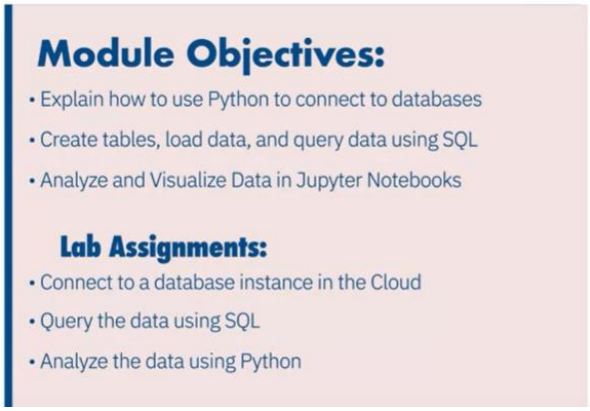
Module 4 - Accessing Databases with Python

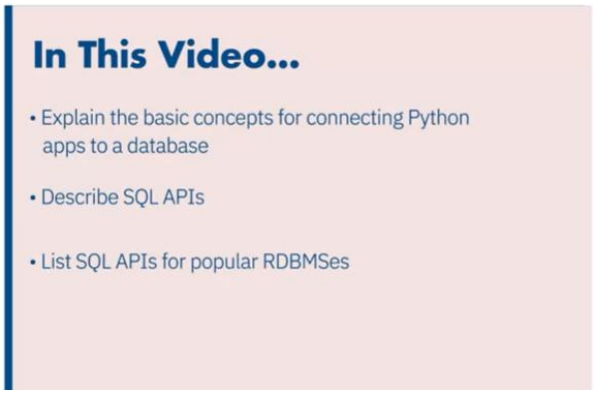


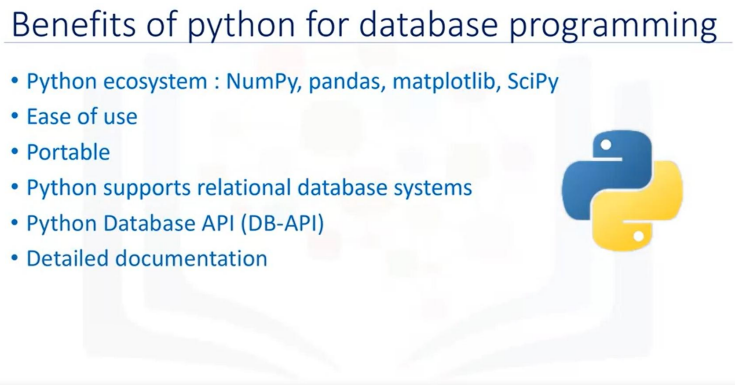
In this module you will learn the basic concepts related to using Python to connect to databases. In a Jupyter Notebook, you will create tables, load data, query data using SQL, and analyze data using Python.

Learning Objective:

- Demonstrate how to connect to a database from a Jupyter notebook
- Demonstrate how to create tables and insert data from Python
- Demonstrate how to perform simplified database access from Python using SQL magic
- Demonstrate writing SQL queries and retrieve result sets from Python
- Describe concepts related to accessing Databases using Python

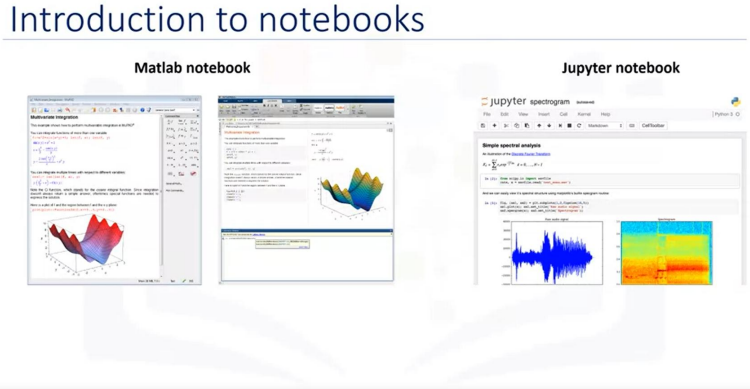
4.1 How to Access Databases Using Python

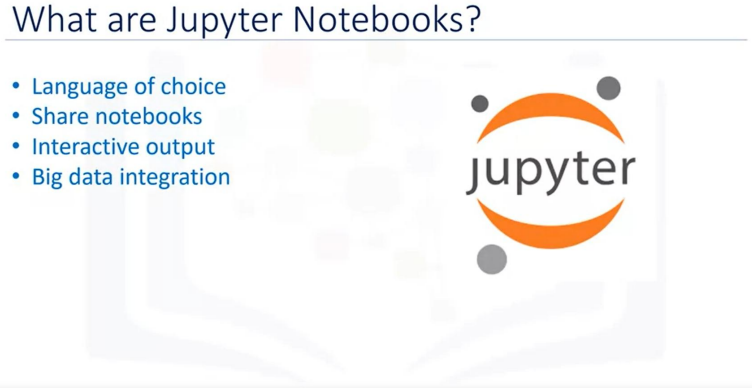




Let's quickly review some of the benefits of using Python, a popular **scripting language** for connecting to databases.

- The Python ecosystem is very rich and provides easy to use tools for data science. Some of the most popular packages are NumPy, pandas, matplotlib, and SciPy.
- Python is easy to learn and has a simple syntax. Due to its open source nature.
- Python has been ported to many platforms. All your python programs can work on any of these platforms without requiring any changes at all. If you are careful and avoid any system dependent features.
- Python supports relational database systems. Writing Python code to access databases is made easier by the presence of the Python database API. Commonly referred to as the DB API.
- And detailed documentation related to Python is easily available.





Notebooks are also very popular in the field of data science because they run in an environment that allows creation and sharing of documents that contain live code, equations, visualizations, and explanatory texts.

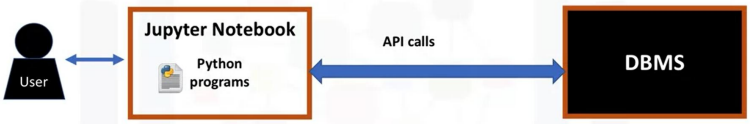
- A notebook interface is a virtual notebook environment used for programming.
- Examples of notebook interfaces include the Mathematica notebook, Maple worksheet, Matlab notebook, IPython Jupyter, R Markdown, Apache Zeppelin, Apache Spark notebook, and the Databricks cloud.

In this module, we will be using Jupyter notebooks. The Jupyter notebook is an open source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative texts. Here are some of the advantages of using Jupyter notebooks.

