PANDAS COMMANDS

import pandas as pd

For Importing The Data

1. pd.read_csv("filename") # From a CSV file
2. pd.read_table("filename") # From a delimited text file (like TSV)
3. pd.read_excel("filename", sheet_name= 'Sheet2') # From an Excel file
4. pd.read_sql(query, connection_object) # Reads from a SQL table/database
5. pd.read_json(json_string) # Reads from a JSON formatted string, URL or file
6. pd.read_html(url) # Extract all the tables from the given url (List view).
   pd.read_html(url)[0]/[1] # Extract the first/second table from the given url (Table view).
7. pd.read_clipboard() # Takes the content you copied and shows the result.
8. pd.DataFrame(dict) # From a dict, keys for columns names, values for data as lists
9. Insert Image in Jupyter Notebook.
   Convert the Cell to MarkDown > Edit Tab > Insert Image > Run
10. Insert Audio in Jupyter Notebook.
    from IPython.display import Audio
    file = '…file-path..'
    Audio(data = file, autoplay = False)
11. Insert/Embed a YouTube Video with link
    from IPython.display import YouTubeVideo
    YouTubeVideo("..video..url id…", height=400, width=400)
12. Insert/Embed a URL
    from IPython.display import IFrame
    IFrame("…url…", width=800, height=450)
13. Insert/Embed a PDF
    from IPython.display import IFrame
    IFrame("https://arxiv.org/pdf/1406.2661.pdf", width=800, height=450)

For Exploring The Data

1. **Attributes (Series)** - s.values, s.index, s.shape, s.size, s.nbytes, s.dtypes
   # To see the attributes of a series.
2. **s.head( ) / s.tail ( )** - s.head(), s.head(n) , s.tail() , s.tail(n)
   # To show the top or last elements of the series.
3. **s.nunique ( )** - s.nunique( ) # It shows the total no. of unique values in the series.
4. **s.unique ( )** - s.unique( ) # It shows the all unique values of the series.
5. **s.value_counts ( )** - s.value_counts( ) # It shows all unique values with their counts in the series.
   If **s.value_counts( )['value']** – It will show counts of this value only.
   If **s.value_counts(normalize=True)** – It will show the unique values in percentage.
   If **s.value_counts(dropna = False)** – It will show the Nan also.
6. **s.count( )** - s.count( ) # It shows the total no. of values in the series.
7. **s.nlargest ( ) / s.nsmallest ( )** - s.nlargest(4) , s.nsmallest()

# To show the n largest or n smallest values in descending or ascending order.

8. **s.mean( ), s.median( ), s.mode( ), s.std( ), s.min( ), s.max( ) -**
   # It shows the mean , median , mode, std. deviation, minimum value, maximum value of the series.
9. **s.skew( )** # It shows the Skewness of the series.
10. **s.describe()** **-** s.describe() # It shows the summary statistics (count, mean , median , std. deviation, minimum value, maximum value) of the series at once.
11. **s.quantile( )** **-** s.quantile(0.7) , s.quantile([0.7, 0.8, 0.3])
    # It shows the no. from the series where (given) percent of values falls below it.
12. **s.axes** # It returns the list of the labels of the series.
13. **s.empty** # It returns Boolean T/F whether the series is empty or not.
14. **Attributes (Data Frame)** **-** df.shape, df.size, df.index, df.columns, df.dtypes, df.values
    # To show the attributes of a DF.
15. **df.head( ) / df.tail ( )** **-** df.head(), df.head(n) , df.tail() , df.tail(n)
    # To show the top or last rows of the dataframe.
16. **df.nunique ( )** **-** df.nunique( ) # It shows the total no. of unique values in each column.
17. **df[‘Col_name’].value_counts ( )** **-** df.Col_name.value_counts( ) , sns.countplot(df.Col_name)
    # It shows all unique values with their counts in the column.
    If **df.Col_name.value_counts( )[‘Delhi ’]** – It will show counts of Delhi only.
    If **df.Col_name.value_counts(normalize=True)** – It will show the unique values in percentage.
    If **df.Col_name.value_counts(dropna = False)** – It will show the Nan also.
    df.Col_name.value_counts( ).plot(kind= ‘bar’) # To draw a graph of unique values of column.
    df.Col_name.value_counts( ).sort_index( ) # To sort with index after value counts.
    df.groupby(‘Col’)[‘Col’].count( ).sort_values(ascending=False) – Alternate method of value counts.
18. **df.count( )** **-** df.count( ), np.count_nonzero(df)
    # It counts the no. of non-null/null values of each column.
19. **df.count(axis=1)** # It counts the no. of non-null/null values of each row.
20. **df.axes** # It returns a list of row axis & column axis.
21. **df.empty** # It returns Boolean T/F whether the dataframe is empty or not.
22. **df.abs( )** # It shows the absolute values.
23. **df.info( )** **-** df.info( ) # It shows indexes, columns, data-types of each column, memory at once.
    df.info(memory_usage = ‘deep’) # It shows the info with actual memory size.
24. **df.sum( )** **-** df.sum( ), df.values.sum( ) # Shows sum of each numerical columns.
25. **df.sum(axis=1)** # Shows sum of each row.
26. **df.prod( ) , df.prod(1)** # Show the product of each numeric column & row.
27. **df.cumsum( ) , df.cumsum(1)** # Shows cumulative sum of each numerical columns & rows.
28. **df.cumprod( ) , df.cumprod(1)** # Show the cumulative product of each numeric column & row.
29. **df.mean( )** **-** df.mean( ), df.values.mean( ) # Shows mean of each numerical columns.
30. **df.mean(axis=1)** # Shows mean of all each row.
31. **df.std( )** **-** df.std( ) df.values.std( ) # Shows std of each numerical columns.
32. **df.std(axis=1)** # Shows std of each row.
33. **df.var( )** **-** df.var( ) , df.values.var( ) # Shows variance of each numerical columns.
34. **df.var(axis=1)** # Shows variance of each row.

