**Introduction**

Data Cleaning is the process of finding and correcting the inaccurate/incorrect data that are present in the dataset. One such process needed is to do something about the values that are missing in the dataset. In real life, many datasets will have many missing values, so dealing with them is an important step.

**Why do you need to fill in the missing data?** Because most of the machine learning models that you want to use will provide an error if you pass NaN values into it. The easiest way is to just fill them up with 0, but this can reduce your model accuracy significantly.

For filling missing values, there are many methods available. For choosing the best method, you need to understand the type of missing value and its significance, before you start filling/deleting the data.

# Working with Missing Data in Pandas

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in a real-life scenarios. Missing Data can also refer to as NA(Not Available) values in pandas. In DataFrame sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed. For Example, Suppose different users being surveyed may choose not to share their income, some users may choose not to share the address in this way many datasets went missing. In Pandas missing data is represented by two value:

* None: None is a Python singleton object that is often used for missing data in Python code.
* NaN : NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation

Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

* [isnull()](https://www.geeksforgeeks.org/python-pandas-isnull-and-notnull/)
* [notnull()](https://www.geeksforgeeks.org/python-pandas-isnull-and-notnull/)
* [dropna()](https://www.geeksforgeeks.org/python-pandas-dataframe-dropna/)
* [fillna()](https://www.geeksforgeeks.org/python-pandas-dataframe-fillna-to-replace-null-values-in-dataframe/)
* [replace()](https://www.geeksforgeeks.org/python-pandas-dataframe-replace/)
* [interpolate()](https://www.geeksforgeeks.org/python-pandas-dataframe-interpolate/)

In this article we are using CSV file, to download the CSV file used, Click [Here](https://media.geeksforgeeks.org/wp-content/uploads/employees.csv).

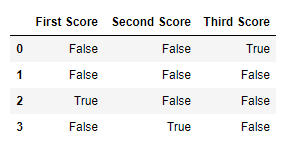
### Checking for missing values using isnull() and notnull()

In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull(). Both function help in checking whether a value is NaN or not. These function can also be used in Pandas Series in order to find null values in a series.

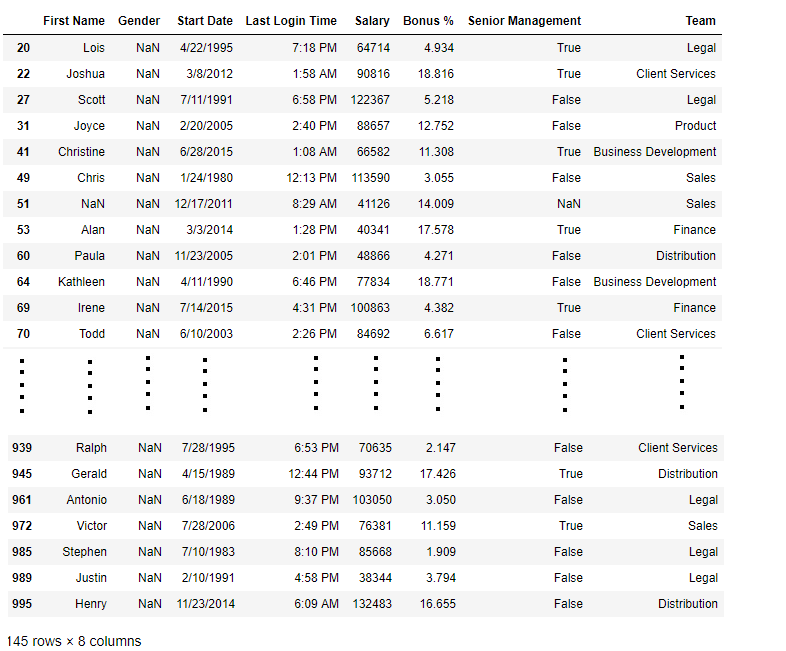
#### Checking for missing values using isnull()

In order to check null values in Pandas DataFrame, we use isnull() function this function return dataframe of Boolean values which are True for NaN values. **Code #1:**

|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, 45, 56, np.nan],          'Third Score':[np.nan, 40, 80, 98]}    # creating a dataframe from list  df = pd.DataFrame(dict)    # using isnull() function  df.isnull() |

**Output:**   **Code #2:**

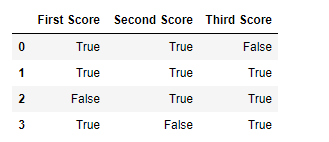
|  |
| --- |
| # importing pandas package  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("employees.csv")    # creating bool series True for NaN values  bool\_series = pd.isnull(data["Gender"])    # filtering data  # displaying data only with Gender = NaN  data[bool\_series] |

**Output:** As shown in the output image, only the rows having Gender = NULL are displayed. 

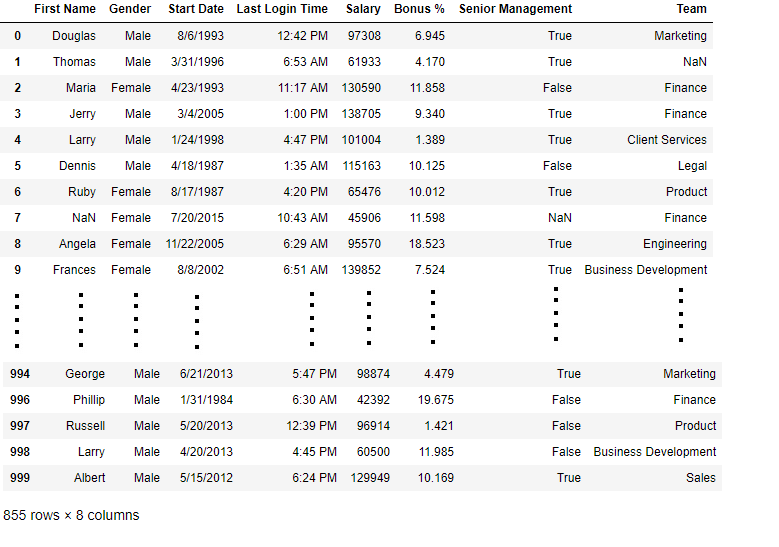
#### Checking for missing values using notnull()

