CAPTER – 4

Basic Data Analysis

Learning Topics

- ✓ Display data
- ✓ Understanding data size
- ✓ Handling rows and indexes
- ✓ Handling columns and it's data types
- ✓ Observe null values
- ✓ Duplicate values.
- ✓ Access subset of data
- ✓ Boolean indexing

Understand the nature and characteristics of the data is the main step in Data Science or in Machine Learning. Some types of analysis techniques are available in pandas to understand the data. we will learn one by one in this chapter.

First load all the data as a pandas DataFrame to start the analysis. The below code loads all the csv file as pandas DataFrame. Now the analyzing data is in ***df*** (pandas DataFrame) variable.

```
import pandas as pd

df = pd.read_csv('File path')
df.head()
```

Display Data:

- There are three ways to display the data as follows.

Keyword / Function	*Use*
`print(df)`	*Display whole data*
`df.head(10)`	*Display staring rows*
`df.tail (10)`	*Display last rows*

- By default, head argument n = 5, so by default it displays first 5 rows. If we want to display specific number of rows then pass needful number.
- head() function can take negative values also. Suppose if n = -3 pass to head() then it will remove last 3 rows and display remaining rows.
- Vice versa tail. By default, tail also has n =5, so it displays last 5 rows.

Understanding data size:

- Size is the basic character of the data. We need to select the optimization coding techniques related to time and space based on the size information.

Keyword / Function	*Use*
`df.shape`	*Displays number of rows and columns.*
`df.size`	*Displays total number of labels / values*
`df.ndim`	*Displays data set dimension.*
`df.info()`	*Displays total information of the data*

Handling rows and indexes:

- Indexes of data is one of the important topics in the data handling. Here will learn about how to see the index numbers and how to set a particular column as index.

Keyword / Function	*Use*
`df.index`	*To know all dataset index numbers*
`df.set_index("clm_1" / 0)`	*To set a specific column value as index numbers.*
`df.drop(index=2, inplace=True)` `df.drop(index=[2,3,4], inplace=True)`	*Drop some specific rows based on index numbers.*

Handling columns and it's data types:

- Sometimes we need to change column names and data types of each column. In those cases, will use the below techniques.

Keyword / Function	*Use*
`df.columns`	*Provides all column names as series.*
`df.dtypes / df.info()`	*To know all columns data types*
`df['clm_1'].astype('dtype')`	*To change a particular column data type. (Column type casting)*
`df.astype({'clm_1': 'int', 'clm_2': 'str'})`	*Type caste the more than one column at a time.*
`df.drop(['clm_1', 'clm_2'], axis=1)` `df.drop(columns=['clm_1', 'clm_2'])`	*To drop the columns*
`df.info()`	*Displays all information of all columns.*

Observe null values:

- Understanding of null values is one of the important step in data preprocessing. The below functions will helpful to understand the null values structure and nature in data.

Keyword / Function	*Use*
`df.isna() / df.isnull()`	*Return the boolean values of data frame contains true where null values is present.*
`df.notna() / df.notnull()`	*Vise verse of* `isna()` *function*
`df.isna().sum() / df.notna().sum()`	*Return total number of null values in each column.*

Duplicate values:

- An important part of Data analysis is analyzing Duplicate Values and removing them. Pandas *duplicated()* method helps in analyzing duplicate values. It returns a Boolean Series denoting duplicate rows.

Syntax: `DataFrame.duplicated(subset=None, keep='first')`

subset: optional
- Only consider certain columns for identifying duplicates, by default use all of the columns.

Keep: {'first', 'last', False} : default 'first'
- first : Mark duplicates as True except for the first occurrence.
- last : Mark duplicates as True except for the last occurrence.

- False : Mark all duplicates as True.

Access subset of data

- Pandas provides 2 interesting functions ***iloc*** and ***loc*** for access subset of data.

Syntax: **`DataFrame.iloc`**

- Iloc[] is purely integer index based selection by position. It takes rose and column index numbers and returns subset of data based on those index numbers
- .iloc[] is primarily integer position based, but may also be used with a boolean array.
- ***Note*** : That contrary to usual python slices, the start included but the stop is excluded.

Keyword / Function	***Use***
`df.iloc[0]`	*Returns 0th row as pandas series.*
`df.iloc[[0]]`	*Returns 0th row as pandas data frame.*
`df.isna().sum() / df.notna().sum()`	*Return total number of null values in each column.*

`df.iloc[[rows_indexes], [clms_indexs]]`
 	
`Ex: df.iloc[[1,2], [0,5]]`
	Return data frame with mentioned row and column indexes,

`df.iloc[1:3, 0:3]`	*1st row to 2nd row and 0th column to 2nd column*
`df.iloc[:3, 0:3]`	*Starting row to 2nd row and 0th column to 2nd column.*
`df.iloc[10:, 0:3]`	*10th row to last row and 0th column to 2rd column*
`df.iloc[:, 0:3]`	*First row to last row and 0th column to 2nd column.*

Syntax: **`DataFrame.loc`**

- .loc[] is purely integer index based selection by position. It takes rose and column index numbers and returns subset of data based on those index numbers
- .loc[] is primarily label based, but may also be used with a Boolean array.
- .loc[] is very helpful when data set has string type of indexes.

- ***Note*** : that contrary to usual python slices, **both** the start and the stop are included.

Keyword / Function	***Use***
`df.loc[['str_1', 'str_2']]`	*Returns data frame with selected string indexed rows only*
`df.loc['str_1':'str_5', 'clm_2':'cls_5']`	*Returns data frame with selected string indexed rows and columns only*

Boolean indexing:

- Boolean indexing is the process of selecting the subset of data with boolean values based on some conditions.
- Comparison operators (`<, >, <=, >=, ==, ===, !=, ~`) are using to creating the boolean indexing.

  Ex: `df[df['column_name'] > value]`

- *`isin()`* function Is one of the way to make *boolean Indexing*. This function is used to check multiple value in a single column.

  Ex: `df[df['column_name'].isin([val_1, val_2, _ _ _, val_n])]`

- Applying boolean indexing on multiple columns is one of the scenarios for this requirement we need to separate each condition with parentheses "()".

  Ex: `df[(df['column_name'] > val ) & () | () & () ……]`