## **Exploring the Dataset with Pandas - Part 3**

Pandas offers a plethora of functionalities for manipulating and analyzing datasets, making it a go-to choice for data scientists and analysts worldwide. In this blog post, we'll delve into three essential data analysis operations in Pandas: rolling mean functionality, converting DataFrame columns to lists, and calculating column variances.

1. **Creating a DataFrame from a Dictionary:**import pandas as pd  
     
   data = {"a": [1, 2, 3, 4],  
    "b": [4, 5, 6, 7],  
    "c": ["sudh", "krish", "hitesh", "navin"]}  
     
   **df = pd.DataFrame(data)**
   * **Explanation:** pd.DataFrame(data) creates a DataFrame from the provided dictionary data, where keys represent column names and values represent column data. Each key-value pair in the dictionary corresponds to a column in the DataFrame.
2. **Setting Index of DataFrame:  
   df.set\_index('a', inplace=True)**
   * **Explanation:** df.set\_index('a', inplace=True) sets the index of DataFrame df to the values in the column labeled 'a'. By setting the index, we can use the values in the specified column as the index labels for the DataFrame.
3. **Resetting Index of DataFrame:  
   df = df.reset\_index()**
   * **Explanation:** df.reset\_index() resets the index of DataFrame df, converting the current index (which could be a column) into a new numbered index. After resetting the index, the previous index becomes a regular column in the DataFrame.
4. **Reindexing DataFrame:  
   df1.reindex(['b', 'c', 'd', 'a'])**
   * **Explanation:** df1.reindex(['b', 'c', 'd', 'a']) reindexes the rows of DataFrame df1 based on the specified index labels. This operation rearranges the rows of the DataFrame according to the order of index labels provided.
5. **Iterating Through DataFrame Rows:**for i, j in df1.iterrows():  
    print(j)
   * **Explanation:** df1.iterrows() iterates over the rows of DataFrame df1, yielding index labels and corresponding row data as Series objects. In each iteration, i represents the index label, and j represents the row data (as a Series object).
6. **Iterating Through DataFrame Columns:**for col\_name, column in df1.iteritems():  
    print(col\_name, column)
   * **Explanation:** df1.iteritems() iterates over the columns of DataFrame df1, yielding column names and corresponding column data as Series objects. In each iteration, col\_name represents the column name, and column represents the column data (as a Series object).
7. **Extracting Values of DataFrame Column as List:**list(df['a'])
   * **Explanation:** list(df['a']) converts the values in the column labeled 'a' of DataFrame df into a Python list. This operation extracts the values from the specified column and stores them as elements in a list data structure.
8. **Applying Function to DataFrame Column:**def test(x):  
    return x.sum()  
     
   df1.apply(test, axis=0)
   * **Explanation:** df1.apply(test, axis=0) applies the function test to each column of DataFrame df1, calculating the sum of values along the specified axis (0 for columns). The function test is applied element-wise to each column, and the result is returned as a Series where the index represents column names.
9. **Creating a Subset DataFrame:**df2 = df1[['a', 'b']]
   * **Explanation:** df1[['a', 'b']] creates a subset DataFrame df2 containing only the columns labeled 'a' and 'b' from the original DataFrame df1. This operation selects specific columns from the DataFrame to create a new DataFrame containing only the desired columns.
10. **Applying Element-wise Operation to DataFrame:**df2.applymap(lambda x: x \*\* 2)
    * **Explanation:** df2.applymap(lambda x: x \*\* 2) applies the provided lambda function element-wise to all elements of DataFrame df2. The lambda function squares each element of the DataFrame, producing a new DataFrame with the squared values.
11. **Sorting DataFrame by Column Values:**df.sort\_values('c')
    * **Explanation:** df.sort\_values('c') sorts the DataFrame df based on the values in the column labeled 'c' in ascending order. This operation rearranges the rows of the DataFrame such that the values in the 'c' column are sorted from smallest to largest.
12. **Sorting DataFrame by Index Values:**df.sort\_index(ascending=False)
    * **Explanation:** df.sort\_index(ascending=False) sorts the DataFrame df based on the index values in descending order. This operation rearranges the rows of the DataFrame such that the index values are sorted from highest to lowest.
13. **Using Rolling Mean Functionality:**df4['a'].rolling(window=3).mean()
    * **Explanation:** df4['a'].rolling(window=3) creates a rolling window of size 3 over the values in the column 'a' of DataFrame df4. .mean() calculates the mean (average) value within each rolling window. Rolling mean is useful for smoothing out fluctuations or noise in time series data and identifying underlying trends.
14. **Converting DataFrame Column to List:**list(df['a'])
    * **Explanation:** list(df['a']) converts the values in the column labeled 'a' of DataFrame df into a Python list. This operation extracts the values from the specified column and stores them as elements in a list data structure.
15. **Calculating Variance of a DataFrame Column:**variance = df['a'].var()
    * **Explanation:** df['a'].var() calculates the variance of the values in the column labeled 'a' of the DataFrame df. Variance is a measure of the spread or dispersion of a set of values. It quantifies how much the values in a dataset deviate from the mean.

These explanations provide insights into the functionality and applications of each code snippet. Let me know if you need further explanations or have any questions!

16. **Python Pandas - Date Functionality:**date = pd.date\_range(start='2023-04-23', end='2023-06-23')  
date  
df\_date = pd.DataFrame({'date': date})  
df\_date.dtypes  
df\_date

* + **Explanation:**
    - pd.date\_range(start='2023-04-23', end='2023-06-23') generates a range of dates starting from April 23, 2023, to June 23, 2023.
    - pd.DataFrame({'date': date}) creates a DataFrame df\_date with a single column 'date' containing the generated date range.
    - df\_date.dtypes displays the data types of columns in the DataFrame df\_date.
    - df\_date displays the DataFrame with the generated date range.

**17. Python Pandas –Time Delta:**pd.Timedelta(days=1, hours=5, minutes=45)  
dt = pd.to\_datetime('2023-06-20')  
td = pd.Timedelta(days=1)  
dt + td

* + **Explanation:**
    - pd.Timedelta(days=1, hours=5, minutes=45) creates a time delta object representing a duration of 1 day, 5 hours, and 45 minutes.
    - pd.to\_datetime('2023-06-20') converts the string '2023-06-20' to a datetime object dt.
    - pd.Timedelta(days=1) creates a time delta object representing a duration of 1 day.
    - dt + td adds the time delta td to the datetime object dt, resulting in a new datetime object representing the date and time one day ahead of '2023-06-20'.

Preparing data for further analysis, or assessing data variability, Pandas provides the tools you need to succeed in your data analysis endeavors. By leveraging these essential operations, analysts can unlock the full potential of their data and drive business success in today's data-driven world.