

## ✓ Engineering with Python and Cassandra

### ✓ Pre-requisite

```
# Install the Cassandra python driver
!pip install cassandra-driver
```

```
➦ Collecting cassandra-driver
  Downloading cassandra_driver-3.29.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (6.2 kB)
Collecting geomet<0.3,>=0.1 (from cassandra-driver)
  Downloading geomet-0.2.1.post1-py3-none-any.whl.metadata (1.0 kB)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from geomet<0.3,>=0.1->cassandra-driver) (8.1.7)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from geomet<0.3,>=0.1->cassandra-driver) (1.16.0)
Downloading cassandra_driver-3.29.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.9 MB)
----- 3.9/3.9 MB 34.7 MB/s eta 0:00:00
Downloading geomet-0.2.1.post1-py3-none-any.whl (18 kB)
Installing collected packages: geomet, cassandra-driver
Successfully installed cassandra-driver-3.29.2 geomet-0.2.1.post1
```

```
# Import the necessary libraries
from cassandra.cluster import Cluster
from cassandra.auth import PlainTextAuthProvider
import json
```

### ✓ Creating a Cassandra database

Here's a step-by-step tutorial on how to install Cassandra on DataStax Astra and how to connect to it using Python. We don't need to give any credit card details for this option.

#### Step 1: Sign up for DataStax Astra

To use DataStax Astra, you must first sign up for an account. Go to the DataStax Astra website (<https://astra.datastax.com/register>) and sign up for a free account.

#### Step 2: Create a database

Once you have created an account and logged in, you can create a new database. Click on the "Create Database" button and follow the prompts to create a new database.

#### Step 3: Create a keyspace

After creating a database, you need to create a keyspace. Click on the "Add Keyspace" button and follow the prompts to create a new keyspace.

#### Step 4: Generate an application token

To connect to your Cassandra database using Python, you'll need to generate an application token. Go to the "Settings" tab and click on the "Generate New Token" button. Copy the token that is generated.

### ✓ 1. Setting up the Connection

```
cloud_config= {
    'secure_connect_bundle': 'secure-connect-db-sales.zip'
}
with open("db_sales-token.json") as f:
    secrets = json.load(f)
CLIENT_ID = secrets["clientId"]
CLIENT_SECRET = secrets["secret"]
auth_provider = PlainTextAuthProvider(CLIENT_ID, CLIENT_SECRET)
cluster = Cluster(cloud=cloud_config, auth_provider=auth_provider)
session = cluster.connect()
if session:
    print('Connected!')
else:
    print("An error occurred.")
```

```
➦ WARNING:cassandra.cluster:Downgrading core protocol version from 66 to 65 for 02a72af1-d4a4-43cf-a139-9b587c46db84-eu-west-1.db.astra.da
WARNING:cassandra.cluster:Downgrading core protocol version from 65 to 5 for 02a72af1-d4a4-43cf-a139-9b587c46db84-eu-west-1.db.astra.dat
```

WARNING:cassandra.cluster:Downgrading core protocol version from 5 to 4 for 02a72af1-d4a4-43cf-a139-9b587c46db84-eu-west-1.db.astra.data  
Connected!

