INTEGRAÇÃO

SQL SERVER e PYTHON

Apostila Completa do Módulo de Integração SQL Server e Python

SUMÁRIO

CASE 01: ESCRITA

SUMÁRIO

CASE 02: LEITURA

CASE 1

ESCRITA

EXPLICANDO O PROJETO DE INTEGRAÇÃO

Neste módulo, aprenderemos como integrar o SQL Server ao Python.

Neste primeiro Case, faremos um projeto de **escrita** no banco de dados SQL Server utilizando o editor de código Jupyter Notebook.

Para isso, dividiremos esse pequeno projeto em quatro etapas:

1ª Etapa: Prepararemos o SQL Server para a integração;

2ª Etapa: Efetuaremos as configurações iniciais no Jupyter;

3ª Etapa: Adicionaremos os comandos SQL no Jupyter;

4ª Etapa: Tornaremos o cadastro de dados automático por meio de variáveis.

Uma observação importante: o objetivo deste módulo é oferecer uma visão geral básica de como se estabelecer uma conexão entre as linguagens SQL e Python. Sendo assim, não nos aprofundaremos em comandos avançados nem em conceitos de Python. Para isso, sugerimos, aos que se interessarem, que façam algum curso de Python, como o nosso Python Impressionador! ☺

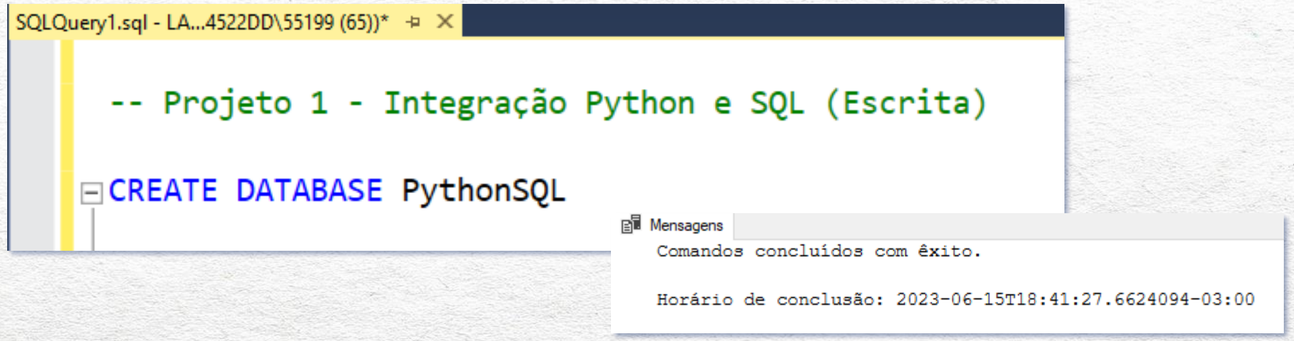
Vamos nessa? Bons estudos!

PREPARANDO O SQL SERVER ANTES DA INTEGRAÇÃO

Inicialmente, precisamos preparar o SQL Server para receber os dados que serão inseridos no banco de dados por meio do Jupyter Notebook.

Para isso, vamos então criar um novo banco de dados e, dentro dele, uma tabela que receberá tais dados.

Utilizando o SSMS, criaremos um novo banco de dados chamado PythonSQL:



PREPARANDO O SQL SERVER ANTES DA INTEGRAÇÃO

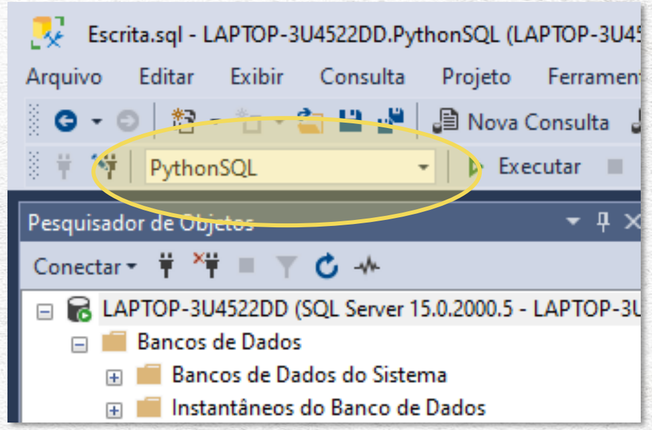
Agora, precisamos passar a utilizar o banco de dados PythonSQL, para poder criar nossa tabela dentro dele.

Para isso, executamos o seguinte comando:

```
USE PythonSQL
```

```
Mensagens
Comandos concluídos com êxito.

Horário de conclusão: 2023-06-15T19:00:45.5390870-03:00
```

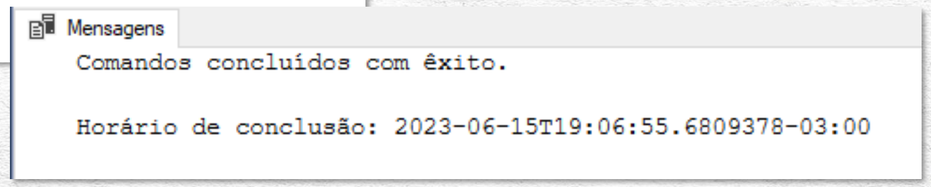
Repare que, agora, o nome do banco de dados PythonSQL aparece no campo de banco de dados em uso do SSMS:



PREPARANDO O SQL SERVER ANTES DA INTEGRAÇÃO

Podemos criar nossa tabela que receberá os dados que serão inseridos via Python. Vamos chamá-la de Vendas:

```
CREATE TABLE Vendas (
    id_venda INT,
    data_venda DATE,
    cliente VARCHAR(100),
    produto VARCHAR(100),
    preco DECIMAL(10, 2),
    quantidade INT
)
```



PREPARANDO O SQL SERVER ANTES DA INTEGRAÇÃO

Inserimos o primeiro registro via SSMS, apenas para testá-la:

```
INSERT INTO Vendas (id_venda, data_venda, cliente, produto, preco, quantidade) VALUES
    (1, '22/04/2022', 'Ana', 'Celular', 2000, 1)
```

Mensagens

```
(1 linha afetada)

Horário de conclusão: 2023-06-15T19:08:44.8849909-03:00
```

Ao consultarmos a tabela, veremos que o registro da primeira venda foi incluído com sucesso:

```
SELECT * FROM Vendas
```

Resultados | Mensagens

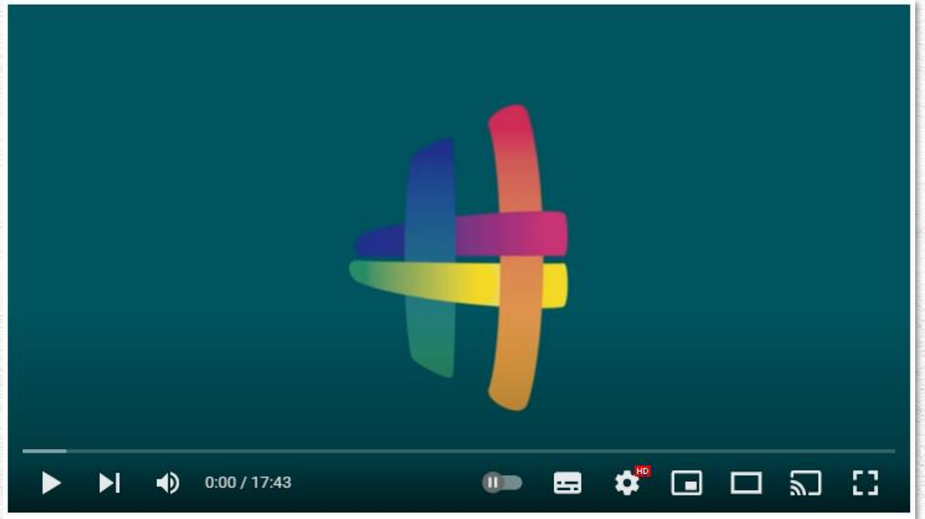
	id_venda	data_venda	cliente	produto	preco	quantidade
1	1	2022-04-22	Ana	Celular	2000.00	1

Sendo assim, podemos partir para a 2ª etapa do nosso projeto.

CONFIGURAÇÕES INICIAIS NO JUPYTER

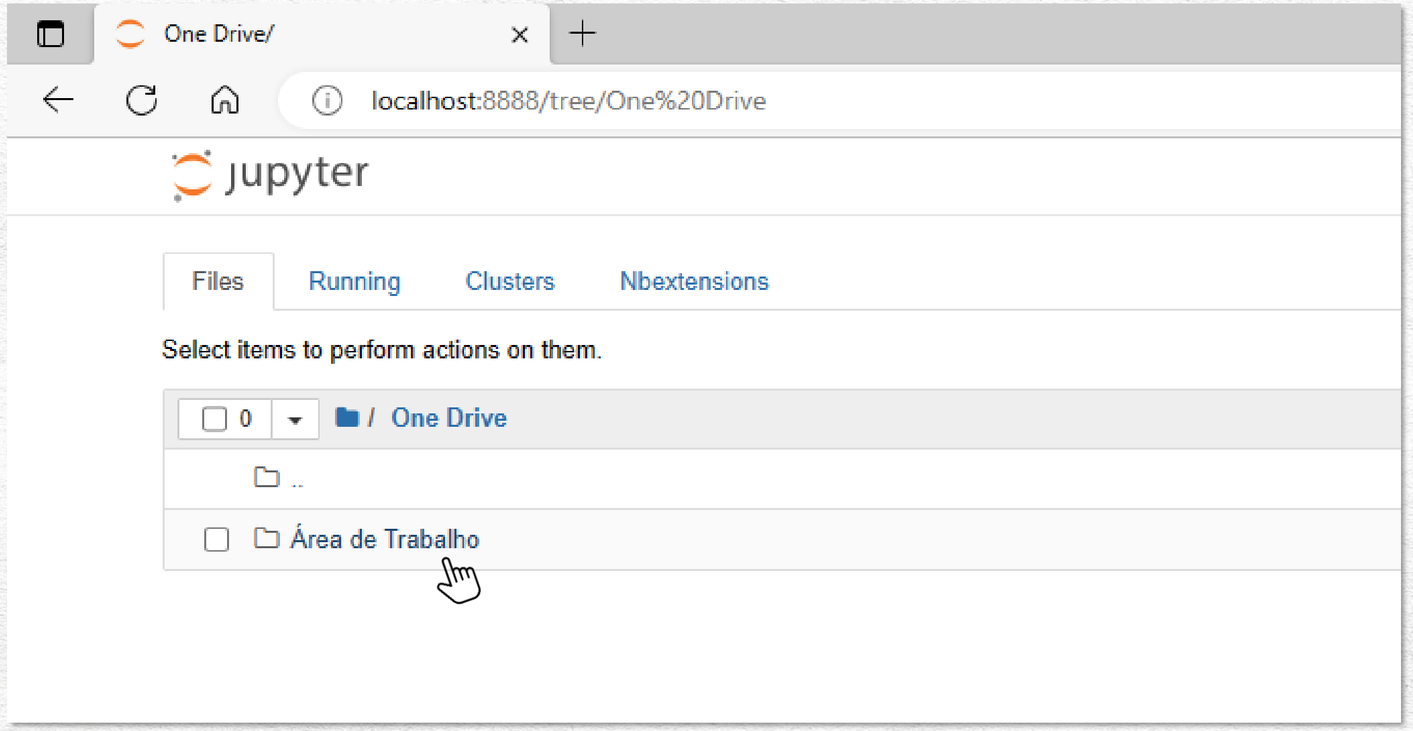
Agora, vamos iniciar as configurações no Jupyter.

Se você não tiver o editor de códigos Jupyter Notebook instalado em seu computador e precise de ajuda para efetuar essa instalação, sugerimos que assista ao vídeo da Hashtag disponibilizado no nosso canal do Youtube, em que o Lira ensina a fazer isso passo a passo. É só clicar na imagem abaixo:



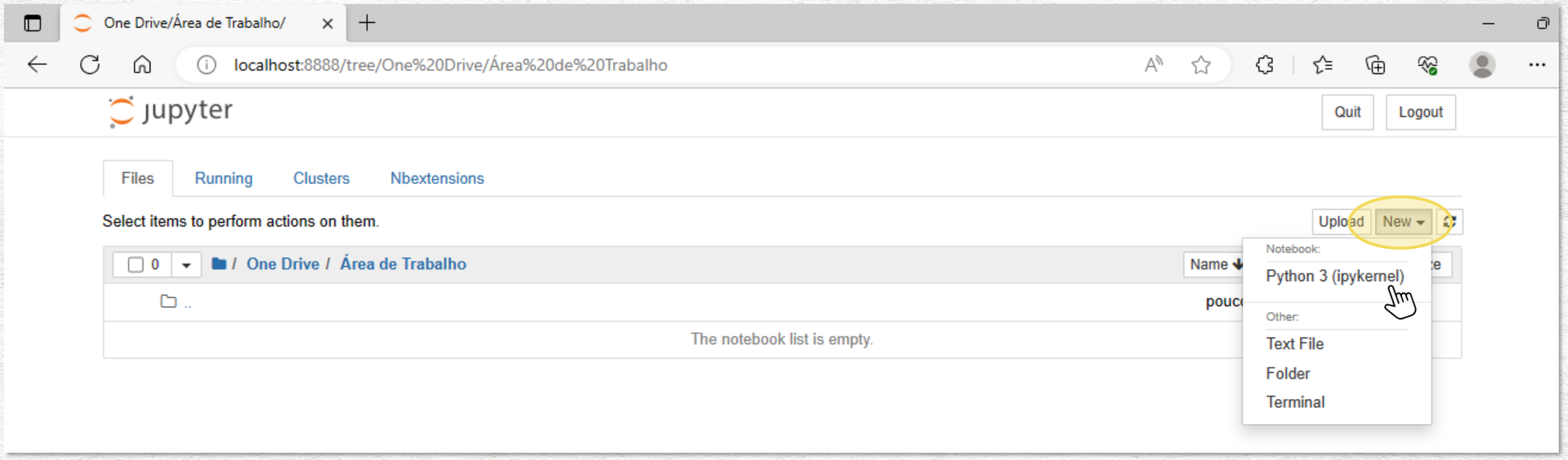
CONFIGURAÇÕES INICIAIS NO JUPYTER

Efetuada a instalação, abra o Jupyter no seu computador e selecione a pasta de sua preferência para salvar o script que vamos construir e utilizar neste nosso case de escrita:



CONFIGURAÇÕES INICIAIS NO JUPYTER

Em seguida, clique em New e selecione Python 3 (ipykernel):

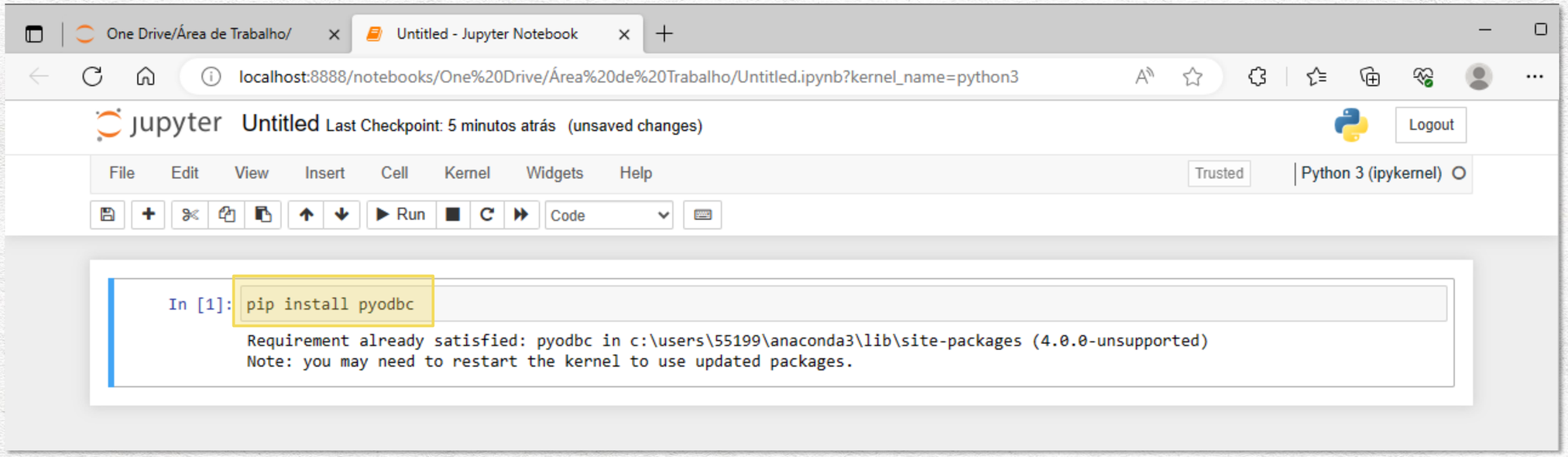


CONFIGURAÇÕES INICIAIS NO JUPYTER

Com a nova janela aberta, o primeiro passo é instalarmos uma biblioteca do Python chamada pyodbc.

Essa biblioteca permite integrarmos o Python com o SQL Server.

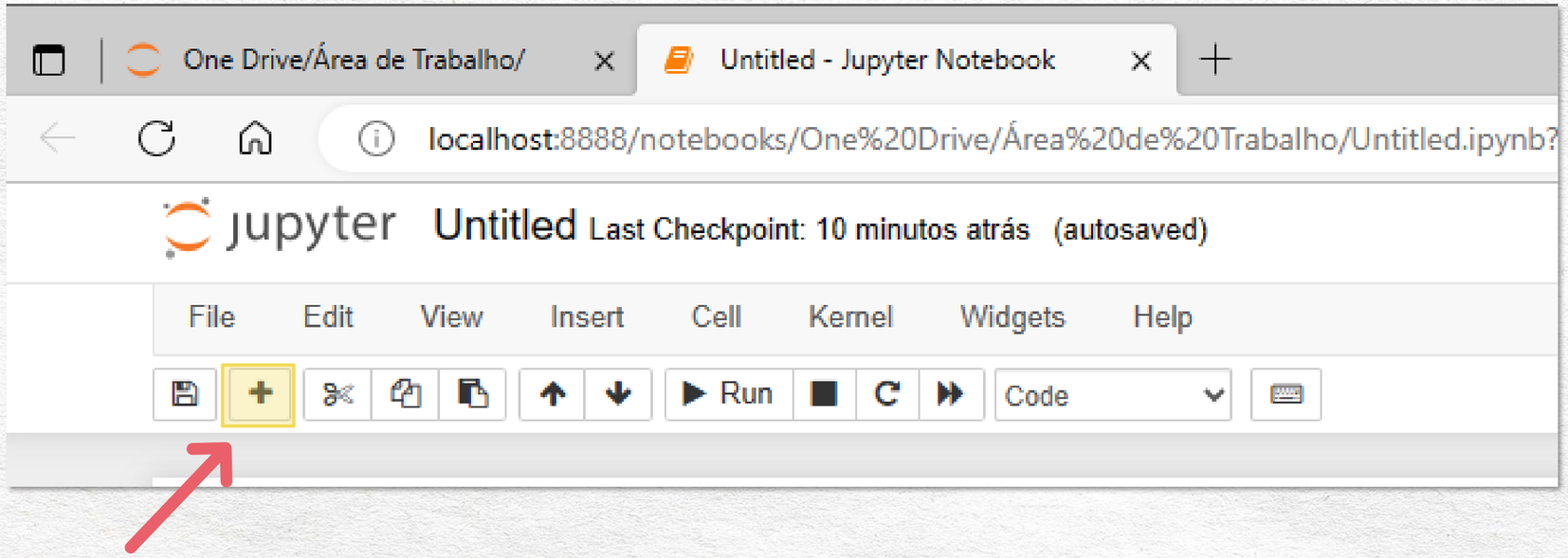
Para instalá-la, devemos digitar o comando **pip install pyodbc** e apertar as teclas de atalho Crtl + Enter para executar o código:



CONFIGURAÇÕES INICIAIS NO JUPYTER

Instalada a biblioteca pyodbc, podemos agora criar o nosso código de conexão com o SQL Server.

Para isso, vamos adicionar uma nova célula, clicando no ícone mostrado abaixo:



CONFIGURAÇÕES INICIAIS NO JUPYTER

Nessa nova célula, vamos:

1) Importar a biblioteca pyodbc (que acabamos de instalar);

2) Criar uma variável que receberá os três parâmetros necessários para conexão (Driver, Server e Database);

3) Efetuar a conexão:

In [ ]:
```
# importanto biblioteca pyodbc:

import pyodbc


# criando dados de conexão:

dados_conexao = (
    "Driver={SQL Server};"
    "Server=LAPTOP-3U4522DD;"
    "Database=PythonSQL;"
)

# conectando:

conexao = pyodbc.connect(dados_conexao)
print("Conexão Bem Sucedida")
```

1

2

3

CONFIGURAÇÕES INICIAIS NO JUPYTER

Repare que:

1) Para importar a biblioteca pyodbc, utilizamos o comando **import pyodbc:**

```
import pyodbc
```

CONFIGURAÇÕES INICIAIS NO JUPYTER

2) Para informar os três parâmetros necessários para conexão (Driver, Server e Database), criamos uma variável, que chamamos de **dados_conexao**, e informamos os dados de cada um dos parâmetros, sendo:

a) Driver: {SQL Server};

b) Server: o nome do computador (ATENÇÃO: cada máquina tem o seu próprio nome. O nome mostrado aqui no exemplo é do computador que estamos utilizando no momento. **Você precisará descobrir e informar o nome da sua máquina**. Mostraremos como fazer isso na próxima página;

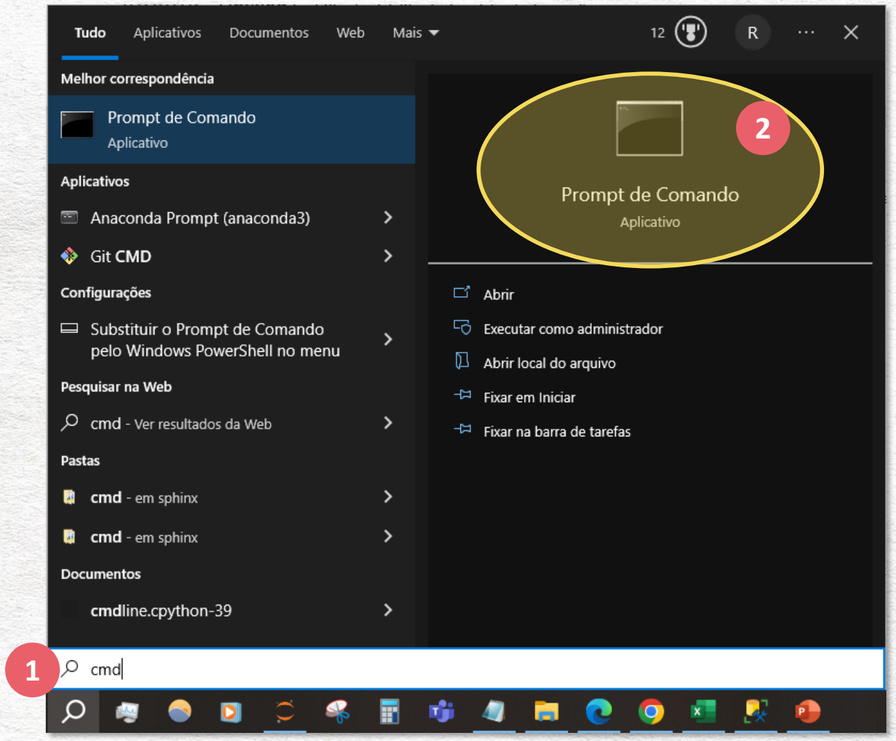
c) Database: PythonSQL (que é o nome do banco de dados que criamos no início do módulo, no qual também criamos a tabela Vendas que receberá os dados a serem inseridos pelo Jupyter).

```
dados_conexao = (
    "Driver={SQL Server};"
    "Server=LAPTOP-3U4522DD;"
    "Database=PythonSQL;"
)
```

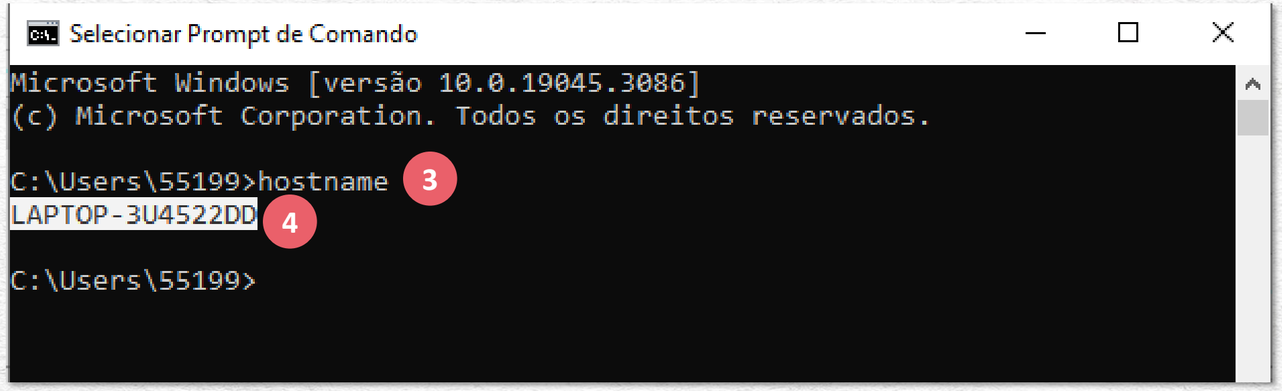
CONFIGURAÇÕES INICIAIS NO JUPYTER

Para descobrir o nome do seu computador para informar ao parâmetro Server da página anterior, você pode fazer o seguinte:

1. Digite **cmd** na barra de pesquisas do seu computador para localizar o prompt de comando;
2. Clique em seu ícone para abri-lo;



3. No prompt que se abrir, digite **hostname** e aperte a tecla Enter;
4. Na linha de baixo, aparecerá o nome do seu computador. Selecione e copie esse nome (clique duplo sobre ele para selecionar, Ctrl + C para copiar).



CONFIGURAÇÕES INICIAIS NO JUPYTER

Voltando ao Jupyter, cole (Ctrl + V) o nome do seu computador no parâmetro Server:

```
dados_conexao = (
    "Driver={SQL Server};"
    "Server=LAPTOP-3U4522DD;"
    "Database=PythonSQL;"
)
```

3) Por fim, para efetuar a conexão, criamos uma variável, que chamamos de **conexao**, e passamos para ela a variável dados_conexao (que contém os parâmetros de conexão). Além disso, acrescentamos um **print** para mostrar na tela a mensagem “Conexão Bem Sucedida” quando conectarmos, assim, saberemos que deu tudo certo:

```
conexao = pyodbc.connect(dados_conexao)
print("Conexão Bem Sucedida")
```

CONFIGURAÇÕES INICIAIS NO JUPYTER

Tudo pronto, agora podemos conectar. Para isso, executamos a célula (Ctrl + Enter):

Note que, enquanto o Jupyter executa o código, aparece um asterisco, indicando que o código está sendo executado:

```
In [*]: # importanto biblioteca pyodbc:

import pyodbc


# criando dados de conexão:

dados_conexao = (
    "Driver={SQL Server};"
    "Server=LAPTOP-3U4522DD;"
    "Database=PythonSQL;"
)

# conectando:

conexao = pyodbc.connect(dados_conexao)
print("Conexão Bem Sucedida")
```

Finalizada a execução do código, o asterisco passa a ser um número, e abaixo da célula aparece a mensagem “Conexão Bem Sucedida”, que configuramos no código:

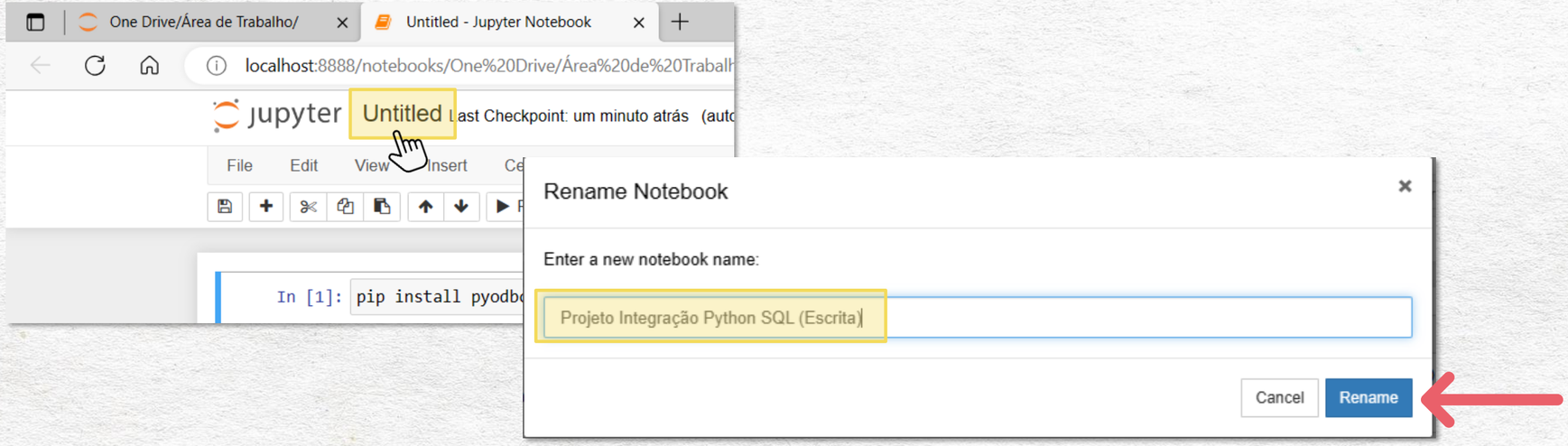
```
In [2]: # importanto biblioteca pyodbc:

import pyodbc


# criando dados de conexão:

dados_conexao = (
    "Driver={SQL Server};"
    "Server=LAPTOP-3U4522DD;"
    "Database=PythonSQL;"
)

# conectando:

conexao = pyodbc.connect(dados_conexao)
print("Conexão Bem Sucedida")

Conexão Bem Sucedida
```

CONFIGURAÇÕES INICIAIS NO JUPYTER

Efetuada a conexão, poderemos partir para a inserção de dados no banco de dados PythonSQL através do Jupyter.

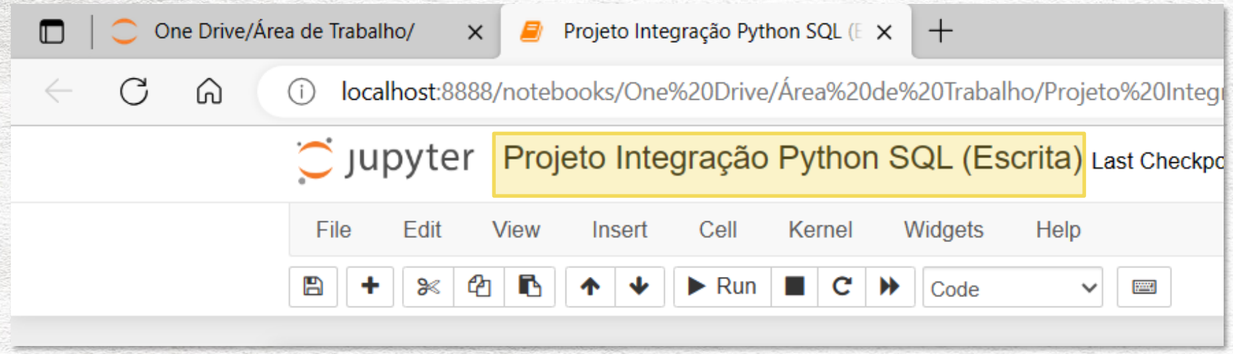
Antes, porém, vamos renomear esse arquivo no qual estamos escrevendo o nosso código / script, para que, sempre que quisermos, possamos facilmente localizá-lo na pasta em que o salvamos.

Para renomeá-lo, basta clicar sobre a palavra “Untitled” na parte superior da página e, na janelinha que se abrir, digitar o novo nome do arquivo e clicar em “Rename”:

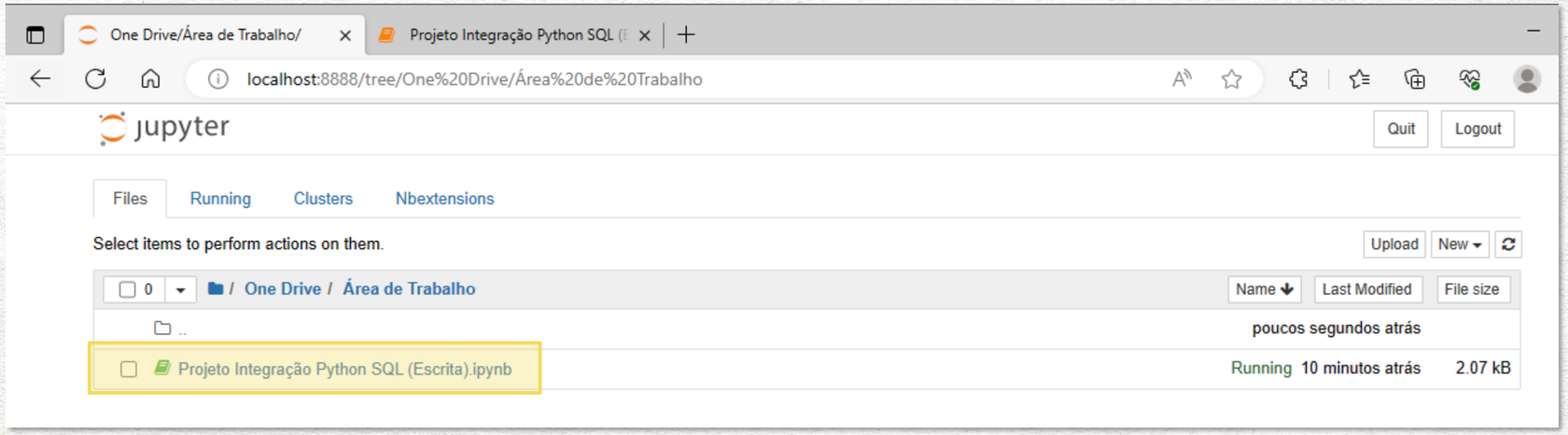


CONFIGURAÇÕES INICIAIS NO JUPYTER

Veja que o arquivo agora aparece com o nome informado:



E se voltarmos à pasta onde o abrimos inicialmente, lá estará ele, com o novo nome:



ADICIONANDO OS COMANDOS EM SQL DENTRO DO JUPYTER

Vejamos agora como executar a 3ª etapa do nosso projeto, para inserir novos dados na tabela Vendas do banco de dados PythonSQL, armazenado no SQL Server, via Python, utilizando o Jupyter.

Vamos adicionar uma nova célula ao nosso arquivo do Jupyter para:

1. Abrir um cursor que receberá as informações da conexão que efetuamos;
2. Criar uma variável, chamada **comando**, que receberá o código SQL de inserção de um novo registro (INSERT INTO);
3. Passaremos a variável comando ao cursor para que ele execute o código SQL de inserção de dados;
4. Daremos um commit com o cursor para que o novo registro seja efetivamente salvo na tabela Vendas.

```
In [ ]: # incluindo registro id = 2

cursor = conexao.cursor()  1

comando = """INSERT INTO Vendas (id_venda, data_venda, cliente, produto, preco, quantidade) VALUES  2
    (2, '10/05/2022', 'Bruno', 'Notebook', 3500, 2)"""

cursor.execute(comando)  3

cursor.commit()  4
```

ADICIONANDO OS COMANDOS EM SQL DENTRO DO JUPYTER

Executamos o código desta célula com um Ctrl + Enter (note que apareceu um número dentro dos colchetes do In, sinalizando que a execução já terminou).

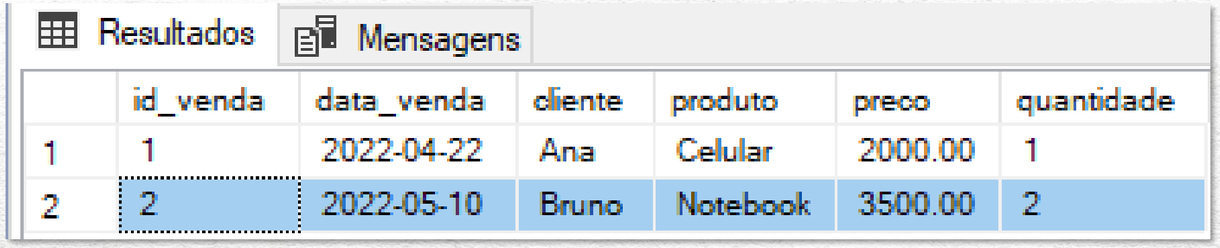
Repare que, desta vez, incluímos o segundo registro na tabela Vendas, de id_venda = 2, data_venda = '10/05/2022', cliente = 'Bruno', produto = 'Notebook', preco = 3500 e quantidade = 2:

```
In [3]: # incluindo registro id = 2

        cursor = conexao.cursor()

        comando = """INSERT INTO Vendas (id venda, data venda, cliente, produto, preco, quantidade) VALUES
        (2, '10/05/2022', 'Bruno', 'Notebook', 3500, 2)"""

        cursor.execute(comando)

        cursor.commit()
```

ADICIONANDO OS COMANDOS EM SQL DENTRO DO JUPYTER

No SSMS, se consultarmos novamente a tabela Vendas, veremos que o registro que inserimos via Python foi realmente armazenado na tabela:

```
SELECT * FROM Vendas
```



Resultados | Mensagens

	id_venda	data_venda	cliente	produto	preco	quantidade
1	1	2022-04-22	Ana	Celular	2000.00	1
2	2	2022-05-10	Bruno	Notebook	3500.00	2

ADICIONANDO OS COMANDOS EM SQL DENTRO DO JUPYTER

Se inserirmos mais um registro via Python...

In [4]:
```
# incluindo registro id = 3

cursor = conexao.cursor()

comando = """INSERT INTO Vendas (id_venda, data_venda, cliente, produto, preco, quantidade) VALUES
    (3, '13/05/2022', 'Carla', 'Tablet', 1200, 1)"""

cursor.execute(comando)

cursor.commit()
```

ADICIONANDO OS COMANDOS EM SQL DENTRO DO JUPYTER

... E novamente consultarmos a tabela Vendas pelo SSMS, veremos que o terceiro registro também foi armazenado:

```
SELECT * FROM Vendas
```

Resultados | Mensagens

	id_venda	data_venda	cliente	produto	preco	quantidade
1	1	2022-04-22	Ana	Celular	2000.00	1
2	2	2022-05-10	Bruno	Notebook	3500.00	2
3	3	2022-05-13	Carla	Tablet	1200.00	1

AUTOMATIZANDO O CADASTRO DE DADOS POR MEIO DE VARIÁVEIS

Que tal deixar nosso código mais “inteligente”?

Vimos até agora como inserir dados informados diretamente ao comando INSERT INTO que criamos no Jupyter.

E se quisermos automatizar isso para que quaisquer dados que sejam recebidos externamente sejam interpretados pelo código do Jupyter e armazenados em seus respectivos campos da tabela Vendas do banco de dados PythonSQL do SQL Server, como fazemos?

Para isso, podemos utilizar **variáveis**.

Como fazer isso, é o que veremos nas próximas páginas. Vamos lá! ☺

AUTOMATIZANDO O CADASTRO DE DADOS POR MEIO DE VARIÁVEIS

Para automatizar, como dissemos, precisaremos criar variáveis que corresponderão à cada campo (item) do registro (linha) que será inserido na tabela.

Na tabela Vendas, temos 6 colunas.

Portanto, a cada INSERT INTO que executamos, inserimos 6 informações distintas, uma em cada campo do registro: **id_venda**, **data_venda**, **cliente**, **produto**, **preco**, **quantidade**.

Sendo assim, precisaremos criar 6 variáveis, que receberão as informações a serem armazenadas em cada campo do registro que iremos inserir na tabela Vendas.

Chamaremos essas **variáveis** de **id**, **data**, **cliente**, **produto**, **preco** e **quantidade**.

AUTOMATIZANDO O CADASTRO DE DADOS POR MEIO DE VARIÁVEIS

Veja como ficará o código:

In [ ]:
```
# incluindo registro id = 4 (automatizando com variáveis)

cursor = conexao.cursor()

id = 4
data = "17/06/2022"
cliente = "Diego"
produto = "Tablet"
preco = 1200
quantidade = 1


comando = f"""INSERT INTO Vendas (id_venda, data_venda, cliente, produto, preco, quantidade) VALUES
    ({id}, '{data}', '{cliente}', '{produto}', {preco}, {quantidade})"""

cursor.execute(comando)

cursor.commit()
```

AUTOMATIZANDO O CADASTRO DE DADOS POR MEIO DE VARIÁVEIS

Precisaremos alterar o seguinte:

Após abrirmos o cursor que recebe os parâmetros da conexão, em vez de já partirmos para a declaração da variável comando (que receberá o código SQL de inserção de dados), primeiro declararemos as variáveis que receberão os dados a serem armazenados:

In [ ]:
```
# incluindo registro id = 4 (automatizando com variáveis)

cursor = conexao.cursor()

id = 4
data = "17/06/2022"
cliente = "Diego"
produto = "Tablet"
preco = 1200
quantidade = 1


comando = f"""INSERT INTO Vendas (id_venda, data_venda, cliente, produto, preco, quantidade) VALUES
    ({id}, '{data}', '{cliente}', '{produto}', {preco}, {quantidade})"""

cursor.execute(comando)

cursor.commit()
```

AUTOMATIZANDO O CADASTRO DE DADOS POR MEIO DE VARIÁVEIS

Depois disso, criaremos a variável comando, a qual receberá o código SQL de inserção de dados.

No entanto, repare que, nesse código SQL, também faremos alterações. Como os dados a serem inseridos na tabela serão armazenados nas variáveis (externas ao código), não informaremos mais tais valores diretamente para o comando INSERT INTO. O que passaremos para o INSERT INTO serão as variáveis externas que armazenarão os referidos valores (**id**, **data**, **cliente**, **produto**, **preco** e **quantidade**):

In [ ]:
```
# incluindo registro id = 4 (automatizando com variáveis)

cursor = conexao.cursor()

id = 4
data = "17/06/2022"
cliente = "Diego"
produto = "Tablet"
preco = 1200
quantidade = 1


comando = f"""INSERT INTO Vendas (id_venda, data_venda, cliente, produto, preco, quantidade) VALUES
    ({id}, '{data}', '{cliente}', '{produto}', {preco}, {quantidade})"""

cursor.execute(comando)

cursor.commit()
```

AUTOMATIZANDO O CADASTRO DE DADOS POR MEIO DE VARIÁVEIS

Nesse momento, precisamos nos atentar a três detalhes importantes:

1) Devemos acrescentar um “f” no início do comando SQL, fora das aspas, para que o código entenda as variáveis como variáveis, não como texto;

2) As variáveis, dentro do INSERT INTO, devem ser informadas entre chaves { };

3) As variáveis que receberem valores não numéricos, ou seja, valores que estejam entre aspas, como é o caso das variáveis data, cliente e produto, no INSERT INTO também precisam ser envolvidas por aspas (porém, **aspas simples**):

```
comando = f"""INSERT INTO Vendas (id venda, data venda, cliente, produto, preco, quantidade) VALUES
    ({id}, '{data}', '{cliente}', '{produto}', {preco}, {quantidade})"""
```

AUTOMATIZANDO O CADASTRO DE DADOS POR MEIO DE VARIÁVEIS

Efetuadas as devidas alterações, executaremos nosso código para inserir o próximo registro na tabela Vendas (id = 4):

In [6]:
```
# incluindo registro id = 4 (automatizando com variáveis)

cursor = conexao.cursor()

id = 4
data = "17/06/2022"
cliente = "Diego"
produto = "Tablet"
preco = 1200
quantidade = 1


comando = f"""INSERT INTO Vendas (id_venda, data_venda, cliente, produto, preco, quantidade) VALUES
    ({id}, '{data}', '{cliente}', '{produto}', {preco}, {quantidade})"""

cursor.execute(comando)

cursor.commit()
```

AUTOMATIZANDO O CADASTRO DE DADOS POR MEIO DE VARIÁVEIS

Ao consultarmos a tabela Vendas pelo SSMS, veremos que o quarto registro também foi armazenado:

```
SELECT * FROM Vendas
```

Resultados | Mensagens

	id_venda	data_venda	cliente	produto	preco	quantidade
1	1	2022-04-22	Ana	Celular	2000.00	1
2	2	2022-05-10	Bruno	Notebook	3500.00	2
3	3	2022-05-13	Carla	Tablet	1200.00	1
4	4	2022-06-17	Diego	Tablet	1200.00	1

AUTOMATIZANDO O CADASTRO DE DADOS POR MEIO DE VARIÁVEIS

Se inserirmos mais um registro na tabela Vendas (id = 5) utilizando as variáveis...

In [7]:
```
# incluindo registro id = 5 (automatizando com variáveis)

cursor = conexao.cursor()

id = 5
data = "17/06/2022"
cliente = "Erica"
produto = "Celular"
preco = 1500
quantidade = 1


comando = f"""INSERT INTO Vendas (id_venda, data_venda, cliente, produto, preco, quantidade) VALUES
    ({id}, '{data}', '{cliente}', '{produto}', {preco}, {quantidade})"""

cursor.execute(comando)

cursor.commit()
```

AUTOMATIZANDO O CADASTRO DE DADOS POR MEIO DE VARIÁVEIS

... E consultarmos a tabela Vendas pelo SSMS, veremos que o quinto registro também foi armazenado:

```
SELECT * FROM Vendas
```

Resultados | Mensagens

	id_venda	data_venda	cliente	produto	preco	quantidade
1	1	2022-04-22	Ana	Celular	2000.00	1
2	2	2022-05-10	Bruno	Notebook	3500.00	2
3	3	2022-05-13	Carla	Tablet	1200.00	1
4	4	2022-06-17	Diego	Tablet	1200.00	1
5	5	2022-06-17	Erica	Celular	1500.00	1

AUTOMATIZANDO O CADASTRO DE DADOS POR MEIO DE VARIÁVEIS

Assim, encerramos o nosso primeiro projeto, o Case 1, no qual vimos como podemos inserir novos registros em uma tabela de um banco de dados armazenado no SQL Server via Python, utilizando o editor de códigos Jupyter Notebook.

Conforme observamos no início do módulo, o objetivo do Case 1 é mostrar como é possível efetuar essa integração entre as duas linguagens, SQL e Python, bem como os comandos básicos que podemos utilizar para escrever no banco de dados. Não nos aprofundamos em comandos e conceitos de Python por fugir ao objetivo almejado para este módulo.

Contudo, com os conhecimentos adquiridos até aqui, note que é possível efetuar diversas operações de escrita em bancos de dados do SQL Server. No exemplo apresentado, efetuamos inserções de novos registros com o INSERT INTO, mas você já será capaz de efetuar alterações com um UPDATE ou uma exclusão com um DELETE, por exemplo. Ficam, portanto, as sugestões para praticar! ☺

Agora, vamos ao próximo projeto deste módulo, o Case 2, em que efetuaremos uma leitura a um banco de dados SQL via Python.

CASE 2

LEITURA

EXPLICANDO O PROJETO DE INTEGRAÇÃO

Neste módulo, veremos novamente como integrar o SQL Server ao Python.

Porém, neste segundo Case, faremos um projeto de leitura de dados de um banco do SQL Server utilizando o Jupyter Notebook.

Essencialmente, o que queremos fazer aqui é ler algumas informações de uma tabela do banco de dados ContosoRetailDW, trazer tais informações para o Jupyter, agrupar esses dados para, por fim, criar um gráfico com base nesse agrupamento.

Para isso, dividiremos esse pequeno projeto em três etapas:

1ª Etapa: Importaremos as bibliotecas necessárias do Python e criaremos a conexão com o banco de dados;

2ª Etapa: Faremos a leitura de uma tabela armazenada em um banco de dados do SQL Server via Python;

3ª Etapa: Faremos agrupamentos básicos no Python e plotaremos um gráfico de barras.

Uma observação importante: o objetivo deste módulo é oferecer uma visão geral básica de como se estabelecer uma conexão entre as linguagens SQL e Python. Sendo assim, não nos aprofundaremos em comandos avançados nem em conceitos de Python. Para isso, sugerimos, aos que se interessarem, que façam algum curso de Python, como o nosso Python Impressionador! ☺

IMPORTANDO BIBLIOTECAS DO PYTHON E CRIANDO A CONEXÃO COM O BANCO

Para criar nosso projeto de leitura de dados do SQL Server via Python, utilizaremos dados da tabela DimProduct do banco de dados ContosoRetailDW.

Basicamente, vamos querer levar para o Python as informações constantes nos campos das colunas ColorName e UnitPrice da tabela DimProduct:

```
-- Projeto 2 - Integração Python e SQL (Leitura)

SELECT ColorName, UnitPrice FROM DimProduct
```

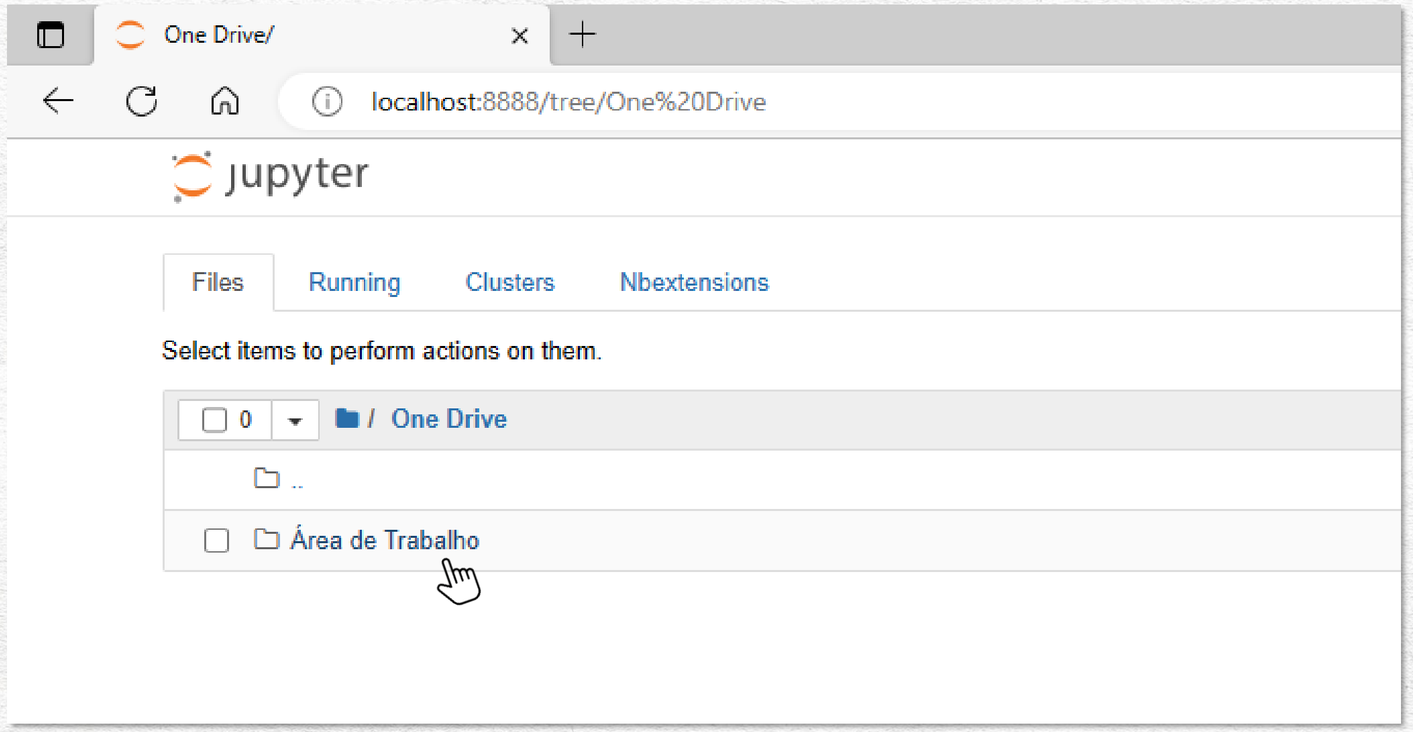
Resultados | Mensagens

	ColorName	UnitPrice
1	Silver	12,99
2	Blue	12,99
3	White	14,52
4	Silver	21,57
5	Red	21,57
6	Black	21,57
7	Blue	21,57
8	Silver	59,99
9	Black	59,99
10	Green	59,99
11	Orange	59,99
12	Blue	77,68
13	Black	77,68
14	Silver	77,68
15	White	77,68
16	White	109,95

ContosoRetailDW | 00:00:00 | 2.517 linhas

IMPORTANDO BIBLIOTECAS DO PYTHON E CRIANDO A CONEXÃO COM O BANCO

Então, inicialmente, precisamos abrir o Jupyter e acessar a pasta em que desejamos salvar o arquivo do nosso projeto:



IMPORTANDO BIBLIOTECAS DO PYTHON E CRIANDO A CONEXÃO COM O BANCO

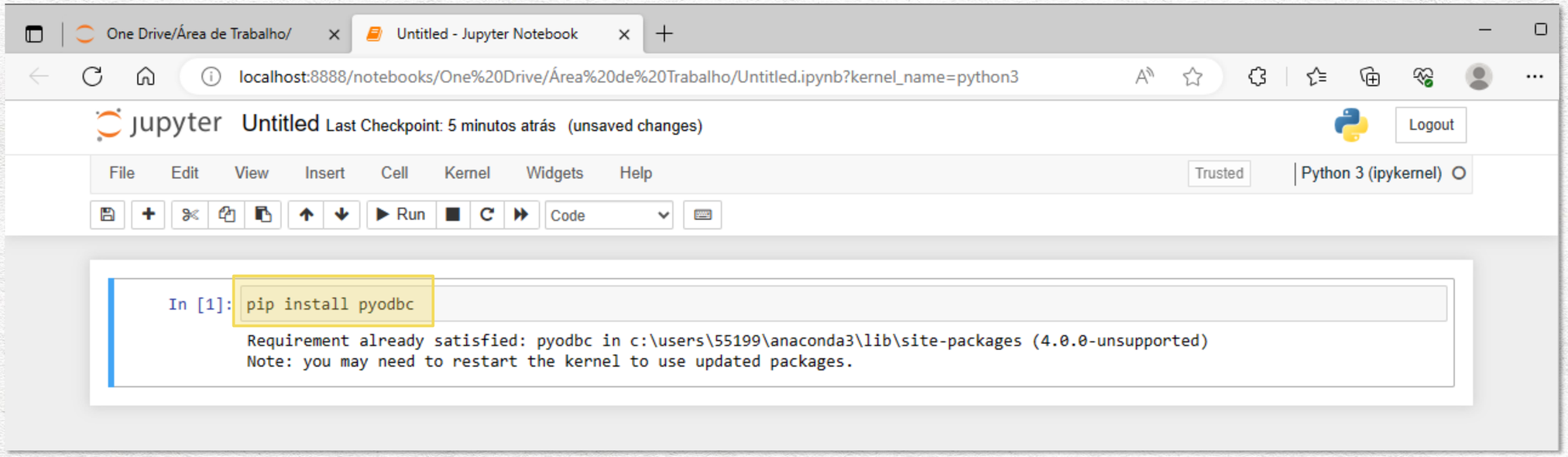
Em seguida, clicamos em New e selecionamos Python 3 (ipykernel):

IMPORTANDO BIBLIOTECAS DO PYTHON E CRIANDO A CONEXÃO COM O BANCO

Com a nova janela aberta, o primeiro passo é instalarmos a biblioteca do Python chamada pyodbc.

Essa biblioteca permite integrarmos o Python com o SQL Server.

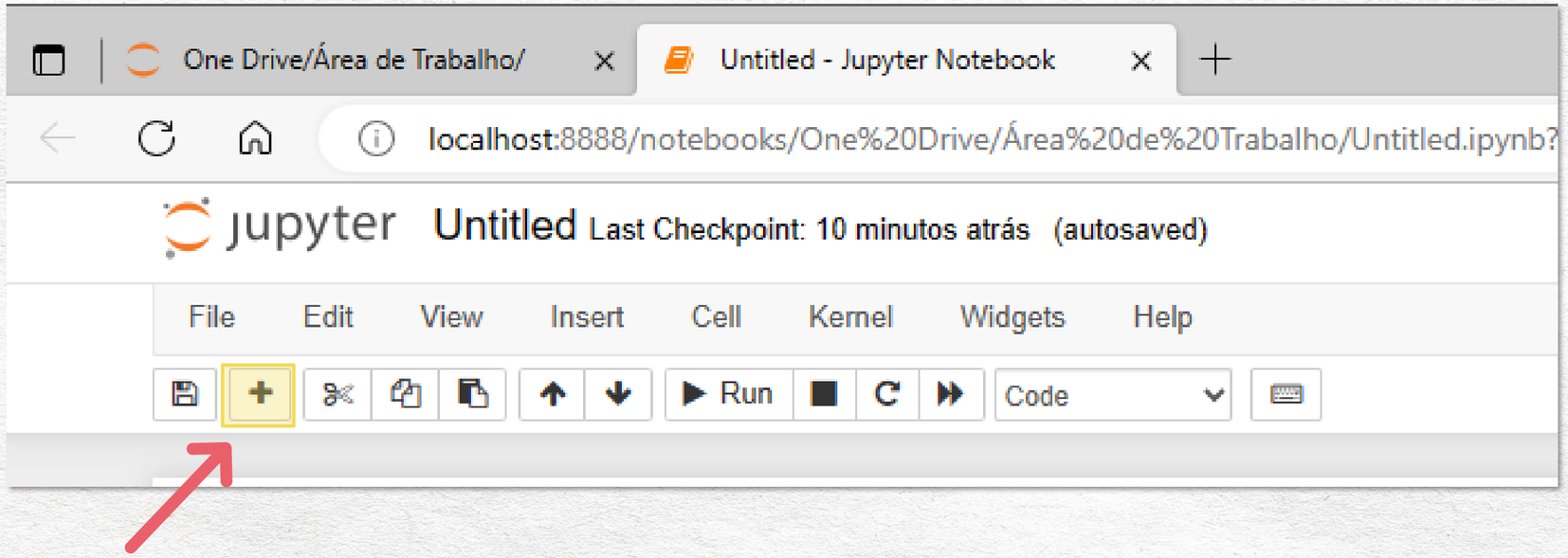
Se você já estudou o projeto anterior deste módulo, o de escrita, você já fez essa instalação antes. Porém, caso ainda não tenha instalado-a, digite o comando **pip install pyodbc** e aperte as teclas de atalho Crtl + Enter para executar o código:



IMPORTANDO BIBLIOTECAS DO PYTHON E CRIANDO A CONEXÃO COM O BANCO

Instalada a biblioteca pyodbc, podemos agora criar o nosso código de conexão com o SQL Server.

Para isso, vamos adicionar uma nova célula, clicando no ícone mostrado abaixo:



IMPORTANDO BIBLIOTECAS DO PYTHON E CRIANDO A CONEXÃO COM O BANCO

Nessa nova célula, vamos:

1) Importar as bibliotecas que vamos precisar;

2) Criar uma variável que receberá os três parâmetros necessários para conexão (Driver, Server e Database);

3) Efetuar a conexão:

```
In [ ]: # importanto as bibliotecas necessárias:

import pyodbc
import pandas as pd
import matplotlib

# criando dados de conexão:

dados_conexao = (
    "Driver={SQL Server};"
    "Server=LAPTOP-3U4522DD;"
    "Database=ContosoRetailDW;"
)

# conectando:

conexao = pyodbc.connect(dados_conexao)
print("Conexão Bem Sucedida")
```

1

2

3

IMPORTANDO BIBLIOTECAS DO PYTHON E CRIANDO A CONEXÃO COM O BANCO

Repare que:

1) Para importar as bibliotecas necessárias, utilizamos o comando abaixo:

```
import pyodbc
import pandas as pd
import matplotlib
```

A biblioteca **pyodbc** permite integrarmos o Python com o SQL Server;

Com a biblioteca **pandas**, podemos fazer análise de dados de forma eficiente. Seguindo o padrão comumente utilizado no mercado, importamos essa biblioteca atribuindo-lhe um alias: **pd**;

Já a biblioteca **matplotlib** nos auxilia a trabalhar com gráficos.

IMPORTANDO BIBLIOTECAS DO PYTHON E CRIANDO A CONEXÃO COM O BANCO

2) Para informar os três parâmetros necessários para conexão (Driver, Server e Database), criamos uma variável, que chamamos de **dados_conexao**, e informamos os dados de cada um dos parâmetros, sendo:

a) Driver: {SQL Server};

b) Server: o nome do computador (ATENÇÃO: cada máquina tem o seu próprio nome. O nome mostrado aqui no exemplo é do computador que estamos utilizando no momento. **Você precisará descobrir e informar o nome da sua máquina**. Mostraremos como fazer isso na próxima página;

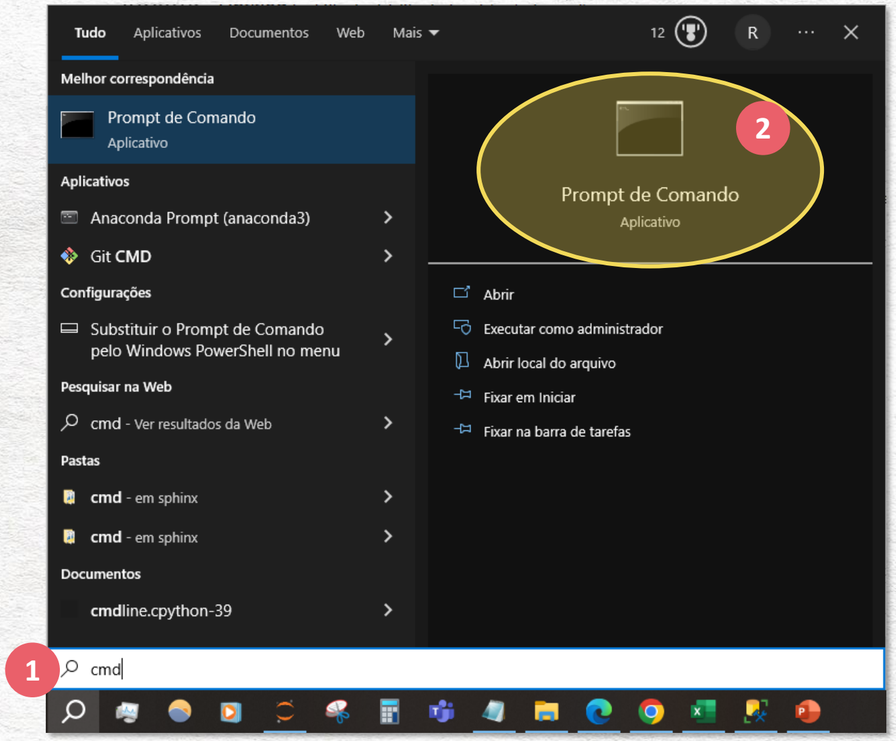
c) Database: ContosoRetailDW (que é o nome do banco no qual a tabela DimProduct, que utilizaremos neste projeto, está armazenada).

```
dados_conexao = (
    "Driver={SQL Server};"
    "Server=LAPTOP-3U4522DD;"
    "Database=ContosoRetailDW;"
)
```

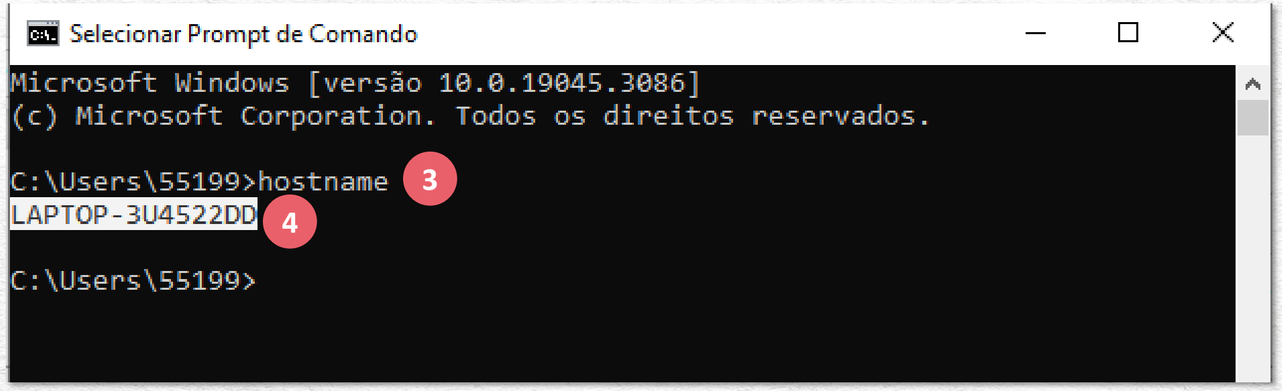
IMPORTANDO BIBLIOTECAS DO PYTHON E CRIANDO A CONEXÃO COM O BANCO

Para descobrir o nome do seu computador para informar ao parâmetro Server da página anterior, você pode fazer o seguinte:

1. Digite **cmd** na barra de pesquisas do seu computador para localizar o prompt de comando;
2. Clique em seu ícone para abri-lo;



3. No prompt que se abrir, digite **hostname** e aperte a tecla Enter;
4. Na linha de baixo, aparecerá o nome do seu computador. Selecione e copie esse nome (clique duplo sobre ele para selecionar, Ctrl + C para copiar).



IMPORTANDO BIBLIOTECAS DO PYTHON E CRIANDO A CONEXÃO COM O BANCO

Voltando ao Jupyter, cole (Ctrl + V) o nome do seu computador no parâmetro Server:

```
dados_conexao = (
    "Driver={SQL Server};"
    "Server=LAPTOP-3U4522DD;"
    "Database=ContosoRetailDW;"
)
```

3) Por fim, para efetuar a conexão, criamos uma variável, que chamamos de **conexao**, e passamos para ela a variável dados_conexao (que contém os parâmetros de conexão). Além disso, acrescentamos um **print** para mostrar na tela a mensagem “Conexão Bem Sucedida” quando conectarmos, assim, saberemos que deu tudo certo:

```
conexao = pyodbc.connect(dados_conexao)
print("Conexão Bem Sucedida")
```

IMPORTANDO BIBLIOTECAS DO PYTHON E CRIANDO A CONEXÃO COM O BANCO

Tudo pronto, agora podemos conectar. Para isso, executamos a célula (Ctrl + Enter):

Note que, enquanto o Jupyter executa o código, aparece um asterisco, indicando que o código está sendo executado:

```
In [*]: # importanto as bibliotecas necessárias:

import pyodbc
import pandas as pd
import matplotlib

# criando dados de conexão:

dados_conexao = (
    "Driver={SQL Server};"
    "Server=LAPTOP-3U4522DD;"
    "Database=ContosoRetailDW;"
)

# conectando:

conexao = pyodbc.connect(dados_conexao)
print("Conexão Bem Sucedida")
```

Finalizada a execução do código, o asterisco passa a ser um número, e abaixo da célula aparece a mensagem “Conexão Bem Sucedida”, que configuramos no código:

```
In [3]: # importanto as bibliotecas necessárias:

import pyodbc
import pandas as pd
import matplotlib

# criando dados de conexão:

dados_conexao = (
    "Driver={SQL Server};"
    "Server=LAPTOP-3U4522DD;"
    "Database=ContosoRetailDW;"
)

# conectando:

conexao = pyodbc.connect(dados_conexao)
print("Conexão Bem Sucedida")

Conexão Bem Sucedida
```

FAZENDO A LEITURA DE UMA TABELA DO SQL SERVER VIA PYTHON

Vejamos agora como executar a 2ª etapa do nosso projeto, para fazermos a leitura das informações que queremos extrair da tabela DimProduct do banco de dados ContosoRetailDW, armazenado no SQL Server, via Python, utilizando o Jupyter.

Vamos adicionar uma nova célula ao nosso arquivo do Jupyter para:

1 Criar uma variável, chamada **comando_sql**, que receberá o código SQL de consulta às colunas ColorName e UnitPrice da tabela DimProduct (SELECT), entre aspas;

2 Criar outra variável, chamada **dados**, que armazenará os dados extraídos do SQL Server. Para isso, passamos para essa variável o método **pd.read_sql** da biblioteca pandas, informando como seus parâmetros as variáveis comando_sql (criada acima, que contém o código SQL de consulta aos dados que queremos) e conexao (que contém as informações da conexão com o SQL Server):

```
In [ ]: # fazendo a leitura da tabela do SQL Server:

        comando_sql = "SELECT ColorName, UnitPrice FROM DimProduct"  1

        dados = pd.read_sql(comando_sql, conexao)  2
```

FAZENDO A LEITURA DE UMA TABELA DO SQL SERVER VIA PYTHON

Executamos o código:

```
In [5]: # fazendo a leitura da tabela do SQL Server:

        comando_sql = "SELECT ColorName, UnitPrice FROM DimProduct"

        dados = pd.read_sql(comando_sql, conexao)
```

E criamos uma nova célula, com o código abaixo, para poder visualizar o resultado armazenado na variável **dados**:

```
In [ ]: # visualizando os dados extraídos do SQL Server:

        display(dados)
```

FAZENDO A LEITURA DE UMA TABELA DO SQL SERVER VIA PYTHON

Executamos o código, e retornarmos na tela todos os campos das 2517 linhas das 2 colunas que extraímos da tabela DimProduct (ColorName e UnitPrice):

```
In [6]: # visualizando os dados extraídos do SQL Server:

        display(dados)
```

	ColorName	UnitPrice
0	Silver	12.99
1	Blue	12.99
2	White	14.52
3	Silver	21.57
4	Red	21.57
...	...	...
2512	Red	129.99
2513	White	129.99
2514	White	3.35
2515	Black	3.35
2516	Silver	3.35

2517 rows × 2 columns

AGRUPAMENTOS BÁSICOS E PLOTANDO UM GRÁFICO DE BARRAS

Partiremos agora para a 3ª etapa do nosso projeto, em que agruparemos a tabela que importamos do SQL Server para fazermos uma contagem de quantos produtos temos de cada cor.

Em seguida, plotaremos um gráfico de barras na tela do Jupyter.

Vejamos: para agruparmos a tabela por cor, junto à variável dados (que armazena a tabela importada do SQL Server), utilizaremos o método **groupby()**, passando para ele como parâmetro o nome da coluna pela qual queremos agrupar (ColorName). Em seguida, faremos uma contagem de linhas desse agrupamento para cada cor, utilizando a função **count()**:

```
In [ ]: # criando um agrupamento do total de produtos por cores:

        dados.groupby('ColorName').count()
```

Repare em um detalhe interessante: os métodos groupby() e count() do Python desempenham as mesmas funções do comando GROUP BY e da função de agregação COUNT do SQL. Quanto mais aprendemos a manipular diversas ferramentas e linguagens de programação, mais percebemos o quanto elas são semelhantes e se relacionam!

AGRUPAMENTOS BÁSICOS E PLOTANDO UM GRÁFICO DE BARRAS

Ao executarmos o código, retornamos na tela o agrupamento efetuado. Assim, descobrimos quantos produtos temos por cores:

```
In [6]: # criando um agrupamento do total de produtos por cores:

        dados.groupby('ColorName').count()
```

Out[6]:

ColorName	UnitPrice
Azure	14
Black	602
Blue	197
Brown	77
Gold	50
Green	74
Grey	283
Orange	55
Pink	84
Purple	6
Red	99
Silver	417
Silver Grey	14
Transparent	1
White	505
Yellow	36
blue	3

AGRUPAMENTOS BÁSICOS E PLOTANDO UM GRÁFICO DE BARRAS

Porém, da forma como criamos nosso agrupamento, repare que ele não ficou armazenado em lugar algum.

Dessa forma, não temos como transformá-lo em um gráfico.

Para fazer isso, primeiro precisamos armazenar a nossa tabela agrupada em uma variável. Assim, poderemos utilizar essa variável posteriormente para plotar nosso gráfico.

Portanto, vamos criar uma variável chamada **total_produtos_por_cor**, informar para ela o comando que cria o agrupamento, e executar o código para que a variável receba nossa tabela agrupada:

```
In [7]: # armazenando o agrupamento em uma variável:

        total_produtos_por_cor = dados.groupby('ColorName').count()
```

AGRUPAMENTOS BÁSICOS E PLOTANDO UM GRÁFICO DE BARRAS

Tudo certo! Agora nosso agrupamento está armazenado na variável total_produtos_por_cor.

Utilizando o comando **display** e a variável total_produtos_por_cor como seu parâmetro, podemos ver que a tabela agrupada foi de fato armazenada na referida variável:

```
In [8]: # visualizando o agrupamento armazenado na variável:

        display(total_produtos_por_cor)
```

ColorName	UnitPrice
Azure	14
Black	602
Blue	197
Brown	77
Gold	50
Green	74
Grey	283
Orange	55
Pink	84
Purple	6
Red	99
Silver	417
Silver Grey	14
Transparent	1
White	505
Yellow	36
blue	3

AGRUPAMENTOS BÁSICOS E PLOTANDO UM GRÁFICO DE BARRAS

Agora sim, podemos plotar um gráfico de barras utilizando os dados do agrupamento efetuado.

Para tanto, junto à variável total_produtos_por_cor (que armazena a tabela agrupada), utilizaremos a função **plot()**, passando para ela o argumento **kind='bar'**, que informa que queremos que seja mostrado na tela um gráfico do tipo (kind) barras (bar):

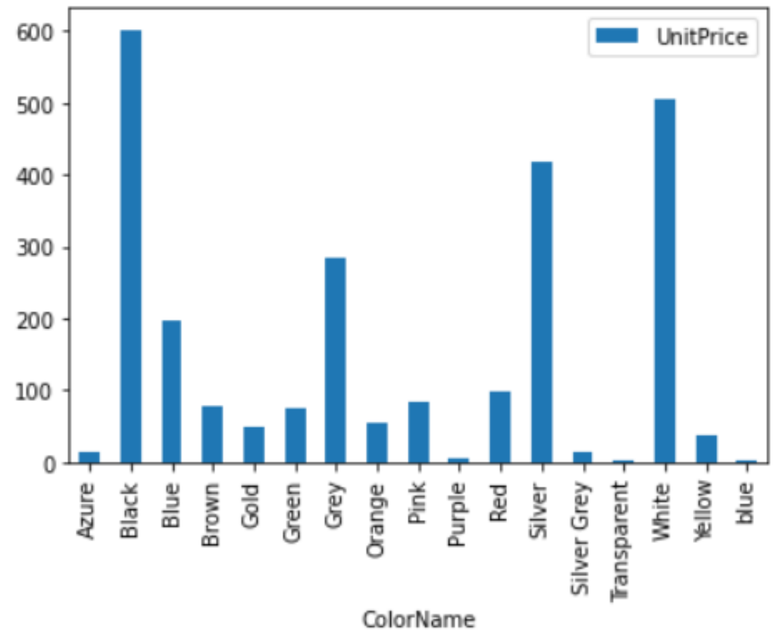
```
In [ ]: # plotando um gráfico do agrupamento:

        total_produtos_por_cor.plot(kind='bar')
```

AGRUPAMENTOS BÁSICOS E PLOTANDO UM GRÁFICO DE BARRAS

Ao executarmos o código, veja só o resultado:

```
In [10]: # plotando um gráfico do agrupamento:

         total_produtos_por_cor.plot(kind='bar')
```

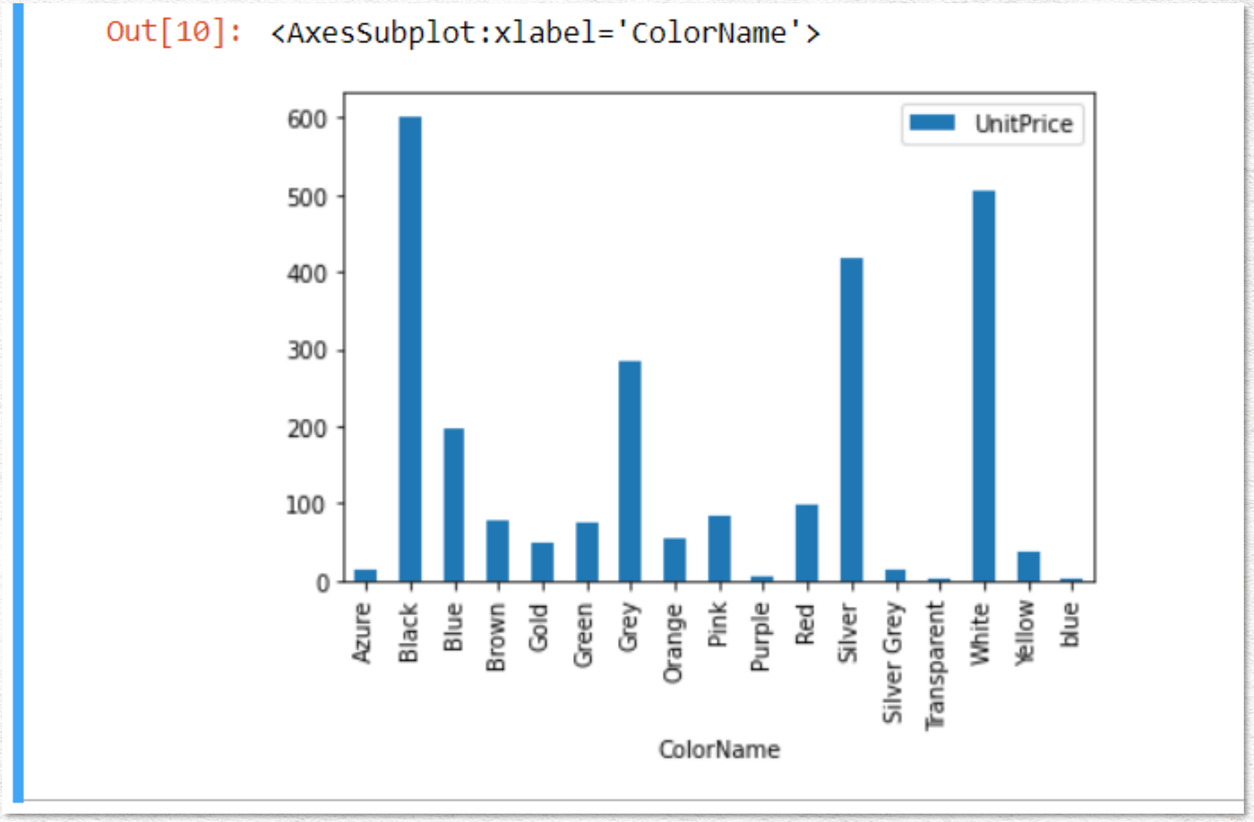
```
Out[10]: <AxesSubplot:xlabel='ColorName'>
```



AGRUPAMENTOS BÁSICOS E PLOTANDO UM GRÁFICO DE BARRAS

Assim, conseguimos visualizar melhor o resultado do nosso agrupamento.

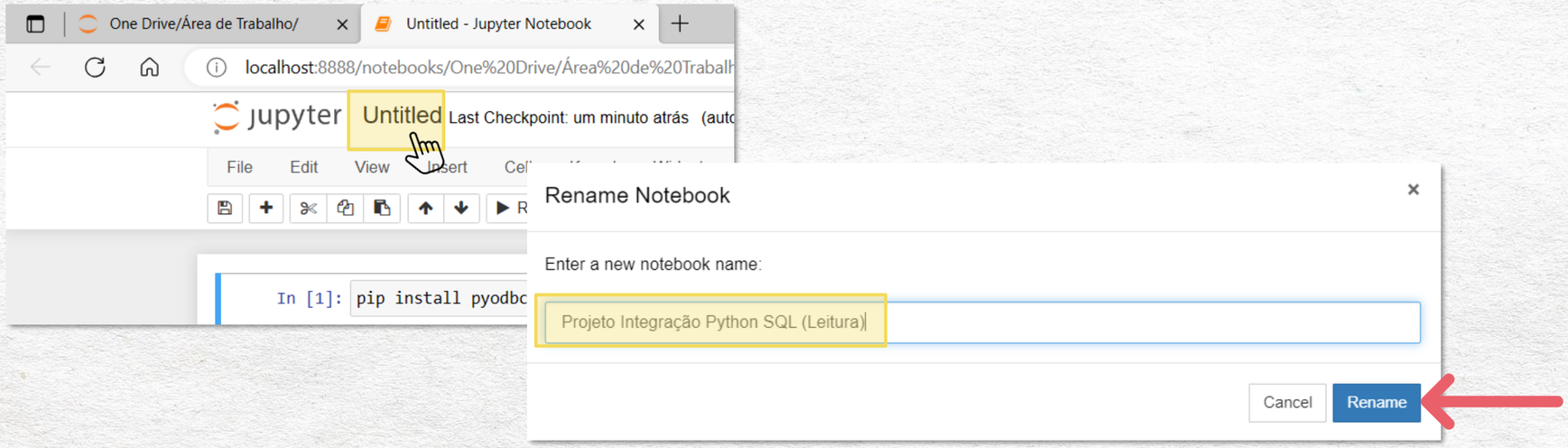
Logo no primeiro momento, já podemos perceber que a cor com maior quantidade de produtos é a Black (preta), seguida da White (branca), Silver (prata), Grey (cinza), etc.



AGRUPAMENTOS BÁSICOS E PLOTANDO UM GRÁFICO DE BARRAS

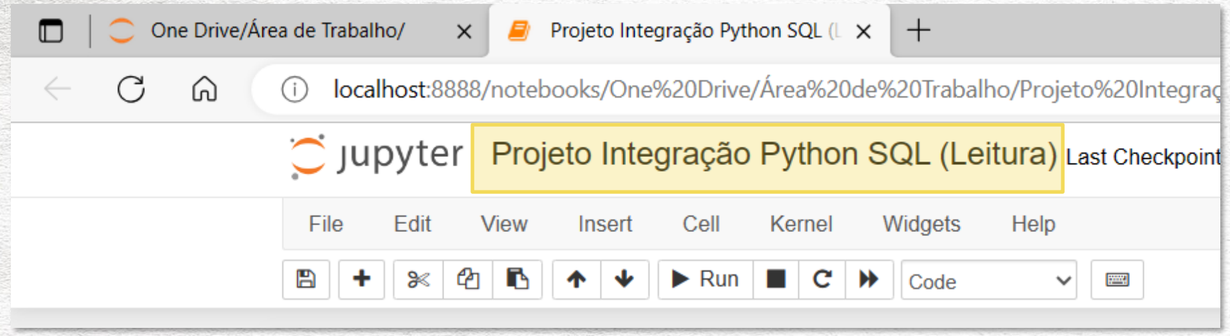
Para finalizar, vamos renomear esse arquivo no qual escrevemos o nosso código / script, para que, sempre que quisermos, possamos facilmente localizá-lo na pasta em que o salvamos.

Para renomeá-lo, basta clicar sobre a palavra “Untitled” na parte superior da página e, na janelinha que se abrir, digitar o novo nome do arquivo e clicar em “Rename”:

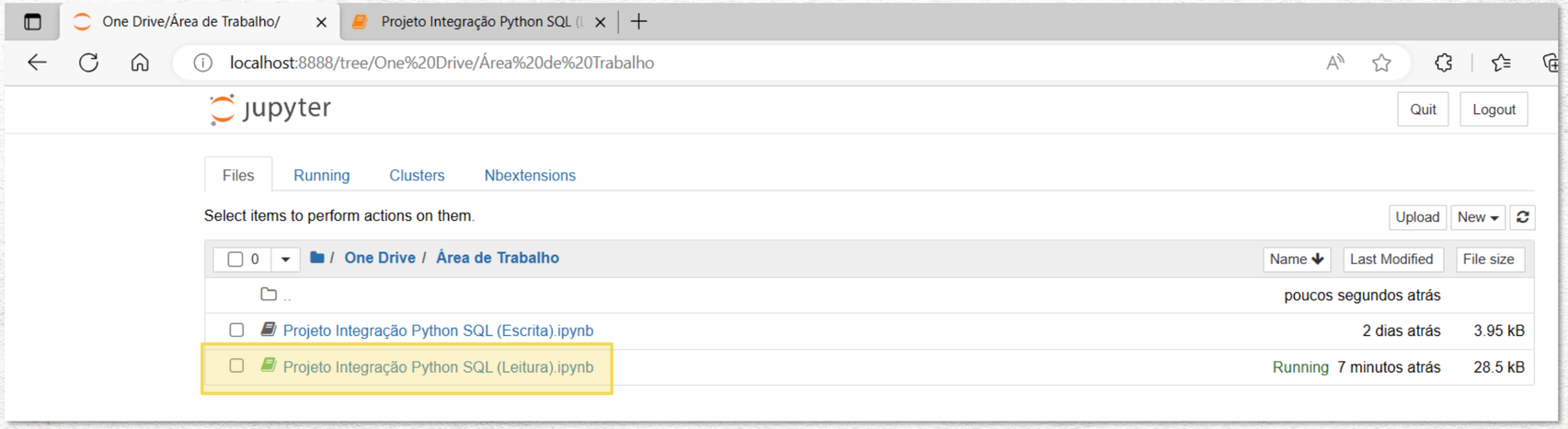


AGRUPAMENTOS BÁSICOS E PLOTANDO UM GRÁFICO DE BARRAS

Veja que o arquivo agora aparece com o nome informado:



E se voltarmos à pasta onde o abrimos inicialmente, lá estará ele, com o novo nome:



AGRUPAMENTOS BÁSICOS E PLOTANDO UM GRÁFICO DE BARRAS

Chegamos ao fim deste projeto de leitura de dados armazenados em um banco de dados do SQL Server via Python, bem como deste módulo de integração.

Conforme observamos no início, o objetivo principal deste Case 2 é mostrar como é possível efetuar essa integração entre as duas linguagens, SQL e Python, bem como os comandos básicos que podemos utilizar para ler dados de um banco do SQL Server. Não nos aprofundamos em comandos e conceitos de Python por fugir ao objetivo almejado para este módulo.

Contudo, com os conhecimentos adquiridos até aqui, perceba que você agora já é capaz de efetuar a leitura dos dados que desejar extrair de um banco de dados do SQL Server para analisá-los via Python, utilizando o Jupyter Notebook, criar os agrupamentos necessários, assim como plotar gráficos para melhor visualizar tais dados.

Esperamos que tenha gostado! ☺

SQL IMPRESSIONADOR

Apostila Completa do Módulo de
Integração SQL Server e Python