**|| Pandas ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| pandas.Series() | pandas.Series( data\*, index\*, dtype\*, copy\*) | Series data structure creation of pandas,if no argument passed it creates an empty series, Dictionary keys are used to construct index |
| pandas.DataFrame() | pandas.DataFrame( data\*, index\*, columns\*, dtype\*, copy\*) | Constructor of creating DataFrame structure |
| Pandas.Panel() | pandas.Panel(data\*, items\*, major\_axis\*, minor\_axis\*, dtype\*, copy\*) | Constructor of creating panel data |
| panel.major\_axis() | panel.major\_xs(index). | Major axis data can be accessed using the method |
| panel.minor\_axis() | panel.minor\_xs(index) | Minor axis data can be accessed using the method |

**||Series Basic Funtionality ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .axes | Series.axes | Returns the list of the labels of the series |
| .empty | Series.empty | Returns the Boolean value saying whether the Object is empty or not |
| .ndim | Series.ndim | Returns the number of dimensions of the object |
| .size | Series.size | Returns the size(length) of the series |
| .values | Series.values | Returns the actual data in the series as an array |
| .head() | Series.head() | head() returns the first n rows(observe the index values). The default number of elements to display is five |
| .tail() | Series.tail() | tail() returns the last n rows(observe the index values). The default number of elements to display is five |

**|| DataFrame Basic Funtionality ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .T | DataFrame.T | Returns the transpose of the DataFrame |
| .axes | DataFrame.axes | Returns the list of row axis labels and column axis labels |
| .dtypes | DataFrame.dtypes | Returns the data type of each column |
| .empty | DataFrame.empty | Returns the Boolean value saying whether the Object is empty or not |
| . ndim | DataFrame.ndim | Returns the number of dimensions of the object |
| .shape | DataFrame.shape | Returns a tuple representing the dimensionality of the DataFrame. Tuple (a,b), where a represents the number of rows and b represents the number of columns |
| .size | DataFrame.size | Returns the number of elements in the DataFrame |
| .values | DataFrame.values | Returns the actual data in the DataFrame as an NDarray |
| .head() | DataFrame.head() | returns the first n rows. The default number of elements to display is five |
| .tail() | DataFrame.tail() | returns the last n rows. The default number of elements to display is five |

**|| StatisTical Function ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .sum() | DataFrame.sum(axis\*) | Returns the sum of the values for the axis,default, axis is 0. |
| .mean() | DataFrame.mean() | Returns the average value |
| .std() | DataFrame.std() | Returns the Bressel standard deviation of the numerical columns |
| .count() | DataFrame.count() | Counts number of non-null observations |
| .median() | DataFrame.median() | Returns the median of Values |
| .mode() | DataFrame.mode() | Return the mode of values |
| .min() | DataFrame.min() | Returns minimum Value |
| .max() | DataFrame.max() | Returns maximum Value |
| .prod() | DataFrame.prod() | Returns the product of Values |
| .cumsum() | DataFrame.cumsum() | Returns the cumulative sum of Values |
| .cumprod() | DataFrame.cumprod() | Returns the cumulative product of Values |
| .describe() | DataFrame.describe(object\*,number\*,all\*) | computes a summary of statistics pertaining to the DataFrame columns |
| .pct\_change() | DataFrame.pct\_change(axis\*) | compares every element with its prior element and computes the change percentage |

**|| Function’s Application ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .pipe() | DataFrame.pipe() | operations can be performed by passing the function and the appropriate number of parameters as pipe arguments |
| .apply() | DataFrame.apply(axis\*) | Arbitrary functions can be applied along the axes of a DataFrame or Panel using the apply() method |
| .applymap() | DataFrame.applymap() | taking a single value and returning a single value |
| .map() | DataFrame[col].map() | analogously map() on Series accept any Python function taking a single value and returning a single value |

**|| Reindexing ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .reindex() | DataFrame.reindex(index,columns\*) | Reorder the existing data. |
| .reindex\_like() | DataFrame.reindex\_like(another DataFrame,method\*=ffil/bfill/  nearest,limit\*) | takes an object and reindex its axes to be labeled the same as another object |
| .rename() | DataFrame.rename(columns,  index,inplace) | method allows to relabel an axis based on some mapping (a dict or Series) or an arbitrary function |