35. **df.median( )** **-** df.median( ) , np.median(df) # Shows median of each numerical columns.
36. **df.median(axis=1)** # Shows median of each row.

37. **df.mode( ) , df.mode(axis=1)** # Shows mode of each numeric column & row.
38. **df.skew( ) , df.skew(axis=1)** # Shows the Skewness of each numeric column & row.
39. **df.describe( )** - # It produces the summary statistics of all numeric columns. It checks extreme outliers and large deviations etc.
40. **df.describe( )** - # For categorical dataframe, it will show a simple summary of unique values & most frequently occurring values.
41. **df.describe(include= ‘number’)** # For numeric columns.
42. **df.describe(include= ‘object’)** # For categorical columns.
43. **df.describe(include= ‘all’)** # To get summaries of all variables.
44. **df.describe( ).loc[‘min’ : ‘max’ , ‘Col_1’ : ‘Col_4’]** # To see only selected results.

For Selecting The Data

1. **s.loc / s.iloc** - **s**.loc[‘index’] , s.loc[‘index1’:’index2’] , s.iloc[0] , s.iloc[0:3]
   # Selection by object Index and Selection by integer Index.
2. **Series Indexing** - s[0] , s[‘index’] , s.index , s[0:5] , s[‘index1’:’index2’] , s[0:-1] , s[-3,-1] , s[::-1]
3. **Boolean Series Indexing** - s[s>=20] # Show a new series with boolean condition satisfied.

4. **df[‘Col_name’] , df.Col_name** # Selecting single column from the DF.
5. **df[[‘Col1’, ‘Col2’ , ‘Col3’]]** # Selecting multiple Columns from the DF.
6. **df.iloc[: , [1, 3, 5]]** # Selecting multiple Columns from the DF.

7. **df.loc[: , ‘Col1’ : ‘Col2’]** # Selecting columns with object slicing.
8. **df.iloc[: , 1:4]** # Selecting columns with integer slicing.

9. **df.loc[‘Row_index’]** # Selecting single row from the DF.
10. **df.loc[[‘index1’ , ‘index2’ , ‘index3’]]** # Selecting multiple rows from the DF.
11. **df.iloc[[1, 3 , 5]]** # Selecting multiple rows from the DF.

12. **df[‘index1’ : ‘index2’]** - df[‘P’ : ‘R’] # Selecting rows with object slicing.
13. **df[index1 : index2]** - df[1:3] # Selecting rows with integer slicing.

14. **df.loc[‘index1’ : ‘index2’ , ‘Col1’ : ‘Col2’]** # Selecting rows & columns with slicing using loc( ).
15. **df.iloc[1:3 , 1:3]** # Selecting rows & columns with slicing using iloc ( ).

16. **df.loc[‘row_label’ , ‘col_label’]** # Selecting one row and one column by label.
17. **df.iloc[‘row_index’ , ‘col_index’]** # Selecting one row and one column by index.
18. **df.iloc[[2,4,6],[2,4,6]]** # Selecting multiple rows & multiple columns.
19. **df>3 , df[df>3]** # Showing the elements of the DF which are greater than 3.
20. **data.loc["2012-01-06", 'Stn Press (kPa)'][2:4]**
21. **Selecting columns by data-type** - df.select_dtypes(include = 'number' / ‘int’ / ‘float’ / ‘object’)
    df.select_dtypes(include = [‘object’ , ‘number’ , ‘category’ , ‘datetime’])
    df.select_dtypes(exclude = 'number' / ‘int’ / ‘float’ / ‘object’)

Adding / Removing

1. **Series** **-** # To create a series. (Homogeneous Data, Size Immutable, Values Mutable)
   pd.Series(data, index=, dtype=, name=)
   Series([1,2,3,4], index=list('abcd'), name='new') ,
   pd.Series(np.random.random(3), index =['delhi','mumbai','agra']),
   Series({'a':1, 2:'b', 'c':'ram'}),
   pd.Series(data) , data=[[1,2,3],[4,5,6]].
2. **DataFrame** **-** # To create a dataframe. (Hetrogenous Data , Size Mutable, Data Mutable)
   pd.DataFrame(data=, index=, columns=) ,
   pd.DataFrame(np.arange(1,10).reshape(3,3), index=['a','b','c'], columns = list('XYZ')) ,
   pd.DataFrame({'A':['a','b'], 'B':list('cd'), 'C':[1,2], 'D':list('34') })
   pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]])
   pd.DataFrame(np.random.rand(4,5))
3. **Adding a new DF to existing** **-** df2 = df2.append(df1)
   # It will append the rows of df1 at the end of df2.
4. **Reindexing a DataFrame - df_r = df1.reindex(index=[0,2,5,7] , columns = ['A', 'C' , 'B'])**
   # It will create a new dataframe with the mentioned indexes only of df1.
   df2.reindex_like(df1, method = 'ffill' , limit = 1)
5. **Adding New Row/Index** # To add a new row in the series
   s.loc['new index'] ,
   s.loc[6]='rohit' ,
   s.loc['new']=39
6. **Removing Row/Index** # To remove a row from the series.
   s.drop('index') ,
   s.drop(6) ,
   s.drop('new')
7. **Adding New Column** **-** # To add new column in the DF.
   df['New_col']= ,
   df['C'] = list('qwerty') ,
   df.insert(2, 'C', [1,2,3,4,5])
   df.insert(index , 'new_column_name', new_column_values) - To insert a New column at a particular position with values in it.
8. **Adding New Row** **-** # To add new row in the DF.
   df.loc['New_row']= ,
   df.loc['R'] = list('12345') ,
   df.loc['R' , 2:5] = 78

9. **Assign ( ) -** # It is used for creating new variables on the fly , or for deriving new column from existing ones.
   df.assign(H = lambda x:x['C']+2) ,

df.assign(I=21 , J = list(‘qwerty’))

10. **Removing Rows** -
df.drop(‘index_name’) ,
df.drop(index_value) ,
df.drop(‘index_name’, axis=0, inplace=True),
df.drop([‘Row1’ , ‘Row2’ , ‘Row3’]) , df.drop([1,2,4]).