In order to check null values in Pandas Dataframe, we use notnull() function this function return dataframe of Boolean values which are False for NaN values. **Code #3:**

|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, 45, 56, np.nan],          'Third Score':[np.nan, 40, 80, 98]}    # creating a dataframe using dictionary  df = pd.DataFrame(dict)    # using notnull() function  df.notnull() |

**Output:**   **Code #4:**

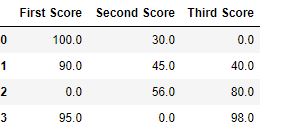
|  |
| --- |
| # importing pandas package  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("employees.csv")    # creating bool series True for NaN values  bool\_series = pd.notnull(data["Gender"])    # filtering data  # displaying data only with Gender = Not NaN  data[bool\_series] |

**Output:** As shown in the output image, only the rows having Gender = NOT NULL are displayed. 

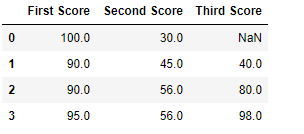
### Filling missing values using fillna(), replace() and interpolate()

In order to fill null values in a datasets, we use fillna(), replace() and interpolate() function these function replace NaN values with some value of their own. All these function help in filling a null values in datasets of a DataFrame. Interpolate() function is basically used to fill NA values in the dataframe but it uses various interpolation technique to fill the missing values rather than hard-coding the value. **Code #1:** Filling null values with a single value

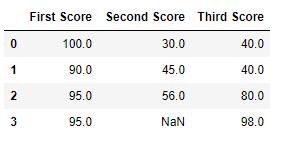
|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, 45, 56, np.nan],          'Third Score':[np.nan, 40, 80, 98]}    # creating a dataframe from dictionary  df = pd.DataFrame(dict)    # filling missing value using fillna()  df.fillna(0) |

**Output:**   **Code #2:** Filling null values with the previous ones

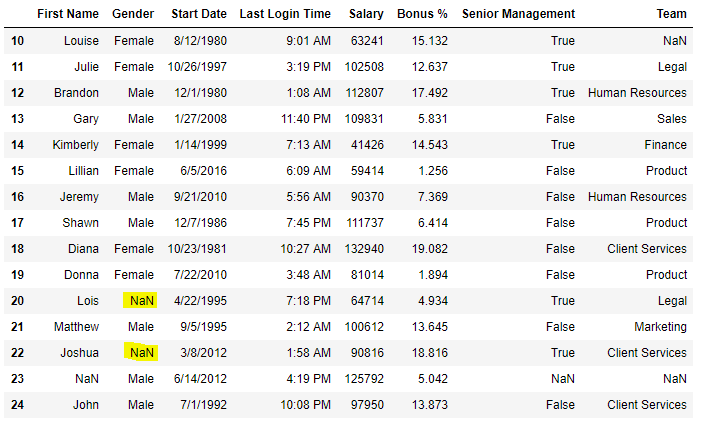
|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, 45, 56, np.nan],          'Third Score':[np.nan, 40, 80, 98]}    # creating a dataframe from dictionary  df = pd.DataFrame(dict)    # filling a missing value with  # previous ones  df.fillna(method ='pad') |

**Output:**   **Code #3:** Filling null value with the next ones

|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, 45, 56, np.nan],          'Third Score':[np.nan, 40, 80, 98]}    # creating a dataframe from dictionary  df = pd.DataFrame(dict)    # filling  null value using fillna() function  df.fillna(method ='bfill') |

**Output:**   **Code #4:** Filling null values in CSV File

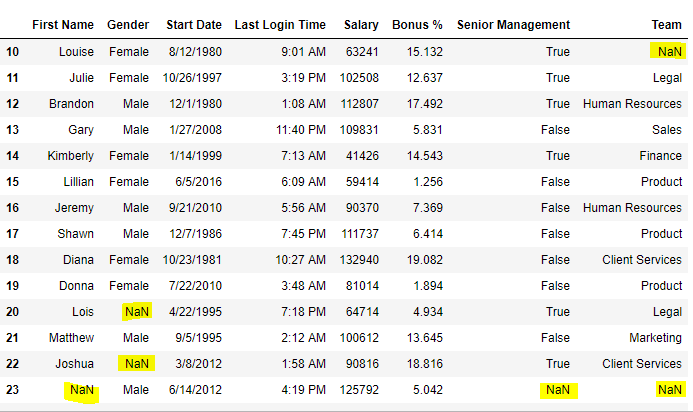
|  |
| --- |
| # importing pandas package  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("employees.csv")    # Printing the first 10 to 24 rows of  # the data frame for visualization  data[10:25] |

Now we are going to fill all the null values in Gender column with “No Gender”

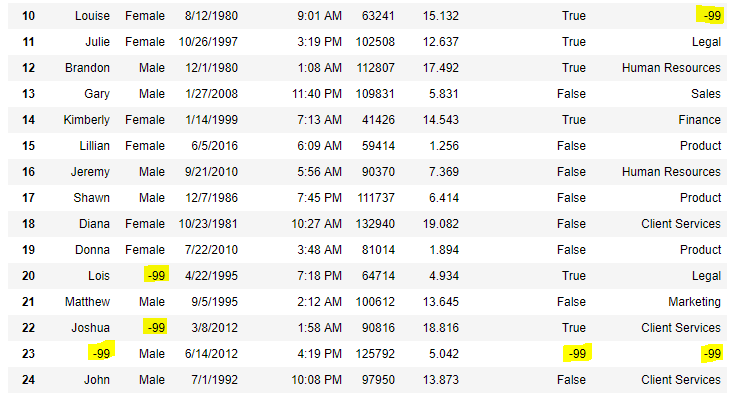
|  |
| --- |
| # importing pandas package  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("employees.csv")    # filling a null values using fillna()  data["Gender"].fillna("No Gender", inplace = True)    data |

