**How to sort dataframe by index and how to sort dataframe by specific column?**

DataFrame\_Name.sort\_index(axis=0)

 **axis-** Specifies on which basis to sort whether based on index or column. By default, it sorts based on index i.e. axis=0.

**What is lambda expression?**

Lambda functions are similar to user-defined functions but without a name. They're commonly referred to as anonymous functions.

Lambda functions are efficient whenever you want to create a function that will only contain simple expressions – that is, expressions that are usually a single line of a statement. They're also useful when you want to use the function once.

lambda argument(s) : expression

1. lambda is a keyword in Python for defining the anonymous function.
2. argument(s) is a placeholder, that is a variable that will be used to hold the value you want to pass into the function expression. A lambda function can have multiple variables depending on what you want to achieve.
3. expression is the code you want to execute in the lambda function.

**How to detect and remove outliers using pandas**

**Detecting Outliers**

There are various techniques to detect outliers in a pandas DataFrame. Let’s discuss some of the most commonly used methods.

**Z-Score**

The z-score is a statistical measure that indicates how many standard deviations a data point is away from the mean. The z-score can be calculated using the following formula:

z = (x - mean) / std

where x is the data point, mean is the mean of the dataset, and std is the standard deviation of the dataset.

To identify outliers using the z-score, we can set a threshold value, say 3. Any data point with a z-score greater than 3 or less than -3 can be considered an outlier. We can use the scipy library in [Python](https://saturncloud.io/glossary/python) to calculate the z-score and identify outliers.

**import** pandas **as** pd

**import** numpy **as** np

**from** scipy **import** stats

*#Create a sample DataFrame of student heights*

df **=** pd**.**DataFrame**({**'Height'**:** **[170,** **160,** **130,** **190,** **180,** **150,** **140,** **200,** **175,** **165]})**

*# Calculate the z-score for each student's height*

z **=** np**.**abs**(**stats**.**zscore**(**df**[**'Height'**]))**

*# Identify outliers as students with a z-score greater than 3*

threshold **=** **3**

outliers **=** df**[**z **>** threshold**]**

*# Print the outliers*

**print(**outliers**)**

In the above code, we calculate the z-score for column Height using stats.zscore() and set a threshold of 3 to identify outliers. We then filter the DataFrame to obtain the outliers.

**Interquartile Range (IQR)**

The interquartile range (IQR) is a measure of the spread of the middle 50% of the data. The IQR can be calculated as the difference between the 75th percentile and the 25th percentile of the dataset. Any data point outside the range of 1.5 times the IQR below the 25th percentile or above the 75th percentile can be considered an outlier.

To identify outliers using the IQR, we can use the quantile() function in pandas to calculate the 25th and 75th percentiles of the dataset. We can then calculate the IQR and use it to identify outliers.

*# calculate IQR for column Height*

Q1 **=** df**[**'Height'**].**quantile**(0.25)**

Q3 **=** df**[**'Height'**].**quantile**(0.75)**

IQR **=** Q3 **-** Q1

*# identify outliers*

threshold **=** **1.5**

outliers **=** df**[(**df**[**'Height'**]** **<** Q1 **-** threshold **\*** IQR**)** **|** **(**df**[**'Height'**]** **>** Q3 **+** threshold **\*** IQR**)]**

In the above code, we calculate the 25th and 75th percentiles of column Height using quantile() and calculate the IQR. We then set a threshold of 1.5 to identify outliers and filter the DataFrame to obtain the outliers.

**Excluding Outliers**

Once we have identified the outliers in our dataset, we can either exclude them from our analysis or replace them with more accurate values.

**Excluding Outliers**

To exclude outliers from our analysis, we can simply remove the rows containing the outliers from our DataFrame. We can use the drop() function in pandas to remove the rows containing the outliers.

*# drop rows containing outliers*

df **=** df**.**drop**(**outliers**.**index**)**

In the above code, we use the drop() function to remove the rows containing the outliers identified in the previous section.

**Replacing Outliers**

To replace outliers with more accurate values, we can use various techniques such as interpolation or imputation. Interpolation involves filling in the missing values using the values of neighboring data points, while imputation involves estimating the missing values based on other features of the dataset.

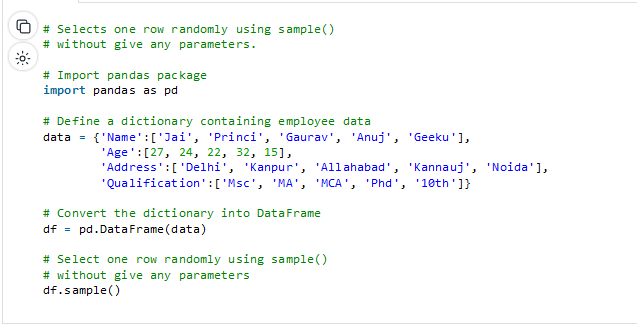
*# replace outliers with median value*

df**.**loc**[**z **>** threshold**,** 'Height'**]** **=** df**[**'Height'**].**median**()**

In the above code, we replace the outliers in column Height with the median value of the column.

**How to read random rows from our data?**

Sample method returns a random sample of items from an axis of object and this object of same type as your caller.



Using parameter *n*, which selects *n* numbers of rows randomly.

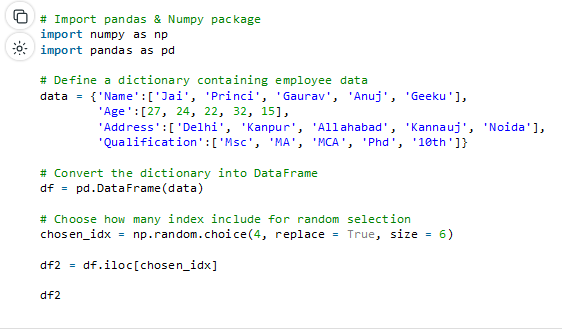
Select *n* numbers of rows randomly using sample(n) or sample(n=n). Each time you run this, you get n different rows.

