

✓ Cassandra Assignment

✓ 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 (3.9 MB)
  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)
  Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from geomet)
  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 22.8 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

I have created database Assignment and keyspace cassandra

✓ Setting up the Connection

```
# This secure connect bundle is autogenerated when you download your SCB,
# if yours is different update the file name below
cloud_config= {
    'secure_connect_bundle': 'secure-connect-assignment.zip'
}

# This token JSON file is autogenerated when you download your token,
# if yours is different update the file name below
with open("Assignment-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 a41a4f43-f0...
 WARNING:cassandra.cluster:Downgrading core protocol version from 65 to 5 for a41a4f43-f0...
 WARNING:cassandra.cluster:Downgrading core protocol version from 5 to 4 for a41a4f43-f0...
 Connected!

✓ Loading CSV into cassandra bronze table

```

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

```

 <cassandra.cluster.ResultSet at 0x7c4945e671f0>

```

session.execute("""
DROP TABLE IF EXISTS cassandra.bronze_sales_table;
""")

```



```

        uuid.uuid4(),
        row['Region'],
        row['Country'],
        row['Item Type'],
        row['Sales Channel'],
        row['Order Priority'],
        order_date,
        int(row['Order ID']),
        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_sales_table;")
data = []
for row in rows:
    data.append(row._asdict())
df = pd.DataFrame(data)

```

```
df.head()
```



	id	country	item_type	order_date	order_id	order_priority	region
0	e1842f99-0371-411a-99e6-f7a9ba1869e2	Dominican Republic	Baby Food	2011-08-25	824714744	H	Central America and the Caribbean
1	9de8c266-0f02-442c-8a7b-45b035e76b6b	Singapore	Snacks	2013-01-28	176461303	C	Asia
2	6599eaef-6baf-487f-8ae1-4171d03a8d1f	The Gambia	Fruits	2011-11-20	862861335	C	Sub-Saharan Africa
3	79fdf3f3-3b28-47d4-afdd-146999eabfa3	South Africa	Fruits	2012-07-27	443368995	M	Sub-Saharan Africa
4	d96e61b8-f9e7-44d5-9753-7b6746cd2f7a	Moldova	Fruits	2013-01-27	180908620	C	Europe



Next steps:

[Generate code with df](#)

[View recommended plots](#)

[New interactive sheet](#)

✓ Before proceeding with creation of silver table I'm performing Data Profiling on the bronze data

```
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
```

✓ No null values

```
continous_columns = ['units_sold', 'unit_price', 'unit_cost', 'total_revenue', 'total_cost',
check_zeros = (df[continous_columns] == 0).sum()
print(check_zeros)
```

```
>>> units_sold      0
     unit_price     0
     unit_cost      0
     total_revenue  0
     total_cost     0
     total_profit   0
dtype: int64
```

No Zeros

```
df['Ship Date Error'] = df.apply(
    lambda row: row['ship_date'] < row['order_date'] if row['order_date'] and row['ship_date']
    axis=1
)

# Display rows where Ship Date is before Order Date
error_rows = df[df['Ship Date Error'] == True]
print("Rows where Ship Date is before Order Date:")
print(error_rows)
```

→ Rows where Ship Date is before Order Date:
 Empty DataFrame
 Columns: [id, country, item_type, order_date, order_id, order_priority, region, sales_channel]
 Index: []

```
df['Ship Date Error'].value_counts()
```

→

	count
Ship Date Error	
False	99

dtype: int64

NO Corner Cases

```
df = df.drop(columns=['Ship Date Error'])
```

Since no updates required with the data as checked earlier i create silver table as it is

```
session.execute("""
    CREATE TABLE IF NOT EXISTS silver_sales_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
    );
    """
)

for _, row in df.iterrows():
    session.execute("""
    INSERT INTO silver_sales_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, t
    ) 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'],
        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'])
    ))

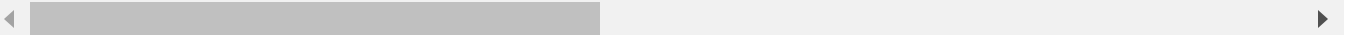
rows = session.execute("SELECT * FROM silver_sales_table;")
data = []
for row in rows:
    data.append(row._asdict())
df = pd.DataFrame(data)

df.head()

```



	id	country	item_type	order_date	order_id	order_priority	region
0	65d1ad5f-f591-40c2-b264-0b4a1b656379	Kuwait	Household	2011-06-13	459386289	C	Middle East and North Africa
1	c63035a2-1183-4605-8e4c-5786a8518811	Dominican Republic	Baby Food	2011-08-25	824714744	H	Central America and the Caribbean
2	547364f5-25e7-4b94-8979-6f5ae009322a	India	Snacks	2012-10-10	440306556	L	Asia
3	b4817e82-bdd3-4e53-a663-98ec3a118c96	Pakistan	Meat	2013-12-28	500371730	M	Middle East and North Africa
4	91d75dfb-04dc-40dc-aefd-614e1805b405	Romania	Cereal	2015-04-16	633134210	M	Europe



Next steps:

[Generate code with df](#)

[View recommended plots](#)

[New interactive sheet](#)

Creating Gold Tables

✓ 1. creating table which shows revenue by country

```
rows = session.execute("SELECT country, total_revenue FROM silver_sales_table;")
data = [row._asdict() for row in rows]
df = pd.DataFrame(data)
```

```
top_countries_df = (
    df.groupby("country", as_index=False)["total_revenue"]
      .sum()
      .sort_values(by="total_revenue", ascending=False)
)
```

```
session.execute("""
CREATE TABLE IF NOT EXISTS gold_top_countries (
    country TEXT PRIMARY KEY,
```



```
        total_revenue FLOAT
    );
    """

for _, row in top_countries_df.iterrows():
    session.execute("""
        INSERT INTO gold_top_countries (country, total_revenue)
        VALUES (%s, %s);
        """, (row["country"], row["total_revenue"]))
```

```
rows = session.execute("SELECT * FROM gold_top_countries;")
data3 = []
for row in rows:
    data3.append(row._asdict())
df3 = pd.DataFrame(data3)
df3
```



1 to 25 of 80 entries

Filter



index	country	total_revenue
0	Malaysia	434357.3125
1	Israel	223442.046875
2	Serbia	802989.4375
3	Djibouti	61415.359375
4	Egypt	130261.7578125
5	Nicaragua	5944506.0
6	Romania	1726589.375
7	Sao Tome and Principe	301612.8125
8	Liberia	337990.71875
9	Vanuatu	1222089.25
10	Morocco	503890.09375
11	Lebanon	3699975.25
12	Madagascar	339860.8125
13	Mali	2884921.5
14	Togo	2101183.25
15	Sri Lanka	12866.0703125
16	Canada	3263260.75
17	Vietnam	652532.3125
18	Mauritius	603225.625
19	Tanzania	3555764.5
20	Tunisia	1071140.0
21	Brunei	868465.375
22	Seychelles	28327.650390625
23	Ghana	1015025.875
24	Turkmenistan	3223939.5

Show 25 per page

1

2

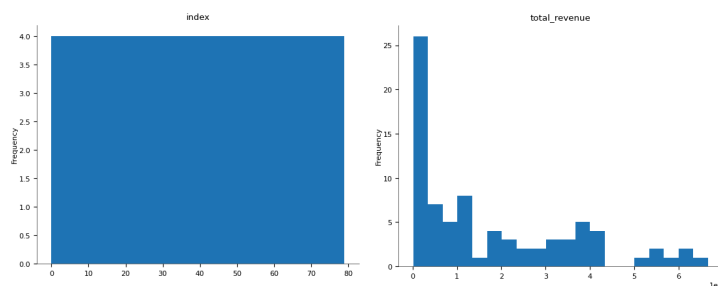
3

4

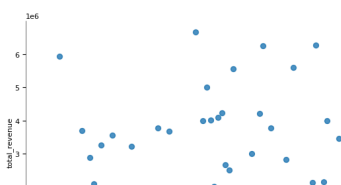


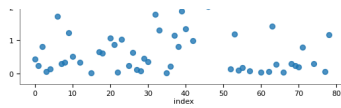
Like what you see? Visit the [data table notebook](#) to learn more about interactive tables.

Distributions

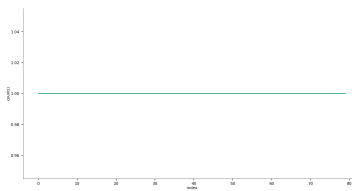
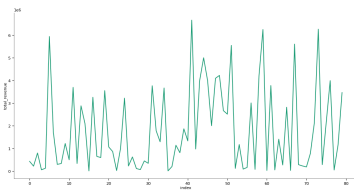


2-d distributions

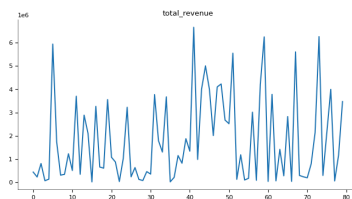
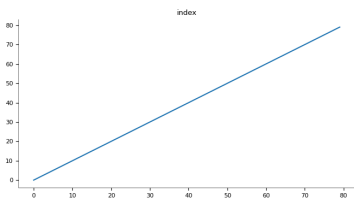




Time series



Values



Next steps:

[Generate code with df3](#)

[View recommended plots](#)

[New interactive sheet](#)

✓ 2. creating table which shows revenue by country

```
rows = session.execute("SELECT order_date, total_revenue, total_profit FROM silver_sales_tat")
data = [row._asdict() for row in rows]
df = pd.DataFrame(data)
```

```
# Convert 'order_date' to pandas datetime type
df["order_date"] = pd.to_datetime(df["order_date"], format="%Y-%m-%d")
```

```
# Now you can extract the year and month
df["year"] = df["order_date"].dt.year
df["month"] = df["order_date"].dt.month
```

```
# View the updated dataframe
df.head()
```

```
monthly_trends_df = (
    df.groupby(["year", "month"], as_index=False)[["total_revenue", "total_profit"]]
    .sum()
)
monthly_trends_df["year"] = monthly_trends_df["year"].astype(int)
```

```
session.execute("""
CREATE TABLE IF NOT EXISTS gold_monthly_trends (
    year INT,
    month INT,
    total_revenue FLOAT,
```

```
total_profit FLOAT,  
PRIMARY KEY (year, month)  
);  
""")  
  
# Insert data into the gold_monthly_trends table  
for _, row in monthly_trends_df.iterrows():  
    session.execute("""  
        INSERT INTO gold_monthly_trends (year, month, total_revenue, total_profit)  
        VALUES (%s, %s, %s, %s);  
        """, (int(row["year"]), int(row["month"]), float(row["total_revenue"]), float(row["total
```

```
rows = session.execute("SELECT * FROM gold_monthly_trends;")  
data4 = []  
for row in rows:  
    data4.append(row._asdict())  
df4 = pd.DataFrame(data4)  
df4
```



1 to 25 of 55 entries

Filter



index	year	month	total_profit	total_revenue
0	2014	2	562533.125	4081224.75
1	2014	3	1592127.625	4003440.5
2	2014	6	514581.84375	1205442.0
3	2014	7	2102895.5	9280963.0
4	2014	10	23133.58984375	89558.671875
5	2014	11	143351.640625	434357.3125
6	2010	4	1032559.3125	2585495.25
7	2010	6	11423.400390625	44224.19921875
8	2010	8	715456.4375	2884921.5
9	2010	10	236007.3125	628499.375
10	2010	11	1380006.25	3470056.5
11	2010	12	959274.25	4159723.5
12	2012	2	71738.4609375	217368.453125
13	2012	6	1311052.375	6552552.0
14	2012	7	1186156.0	3100162.0
15	2012	9	57264.84375	188951.875
16	2012	10	1178729.375	3256076.75
17	2012	11	405388.8125	603225.625
18	2017	3	614764.0	4283459.5
19	2017	4	297783.1875	2196359.25
20	2017	5	41273.28125	61415.359375
21	2015	2	1300347.25	4493491.0
22	2015	4	1071061.875	2536801.0
23	2015	5	201069.0625	605694.0
24	2015	6	215455.625	626742.8125

Show 25 per page

1

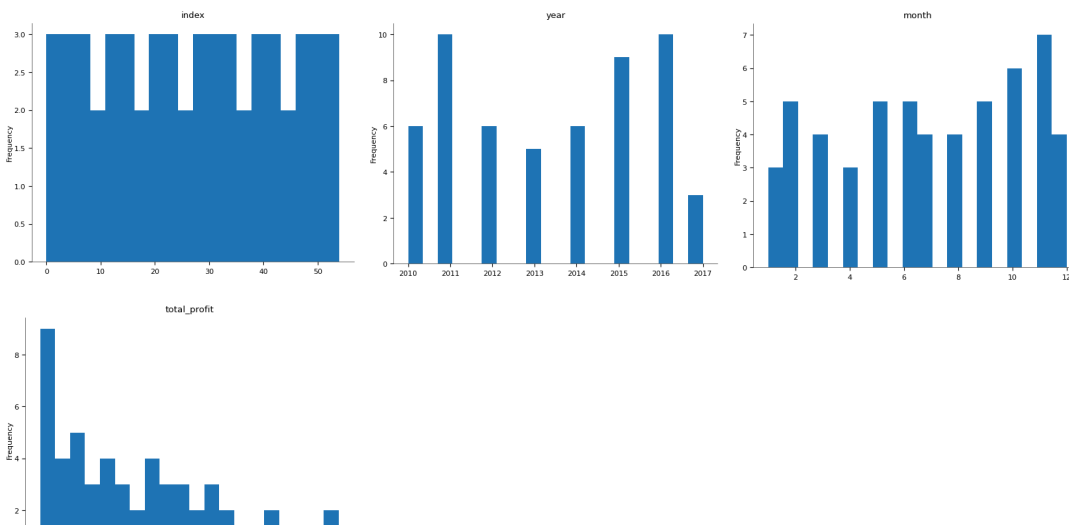
2

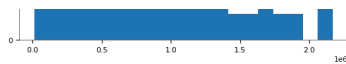
3



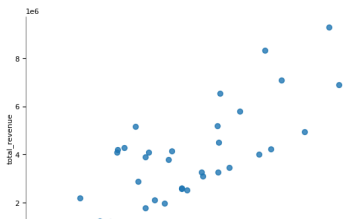
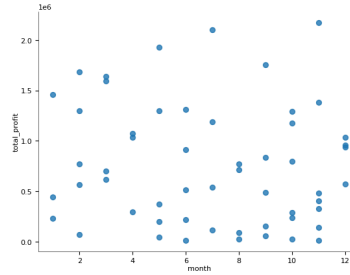
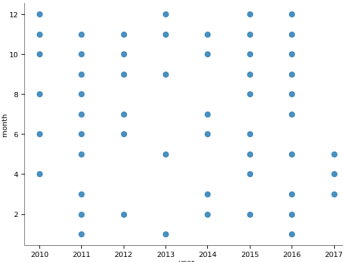
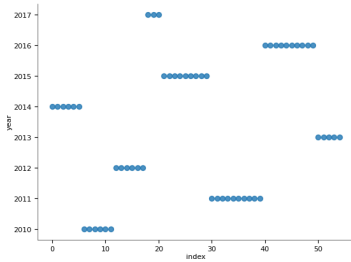
Like what you see? Visit the [data table notebook](#) to learn more about interactive tables.

Distributions





2-d distributions



Next steps:

[Generate code with df4](#)



[View recommended plots](#)

[New interactive sheet](#)

✓ 3. Calculating Avg profit by item_type

```
# Fetch all data (or a subset) from Cassandra
query = """
    SELECT item_type, total_revenue, total_profit
    FROM silver_sales_table;
"""

rows = session.execute(query)
data = [row._asdict() for row in rows]
df = pd.DataFrame(data)


# Group by 'item_type' in Python using pandas
item_performance_df = df.groupby('item_type')[['total_revenue', 'total_profit']].sum().reset

session.execute("""CREATE TABLE IF NOT EXISTS gold_item_performance (
    item_type TEXT,
    total_revenue FLOAT,
    total_profit FLOAT,
    PRIMARY KEY (item_type)
);""")

# Insert the grouped data into the gold_item_performance table
for _, row in item_performance_df.iterrows():
    session.execute("""
        INSERT INTO gold_item_performance (item_type, total_revenue, total_profit)
        VALUES (%s, %s, %s);
        """, (row["item_type"], float(row["total_revenue"]), float(row["total_profit"])))
```

Double-click (or enter) to edit

```
rows4 = session.execute("SELECT * FROM gold_item_performance;")
data5 = []
for row in rows4:
    data5.append(row._asdict())
df5 = pd.DataFrame(data5)
df5
```

1 to 12 of 12 entries Filter  

index	item_type	total_profit	total_revenue
-------	-----------	--------------	---------------