# MRA DAV PUBLIC SCHOOL

# INFORMATICS PRACTICES

## Code No-065

## CLASS-XII

### 2021-2022

# Blue Print:

| Unit No | Unit Name | Marks |
|---|---|---|
| 1 | Data Handling using Pandas and Data Visualization | 30 |
| 2 | Database Query using SQL | 25 |
| 3 | Introduction to Computer Networks | 7 |
| 4 | Societal Impacts | 8 |
| | Practical | 30 |
| | Total | 100 |

# Importing/Exporting Data between CSV Files/MYSQL and Pandas

By

Rajesh Verma

# Interface Python with MYSQL

# Database Programming

- Python supports different databases.
- Python database API specification has been defined to provide similarity between modules to access different databases. It is known as Python DB-API 2.0
- It enables code that is generally more portable across databases as all databases modules provide the same API.
- Module required to access database are to be download.

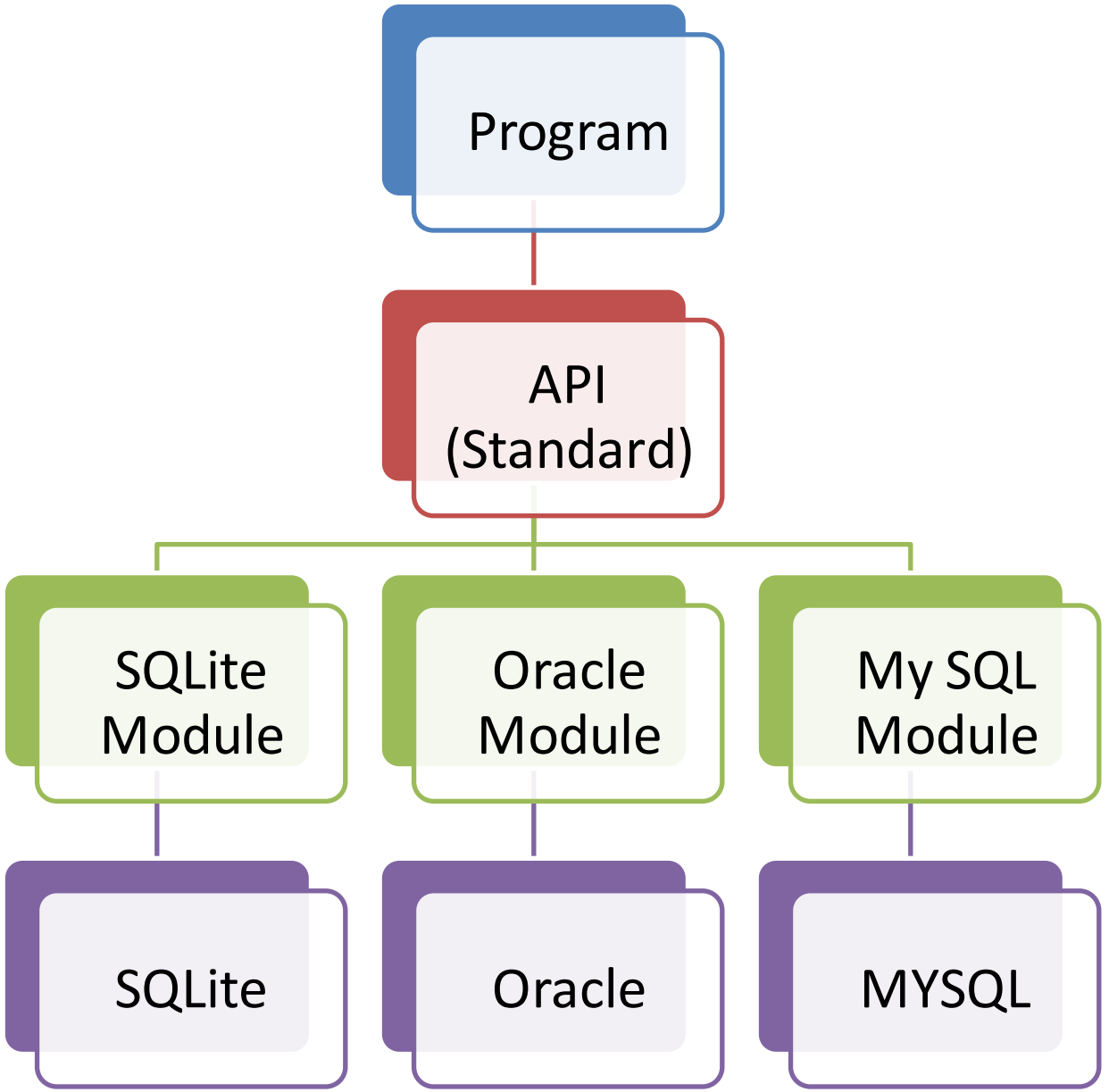
# Python Database API

1. GadFly
2. mSQL
3. MySQL
4. PostgreSQL
5. Microsoft SQL Server 2000
6. Informix
7. Interbase
8. Oracle
9. Sybase
10. SQLite

