**PANDAS ASSIGNMENT**

**Q1. How do you load a CSV file into a Pandas DataFrame?**

**#reading data in pandas**

df = pd.read\_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")

df

**Q2. How do you check the data type of a column in a Pandas DataFrame?**

**# checking the datatypes of the columns**

df.dtypes

**Q3. How do you select rows from a Pandas DataFrame based on a condition?**

Selecting all the rows from the given dataframe in which ‘Percentage’ is greater than 80 using loc[].

# importing pandas

import pandas as pd

record = {

'Name': ['Ankit', 'Amit', 'Aishwarya', 'Priyanka', 'Priya', 'Shaurya' ],

'Age': [21, 19, 20, 18, 17, 21],

'Stream': ['Math', 'Commerce', 'Science', 'Math', 'Math', 'Science'],

'Percentage': [88, 92, 95, 70, 65, 78]}

# create a dataframe

dataframe = pd.DataFrame(record, columns = ['Name', 'Age', 'Stream', 'Percentage'])

print("Given Dataframe :\n", dataframe)

# selecting rows based on condition

rslt\_df = dataframe.loc[dataframe['Percentage'] > 80]

print('\nResult dataframe :\n', rslt\_df)

**Selecting all the rows from the given dataframe in which ‘Stream’ is present in the options list using basic method.**

# importing pandas

import pandas as pd

record = {

'Name': ['Ankit', 'Amit', 'Aishwarya', 'Priyanka', 'Priya', 'Shaurya' ],

'Age': [21, 19, 20, 18, 17, 21],

'Stream': ['Math', 'Commerce', 'Science', 'Math', 'Math', 'Science'],

'Percentage': [88, 92, 95, 70, 65, 78]}

# create a dataframe

dataframe = pd.DataFrame(record, columns = ['Name', 'Age', 'Stream', 'Percentage'])

print("Given Dataframe :\n", dataframe)

options = ['Math', 'Commerce']

# selecting rows based on condition

rslt\_df = dataframe[dataframe['Stream'].isin(options)]

print('\nResult dataframe :\n', rslt\_df)

**Q4. How do you rename columns in a Pandas DataFrame?**

import pandas as pd

data1 = {'Name':['Jai', 'Anuj', 'Jai', 'Princi',

'Gaurav', 'Anuj', 'Princi', 'Abhi'],

'Age':[27, 24, 22, 32,

33, 36, 27, 32],

'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj',

'Jaunpur', 'Kanpur', 'Allahabad', 'Aligarh'],

'Qualification':['Msc', 'MA', 'MCA', 'Phd',

'B.Tech', 'B.com', 'Msc', 'MA']}

df = pd.DataFrame(data1)

df

#Before renaming

df.columns

rename = df.rename(columns = {'Age' : 'age', 'Address': 'address'}, inplace = True)

#AFTER RENAMING

df.columns

**O/P:**

Index(['Name', 'age', 'address', 'Qualification'], dtype='object')

Index(['Name', 'age', 'address', 'Qualification'], dtype='object')

**Q5. How do you drop columns in a Pandas DataFrame?**

**#Dropping multiple columns**

df.drop(['age', 'address'], axis=1, inplace=True)

df

**#using index**

**# Remove two columns as index base**

df.drop(df.columns[[0, 1]], axis=1, inplace=True)

**Q6. How do you find the unique values in a column of a Pandas DataFrame?**

df['Name'].unique()

array(['Jai', 'Anuj', 'Princi', 'Gaurav', 'Abhi'], dtype=object)

**Q7. How do you find the number of missing values in each column of a Pandas DataFrame?**

**# Create a DataFrame object from list of tuples**

**# with columns and indices**.

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

'Place', 'College'],

index =['a', 'b', 'c', 'd', 'e',

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

**# show the boolean dataframe**

print(" \nshow the boolean Dataframe : \n\n", details.isnull())

show the boolean Dataframe :