**Output:** **Code #5:** Filling a null values using replace() method

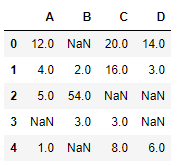
|  |
| --- |
| # importing pandas package  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("employees.csv")    # Printing the first 10 to 24 rows of  # the data frame for visualization  data[10:25] |

**Output:** Now we are going to replace the all Nan value in the data frame with -99 value.

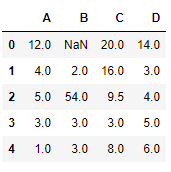
|  |
| --- |
| # importing pandas package  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("employees.csv")    # will replace  Nan value in dataframe with value -99  data.replace(to\_replace = np.nan, value = -99) |

**Output:**   **Code #6:** Using interpolate() function to fill the missing values using linear method.

|  |
| --- |
| # importing pandas as pd  import pandas as pd    # Creating the dataframe  df = pd.DataFrame({"A":[12, 4, 5, None, 1],                     "B":[None, 2, 54, 3, None],                     "C":[20, 16, None, 3, 8],                     "D":[14, 3, None, None, 6]})    # Print the dataframe  df |

Let’s interpolate the missing values using Linear method. Note that Linear method ignore the index and treat the values as equally spaced.

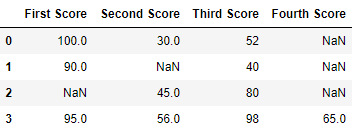
|  |
| --- |
| # to interpolate the missing values  df.interpolate(method ='linear', limit\_direction ='forward') |

**Output:** As we can see the output, values in the first row could not get filled as the direction of filling of values is forward and there is no previous value which could have been used in interpolation.

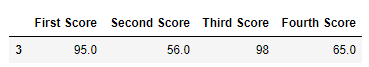
### Dropping missing values using dropna()

In order to drop a null values from a dataframe, we used dropna() function this function drop Rows/Columns of datasets with Null values in different ways. **Code #1:** Dropping rows with at least 1 null value.

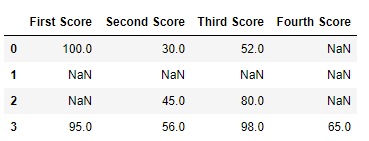
|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, np.nan, 45, 56],          'Third Score':[52, 40, 80, 98],          'Fourth Score':[np.nan, np.nan, np.nan, 65]}    # creating a dataframe from dictionary  df = pd.DataFrame(dict)    df |

Now we drop rows with at least one Nan value (Null value)

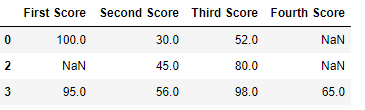
|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, 90, np.nan, 95],          'Second Score': [30, np.nan, 45, 56],          'Third Score':[52, 40, 80, 98],          'Fourth Score':[np.nan, np.nan, np.nan, 65]}    # creating a dataframe from dictionary  df = pd.DataFrame(dict)    # using dropna() function  df.dropna() |

**Output:** **Code #2:** Dropping rows if all values in that row are missing.

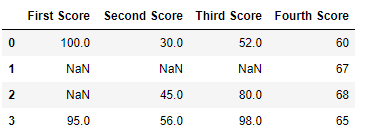
|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, np.nan, np.nan, 95],          'Second Score': [30, np.nan, 45, 56],          'Third Score':[52, np.nan, 80, 98],          'Fourth Score':[np.nan, np.nan, np.nan, 65]}    # creating a dataframe from dictionary  df = pd.DataFrame(dict)    df |

Now we drop a rows whose all data is missing or contain null values(NaN)

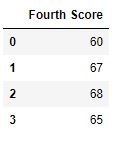
|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, np.nan, np.nan, 95],          'Second Score': [30, np.nan, 45, 56],          'Third Score':[52, np.nan, 80, 98],          'Fourth Score':[np.nan, np.nan, np.nan, 65]}    df = pd.DataFrame(dict)    # using dropna() function  df.dropna(how = 'all') |

**Output:** **Code #3:** Dropping columns with at least 1 null value.

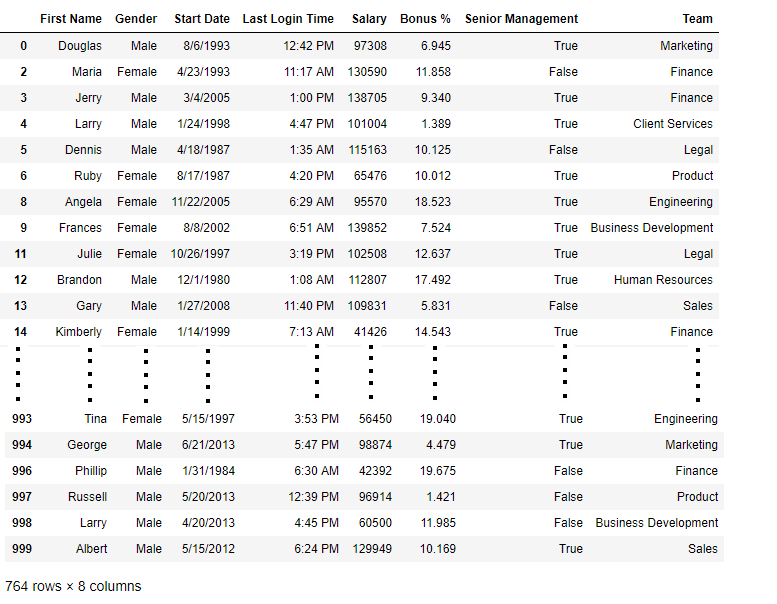
|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, np.nan, np.nan, 95],          'Second Score': [30, np.nan, 45, 56],          'Third Score':[52, np.nan, 80, 98],          'Fourth Score':[60, 67, 68, 65]}    # creating a dataframe from dictionary  df = pd.DataFrame(dict)    df |

