**Day 1**

1. To run jupyter programs from you directory
2. Open that folder in explorer-> cmd 🡪 in cmd give command like jupyter notebook and press enter

**Day 2**

1. markdown :Esc M
2. code :Esc y
3. for pupup: .tab
4. help : shift tab
5. data.read\_csv / data.read\_excel to read the files
6. data.head() : displays the first five records
7. data.tail() : displays the last five records
8. data.head(2)/data.tail(2) displays first 2 / last 2 records
9. data.columns : displays the columns in the file
10. list(data.columns) : displays columns in the list format
11. data.info() : displays the fields and datatypes
12. data.describe() : displays count, std, mean, max, min, 25%, 50%, 75% for integer columns and for string columns it shows count, unique, top, freq

**Day :6**

1. data['column name']: displays records of the mentioned column
2. data[['address\_2', 'city', 'state\_province']] : you can give more than one column in a list
3. data[['address\_2', 'city', 'state\_province']].head() : displays first five rows of the mentioned columns
4. data1 = pd.read\_csv("addresses.csv", header = None, names = [1,2,3,4,5,6,7,8]) : gives numbers in place of column names
5. data1.to\_excel("myAddress.xlsx", index = False) : copy data of data1. in excel file named myaddress.xlsx
6. data.shape : displays size as no. of rows and columns
7. data.dtypes : displays all column names and its datatype
8. data.dtypes[data.dtypes == 'int64'] : displays the columns having datatype int64
9. list(data.dtypes[data.dtypes == 'int64'].index) : displays the columns in a list format having datatype int64
10. to filter the records from table based on some condition

data[data['Embarked']=='S']

1. For filtering data ( using add)

filter1 = data['Survived'] == 0

filter2 = data['Sex'] == 'male'

data[filter1 & filter2]

**or**

data[(data['Survived'] == 0) & (data['Sex'] == 'male')]

1. To print the count of filtered data

data[filter1 & filter2].count()

1. To print the count of filtered data

len(data[filter1 & filter2])

1. For filtering data ( using or)

len(data[(data['Survived']==0) | (data['Sex']=='male')])

1. To find unique values

data['Embarked'].unique()

1. To find unique values in list format

list(data['Embarked'].unique())

1. To find the count of unique values for a column

data['Embarked'].value\_counts()

it gives the output as

Embarked

S 644

C 168

Q 77

1. Find the passenger paid max fair

data[data['Fare'] == max(data['Fare'])]

1. Find the passenger paid max fair

data[data['Fare']==min(data['Fare'])]

1. To find the null values count for a column

data[data['Cabin'].isnull()==False]

1. To create a new column

data['newColumn']='India'

1. Assigning a value of another column to the new column

data['newColumn2'] =data['Fare']

1. To drop a column

data = data.drop(newColumn, axis=1) can be used only once, if we run it again it will give error as it is already deleted

To drop a column but to keep it to the same memory location

data.drop('newColumn', axis =1, inplace=True)

To delete more than one columns, we have to write in a list

data.drop(['newColumn2','newColumn3'], axis =1, inplace=True)

1. To access single record

data.iloc[3] // index location

1. To find the data wrt row and column position

data.iat[3,4] // row 3 column 4

1. To find the data wrt row and column position using column name

data.loc[2,'Ticket']

1. To filter the range of records by giving rows range and columns range

data.loc[50:55,'PassengerId':'Sex']

1. We can give random row number and column names in a list

data.loc[[1,2,3,8,4,90,80],['PassengerId','Survived','Cabin','Name','Fare']]

1. Filter based on condition

If age <20

Data[data[‘Age’] < 20]

If age >15 and <20

temp = data[(data['Age'] > 15) & (data['Age']<20)]

To find how many persons are there of age 18

len(data[data['Age']==18])

1. To set a column as an index

data.set\_index('Name')

1. To remove the rows having missing values

temp = data.dropna()

1. To place some value in a rows having null value

temp = data.fillna(555555)

temp =data.isna()

temp

1. Grouping

data.groupby('Cabin').sum('Fare')

data.groupby('Sex').sum('Fare')

data.groupby('Sex').sum()['Fare']

1. To ignore warnings

Import warnings

warnings.filterwarnings('ignore')

1. if want to apply some function on some column

data[‘new\_column’] = data[‘Fare’].apply(funname)

def apply\_group(x):

if x<=100:

return 'A'

elif x>100 and x<=200:

return 'B'

else:

return 'C'

data['Fare\_Group'] = data['Fare'].apply(apply\_group)

