Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures.

Advantages:

==========

Easily handles missing data

It uses Series for one-dimensional data structure and DataFrame for multi-dimensional data structure

It provides an efficient way to slice the data

It provides a flexible way to merge, concatenate or reshape the data

Pandas Environment Setup in Python

==========

pip install pandas

If you install Anaconda Python package, Pandas will be installed by default

Series

======

A series is a one-dimensional data structure with homogenous data type values.

A Series is essentially a column, and a DataFrame is a multi-dimensional table made up of a collection of Series.

```
import pandas as pd
a = pd.Series([10, 20, 30])
print(a)
a = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
print(a)
a = pd.Series([10, 20,np.nan])
print(a)
type(a)
```


DataFrame

=========

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print df

a = [['Srinu',97],['Vasu',88],['Nivas',90]]
df = pd.DataFrame(a, columns=['Name','Marks'])
```

```
print(df)
type(df)   # DataFrame
df['Name']
b = df['Name']
type(b)          # Series
c = df[['Name']]
type(c)          # DataFrame

a = {'Name': ["Srinu", "Vasu"], 'Age': [35, 40]}
pd.DataFrame(a)

## Numpy to pandas
import numpy as np
h = np.array([[1,2],[3,4]])
print(h)
df_h = pd.DataFrame(h)
print(df_h)

## Pandas to numpy
df_h_n = np.array(df_h)
print(df_h_n)
```

Range Data

==========

Pandas have a convenient API to create a range of date

Syntax:

pd.data_range(date,period,frequency)

The first parameter is the starting date

The second parameter is the number of periods (optional if the end date is specified)

The last parameter is the frequency: day: 'D,' month: 'M' and year: 'Y.'

Create date

Days

```
dates_d = pd.date_range('20191110', periods=10, freq='D')
print(dates_d)
```

Months

```
dates_m = pd.date_range('20191110', periods=10, freq='M')
print(dates_m)
```

Inspecting data

==========

We can check the head or tail of the dataset with head(), or tail() preceded by the name of the panda's data frame

Create a random sequence with numpy. The sequence has 10 rows and 4 columns

random = np.random.randn(10,4)

Create a data frame using pandas.

Use dates_m as an index for the data frame. It means each row will be given a "name" or an index, corresponding to a date.

Finally, you give a name to the 4 columns with the argument columns

Create data with date

df = pd.DataFrame(random, index=dates_m, columns=list('ABCD'))

info() is used to get the information about dataframe

df.info()

df.shape

df.columns

```
# Work with missing values

# isnull() returns a DataFrame where each cell is either True or False depending on that cell's null status.

df.isnull()

# To count the number of nulls in each column we use an aggregate function for summing

df.isnull().sum()


# Removing null values

df.dropna()

# We can also drop columns with null values by setting axis=1:

df.dropna(axis=1)


# Using head function, it returns first 5 records

df.head()

df.head(n=5)

df.head(n=3)

df.head(3)


# Using tail function, it returns last 5 records

df.tail()

df.tail(3)
```

value_counts() is used to count the frequency of all values in a column

df['A'].value_counts()

corr() is used to generate the Relationships between continuous variables

By using the correlation method .corr() we can generate the relationship between df.corr()

For summarizing datause describe().

It provides the counts, mean, std, min, max and percentile of the dataset.

df.describe() # Summarizes Numeric columns

describe(include=['object']) # Summarizes String columns

describe(include='all') # Summarizes all columns together

Slice data:

==========

Using name

df['A']

To select multiple columns, you need to use two times the bracket, [[..,..]]

The first pair of bracket means you want to select columns, the second pairs of bracket tells what columns you want to return.

```
df[['A', 'B']]
```

using a slice for row

```
df[0:3]
```

The loc function is used to select columns by names. As usual, the values before the coma stand for the rows and after refer to the column. We need to use the brackets to select more than one column.

```
df.loc[:,['A','B']]
```

There is another method to select multiple rows and columns in Pandas. You can use iloc[]. This method uses the index instead of the columns name. The code below returns the same data frame as above

```
df.iloc[:, :2]
```

Drop columns using pd.drop()

```
df.drop(columns=['A', 'C'])
```

```
# pd.concat() is used to concatenate two DataFrame in Pandas.

