ASSIGNMENT

~Data and Cloud Analytics internship~

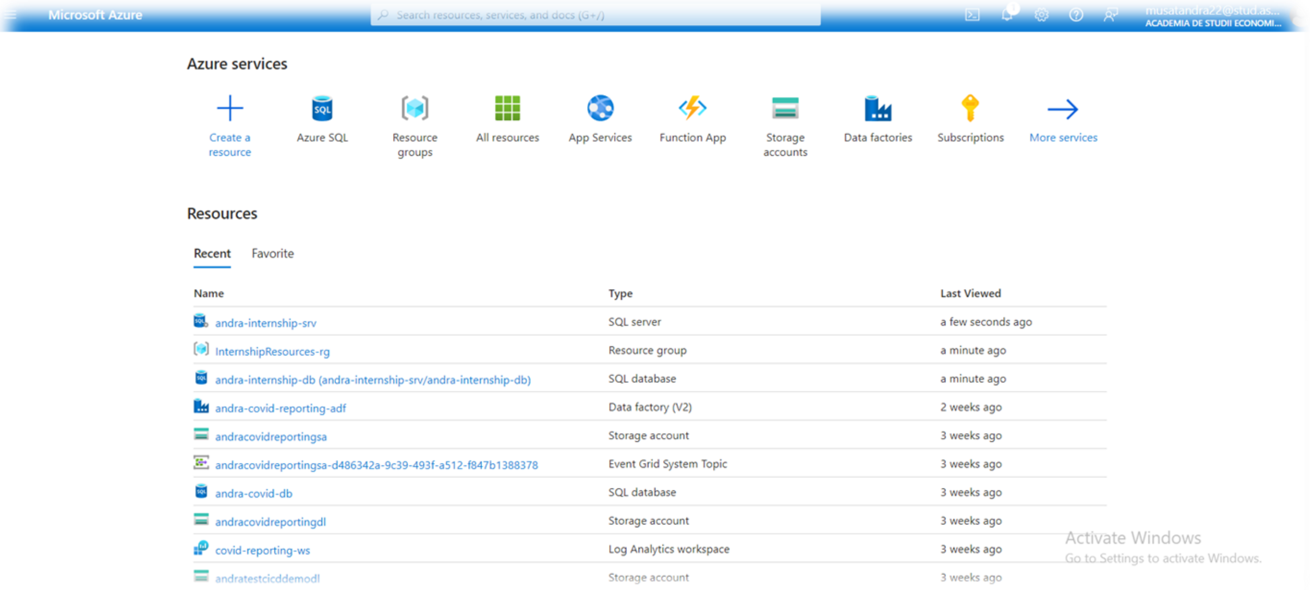
MUȘAT ANDRA

CUPRINS

A) Provision an Azure SQL Database.....3

B) In debt analysis of a dataset.....6

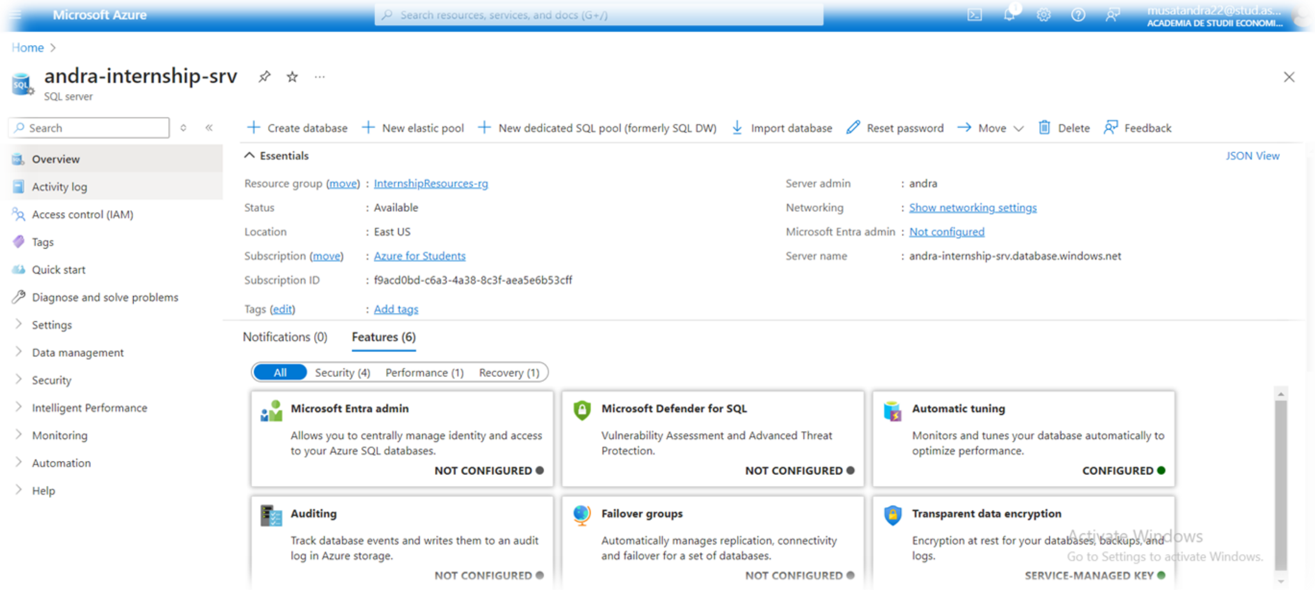
A) Provision an Azure SQL Database


Microsoft Azure
Search resources, services, and docs (G+/)
musatandra22@stud.as...
ACADEMIA DE STUDII ECONOMI...

Azure services

Resources

Name	Type	Last Viewed
andra-internship-srv	SQL server	a few seconds ago
InternshipResources-rg	Resource group	a minute ago
andra-internship-db (andra-internship-srv/andra-internship-db)	SQL database	a minute ago
andra-covid-reporting-adf	Data factory (V2)	2 weeks ago
andracovidreportingsa	Storage account	3 weeks ago
andracovidreportingsa-d486342a-9c39-493f-a512-f847b1388378	Event Grid System Topic	3 weeks ago
andra-covid-db	SQL database	3 weeks ago
andracovidreportingdl	Storage account	3 weeks ago
covid-reporting-ws	Log Analytics workspace	3 weeks ago
andratestcicddemodl	Storage account	3 weeks ago


Microsoft Azure
Search resources, services, and docs (G+/)
musatandra22@stud.as...
ACADEMIA DE STUDII ECONOMI...
Home >

andra-internship-srv

SQL server

Essentials

Resource group (move) : InternshipResources-rg

Status : Available

Location : East US

Subscription (move) : Azure for Students

Subscription ID : f9acd0bd-c6a3-4a38-8c3f-aea5e6b53cff

Tags (edit) : Add tags

Server admin : andra

Networking : Show networking settings

Microsoft Entra admin : Not configured

Server name : andra-internship-srv.database.windows.net

Microsoft Entra admin
Allows you to centrally manage identity and access to your Azure SQL databases.
NOT CONFIGURED ●