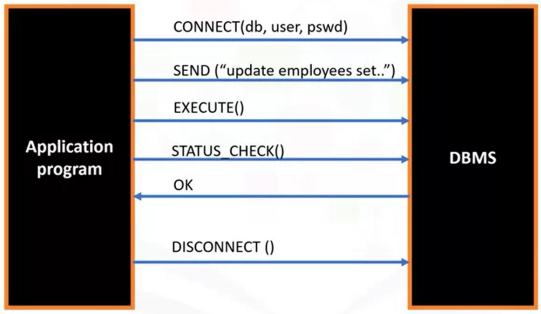
- Notebook support for over 40 programming languages including Python, R, Julia, and Scala.
- Notebooks can be shared with others by email, Dropbox, GitHub, and the Jupyter notebook viewer.
- Your code can produce rich interactive output HTML, images, videos, LaTex, and customized types.
- You can leverage big data tools such as Apache Spark from Python, R, and Scala, and explore that same data with pandas, scikit-learn, ggplot2, and TensorFlow.

总结:

Accessing databases using Python



What is a SQL API?



This is how a typical user accesses databases using Python code written on a Jupyter notebook, a **web based editor**. There is a mechanism by which the Python program communicates with the DBMS.

The Python code connects to the database using **API calls**. We will explain the basics of **SQL APIs** and **Python DB APIs**.

- An application programming interface is a set of functions that you can call to get access to some type of service.
- The SQL API consists of library function calls as an application programming interface, API, for the DBMS. To pass SQL statements to the DBMS, an application program calls functions in the API, and it calls other functions to retrieve query results and status information from the DBMS.

The basic operation of a typical SQL API is illustrated in the figure. The application program begins its database access with one or more API calls that connect the program to the DBMS. To send the SQL statement to the DBMS, the program builds the statement as a text string in a buffer and then makes an API call to pass the buffer contents to the DBMS. The application program makes API calls to check the status of its DBMS request and to handle errors. The application program ends its database access with an API call that disconnects it from the database.

APIs used by popular SQL-based DBMS systems

Application or Database	SQL API
MySQL	MySQL C API
PostgreSQL	psycopg2
IBM DB2	ibm_db
SQL Server	dblib API
Database access for Microsoft Windows OS	ODBC
Oracle	OCI
Java	JDBC

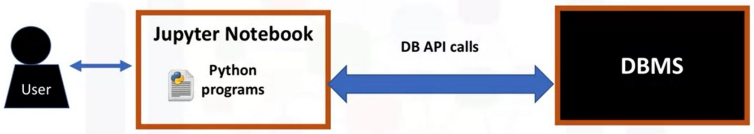
Now, lets learn basic concepts about some of the proprietary APIs used by popular **SQL-based DBMS systems**. Each database system has its own library. As you can see, the table shows a list of a few applications and corresponding SQL APIs.

- **MySQL C API** provides low level access to the MySQL client server protocol and enables C programs to access database contents.
- **psycopg2 API** connects Python applications in PostgreSQL databases.
- **ibm_db API** is used to connect Python applications to IBM DB2 databases.
- **dblib API** is used to connect to SQL server databases.
- **ODBC** is used for database access for Microsoft Windows OS.
- **OCI** is used by Oracle databases.
- **JDBC** is used by Java applications.

总结:

4.2 Writing code using DB-API

What is a DB-API?



- Python's standard API for accessing relational databases
- Allows a single program that to work with multiple kinds of relational databases
- Learn DB-API functions once, use them with any database

Benefits of using DB-API

- Easy to implement and understand
- Encourages similarity between the Python modules used to access databases
- Achieves consistency
- Portable across databases
- Broad reach of database connectivity from Python

Hello and welcome to writing code using DB-APIs. After completing this video, you will be able to explain the basic concepts related to the **Python DB-API** and **database cursors**. And also write code using DB-APIs. As we saw in the beginning of this module, the user writes Python programs using a Jupyter notebook.

- There is a mechanism by which the Python code communicates with the DBMS.
- The Python code connects to the database using **DB-API calls**. DB-API is Python's standard API for accessing relational databases. It is a standard that allows you to write a single program that works with multiple kinds of relational databases instead of writing a separate program for each one. So, if you learn the DB-API functions, then you can apply that knowledge to use any database with Python.

Here are some advantages of using the DB-API.

- It's easy to implement and understand.
- This API has been defined to encourage similarity between the Python modules that are used to access databases.
- It achieves consistency which leads to more easily understood modules.
- The code is generally more portable across databases,
- and it has a broader reach of database connectivity from Python.

Examples of libraries used by database systems to connect to Python applications

Database	DB API
IBM Db2	Ibm_db
Compose for MySQL	MySQL Connector/Python
Compose for PostgreSQL	psycopg2
Compose for MongoDB	PyMongo

Concepts of the Python DB API

Connection Objects

- Database connections
- Manage transactions

Cursor Objects

- Database Queries
- Scroll through result set
- Retrieve results

As we know, each database system has its own library. As you can see, the table shows a list of a few databases and corresponding DB-APIs to connect to Python applications.

- The IBM_db library is used to connect to an IBM DB2 database.
- The MySQL Connector/Python library is used to connect to a Compose for MySQL database.
- The psycopg2 library is used to connect to a Compose from PostgreSQL database.
- And finally, the PyMongo library is used to connect to a Compose for MongoDB database.

The two main concepts in the Python DB-API are **connection objects** and **query objects**.

- You use connection objects to connect to a database and manage your transactions.
- Cursor objects are used to run queries. You open a cursor object and then run queries. The cursor works similar to a cursor in a text processing system where you scroll down in your result set and get your data into the application. Cursors are used to scan through the results of a database.

The DB_API includes a connect constructor for creating a connection to the database. It returns a Connection Object, which is then used by the various connection methods.

总结:

What are Connection methods?

- .cursor()
- .commit()
- .rollback()
- .close()

What are cursor methods?

- .callproc()
- .execute()
- .executemany()
- .fetchone()
- .fetchmany()
- .fetchall()
- .nextset()
- .arraysize()
- .close()

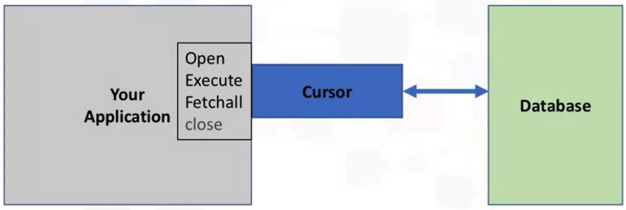
The DB_API includes a connect constructor for creating a connection to the database. It returns a Connection Object, which is then used by the various connection methods. These connection methods are:

- The cursor() method, which returns a new cursor object using the connection.
- The commit() method, which is used to commit any pending transaction to the database.
- The rollback() method, which causes the database to roll back to the start of any pending transaction.
- The close() method, which is used to close a database connection.