Name Age Place College

a False False False False

b False True False True

c False False False False

d False False False False

e False True False False

f False False False True

g False False True False

i True False False False

j False False False False

k True True True True

**# Count total NaN at each column in a DataFrame**

print(" \nCount total NaN at each column in a DataFrame : \n\n",

details.isnull().sum())

Count total NaN at each column in a DataFrame :

Name 2

Age 3

Place 2

College 3

dtype: int64

**# Count total NaN at each row in a DataFrame**

for i in range(len(details.index)) :

print(" Total NaN in row", i + 1, ":",

details.iloc[i].isnull().sum())

Total NaN in row 1 : 0

Total NaN in row 2 : 2

Total NaN in row 3 : 0

Total NaN in row 4 : 0

Total NaN in row 5 : 1

Total NaN in row 6 : 1

Total NaN in row 7 : 1

Total NaN in row 8 : 1

Total NaN in row 9 : 0

Total NaN in row 10 : 4

**# Count total NaN in a DataFrame**

print(" \nCount total NaN in a DataFrame : \n\n",

details.isnull().sum().sum())

Count total NaN in a DataFrame :

10

**Q8. How do you fill missing values in a Pandas DataFrame with a specific value?**

**Filling missing values using fillna(), replace() and interpolate()**

**# dictionary of lists**

dict = {'First Score':[100, 90, np.nan, 95],

'Second Score': [30, 45, 56, np.nan],

'Third Score':[np.nan, 40, 80, 98]}

df = pd.DataFrame(dict)

df

**# filling missing value using fillna()**

df.fillna(0)

**# filling a missing value with**

**# previous ones**

df.fillna(method ='pad')

**Code #3: Filling null value with the next ones**

df.fillna(method ='bfill')

**# filling a null values using fillna() for specific column**

df["First Score"].fillna("NOT AVAIALBLE", inplace = True)

**#using replace()**

**# will replace Nan value in dataframe with value -99**

df.replace(to\_replace = np.nan, value = -99)

**Q9. How do you concatenate two Pandas DataFrames?**

# First DataFrame

df1 = pd.DataFrame({'id': ['A01', 'A02', 'A03', 'A04'],

'Name': ['ABC', 'PQR', 'DEF', 'GHI']})

df1

# Second DataFrame

df2 = pd.DataFrame({'id': ['B05', 'B06', 'B07', 'B08'],

'Name': ['XYZ', 'TUV', 'MNO', 'JKL']})

df2

frames = [df1, df2]

result = pd.concat(frames)

display(result)

**Q10. How do you merge two Pandas DataFrames on a specific column?**

dataframe1 = { 'Company name' : ['TATA', 'CTS', 'DELL'],

'Standard' : [1,2,3] }

df\_new = pd.DataFrame(dataframe1)

df\_new

dataframe2 = { 'Company name' : ['TATA', 'CTS', 'DELL', 'HCL'],

'Return' : [71,82,93,100] }

df\_new1 = pd.DataFrame(dataframe2)

df\_new1

**#merging based on coumn**

print(pd.merge(df\_new, df\_new1, on = 'Company name'))

**Q11. How do you group data in a Pandas DataFrame by a specific column and apply an aggregation function?**

**# GroupBy**

data1 = {'Name':['Jai', 'Anuj', 'Jai', 'Princi',

'Gaurav', 'Anuj', 'Princi', 'Abhi'],

'Age':[27, 24, 22, 32,

33, 36, 27, 32],

'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj',

'Jaunpur', 'Kanpur', 'Allahabad', 'Aligarh'],

'Qualification':['Msc', 'MA', 'MCA', 'Phd',

'B.Tech', 'B.com', 'Msc', 'MA']}

df = pd.DataFrame(data1)

df

df.groupby('Name')

print(df.groupby('Name').groups)

**#using aggregate function**

df.groupby(['Name']).sum()

**Q12. How do you pivot a Pandas DataFrame?**

pandas.pivot(index, columns, values) function produces pivot table based on 3 columns of the DataFrame. Uses unique values from index / columns and fills with values.