## 2. Loading and Reading Data from Cassandra

```
session.execute("Use sales;")
session.execute("""
CREATE TABLE IF NOT EXISTS bronze_table (
    id UUID PRIMARY KEY,
    region TEXT,
    country TEXT,
    item_type TEXT,
    sales_channel TEXT,
    order_priority TEXT,
    order_date DATE,
    order_id BIGINT,
    ship_date DATE,
    units_sold INT,
    unit_price FLOAT,
    unit_cost FLOAT,
    total_revenue FLOAT,
    total_cost FLOAT,
    total_profit FLOAT
)
""")
<cassandra.cluster.ResultSet at 0x7cd50a643790>
```

```
import pandas as pd

# Load the CSV file
file = 'sales_100.csv'
data = pd.read_csv(file)

# Display the first few rows to verify
print(data.head())
```

```

Region      Country  Item Type Sales Channel \
0  Sub-Saharan Africa  South Africa  Fruits  Offline
1  Middle East and North Africa  Morocco  Clothes  Online
2  Australia and Oceania  Papua New Guinea  Meat  Offline
3  Sub-Saharan Africa  Djibouti  Clothes  Offline
4  Europe  Slovakia  Beverages  Offline

Order Priority  Order Date  Order ID  Ship Date  UnitsSold  UnitPrice \
0      M  7/27/2012  443368995  7/28/2012  1593  9.33
1      M  9/14/2013  667593514  10/19/2013  4611  109.28
2      M  5/15/2015  940995585  6/4/2015  360  421.89
3      H  5/17/2017  880811536  7/2/2017  562  109.28
4      L  10/26/2016  174590194  12/4/2016  3973  47.45

UnitCost  TotalRevenue  TotalCost  TotalProfit
0      6.92  14862.69  11023.56  3839.13
1     35.84  503890.08  165258.24  338631.84
2    364.69  151880.40  131288.40  20592.00
3     35.84  61415.36  20142.08  41273.28
4     31.79  188518.85  126301.67  62217.18
```

```
# Ensure dates are properly formatted
data['Order Date'] = pd.to_datetime(data['Order Date'], format='%m/%d/%Y').dt.date
data['Ship Date'] = pd.to_datetime(data['Ship Date'], format='%m/%d/%Y').dt.date
```

```
# Preview data types
print(data.dtypes)
```

```

Region      object
Country     object
Item Type   object
Sales Channel  object
Order Priority object
Order Date   object
Order ID     int64
```

```

Ship Date      object
UnitsSold      int64
UnitPrice      float64
UnitCost       float64
TotalRevenue   float64
TotalCost      float64
TotalProfit    float64
dtype: object

```

```
from datetime import datetime
```

```

for _, row in data.iterrows():
    session.execute("""
        INSERT INTO bronze_table (
            id,
            region,
            country,
            item_type,
            sales_channel,
            order_priority,
            order_date,
            order_id,
            ship_date,
            units_sold,
            unit_price,
            unit_cost,
            total_revenue,
            total_cost,
            total_profit
        )
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    """, (
        uuid.uuid4(), # Generate unique UUID
        row['Region'],
        row['Country'],
        row['Item Type'],
        row['Sales Channel'],
        row['Order Priority'],
        row['Order Date'],
        int(row['Order ID']),
        row['Ship Date'],
        int(row['UnitsSold']),
        float(row['UnitPrice']),
        float(row['UnitCost']),
        float(row['TotalRevenue']),
        float(row['TotalCost']),
        float(row['TotalProfit'])
    ))

```

```

rows = session.execute("SELECT * FROM bronze_table LIMIT 10;")
for row in rows:
    print(row)

```

```

➡ Row(id=UUID('343391bb-5d0f-4205-8589-fefed0d2d482'), country='Dominica', item_type='Beverages', order_date=Date(15503), order_id=4380118
Row(id=UUID('0782d541-5e5f-4278-9cec-affa56f2b72f'), country='Serbia', item_type='Clothes', order_date=Date(16988), order_id=925136649,
Row(id=UUID('981eed99-6906-4fa7-8300-d6f48acb9988'), country='The Bahamas', item_type='Fruits', order_date=Date(14778), order_id=4881211
Row(id=UUID('c805de87-0906-4f5d-9be1-c187e8482634'), country='Liberia', item_type='Baby Food', order_date=Date(16592), order_id=14663470
Row(id=UUID('663a832e-e87f-4ef6-8540-ec7aac7d22dd'), country='Switzerland', item_type='Office Supplies', order_date=Date(16280), order_i
Row(id=UUID('437233f1-4358-452b-99f5-d63e37ace658'), country='Haiti', item_type='Office Supplies', order_date=Date(14974), order_id=4850
Row(id=UUID('ba26b737-cffb-44c5-acc6-9db6a9482f6d'), country='Burundi', item_type='Beverages', order_date=Date(15042), order_id=52927650
Row(id=UUID('6519af68-df7c-457d-bed2-15640e2a6bfd'), country='Montenegro', item_type='Clothes', order_date=Date(17048), order_id=9025116
Row(id=UUID('af5a8bdb-d39f-4807-83ad-5d96d473af6b'), country='Indonesia', item_type='Household', order_date=Date(15245), order_id=520480
Row(id=UUID('27bebab6-826a-4dde-b540-19778c0eeddf'), country='Tonga', item_type='Baby Food', order_date=Date(16932), order_id=839094388,

