PANDAS ASSIGNMENT

1Ans: To load a csv file into a pandas DataFrame, we have to import pandas library. We can read or load a csv into a pandas DataFrame using pandas.read\_csv() method.

Example:

# Importing pandas library

import pandas as pd

# Using the function to load

# the data of example.csv

# into a Dataframe df

df = pd.read\_csv('example1.csv')

# Print the Dataframe

df

2Ans: To check the data type in pandas DataFrame we can use the “dtype” attribute. The attribute returns a series with the data type of each column. And the column names of the DataFrame are represented as the index of the resultant series object and the corresponding data types are returned as values of the series object. If any column has mixed data types are stored then the data type of the entire column is indicated as object dtype.

Example:

import pandas as pd

df = pd.DataFrame({'Col1':[4.1, 23.43], 'Col2':['a', 'w'], 'Col3':[1, 8]})

result=df.dtypes

print(result)

output:

Col1 float64

Col2 object

Col3 int64

dtype: object

3Ans: pandas.DataFrame.loc is a function used to select rows from Pandas DataFrame based on the condition provided.

Syntax: **Syntax:**df.loc[df[‘cname’] ‘condition’]

**Parameters**  
**df:** represents data frame  
**cname:** represents column name  
**condition:** represents condition on which rows has to be selected

Example:

import pandas as pd

cart={'Product': ['Mobile', 'AC', 'Laptop', 'TV', 'Football'],

        'Type': ['Electronic', 'HomeAppliances', 'Electronic', 'HomeAppliances', 'Sports'],

        'Price': [10000, 35000, 50000, 30000, 799]

       }

Df=pd.DataFrame(cart)

# Selecting the product of Electronic Type

select\_prod **=** df.loc[df['Type'] **==** 'Electronic']

print(select\_prod)

4Ans: One way of renaming the columns in a Pandas Dataframe is by using the rename() function.

import pandas as pd

rankings **=** {'test': ['India', 'South Africa', 'England',

                            'New Zealand', 'Australia'],

             'odi': ['England', 'India', 'New Zealand',

                            'South Africa', 'Pakistan'],

             't20': ['Pakistan', 'India', 'Australia',

                            'England', 'New Zealand']}

Df=pd.DataFrame(rankings)

Df.rename(columns={‘test’:’TEST’}, inplace=True)

Print(Df)

#renaming multiple columns

Df.rename(columns={‘odi’:’ODI’,’t20’:’T20’}, inplace=True)

Print(Df)

5Ans: The most common way to remove a column is using df.drop().

import pandas as pd

rankings **=** {'test': ['India', 'South Africa', 'England',

                            'New Zealand', 'Australia'],

             'odi': ['England', 'India', 'New Zealand',

                            'South Africa', 'Pakistan'],

             't20': ['Pakistan', 'India', 'Australia',

                            'England', 'New Zealand']}

Df=pd.DataFrame(rankings)

Df.drop(‘t20’,axis=1,inplace=True)

Print(Df)

Axis= 1 for droping columns

Axis= 0 for droping rows

6Ans: using unique() method we can get unique values in a dataframe.

Example:

import pandas as pd

data **=** {

    'A':['A1', 'A2', 'A3', 'A4', 'A5'],

    'B':['B1', 'B2', 'B3', 'B4', 'B4'],

    'C':['C1', 'C2', 'C3', 'C3', 'C3'],

    'D':['D1', 'D2', 'D2', 'D2', 'D2'],

    'E':['E1', 'E1', 'E1', 'E1', 'E1']

}

Df=pd.DataFrame(data)

Print(Df.B.unique())

#To get count of unique values

Print(Df.B.nunique())

Output: [‘B1’,’B2’,’B3’,’B4’]

4

7Ans: To get the count of missing values in each column of a dataframe, we can use the pandas isnull() and sum() functions together.

# count of missing values in each column

df.isnull().sum()

8Ans: Use the fillna() method and set a constant value in it for all the missing values using the parameter value.

Example:

import pandas as pd

import numpy as np

dataFrame = pd.DataFrame(

{

"Car": ['BMW', 'Lexus', 'Lexus', 'Mustang', 'Bentley', 'Mustang'],

"Units": [100, 150, np.NaN, 80, np.NaN, np.NaN]

}

)

constVal=200

dataFrame['Units'].fillna(value=constVal, inplace=True)

print(dataFrame)

9Ans:

df1 = pd.DataFrame({'team': ['A', 'A', 'A', 'A'],

'assists': [5, 7, 7, 9],

'points': [11, 8, 10, 6]})

df2 = pd.DataFrame({'team': ['B', 'B', 'B', 'B'],

'assists': [4, 4, 3, 7],

'points': [14, 11, 7, 6]})

df3 = pd.concat([df1, df2], ignore\_index=True)

print(df3)

# concatenating df1 and df2 along rows

df3=pd.concat([df1,df2],axis=0)

# concatenating df1 and df2 along columns

df3=pd.concat([df1,df2],axis=1)

10Ans: We can merge two Pandas DataFrames on certain columns using the merge function by simply specifying the certain columns for merge.

Import pandas as pd

df1 **=** pd.DataFrame({'Name':['Raju', 'Rani', 'Geeta', 'Sita', 'Sohit'],

                    'Marks':[80, 90, 75, 88, 59]})

df2 **=** pd.DataFrame({'Name':['Raju', 'Divya', 'Geeta', 'Sita'],

                    'Grade':['A', 'A', 'B', 'A'],

                    'Rank':[3, 1, 4, 2 ],

                    'Gender':['Male', 'Female', 'Female', 'Female']})

df1.merge(df2[['Name', 'Grade', 'Rank']])

The resultant dataframe contains all the columns of df1 but certain specified columns of df2 with key column Name i.e. the resultant column contains Name, Marks, Grade, Rank column.

Example2: In the resultant dataframe Grade column of df2 is merged with df1 based on key column Name with merge type left i.e. all the values of left dataframe (df1) will be displayed.

df1.merge(df2[['Grade', 'Name']], on **=** 'Name', how **=** 'left')

Example3: In this example, we have merged df1 with df2. The Marks column of df1 is merged with df2 and only the common values based on key column Name in both the dataframes are displayed here.

df2.merge(df1[['Marks', 'Name']])

11Ans: [.groupby()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html) and [.agg()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.agg.html) functions are used to group data in a pandas dataframe by specific column and apply an aggregate function

import pandas as pd

df = pd.DataFrame({'team': ['A', 'B', 'B', 'B', 'B', 'M', 'M', 'M'],

'position': ['G', 'G', 'F', 'G', 'F', 'F', 'C', 'C'],

'assists': [5, 7, 7, 8, 5, 7, 6, 9],

'rebounds': [11, 8, 10, 6, 6, 9, 6, 10]})

print(df)

print(df.groupby(['team', 'position']).agg({'assists': ['mean']})

print(df.groupby(['team', 'position']).agg({'rebounds': ['median', 'max']})

12Ans:DataFrame.pivot(*\**, *index=None*, *columns=None*, *values=None*)

Return reshaped DataFrame organized by given index / column values.

import pandas as pd

df **=** pd**.**DataFrame({'foo': ['one', 'one', 'one', 'two', 'two', 'two'],

'bar': ['A', 'B', 'C', 'A', 'B', 'C'],

'baz': [1, 2, 3, 4, 5, 6],

'zoo': ['x', 'y', 'z', 'q', 'w', 't']})

Print(df**.**pivot**(**index**=**'foo'**,** columns**=**'bar'**,** values**=**'baz'**))**

Output:

bar A B C

foo

one 1 2 3

two 4 5 6

13Ans: The astype() method we can impose a new data type to an existing column or all columns of a pandas data frame.

Example:

Import pandas as pd

df = pd.DataFrame({

   'DayNo': [1, 2, 3, 4, 5,6,7],

   'Name': ['Sun', 'Mon', 'Tue', 'Wed', 'Thu','Fri','Sat'],

   'Qty': [2.6, 5, 11.8, 2, 5.6,0,0.25]})

print(df.dtypes)

#Convert to string data type

df\_str = df.astype(str)

# Verify the conversion

print("\*\*\*After Conversion\*\*\*")

print(df\_str.dtypes)

output: DayNo       int64

Name       object

Qty     float64

dtype: object

\*\*\*After Conversion\*\*\*

DayNo    object

Name     object

Qty     object

dtype: object

14Ans: sort\_values() method sorts a data frame in Ascending or Descending order of passed Column.

