PANDAS PROFILLING			
Sr.No.	**Syntax**	**Output**	**Remarks**
1	Import pandas as pd		
	df = pd.read_excel('Attribute DataSet.xlsx')		
	type(df)	pandas.core.frame.DataFrame	
2	df = pd.read_excel(r'D:\Data Science\Ineuron\Main course\Python\Pandas\data fsds - 20221122T052032Z-001\data fsds\Attribute DataSet.xlsx')		Document in the form of CSV, excel, JSON format can be loaded by either uploading in directory or by using location link from system.
3	df1 = pd.read_excel(r'Attribute DataSet.xlsx', sheet_name="Sheet1")		You can specify sheet name or index to open particular sheet in the excel document.
	df1 = pd.read_excel(r'Attribute DataSet.xlsx', 0)		
4	df1 = pd.read_excel(r'Attribute DataSet.xlsx', sheet_name="Sheet1", header=1)		To choose a different header row for pandas profiling. Pandas uses first row as deault header.
5	df1 = pd.read_excel(r'Attribute DataSet.xlsx', sheet_name="Sheet1", header=None , names=['A','B','C','D','E','F','G','H','I','J','K','L','M','N'])		Use custom column names if it is not present in the dataset.
6	df.head(2)		It will show top rows as per arg parsed.
7	df.tail()		It will show bottom most rows as per arg parsed.
8	df3 = pd.read_csv(r'haberman test.txt',sep ='@')		If Delimiter other than comma in csv, the same shall be specified to get individual columns.
9	pd.read_csv('https://raw.githubusercontent.com/Opensourcefordatascience/Data-sets/master/blood_pressure.csv')		To load data from github, use its raw format from github repository to load dataset.
10	a = pd.read_html('https://www.basketball-reference.com/leagues/NBA_2015_totals.html')		To extract table from the website. It will return a list in case of multiple tables. Using index of the particular table will return a dataframe.
	a[0]		
11	js=pd.read_json('https://api.github.com/repos/pandas-dev/pandas/issues')		Read json file in pandas using json viewer to get idea about the structure
	js.columns		
	js['user']		
12	df.to_csv('D:\Data Science\Ineuron\Main course\Python\Pandas\Pandas_practice.csv', sep='#')		To save file at required location and using # instead of comma as delimiter
13	df.to_csv('D:\Data Science\Ineuron\Main course\Python\Pandas\Pandas_practice1.csv', sep='#',index=False)		Index will be removed before saving the file
	pwd		Return the current working directory path.
	Analysising the data		
14	df.head()		It will show top 5 rows of dataset
15	df.tail()		It will show bottom 5 rows of dataset
16	df.columns		It will return list of column names
17	df.dtypes		It will give data type of each of the columns/feature in the dataset
18	df['profit']	Same output	It will return data under profit in series format
	df.profit		
19	df['profit','year']	Error	You cannot pass 2 argument without additional brackets

20	df[['profit','year']]		It will retun dataframe of 2 columns. You have to pass the columns in list parameter.
21	df[['profit']]		It will retun dataframe of single column
22	df[['order_id','product_id','quantity']]		It will showcase all these columns
23	df.describe()		It will give statistical data of only numerical columns in the dataset
24	df.dtypes == 'object'		It will return boolean True for all object columns and false for numerical columns.
25	df.dtypes[df.dtypes == 'object']		It will return series format of the object columns
26	df.dtypes[df.dtypes == 'object'].index		It will give list of all the object columns in the dataset.
27	df[df.dtypes[df.dtypes == 'object'].index]		It will return dataframe of object columns
28	df[df.dtypes[df.dtypes == 'object'].index].describe()		It will give statistical data of only object columns in the dataset
29	df[df.dtypes[df.dtypes == 'float64'].index]		It will filter out only float type columns in the dataset with dataframe
30	df['order_id'][1:40:2]		It will return series data from index 1 to 40 with a jump of 2 **(slicing operation)**
31	df['category1'] = "sudh"		It will add a new column "catagory1" and assign the same value to all rows in the column
32	df['order_id'].isnull()		It will return series of boolean true and false values over each value of the column.
33	df[df['profit'] == max(df['profit'])]		To filter out row who has got max profit
34	df[df['profit'] == max(df['profit'])]['customer_name']		Name of customer who has got max profit
35	df[df['country'] == 'Sweden']		It will give dataframe of all sweden countries
36	len(df[df['country'] == 'Sweden'])		It will give count of number of rows in the new dataset.
37	df[df['shipping_cost'] > 80]['country']		Series format of all countries where shipping cost is > 80
38	df[(df['shipping_cost'] > 100)& (df['profit']< 10)]		Dataframe with rows satisfying both the conditions
39	df[['profit','customer_name']].max()	profit 8399.976 customer_name Zuschuss Donatelli dtype: object	Profit value along with name of customer with max profit. The sequence dosent matter.
40	df['converted_order_date'] = pd.to_datetime(df['order_date'])		Convert date from string format to datetime format and create a separate column for it
41	df['order_date_year'] = df['converted_order_date'].dt.year		It will extract the year from date and create a separate column for year
42	df['order_date_month'] = df['converted_order_date'].dt.month		It will extract the month from date and create a separate column for month
43	df['order_date_month'].value_counts()		It will perform groupby count operation
44	df['cost_to_comapny'] = df['discount'] + df['shipping_cost']		It will create new column by adding values from 2 other columns
45	df[df['cost_to_comapny'] == max(df['cost_to_comapny'])]['product_name']		Find the product where cost to company is max

46	df.drop('cost_to_comapny',axis=1,inplace=True)		To drop a **column** from dataset. In order to retain changes use inplace=True. Or you can reassign the changes to retain the same.
47	df.drop(1, inplace=True)		It will drop the **1 index row** and due to inplace=True it will retain the changes
48	df.loc[[2,3]]		It will filter out record from 2 and 3 index rows or labels
49	df.loc[0:4,['order_id','order_date','ship_date']]	Same output	It will filter data with reference to rows and columns
50	df[["Dress_ID", "Style", "Price"]].loc[0:4]		
51	df.iloc[0:4 , 0:3]		It will filter data with reference to default index of rows and columns
52	df.dtypes[(df.dtypes == 'float64') |(df.dtypes == 'int64')]		It will show all columns which are either float type or int type
53	df2 = df[df.dtypes[(df.dtypes == 'float64') |(df.dtypes == 'int64')].index]		Dataframe of only numerical columns
54	df3.dropna()		It will drop rows even if it has a single NaN value. Default axis = 0. axis = 0, or 'index' : Drop rows which contain missing values.
55	df3.dropna(axis=1)		It will drop columns even if it has a single NaN value. Axis = 1, or 'columns' : Drop columns which contain missing value.
56	df.dropna(how='all')		Drop the rows where all elements are missing.
57	df.dropna(thresh=2)		Providing condition for NaN values to drop the column
58	df.dropna(subset=['name', 'toy'])		Define columns to look for missing values. If you are dropping rows these would be a list of columns to include.
59	df3.fillna(value=4)		It will fill all NaN values with 4
60	df3.fillna(value=df3['profit'].mean())		It will fill all NaN values with mean of profit
61	df.groupby('order_date_year')['profit'].mean()		This will give avg of profit each year
62	df.groupby('order_date_year')['sales'].mean()		Avg sales
63	df.groupby('order_date_year')['shipping_cost'].mean()		Avg shipping cost
64	df.groupby('order_date_year')['discount'].mean()		Avg discount
	df['sales'] = df.sales.str.replace(',' , "").astype(int)		Convert str to int
	dfs3['Pin Code'] = dfs3['Pin Code'].astype(int)		
65	df.groupby('order_date_year')['sales'].sum()		Sum of sales each year
	ADVANCE		
18	df = pd.DataFrame(data,index=(4,5,6,7))		It will change the indexing of rows in the dataset
19	df.loc[5:6]	Both will show same result	It will extract data between the two row indexes that are labeled.
20	df.iloc[1:3]		It will extract data between the two row indexes that are default indexes
	df.iloc[0:5,2:7]		Data between (rows,columns) indexes.
21	pd.concat([df,df1])		Performing concatination operation **horizontally** (where axis=1)
	pd.concat([df,df1] , axis=1)		
23	pd.concat([df2,df3], axis=0)		Performing concatination operation **vertically** of datasets with same column names (where axis=0)

24	pd.merge(df4,df5)		It will filter out only that data which is common
25	pd.merge(df4,df5 , how ='left')		Similar to MySQL where we perform left join
26	pd.merge(df4, df5 , how = 'right' , on = 'emp_id')		Perform right joining operation based on certain column labels which is common between the 2 dataset.
27	pd.merge(df6,df7 , left_on='emp_id1' , right_on='emp_id2',how = 'inner')		Similar to MySQL where we perform inner join
28	pd.merge(df8,df9 , on = ['emp_id' , 'salary'])		it will merge the two dataset based on the two columns and only data which is common in both columns will be shown.
29	df10.join(df11)		Perform joining operation based on indexes

30	def profit_flag(a) :
	  if a >= 0 :
	  return 'positive'
	  else :
	  return 'negative'
			defiing functions and using it to create a separate column

31	df_sales['flag_profit'] = df_sales['profit'].apply(profit_flag)		
32	df_sales['len_cust_name'] = df_sales['customer_name'].apply(len)		Apply shall take argument in function
33	df_sales['square_quantity'] = df_sales['quantity'].apply(lambda a : a**2)		Applying lambda function