Now we drop a columns which have at least 1 missing values

|  |
| --- |
| # importing pandas as pd  import pandas as pd    # importing numpy as np  import numpy as np    # dictionary of lists  dict = {'First Score':[100, np.nan, np.nan, 95],          'Second Score': [30, np.nan, 45, 56],          'Third Score':[52, np.nan, 80, 98],          'Fourth Score':[60, 67, 68, 65]}    # creating a dataframe from dictionary  df = pd.DataFrame(dict)    # using dropna() function  df.dropna(axis = 1) |

**Output :**   **Code #4:** Dropping Rows with at least 1 null value in CSV file

|  |
| --- |
| # importing pandas module  import pandas as pd    # making data frame from csv file  data = pd.read\_csv("employees.csv")    # making new data frame with dropped NA values  new\_data = data.dropna(axis = 0, how ='any')    new\_data |

**Output:** Now we compare sizes of data frames so that we can come to know how many rows had at least 1 Null value

|  |
| --- |
| print("Old data frame length:", len(data))  print("New data frame length:", len(new\_data))  print("Number of rows with at least 1 NA value: ", (len(data)-len(new\_data))) |

**Output :**

Old data frame length: 1000

New data frame length: 764

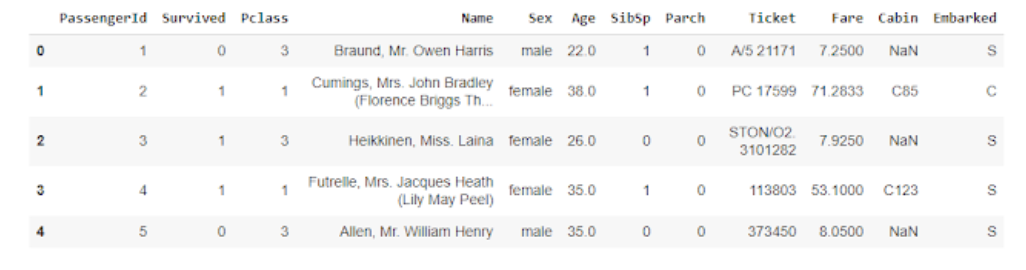
Number of rows with at least 1 NA value: 236

Since the difference is 236, there were 236 rows which had at least 1 Null value in any column.

**First Look at the Dataset**

* Import the required libraries that you will be using – numpy and pandas.

**Python Code:**



See that the contains many columns like PassengerId, Name, Age, etc..

 df.drop("Name",axis=1,inplace=True)

df.drop("Ticket",axis=1,inplace=True)

df.drop("PassengerId",axis=1,inplace=True)

df.drop("Cabin",axis=1,inplace=True)

df.drop("Embarked",axis=1,inplace=True)

See that there are also categorical values in the dataset, for this, you need to use Label Encoding or One Hot Encoding.

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df['Sex'] = le.fit\_transform(df['Sex'])

newdf=df

#splitting the data into x and y

y = df['Survived']

df.drop("Survived",axis=1,inplace=True)

**How to know whether the data has missing values?**

Missing values are usually represented in the form of Nan or null or None in the dataset.

df.info() the function can be used to give information about the dataset. This will provide you with the column names along with the number of non – null values in each column.

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 891 entries, 0 to 890

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Pclass 891 non-null int64

1 Sex 891 non-null int64

2 Age 714 non-null float64

3 SibSp 891 non-null int64

4 Parch 891 non-null int64

5 Fare 891 non-null float64

dtypes: float64(2), int64(4)

memory usage: 41.9 KB

See that there are null values in the column Age.

The second way of finding whether we have null values in the data is by using the isnull() function.

print(df.isnull().sum())

Pclass 0

Sex 0

Age 177

SibSp 0

Parch 0

Fare 0

dtype: int64

See that all the null values in the dataset are in the column – Age.

Let’s try fitting the data using logistic regression.

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test,y\_train,y\_test = train\_test\_split(df,y,test\_size=0.3)

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X\_train,y\_train)

---------------------------------------------------------------------------

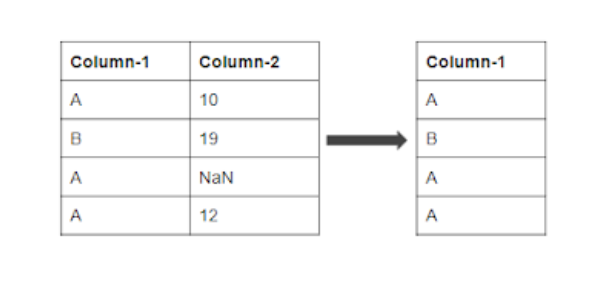
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

Here we can see that the logistic regression model does not work as we have NaN values in the dataset. Only some of the machine learning algorithms can work with missing data like KNN, which will ignore the values with Nan values.

Different methods that can be used to deal with the missing data.

1. Deleting the columns with missing data
2. Deleting the rows with missing data
3. Filling the missing data with a value – Imputation
4. Imputation with an additional column
5. Filling with a Regression Model

**1. Deleting the column with missing data**



updated\_df = df.dropna(axis=1)

updated\_df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 891 entries, 0 to 890

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Pclass 891 non-null int64

1 Sex 891 non-null int64

2 SibSp 891 non-null int64

3 Parch 891 non-null int64

4 Fare 891 non-null float64

dtypes: float64(1), int64(4)

memory usage: 34.9 KB

from sklearn import metrics

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test,y\_train,y\_test = train\_test\_split(updated\_df,y,test\_size=0.3)

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X\_train,y\_train)

pred = lr.predict(X\_test)

print(metrics.accuracy\_score(pred,y\_test))

0.7947761194029851

See that we are able to achieve an accuracy of 79.4%.

The problem with this method is that we may lose valuable information on that feature, as we have deleted it completely due to some null values.

Should only be used if there are too many null values.