Example: Sort Dataframe rows based on a single column

import pandas as pd

students **=** [('Ankit', 22, 'Up', 'Geu'),

           ('Ankita', 31, 'Delhi', 'Gehu'),

           ('Rahul', 16, 'Tokyo', 'Abes'),

           ('Simran', 41, 'Delhi', 'Gehu'),

           ('Shaurya', 33, 'Delhi', 'Geu'),

           ('Harshita', 35, 'Mumbai', 'Bhu' ),

           ('Swapnil', 35, 'Mp', 'Geu'),

           ('Priya', 35, 'Uk', 'Geu'),

           ('Jeet', 35, 'Guj', 'Gehu'),

           ('Ananya', 35, 'Up', 'Bhu')

            ]

  details **=** pd.DataFrame(students, columns **=**['Name', 'Age',

                                           'Place', 'College'],

                        index **=**[ 'b', 'c', 'a', 'e', 'f',

                                'g', 'i', 'j', 'k', 'd'])

# Sort the rows of dataframe by 'Name' column

rslt\_df **=** details.sort\_values(by **=** 'Name')

# Sort Dataframe rows based on a multiple columns.

rslt\_df **=** details.sort\_values(by **=** ['Name', 'Age'])

# Sort Dataframe rows based on columns in Descending Order.

rslt\_df **=** details.sort\_values(by **=** 'Name', ascending **=** False)

15Ans: copy() function is used to create a copy of a pandas dataframe

The copy() function takes a single parameter deep as an argument. This parameter has two possible values, True and False.

When deep=True, (as it is by default), a new object is created with a copy of the original (calling) object’s data and indices. Changes made to this new object (the deep copy) **will not be reflected** in the original object.

On the other hand, when deep=False, a new object is created without copying the original (calling) objects data and indices, just the references to the data and the indices are copies. Changes made to this new object (the shallow copy) **will be reflected** in the original object.

import pandas as pd

df = pd.DataFrame({

"Roll Number": [1, 2, 3],

"Name": ["Dwight", "Jim", "Pam"]

})

Print(df)

# create a deep copy

df1 = df.copy() # deep copy is created by default

print(df1)

# create a shallow copy

df2 = df.copy(deep=False)

# display the dataframe

Print(df2)

16Ans: import pandas as pd

df1 **=** pd.DataFrame({'Name':['Raju', 'Rani', 'Geeta', 'Sita', 'Sohit'],

                    'Marks':[80, 90, 75, 88, 59]})

Filt=(df[‘Marks’]>80 & df[‘Marks’]<=90)

Print(df[Filt])

17Ans: By using mean() we can calculate the mean of a column in a pandas dataframe.

import pandas as pd

df1 **=** pd.DataFrame({'Name':['Raju', 'Rani', 'Geeta', 'Sita', 'Sohit'],

                    'Marks':[80, 90, 75, 88, 59]})

Print(df1[‘Marks’].mean())

18Ans: By using std() we can calculate the standard deviation of a column in a pandas dataframe.

import pandas as pd

df1 **=** pd.DataFrame({'Name':['Raju', 'Rani', 'Geeta', 'Sita', 'Sohit'],

                    'Marks':[80, 90, 75, 88, 59]})

Print(df1[‘Marks’].std())

19Ans: We can use the .corr() method to get the correlation between two columns in Pandas.

import pandas as pd

df = pd.DataFrame(

{

"x": [5, 2, 7, 0],

"y": [4, 7, 5, 1],

"z": [9, 3, 5, 1]

}

)

col1, col2 = "x", "y"

corr = df[col1].corr(df[col2])

print(corr)

20Ans: using [], loc & iloc we can select specific columns in a dataframe using their labels.

import pandas as pd

employees = [('Stuti', 28, 'Varanasi', 20000),

('Saumya', 32, 'Delhi', 25000),

('Aaditya', 25, 'Mumbai', 40000),

('Saumya', 32, 'Delhi', 35000),

('Saumya', 32, 'Delhi', 30000),

('Saumya', 32, 'Mumbai', 20000),

('Aaditya', 40, 'Dehradun', 24000),

('Seema', 32, 'Delhi', 70000)

]

df = pd.DataFrame(employees,

columns =['Name', 'Age',

'City', 'Salary'])

Print(df)

#using [] to select a single column

Print(df[‘City’])

#using [] to select multiple columns

Print(df[‘Name’,’City’])

#using loc to select single column

df.set\_index("Name", inplace **=** True)

print(df.loc["Stuti"])

#using loc to select multiple columns

print(df.loc[["Stuti", "Seema"]])

#using loc Select multiple rows and particular columns.

Print(df.loc[["Stuti", "Seema"], ["City", "Salary"]])

#using loc to Select all the rows with some particular columns.