1. Create dataframe

Use dictionary

my\_dict={

"key1" : [1,2,3,4,9,8],

"key2" : [25,36,14,58,69,8],

"key3" : [96,85,41,25,36,96]

}

my\_dict

output:

{'key1': [1, 2, 3, 4, 9, 8],

'key2': [25, 36, 14, 58, 69, 8],

'key3': [96, 85, 41, 25, 36, 96]}

df1 = pd.DataFrame(my\_dict)

df1

key1 key2 key3

0 1 25 96

1 2 36 85

2 3 14 41

3 4 58 25

4 9 69 36

5 8 8 96

type(df1)

output

pandas.core.frame.DataFrame

**Day 7**

1.

import mysql.connector as conn

if it is giving error like

**ModuleNotFoundError** Traceback (most recent call last)

Cell **In[1], line 1**

**----> 1** **import** **mysql.connector** **as** **conn**

**ModuleNotFoundError**: No module named 'mysql'

Then try this command as

pip install mysql-connector-python

execute this command

then use this command

pip install mysql-connector

execute this command

after installing both this packages, restart the Kernel

2. **myconn = conn.connect(host='localhost', user='root', passwd='root')**

**cursor = myconn.cursor()**

If this is giving error like

**NotSupportedError**: Authentication plugin 'caching\_sha2\_password' is not supported

ALTER USER ‘bhagyalaxmi'@'your\_host' IDENTIFIED WITH 'mysql\_native\_password' BY 'your\_password';

If this not works, try this

UPDATE mysql.user SET Password=PASSWORD('1234') WHERE User=’bhagyalaxmi’;

If this not works, try this

ALTER USER 'bhagyalaxmi'@'localhost' IDENTIFIED BY ‘root’;

If this not works, try this

default\_authentication\_plugin=1234

version = mysql.connector.\_\_version\_\_

If this not works, try this

**pip install mysql-connector-python==8.0.32 –user # this works**

**restart the kernel**

3. query ='show tables'

**cursor.execute(query)**

**cursor.fetchall()**

when you want dataset use pd. read\_sql\_query()

4. query = "select \* from city"

data = pd.read\_sql\_query(query,myconn)

data

5. query = "select CountryCode,Language from countrylanguage"

data = pd.read\_sql\_query(query,myconn)

data.head(2)

6. Extract unique values from table column

query = "select distinct CountryCode from countrylanguage"

data = pd.read\_sql\_query(query,myconn)

data

check with

data['IsOfficial'].unique()

7. ‘in’ clause

=['Delhi', 'Mumbai', 'Gujarat','Haryana']

query = f"select \* from city where CountryCode ='IND' and Name in {tuple(cityList)}"

**day8**

* To find the records having null value

query = """

select \*

from customer

where first\_name is NULL

"""

data = pd.read\_sql\_query(query,myconn)

print(f"data.shape : {data.shape}")

data.head()

* To find the records having Is not null value

query = """

select \*

from customer

where first\_name is not NULL

"""

data = pd.read\_sql\_query(query,myconn)

print(f"data.shape : {data.shape}")

data.head()

* **Create table**

query ="""

create table if not exists student\_datawave(

attendance\_number INT,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

percentage DECIMAL(5,2),

std INT,

DIVISION VARCHAR(20)

)

"""

**myconn.commit()**

cursor.execute(query)

* **Drop multiple tables**

table\_to\_drop = ['test2', 'test3']

for table\_name in table\_to\_drop:

query = f"drop table if exists {table\_name}"

cursor.execute(query)

cursor.execute('show tables')

cursor.fetchall()

* **Limit**

select \* from student\_datawave limit 2

* **Aggregation**

cursor.execute('use world')

query = """

select max(population) from city

"""

data = pd.read\_sql\_query(query,myconn)

data

**similary use for min, avg, sum, count as**

* query = """

select min(population) from city

"""

* query = """

select count(\*) from city

"""

* query = """

select avg(population) from city

"""

* query = """

select sum(population) from city

"""

* **Like operator**

% : zero or one or multiple character

\_ : single character

* 1. Name starting with m

query = """

select \* from city where name like 'm%'

"""

data = pd.read\_sql\_query(query,myconn)

data

* 1. Name having 5 character starting with m

query = """

select \* from city where name like 'm\_\_\_\_'

"""

data = pd.read\_sql\_query(query,myconn)

data

* 1. **Name having second letter m**

query = """

select \* from city where name like '\_m%'