**|| Iteration ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .iteritems() | DataFrame.iteritems() | Iterates over each column as key, value pair with label as key and column value as a Series object |
| .iterrows() | DataFrame.iterrows() | returns the iterator yielding each index value along with a series containing the data in each row |
| .itertuples() | DataFrame.itertuples() | will return an iterator yielding a named tuple for each row in the DataFrame |

**|| Sorting ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .sort\_index() | DataDrame. sort\_index(axis=0\*, level=None\*, ascending=True\*, inplace=False\*, kind=’quicksort’\*, na\_position=’last’\*, sort\_remaining=True\*, by=None\*) | sorts objects by labels along the given axis |
| . sort\_values() | DataFrame. sort\_values(by,kind\*) | method for sorting by values. It accepts a 'by' argument which will use the column name of the DataFrame with which the values are to be sorted |

**|| String function ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .str.lower() | Series.str.lower() | Converts strings in the Series/Index to lower case |
| .str.upper() | Series.str.upper() | Converts strings in the Series/Index to upper case |
| .str.len() | Series.str.len() | Computes String length() |
| .str.strip() | Series.str.strip() | Helps strip whitespace(including newline) from each string in the Series/index from both the sides |
| .str.split() | Series.str.split(pattern\*) | Splits each string with the given pattern |
| .str.cat() | Series.str.cat(sep=pattern\*) | Concatenates the series/index elements with given separator |
| .str.get\_dummies() | Series.str.get\_dummies(sep\*) | Returns the DataFrame with One-Hot Encoded values |
| .str.contains() | Series.str.contains(pattern) | Returns a Boolean value True for each element if the substring contains in the element, else False |
| .str.replace() | Series.str.replace(a,b) | Replaces the value a with the value b |
| .str. repeat() | Series.str. repeat(value) | Repeats each element with specified number of times |
| .str.count() | Series.str.count(pattern) | Returns count of appearance of pattern in each element |
| .str.startswith() | Series.str.startswith(pattern) | Returns true if the element in the Series/Index starts with the pattern |
| .str.endswith() | Series.str.endswith(pattern) | Returns true if the element in the Series/Index ends with the pattern |
| .str.find() | Series.str.find(pattern) | Returns the first position of the first occurrence of the pattern |
| .str.findall() | Series.str.findall(pattern) | Returns a list of all occurrence of the pattern |
| .str.swapcase() | Series.str.swapcase() | Swaps the case lower/upper |
| .str.islower() | Series.str.islower() | Checks whether all characters in each string in the Series/Index in lower case or not. Returns Boolean |
| .str.isupper() | Series.str.isupper() | Checks whether all characters in each string in the Series/Index in upper case or not. Returns Boolean |
| .str.isnumeric() | Series.str.isnumeric() | Checks whether all characters in each string in the Series/Index are numeric. Returns Boolean |

**|| DataFrame Display Function ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .get\_option() | pandas.get\_option ("display.max\_rows") | returns the max rows value to be displayed |
| .get\_option() | pandas.get\_option ("display.max\_columns") | returns the max columns value to be displayed |
| .get\_option() | Pandas. .get\_option(“display.precision”) | returns the precision to be displayed after decimal |
| .set\_option() | pandas.set\_option ("display.max\_rows",value) | changes the default number of rows to be displayed |
| .set\_option() | pandas.set\_option ("display.max\_ columns ",value) | changes the default number of columns to be displayed |
| .set\_option() | Pandas. .set\_option(“display.precision”,value) | returns the precision to be displayed after decimal as value passed |
| .reset\_option() | Pandas. .reset\_option(("display.max\_rows") | changes the value back to the default number of rows to be displayed |
| .reset\_option() | Pandas. .reset\_option(("display.max\_columns") | changes the value back to the default number of columns to be displayed |
| .reset\_option() | Pandas. .reset\_option(“display.precision”) | changes the precision back to the default number to be displayed |

**|| Indexing and Slicing ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .loc[] | DataFrame.loc[row,columns] | Returns purely label based indexing |
| .iloc[] | DataFrame.iloc[row,columns] | Returns integer based indexing |
| .ix[] | DataFrame.ix[row,columns] | Returns both integer based and label based indexing |

