CORE PYTHON

LIST – [int /float / str] → A = [1 , 2 , 3.4 , 3.4, ‘a’ , ‘bcd’]
→ Collection of data-types, Mutable : Values can be changed , Ordered : Values order will be as it is , Changeable , Homogeneous Data, Allows duplicate values.

TUPLE – (int / float / str) → B = (1 , 2 , 3.4 , 3.4 , ‘a’ , ‘bcd’)
→Immutable : Values can’t be changed , Ordered : Values order will be as it is , Unchangeable, Heterogeneous Data, Allows duplicate values.

SET – { int / float /str } → C = { 1 , 2 , 3.4 , 5.6 , ‘a’ , ‘bcd’ }
→Values can’t be changed but new values can be added , Unordered : Values order may change , Arrange the items in ascending order, Doesn’t allow duplicate values, Un-indexed.

DICTIONARY – { Key : Value } → D = { K1 : 1 , K2 : 2 , K3 : 3.4 , K4 : 5.6 , K5 : ‘ab’ , K6 : ‘bcd’ }
→Mutable , Unordered , Doesn’t allows duplicate keys , Indexed.

LIST FUNCTONS

A.append(55) - To add a new value at the end of the list.
A.clear( ) – To clear/delete/blank a list.

DICTIONAY FUNCTONS

D.clear( ) – To delete the dictionary.
E = D.copy( ) – To copy a dictionary.

LAMBDA – fun_name = lambda parameters : single line statement ,
Ex : sum = lambda a ,b : a + b

ENUMERATE FUNCTION

It is used to display output with index. We can enumerate as list, tuple, set, dictionary.

Syntax : enumerate(list)
Ex : list (enumerate [‘apple’ , ‘mango’ , ‘orange’])

****

NUMPY

1. **Import numpy as np**

2. **1-D Array** - A = np.array([1,2,3,4,5]) # To create a One-dimensional array.

3. **2-D Array** - A = np.array([[1,2,3],[4,5,6]]) # To create a Two-dimensional array.

4. **3-D Array** - A = np.array([[[1,2,3],[4,5,6],[7,8,9]]]) # To create a Three-dimensional array.
5. **np.random()** - A = np.random.random() # Create an array with random values.
   A = np.random.random((2,3))

6. **np.linspace ()** - A = np.linspace (1,100,12) # It returns evenly spaced values within a given interval.
   np.linspace(start, stop , num=50, endpoint=True, retstep=True, dtype=None)

7. **Array Indexing** - a[1:2,1:2,1:2]
   # Since arrays may be multidimensional, we must specify a slice for each dimension of the array.

8. **random()** -

np.random.random(5) # It takes only one number x(5 here) & displays values equal to number quantity.

np.random.randint(5,20,4) # It displays given no. of values(4 here) between given input numbers 5 & 20.

np.random.randn(2,3,4) # It displays values (+/-) in the form of arrays.

np.random.uniform(1,5,50) # It displays given no. of unique values between given input numbers.

****

D S L
DATA SCIENCE LOVERS

PANDAS

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)

For Exploring The Data

1. **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 a 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 Null Values also.

2. **df.nunique ( )** - df.nunique( ) # It shows the total no. of unique values in each column.

3. **df.describe( )** - # It shows all summary of the dataframe.
   For a categorical dataframe, it will show a simple summary of unique values & most frequently occurring values.

For Selecting The Data

1. **df[[‘Col1’, ‘Col2’ , ‘Col3’]]** # Selecting multiple Columns from the DF.
2. **df.loc[: , ‘Col1’ : ‘Col2’]** # Selecting columns with object slicing.
3. **df.iloc[: , 1:4]** # Selecting columns with integer slicing.

Adding / Removing

1. **DataFrame** - # To create a dataframe.
   pd.DataFrame(data=, index=, columns=) ,
   pd.DataFrame(np.arange(1,10).reshape(3,3), index=[‘a’,’b’,’c’], columns = list(‘XYZ’))
2. **Adding New Row/Index** # To add a new row in the series
   s.loc[‘new index’] ,
3. **Adding New Column** - # To add new column in the DF.
   df[‘New_col’] ,
4. **Adding New Row** - # To add new row in the DF.
   df.loc[‘R’ , 2:5] = 78
5. **Removing Columns** -
   df.drop(‘Col_name’ , axis=1) ,
6. **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.
7. **pd.concat([df1,df2] , axis=0/1 , join= ‘inner/outer’)**

For Cleaning The Data

1. **.astype()** - s.astype(int), s.astype(float), s.astype(str)
   # Converting the data type of the series to a new data type.
2. **s.replace( )** - s.replace({1:‘one’ , ‘b’:‘bombay’})
   # To replace any data of the series with a new value using dictionary format.
8. **df.isnull( )** - df.isnull( ) , df.isnull( ).sum( )
   # It detects the missing values from the dataframe.
9. **df.notnull ( )** - df.notnull( ) df.notnull.sum( )

It detects the existing (non-missing) values from the dataframe.