"""

data = pd.read\_sql\_query(query,myconn)

data

1. **in Operator**

**day9**

**Mongodb**

**For getting output in better formatting use**

from pprint import pprint

**Create connection**

from pymongo.mongo\_client import MongoClient

mongodb\_url\_key = "mongodb+srv://clarityai:N0lTZ4zedDxGbn9N@cluster0.qketnjc.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0&tlsCAFile=isrgrootx1.pem"

# Create a new client and connect to the server

client = MongoClient(mongodb\_url\_key)

# Send a ping to confirm a successful connection

try:

client.admin.command('ping')

print("Pinged your deployment. You successfully connected to MongoDB!")

except Exception as e:

print(e)

**Create Database**

database = client['clarityai']

database

**Create collection (table)**

collection = database['learn']

collection

**Insert document (Record) into collection**

data = {

"class name" : "Data science project",

"Topic name" : "Db : mysql, mongodb",

"today's date" : '03-03-2024'

}

collection.insert\_one(data)

**Insert multiple documents**

data = [{

"class name" : "web development fundamental",

"Topic name" : "html, css, js , react",

"fees" :50000,

"start date" : '1-jan-2024'

},{

"class name" : "machine learning fundamental",

"Topic name" : "supervised, unsupervised learning",

"fees" :50000,

"start date" : '5-jan-2024'

},{

"class name" : "cloud computing fundamental",

"Topic name" : "supervised, unsupervised learning",

"fees" :50000,

"start date" : '5-jan-2024'

},{

"class name" : "cyber secuirty fundamental",

"Topic name" : "network security , encryption, decryption",

"fees" :50000,

"start date" : '5-jan-2024'

}]

collection.insert\_many(data)

**Insert single document with different set of fields/columns**

data = {

"class name" : "cyber secuirty fundamental",

"Topic name" : "network security , encryption, decryption",

"fees" :50000,

"start date" : '5-jan-2024',

"instructor" : "pravin",

"teaching experience" : "5 years"

}

collection.insert\_one(data)

**insert document with less number of fields than existing in collection**

course\_list = [

{

"course\_name": 'data science',

"course\_instructor": 'john\_doe',

"start\_date": '01-01-2023',

"duration": 'three months',

"certificate": 'yes',

"prerequisite": 'programming knowledge',

"resume": 'yes',

"mockinterview": 'yes'

},

{

"course\_name": 'machine learning',

"course\_instructor": 'jane\_smith',

"start\_date": '15-02-2023',

"duration": 'four months',

"mode": 'in-person',

"certificate": 'yes',

"resume": 'no',

"mockinterview": 'yes'

},

{

"course\_name": 'deep learning',

"course\_instructor": 'mike\_jones',

"start\_date": '10-03-2023',

"duration": 'six months',

"mode": 'hybrid',

"certificate": 'yes',

"prerequisite": 'neural networks knowledge',

"resume": 'yes',

"mockinterview": 'no'

},

{

"course\_name": 'big data',

"course\_instructor": 'emma\_wilson',

"start\_date": '05-04-2023',

"duration": 'five months',

"mode": 'live',

"certificate": 'yes',

},

{

"course\_name": 'devops',

"course\_instructor": 'robert\_smith',

"start\_date": '20-05-2023',

"duration": 'three months',

"mode": 'remote',

"certificate": 'yes',

"prerequisite": 'version control knowledge',

"resume": 'no',

}

]

collection.insert\_many(course\_list)

### Extract the records from collection

record = collection.find()

for i in record:

print(i)

this will not give in neat format

so use **pprint**

record = collection.find()

for i in record:

pprint(i)

**Extract single column from collection**

record = collection.find({},{"class name"})

for i in record:

pprint(i)

**Extract multiple fields from collection**

record = collection.find({},{"class name","fees","certificate"})

for i in record:

pprint(i)

### Update document field based on single condition

* collection.update\_one({"class name" : "web development fundamental"},{"$set" :{"class name" : "Prompt Engineering" }})
* collection.update\_one({"class name" : "Prompt Engineering"},{"$set" :{"Topic name" : "LLM, Langchain, GuggingFace" }})

### Extract records with fees less than or equal to 50000

record = collection.find({"fees": {"$lt":50001}})

for i in record:

pprint(i)

### Delete record based on condition

collection.delete\_many({"fees" : 25000})

### Delete record based on multiple conditions

collection.delete\_many({ 'start date': '1-jan-2024', 'topic name': 'LLM, Langchain, GuggingFace'})

**day11**

exclude

include

plot

Dependent columns: column that we predict is called dependent columns

Independent columns: columns which calculate the sale price

Skewness

Handle Category and convert into a number

Handle skewness

Convert year to categorical data