Slicing and Indexing

df[2:5] → provides all rows from 2 to 4

df[:] → provides all rows

df[['name','age']][2:5] → will provide all rows from 2 to 4 with columns name and age.

Sort dataframes

sort_values()

dataframe.sort_values('columnname', ascending = True)

Basic Operations

- max() →
  dataframe['column'].max()
- min() →
  dataframe['column'].min()
- mean() → provides the average salary of the dataframe
  dataframe.salary.mean()
- std() → provides the standard deviation.
  dataframe.age
- describe() → provides all the stats

Conditional Selecting:-

dataframe[dataframe.columnname > 2000000]

Output

all column which is more 2000000 salary

Date as the index

→ dataframe

- d = pd.read_csv(r'path\filename.csv')

  Output

  read data.

- call

  d → show all data.

- dataframe.set_index('column')

  Output

  select column represent index form.

- type(dataframe['columnname'][0])

  Output

  str.

- d = pd.read_csv(r'path\filename.csv', parse_dates=['date'])

  Output

  call (d) → all data

- type(dataframe['columnname'][0])

  Output

  Timestamp

Insert Missing Dates

→ variablename

ind = pd.date_range('starting date', 'ending date')

index = pd.DatetimeIndex(ind)

dataframe.reindex(index)

This range to delete the previous data and change newdate that means it is remove all black data and change to timestamp format.

Replace missing values in a dataframe.

- fillna()

```
new_df = data.fillna(0)
new_df = data.fillna({'temperature': 0,
                      'windspeed': 0, 'event': 'unknown'})
```

- fillna() → forward fill.

```
new_df = data.fillna(method='ffill')
```
→ It fill with previous row value.

- fillna() → backward fill.

```
new_df = data.fillna(method='bfill')
```
It fill with next row data

```
new_df = data.fillna(method='ffill', limit=1)
```

- interpolate()

```
new_df = data.interpolate()
new_df = data.interpolate(method='time')
```
It is mainly used time polation data, It fill middle data → (previous + last)/2 (average data).

Linear Interpolation

Linear interpolation is often used to approximate a value of some function f using two known values of that function at other points.

Drop rows having NaN values

dropna()

```
new1 = x.dropna()
```
(drop row having any NaN value)

~~call new1 → all data is remove but column is present because which data have any null value it remove~~

```
new1 = x.dropna(how='all')
```
~~→ represent all data (same)~~

drop row having all column value NaN.
(row में all column value NaN होगी तो)

dropna() with threshold value

new1 = x.dropna(thresh=2) → maintain all rows having at least 2 non NaN values.

Discovering Duplicates.

Duplicate rows are rows that have been registered more than one time. To discover duplicates, we can use the duplicated() method. The duplicated() method returns a Boolean values for each row.

e.g: Returns True for every row that is a duplicate, otherwise false:

print(df.duplicated())

Replace():

It can use in different ways

⊙ data.replace(-9999, np.NaN) → replace the value with NaN

⊙ data.replace([-57, 68], np.NaN) → Replace multiple value with NaN.

⊙ Replace values with specific to columns.

e.g→ data.replace({'temperature': -8888, 'windspeed': -9999, 'event': '0'}, np.NaN)

data.replace({'temperature': [-8888, -9999], 'windspeed': [-9999, -8888], event: '0'}, np.NaN)

Map data with replace()

data.replace({-9999: np.NaN, -8888: np.NaN, '0': 'sunny'})

⊙ Replace value with regex.

data.replace({'Temperature': '[A-Za-z]', 'windspeed': '[A-Za-z]', '', regex = True})

⊙ Replace list of values with another list of values.

data.replace(['Rain', 'Sunny', 'Snow'], [101, 102, 103])