```

## ▼ Silver table

## ▼ Cleaning data

```

rows = session.execute("SELECT * FROM bronze_table;")
# Convert rows to a list of dictionaries
data = [dict(row._asdict()) for row in rows]

```

```
# Create a DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print(df.head(2))
```

```
↵ id country item_type order_date \
0 343391bb-5d0f-4205-8589-fefed0d2d482 Dominica Beverages 2012-06-12
1 0782d541-5e5f-4278-9cec-affa56f2b72f Serbia Clothes 2016-07-06

order_id order_priority region sales_channel \
0 438011872 L Central America and the Caribbean Online
1 925136649 L Europe Offline

ship_date total_cost total_profit total_revenue unit_cost \
0 2012-07-18 200308.796875 98673.65625 298982.4375 31.790001
1 2016-07-13 263352.312500 539637.12500 802989.4375 35.840000

unit_price units_sold
0 47.450001 6301
1 109.279999 7348
```

▼ checking for missing values

```
print(df.isnull().sum())
```

```
↵ id 0
country 0
item_type 0
order_date 0
order_id 0
order_priority 0
region 0
sales_channel 0
ship_date 0
total_cost 0
total_profit 0
total_revenue 0
unit_cost 0
unit_price 0
units_sold 0
dtype: int64
```

Double-click (or enter) to edit

```
print(df.duplicated().sum())
```

```
↵ 0
```

```
print(df['region'].unique())
print(df['sales_channel'].unique())
```

```
↵ ['Central America and the Caribbean' 'Europe' 'Sub-Saharan Africa' 'Asia'
'Australia and Oceania' 'Middle East and North Africa' 'North America']
['Online' 'Offline']
```

```
print(df.info())
```

```
↵ <class 'pandas.core.frame.DataFrame'>
RangeIndex: 99 entries, 0 to 98
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   id                    99 non-null    object
1   country               99 non-null    object
2   item_type             99 non-null    object
3   order_date            99 non-null    object
4   order_id              99 non-null    int64
5   order_priority        99 non-null    object
6   region                99 non-null    object
7   sales_channel         99 non-null    object
```

```

8  ship_date      99 non-null    object
9  total_cost     99 non-null    float64
10 total_profit   99 non-null    float64
11 total_revenue  99 non-null    float64
12 unit_cost      99 non-null    float64
13 unit_price     99 non-null    float64
14 units_sold     99 non-null    int64
dtypes: float64(5), int64(2), object(8)
memory usage: 11.7+ KB
None

```

## ▼ Data looks clean

```

session.execute("""
CREATE TABLE IF NOT EXISTS silver_table (
    id UUID PRIMARY KEY,
    region TEXT,
    country TEXT,
    item_type TEXT,
    sales_channel TEXT,
    order_priority TEXT,
    order_date DATE,
    order_id BIGINT,
    ship_date DATE,
    units_sold INT,
    unit_price FLOAT,
    unit_cost FLOAT,
    total_revenue FLOAT,
    total_cost FLOAT,
    total_profit FLOAT
)
""")

↩ <cassandra.cluster.ResultSet at 0x7d5b96b11a80>

for _, row in df.iterrows():
    session.execute("""
        INSERT INTO silver_table (
            id,
            region,
            country,
            item_type,
            sales_channel,
            order_priority,
            order_date,
            order_id,
            ship_date,
            units_sold,
            unit_price,
            unit_cost,
            total_revenue,
            total_cost,
            total_profit
        )
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    """, (
        uuid.uuid4(),
        row['region'],
        row['country'],
        row['item_type'],
        row['sales_channel'],
        row['order_priority'],
        row['order_date'],
        int(row['order_id']),
        row['ship_date'],
        int(row['units_sold']),
        float(row['unit_price']),
        float(row['unit_cost']),
        float(row['total_revenue']),
        float(row['total_cost']),
        float(row['total_profit'])
    ))

print("Data inserted successfully.")

