**Accessing Databases using Python**

In this module you will learn the basic concepts of using Python to connect to databases. In a Jupyter Notebook, you will create tables, load data, query data using SQL magic and SQLite python library. You will also learn how to analyze data using Python.

**Learning Objectives**

* Access databases from Python using SQL magic
* Describe concepts related to accessing Databases using Python
* Create tables and insert data using Python
* Analyze a data set using SQL and Python

# **Accessing Databases using Python**

## **How to Access Databases using Python**

* **Introduction to Databases and Python**:
  + Databases are essential tools for data scientists.
  + The module will cover connecting Python to databases, creating tables, loading data, querying data using SQL, and analyzing data.
* **Benefits of Using Python**:
  + Python is a popular scripting language for database connections.
  + It has a rich ecosystem with tools like **NumPy**, **Pandas**, **matplotlib**, and **SciPy**.
  + Python is easy to learn and has a simple syntax.
* **Python Database API (DB API)**:
  + The DB API simplifies writing Python code to access databases.
  + Documentation for Python is readily available.
* **Jupyter Notebooks**:
  + Jupyter notebooks allow for the creation and sharing of documents containing live code, equations, visualizations, and text.
  + They support over 40 programming languages, including Python, R, and Julia.
  + Notebooks can be shared via email, Dropbox, GitHub, etc.
* **Accessing Databases**:
  + Python communicates with the Database Management System (DBMS) through API calls.
  + The process involves:
    - Connecting to the DBMS.
    - Sending SQL statements.
    - Retrieving query results and status information.
    - Disconnecting from the database.
* **SQL APIs**:
  + Each database system has its own library for SQL APIs.
  + Examples include:
    - **MySQL C API**: Access to MySQL client-server protocol.
    - **psycopg2**: Connects Python to PostgreSQL.
    - **IBM\_DB**: Connects to IBM DB2.
    - **dblib**: Connects to SQL Server.
    - **ODBC**: For Microsoft Windows OS.
    - **OCI**: For Oracle databases.
    - **JDBC**: For Java applications.

## **Writing code using DB-API**

1. **DB-API Overview**

The **DB-API** is a standard interface for connecting to relational databases in Python. It allows you to write code that can work with different databases without changing the core logic.

2. **Advantages of DB-API**

* **Ease of Implementation**: The API is designed to be straightforward.
* **Consistency**: Similar methods across different database modules.
* **Portability**: Code can be reused with different databases.

3. **Connection and Cursor Objects**

* **Connection Object**: Represents the connection to the database.
* **Cursor Object**: Used to execute SQL commands and fetch results.

4. **Key Methods**

Here are the key methods associated with connection and cursor objects:

* **connect()**: Establishes a connection to the database.
* **cursor()**: Creates a cursor object for executing SQL commands.
* **commit()**: Saves changes made during the current transaction.
* **rollback()**: Reverts changes made during the current transaction.
* **close()**: Closes the connection to the database.

5. **Cursor Behavior**

* Cursors from the same connection share changes.
* Cursors allow you to navigate through query results.

6. **Example Code**

Here’s a detailed example of how to use the DB-API to connect to a database, execute a query, and fetch results:

# Import the necessary library

import sqlite3 # Example using SQLite, but you can use other libraries like psycopg2 for PostgreSQL, etc.

# Step 1: Establish a connection to the database

connection = sqlite3.connect('example.db') # Replace 'example.db' with your database name

# Step 2: Create a cursor object

cursor = connection.cursor()

# Step 3: Create a table (if it doesn't exist)

cursor.execute('''

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY,

name TEXT NOT NULL,

age INTEGER NOT NULL

)

''')

# Step 4: Insert data into the table

cursor.execute('''

INSERT INTO users (name, age) VALUES (?, ?)

''', ('Alice', 30))

# Step 5: Commit the transaction

connection.commit()

# Step 6: Query the database

cursor.execute('SELECT \* FROM users')

# Step 7: Fetch results

results = cursor.fetchall()

for row in results:

print(row) # Print each row

# Step 8: Close the cursor and connection

cursor.close()

connection.close()

7. **Important Notes**

* Always ensure to **close** the cursor and connection to free up resources.
* Use **parameterized queries** (like ? in the example) to prevent SQL injection attacks.

## **Accessing Databases with SQL Magic**

accessing databases using SQL Magic:

Introduction to Magic Statements

* **Definition**: Magic commands or functions in Jupyter Notebooks that provide special functionalities not available in standard Python code.
* **Purpose**: Designed to solve common problems in data analysis and enhance the user experience in Jupyter Notebooks.

Types of Magic Statements

1. **Line Magics**:
   * **Prefix**: Single percentage character %.
   * **Functionality**: Operate on a single line of input, similar to command line calls.
   * **Examples**:
     + %pwd: Prints the current working directory.
     + %ls: Lists all files in the current directory.
     + %history: Displays the command history.
     + %reset: Resets the namespace by removing all user-defined names.
     + %who: Lists all variables in the namespace.
     + %whos: Provides detailed information about all variables in the namespace.
     + %matplotlib inline: Ensures that matplotlib plots appear within the notebook.
     + %timeit: Times the execution of a single statement.
2. **Cell Magics**:
   * **Prefix**: Double percentage characters %%.
   * **Functionality**: Operate on multiple lines of input and can execute code in different programming languages.
   * **Examples**:
     + %%writefile myfile.txt: Writes the contents of the cell to a specified file.
     + %%html: Allows writing HTML code, which will be rendered accordingly.
     + %%javascript or %%js: Executes JavaScript code.
     + %%bash: Executes bash commands.

Using SQL Magic in Jupyter Notebooks

* **Installation**:
  + To use SQL Magic, first install the iPython\_SQL library by running:

!pip install iPython\_SQL

* **Loading SQL Extension**:
  + Load the SQL extension in your notebook with:

%load\_ext sql

* **Establishing a Database Connection**:
  + To use SQL Magic, you need to establish a connection to a database. For example, to connect to an SQLite database named HR.db, use:

%sql sqlite:///HR.db

Executing SQL Queries

* **Running Queries**:
  + Use %sql for line magic to run a single SQL query.
  + Use %%sql for cell magic to execute multiple SQL statements in a cell.

Examples of SQL Magic Usage

* **Displaying Table Contents**:
  + After establishing a connection, you can execute SQL queries to display data from tables. For example:
  + %%sql

SELECT \* FROM employee;

Summary of Key Concepts Learned

* **Magic Statements**: Understand the difference between line magics and cell magics.
* **Popular Line Magics**: Familiarize yourself with commonly used line magic commands.
* **SQL Magic**: Learn how to install, load, and use SQL Magic in Jupyter Notebooks to access databases.

## **Analyzing Data with Python**

1. Designing a Hands-on Lab

* **Objective**: Create effective hands-on labs for data analysis.
* **Components**:
  + Clear instructions for learners.
  + Tools and resources needed for the lab.

2. McDonald's Nutritional Data

* **Dataset**: Nutritional facts for popular menu items from McDonald's.
* **Source**: Obtained from Kaggle.
* **Purpose**: To perform exploratory data analysis using Python.

3. Setting Up the Database

* **SQLite 3**:
  + A self-contained, serverless SQL database engine.
* **Loading Data**:
* import pandas as pd
* # Load the CSV file into a DataFrame

data = pd.read\_csv('mcdonalds\_nutrition.csv')

4. Connecting to the Database

* **Connection**:
* import sqlite3
* # Connect to the SQLite database

conn = sqlite3.connect('mcdonalds.db')

* **Loading Data into Database**:
* # Load the DataFrame into the SQLite database

data.to\_sql('MCDONALDS\_NUTRITION', conn, if\_exists='replace', index=False)

5. Retrieving Data with Pandas

* **Loading Data**:
* # Load data from the MCDONALDS\_NUTRITION table into a DataFrame

df = pd.read\_sql('SELECT \* FROM MCDONALDS\_NUTRITION', conn)

* **Exploring Data**:
* # View the first few rows of the DataFrame

print(df.head())

6. Exploratory Data Analysis

* **Summary Statistics**:
* # Get summary statistics of the DataFrame
* summary\_stats = df.describe()

print(summary\_stats)

* **Sodium Content**:
  + Discuss the health implications of sodium.
  + Target: Limit sodium intake to less than 2000 mg per day.

7. Data Visualization

* **Swarm Plot**:
* import seaborn as sns
* import matplotlib.pyplot as plt
* # Create a swarm plot for sodium content
* sns.swarmplot(x='Category', y='Sodium', data=df)
* plt.title('Sodium Content by Food Category')

plt.show()

* **Identifying Maximum Sodium**:
* # Find the maximum sodium value
* max\_sodium = df['Sodium'].max()
* print(f'Maximum Sodium Value: {max\_sodium}')
* # Find the index of the maximum sodium value
* max\_index = df['Sodium'].idxmax()

print(f'Index of Maximum Sodium: {max\_index}')

8. Correlation Analysis

* **Scatter Plot**:
* # Create a scatter plot to analyze the relationship between protein and total fat
* sns.jointplot(x='Protein', y='Total\_Fat', data=df)
* plt.title('Protein vs Total Fat')

plt.show()

* **Correlation Value**:
* # Calculate the Pearson correlation
* correlation = df['Protein'].corr(df['Total\_Fat'])

print(f'Pearson Correlation: {correlation}')

9. Box Plots

* **Box Plot for Sugar**:
* # Create a box plot for sugar content
* sns.boxplot(y='Sugars', data=df)
* plt.title('Distribution of Sugar Content')

plt.show()

This structured breakdown includes key concepts and relevant code snippets to help you understand the material in depth.

**Summary: Accessing databases using Python**

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

* Magic commands are special commands that provide special functionalities.
* Cell magics are commands prefixed with two %% characters and operate on multiple input lines.
* DB APIs are commands prefixed with two %% characters and operate on multiple input lines.
* The two main concepts in the Python DB API are Connection Objects and Query Objects.
* A database cursor is a control structure that enables traversal over the records in a database.
* Pandas’ methods are equipped with common mathematical and statistical methods.
* The pandas.read\_csv() function is used to read the database CSV file.
* The sqlite3.connect() function is used to connect to a database.
* To use pandas to retrieve data from the database tables, load data using the read\_sql method and select the SQL Select Query
* A categorical scatterplot is created using the swarmplot() method by the seaborn package.