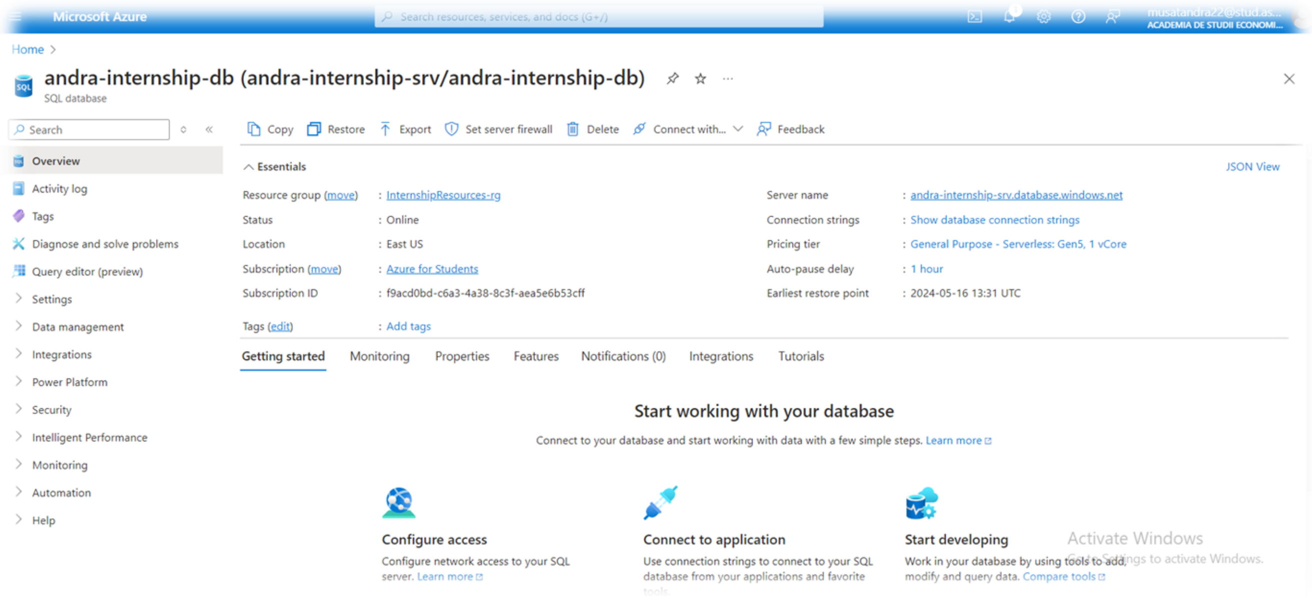
Microsoft Defender for SQL
Vulnerability Assessment and Advanced Threat Protection.
NOT CONFIGURED ●

Automatic tuning
Monitors and tunes your database automatically to optimize performance.
CONFIGURED ●

Auditing
Track database events and writes them to an audit log in Azure storage.
NOT CONFIGURED ●

Failover groups
Automatically manages replication, connectivity and failover for a set of databases.
NOT CONFIGURED ●

Transparent data encryption
Encryption at rest for your databases, backups, and logs.
SERVICE-MANAGED KEY ●



andra-internship-db (andra-internship-srv/andra-internship-db)

SQL database

Essentials

Resource group (move) : InternshipResources-rg
Status : Online
Location : East US
Subscription (move) : Azure for Students
Subscription ID : f9acd0bd-c6a3-4a38-8c3f-aea5e6b53cff
Tags (edit) : Add tags

Server name : andra-internship-srv.database.windows.net
Connection strings : Show database connection strings
Pricing tier : General Purpose - Serverless: Gen5, 1 vCore
Auto-pause delay : 1 hour
Earliest restore point : 2024-05-16 13:31 UTC

Start working with your database

Connect to your database and start working with data with a few simple steps. Learn more