These objects represent a database cursor, which is used to manage the content of a **fetch operation**.

- Cursors created from the same connection are not isolated that is, any changes done to the database by a cursor are immediately visible by the other cursors.
- Cursors created from different connections can or cannot be isolated depending on how the transaction support is implemented.

What is a database cursor?



A database cursor is a control structure that enables traversal over the records in a database.

- It behaves like a file name or file handle in a programming language. Just as a program opens a file to access its contents, it opens a cursor to gain access to the query results. Similarly, the program closes a file to end its access and closes a cursor to end access to the query results.
- Another similarity is that just as file handle keeps track of the program's current position within an open file, a cursor keeps track of the program's current position within the query results.

Writing code using DB-API

```
from dbmodule import connect
#Create connection object
Connection =
connect('databasename',
'username', 'pswd')

#Create a cursor object
Cursor=connection.cursor()

#Run Queries
Cursor.execute('select *
from mytable')
Results=cursor.fetchall()

#Free resources
Cursor.close()
Connection.close()
```

Let's walk through a Python application that uses the DB-API to query a database.

- First, you import your database module by using the connect API from that module.
- To open a connection to the database, you use the connect constructor and pass in the parameters, that is, the database name, username, and password. The connect function returns connection object.
- After this, you create a cursor object on the connection object.
- The cursor is used to run queries and fetch results. After running the queries, using the cursor, we also use the cursor to fetch the results of the query.
- Finally, when the system is done running the queries, it frees all resources by closing the connection. Remember that it is always important to close connections to avoid unused connections taking up resources.

总结:

4.3 Connecting to a database using ibm_db API

Overview

At the end of this lesson, you will be able to:

- Describe the ibm_db API
- List the credentials required to connect to a database
- Connect to an IBM db2 database using Python

Hello, and welcome to connecting to a database using the ibm_db API. After completing this lesson, you will be able to

- understand the ibm_db API,
- as well as the credentials required to connect to a database using Python.
- We will also demonstrate how to connect to an IBM DB2 database using Python code written on a Jupyter notebook.

What is ibm_db?

- The ibm_db API provides a variety of useful Python functions for accessing and manipulating data in an IBM data server Database
- Ibm_db API uses the IBM Data Server Driver for ODBC and CLI APIs to connect to IBM DB2 and Informix

The ibm_db API provides a variety of useful Python functions for accessing and manipulating data in an IBM data server database, including functions for connecting to a database, preparing and issuing SQL statements, fetching rows from result sets, calling stored procedures, committing and rolling back transactions, handling errors and retrieving metadata.

The ibm_db API uses the IBM Data Server Driver for ODBC, and CLI APIs to connect to IBM, DB2, and Informix.

Identify database connection credentials

```
dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "BLUDB"            # e.g. "BLUDB"
dsn_hostname = "YourDb2Hostname" # e.g.: "dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net"
dsn_port = "50000"                # e.g. "50000"
dsn_protocol = "TCPIP"            # i.e. "TCPIP"
dsn_uid = "***************"        # e.g. "abc12345"
dsn_pwd = "***************"      # e.g. "7dBZ3wWt9XN6$o0J"
```

We import the ibm_db library into our Python application. Connecting to the DB2 requires the following information:

- a driver name,
- a database name,
- a host DNS name or IP address,
- a host port,
- a connection protocol,
- a user ID,
- and a user password.

总结:

Create a database connection

```
#Create database connection
dsn = (
    "DRIVER={{IBM DB2 ODBC DRIVER}};"
    "DATABASE={0};"
    "HOSTNAME={1};"
    "PORT={2};"
    "PROTOCOL=TCPIP;"
    "UID={3};"
    "PWD={4};").format(dsn_database, dsn_hostname, dsn_port, dsn_uid, dsn_pwd)

try:
    conn = ibm_db.connect(dsn, "", "")
    print ("Connected!")

except:
    print ("Unable to connect to database")
```

```
Connected!
```

Close the database connection

```
In [8]: ibm_db.close(conn)
Out[8]: True
```

Here is an example of creating a DB2 database connection in Python.

- We create a connection object DSN, which stores the connection credentials.
- The connect function of the ibm_db API will be used to create a non persistent connection. The DSN object is passed as a parameter to the connection function.
- If a connection has been established with the database, then the code returns connected, as the output. Otherwise, the output will be unable to connect to database.

Then we free all resources by closing the connection. Remember that it is always important to close connections so that we can avoid unused connections taking up resources. Thank you for watching this video.

总结:

4.4 Creating tables, loading data and querying data

Hello, and welcome to creating tables, loading data, and querying data. After completing this lesson, you will be able to

- understand basic concepts related to creating tables, loading data, and querying data using Python.
- demonstrate an example of how to perform these tasks using the IBM DB2 on Cloud database and Jupyter notebooks. For this example, we will be using DB2 as the database.

Connect to the database

```
import ibm_db
```

```
dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "BLUDB"            # e.g. "BLUDB"
dsn_hostname = "YourDb2Hostname" # e.g.: "dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net"
dsn_port = "50000"                # e.g. "50000"
dsn_protocol = "TCPIP"            # i.e. "TCPIP"
```

```
#Create database connection
dsn = (
    "DRIVER={{IBM DB2 ODBC DRIVER}};"
    "DATABASE={0};"
    "HOSTNAME={1};"
    "PORT={2};"
    "PROTOCOL=TCPIP;"
    "UID={3};"
    "PWD={4};").format(dsn_database, dsn_hostname, dsn_port, dsn_uid, dsn_pwd)

try:
    conn = ibm_db.connect(dsn, "", "")
    print ("Connected!")

except:
    print ("Unable to connect to database")
```

```
Connected!
```

Creating tables

Serial No	Model	Manufacturer	Engine Size	Class
A1234	Lonestar	International Trucks	Cummins ISX15	Class 8
B5432	Volvo VN	Volvo Trucks	Volvo D11	Heavy Duty Tractor Class 8
C5674	Kenworth W900	Kenworth Truck Company	Caterpillar C9	Class 8

ibm_db.exec_immediate()

The parameters for the function are:

- Connection
- Statement
- Options

We first obtain a connection resource by connecting to the database by using the connect method of the ibm_db api.
There are different ways of creating tables in DB2.

- One is using the Web console provided by DB2,
- the other option is to create tables from any SQL, R, or Python environments.

Let's take a look at how to create tables in DB2 from our Python application. Here is a sample table of a commercial Trucks database. Let's see how we can create the Trucks table in the DB2 using Python code. To create a table, we use the *ibm_db.exec_immediate function*. The parameters for the function are

- connection, which is a valid database connection resource returned from the *ibm_db.connect* or *ibm_db.pconnect function*.
- statement, which is a string that contains the SQL statement,
- and options which is an optional parameter that includes a dictionary that specifies the type of cursor to return for results sets.