**|| Window Function ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .rolling() | DataFrame.rolling(window, min\_periods=None\*, freq=None\*, center=False\*, win\_type=None\*, on=None\*, axis=0\*, closed=None\*) | Returns rolling aggregated functions result on a specific window |
| .expanding() | DataFrame.expanding(min\_periods,center\*,axis\*) | Provide expanding transformations |
| .ewm() | DataFrame.ewm(self, com=None, span=None, halflife=None, alpha=None, min\_periods=0, adjust=True, ignore\_na=False, axis=0 ) | Provide exponential weighted functions |
| .aggregate() | DataFrame.aggregate(self, func, axis=0, \*args, \*\*kwargs) | Aggregate using one or more operations over the specified axis |

**|| Working with missing Data ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .isnull() | DataFrame.isnull() | Returns True if null |
| .notnull() | DataFrame.notnull() | Returns True if not null |
| .fillna() | DataFrame.fillna() | function can “fill in” NA values with non-null data |
| .dropna() | DataFrame.dropna(axis\*) | if any value within is NA then the whole row or columns is excluded |
| .replace() | DataFrame.replace() | replaces a generic value with some specific value |

**|| Groupby,Merging,Date-time ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .groupby() | DataFrame.groupby(label).groups | Returns grouped data depending on label |
| .groupby.get\_group() | DataFrame.groupby().get\_group() | Returns grouped data depending on label’s data |
| .groupby().filter() | DataFrame.groupby().filter() | Return a copy of a DataFrame excluding elements from groups that do not satisfy the boolean criterion specified by func |
| .merge() | pandas.merge(left, right, how='inner', on=None, left\_on=None, right\_on=None,  left\_index=False, right\_index=False, sort=True) | The join is done on columns or indexes. If joining columns on columns, the DataFrame indexes will be ignored. Otherwise if joining indexes on indexes or indexes on a column or columns, the index will be passed on |
| .concat() | pandas.concat(objs,axis=0,join='outer',join\_axes=None,ignore\_index=False) | performing concatenation operations along an axis |
| .append() | DataFrame.append(self, other, ignore\_index=False, verify\_integrity=False, sort=None) | Append rows of other to the end of caller, returning a new object |
| .datetime.now() | Pandas.datetime.now() | Returns current date and time |
| .Timestamp() | pandas.Timestamp(ts\_input,freq,tz,unit,year,month,day,hour, minute, second, microsecond,nanosecond,tzinfo) | Timestamp is the pandas equivalent of python’s Datetime and is interchangeable with it in most cases. It’s the type used for the entries that make up a DatetimeIndex, and other timeseries oriented data structures in pandas |
| .date\_range() | pandas.date\_range(start=None, end=None, periods=None, freq=None, tz=None, normalize=False, name=None, closed=None, \*\*kwargs) | Return a fixed frequency DatetimeIndex |
| .to\_datetime() | pandas.to\_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, box=True, format=None, exact=True, unit=None, infer\_datetime\_format=False, origin='unix', cache=True) | Convert argument to datetime |
| .Timedelta() | pandas.Timedelta(value,unit\*,days\*, seconds\*, microseconds\*,milliseconds\*, minutes\*, hours\*, weeks\*) | Represents a duration, the difference between two dates or times |
| .Categorical() | pandas.Categorical(values, categories=None, ordered=None, dtype=None, fastpath=False) | Represent a categorical variable in classic R / S-plus fashion |

**|| Visualization ||**

|  |  |  |
| --- | --- | --- |
| **Name** | **Syntax** | **Details** |
| .plot() | DataFrame.plot() | Make plots of DataFrame using matplotlib / pylab |
| .plot.bar() | DataFrame.plot.bar() | Make bar plots |
| .plot.barh() | DataFrame.plot.barh() | Make horizontal bar plots |
| .plot.hist() | DataFrame.plot.hist() | Returns a histogram from the data |
| .diff().hist() | DataFrame.diff().hist() | Returns histograms from the data for each columns |
| .plot.box() | DataFrame.plot.box() | Returns a box plot to visualize the distribution of values within each column |
| .plot.area() | DataFrame.plot.area() | Returns an area plot |
| .plot.scatter() | DataFrame.plot.scatter() | Returns a scatter plot |
| .plot.pie() | DataFrame.plot.pie() | Returns a pie chart from the data |

*Created By:*

*\*\*\* Arif Hossain Jim \*\*\**