Configure access

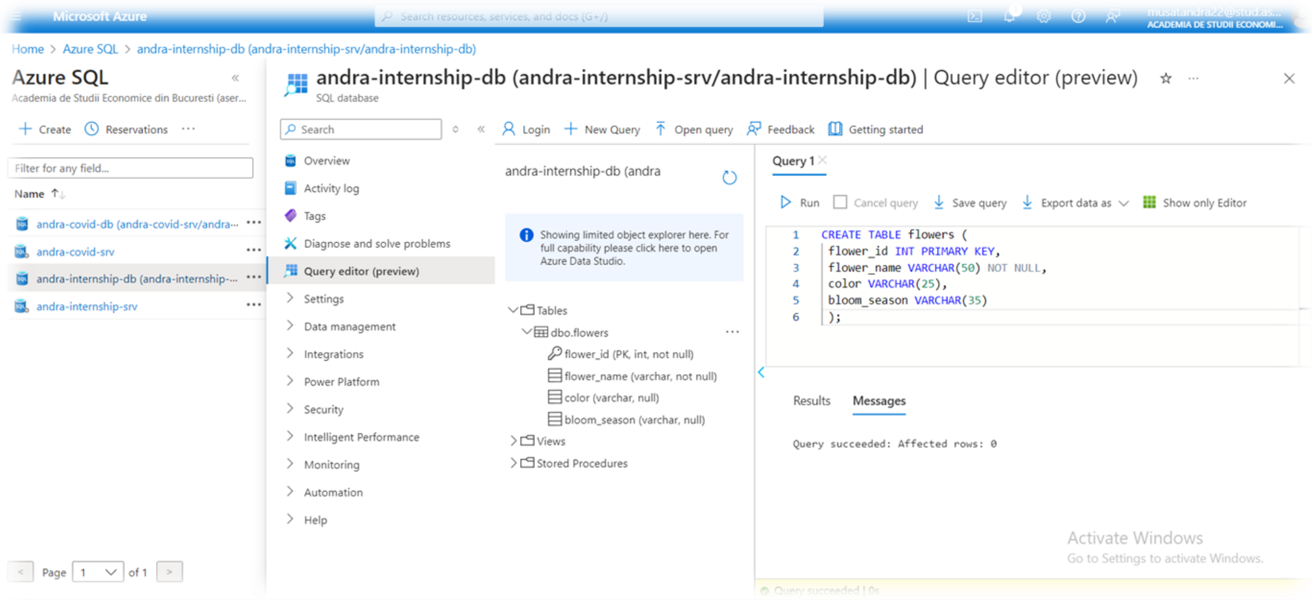
Configure network access to your SQL server. Learn more

Connect to application

Use connection strings to connect to your SQL database from your applications and favorite tools.

Start developing

Work in your database by using tools to add, modify and query data. Compare tools



Azure SQL

Academia de Studii Economice din Bucuresti (aser...

andra-covid-db (andra-covid-srv/andra-...
andra-covid-srv
andra-internship-db (andra-internship-...
andra-internship-srv

andra-internship-db (andra-internship-srv/andra-internship-db) | Query editor (preview)

SQL database

andra-internship-db (andra

Showing limited object explorer here. For full capability please click here to open Azure Data Studio.

- Tables
  - dbo.flowers
    - flower_id (PK, int, not null)
    - flower_name (varchar, not null)
    - color (varchar, null)
    - bloom_season (varchar, null)
- Views
- Stored Procedures

```
CREATE TABLE flowers (
 flower_id INT PRIMARY KEY,
 flower_name VARCHAR(50) NOT NULL,
 color VARCHAR(25),
 bloom_season VARCHAR(35)
 );
```

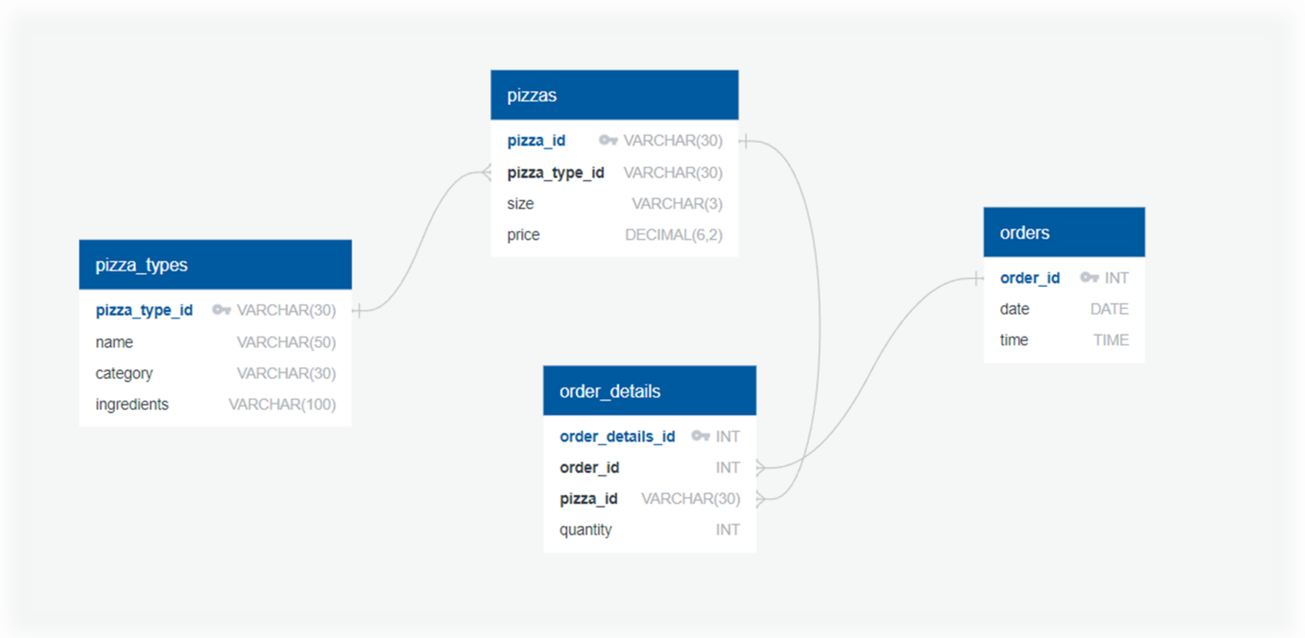
```
Query succeeded: Affected rows: 0
```

As a student account holder, I took the following steps:

- Before provisioning any resources, I created a new resource group to organize my resources logically. I named the resource group "InternshipResources-rg" and selected the East US location;
- Provisioning a single SQL database: From the search bar in Azure, I looked for Azure SQL, clicked on SQL databases, and left the Resource type set to Single database, then selected Create. I selected my student subscription and proceeded to give my database a name, “andra_internship-db”. For Server, I selected Create new and filled out the New server form with the following values: Server name: “andra-internship-srv”, Location: East US, Authentication method: Use SQL authentication, created a Server admin login and password below. After that, I selected the Workload Environment to be Development and clicked on Configure Database. I selected the Service Tier to be General Purpose and Compute tier to be Serverless, and applied the changes, ensuring the SQL elastic pool is set to No and Backup storage redundancy option is locally redundant storage. Next, I clicked on Networking and set the Connectivity method to Public endpoint. For Firewall rules, I set Add current client IP address to Yes, and left Allow Azure services and resources to access this server set to No. Then, I clicked on security to proceed and kept Enable Microsoft Defender for SQL set to No. Next, I clicked on additional settings and kept Use existing data set to None to create an empty database. Finally, I clicked on Review and Create and then on Create.
- Waited for Deployment: Azure deployed my SQL Server instance and database. This process took a few minutes. I monitored the deployment progress from the notification area in the upper right area of the portal;
- Verified Deployment: Once the deployment was complete, I visited the Azure SQL area again to ensure that both the SQL Server instance and the database were provisioned successfully and were in a running state;
- As an additional task, I went to my newly created database, navigated to the Query editor, and created a simple table (flowers table) using an SQL query;
- At the end, I took screenshots.

B) In debt analysis of a dataset

1. I used QuickDBD



pizzas	
pizza_id	VARCHAR(30)
pizza_type_id	VARCHAR(30)
size	VARCHAR(3)
price	DECIMAL(6,2)

pizza_types	
pizza_type_id	VARCHAR(30)
name	VARCHAR(50)
category	VARCHAR(30)
ingredients	VARCHAR(100)

orders	
order_id	INT
date	DATE
time	TIME

order_details	
order_details_id	INT
order_id	INT
pizza_id	VARCHAR(30)
quantity	INT

2. I expect the type of relationship between tables to be:

- The relationship between "pizzas" and "pizza_types" is a one to many relationship. Each pizza belongs to only one pizza type, but each pizza type can have multiple pizzas associated with it.
- The relationship between "pizzas" and "order_details" is a one to many relationship. Each pizza can have multiple order details associated with it, but each order detail is related to only one pizza.
- The relationship between "orders" and "order_details" is a one to many relationship. Each order can have multiple order details associated with it, but each order detail is related to only one order.