总结:

Python code to create a table

```
stmt = ibm_db.exec_immediate(conn,
"CREATE TABLE Trucks(
serial_no varchar(20) PRIMARY KEY NOT NULL,
model VARCHAR(20) NOT NULL,
manufacturer VARCHAR(20) NOT NULL,
Engine_size VARCHAR(20) NOT NULL,
Truck_Class VARCHAR(20) NOT NULL) "
)
```

Here is the code to create a table called Trucks in Python.
We use the ibm_db.exec_immediate function of the ibm_db api.

The connection resource that was created is passed as the first parameter to this function.

The next parameter is the SQL statement, which is the create table query used to create the Trucks table.
The new table created will have five columns, serial_no will be the primary key.

Python code to insert data into the table

```
stmt = ibm_db.exec_immediate(conn,
"INSERT INTO Trucks(serial_no,
model,manufacturer,Engine_size, Truck_Class)
VALUES('A1234','Lonestar','International
Trucks','Cummins ISX15','Class 8');")
```

Now let's take a look at loading data. We use the ibm_db.exec_immediate function of the ibm_db api.

- The connection resource that was created is passed as the first parameter to this function.
- The next parameter is the SQL statement, which is the INSERT INTO query used to insert data in the Trucks table. A new row will be added to the Trucks table.

Insert more rows to the table

```
stmt = ibm_db.exec_immediate(conn,
"INSERT INTO Trucks(serial_no,model,manufacturer,Engine_size, Truck_Class)
VALUES('B5432','Volvo VN','Volvo Trucks','Volvo D11','Heavy Duty Class 8');")


stmt = ibm_db.exec_immediate(conn,
"INSERT INTO Trucks(serial_no,model,manufacturer,Engine_size, Truck_Class)
VALUES('C5674','Kenworth W900','Kenworth Truck Co','Caterpillar C9','Class 8');")
```

Similarly, we add more rows to the Trucks table using the ibm_db.exec_immediate function.

Python code to query data

```
In [10]: stmt = ibm_db.exec_immediate(conn,"SELECT * FROM Trucks")

         ibm_db.fetch_both(stmt)

Out[10]: {0: 'A1234',
          1: 'Lonestar',
          'MANUFACTURER': 'International Trucks',
          3: 'Cummins ISX15',
          'SERIAL_NO': 'A1234',
          'ENGINE_SIZE': 'Cummins ISX15',
          'MODEL': 'Lonestar',
          'TRUCK_CLASS': 'Class 8',
          2: 'International Trucks',
          4: 'Class 8'}
```

Now that your Python code has been connected to a database instance and the database table has been created and populated with data, let's see how we can fetch data from the Trucks table that we created on DB2 using Python code. We use the ibm_db.exec_immediate function of the ibm_db api.

- The connection resource that was created is passed as the first parameter to this function.
- The next parameter is the SQL statement, which is the select from table query.

The Python code returns the output, which shows the fields of the data in the Trucks table. You can check if the output returned by the select query shown is correct, by referring to the DB2 console.

Using pandas

```
In [19]: import pandas
         import ibm_db_dbi
         pconn = ibm_db_dbi.Connection(conn)
         df = pandas.read_sql('SELECT * FROM Trucks', pconn)
         df
```

Out[19]:

	SERIAL_NO	MODEL	MANUFACTURER	ENGINE_SIZE	TRUCK_CLASS
0	A1234	Lonestar	International Trucks	Cummins ISX15	Class 8
1	B5432	Volvo VN	Volvo Trucks	Volvo D11	Heavy Duty Class 8
2	C5674	Kenworth W900	Kenworth Truck Co	Caterpillar C9	Class 8

Let's look at how we can use pandas to retrieve data from the database tables.

Pandas is a popular Python library that contains high level data structures and manipulation tools designed to make data analysis fast and easy in Python.

We load data from the Trucks table into a data frame called df. A data frame represents a tabular spreadsheet like data structure containing an ordered collection of columns, each of which can be a different value type.

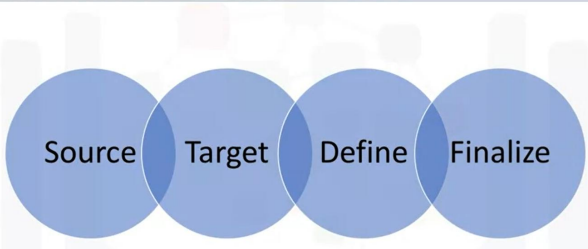
总结:

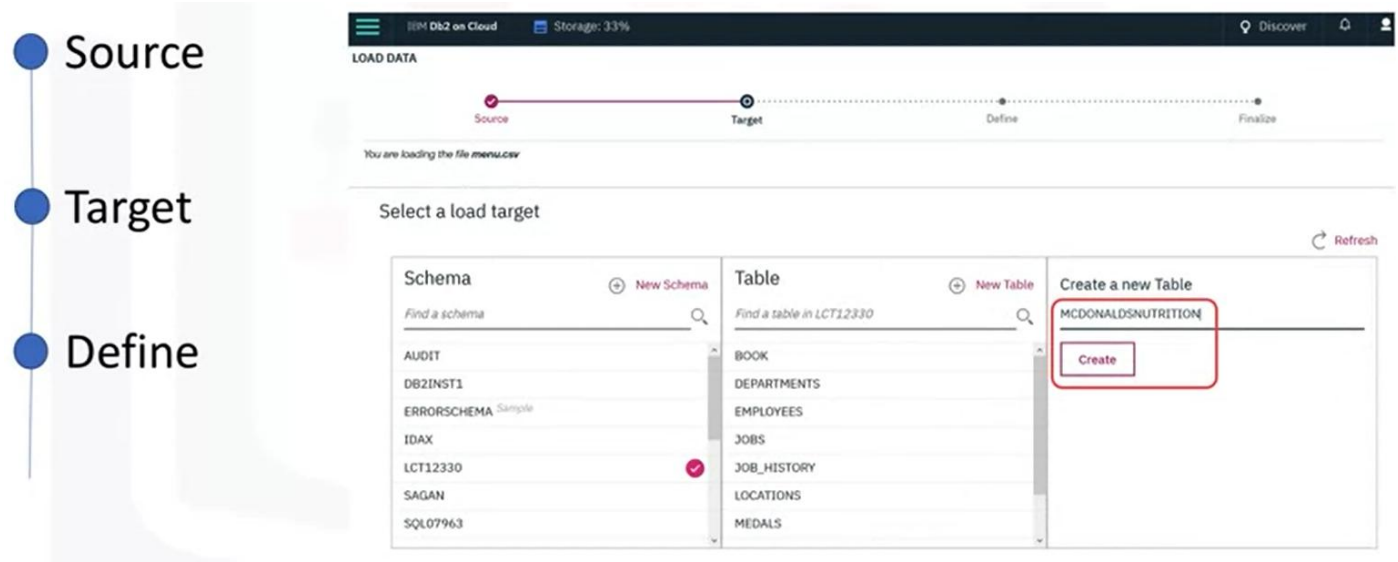
4.5 Analyzing data with Python

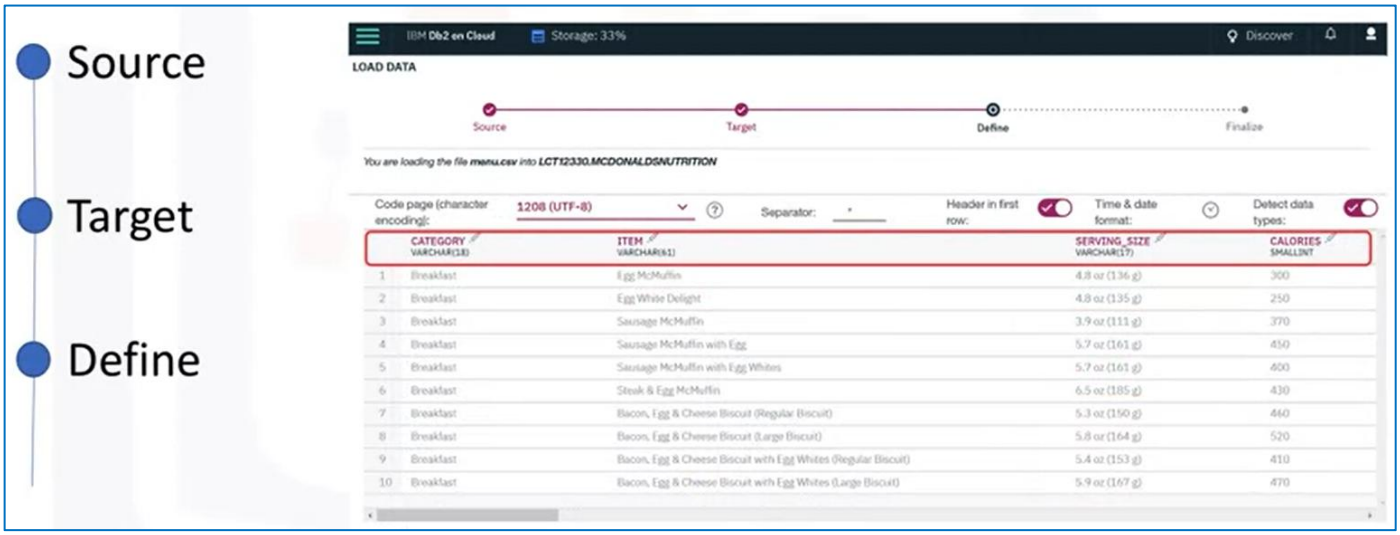
After completing this video, you will be able to understand basic concepts related to performing exploratory analysis on data. We will demonstrate an example of how to store data using the IBM Db2 on Cloud database, and then use Python to do some basic data analysis on this data.

In this video, we will be using the McDonald's menu nutritional facts data for popular menu items at McDonald's, while using Python to perform basic exploratory analysis. McDonald's is an American fast food company and the world's largest restaurant chain by revenue. Although McDonald's is known for fast food items such as hamburgers, French fries, soft drinks, milkshakes, and desserts, the company has added to its menu salads, fish, smoothies, and fruit. McDonald's provides nutrition analysis of their menu items to help you balance your McDonald's meal with other foods you eat. The data set used in this lesson has been obtained from the nutritional facts for McDonald's menu from Kaggle. We need to create a table on Db2 to store the McDonald's menu nutrition facts data set that we will be using. We will also be using the console provided by Db2 for this process.