```

➡ Data inserted successfully.

```
rows = session.execute("SELECT * FROM silver_table;")
# Convert rows to a list of dictionaries
data = [dict(row._asdict()) for row in rows]
```

```
# Create a DataFrame
silver_df = pd.DataFrame(data)
```

```
# Display the DataFrame
print(silver_df.head(2))
```

➡

		id	country	item_type	order_date	\
0	1df4f488-c502-4a38-bbe1-96dfe8814439	Belgium	Personal Care	2011-11-01		
1	9fa31f82-1c43-42a1-a0d9-d0337e72848d	Iceland	Baby Food	2010-10-02		

  

	order_id	order_priority	region	sales_channel	ship_date	total_cost	\
0	222504317	H	Europe	Online	2011-11-20	160206.09375	
1	678230941	M	Europe	Offline	2010-11-03	392492.03125	

  

	total_profit	total_revenue	unit_cost	unit_price	units_sold
0	70844.617188	231050.703125	56.669998	81.730003	2827
1	236007.312500	628499.375000	159.419998	255.279999	2462

## ✓ 1. Lets see the profitability with sales channel

```
session.execute('''
CREATE TABLE IF NOT EXISTS profitability_by_sales_channel (
    sales_channel TEXT PRIMARY KEY,
    total_orders INT,
    total_units_sold INT,
    total_revenue FLOAT,
    total_profit FLOAT,
    avg_order_value FLOAT,
    avg_profit_margin FLOAT
);''')
```

➡ <cassandra.cluster.ResultSet at 0x7d5b66547970>

```
channel_data = silver_df.groupby('sales_channel').agg({
    'order_id': 'nunique',
    'units_sold': 'sum',
    'total_revenue': 'sum',
    'total_profit': 'sum'
}).reset_index()
```

```
channel_data.rename(columns={'order_id': 'TotalOrders'}, inplace=True)
channel_data['AvgOrderValue'] = channel_data['total_revenue'] / channel_data['TotalOrders']
channel_data['AvgProfitMargin'] = channel_data['total_profit'] / channel_data['total_revenue']
```

```
for _, row in channel_data.iterrows():
    session.execute("""
        INSERT INTO profitability_by_sales_channel (
            sales_channel, total_orders, total_units_sold, total_revenue, total_profit, avg_order_value, avg_profit_margin
        ) VALUES (%s, %s, %s, %s, %s, %s, %s)
    """, (
        row['sales_channel'],
        int(row['TotalOrders']),
        int(row['units_sold']),
        float(row['total_revenue']),
        float(row['total_profit']),
        float(row['AvgOrderValue']),
        float(row['AvgProfitMargin'])
    ))
```

```
rows = session.execute("SELECT * FROM profitability_by_sales_channel;")
# Convert rows to a list of dictionaries
data2 = [dict(row._asdict()) for row in rows]
```

```
# Create a DataFrame
profitability_by_sales_channel_df = pd.DataFrame(data2)
```

```
# Display the DataFrame
profitability_by_sales_channel_df.head()
```

	sales_channel	avg_order_value	avg_profit_margin	total_orders	total_profit	total_revenue	total_units_sold
0	Online	1434376.000	0.294982	59	24963806.0	84628184.0	308320
1	Offline	1501544.875	0.272443	40	16363386.0	60061792.0	186490

Next steps:

[Generate code with profitability\\_by\\_sales\\_channel\\_df](#)

[View recommended plots](#)

[New interactive sheet](#)

Online mode has more profit

## 2. sales\_performance\_region

```
session.execute('''
CREATE TABLE IF NOT EXISTS sales_performance_region (
    region TEXT PRIMARY KEY,
    total_units_sold INT,
    total_revenue FLOAT,
    total_profit FLOAT,
    avg_profit_margin FLOAT
);
''')

<cassandra.cluster.ResultSet at 0x7d5b4ba73ca0>

region_data = silver_df.groupby('region').agg({
    'units_sold': 'sum',
    'total_revenue': 'sum',
    'total_profit': 'sum'
}).reset_index()
region_data['AvgProfitMargin'] = (region_data['total_profit'] / region_data['total_revenue']) * 100

for _, row in region_data.iterrows():
    session.execute("""
        INSERT INTO sales_performance_region (
            region, total_units_sold, total_revenue, total_profit, avg_profit_margin
        ) VALUES (%s, %s, %s, %s, %s)
    """, (
        row['region'],
        int(row['units_sold']),
        float(row['total_revenue']),
        float(row['total_profit']),
        float(row['AvgProfitMargin'])
    ))

rows = session.execute("SELECT * FROM sales_performance_region;")
data4 = [dict(row._asdict()) for row in rows]
sales_performance_region_df = pd.DataFrame(data4)
sales_performance_region_df.head()
```

	region	avg_profit_margin	total_profit	total_revenue	total_units_sold
0	Australia and Oceania	32.553974	3486940.0	10711258.0	42328
1	Europe	32.224686	11267281.0	34964748.0	121002
2	Middle East and North Africa	26.304173	6514262.0	24765128.0	60376
3	Central America and the Caribbean	24.200899	4252300.0	17570836.0	53641
4	Asia	23.403975	6749896.0	28840812.0	113129

Next steps:

[Generate code with sales\\_performance\\_region\\_df](#)

[View recommended plots](#)

[New interactive sheet](#)

## 3. Top Selling Product Categories

```
session.execute('''
CREATE TABLE IF NOT EXISTS top_selling_categories (
    item_type TEXT PRIMARY KEY,
    total_units_sold INT,
    total_revenue FLOAT,
    total_profit FLOAT,
    avg_unit_price FLOAT
);''')

category_data = df.groupby('item_type').agg({
    'units_sold': 'sum',
    'total_revenue': 'sum',
    'total_profit': 'sum'
}).reset_index()
category_data['AvgUnitPrice'] = category_data['total_revenue'] / category_data['units_sold']
for _, row in category_data.iterrows():
    session.execute("""
        INSERT INTO top_selling_categories (
            item_type, total_units_sold, total_revenue, total_profit, avg_unit_price
        ) VALUES (%s, %s, %s, %s, %s)
    """, (
        row['item_type'],
        int(row['units_sold']),
        float(row['total_revenue']),
        float(row['total_profit']),
        float(row['AvgUnitPrice'])
    ))

rows = session.execute("SELECT * FROM top_selling_categories;")
data4 = [dict(row._asdict()) for row in rows]
sales_performance_region_df = pd.DataFrame(data4)
sales_performance_region_df.head()
```

	item_type	avg_unit_price	total_profit	total_revenue	total_units_sold
0	Household	668.270020	9.552677e+06	3.851908e+07	57640
1	Office Supplies	651.210022	5.405268e+06	2.788090e+07	42814
2	Vegetables	154.059998	4.651418e+05	1.135114e+06	7368
3	Snacks	152.580002	7.927478e+05	2.193643e+06	14377
4	Personal Care	81.729996	9.784677e+05	3.191148e+06	39045

Next steps:

[Generate code with sales\\_performance\\_region\\_df](#)

[View recommended plots](#)

[New interactive sheet](#)

Start coding or [generate](#) with AI.