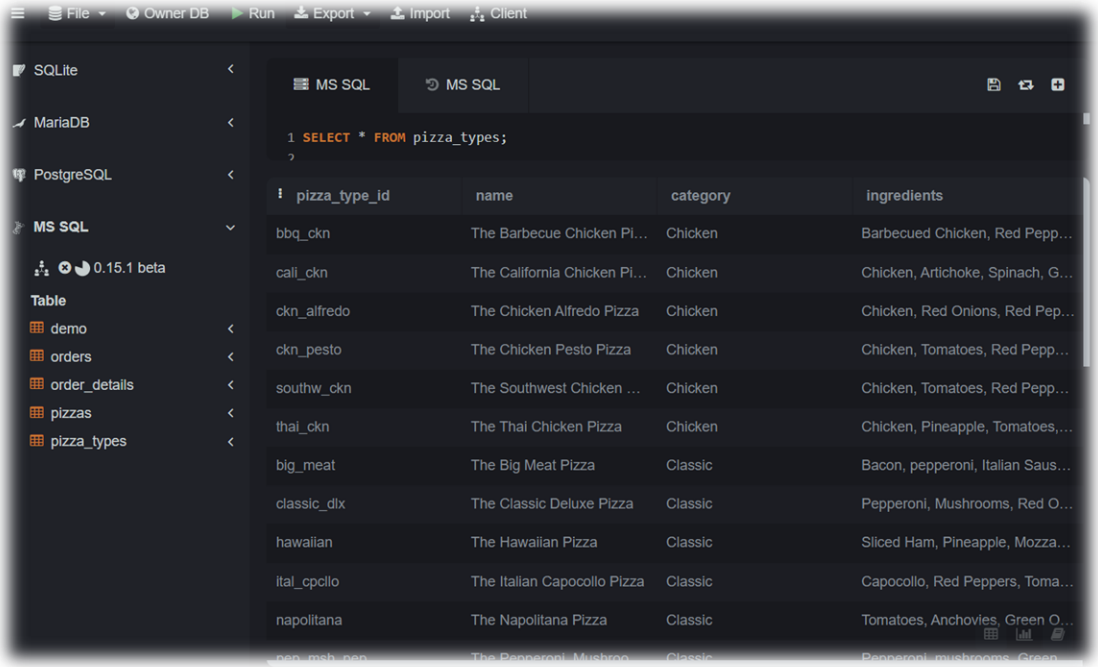
3. For this requirement:

- I split the sheets with tables from the given Excel file and saved them in CSV format. Then, using MS SQL from the online SQL server, I imported the CSV files with tables, and for each column in the tables, I adjusted the data type and added the necessary constraints.
- The code:

```
SELECT * FROM pizza_types;

ALTER TABLE pizza_types
ALTER COLUMN pizza_type_id VARCHAR(30) NOT NULL;
ALTER TABLE pizza_types
ALTER COLUMN name VARCHAR(50);
ALTER TABLE pizza_types
ALTER COLUMN category VARCHAR(30);
ALTER TABLE pizza_types
ALTER COLUMN ingredients VARCHAR(100);
ALTER TABLE pizza_types
ADD CONSTRAINT pk_pizza_type_id PRIMARY KEY (pizza_type_id);

SELECT * FROM pizzas;

ALTER TABLE pizzas
ALTER COLUMN pizza_id VARCHAR(30) NOT NULL;
ALTER TABLE pizzas
ALTER COLUMN pizza_type_id VARCHAR(30);
ALTER TABLE pizzas
ALTER COLUMN size VARCHAR(3);
ALTER TABLE pizzas
ALTER COLUMN price DECIMAL(6,2);
ALTER TABLE pizzas
ADD CONSTRAINT pk_pizza_id PRIMARY KEY (pizza_id);
ALTER TABLE pizzas
add CONSTRAINT fk_pizza_type_id FOREIGN KEY(pizza_type_id) REFERENCES
pizza_types(pizza_type_id);

SELECT TOP 100 * FROM orders;

ALTER TABLE orders
ALTER COLUMN order_id INT NOT NULL;
ALTER TABLE orders
ALTER COLUMN date DATE;
ALTER TABLE orders
ALTER COLUMN time TIME;
alter TABLE orders
```

```
ADD CONSTRAINT pk_order_id PRIMARY KEY (order_id);


SELECT TOP 100 * FROM order_details;

ALTER TABLE order_details
ALTER COLUMN order_details_id INT NOT NULL;
ALTER TABLE order_details
ALTER COLUMN order_id INT NOT NULL;
ALTER TABLE order_details
ALTER COLUMN pizza_id VARCHAR(30);
ALTER TABLE order_details
ALTER COLUMN quantity INT;
ALTER TABLE order_details
ADD CONSTRAINT pk_order_details_id PRIMARY KEY (order_details_id);
ALTER TABLE order_details
ADD CONSTRAINT fk_order_id FOREIGN KEY(order_id) REFERENCES orders(order_id);
ALTER TABLE order_details
ADD CONSTRAINT fk_pizza_id FOREIGN KEY(pizza_id) REFERENCES pizzas(pizza_id);
```

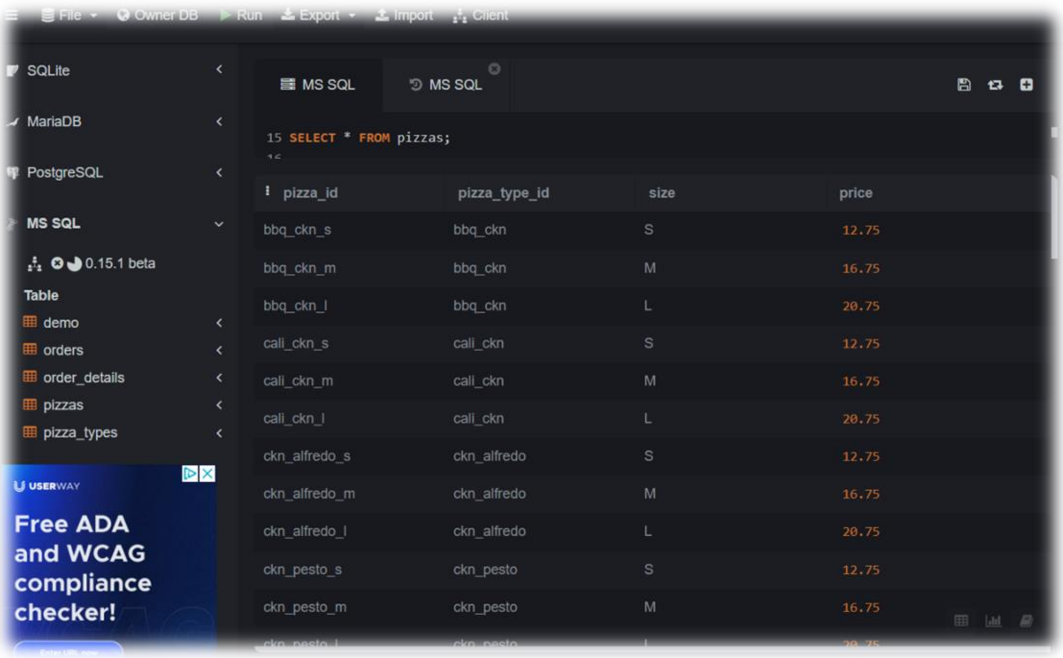
- The results:



```
SELECT * FROM pizza_types;
```