Load csv file into DB2 on cloud



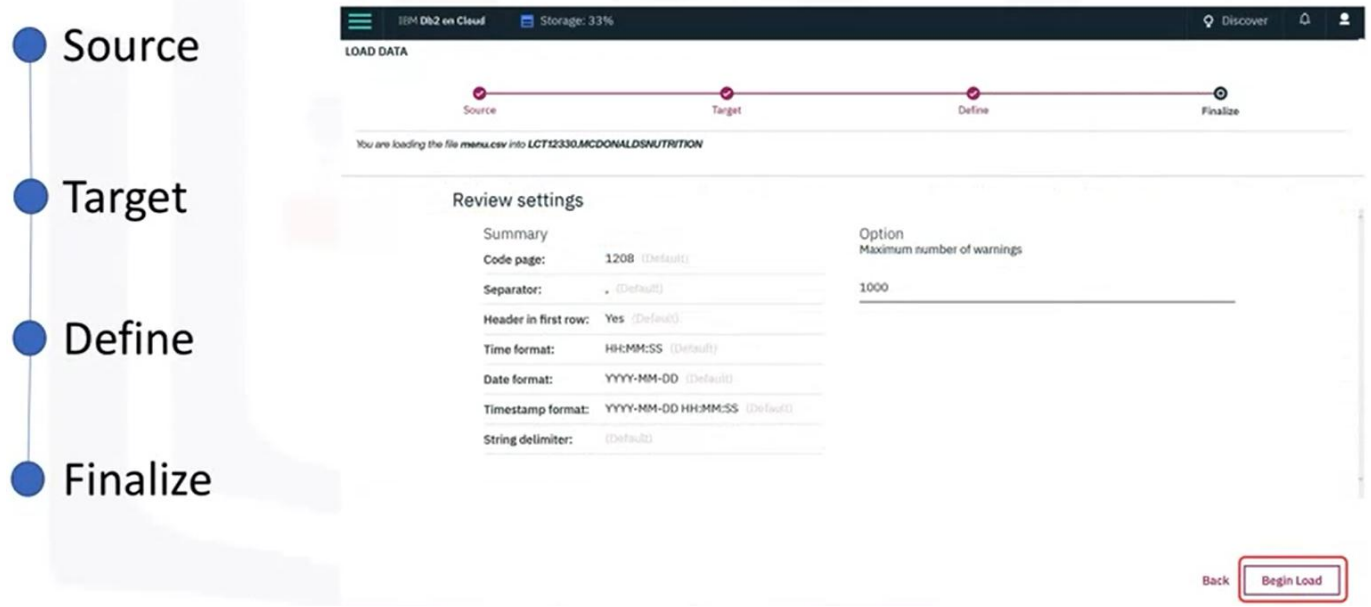


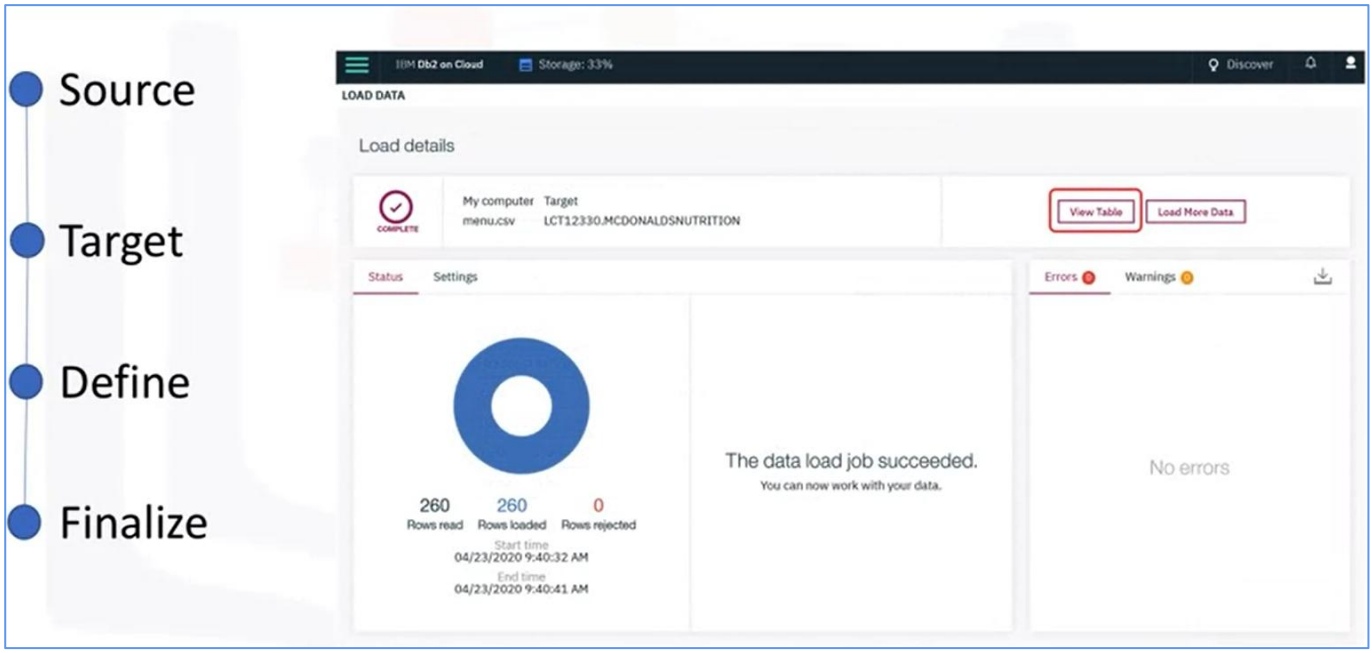


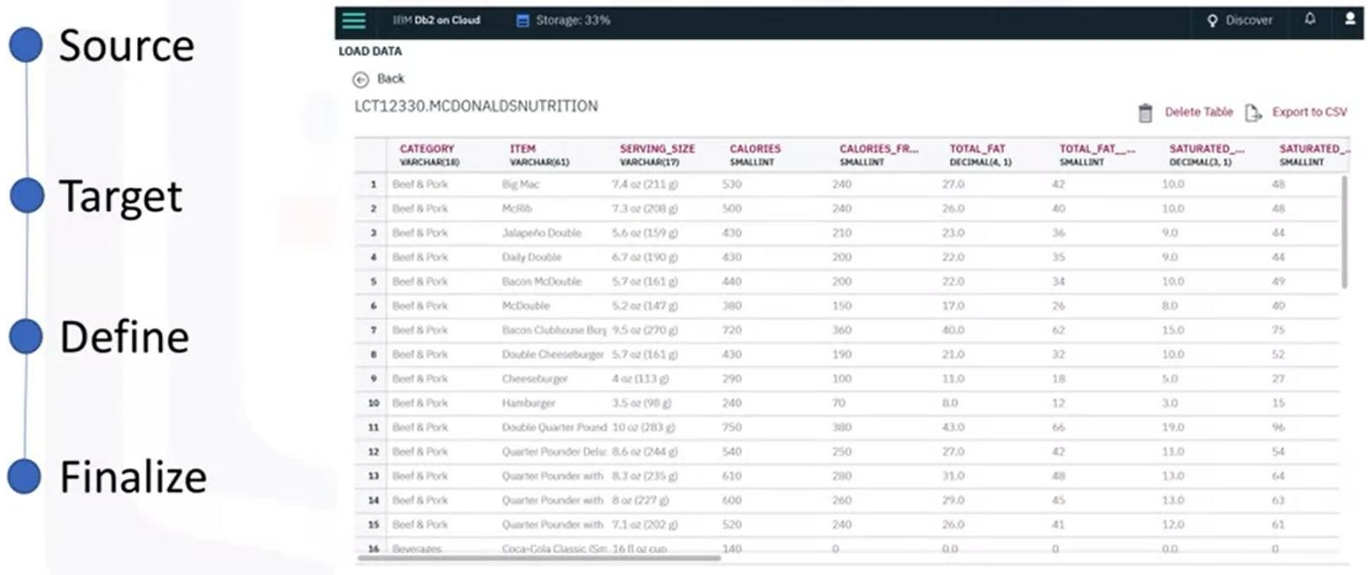
There are four steps involved in loading data into a table, source, target, define, and finalize.

1. We first load the spreadsheet into the Db2 using the console.
2. We then select the target schema, and then you will be given an option to load the data into an existing table or create a new table. When you choose to create a new table, you have the option to specify the table name.
3. Next, you will see a preview of the data where you can also define the columns and data types.

总结:







4. Review the settings and begin the load. When the loading is complete, you can see the statistics on the loaded data. Next, view the table to explore further.

总结:

Verify Loaded Data Using SQL

Mc Donalds nurtition

```
In [ ]: ### Verify Loaded Data Using SQL
```

```
In [7]: stmt = ibm_db.exec_immediate(conn,"SELECT count(*) FROM MCDONALDS_NUTRITION")

        ibm_db.fetch_both(stmt)

Out[7]: {0: '260', '1': '260'}
```

Db2 Warehouse allows you to analyze data using in-database analytics, APIs, RStudio or Python. The data has been loaded into our relational database. You can run Python scripts that retrieve data from and write data to a Db2 database. Such scripts can be powerful tools to help you analyze your data. For example, you can use them to generate statistical models based on data in your database, and to plot the results of these models. In this lesson, we will be using Python scripts that will be run within a Jupyter notebook. Now, after obtaining a connection resource, by connecting to the database, by using the connect method of the IBM_DB API, we use the SQL select query to verify the number of rows that have been loaded in the table created. The figure shows a snapshot of the output. The output obtained is 260 which is similar to the number of rows in the Db2 console.