11. **Removing Columns** -
df.drop(‘Col_name’ , axis=1) ,
del df[‘Col_name’] ,
df.drop([‘Col1’ , ‘Col2’, ‘Col3’], axis=1)
df.pop(‘Col_name’)

12. **Creating missing values** -
df.loc[‘Row_index’] = np.nan ,
df[‘Col_name’] = None
13. **pd.merge(df1, df2, on=’Col_Name’ , how=’inner/outer/left/right’)**
# Merge ( ) - Column names must be same. Default – Inner Join.
14. **pd.merge(df1, df2, left_on=’df1_Col_Name’ , right_on=’df2_Col_Name’, how=’inner/outer’)**
    - inner - works on common indexes/columns/elements only , outer - works on all indexes/columns/elements
    - left - works on left DF , right - works on right DF.
15. **df1.join(df2, how = ‘inner/outer/left/right’) , df1.join([df2,df3])**
# Join ( ) - Indexes may or may not be same. Column names must be different. Default - Left join.
16. **pd.concat([df1,df2] , axis=0/1 , join=’inner/outer’)**
# DF Concat - In concat(with rows axis) - Rows below Rows without merge/sort and Columns will merge/sort . In concat(with columns axis1) - Columns side by side without merge/sort and Rows will merge/sort.
17. **pd.concat([S1,S2,S3], axis=1, join=’outer/inner’)** # Series Concat.

,

For Cleaning The Data

18. **.astype()** - s.astype(int), s.astype(float), s.astype(str), s.astype(‘category’)
# Converting the data type of the series to a new data type.
19. **s.replace( )** - s.replace({1:‘one’ , ‘b’:‘bombay’, np.nan:s.mean( )})
# To replace any(or missing) data of the series with a new value using dictionary format.
20. **s.duplicated ( )** - s.duplicated( ) , s[s.duplicated( )]
# To detect the duplicate values in the series.
21. **s.drop_duplicates( )** - s.drop_duplicates( ) , s.drop_duplicates(inplace=True)
# To remove the duplicate values from the series.
22. **s.isnull( )** - s.isnull( ), s[s.isnull( )]

# It detects the missing values from the series.

23. s.notnull( ) - s.notnull( ) , s[s.notnull( )]
# It detects the existing (non-missing) values from the series.

24. s.fillna( ) - s.fillna(4) , s.fillna(s.mean( ))
# It fills all the missing values with a given number.

25. s.ffill( ) - s.ffill( ) # It fills the missing values using forward fill method.

26. s.bfill ( ) - s.bfill ( ) # It fills the missing values using backward fill method.

27. s.dropna( ) - s.dropna( ) # It removes all the missing values.

28. df[‘Col_name’].astype() , df.astype({‘Col1’ : float , ‘Col2’ : int}) -
df.Col_name.astype(int / float / str) # Converting the data type of any column to a new data type.

29. df.Col_name.replace( ) - df.Col.replace({1:‘one’ , ‘b’:‘bombay’, np.nan:s.mean( )}) }, inplace = True)
# To replace any(or missing) item of the series with a new value using dictionary format.

30. df.rename ( ) - # For renaming rows.
df.rename(index = { ‘old1’ : ‘new1’ , ‘old2’ : ‘new2’ }),
df.rename(columns = { ‘old1’ : ‘new1’ , ‘old2’ : ’new2’ }),
df.columns = [‘New1’ , ‘New2’ , ‘New3] # For renaming all columns at once or setting headers.
df.add_prefix(‘R_’) , df.add_suffix(‘_S’) # To add prefix and suffix to the column names.

31. df.isnull( ) - df.isnull( ) , df.isnull( ).sum( ) , df.isna( ).sum( )
# It detects the missing values from the dataframe.

32. df[df.Col_name.isnull( )] # It detects the missing values from a column of the dataframe.

33. sns.heatmap(data.isnull()) - plt.figure(figsize=(8,8)) , sns.heatmap(data.isnull())
# It will show the all columns & missing values in them in heat map form.

34. df.columns[df.isna( ).any( )] - df.columns[df.isna( ).all( )]
# To show the column names which contains Any or All null values.

35. x_inputs.columns[x_inputs.isna().any()]

36. df[df.isna( ).any(axis=1)] - df[df.isna( ).all(axis=1)]
# To show the records which contains Any or All null values.

37. df.notnull ( ) - df.notnull( ) , df.notnull( ).sum( ) , data.notna( ).sum( )
# It detects the existing (non-missing) values from the dataframe.

38. df[df.Col_name.notnull( )] #It detects the existing (non-missing) values from a column of the dataframe.

39. df[df.notna( ).any(axis=1)] - df[df.notna( ).all(axis=1)]
# To show the records which doesn’t contains Any or All null values.

40. df.fillna( ) - df.fillna(3) , df.fillna(3, inplace=True) , df[‘Col’].fillna(df[‘Col’].mean( )) ,
df.fillna(method = ‘pad/ffill/bfill/backfill’) # It fills all the missing values with a given number.

41. df.ffill( ) - df.ffill( ) , df[‘Col_name’].ffill( )
# It fills the missing values column wise using forward fill method.

42. df.ffill(axis=1) # It fills the missing values row wise using forward fill method.

43. df.bfill( ) - df.bfill( ) , df[‘Col_name’].ffill( )
# It fills the missing values column wise using backward fill method.

44. df.bfill(axis=1) # It fills the missing values row wise using backward fill method.

45. df.dropna( ) - df.dropna( ), df.dropna(how= ‘all’) , df.dropna(how= ‘any’)
# It drops the rows that contains all or any missing values.

46. df.dropna(axis=1) - # It drops the columns that contains all or any missing values.
df.dropna(axis=1) , df.dropna(how=’all’, axis=1) , df.dropna(how=’any’, axis=1).