# Create a simple dataframe

# importing pandas as pd

import pandas as pd

# creating a dataframe

df = pd.DataFrame({'A': ['John', 'Boby', 'Mina'],

'B': ['Masters', 'Graduate', 'Graduate'],

'C': [27, 23, 21]})

Df

# values can be an object or a list

df.pivot('A', 'B', 'C')

# value is a list

df.pivot(index ='A', columns ='B', values =['C', 'A'])

**Q13. How do you change the data type of a column in a Pandas DataFrame?**

**#changing all coumns datatype to string**

# importing pandas as pd

import pandas as pd

# sample dataframe

df = pd.DataFrame({

'A': [1, 2, 3, 4, 5],

'B': ['a', 'b', 'c', 'd', 'e'],

'C': [1.1, '1.0', '1.3', 2, 5]})

# converting all columns to string type

df = df.astype(str)

print(df.dtypes)

**Change column type in pandas using dictionary and DataFrame.astype()**

We can pass any Python, Numpy, or Pandas datatype to change all columns of a Dataframe to that type, or we can pass a dictionary having column names as keys and datatype as values to change the type of selected columns.

# sample dataframe

df = pd.DataFrame({

'A': [1, 2, 3, 4, 5],

'B': ['a', 'b', 'c', 'd', 'e'],

'C': [1.1, '1.0', '1.3', 2, 5]})

# using dictionary to convert specific columns

convert\_dict = {'A': int,

'C': float

}

df = df.astype(convert\_dict)

print(df.dtypes)

**Q14. How do you sort a Pandas DataFrame by a specific column?**

**Sort Dataframe rows based on a single column.**

# import pandas library as pd

import pandas as pd

# List of Tuples

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')

]

# Create a DataFrame object from

# list of tuples with columns

# and indices.

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')

# show the resultant Dataframe

rslt\_df

**#Sort based on multiple rows**

# if duplicate value is present in 'Name' column

# then sorting will be done according to 'Age' column

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

**#Sorting based on name in descending order**

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

# show the resultant Dataframe

rslt\_df

**Q15. How do you create a copy of a Pandas DataFrame?**

The copy() method returns a copy of the DataFrame.

By default, the copy is a "deep copy" meaning that any changes made in the original DataFrame will NOT be reflected in the copy.

**Note:** With the parameter deep=False, it is only the reference to the data (and index) that will be copied, and any changes made in the original will be reflected in the copy, and, any changes made in the copy will be reflected in the original.

**SYNTAX:** dataframe.copy(deep)

Syntax: DataFrame.copy(deep=True)

When deep=True (default), a new object will be created with a copy of the calling object’s data and indices. Modifications to the data or indices of the copy will not be reflected in the original object (see notes below).

Step 1) Let us first make a dummy data frame, which we will use for our illustration

Step 2) Assign that dataframe object to a variable

Step 3) Make changes in the original dataframe to see if there is any difference in copied variable

#Create Series

s = pd.Series([3,4,5],['earth','mars','jupiter'])

k = pd.Series([1,2,3],['earth','mars','jupiter'])

#Create DataFrame df from two series

df = pd.DataFrame({'mass':s,'diameter':k})

df

**Now, let’s assign the dataframe df to a variable and perform changes:**

#Assign df to variable\_copy

variable\_copy = df

print(variable\_copy)

#Update the value of mass of earth in original dataframe

df['mass']['earth']=8

print(variable\_copy)

Here, we can see that if we change the values in the original dataframe, then the data in the copied variable also changes. To overcome this, we use DataFrame.copy()

**Let us see this, with examples when deep=True(default ):**

res = df.copy(deep=True)

print(res)

**Q16. How do you filter rows of a Pandas DataFrame by multiple conditions?**

# import module

import pandas as pd

# assign data

dataFrame = pd.DataFrame({'Name': [' RACHEL ', ' MONICA ', ' PHOEBE ',

' ROSS ', 'CHANDLER', ' JOEY '],

'Age': [30, 35, 37, 33, 34, 30],

'Salary': [100000, 93000, 88000, 120000, 94000, 95000],

'JOB': ['DESIGNER', 'CHEF', 'MASUS', 'PALENTOLOGY',

'IT', 'ARTIST']})

# display dataframe

display(dataFrame)

**Method 1: Using loc**

Here we will get all rows having Salary greater or equal to 100000 and Age < 40 and their JOB starts with ‘D’ from the dataframe. Print the details with Name and their JOB. For the above requirement, we can achieve this by using loc. It is used to access single or more rows and columns by label(s) or by a boolean array. loc works with column labels and indexes.

