PANDAS TUTORIAL

In [ ]:
```
import numpy as np
import pandas as pd
```

Series

- A Pandas Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.)

In [ ]:
```
series1=pd.Series(list('abcde'))
series1
```

In [ ]:
```
# series from array
names=np.array(['adam','derek','amy','rosa'])
names_series=pd.Series(names)
names_series
```

In [ ]:
```
# series from dictionary
dictionary = {'A': 10, 'B': 20, 'C': 30}
dict_series = pd.Series(dictionary)
print(dict_series)
```

In [ ]:
```
# scalar series
scalar_series=pd.Series(5,index=['a','b','c','d'])
scalar_series
```

accessing elements in a series

In [ ]:
```
# using index
dict_series[0]
```

In [ ]:
```
dict_series[0:3]
```

In [ ]:
```
dict_series[1:2]
```

In [ ]:
```
# using name
dict_series.loc['A']
```

In [ ]:
```
# using position
dict_series.iloc[0]
```

vectorized operations in series

- Vectorization in Python is a programming technique that allows you to perform operations on entire arrays at once, instead of having to loop through each element of the array individually

In [ ]:
```
vector_series1 = pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])
vector_series2 = pd.Series([10,20,30,40,50],index=['a','b','c','d','e'])
print(vector_series1,"\n",vector_series2)
```

In [ ]:
```
# index value has to be same for addition
vector_series1 + vector_series2
```

In [ ]:
```
vector_series3=pd.Series([1,2,3,4,5],index=['s','b','f','d','g'])
vector_series1 + vector_series3
```

Dataframes

- A DataFrame is a two-dimensional data structure in Python that is used to store and manipulate data.

In [ ]:
```
# creating dataframe from list
lst = [['tom', 25], ['krish', 30],
       ['nick', 26], ['juli', 22]]
df = pd.DataFrame(lst, columns =['Name', 'Age'])
print(df)
```

In [ ]:
```
# creating dataframe from dictionary with lists
olympic_data= {'Hostcity':['London','Beijing','Athens','Sydney','Atlanta'],
                    'No of participating countries':[205,204,201,200,197]}
df_olympic_data = pd.DataFrame(olympic_data,index=[1,2,3,4,5])
df_olympic_data
```

In [ ]:
```
olympic_data_dict = {'London':{2021:205},'Beijing':{2008:204}}
df_olympic_data_dict = pd.DataFrame(olympic_data_dict)
df_olympic_data_dict
```

In [ ]:
```
# creating dataframe from dict of series
olympic_series_participation = pd.Series([205,204,201,200,197],index=[2012,
olympic_series_country = pd.Series(['London','Beijing','Athens','Sydney','A
                                    index=[2012,2008,2004,2000,1996])
df_olympic_series = pd.DataFrame({'No of participating Countries':olympic_s
df_olympic_series
```

In [ ]:
```
# creating dataframe from ndarray
array=np.array([210,2008,2004,2000,1996])
dict={'year':array}
df=pd.DataFrame(dict)
df
```

viewing the dataframe

In [ ]:
```
df_olympic_data
```

In [ ]:
```
df_olympic_data.Hostcity
```

In [ ]:
```
# shows statistical summary of columns with numeric values
df_olympic_data.describe()
```

In [ ]:
```
# viewing first 2 rows
df_olympic_data.head(2)
```

In [ ]:
```
# viewing last 3 rows
df_olympic_data.tail(3)
```

In [ ]:
```
# view index of dataset
df_olympic_data.index
```

In [ ]:
```
# view columns of dataset
df_olympic_data.columns
```

selecting data

In [ ]:
```
# select data for host city
df_olympic_data['Hostcity']
```

In [ ]:
```
# select by label
df_olympic_data.loc[2] # 2 is the index
```

In [ ]:
```
# select by positon
df_olympic_data.iloc[0:2] # first 2 rows
```

In [ ]:
```
# selection based on index value 3- row 1- column
df_olympic_data.iat[3,1]
```

In [ ]:
```
# select data element by condition
df_olympic_data[df_olympic_data['No of participating countries'] == 200]
```

Handling missing values

In [ ]:
```
series1 = pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])
series2 = pd.Series([10,20,30,40,50],index=['c','e','f','g','h'])
series3 = series1 + series2
```

In [ ]:
```
series3
```

In [ ]:
```
# drop NaN values from dataset
dropna_s=series3.dropna()
dropna_s
```

In [ ]:
```
# replacing NaN values with 0
fillna_s=series3.fillna(0)
fillna_s
```

In [ ]:
```
# fill NaN with 0 before addition
fillna_before= series1.add(series2, fill_value=0)
fillna_before
```

data operations

In [ ]:
```
# declare movie rating dataframe: ratings from 1 to 5 (star * rating)
df_movie_rating = pd.DataFrame({'movie 1':[5,4,3,3,2,1],'movie 2':[4,5,2,3,
df_movie_rating
```