Print(df.loc[:, ["City", "Salary"]])

#using iloc to select a single column

Print(df.iloc[2])

#using iloc to select multiple columns

Print(df.iloc[[2, 3, 5]])

#using iloc to Select multiple rows with some particular columns.

Print(df.iloc[[2, 3, 5], [0, 1]])

#using iloc to Select all the rows with some particular columns.

Print(df.iloc[:, [0, 1]])

21Ans: Using loc[] and iloc[] we can select specific rows in a dataframe using their indexes

import pandas as pd

employees = [('Stuti', 28, 'Varanasi', 20000),

('Saumya', 32, 'Delhi', 25000),

('Aaditya', 25, 'Mumbai', 40000),

('Saumya', 32, 'Delhi', 35000),

('Saumya', 32, 'Delhi', 30000),

('Saumya', 32, 'Mumbai', 20000),

('Aaditya', 40, 'Dehradun', 24000),

('Seema', 32, 'Delhi', 70000)

]

df = pd.DataFrame(employees,

columns =['Name', 'Age',

'City', 'Salary'])

Print(df)

#using loc to select single column

df.set\_index("Name", inplace **=** True)

print(df.loc["Stuti"])

#using loc to select multiple columns

print(df.loc[["Stuti", "Seema"]])

#using loc Select multiple rows and particular columns.

Print(df.loc[["Stuti", "Seema"], ["City", "Salary"]])

#using loc to Select all the rows with some particular columns.

Print(df.loc[:, ["City", "Salary"]])

#using iloc to select a single column

Print(df.iloc[2])

#using iloc to select multiple columns

Print(df.iloc[[2, 3, 5]])

#using iloc to Select multiple rows with some particular columns.

Print(df.iloc[[2, 3, 5], [0, 1]])

#using iloc to Select all the rows with some particular columns.

Print(df.iloc[:, [0, 1]])

22Ans: sort\_values() method sorts a data frame in Ascending or Descending order of passed Column.

Example: Sort Dataframe rows based on a single column

import pandas as pd

students **=** [('Ankit', 22, 'Up', 'Geu'),

           ('Ankita', 31, 'Delhi', 'Gehu'),

           ('Rahul', 16, 'Tokyo', 'Abes'),

           ('Simran', 41, 'Delhi', 'Gehu'),

           ('Shaurya', 33, 'Delhi', 'Geu'),

           ('Harshita', 35, 'Mumbai', 'Bhu' ),

           ('Swapnil', 35, 'Mp', 'Geu'),

           ('Priya', 35, 'Uk', 'Geu'),

           ('Jeet', 35, 'Guj', 'Gehu'),

           ('Ananya', 35, 'Up', 'Bhu')

            ]

  details **=** pd.DataFrame(students, columns **=**['Name', 'Age',

                                           'Place', 'College'],

                        index **=**[ 'b', 'c', 'a', 'e', 'f',

                                'g', 'i', 'j', 'k', 'd'])

# Sort the rows of dataframe by 'Name' column

rslt\_df **=** details.sort\_values(by **=** 'Name')

# Sort Dataframe rows based on a multiple columns.

rslt\_df **=** details.sort\_values(by **=** ['Name', 'Age'])

# Sort Dataframe rows based on columns in Descending Order.

rslt\_df **=** details.sort\_values(by **=** 'Name', ascending **=** False)

23Ans: import pandas as pd

Df=pd.DataFrame({‘firstName’:[‘Praveen’,’Pranay’,’Sathvika’],

‘lastName’:[‘Nakkanaboina’,’Nallagasha’,’Kummari’],

‘Age’:[23,19,21]

})

Df[‘fullName’]=Df[‘firstName’]+Df[‘lastName’]

Print(Df)

24Ans: using drop\_duplicates() method we can remove duplicates from a dataframe

import pandas as pd

boxes = {'Color': ['Green','Green','Green','Blue','Blue','Red','Red','Red'],

'Shape':['Rectangle','Rectangle','Square','Rectangle','Square','Square','Square','Rectangle'] }

df = pd.DataFrame(boxes, columns = ['Color', 'Shape'])

df\_duplicates\_removed = df.drop\_duplicates()

print(df\_duplicates\_removed)

#To remove the duplicates on a specific column

df\_duplicates\_removed = df.drop\_duplicates(subset=['Color'])

print(df\_duplicates\_removed)

25Ans: The main difference between loc and iloc is that**loc is label-based** (you need to specify the row and column labels) while iloc is integer-position based (you need to specify the row and column by the integer position values, which start with 0).

|  |
| --- |
|  |