Using pandas

Exploratory analysis using pandas

```
In [5]: import pandas
        import ibm_db_dbi
        pconn = ibm_db_dbi.Connection(conn)
        df = pandas.read_sql('SELECT * FROM MCDONALDS_NUTRITION', pconn)
        df
```

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Carbohydrates (% Daily Value)	Dietary Fiber	Di Fit (% Da Va
0	Breakfast	Egg McMuffin	4.8 oz (136 g)	300	120	13.0	20	5.0	25	0.0	...	31	10	4	17
1	Breakfast	Egg White Delight	4.8 oz (135 g)	250	70	8.0	12	3.0	15	0.0	...	30	10	4	17
2	Breakfast	Sausage McMuffin	3.9 oz (111 g)	370	200	23.0	35	8.0	42	0.0	...	29	10	4	17
3	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	450	250	28.0	43	10.0	52	0.0	...	30	10	4	17
4	Breakfast	Sausage McMuffin with Egg Whites	5.7 oz (161 g)	400	210	23.0	35	8.0	42	0.0	...	30	10	4	17
5	Breakfast	Steak & Egg McMuffin	6.5 oz (185 g)	430	210	23.0	36	9.0	46	1.0	...	31	10	4	18

View first few rows

```
import pandas
import ibm_db_dbi
pconn = ibm_db_dbi.Connection(conn)
df = pandas.read_sql('SELECT * FROM MCDONALDS_NUTRITION', pconn)
df.head()
```

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Carbohydrates (% Daily Value)	Dietary Fiber	Dietary Fiber (% Daily Value)	S
0	Breakfast	Egg McMuffin	4.8 oz (136 g)	300	120	13.0	20	5.0	25	0.0	...	31	10	4	17	3
1	Breakfast	Egg White Delight	4.8 oz (135 g)	250	70	8.0	12	3.0	15	0.0	...	30	10	4	17	3
2	Breakfast	Sausage McMuffin	3.9 oz (111 g)	370	200	23.0	35	8.0	42	0.0	...	29	10	4	17	2
3	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	450	250	28.0	43	10.0	52	0.0	...	30	10	4	17	2
4	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	400	210	23.0	35	8.0	42	0.0	...	30	10	4	17	2

Learn about your data

```
In [34]: df.describe(include='all')
```

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Ca (% Va
count	260	260	260	260.000000	260.000000	260.000000	260.000000	260.000000	260.000000	260.000000	...	260.000000	26
unique	9	260	107	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
top	Coffee & Tea	Nonfat Caramel Mocha (Large)	16 fl oz cup	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
freq	95	1	45	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
mean	NaN	NaN	NaN	368.269231	127.096154	14.165385	21.815385	6.007692	29.965385	0.203846	...	47.346154	15
std	NaN	NaN	NaN	240.269886	127.875914	14.205998	21.885199	5.321873	26.639209	0.429133	...	28.252232	9.
min	NaN	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.
25%	NaN	NaN	NaN	210.000000	20.000000	2.375000	3.750000	1.000000	4.750000	0.000000	...	30.000000	10
50%	NaN	NaN	NaN	340.000000	100.000000	11.000000	17.000000	5.000000	24.000000	0.000000	...	44.000000	15
75%	NaN	NaN	NaN	500.000000	200.000000	22.250000	35.000000	10.000000	48.000000	0.000000	...	60.000000	20
max	NaN	NaN	NaN	1880.000000	1060.000000	118.000000	182.000000	20.000000	102.000000	2.500000	...	141.000000	47

Now let's see how we can use Pandas to retrieve data from the database tables. We load data from the McDonalds_nutrition table into the data frame DF using the read_SQL method. The SQL select query and the connection object are passed as parameters to the read_SQL method.
We can view the first few rows of the data frame df that we created using the head method. Now it's time to learn about your data. Pandas methods are equipped with a set of common mathematical and statistical methods.
Let's use the describe method to view the summary statistics of the data in the data frame, then explore the output of the describe method. We see that there are 260 observations or food items in our data frame. We also see that there are nine unique categories of food items in our data frame. We can also see summary statistics information such as frequency, mean, median, standard deviation, et cetera for the 260 food items across the different variables. For example, the maximum value for total fat is 118.

总结:

Which food item has maximum sodium content?

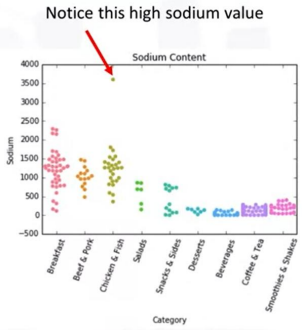
- Main source of sodium is table salt
- Average American eats 5 teaspoons/day
- Sodium mostly added during preparation
- Foods that don't taste salty may be high in sodium
- Sodium controls fluid balance in our bodies
- Too much sodium may raise blood pressure
- Target less than 2,000 milligrams/day

Which food item has maximum sodium content?

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

### Categorical scatterplots

plot = sns.swarmplot(x="Category", y='Sodium', data=df)
plt.setp(plt.get_xticklabels(), rotation=70)
plt.title('Sodium Content')
plt.show()
```



Which food item has maximum sodium content?

Code 1

```
In [17]: df['Sodium'].describe()

Out[17]: count     260.000000
         mean      495.750000
         std       577.026323
         min         0.000000
         25%       107.500000
         50%       190.000000
         75%       865.000000
         max      3600.000000
         Name: Sodium, dtype: float64
```

Code 2

```
In [24]: df['Sodium'].idxmax()

Out[24]: 82
```

Code 3

```
In [56]: df.at[82,'Item']