**2. Deleting the row with missing data**

If there is a certain row with missing data, then you can delete the entire row with all the features in that row.

axis=1 is used to drop the column with `NaN` values.

axis=0 is used to drop the row with `NaN` values.

updated\_df = newdf.dropna(axis=0)

y1 = updated\_df['Survived']

updated\_df.drop("Survived",axis=1,inplace=True)

updated\_df.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 714 entries, 0 to 890

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Pclass 714 non-null int64

1 Sex 714 non-null int64

2 Age 714 non-null float64

3 SibSp 714 non-null int64

4 Parch 714 non-null int64

5 Fare 714 non-null float64

dtypes: float64(2), int64(4)

memory usage: 39.0 KB

from sklearn import metrics

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test,y\_train,y\_test = train\_test\_split(updated\_df,y1,test\_size=0.3)

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X\_train,y\_train)

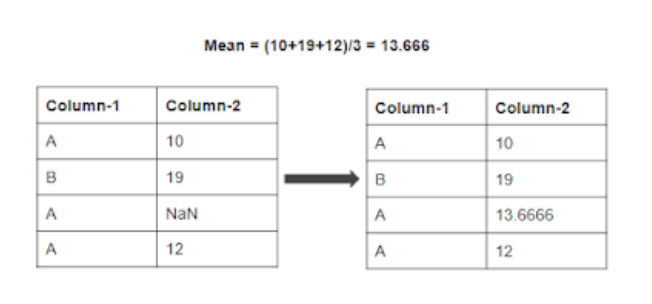
pred = lr.predict(X\_test)

print(metrics.accuracy\_score(pred,y\_test))

0.8232558139534883

In this case, see that we are able to achieve better accuracy than before. This is maybe because the column Age contains more valuable information than we expected.

**3. Filling the Missing Values – Imputation**



In this case, we will be filling the missing values with a certain number.

The possible ways to do this are:

1. Filling the missing data with the mean or median value if it’s a numerical variable.
2. Filling the missing data with mode if it’s a categorical value.
3. Filling the numerical value with 0 or -999, or some other number that will not occur in the data. This can be done so that the machine can recognize that the data is not real or is different.
4. Filling the categorical value with a new type for the missing values.

You can use the fillna() function to fill the null values in the dataset.

updated\_df = df

updated\_df['Age']=updated\_df['Age'].fillna(updated\_df['Age'].mean())

updated\_df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 891 entries, 0 to 890

Data columns (total 7 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Survived 891 non-null int64

1 Pclass 891 non-null int64

2 Sex 891 non-null int64

3 Age 891 non-null float64

4 SibSp 891 non-null int64

5 Parch 891 non-null int64

6 Fare 891 non-null float64

dtypes: float64(2), int64(5)

memory usage: 48.9 KB

y1 = updated\_df['Survived']

updated\_df.drop("Survived",axis=1,inplace=True)

from sklearn import metrics

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test,y\_train,y\_test = train\_test\_split(updated\_df,y1,test\_size=0.3)

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X\_train,y\_train)

pred = lr.predict(X\_test)

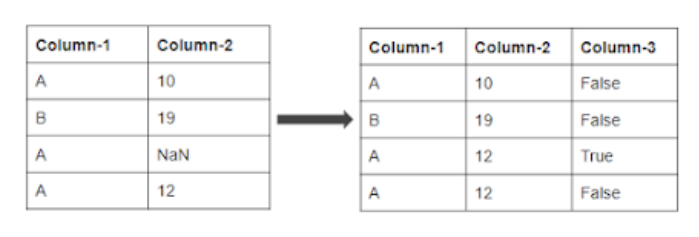
print(metrics.accuracy\_score(pred,y\_test))

0.7798507462686567

The accuracy value comes out to be 77.98% which is a reduction over the previous case.

This will not happen in general, in this case, it means that the mean has not filled the null value properly.

**4. Imputation with an additional column**



Use the SimpleImputer() function from sklearn module to impute the values.

Pass the strategy as an argument to the function. It can be either mean or mode or median.

The problem with the previous model is that the model does not know whether the values came from the original data or the imputed value. To make sure the model knows this, we are adding Ageismissing the column which will have True as value, if it is a null value and False if it is not a null value.

updated\_df = df

updated\_df['Ageismissing'] = updated\_df['Age'].isnull()

from sklearn.impute import SimpleImputer

my\_imputer = SimpleImputer(strategy = 'median')

data\_new = my\_imputer.fit\_transform(updated\_df)

updated\_df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 891 entries, 0 to 890

Data columns (total 7 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Pclass 891 non-null int64

1 Sex 891 non-null int64

2 Age 891 non-null float64

3 SibSp 891 non-null int64

4 Parch 891 non-null int64

5 Fare 891 non-null float64

6 Ageismissing 891 non-null bool

dtypes: bool(1), float64(2), int64(4)

memory usage: 42.8 KB

from sklearn import metrics

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test,y\_train,y\_test = train\_test\_split(updated\_df,y1,test\_size=0.3)

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X\_train,y\_train)

pred = lr.predict(X\_test)

print(metrics.accuracy\_score(pred,y\_test))

0.7649253731343284

**5. Filling with a Regression Model**

In this case, the null values in one column are filled by fitting a regression model using other columns in the dataset.

I.E in this case the regression model will contain all the columns except Age in X and Age in Y.

Then after filling the values in the Age column, then we will use logistic regression to calculate accuracy.

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

df.head()

testdf = df[df['Age'].isnull()==True]

traindf = df[df['Age'].isnull()==False]

y = traindf['Age']

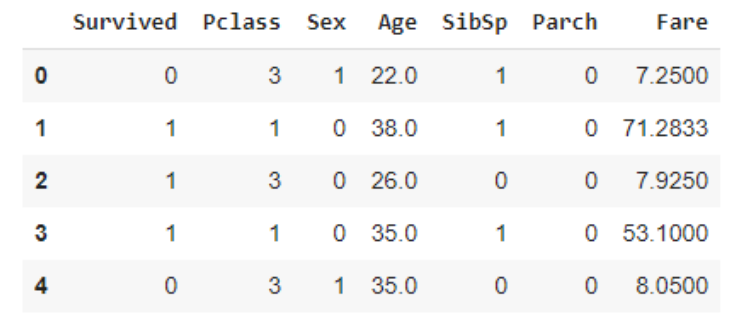
traindf.drop("Age",axis=1,inplace=True)

lr.fit(traindf,y)

testdf.drop("Age",axis=1,inplace=True)

pred = lr.predict(testdf)

testdf['Age']= pred



traindf['Age']=y

y = traindf['Survived']

traindf.drop("Survived",axis=1,inplace=True)

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr.fit(traindf,y)

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='auto', n\_jobs=None, penalty='l2',

random\_state=None, solver='lbfgs', tol=0.0001, verbose=0,

warm\_start=False)

y\_test = testdf['Survived']

testdf.drop("Survived",axis=1,inplace=True)

pred = lr.predict(testdf)

print(metrics.accuracy\_score(pred,y\_test))

0.8361581920903954

See that this model produces more accuracy than the previous model as we are using a specific regression model for filling the missing values.

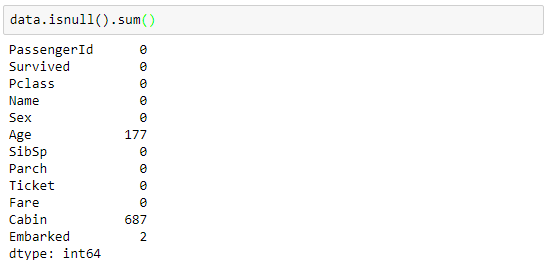
We can also use models  KNN for filling the missing values. But sometimes, using models for imputation can result in overfitting the data.

Imputing missing values using the regression model allowed us to improve our model compared to dropping those columns.

But you have to understand that There is no perfect way for filling the missing values in a dataset.

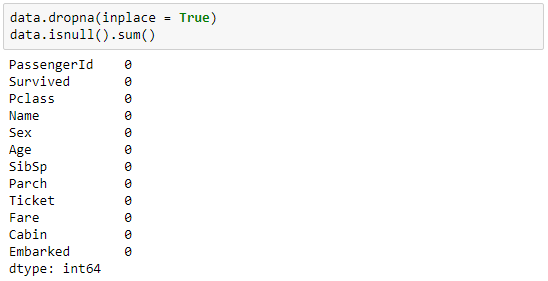
Each of the methods that I have discussed in this blog, may work well with different types of datasets. You have to experiment through different methods, to check which method works the best for your dataset.

Let us get started. To understand various methods we will be working on the [Titanic](https://analyticsindiamag.com/solving-the-titanic-ml-survival-problem-using-random-forest-vs-neural-networks-on-tensorflow-which-one-is-better/) dataset:

[](https://149695847.v2.pressablecdn.com/wp-content/uploads/2018/02/missing-values-1.png)

### 1. Deleting Rows

This method commonly used to handle the null values. Here, we either delete a particular row if it has a null value for a particular feature and a particular column if it has more than 70-75% of missing values. This method is advised only when there are enough samples in the data set. One has to make sure that after we have deleted the data, there is no addition of bias. Removing the data will lead to loss of information which will not give the expected results while predicting the output.

[](https://149695847.v2.pressablecdn.com/wp-content/uploads/2018/02/deleted.png)

### Pros:

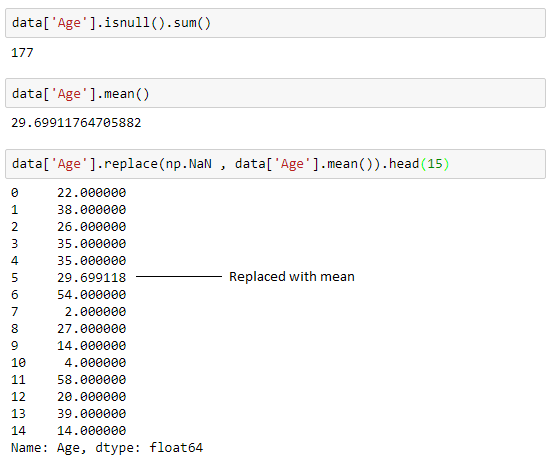
* Complete removal of data with missing values results in robust and highly accurate model
* Deleting a particular row or a column with no specific information is better, since it does not have a high weightage

### Cons:

* Loss of information and data
* Works poorly if the percentage of missing values is high (say 30%), compared to the whole dataset

## 2. Replacing With Mean/Median/Mode

This strategy can be applied on a feature which has numeric data like the age of a person or the ticket fare. We can calculate the mean, median or mode of the feature and replace it with the missing values. This is an approximation which can add variance to the data set. But the loss of the data can be negated by this method which yields better results compared to removal of rows and columns. Replacing with the above three approximations are a statistical approach of handling the missing values. This method is also called as leaking the data while training. Another way is to approximate it with the deviation of neighbouring values. This works better if the data is linear.

[](https://149695847.v2.pressablecdn.com/wp-content/uploads/2018/02/mean.png)

To replace it with median and mode we can use the following to calculate the same:

[](https://149695847.v2.pressablecdn.com/wp-content/uploads/2018/02/median-and-mode.png)

### Pros:

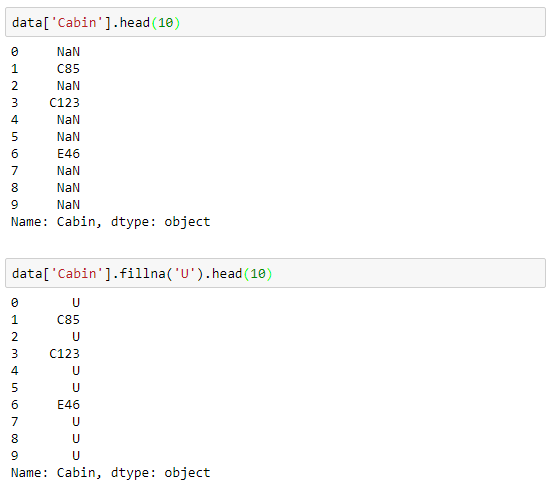
* This is a better approach when the data size is small
* It can prevent data loss which results in removal of the rows and columns

### Cons:

* Imputing the approximations add variance and bias
* Works poorly compared to other multiple-imputations method

## 3. Assigning An Unique Category

A categorical feature will have a definite number of possibilities, such as gender, for example. Since they have a definite number of classes, we can assign another class for the missing values. Here, the features Cabin and Embarked have missing values which can be replaced with a new category, say, U for ‘unknown’. This strategy will add more information into the dataset which will result in the change of variance. Since they are categorical, we need to find one hot encoding to convert it to a numeric form for the algorithm to understand it. Let us look at how it can be done in Python:

[](https://149695847.v2.pressablecdn.com/wp-content/uploads/2018/02/cabin.png)

### Pros:

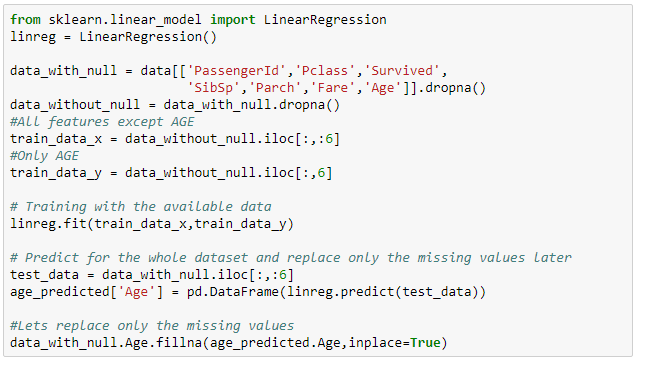
* Less possibilities with one extra category, resulting in low variance after one hot encoding — since it is categorical
* Negates the loss of data by adding an unique category

### Cons:

* Adds less variance
* Adds another feature to the model while encoding, which may result in poor performance

## 4. Predicting The Missing Values

Using the features which do not have missing values, we can predict the nulls with the help of a machine learning algorithm. This method may result in better accuracy, unless a missing value is expected to have a very high variance. We will be using linear [regression](https://analyticsindiamag.com/top-6-regression-algorithms-used-data-mining-applications-industry/) to replace the nulls in the feature ‘age’, using other available features. One can experiment with different algorithms and check which gives the best accuracy instead of sticking to a single algorithm.

[](https://149695847.v2.pressablecdn.com/wp-content/uploads/2018/02/predicted-1.png)

### Pros:

* Imputing the missing variable is an improvement as long as the bias from the same is smaller than the omitted variable bias
* Yields unbiased estimates of the model parameters

### Cons:

* Bias also arises when an incomplete conditioning set is used for a categorical variable
* Considered only as a proxy for the true values

## 5. Using Algorithms Which Support Missing Values

KNN is a machine learning algorithm which works on the principle of distance measure. This algorithm can be used when there are nulls present in the dataset. While the algorithm is applied, KNN considers the missing values by taking the majority of the K nearest values. In this particular dataset, taking into account the person’s age, sex, class etc, we will assume that people having same data for the above mentioned features will have the same kind of fare.

Unfortunately, the SciKit Learn library for the K – Nearest Neighbour algorithm in Python does not support the presence of the missing values.

Another algorithm which can be used here is RandomForest. This model produces a robust result because it works well on non-linear and the categorical data. It adapts to the data structure taking into consideration of the high variance or the bias, producing better results on large datasets.

### Pros:

* Does not require creation of a predictive model for each attribute with missing data in the dataset
* Correlation of the data is neglected

### Cons:

* Is a very time consuming process and it can be critical in data mining where large databases are being extracted
* Choice of distance functions can be Euclidean, Manhattan etc. which is do not yield a robust result

## Conclusion

Every dataset we come across will almost have some missing values which need to be dealt with. But handling them in an intelligent way and giving rise to robust models is a challenging task. We have gone through a number of ways in which nulls can be replaced. It is not necessary to handle a particular dataset in one single manner. One can use various methods on different features depending on how and what the data is about. Having a small domain knowledge about the data is important, which can give you an insight about how to approach the problem.

# Data Normalization with Pandas

* **Last Updated :** 11 Dec, 2020

 Read

 Discuss

In this article, we will learn how to normalize data in Pandas. Let’s discuss some concepts first :

* **Pandas:** Pandas is an open-source library that’s built on top of NumPy library. it is a Python package that provides various data structures and operations for manipulating numerical data and statistics. It’s mainly popular for importing and analyzing data much easier. Pandas is fast and it’s high-performance & productive for users.
* **Data Normalization**: Data Normalization could also be a typical practice in machine learning which consists of transforming numeric columns to a standard scale. In machine learning, some feature values differ from others multiple times. The features with higher values will dominate the learning process.

### Steps Needed

Here, we will apply some techniques to normalize the data and discuss these with the help of examples. For this, let’s understand the steps needed for data normalization with Pandas.

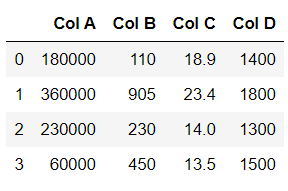
1. Import Library (Pandas)
2. Import / Load / Create data.
3. Use the technique to normalize the data.

### ****Examples****

Here, we create data by some random values and apply some normalization techniques to it.

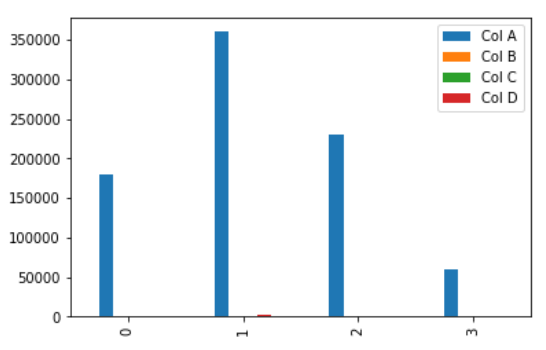
|  |
| --- |
| # importing packages  import pandas as pd    # create data  df = pd.DataFrame([                     [180000, 110, 18.9, 1400],                     [360000, 905, 23.4, 1800],                     [230000, 230, 14.0, 1300],                     [60000, 450, 13.5, 1500]],                       columns=['Col A', 'Col B',                              'Col C', 'Col D'])    # view data  display(df) |

**Output:**



See the plot of this dataframe:

|  |
| --- |
| import matplotlib.pyplot as plt  df.plot(kind = 'bar') |



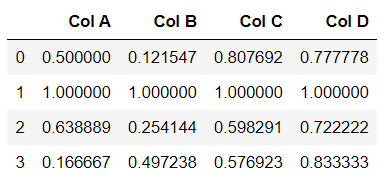
### Let’s apply normalization techniques one by one.

### Using The maximum absolute scaling

The maximum absolute scaling rescales each feature between -1 and 1 by dividing every observation by its maximum absolute value. We can apply the maximum absolute scaling in Pandas using the .max() and .abs() methods, as shown below.

|  |
| --- |
| # copy the data  df\_max\_scaled = df.copy()    # apply normalization techniques  for column in df\_max\_scaled.columns:      df\_max\_scaled[column] = df\_max\_scaled[column]  / df\_max\_scaled[column].abs().max()    # view normalized data  display(df\_max\_scaled) |

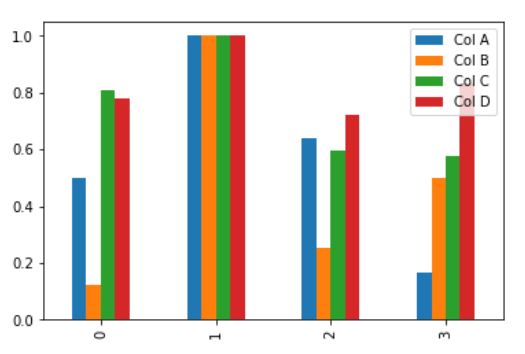
**Output :**



See the plot of this dataframe:

|  |
| --- |
| import matplotlib.pyplot as plt  df\_max\_scaled.plot(kind = 'bar') |

**Output:**

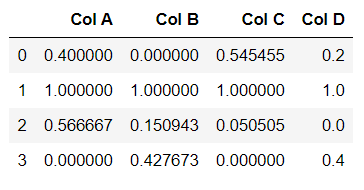


### Using The min-max feature scaling

The min-max approach (often called normalization) rescales the feature to a hard and fast range of [0,1] by subtracting the minimum value of the feature then dividing by the range. We can apply the min-max scaling in Pandas using the .min() and .max() methods.

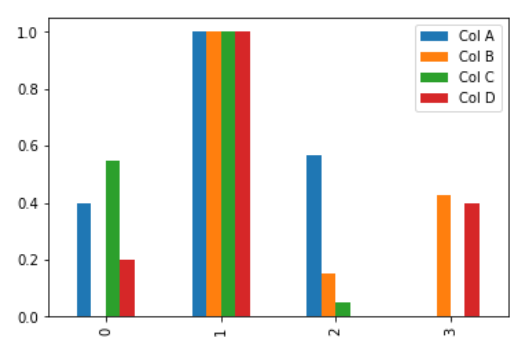
|  |
| --- |
| # copy the data  df\_min\_max\_scaled = df.copy()    # apply normalization techniques  for column in df\_min\_max\_scaled.columns:      df\_min\_max\_scaled[column] = (df\_min\_max\_scaled[column] - df\_min\_max\_scaled[column].min()) / (df\_min\_max\_scaled[column].max() - df\_min\_max\_scaled[column].min())    # view normalized data  print(df\_min\_max\_scaled) |

**Output :**



Let’s draw a plot with this dataframe:

|  |
| --- |
| import matplotlib.pyplot as plt  df\_min\_max\_scaled.plot(kind = 'bar') |

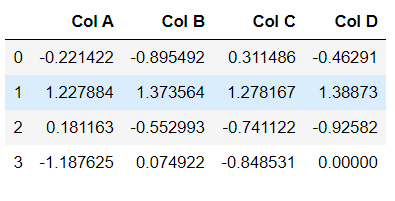


### Using The z-score method

The z-score method (often called standardization) transforms the info into distribution with a mean of 0 and a typical deviation of 1. Each standardized value is computed by subtracting the mean of the corresponding feature then dividing by the quality deviation.

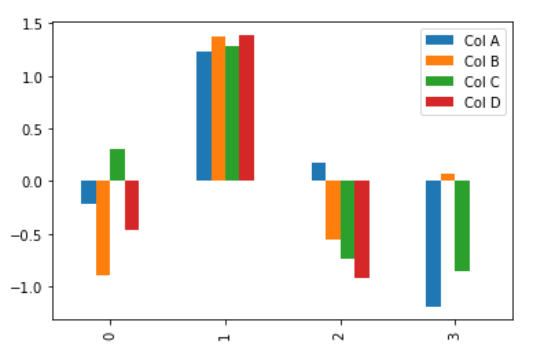
|  |
| --- |
| # copy the data  df\_z\_scaled = df.copy()    # apply normalization techniques  for column in df\_z\_scaled.columns:      df\_z\_scaled[column] = (df\_z\_scaled[column] -                             df\_z\_scaled[column].mean()) / df\_z\_scaled[column].std()    # view normalized data  display(df\_z\_scaled) |

**Output :**



Let’s draw a plot with this dataframe.

|  |
| --- |
| import matplotlib.pyplot as plt  df\_z\_scaled.plot(kind='bar') |



### Summary

Data normalization consists of remodeling numeric columns to a standard scale. In Python, we will implement data normalization in a very simple way. The Pandas library contains multiple built-in methods for calculating the foremost common descriptive statistical functions which make data normalization techniques very easy to implement.