# import module

import pandas as pd

# assign data

dataFrame = pd.DataFrame({'Name': [' RACHEL ', ' MONICA ', ' PHOEBE ',

' ROSS ', 'CHANDLER', ' JOEY '],

'Age': [30, 35, 37, 33, 34, 30],

'Salary': [100000, 93000, 88000, 120000, 94000, 95000],

'JOB': ['DESIGNER', 'CHEF', 'MASUS', 'PALENTOLOGY',

'IT', 'ARTIST']})

# filter dataframe

display(dataFrame.loc[(dataFrame['Salary']>=100000) & (dataFrame['Age']< 40) & (dataFrame['JOB'].str.startswith('D')),

['Name','JOB']])

**Q17. How do you calculate the mean of a column in a Pandas DataFrame?**

Use mean() function to find the mean of all the observations over the index axis.

# importing pandas as pd

import pandas as pd

# Creating the dataframe

df = pd.DataFrame({"A":[12, 4, 5, 44, 1],

"B":[5, 2, 54, 3, 2],

"C":[20, 16, 7, 3, 8],

"D":[14, 3, 17, 2, 6]})

# Print the dataframe

Df

Let’s use the dataframe.mean() function to find the mean over the index axis.

# Even if we do not specify axis = 0,

# the method will return the mean over

# the index axis by default

df.mean(axis = 0)

**or**

# skip the Na values while finding the mean

df.mean(axis = 1, skipna = True)

**Q18. How do you calculate the standard deviation of a column in a Pandas DataFrame?**

import pandas as pd

my\_data = {'Name':pd.Series(['Tom','Jane','Vin','Eve','Will']),'Age':pd.Series([45, 67, 89, 12, 23]),'value':pd.Series([8.79,23.24,31.98,78.56,90.20])}

print("The dataframe is :")

my\_df = pd.DataFrame(my\_data)

print(my\_df)

print("The standard deviation of column 'Age' is :")

print(my\_df['Age'].std())

print("The standard deviation of column 'value' is :")

print(my\_df['value'].std())

**Q19. How do you calculate the correlation between two columns in a Pandas DataFrame?**

Correlation is used to summarize the strength and direction of the linear association between two quantitative variables. It is denoted by r and values between -1 and +1. A positive value for r indicates a positive association, and a negative value for r indicates a negative association.

By using corr() function we can get the correlation between two columns in the dataframe.

**SYNTAX:** dataframe[‘first\_column’].corr(dataframe[‘second\_column’])

Example 1: Python program to get the correlation among two columns

# import pandas module

import pandas as pd

# create dataframe with 3 columns

data = pd.DataFrame({

"column1": [12, 23, 45, 67],

"column2": [67, 54, 32, 1],

"column3": [34, 23, 56, 23]

}

)

# display dataframe

print(data)

# correlation between column 1 and column2

print(data['column1'].corr(data['column2']))

# correlation between column 2 and column3

print(data['column2'].corr(data['column3']))

# correlation between column 1 and column3

print(data['column1'].corr(data['column3']))

It is also possible to get element-wise correlation for numeric valued columns using just corr() function.