Out[56]: 'Chicken McNuggets (40 piece)'
```

Let's investigate this data further. Let's try to understand one of the nutrients in the food items which is sodium. A main source of sodium is table salt. The average American eats five or more teaspoons of salt each day. This is about 20 times as much as the body needs. Sodium is found naturally in foods, but a lot of it is added during processing and preparation. Many foods that do not taste salty, may still be high in sodium. Large amounts of sodium can be hidden in canned, processed and convenience foods. Sodium controls fluid balance in our bodies, and maintains blood volume and blood pressure. Eating too much sodium may raise blood pressure and cause fluid retention, which could lead to swelling of the legs, and feet, or other health issues. When limiting sodium in your diet, a common target is to eat less than 2,000 milligrams of sodium per day.

Now using the nutrition data set for McDonald's, let's do some basic data analysis to answer the question.

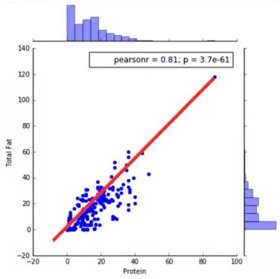
- Which food item has the maximum sodium content? We first use visualization to explore the sodium content of food items. Using the swarm plot method provided by the Seaborne package, we create a categorical scatter plot as shown on the right, then give as the input, category on the x-axis, sodium on the y-axis, and the data will be the data frame DF that contains the nutritional data set from McDonald's. The plot shows the sodium values for the different food items by category. Notice a high value of around 3,600 for sodium on the scatter plot. We will be learning about visualizations later in this module. Let's further explore this high sodium value and identify which food items on the menu have this value for sodium. Let's do some basic data analysis using Python to find which food items on the menu have maximum sodium content. To check the values of sodium levels in the food items within the dataset, we use the code as shown in code 1.
- The describe method is used to understand the summary statistics associated with sodium. Notice that the maximum value of sodium is given as 3,600. Now let's further explore the row associated with the maximum sodium variable as shown in code 2. We use the idxmax method to compute the index values, at which the maximum value of sodium is obtained in the data frame. We see that the output is 82.
- Now lets find the item name associated with the 82nd item in our data frame. As shown in code 3, we will use the .at method to find the item name by passing the index of 82 and the column name item, to be returned for the 82nd row. Finally, we find that the food item on the menu that has a highest sodium content is Chicken McNuggets, 40 pieces.

总结:

Further data exploration using visualizations

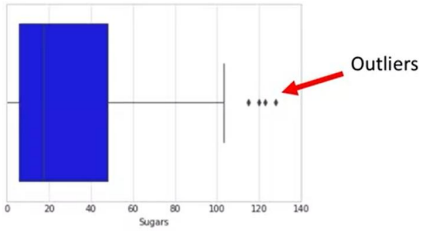
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

plot = sns.jointplot(x="Protein", y='Total Fat', data=df)
plot.show()
```




```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

plot = sns.set_style("whitegrid")
ax = sns.boxplot(x=df["Sugars"])
plot.show()
```



Visualizations are very useful for initial data exploration. They can help us understand relationships, patterns, and outliers in the data.

- Let's first create a scatter plot with protein on the x-axis, and total fat on the y-axis. Scatter plots are very popular visualization tools and show the relationship between two variables with a point for each observation. To do this, we can use the jointplot function provided by the Seaborn package, and give as input, protein on the x-axis and total fat on the y-axis. And the data will be the data frame DF that contains the nutritional data set from McDonald's. The output scatter plot is shown on the right side. The plot has an interesting shape. It shows the correlation between the two variables: protein and fat. Correlation is a measure of association between two variables, and has a value of between -1 and +1. We see that the points on the scatter plot are closer to a straight line in the positive direction. So we have a positive correlation between the two variables. On the top right corner of the scatter plot, we have the values of the Pearson correlation- 0.81 and the significance of the correlation denoted as P - which is a good value that shows the variables are certainly correlated. The plot also shows two histograms: one on the top and the other on the right side. The histogram on the top is that of the variable protein, and the histogram on the right side is that of the variable total fat. We also noticed that there is a point on the scatter plot outside the general pattern. This is a possible outlier.

- Now let's see how we can visualize data using box plots. Box plots are charts that indicate the distribution of one or more variables. The box in a box plot captures the middle 50 percent of data. Lines and points indicate possible skewness and outliers. Let's create a box plot for sugar. The function we are going to use is box plot from the Seaborn package. We give the column name sugars as input to the box plot function. The output is shown on the right side, where we had the box plot with average values of sugar and food items around 30 grams. We also notice a few outliers that indicate food items with extreme values of sugar. There exist food items in the data set that have sugar content of around 128 grams. Candies maybe among these high sugar content food items on the menu.

Now that you know how to do basic exploratory data analysis using Pandas and visualization tools, proceed to the labs in this module where you can practice the concepts learned. Thank you for watching this video.

总结:

Summary & Highlights

Congratulations! You have completed this lesson. At this point in the course, you know:

- You can access a database from a language like Python by using the appropriate API. Examples include ibm_db API for IBM DB2, psycopg2 for ProstgreSQL, and dblib API for SQL Server.

- DB-API is Python's standard API for accessing relational databases. It allows you to write a single program that works with multiple kinds of relational databases instead of writing a separate program for each one.

- The DB_API connect constructor creates a connection to the database and returns a Connection Object, which is then used by the various connection methods.

- The connection methods are:
    - The cursor() method, which returns a new cursor object using the connection.
    - The commit() method, which is used to commit any pending transaction to the database.
    - The rollback() method, which causes the database to roll-back to the start of any pending transaction.
    - The close() method, which is used to close a database connection.

- You can use **SQL Magic commands** to execute queries more easily from Jupyter Notebooks.
    - Magic commands have the general format %sql select * from tablename.
    - Cell magics start with a double %% (percent) sign and apply to the entire cell.
    - Line magics start with a single % (percent) sign and apply to a particular line in a cell.

总结:

1. Which API do you use to connect to a database from Python?

- ○ Census API
- ◉ DB API such as ibm_db API
- ○ Watson API
- ○ REST API

✓ **正确**

Correct. A DB API such as ibm_db will enable you to connect to a database from Python to access and manipulate data.

2. In the ibm_db API, what is the commit() method used for?

- ○ The commit() method is used to reverse any transactions that fail.
- ◉ The commit() method is used to commit any pending transaction to the database.
- ○ The commit() method is used to open a database for connection.
- ○ The commit() method is used to close a database connection.

✓ **正确**

Correct. The commit() method is used to commit any pending transaction to the database.

3. True or false: Resources used by the ibm_db API are released automatically when the program ends. There is no need to specifically close the connection.

- ○ True
- ◉ False

✓ **正确**

Feedback: Correct. It is important to use the close() method to close connections and avoid unused connections taking up resources.

总结:

4. To create a table from Python, you would use...

- ○ An ibm_db.exec_immediate function that includes a SQL statement to create the table.
- ○ You cannot create a table from Python.
- ○ An ibm_db.exec_immediate function that includes connection information.
- ◉ An ibm_db.exec_immediate function that includes connection information and a SQL statement to create the table.

✓ **正确**

Correct. To create a table from Python, use the ibm_db.exec_immediate function and include connection information as the first parameter and a SQL statement to create the table as the second parameter.

5. Which of the following is a correct example of the connect function?

- ○ connect('database name', 'username', 'database type')
- ◉ connect('database name', 'username', 'password')
- ○ connect('database port', 'username', 'password')
- ○ connect('username', 'password')

✓ **正确**

Correct. You must pass the database name, username, and password parameters to connect to the database.

总结: