The Python program in the Jupyter Notebook simulates an ATM machine, allowing users to perform various banking transactions. Here's a breakdown of the code:

1. Importing Required Libraries

```python
import time
import mysql.connector
```

- `time`: Used for simulating delays to mimic the behavior of an ATM.

- `mysql.connector`: Used to interact with a MySQL database.

2. ATM_Machine Class

The `ATM_Machine` class models the basic functionalities of an ATM. It includes several methods to handle different ATM operations.

- `__init__` Method

```python
def __init__(self, cust_id, pin, balance=0):
    self.cust_id = cust_id
    self.pin = pin
    self.balance = balance
    self.transaction_history = []
```

Initializes the class with a customer ID, PIN, and an optional balance (default is 0). It also initializes an empty transaction history list.

- `authenticate` Method

```python
def authenticate(self, pin):
```

```
        return self.pin == pin
```

Checks if the entered PIN matches the stored PIN.

- `check_balance` Method

```python
def check_balance(self):
    print(f"Your current balance is: ₹{self.balance}")
    self.transaction_history.append("Checked balance")
```

Displays the current balance and records the action in the transaction history.

- `deposit` Method

```python
def deposit(self, amount):
    if amount > 0:
        self.balance += amount
        print(f"You have successfully deposited ₹{amount}. Your new balance is: ₹{self.balance}")
        self.transaction_history.append(f"Deposited ₹{amount}")
    else:
        print("Deposit amount must be greater than zero.")
```

Allows the user to deposit money into their account, updating the balance and recording the transaction.

- `withdraw` Method

```python
def withdraw(self, amount):
```

```
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"You have successfully withdrawn ₹{amount}. Your new balance is: ₹{self.balance}")
                self.transaction_history.append(f"Withdrew ₹{amount}")
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be greater than zero.")
```

Allows the user to withdraw money, ensuring there are sufficient funds and updating the balance accordingly.

- `change_pin` Method

```
    def change_pin(self, old_pin, new_pin):
        if self.authenticate(old_pin):
            self.pin = new_pin
            print("PIN successfully changed.")
            self.transaction_history.append("Changed PIN")
        else:
            print("Incorrect old PIN.")
```

Enables the user to change their PIN after verifying the old one.

- `show_transaction_history` Method

```
    def show_transaction_history(self):
```

```
        print("Transaction History:")
        for transaction in self.transaction_history:
            print(transaction)
```

Displays all transactions that the user has made during the session.

- `save_to_db` Method

```python
def save_to_db(self):
    mydb = mysql.connector.connect(
        host="localhost",
        user="username",
        password="password",
        database="atm"
    )
    mycursor = mydb.cursor()
    sql = "INSERT INTO atm (Customer_id, Balance) VALUES (%s, %s)"
    val = (self.cust_id, self.balance)
    mycursor.execute(sql, val)
    mydb.commit()
    mydb.close()
```

Saves the customer's ID and balance to a MySQL database.

**** Change Username and Password for your account ****

3. ATM Simulation Function

```python
def atm_simulation():
```

```
    print("\nPlease insert your card...")
    time.sleep(2)
    cust_id = int(input("Please Enter your Cust ID: "))
    input_pin = input("Enter your PIN: ")

    atm = ATM_Machine(cust_id, input_pin)
```

This function initiates the ATM process by prompting the user for their customer ID and PIN, then creates an `ATM_Machine` object.

- Main Loop

```python
  while True:
    print("\nWelcome")
    print("1. Check Balance")
    print("2. Deposit")
    print("3. Withdraw")
    print("4. Change PIN")
    print("5. Transaction History")
    print("6. Exit")
    choice = input("Please choose an option: ")

    if choice == '1':
      atm.check_balance()
    elif choice == '2':
      amount = float(input("Enter amount to Deposit: "))
      atm.deposit(amount)
    elif choice == '3':
```

```
            amount = float(input("Enter amount to Withdraw: "))
            atm.withdraw(amount)
        elif choice == '4':
            old_pin = input("Enter your old PIN: ")
            new_pin = input("Enter your new PIN: ")
            if old_pin != new_pin:
                atm.change_pin(old_pin, new_pin)
            else:
                print("Do Not Enter Pin which is used before")
        elif choice == '5':
            atm.show_transaction_history()
        elif choice == '6':
            atm.save_to_db()
            print("Thank you! Have a Good Day!")
            break
        else:
            print("Invalid Input")
```

This loop provides the user with options to check their balance, deposit money, withdraw money, change their PIN, view their transaction history, or exit the simulation.

4. Running the Simulation

```python
atm_simulation()
```

Finally, the simulation is executed.

This code provides a simple but functional ATM simulation, which can be very educational for freshers to understand basic programming concepts like classes, methods, conditionals, loops, and interacting with databases.

To understand how the database works with the ATM simulation, let's break down the "ATM Database.sql" file, which likely contains the SQL commands needed to create and manage the database used by the ATM simulation.

I'll first open the SQL file and then explain its contents.

The SQL file contains the following commands:

1. Use Database

```sql
USE atm;
```

This command selects the `atm` database, which means all subsequent commands will be executed within this database. If the database doesn't exist, you would need to create it first with:

```sql
CREATE DATABASE atm;
```

2. Create Table

```sql
CREATE TABLE atm (
    Customer_id INT,
    Balance INT
);
```

This command creates a table named `atm` with the following columns:

- `Customer_id`: An integer that uniquely identifies each customer.

- `Balance`: An integer representing the customer's account balance.

3. Select All from Table

```sql
SELECT * FROM atm;
```

This command retrieves all records from the `atm` table, showing both the `Customer_id` and `Balance` columns.

How the Database Integrates with the Python Program

In the Python program, there is a method called `save_to_db` that interacts with this database. Here's how it fits together:

- Database Connection: The Python code uses `mysql.connector` to connect to the MySQL database specified by `host`, `user`, `password`, and `database` (which is `atm` in this case).

- Inserting Data: When the user exits the simulation, the `save_to_db` method is called to save the `Customer_id` and `Balance` to the `atm` table:

```python
sql = "INSERT INTO atm (Customer_id, Balance) VALUES (%s, %s)"
val = (self.cust_id, self.balance)
mycursor.execute(sql, val)
```

This SQL command inserts the customer's ID and current balance into the `atm` table.

- Persistence: Once the data is inserted, it's stored in the database, making it persistent even after the program ends. This is useful for maintaining records of customer balances across multiple sessions.

Summary

- Database: A structured way to store data, in this case, customer IDs and balances, that can be easily accessed and manipulated.

- Table: A table in a database holds data in rows and columns, where each row is a record and each column is a data field (like `Customer_id` and `Balance`).

- SQL Commands: These are instructions to interact with the database, such as creating tables and inserting or retrieving data.

This setup is a simple example of how applications can use databases to store and retrieve important information, which is a common practice in real-world software development.