algorithms, particularly those that use gradient-based optimization.

Q4. Why do we normalize values?

A. We normalize values to bring them into a common scale, making it easier to compare and analyze data. Normalization also helps to reduce the impact of outliers and improve the accuracy and stability of statistical models.

Q5. How do you normalize a set of values?

A. To normalize a set of values, we first calculate the mean and standard deviation of the data. Then, we subtract the mean from each value and divide by the standard deviation to obtain standardized values with a mean of 0 and a standard deviation of 1. Alternatively, we can use other normalization techniques such as min-max normalization, where we scale the values to a range of 0 to 1, or unit vector normalization, where we scale the values to have a length of 1.

Feature scaling feature scaling machine learning feature scaling python live coding normalizaiton vs. standardization normalization standardization

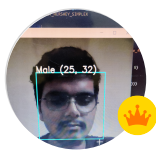
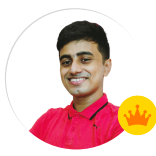
About the Author

Aniruddha Bhandari

Our Top Authors






view more

Download

Analytics Vidhya App for the Latest blog/Article

Previous Post

6 Open Source Data Science Projects to

Next Post

Supervised Learning vs. Unsupervised

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

2023)"

Ali says:
April 12, 2020 at 11:18 pm

Excelent article! Thank you very much for sharing. I have one question. In the post you say: "It is a good practice to fit the scaler on the training data and then use it to transform the testing data.", but I didn't see that in the code you posted. Am I wrong? How would one "fit the scaler on the training data and then use it to transform the testing data"? Thanks a lot again

Reply

Aniruddha Bhandari says:
April 13, 2020 at 11:44 am

Hi Ali You fit the scaler on the training data so that it can calculate the necessary parameters, like mean and standard deviation for standardization, and store it for later use using the fit() method. Later you use the transform() function to apply the same transformation on both, train and test dataset. I have used this approach for both, normalization and standardization, in the article in the gists "NormalizationVsStandarization_2.py" and "NormalizationVsStandarization_3.py" respectively. I hope this cleared your doubt. Thanks

Reply

sahil kamboj says:
April 28, 2020 at 4:25 pm

Good article! Thank you very much for sharing. I have one question. What the difference between sklearn.preprocessing import MinMaxScaler Normalization and sklearn.preprocessing.Normalizer? When to use MinMaxScaler and when to Normalize?

Reply

Aniruddha Bhandari says:
April 29, 2020 at 2:15 pm

Hi I hope MinMaxScaler is already clear from the article. Normalizer is also a normalization technique. The only difference is the way it computes the normalized values. By default, it is calculating the l2 norm of the row values i.e. each element of a row is normalized by the square root of the sum of squared values of all elements in that row. As mentioned in the documentation, it is useful in text classification where the dot product of two Tf-IDF vectors gives a cosine similarity between the different sentences/documents in the dataset. Other than that, as I mentioned in the article, there is no sure way to know which scaling technique should be used when. The best way is to create multiple scaled copies of the data and then try them out and see which one gives the best result. Hope this helps.

Reply

Subhash Kumar Nadar says:
May 23, 2020 at 11:25 am

Excellent article! Easy to understand and good coverage One question: I see that there is a scale() funtion as well from sklearn and short description suggest it to be similar to StandardScaler i.e. scaling to unit variance I could not find more than this explanation. Please can you suggest which on to use which scenario? Thanks in advance!

Reply

Golla kedarkumar says:
May 24, 2020 at 9:56 am

Hi ANIRUDDHA, If we use the same scaler for train and testing, does it affect the testing data because in standardization we need to use the mean of the data. If we take the mean of the train data and scale the test data, it will influence the test data, right?

Reply

Aniruddha Bhandari says:
May 24, 2020 at 12:11 pm

Scaling your test data according to the train data makes sure that the test data is on the same scale as the training data on which our model was trained on. This way our model will be able to apply the learnings from the training dataset on the testing dataset, which is exactly what we want! If instead, we scale the test data differently, then our model might not be able to discern that difference, thereby giving us incorrect outputs. That way we will never know how well our model is performing. I hope this helps!

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

happens in StandardScaler(). Second difference was that the scale function has no fit and transform methods, so you cannot apply the same scaling to your test dataset. I would suggest using the StandardScaler() function as I have never used the scale() personally. I hope this helps!

Reply

Inas says:
May 27, 2020 at 6:42 am

Excellent article, thank you for sharing.

Reply

soumadip roy says:
July 04, 2020 at 12:59 am

This is an excellent write up. Thanks for this.

Reply

HARSHVARDHAN BHATT says:
July 05, 2020 at 3:03 am

That graphs really helps in putting things in perspective...thanks !

Reply

Arnob says:
July 05, 2020 at 4:13 pm

Hey bro! Great article. It covered a lots of topics that were unclear to me before. I have a basic question. How can I check my data after normalization. You have mentioned to use pd.describe() in "Normalization using sklearn' section. But when I use it I get an error - " module 'pandas' has no attribute 'describe'". Can you tell me how to check my data after normalization? Thank you for your time.

Reply

deeps says:
July 15, 2020 at 7:59 pm

Excellent article !

Reply

Kunal says:
July 31, 2020 at 1:19 pm

Thanks for Great Article..!!!

Reply

Zineb says:
August 17, 2020 at 3:29 pm

Thanks Bhandari. Easy to understand and very helpful.

Reply

Aniruddha Bhandari says:
August 22, 2020 at 8:06 pm

Hi Arnob, glad you liked the article. The command you are looking for is df.describe() not pd.describe(). Try using that, it should work.

Reply

Tony says:
December 06, 2021 at 9:12 am

Hi Aniruddha!Quick question - I've seen a few people already mention that standardization is good if the feature follows Gaussian distribution but don't really get why. Mind shedding more light on that?

Reply

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)

Geofrey says:
December 13, 2022 at 4:36 pm

Quite an outstanding article! It answered most of my pressing questions in regards to scaling... Thank You so much.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

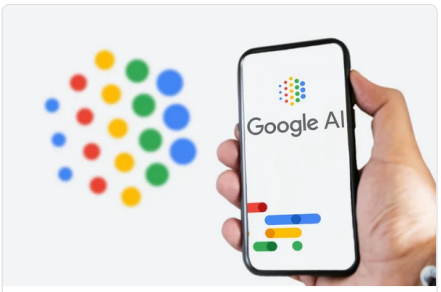
Website

Notify me of follow-up comments by email.

Notify me of new posts by email.

Submit

Top Resources



Google Adds AI-Powered Grammar Checker Feature: Learn How to Activate..

Swati Sharma -
AUG 09, 2023

10 AI Tools That Can Generate Code To Help Programmers

avcontentteam -
AUG 07, 2023

Generative AI in Healthcare

Mobarak Inuwa -
AUG 04, 2023

Understand Random Forest Algorithms With Examples (Updated 2023)

Sruthi E R - JUN 17, 2021

Feature Engineering: Scaling, Normalization, and Standardization (Updated 2023)