In [ ]:
```
def movie_grade(rating):
    if rating==5:
        return 'A'
    if rating==4:
        return 'B'
    if rating==3:
        return 'C'
    if rating==2:
        return 'D'
    if rating==1:
        return 'E'
    else:
        return 'F'

movie_grade(5)
```

In [ ]:
```
df_movie_rating.map(movie_grade)
```

data operations with statistical function

In [ ]:
```
df_test_scores = pd.DataFrame({'Test1':[95,84,73,88,82,61], 'Test2':[74,85,
df_test_scores
```

In [ ]:
```
# find maximum score
df_test_scores.max()
```

In [ ]:
```
# getting inded of max
df_test_scores.idxmax()
```

In [ ]:
```
# finding mean
df_test_scores.mean()
```

In [ ]:
```
# finding standard deviation
df_test_scores.std()
```

group by function

In [ ]:
```
df_president_names = pd.DataFrame({'first':['George','Bill','Ronald','Jimmy
df_president_names
```

In [ ]:
```
# grouping on thr basis of first name
first_only = df_president_names.groupby('first')
```

In [ ]:
```
# getting the entries in one group
first_only.get_group('George')
```

In [ ]:
```
# sorting
df_president_names.sort_values('last')
```

standardizing the dataset

In [ ]:
```
df_test_scores
```

In [ ]:
```
def standardize_tests(test):
    return (test-test.mean())/test.std()

standardize_tests(df_test_scores)
```

Pandas data operations

In [ ]:
```
df_student_math = pd.DataFrame({'student':['Tom','Jack','Dan','Ram','Jeff',
df_student_science = pd.DataFrame({'student':['Tom','Ram','David'],'ID':[10
```

In [ ]:
```
# merge both data to form single dataframe with math & science data
# common rows
pd.merge(df_student_math,df_student_science)
```

In [ ]:
```
# merge with key on student
pd.merge(df_student_math,df_student_science, on='student')
```

In [ ]:
```
# merge with key as id and left join
pd.merge(df_student_math,df_student_science, on='ID',how='left').fillna('X'
```

In [ ]:
```
# concat data of both subjects
pd.concat([df_student_math, df_student_science],ignore_index=True)
```

In [ ]:
```
# new dataframe
df_student=pd.DataFrame({'student':['tom','ram','harry','nick','tom','ram']
df_student
```

In [ ]:
```
# to check for duplicates
df_student.duplicated()
```

In [ ]:
```
# dropping duplicates
df_student.drop_duplicates()
```

In [ ]:
```
# dropping duplicates with student as key
df_student.drop_duplicates('student')
```

Pandas SQL operations

In [ ]:
```
import sqlite3
```

In [ ]:
```
# create table
create_table = """CREATE TABLE student_score(Id INTEGER, Name VARCHAR(20),M
```

In [ ]:
```
# Execute SQL statement
execute_SQL = sqlite3.connect(':memory:')
execute_SQL.execute(create_table)
execute_SQL.commit()
```

In [ ]:
```
#prepare SQL query
SQL_query = execute_SQL.execute('select * from student_score')
```

In [ ]:
```
# fetch result from sqlite database
resultset = SQL_query.fetchall()
resultset #expty
```

In [ ]:
```
# records to be inserted in table
insert= [(10,'Jack',85,92),(29,'Tom',73,89),(65,'Ram',65.5,77),(5,'Steve',5
```

In [ ]:
```
# inserting in SQL table , ? is a placeholder for columns
insert_statement = "Insert into student_score values (?,?,?,?)"
execute_SQL.executemany(insert_statement, insert)
```

In [ ]:
```
# prepare query
SQL_query = execute_SQL.execute("select * from student_score")
# Fetch resultant for the query
resultset = SQL_query.fetchall()
resultset
```

In [ ]:
```
# Put records into a dataframe
df_student_records = pd.DataFrame(resultset, columns=[ 'ID','Name','Math','
df_student_records
```

In [ ]:
```
# another example
ct= """CREATE TABLE student_v(Id INTENGER, Name );"""
```

In [ ]:
```
ex=sqlite3.connect(':memory:')
ex.execute(ct)
ex.commit()
```

In [ ]:
```
q=ex.execute('select * from student_v')
result=q.fetchall()
result
```

In [ ]:
```
ins=[(1,'a'),(2,'b')]
ins_s="Insert into student_v values (?,?)"
ex.executemany(ins_s,ins)
```

In [ ]:
```
q1=ex.execute("SELECT * FROM student_v")
```

In [ ]:
```
re=q1.fetchall()
re
```

In [ ]:
```
df=pd.DataFrame(re, columns=['id','name'])
```

In [ ]:
```
df
```

In [ ]: