Pandas Notes in CDE

an open-source software library built on top of Python specifically for data manipulation and analysis, Pandas offers data structure and operations for powerful, flexible, and easy-to-use data analysis and manipulation.

# Making DF

You can make dataframe using pd.DataFrame(). These parentheses should include list, or 2d array or anything that can be changed into a dataframe.

# Head () & Tail ()

This method prints first five rows of the dataframe for information purpose. If you specify like df.head(3) so it’ll print the required rows. Tail prints the last 5 rows of dataframe.

# Column names and index:

While making a df, pandas will automatically take the first row as column names. But if you want to name the column differently you can give column parameter the names as list. Same can happen with the index. For example:

pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]], columns=["first", "second", "third"], index=["a","b","c"])

since the df is an object, so we can reach its attributes like columns and index. And we will get them as a list. The following code can be used for this purpose:

df.columns

df.columns.tolist()

the same code works for index.

# Info and describe:

To get general information about the df info() method can be used and for statistical information we can use describe() method.

# Shape and unique values:

Shape() will tell about the number of rows and columns in the dataframe and unique() returns an array of unique values in a Series or DataFrame column, while nunique() returns the count of unique values.

# Reading csv files:

For reading csv file and making df from it we can use read\_csv method. pd.read\_csv() automatically creates a DataFrame from the CSV file you provide. It reads the file and converts it into a pandas DataFrame object.

# Sample()

If you want to get random rows from dataframe so you can use sample() and input a number of rows you want to get In result.

# Loc()

Loc can help to get slicing or specific rows from dataframe. The following syntax can be used for slicing:

Df.loc(rows,columns)

If you ignore putting column or rows so it’ll consider taking data from all.

For single row df1.loc[10]

For range of rows df1.loc[5:10]

For range of rows with single column df1.loc[5:10, "Name"]

For range of rows with several columns df1.loc[5:10, ["Name","NOC"]]

For all rows but several columns df1.loc[:, ["Name","NOC"]]

# Iloc()

Its doing the same thing like loc() but the difference is that it needs index of columns instead of names.

df1.iloc[:,[1,3]]

# at and iat():

I you want to arrive at a specific value you can use at and Iat. Loc and iloc can also be used but they’re specifically for ranges instead of single values. At and iat are same as loc and iloc.

Targeting column:

If you want to target one specific column so you can do it in two ways:

Df.column\_name

Df[“column\_name”]

But if you’re targeting multiple column then only the second way can be used.

Sort\_Values():

Values of dataframe can be sorted by using the given function but one column name should be given as parameter so based on that folder the values can be sorted. If you wanna give several column names so you can give them in square brackets.

df1.sort\_values(["Age","ID"])

conditional targeting:

if you want to get data from specific columns with specific conditions you can use the following two ways:

df1[(df1["Age"]<=13) & (df1["Year"]>1999)][["Name","Year","NOC"]]

df1.loc[df1["Age"]<=15, ["Name","Age", "City","Sport","Year"]]

specific word in a column:

if you want to search all rows with a specific word in a column so you can use the following code:

df1[(df1["Name"].str.contains("Khan")) & (df1["NOC"].str.contains("PAK"))]

here I am search that gimme all columns where Name column contains Khan and noc column contains PAK.