**Syntax:** dataset.corr()

**Example 2: Get the element-wise correlation**

# import pandas module

import pandas as pd

# create dataframe with 3 columns

data = pd.DataFrame({

"column1": [12, 23, 45, 67],

"column2": [67, 54, 32, 1],

"column3": [34, 23, 56, 23]

}

)

# get correlation between element wise

print(data.corr())

**Q20. How do you select specific columns in a DataFrame using their labels?**

df[['Name', 'Age']]

**Q21. How do you select specific rows in a DataFrame using their indexes?**

df[1:4]

//

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

new = df.loc["Anuj"]

new

**Q22. How do you sort a DataFrame by a specific column?**

**By sort\_values()**

data = {'name': ['Somu', 'Kiku', 'Amol', 'Lini'],

'physics': [68, 74, 77, 78],

'chemistry': [84, 56, 73, 69],

'algebra': [78, 88, 82, 87]}

#create dataframe

df\_marks = pd.DataFrame(data)

#sort dataframe in ascending

sorted\_df = df\_marks.sort\_values(by='algebra')

print(sorted\_df)

#in descending

sorted\_df = df\_marks.sort\_values(by='algebra', ascending=False)

print(sorted\_df)

**Q23. How do you create a new column in a DataFrame based on the values of another column?**

**#titanic dataset**

**# creating new column**

df['new\_col'] = df['Survived'] + df['Pclass']

df.head()

**Q24. How do you remove duplicates from a DataFrame?**

data = {

"A": ["TeamA", "TeamB", "TeamB", "TeamC", "TeamA"],

"B": [50, 40, 40, 30, 50],

"C": [True, False, False, False, True]

}

df = pd.DataFrame(data)

display(df.drop\_duplicates())

**Example 1: Removing rows with the same First Name**

In the following example, rows having the same First Name are removed and a new data frame is returned.

# making data frame from csv file

data = pd.read\_csv("employees.csv")

# sorting by first name

data.sort\_values("First Name", inplace=True)

# dropping ALL duplicate values

data.drop\_duplicates(subset="First Name",

keep=False, inplace=True)

# displaying data

data

**Q25. What is the difference between .loc and .iloc in Pandas?**

These are used in slicing data from the Pandas DataFrame. They help in the convenient selection of data from the DataFrame in Python. They are used in filtering the data according to some conditions.

**# creating a sample dataframe**

data = pd.DataFrame({'Brand': ['Maruti', 'Hyundai', 'Tata',

'Mahindra', 'Maruti', 'Hyundai',

'Renault', 'Tata', 'Maruti'],

'Year': [2012, 2014, 2011, 2015, 2012,

2016, 2014, 2018, 2019],

'Kms Driven': [50000, 30000, 60000,

25000, 10000, 46000,

31000, 15000, 12000],

'City': ['Gurgaon', 'Delhi', 'Mumbai',

'Delhi', 'Mumbai', 'Delhi',

'Mumbai', 'Chennai', 'Ghaziabad'],

'Mileage': [28, 27, 25, 26, 28,

29, 24, 21, 24]})

# displaying the DataFrame

display(data)

**LOC()**

The loc() function is label based data selecting method which means that we have to pass the name of the row or column which we want to select. This method includes the last element of the range passed in it, unlike iloc(). loc() can accept the boolean data unlike iloc(). Many operations can be performed using the loc() method like

**Example 1:**

Selecting data according to some conditions

# selecting cars with brand 'Maruti' and Mileage > 25

display(data.loc[(data.Brand == 'Maruti') & amp

(data.Mileage & gt

25)])

**2)Selecting a range of rows from the DataFrame**

# selecting range of rows from 2 to 5

display(data.loc[2: 5])

**3)Updating the value of any column**

# updating values of Mileage if Year < 2015

data.loc[(data.Year & lt

2015), ['Mileage']] = 22

display(data)

**ILOC()**

The iloc() function is an indexed-based selecting method which means that we have to pass an integer index in the method to select a specific row/column. This method does not include the last element of the range passed in it unlike loc(). iloc() does not accept the boolean data unlike loc(). Operations performed using iloc() are:

**Example 1:**

Selecting rows using integer indices

# selecting 0th, 2th, 4th, and 7th index rows

display(data.iloc[[0, 2, 4, 7]])

**2) Selecting a range of columns and rows simultaneously**

# selecting rows from 1 to 4 and columns from 2 to 4

display(data.iloc[1: 5, 2: 5])