# Create two DataFrames.
df1 = pd.DataFrame({'name': ['Srinivas', 'Vasu','Nivas'],
                    'Age': ['35', '40', '25']},
                   index=[0, 1, 2])
df2 = pd.DataFrame({'name': ['Srinivas', 'Reddy' ],
                    'Age': ['35', '26']},
                   index=[3, 4])


df_concat = pd.concat([df1,df2])
print(df_concat)
df_concat['name']
df_concat['name'] == "Srinivas"     # returns True or False
df_concat[df_concat['name'] == "Srinivas"]   # returns data



# Drop_duplicates
# If a dataset can contain duplicates information use, `drop_duplicates()` is an easy to exclude duplicate rows. You can see that `df_concat` has a duplicate observation, `Srinivas` appears twice in the column `name.`
df_concat.drop_duplicates('name')
```

```
# Sort value with sort_values()
df_concat.sort_values('Age')


# Rename: change of index
# rename() is used to rename a column in Pandas. The first value is the current column name and the second value is the new column name.
df_concat.rename(columns={"name": "Emp_Name", "Age": "Emp_Age"})
```

Descriptive Statistics

==========

a.sum()

count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of Values
median()	Median of Values
mode()	Mode of values
std()	Standard Deviation of the Values
min()	Minimum Value
max()	Maximum Value
describe()	Summarizing Data Summarizes Numeric columns

describe(include=['object']) Summarizes String columns

describe(include='all') Summarizes all columns together

Merging/Joining

==========

pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None,

left_index=False, right_index=False, sort=True)

pd.merge(student_details,course_details,on='s_name')

pd.merge(student_details, course_details, left_on='student_name', right_on='s_name', how='left')

pd.merge(student_details, course_details, left_on='student_name', right_on='s_name', how='right')

pd.merge(student_details, course_details, left_on='student_name', right_on='s_name', how='outer')

pd.merge(student_details, course_details, left_on='student_name', right_on='s_name', how='inner')

Working with Text Data

==========

s = pd.Series(['Srinivas', 'DATAhill', '9292005440', 'Hyderabad', 'info@datahill.in, 'dataanalysis','PYTHON','Pandas'])

```
s.str.islower()
s.str.isupper()
s.str.isnumeric()
s.str.lower()
s.str.upper()
s.str.swapcase()
s.str.len()
s.str.cat(sep='_')
s.str.replace('@','$')
s.str.repeat(2)
s.str.count('s')
s.str.startswith ('P')
s.str.endswith('s')
```

Function Application:

==========

Table wise Function Application: pipe()

Row or Column Wise Function Application: apply()

```
def add(a,b):
   return a+b
```

```
df.pipe(type)

# pipe() is used to perform on the whole DataFrame.

df.pipe(add,10)

# By default, the operation performs column wise

df.apply(type)

df.apply(np.mean)

# By passing axis parameter, operations can be performed row wise.

df.apply(type, axis=1)

df.apply(np.mean,axis=1)
```

Loading data in python:

==========

```
# Check working directory before to load data.

import os

os.getcwd()

# Incase you want to change the working directory, you can specify it in under os.chdir( ) function.

# Single backslash does not work in Python so use 2 backslashes while specifying file location.

os.chdir("C:\\Users\\DELL\\Documents\\")

os.getcwd()

os.chdir("E:/MLDatasets/")
```

os.getcwd()

Read CSV file with header row

==========

It's the basic syntax of read_csv() function. You just need to mention the filename. It assumes you have column names in first row of your CSV file.

```
emp = pd.read_csv("emp10.csv")
emp = pd.read_csv("E:/MLDatasets/emp10.csv")
print(emp)
```

It stores the data the way It should be as we have headers in the first row of our datafile. It is important to highlight that header=0 is the default value. Hence we don't need to mention the header= parameter. It means header starts from first row as indexing in python starts from 0. The above code is equivalent to this line of code.

```
emp = pd.read_csv("emp10.csv", header=0)
print(emp)
```

Inspect data after importing

```
emp.shape
emp.columns
emp.dtypes
emp.info()
```

Read CSV file with header in second row

Suppose you have column or variable names in second row. To read this kind of CSV file, you can submit the following command.

```
emp = pd.read_csv("emp10.csv", header = 1)

print(emp)
```

header=1 tells python to pick header from second row. It's setting second row as header. It's not a realistic example. I just used it for illustration so that you get an idea how to solve it. To make it practical, you can add random values in first row in CSV file and then import it again.


```
# Define your own column names instead of header row from CSV file

emp1 = pd.read_csv("emp10.csv", skiprows=1, names=['Emp_ID','Emp_Name','Emp_Desig','Emp_DOJ','Emp_Sal'])

print(emp1)
```

skiprows = 1 means we are ignoring first row and names= option is used to assign variable names manually.

Skip rows but keep header

```
emp = pd.read_csv("emp10.csv", skiprows=[1,2])
print(emp)
```

In this case, we are skipping second and third rows while importing. Don't forget index starts from 0 in python so 0 refers to first row and 1 refers to second row and 2 implies third row.

Instead of [1,2] you can also write range(1,3). Both means the same thing but range( ) function is very useful when you want to skip many rows so it saves time of manually defining row position.

NOTE:

When skiprows = 4, it means skipping four rows from top. skiprows=[1,2,3,4] means skipping rows from second through fifth. It is because when list is specified in skiprows= option, it skips rows at index positions. When a single integer value is specified in the option, it considers skip those rows from top

Read CSV file without header row

If you specify "header = None", python would assign a series of numbers starting from 0 to (number of columns - 1) as column names. In this datafile, we have column names in first row.

```
emp1 = pd.read_csv("emp10.csv", header = None)

print(emp1)


# Add prefix to column names

emp1 = pd.read_csv("emp10.csv", header = None,
prefix="var")

print(emp1)
```

In this case, we are setting var as prefix which tells python to include this keyword before each column name.

Specify missing values

The na_values= options is used to set some values as blank / missing values while importing CSV file.

```
emp1 = pd.read_csv("emp10.csv", na_values=['.'])

print(emp1)
```

Set Index Column

```
emp1 = pd.read_csv("emp10.csv", index_col ='eid')
```

print(emp1)

As you can see in the above output, the column ID has been set as index column

Read file with semi colon delimiter

---

wine = pd.read_csv("winequality-red.csv")

wine.shape

wine = pd.read_csv("winequality-red.csv", sep = ';')

wine.shape

wine.head()

Using sep= parameter in read_csv( ) function, you can import file with any delimiter other than default comma. In this case, we are using semi-colon as a separator.

Change column type while importing CSV

Suppose you want to change column format from int64 to float64 while loading CSV file into Python. We can use dtype = option for the same.

emp = pd.read_csv("emp10.csv", dtype = {"sal" : "float64"})

print(emp)

emp.info()

Measure time taken to import big CSV file

With the use of verbose=True, you can capture time taken for Tokenization, conversion and Parser memory cleanup.

```
emp = pd.read_csv("emp10.csv", verbose=True)
```

How to read CSV file without using Pandas package

To import CSV file with pure python way, you can submit the following command :

```
import csv
with open("E:/MLDataSets/emp10.csv") as a:
  d = csv.DictReader(a)
  l=list(d)
```

How to read CSV file from URL without using Pandas package

```
import csv
import requests

response =
requests.get('https://dyurovsky.github.io/psyc201/data/lab2/nycflights.csv').text
```

```
lines = response.splitlines()
d = csv.DictReader(lines)
l = list(d)
```

Read CSV File from External URL

==========

You can directly read data from the CSV file that is stored on a web link.

```
a = pd.read_csv("http://winterolympicsmedals.com/medals.csv")
a.shape
a.head()
```

This DataFrame contains 2311 rows and 8 columns. Using mydata02.shape, you can generate this summary.

Skip Last 5 Rows While Importing CSV

```
a = pd.read_csv("http://winterolympicsmedals.com/medals.csv", skipfooter=5)
a.shape
a.head()
```

In the above code, we are excluding bottom 5 rows using skipfooter= parameter.

Read only first 5 rows

```
a = pd.read_csv("http://winterolympicsmedals.com/medals.csv", nrows=5)
print(a)
# Using nrows= option, you can load top K number of rows.
```

Interpreting "," as thousands separator

```
a = pd.read_csv("http://winterolympicsmedals.com/medals.csv", thousands=",")
print(a)
```

Read only specific columns

```
a = pd.read_csv("http://winterolympicsmedals.com/medals.csv", usecols=[1,5,7])
a.shape
a.head()
```

The above code reads only columns based on index positions which are second, sixth and eighth position.

Read some rows and columns

---

a = pd.read_csv("http://winterolympicsmedals.com/medals.csv", usecols=[1,5,7], nrows=5)

a.shape

a.head()

In the above command, we have combined usecols= and nrows= options. It will select only first 5 rows and selected columns.

Write data in CSV format:

==========

emp.to_csv('new_emp.csv')

emp.to_csv('new_emp.csv', index=False)

Read Text File

==========

We can use read_table() function to pull data from text file. We can also use read_csv() with sep= "\t" to read data from tab-separated file.

```
a = pd.read_table("E:/MLDataSets/demo.txt")
a = pd.read_csv("E:/MLDataSets/demo.csv", sep ="\t")
```

Read Excel File

==========

The read_excel() function can be used to import excel data into Python.

```
emp = pd.read_excel("E:/MLDataSets/emp10.xlsx")
print(emp)
```

If you do not specify name of sheet in sheetname= option, it would take by default first sheet.

Read delimited file

==========

Suppose you need to import a file that is separated with white spaces.

```
mydata2 =
pd.read_table("http://www.ssc.wisc.edu/~bhansen/econometrics/invest.dat", sep="\s+", header = None)
```

NOTE:

Here it is difficult to remind all the modules in packages, functions in modules, arguments in functions, syntax of the function, so it is better to take help

List of the functions - Use Tab at the time of typing function

Function syntax & arguments - Use Shift+Tab in the function paranthesis

Visualization

==========

df['price'].plot.box()

df['price'].plot.box(vert=False)

df['price'].plot()

df['price'].plot.bar()

df['price'].plot.barh()

df['price'].plot.hist(bins=20)

df['price'].diff.hist(bins=20)

df['price'].plot.area()

df['price'].plot.scatter(x='a', y='b')

df['price'].plot.pie(subplots=True)