# DB API Module includes the following:

1. Importing the API module.
2. Acquiring a connection with the database.
3. Issuing SQL statements and stored procedures.
4. Closing the connection

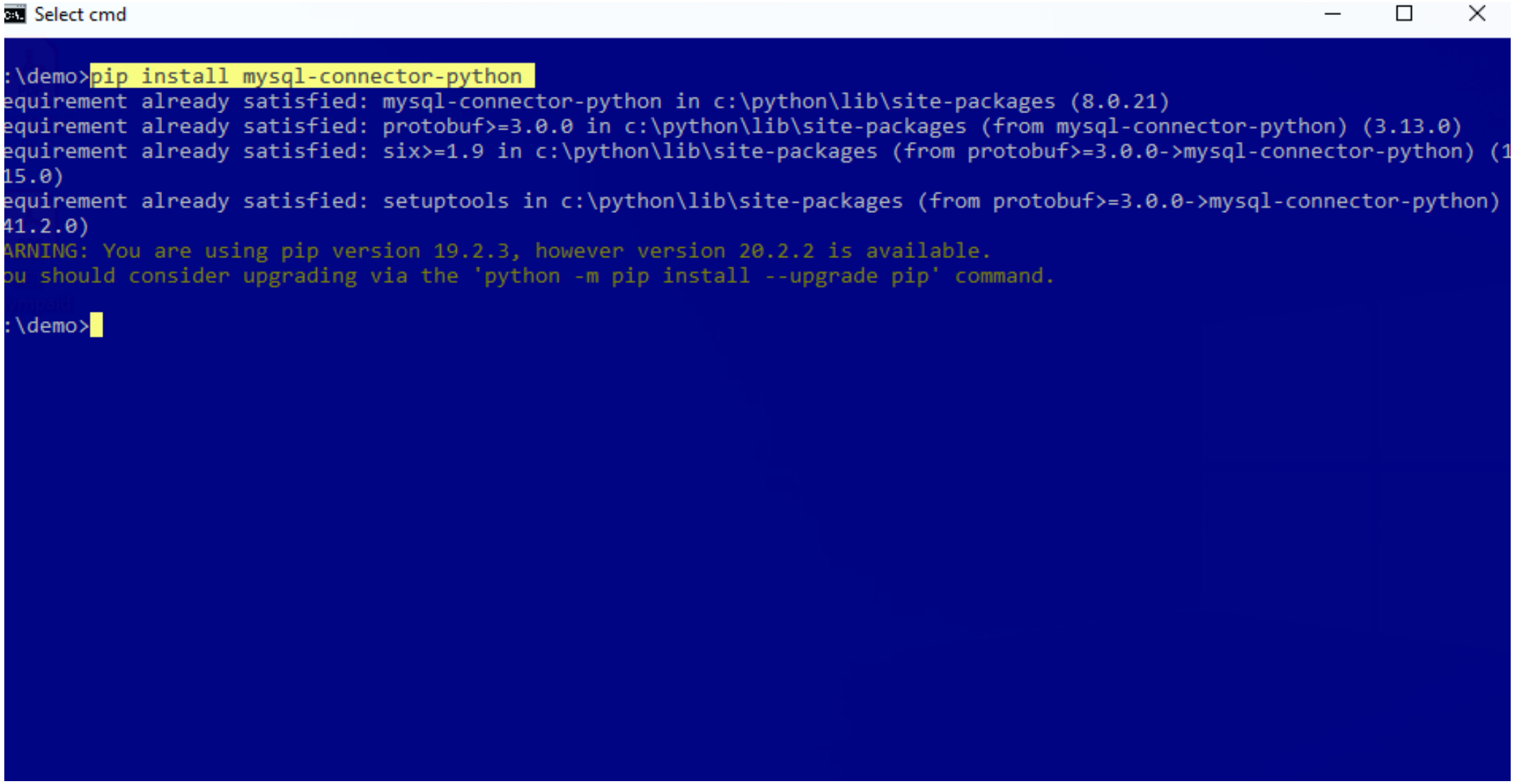
# Python use standard DB-API 2.0



# Connecting to MYSQL from python

Python needs a MySQL driver to access the MySQL database.
By using PIP we will install the driver "MySQL Connector".

# Command to install MYSQL Connector python is:



# Test MySQL Connector

To test if the installation was successful, create a Python page with the following content:

```
import mysql.connector
```

Another method to test MYSQL Connector on python prompt.

```
>>>import mysql.connector
```

Once you run this command , there is no error it means you have successfully installed.