Using frac parameter.

One can do fraction of axis items and get rows. For example, if frac= .5 then sample method return 50% of rows.

The axis accepts number or name. sample() method also allows users to sample columns instead of rows using the axis argument.

Numpy choose how many index include for random selection and we can allow replacement.

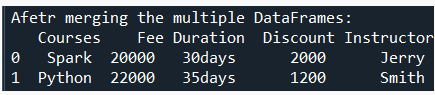


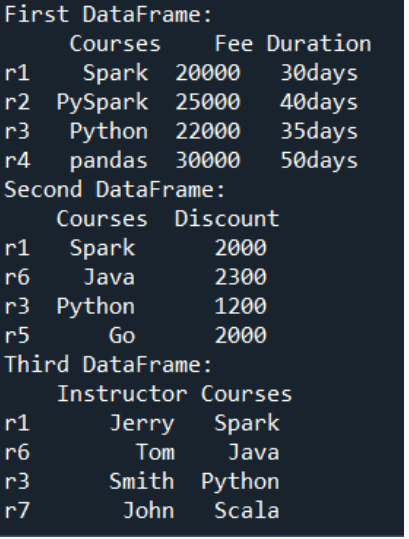
**How to combine more than two dataframes at the same tiem?**

Pandas merge() function is used to merge multiple Dataframes. We can use either pandas.merge() or DataFrame.merge() to merge multiple Dataframes. Merging multiple Dataframes is similar to SQL join and supports different types of join inner, left, right, outer, cross.

Similar to the Database join, the [merge() method](https://sparkbyexamples.com/python/pandas-merge-dataframes-explained-examples/) also supports several join types like left, right, inner, outer and cross. Use how param to specify the join type.

By default merge() uses inner join on columns that are present on both DataFrames. You can also explicitly specify the columns you want to join and join by row index.





**Replace data in specific columns with specific value؟**

### Using the .replace() Method

The .replace() method is a versatile way to replace values in a DataFrame. You can specify the column, the value to replace, and the replacement value. Here’s an example:

*# Replace 'New York' with 'NY' in the 'City' column*

df\_copy **=** df**.**copy**()**

df\_copy**[**'City'**].**replace**(**'New York'**,** 'NY'**,** inplace**=**True**)**

**print(**df\_copy**)**

Output:

name age City

0 Alice 25 NY

1 Bob 30 Los Angeles

2 Charlie 35 San Francisco

3 David 40 Chicago

This code snippet will replace all occurrences of New York in the City column with NY. Setting inplace=True modifies the original DataFrame; otherwise, it returns a new DataFrame with the replacements.

### Using .loc[] for Conditional Replacement

You can use .loc[] to replace values based on a condition. For instance, let’s say you want to replace ages greater than 30 with a default value of 30:

*# Replace ages greater than 30 with 30*

df\_copy **=** df**.**copy**()**

df\_copy**.**loc**[**df**[**'Age'**]** **>** **30,** 'Age'**]** **=** **30**

**print(**df\_copy**)**

Output:

name age City

0 Alice 25 New York

1 Bob 30 Los Angeles

2 Charlie 30 San Francisco

3 David 30 Chicago

### Using apply() with a Custom Function

You can also use the apply() function with a custom function for more complex replacements. Here’s an example where we replace city names with their respective state names:

state\_mapping **=** **{**

'New York'**:** 'NY'**,**

'Los Angeles'**:** 'CA'**,**

'San Francisco'**:** 'CA'**,**

'Chicago'**:** 'IL'

**}**

*# Define a custom function*

**def** replace\_city\_with\_state**(**city**):**

**return** state\_mapping**.**get**(**city**,** city**)**

*# Apply the custom function to the 'City' column*

df\_copy **=** df**.**copy**()**

df\_copy**[**'City'**]** **=** df\_copy**[**'City'**].**apply**(**replace\_city\_with\_state**)**

**print(**df\_copy**)**

This method is especially useful when you need to perform replacements that are not straightforward to express with a single value-to-value mapping.

Output:

name age City

0 Alice 25 NY

1 Bob 30 CA

2 Charlie 30 CA

3 David 30 IL

### Using str.replace() for String Columns

If you want to replace values within string columns, you can use the .str.replace() method. For instance, let’s replace Alice with Alicia in the Name column:

df\_copy **=** df**.**copy**()**

df\_copy**[**'Name'**]** **=** df**[**'Name'**].**str**.**replace**(**'Alice'**,** 'Alicia'**)**

**print(**df\_copy**)**

**Write dataframes to two separate sheets in excel**

**Pandas** provide a function called **xlsxwriter** for this purpose. ExcelWriter() is a class that allows you to write DataFrame objects into Microsoft Excel sheets. Text, numbers, strings, and formulas can all be written using ExcelWriter(). It can also be used on several worksheets.

The to\_excel() method is used to export the DataFrame to the excel file. To write a single object to the excel file, we have to specify the target file name. If we want to write to multiple sheets, we need to create an ExcelWriter object with target filename and also need to specify the sheet in the file in which we have to write. The multiple sheets can also be written by specifying the unique sheet\_name. It is necessary to save the changes for all the data written to the file.

Create some **sample data frames** using **pandas.DataFrame** function. Now, create a **writer variable** and specify the **path** in which you wish to store the excel file and the **file name**, inside the pandas **excelwriter function**.