10. **df.duplicated( )** - df.duplicated( ) , df[df.duplicated( )]
It checks row wise and detects the duplicate rows.

For Analyzing TheData

1. **df.pivot_table(values= 'Col1' , index= 'Col2' , columns= 'Col3')** ,
It creates a spreadsheet style pivot table as a DF.

2. **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.

3. **df[df.Col1 = = 'Element1'].Col2.value_counts( ) , df[df.Col1 = = 'Element1'].Col2.max( ) / sum ( )**
From Col1 selecting rows with element1 & show result of Col2.

4. **len( )** - To check the length of object like list, string etc.

For Saving/Writing The Data

1. df.to_csv(filename) # Writes to a CSV file
2. df.to_excel(filename) # Writes to an Excel file

Date-Time

1. **to_datetime ( )** - pd.to_datetime(DF.Date_Time_Col)
Converts the data-type of Date-Time Column into datetime[ns] datatype.

2. **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.

3. **From the Date-Time column, showing only hour, minute, month, weekdays** -
df['Time_Col'].dt.hour ,

OTHERS

1. **Dummies** - **df[‘Col_name ‘]= = ‘a’** # Creates dummy for level ‘a’ in True & False format.

2. **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’])

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

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

5. **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’])

6. **DataFrame Profiling –**
   conda install -c anaconda pandas-profiling
   import pandas_profiling
   pandas_profiling.ProfileReport(df)

D S L
DATA SCIENCE LOVERS

MATPLOTLIB

1. **from matplotlib import style , style.use(“ggplot”)** # For style purpose.

2. **plt.xlabel(‘Year’) , plt.ylabel(‘Sales’)** # To show the labels on x-axis and y-axis.

3. **plt.title(‘Year Sales Diagram’, fontsize=24)** # To show the title on the graph.

4. **plt.figure(figsize=(10, 20))** # To adjust the figure size.

5. **Bar Plot** - plt.bar(x-elements, y-elements)

6. **Scatter Plot** - plt.scatter(x-elements, y-elements , color = ‘r’, s = 20 , edgecolor= ‘red’ . style='*-')

7. **Stack Plot** - plt.stackplot(list1, list2, list3, list4 , color = ‘mcbr’)

8. **Graph from Pandas directly :**
   df.plot(x = ‘Year’, y = ‘Sales’ , kind = “ line/scatter/box/area/stack/pie/bar”, figsize = (25,4)).

9. **To check the relationship between two columns :**
   sns.relplot(x = ‘Col_1’ , y = ‘Col_2’ , data = df_name)

SQL

Types of Database:

1) Distributed Database……2) Object Oriented Database…..3) Centralized Database…..

Remove Database

Syntax: DROP DATABASE database_name;

CREATE DATABASE

Syntax: CREATE DATABASE database_name;

CREATE TABLE

A Table is a collection of data in a tabular form.

Syntax 1 : CREATE TABLE table-name

Delete Table

Syntax: DROP TABLE table-name;

ADD Column - To add a new column in the existing table.
ALTER TABLE table-name

Describe Table

Syntax: DESC table-name;

DATE

It displays Date values in yyyy-mm-dd format.

VIEW

A view is a virtual table, which contains rows and columns just like a real table.

Syntax:
CREATE VIEW *view_name* AS

SELECT

Syntax: SELECT * FROM table-name;
These operators are used during WHERE query.

= , != , > , < , >= , <= , BETWEEN , LIKE, IN

OPERATORS

AND OPERATOR

Syntax: SELECT * FROM table-name

MAX – This function returns the largest value of the selected column.

Syntax: SELECT MAX (Col_name)

GROUPBY

It is used in SQL to arrange the identical data into groups with the help of some functions.

Syntax: SELECT Col_name(s)

INNER JOIN

This type of join returns those records which have matching values in both tables.

Syntax:
SELECT Table1.Col1, Table1.Col2, Table2.Col1….

RIGHT JOIN (Right Outer Join)

Syntax:
SELECT Table1.Col1, Table1.Col2, Table2.Col1….

LIKE OPERATOR

% - The percent sign represents zero, one, or multiple characters.