47. **df.dropna(subset=[‘Col1’, ‘Col2’])** #It drops the rows which contains missing values in Col1 or Col2.
48. **df.dropna(axis=1,thresh=n)** # Drops all rows which have less than n non null values.
49. **dr.dropna(thresh = len(data)*0.9, axis= ‘columns’)** # To drop the columns having more than 10% missing values.
50. **df.Col.replace(np.nan, col.mean)** # To replace any missing value in the column.
51. **df.duplicated( )** - df.duplicated( ) , df[df.duplicated( )] # It checks row wise and detects the duplicate rows.
52. **df.loc[‘R’].duplicated( )** # It detects the duplicate values in R row.
53. **df.drop_duplicates( )** - # It ignores duplicate rows.
54. **df[(df != 0).all(1)]** - # Deleting rows that contains nothing.
55. **df[‘Col_name’].str.strip(‘$')** - # Delete a sign from the values of a string colum.

For Analyzing TheData

1. **s.isin( )** - s.isin([1,’ram’]) , s[s.isin([1,’ram’])] # It checks whether values are contained in the series or not.
2. **s.sort_values( )** - s.sort_values( ) , a.sort_values(ascending=False) # It sorts a series by values.
3. **s.sort_index( )** - s.sort_index( ), a.sort_index(inplace=True) # It sorts a series by index.
4. **df.sort_index( )** - # It sorts the entire dataframe by index names.
5. **df.sort_index(axis=1)** # It sorts the entire dataframe by column names.
6. **df.sort_values( )** # Sort the entire dataframe by the values of the given column.
   df.sort_values(by=‘Col’, kind= ‘mergesort/quicksort/heapsort’) , df.sort_values(by= [‘Col1’, ‘Col2’]).
7. **df.sort_values(axis=1)** df.sort_values(by=‘Row_name’, axis=1) # Sort the entire dataframe by the values of the given row.
8. **df.Col_name.sort_values( )** # Sort the values of a single column only. # Sort the dataframe by Col1 in ascending and by Col2 in descending order.
9. **df.sort_values(by= [‘Col1’, ‘Col2’] , ascending=[1,0])** # Sort the dataframe by Col1 in ascending and by Col2 in descending order.
10. **df.pivot(index= ‘Col_1’ , columns= ‘Col_2’ , values=’Col_3’)** , # It returns a reshaped DF organized by given index/columns values.
    df.pivot(‘Col1’, ‘Col2’, ‘Col3’) ,
    df.pivot(‘Col1’ , ‘Col2’) ,
    df.pivot(‘Col1’, ‘Col2’, values=[‘Col1’, ‘Col2’])
11. **df.pivot_table(values= ‘Col1’ , index= ‘Col2’ , columns= ‘Col3’)** , # It creates a spreadsheet style pivot table as a DF. By default, it shows Mean of values.
    df.pivot_table(‘Col1’, ‘Col2’ , ‘Col3’, aggfunc = ‘sum/mean’, margins = True) ,
    pd.pivot_table(df,‘Col1’,‘Col2’,’Col3’)
12. **df.melt(df , id_vars=‘Col1’ , value_vars=‘Col2’ , var_name=‘var’ , value_name=‘value’)** # It unpivot a dataframe.
13. **df.groupby(‘Col_name’).groups** # To create groups and view the created groups. # GroupBy - One Key – Groups formed of all unique values of the Column.

Groupby is used to split the data into groups based on some criteria.
df.groupby(‘Col_name’).first(),
df.groupby(‘Col_name’).last() ,
df.groupby(‘Col_name’).mean() ,
df.groupby(‘Col_name’).sum() ,
df.groupby(‘Col_name’).max() ,
df.groupby(‘Col_name’).min()
df.groupby(‘Col_name’, as_index=False).mean() # To show the index as 0,1,2,3

14. **df.groupby(‘Col_1’)[‘Col_2’] .value_counts( )** , **df.groupby(‘Col_1’)[‘Col_2’] .sum( )[‘value’]** ,
# GroupBy – Two Keys – Apply on Col_2 grouped by Col_1
df.groupby(‘Col_1’)[‘Col_2’,’Col_3’].max( )
df.groupby([‘Col_1’,‘Col_2’]).Col_3.value_counts( )
15. **df.groupby(‘Col_name’).get_group(‘Element’)** ,
# Get group is used to find the entries contained in any of the group.
df.groupby([‘Col_1’, ‘Col_2’]).get_group((‘Element1’,’Element2’))
It will show all the entries where Element1 is in Col1 and Element2 is in Col2.
16. **df.groupby(‘Col_name’).agg([‘max’,’min])** # Apply more than 1 function on selected columns.
df.groupby(‘Col1’)[‘Col2’,’Col3’].agg([‘max’,’min])
17. **df.groupby(‘Col_1’).agg([(‘new_max’, ‘max’), (‘new_min’, ‘min’)])**
# Providing new names to the applied functions.
18. **df.groupby(‘Col_1’)[‘Col2’,’Col3’].agg({‘Col2’:‘max’ , ‘Col3’:[‘min’, ‘sum’]})**
# Applying different function to different columns.
19. **df[df[‘Col_name’]==df[‘Col_name’].max( )/min( )]** # Accessing the row which has the maximum/minimum value in the given column.
20. **df[df.Col_1 = = ‘Element1’]**
# Filtering – We are accessing all records with Element1 only of Col_1.
21. **df[df.Col_1 = = ‘Element1’][‘Col_2’]**
# We are accessing all records of Col_2 only with Element1 only of the Col_1.
22. **df[df.Col_1 = = ‘Element1’][[‘Col_2’, ‘Col_3’]]**
23. **df.loc[condition]** – df.loc[df.Col_name > 1000] # To show all records with a particular condition.
24. **df.query(‘condition’)** - df.query(‘Col_name > 1000’) # To show the records for a particular query.
25. **df[df.Col_1 = = ‘Element1’].max( )/count( )/sum( )**
26. **df[df.Col1 = = ‘Element1’].Col2.value_counts( )** , **df[df.Col1 = = ‘Element1’].Col2.max( ) / sum ( )**
# From Col1 selecting rows with element1 & show result of Col2.
27. **df[(df.Col1 = = ‘Element1’) | (df.Col2 < ‘Element2’)]**
# OR Filter – Filtering the data with two or more items.
28. **df[(df.Col1 = = ‘Element1’) & (df.Col2 > ‘Element2’)]**
# And Filter – Filtering the data with two & more items.
29. **df[(df.Col1 = = ‘Element1’) & (df.Col2 = = ‘Element2’) | (df.Col3 = = ‘Element3’)]**
# Applying And & OR filter at once.
30. **isin( ) -** # To show all records including /excluding particular elements.
df[df.Col_Name.isin([‘Element1’ , ‘Element2’, ‘Element3’])] , df[~df.Col.isin([‘E1’ , ‘E2’, ‘El3’])]
31. **len( ) -** To check the length of anything.

32. **Graph from Pandas directly :**
    df.plot(x = ‘Year’, y = ‘Sales’ , kind = “ line/scatter/box/area/stack/pie/bar”, figsize = (25,4), color=['red', 'black', 'green', 'yellow', 'orange']).
    df.Col_name.plot(style='*-' , figsize = (25,4)
    # Pandas can make graphs by calling plot directly from the DF (using df.plot( )). Plots can be called by defining plot kinds

For Saving/Writing The Data

1. df.to_csv(filename, index=True) # Writes to a CSV file , with indexes
2. df.to_excel(filename) # Writes to an Excel file
3. df.to_sql(table_name, connection_object) # Writes to a SQL table
4. df.to_json(orient=’records’, lines=True) # Writes to a file in JSON format
5. df.to_html(filename) # Saves as an HTML table
6. df.to_clipboard() # Writes to the clipboard

Date-Time

1. **datetime** - import datetime, x = datetime.datetime.now( ) / today( )
2. **calendar** - import calendar , x = calendar.month(Year, Month)
3. **To slice hour from Time column** - DF[‘Time_Col’].str.slice(-5,-3).astype(int)
4. **to_datetime ( )** - pd.to_datetime(DF.Date_Time_Col)
   # Converts the data-type of Date-Time Column into datetime[ns] datatype.
5. **To set a date →** a = datetime.date(2020, 5, 12) , b = datetime.date(1993,12,25)
6. **To see difference between two dates →**a - b
7. **To compare Date-Time →** if a = = b ; if a > b ; if a < b
8. **For Unix Epoch Time -** pd.to_date_time(DF.Date_time_col, unit = ‘s’)
9. **From the Date-Time column, showing only hour, minute, month, weekdays -**
   df[‘Time_Col’].dt.hour ,
   df[‘Time_Col’].dt.minute ,
   df[‘Time_Col’].dt.month , df[‘Time_Col’].dt.day
   df[‘Time_Col’].dt.weekday_name, df[‘Time_Col’].dt.dayofweek, df[‘Time_Col’].dt.weekofyear
10. df['Time_Col'].apply(lambda x:x.strftime("%A or %a")) – To show day name
11. df[Time_Col"].apply(lambda x:x.strftime("%B or %b")) – To show month name
12. df['Time_Col'].apply(lambda x:x.strftime("%Y or %y")) – To show year only
13. df['Time_Col'].apply(lambda x:x.strftime("%m")) – To show month only
14. df[Time_Col"].apply(lambda x:x.strftime("%A-%b-%y")) – To show day name, month name, year
15. **df[‘Hours’] =** df.Time_Col.dt.hour # Creating a new column with only hours values.

16. **df[‘Month’] =** df.Time_Col.dt.month # Creating a new column with only month values.
17. **df[‘Year’] =** df.Time_Col.dt.year. # Creating a new column with only year values.
18. **timestamp ( ) -** x = pd.to_datetime(‘2020-12-25 04:00:00’) , df.loc[DF.Time <=x , :].
    # Setting the given date-time as a fix value.
19. **df[‘Time_Col’].dt.year.value_counts( )**
    # It counts the occurrence of all individual years in Time column.
20. **df[‘Time_Col’].dt.month.value_counts( ).sort_index( )**
    # It counts the occurrence of all individual months in Time column in ascending order.
21. **df.dat_time_col.max( ) / min( ) -** # To show the maximum or minimum date.
22. **pd.date_range(start='01-01-2018' , end = ‘01-01-2019’ , periods = 10, freq='A/Q/M/W/D/H/T/S/L/U/N/B/SM/BM/MS/SMS/BMS/BQ/QS/BQS/BA/BAS/BH')**
    # To generate a date series. It returns a fixed frequency DatetimeIndex.
23. **pd.bdate_range('05-07-2020', periods = 10) #** It creates business dates. Exc. Saturdays & Sundays.
24. **Timedeltas -** A timedelta object represents a duration, the difference between two dates or times.
    pd.Timedelta('2 days 2 hours 15 minutes 30 seconds') , pd.Timedelta(6 , unit='T') ,
    pd.Timedelta(minutes=7), datetime.timedelta(days=4)
25. **datetime.datetime.today( ).**year/month/day/strftime(‘%B’)/strftime(‘%A’) # To show today’s year , month, day, Month name, Day name.
26. **Parse Dates -** import datetime , import dateutil.parser
    x = '25th Dec 2020' , parse_date = dateutil.parser.parse(x) ,
    op_date = datetime.datetime.strftime(parse_date, '%d-%m-%y')
27. **To combine Date & Time Column –**
    pd.read_csv(‘..data..’, **parse_dates = True/[‘Times’/‘Dates’] , index_col='Date/Time'**)
    pd.read_csv(‘..data..’, **parse_dates = [‘Date-column’]**)

Others

1. **Clip Lower** # All values that are less than threshold value become equal to it.
   Df[‘Col_Name’].clip(lower=value) ,
   Df[‘Col_Name’].clip(lower=[Val1,Val2,Val3])
2. **Clip Upper** # All values that are more than threshold value become equal to it.
   Df[‘Col_Name’].clip(upper=value),
   Df[‘Col_Name’].clip(upper=[Val1,Val2,Val3])
3. **df.unstack ( )** # Converts Rows into Columns (long to wide) Ex : Reshape a MultiIndexed series.
4. **df.stack( )** # Converts Columns into Rows (wide to long).
5. **Dummies - df[‘Col_name ‘]= = ‘a’** # Creates dummy for level ‘a’ in True & False format.
6. **(df[‘Col_name’] = = ‘b’).astype(int)** # Creates dummy for level ‘b’ in 0 & 1 format.
7. **pd.get_dummies( ) -**
   # This function takes as input a categorical variable (column) for supplying names to created variables.
   pd.get_dummies(df[‘Col_name’]) ,
   pd.get_dummies(df.Col_name , prefix=’dummy’)
8. **Dummy Variable Trap -**
   df.join(pd.get_dummies(df['Col_name'])).drop('Col_name', axis=1).drop('1dummy_col', axis=1).

9. **df[‘Col_Name’].apply(lambda x:x+2)** #Apply( ) – To apply a function along on any axis of DF.

10. **df.applymap(lambda x:x.upper( ))** # Applymap ( ) – Apply a function to each element of the DF.

11. **df.pipe(lambda x:x+10) , df.Col.pipe(fun)** # Apply a function to each element of the DF or column.

12. **df[‘Col_Name’].map({‘Y’:’Yes’ , ‘N’:’No’})** #Map( )–Change the all values of a column from old to new. We have to write for all values of column otherwise Nan will appear.

13. **map( )** - s.map(lambda x:x+20)

14. **df.set_index(‘Col_Name’) , df.index = df.Col_name**
# Set index - To set any column of a DF as an index. df.set_index([‘Col1’, ‘Col2’])

15. **df.reset_index( )** # To convert the index of a Series into a column to form a DataFrame.

16. **Series to List/Dictionary** - s.tolist() , s.to_dict() # Converting a Series into list or dictionary.

17. **df[‘Col_name'] + 10** - It will add 10 to all values of the given column.

18. **df[‘Col’] * df[‘Col2’]** - It will multiply values of column 1 to the values of column 2.

19. def times2(value): - **Apply a function ‘times2’ to the head only of the column of dataframe.**
```
def times2(value):
    return value * 2
```
df["Col_name"].apply(times2).head()

20. **Partial Matches** - df["New_Col"] = df.Col_name.str.contains('Value_to_match’) ,
df.Col_name.str.lower( ).str.contains(‘Value’).

21. df[‘bhk’] **= df[‘size’].apply(lambda x : int(x.split(‘ ’)[0]))** – To split the entries of a column.
(Also selecting the first value only before separator with converting it into integer type).
Use [1] , [2] for second & third value after separator.
For further split : df[‘Col_name’].apply(lambda x : int(x.split(‘ ’)[0] .split(' - ')[0]))
For adding ( ) also : df[‘Col_name’].apply(lambda x : ‘(’ + x.split(‘ ’)[0]) + ‘)’)

22. **Defining a function to convert 10-30 into average i.e, 20**
```
def convert_hypen-values_to_avg_num(x):
    tokens = x.split(‘ - ‘)
    if len(tokens) == 2:
        return (float(tokens[0]) + float(tokens[1])) / 2
    try:
        return float(x)
    except:
        return None
```

23. **To remove the records that matches any condition**
df [~(condition)] → df [~(df.col1 / df.col2 < 200)] or df [~(df['Col_name'] = = 'Value')]

24. **pd.to_numeric(df.Col_name , errors = ‘coerce’)** - # To convert the argument into numeric.
df.**apply(pd.to_numeric , errors = ‘coerce’).fillna(0)** # To apply on whole dataframe at once.

25. **To see the month/year wise sales (or any thing)** → X = df.groupby(‘month/year_col’)[‘Sales’].sum()

26. **To draw the month wise sales on graph –**
months = range(1,13) , plt.bar(months , X) or plt.bar(df.Col.unique( ) , X)

27. **Query –** df.query(‘condition’) # To show the records for a particular query.

28. **Sample - df.sample(3) , df[[‘Col1’, ‘Col2’]].sample(5)** # It shows some sample of items.

29. **Col.str.lower( ) , Col.str.upper( ) -** # To convert the items of a column into lower or upper case.

30. **df.Cat_Col.str.len( )** # It shows the length of each word in the categorical column.

31. **df.Col.str.strip( )** # It strips the white space from both sides of each word of the column.

32. **df.Col.str.cat(sep= ‘_’)** # It concatenates the all elements of a column with the given separator.

33. **df.Col.str.get_dummies( )** # It return the dataframe with one-hot encoding.
34. **df.Col.str.split(‘ ’)** # It splits a string into columns.
    df[[‘Col_1’, ‘Col_2’]] = df.Col_name.str.split(‘ ’, expand = True)
35. **df.Col.str.replace(‘a’, ‘b’)** # It replace ‘a’ of a string of the entire column by ‘b’.
    data.Col_name.str.replace(‘ ’ , ‘and’) , or data.Col_name.str.replace(‘$’ , ‘#’)
36. **df.Col.str.repeat(2)** # It repeats each element of the column 2 times.
37. **df.Col.str.count(‘A’)** # It returns counts of ‘A’ in each element of the column.
38. **df.Col.str.startswith(‘A’)** # It returns True/False if the element starts with A.
39. **df.Col.str.endswith(‘A’)** # It returns True/False if the element ends with A.
40. **df.Col.str.find(‘S’)** # It returns the first position of first occurrence of ‘S’.
41. **df.Col.str.findall(‘S’)** # It returns a list of all occurrence of the ‘S’.
42. **df.Col.str.swapcase( )** # It swaps the lower case to upper and upper case to lower of each element.
43. **df.Col.str.islower( )** # It checks all characters of each element of the column are in lower case or not.
44. **df.Col.str.isupper( )** # It checks all characters of each element of the column are in upper case or not.
45. **df.Col.str.isnumeric( )** # It checks all the characters of each element are numeric.
46. **ast.literal_eval(df.Col[0]) , df.Col.apply(str)**
    # Applying the function to 0 index and full column. It converts the string containing lists into list only.
47. **df.loc[::-1]** # To reverse the order of rows.
48. **df.loc[::-1].reset_index(drop=True)** # To reset the reverse order of rows.
49. **df.loc[: , ::-1]** # To reverse the order of columns.

50. **Build a DataFrame from multiple files -** 1=data/report1 , 2= data/report2 , 3= data/report3
    from glob import glob
    new_df = sorted(glob (‘data/report*.csv’)
    # Row – wise → pd.concat((pd.read_csv(file) for file in new_df) , ignore_index = True)
    # Column - wise → pd.concat((pd.read_csv(file) for file in new_df) , axis = ‘columns’)

51. **Split a Dataframe into random sets –**
    df1 = df.sample (frac = 0.75 , random_state = 1234) , df2 = df.drop (df1.index)
52. **Expand a column, which contains items in the form of a list , into a dataframe :**
    new_df = df.Col_name.apply(pd.Series) , pd.concat([df , new_df] , axis = ‘columns’)
53. **To combine the output of an aggregation function with dataframe :**
    df[‘new_col’] = df.groupby(‘Col_1’).Col_2.transform(‘sum’)
    Ex : df[‘total_order_price’] = df.groupby(‘Order_id’).item_price.transform(‘sum’)
54. **To filter out the records from the dataframe using Filter function.**
    df.groupby('name').filter(lambda x : len(x) > 4) # It shows records of those whose name occurs > 4 in df.
55. **Convert Numeric Data into Categorical Data of a column:**
    pd.cut(df.Col_name , bins = [1,3,6,9,12] , labels = [‘A’ , ‘B’ , ‘C’ , ‘D’])
56. **Change the display of the column :** pd.set_option('display.float_format' , '{:.2f}'.format)
    # It changes the values of all numeric columns of the dataframe to 2 point decimal.
    pd.reset_option('display.float_format') # To reset the changes made.
57. **pd.get_option(“display.max_rows / display.max_columns”)** → 60 / 20 default, it takes one argument.
58. **pd.set_option(“display.max_rows , 10 / display.max_columns , 5”)** # It takes two arguments.
59. **pd.reset_option(“display.max_rows / display.max_columns”)** # It resets the value of arguments.
60. **pd.describe_option(“display.max_rows / display.max_columns”)** # It describe about the argument.

61. **s.pct_change( ) , df.pct_change( ) , df.pct_change(axis=1)**
# It compare every element with prior element and computes the change percentage.

62. **df.Col_1.cov(df.Col_2)** # To check the covariance between two columns. NA excluded automatically.

63. **df.cov( )** # It shows the covariance of each column with other column of the dataframe. Non-numeric columns excluded automatically.

64. **df.Col_1.corr(df.Col_2) , df[['Col_1', 'Col_2']].corr( )** # To check the correlation between two columns. NA excluded automatically.

65. **df.corr( )** # It shows the coorelation of each column with other column of the dataframe. Non-numeric columns excluded automatically.

66. **Style The DataFrame**

A. df.style.set_caption('Description of the dataframe') # To give a caption to the dataframe.
B. df.style.hide_index( ) # To hide the index of the dataframe.
C. df.style.format({'Date' : '{:%m/%d/%y}'}) # To change the format of date-time column.
D. df.style.format({'Col_name' : '${:}*'}) # To put $ sign in front & * in end of each item of column.
E. data.style.format({'Col_name' : '{:,}'}) # To put , in each value of a numeric column.
F. df.style.highlight_min('Col_name' , color = 'red') # To highlight the minimum value of a numeric column.
G. df.style.highlight_max('Col_name' , color = 'green') # To highlight the maximum value of a numeric column.
H. df.style.background_gradient(subset = 'Col_name' , cmap='Blues') # To fill a numeric column with color gradient.
I. df.style.bar('Col_name' , color = 'green' , align = 'zero') # To show the values of a numeric column with colored bar.

67. **DataFrame Profiling –**
conda install -c anaconda pandas-profiling # install the pandas-profiling package
import pandas_profiling
pandas_profiling.ProfileReport(df)

68. To show installed versions : **pd.__version__ , pd.show_versions( )**

69. **df.transpose( )** # It converts the rows into columns and columns into rows of the dataframe.

70. **Iteration in Pandas :**

- Iteration on Series produces Value.

```
for value in df.Col_name:
      print (value)
```

- Iteration on DataFrame produces Column Names.

```
for col in df:
      print (col)
```

To iterate over the rows of the dataframe, we can use:
A. iteritems ( ) – for key,values in df.iteritems( ) : print (key,value)
B. iterrows( ) – for row_index, row in df.iterrows( ) : print (row_index , row)
C. itertuples( ) – for row in df.itertuples( ) : print (row)

71. df.Col_name.rank(method= ‘average/min/max/first/dense’ , na_option = ‘keep/top/bottom’)
# It computes the rank of numerical data (1 through n) along axis.

72. df.rolling(window=3).mean( ) , df.Col_name(window=2).min( )
# Window is the the number of observation used for calculating the statistics.

73. df.expanding(min_periods = 2).mean( ) , df.Col_name(min_periods=3).mean( )
# Min_periods is the minimum number of observations in window required to have a value.

74. df.ewm(com/span/halflife/alpha=)mean( ) , df.Col_name(com/span/halflife/alpha=).mean( )
It assigns the weight exponentially.

75. Aggregations Methods

A. df.agg(np.sum) # Applying aggregation on whole data frame.
B. df.Col_name.agg(np.sum) # Applying aggregation on one column of the data frame.
C. df[[‘Col1’, ‘Col2’]].agg(np.sum) # Applying aggregation on multiple columns of the data frame.
D. df.Col_name.agg([np.mean , np.sum]) # Applying multiple functions on a single column.
E. df[[‘Col1’, ‘Col2’]].agg([np.mean , np.sum]) # Applying multiple functions on multiple columns.
F. df.agg({‘Col1’ : np.sum , ‘Col2’ : np.mean}) # Applying different functions on different columns.

76. pd.Categorical([‘a’, ‘b’ , ‘c’] , ordered = True) # It creates a categorical variable, where a>b>c.
77. s.categories # To get the categories of the series.
78. s.add_categories(['d']) # To append new categories.
79. s.remove_categories([‘d’]) # To remove a category.
80. cat_s1 > cat_s2 # To compare the categorical data.
81. pd.read_csv(“filename”, **usecols = [1,3,5,7]**) #We can also import particular columns only.
82. pd.read_csv(“filename”, dtype={‘Col’ : np.float64}) # To set the data type of the column.
83. pd.read_csv(“filename”, index_col=[‘Col_name’] # To set a column as the index.
84. pd.read_csv(“filename”, names=[‘A’, ‘B’ , ‘C’]) # To set the headers of the dataframe.
85. pd.read_csv(“filename”, header=5) # To set the fifth row as the header.
86. pd.read_csv(“filename”, header=None) # If there is no header.
87. pd.read_csv(“filename”, skiprows=15) # To skip the top 15 rows.

88. Reading Multiple Sheets from Excel File

```
with pd.ExcelFile("filename.xls") as xls:
   df1 = pd.read_excel(xls, 'Sheet1')
   df2 = pd.read_excel(xls, 'Sheet2’)
```

89. Data Normalization → Simple Feature Scaling > df.Col = df.Col / df.Col.max( )
Data Normalization → Min-Max > df.Col = (df.Col – df.Col.min( )) / (df.Col.max( ) – df.Col.min( ))
Data Normalization →Z-Score > (df.Col = df.Col - df.Col.mean( )) / df.Col.std( )

90. Creating Bins

```
bins = np.linspace( min(df.col) , max(df.col), 4)
group_names = [‘Low’, ‘Medium’, ‘High’]
df.binned_col = pd.cut( df.col , bins, labels=group_names, include_lowest=True)
```

91. To install SQLAlchemy Package - conda install sqlalchemy
from sqlalchemy import create_engine

92. To read a text file(unstructured data) and showing each line separately

```
with open(“filename.txt”) as fn:
# Read each line
   ln = fn.readline()
#Kee count of lines
   lncnt = 1
   while ln:
      print("Line {} : {}".format(lncnt, ln.strip()))
      ln = fn.readline()
      lncnt +=1
```

93. **Counting Word Frequency –** To count the frequency of words in file, we use Counter function.

```
with open('filename.txt') as f:
   p = Counter( f.read( ).split( ) )
   print(p)
```

94. **Word Tokenization –** A process of splitting a large sample of text into parts. This is a requirement in NLP tasks where each word needs to be captured and subjected to further analysis like classifying and counting them for a particular sentiment etc. The NLP kit is a library used to achieve this.

```
conda install –c ananconda nltk
# To split the paragraph into words
import nltk
word_data = ‘My name is Ram’
nltk_tokens = nltk.word_tokenize(word_data)
print(nltk_tokens)
# To split the paragraph into sentences
import nltk
sentence_data = 'My name is Ram. Sham is my friend'
nltk_tokens = nltk.sent_tokenize(sentence_data)
print(nltk_tokens)
```

95. **Stemming –** In NPL, we come across situations where two or more words have a common root. Ex: the three words – agreed, agreeing, and agreeable have the same root word agree. A search involving any of these words should treat them as the same word which is the root word. So, it becomes essential to link all the words into their root word. The NLTK library has methods to do this linking and give the output showing the root word. To use Porter Stemming Algorithm for stemming:

```
import nltk
nltk.download('punkt')
from nltk.stem.porter import PorterStemmer
porter_stemmer = PorterStemmer()
word_data = "My Name is Ram"
# First word tokenization
nltk_tokens = nltk.word_tokenize(word_data)
# Next find the roots of the word
for w in nltk_tokens:
   print("Actual : %s Stem : %s" % (w,porter_stemmer.stem(w)))
```

96. **Lemmatization –** It is similar to stemming but it brings context to the words. It link words with similar meaning to one word. Ex. If a paragraph has words like cars, trains & automobile, then it will link all of them to automobile.To use Wordnet lexical database for lemmatization.

```
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
word_data = "I have two cars , one train , one bicycle which are in automobile sector"
nltk_tokens = nltk.word_tokenize(word_data)
for w in nltk_tokens:
   print("Actual : %s , Lemma : %s" % (w, wordnet_lemmatizer.lemmatize(w)))
```

97. **To load any dataset from Seaborn Library –** df = sns.load_dataset('dataset_name-iris/tips')
98. **ANOVA TEST –** import scipy , from scipy import stats
    stats.f_oneway(df.groupby('cat_col').get_group('element1')['num_col'] , df.groupby('cat_col').get_group('element2')['num_col'])
99. **Pearson Correlation** – stats.pearsonr(df.Col1 , df.Col2)
    Pearson Correlation Heatmap : sns.heatmap(df.corr() , vmin=-1, vmax=1, center=0)
100. **CrossTab : It is used to compute a simple cross-tabulation of two or more factors.**
    pd.crosstab(data['Col_1'] , data['Col_2'] , margins=True , normalize=False)

1. Extract only Categorical columns from dataset : data.select_dtypes("object")
2. Extract only Integer columns from dataset : data.select_dtypes("int64")
3. Extract only Numeric columns from dataset : data.select_dtypes("number")
4. To get month/year/day wise sales – df.groupby('month/year/day_column').sum()
5. Interactive Shell : Run 2 or more commands simultaneously
   from IPython.core.interactiveshell import InteractiveShell
   InteractiveShell.ast_node_interactivity = 'all'