# Create Connection

Start by creating a connection to the database.

Use the username and password from your MySQL database:

```
import mysql.connector

con= mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

print(con)
```

# Method connect()

1. This is a method in MYSQL module
2. It is used to establish a connection to database with given parameters.
3. Parameters vary from database to database.
4. It returns connection objects.

# Connection objects

Connection objects represents a connection to database.

| Method | Meaning |
| --- | --- |
| close | Closes connection. |
| commit | Commits pending changes in transaction to database |
| rollback | Causes the database to roll back to the start of any pending transaction. Closing a connection without committing the changes first will cause an implicit rollback to be performed. |
| cursor | Returns a cursor object using this connection or Execute the command using cursor |

# Example Create Connection

```
import mysql.connector as sql
con = sql.connect(host = "localhost",
                  user = "root",
                  password = "admin@123"

                  )
if con.is_connected():
    print("Successfully connected")
    con.close()
```

# Cursor Object

- Cursor represents a database cursor, which is used to manage the context of a fetch operation.
- Cursors created from the same connection are not isolated, i.e. any changes done to the database by a cursor are immediately visible by the cursors.

# Cursor Objects

| Method | Meaning |
|---|---|
| rowcount | This read-only attribute specifies the number of rows that the last execute() method retrieved or affected. |
| close() | Closes cursor |
| execute (Operations [,parameters]) | Prepare and execute a database operation. |
| executemany(operations, seq_of_parameters) | Prepare a database operation(query or command) and then execute it against all parameter sequence or mappings found in the sequence of parameters |
| fetchone() | Fetch the next row of a query result set, returning a single sequence, or None when no more data is available. |
| fetchmany([size = cursor.arraysize]) | Fetch the next set of rows of a query result, returning a sequence of sequence. An empty sequence is returned when no more rows are available. |
| Fetchall() | Fetchall (remaining) rows of a query result, returning them as a sequence of sequence. Note that the cursor's array size attribute can affect the performance of this operation. |

# Connecting python with MYSQL

1. Create database
2. Create table
3. Listing records
4. Adding records
5. Updating records
6. Deleting records

# Syntax to create database using python

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE mydatabase")
```

# Syntax to Create table using python

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address
VARCHAR(255))")
```

# Add records

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("John", "Highway 21")
mycursor.execute(sql, val)
mydb.commit()
print(mycursor.rowcount, "record inserted.")
```

# Listing Records

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

# Listing records using where clause in python program

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address ='Park Lane 38'"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

# Listing records using order by

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "SELECT * FROM customers ORDER BY name“
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

# Modifying records

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()

sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"

mycursor.execute(sql)
mydb.commit()
print(mycursor.rowcount, "record(s) affected")
```

# Deleting table

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "DROP TABLE customers"

mycursor.execute(sql)
```

# Deleting records

- import mysql.connector

```
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "DELETE FROM customers WHERE address = 'Mountain 21'"

mycursor.execute(sql)

mydb.commit()

print(mycursor.rowcount, "record(s) deleted")
```

# Importing Data From Mysql To DataFrame

```
import mysql.connector           # Import MySQL connector
import pandas as pd
 # Open a connection to a database
con1 = mysql.connector.connect(host='localhost', database='school12',
                               user='root', password='admin@123')
cursor1 = con1.cursor()            #Create a cursor object
#Create an empty dictionary
dict1={'name':(),'adm_no':(),'class':()}
#Create a dataframe using the empty dictionary
df1=pd.DataFrame(dict1)
#Execute the SQL query
cursor1.execute("select * from student;")
# fetch all rows of a result set and store in rs as a list of tuples
rs = cursor1.fetchall()
#Store the total number of rows affected in the last query in the variable n
n = cursor1.rowcount
print("Total number of records=", n)
# for loop to store all records in the dataframe df1
for i in range(n):
    df1.loc[i]=rs[i]
con1.close()                                  #Close connection object
cursor1.close()                             #Close cursor object
print(df1)                                      # Display the dataframe
```

```
Total number of records= 8
          name adm_no class
0  Rajesh Verma      1   XII
1         Raman    101   XII
2       Devansh    102   XII
3        Nupoor      2     8
4         Kavya      3     6
5         Taran      4     9
6       Lavanya      5     4
7          vyom      6     7
```

# Exporting Data From DataFrame to mysql

```
import pandas as pd
import mysql.connector as sql # Import MySQL connector
# Open a connection to a database
con1 = sql.connect(host='localhost', database='School12', user='root', password='admin@123')
cursor1 = con1.cursor()           #Create a cursor object
#Create a dictionary dict1
dict1={'name1':['Raman','Devansh'],'adm_no1':['103','104'],'SCLASS1':['XII','XII']}
#Create a dataframe df1
df1=pd.DataFrame(dict1)
#use iterrows() function to iterate over DataFrame rows
for index,row in df1.iterrows():
    #name=row[0]
    adm_no1= row[0]
    name1 = row[1]
    #price1=str(row[2])
    SCLASS1=row[2]
    query="insert into student values('"+ adm_no1+"', '"+ name1+ "',    '"+ SCLASS1 +"')"
    cursor1.execute(query)
con1.commit()
con1.close()                          #Close connection object
cursor1.close()                       #Close cursor object
```

```
mysql> select * from student;
+--------------+--------+--------+
| name         | adm_no | SCLASS |
+--------------+--------+--------+
| Rajesh Verma | 1      | XII    |
| Raman        | 101    | XII    |
| Devansh      | 102    | XII    |
| Raman        | 103    | XII    |
| Devansh      | 104    | XII    |
| Nupoor       | 2      | 8      |
| Kavya        | 3      | 6      |
| Taran        | 4      | 9      |
| Lavanya      | 5      | 4      |
| vyom         | 6      | 7      |
+--------------+--------+--------+
```

# INTERFACING CSV FILES WITH PANDAS

## INTRODUCTION

Data collection is one of the crucial steps in project development specially machine learning models. Mostly the data we collect is not clean and complete. It is not well structured or is simply incomplete. There are missing values, unbalanced data, column names without any meaning, deduped data, outdated data and other reasons. The predictive models made with these bad datasets will have low accuracy and low efficiency. Training a model with incorrect data will create a bias and give unpredictable results.

Data cleaning and preparation is the most consuming part of any data science project. However, there are many powerful tools to expedite this process. One of them is Pandas which is a widely used data analysis library for Python. It offers variety of functions to import CSV, Excel, SAS, and Stata files to name a few.

In this section we will learn how to use Pandas DataFrame to effectively analyse the data stored in CSV files.

# INTRODUCTION TO CSV FILES

**CSV** (Comma Separated Values) is a type of plain text file used to store tabular data. They are a convenient way to export data from spreadsheets and databases as well as import or use it in other programs. These files are saved with the **.csv** file extension.

A CSV file stores tabular data (numbers and text) in plain text. Here's what the structure of CSV file looks like:

```
SalesmanNo,Name,Sales,Comm
101,Aadarsh,250000,10000
102,Pranav,875100,12000
103,Swati,652300,11000
104,Mannat,594120,11000
105,Tushar,654320,11000
```

# ADVANTAGES OF CSV FILES

1. CSV is human readable and easy to edit manually.
2. CSV is simple to implement and parse.
3. CSV is processed by almost all existing applications.
4. CSV is easy to handle and smaller in size.
5. CSV is considered to be standard format.
6. CSV is compact and easy to generate.

# DIFFERENCE BETWEEN CSV AND EXCEL FILES

1. Excel and CSV both help store data in tabular format. Besides this commonality, there are many important differences in their respective features and usages.
2. CSV is a format for saving tabular information into a delimited text file with extension .csv whereas Excel is a spreadsheet that keeps files into its own proprietary format viz xls or xlsx.
3. CSV is a plain text format with a series of values separated by commas whereas Excel is a binary file that holds information about all the worksheets in a workbook.
4. CSV file can’t perform operations on data while Excel can perform operations on the data.
5. CSV files are faster and also consumes less memory whereas Excel consumes more memory while importing data.
6. CSV files can be opened with any text editor in windows while Excel files can't be opened with text editors.

# IMPORTING CSV FILES INTO DATAFRAMES

The read_csv() function of Pandas is used to read a csv(comma separated values) file and convert to pandas dataframe.

**SYNTAX**

DataFrame_Name=pd.read_csv(filename, [sep=" "], [header=0], [names], [index_col], [skiprows<n>])

Where

**filename:** name of the csv file along its path

**sep:** the separator between each field of data in csv file. it can be comma, tab, semicolon, space etc. the default value is comma.

**header:** to specify which line in your data is to be considered as header

**names**: to specify the column headers of our own choice

**index_col:** to specify columns from a csv file as index labels for dataframe

**skiprows<n>:** takes a number(n) which either indicates the number of rows to be skipped from the beginning or a list of row numbers to be skipped

# EXPORTING DATAFRAMES TO CSV FILES

Pandas provides to_csv() function to save data from dataframe to a CSV file.

**SYNTAX**

Dataframe_name.to_csv(filename, [sep=<character>], [na_rep=<string>])

Where

1. **filename:** name of the CSV file along with path
2. **sep:** specifies the separator character. It must be one character long. By default, comma is taken as a separator character.
3. **na_rep**: specifies a string to be written for missing/NaN values. By default, missing/NaN values are stored as empty strings.