pizza_type_id	name	category	ingredients
bbq_ckn	The Barbecue Chicken Pi…	Chicken	Barbecued Chicken, Red Pepp…
cali_ckn	The California Chicken Pi…	Chicken	Chicken, Artichoke, Spinach, G…
ckn_alfredo	The Chicken Alfredo Pizza	Chicken	Chicken, Red Onions, Red Pep…
ckn_pesto	The Chicken Pesto Pizza	Chicken	Chicken, Tomatoes, Red Pepp…
southw_ckn	The Southwest Chicken …	Chicken	Chicken, Tomatoes, Red Pepp…
thai_ckn	The Thai Chicken Pizza	Chicken	Chicken, Pineapple, Tomatoes,…
big_meat	The Big Meat Pizza	Classic	Bacon, pepperoni, Italian Saus…
classic_dlx	The Classic Deluxe Pizza	Classic	Pepperoni, Mushrooms, Red O…
hawaiian	The Hawaiian Pizza	Classic	Sliced Ham, Pineapple, Mozza…
ital_cpcllo	The Italian Capocollo Pizza	Classic	Capocollo, Red Peppers, Toma…
napolitana	The Napolitana Pizza	Classic	Tomatoes, Anchovies, Green O…
pep_msh_pep	The Pepperoni, Mushroo…	Classic	Pepperoni, mushrooms, Green…

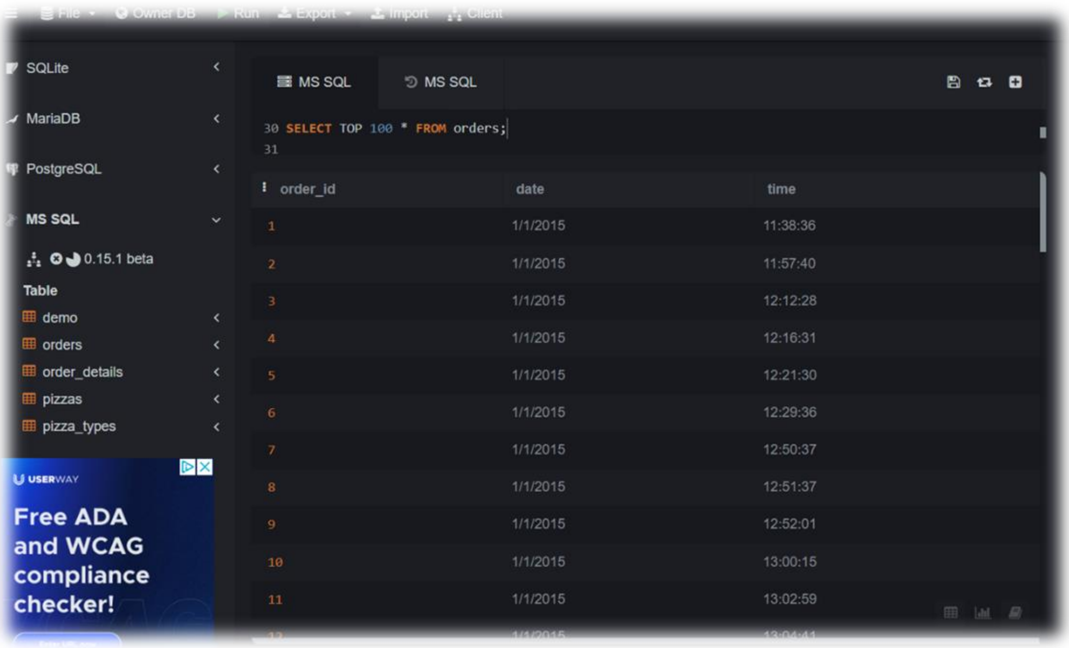
pizza_types table



```sql
SELECT * FROM pizzas;
```

pizza_id	pizza_type_id	size	price
bbq_ckn_s	bbq_ckn	S	12.75
bbq_ckn_m	bbq_ckn	M	16.75
bbq_ckn_l	bbq_ckn	L	20.75
cali_ckn_s	cali_ckn	S	12.75
cali_ckn_m	cali_ckn	M	16.75
cali_ckn_l	cali_ckn	L	20.75
ckn_alfredo_s	ckn_alfredo	S	12.75
ckn_alfredo_m	ckn_alfredo	M	16.75
ckn_alfredo_l	ckn_alfredo	L	20.75
ckn_pesto_s	ckn_pesto	S	12.75
ckn_pesto_m	ckn_pesto	M	16.75
ckn_pesto_l	ckn_pesto	L	20.75

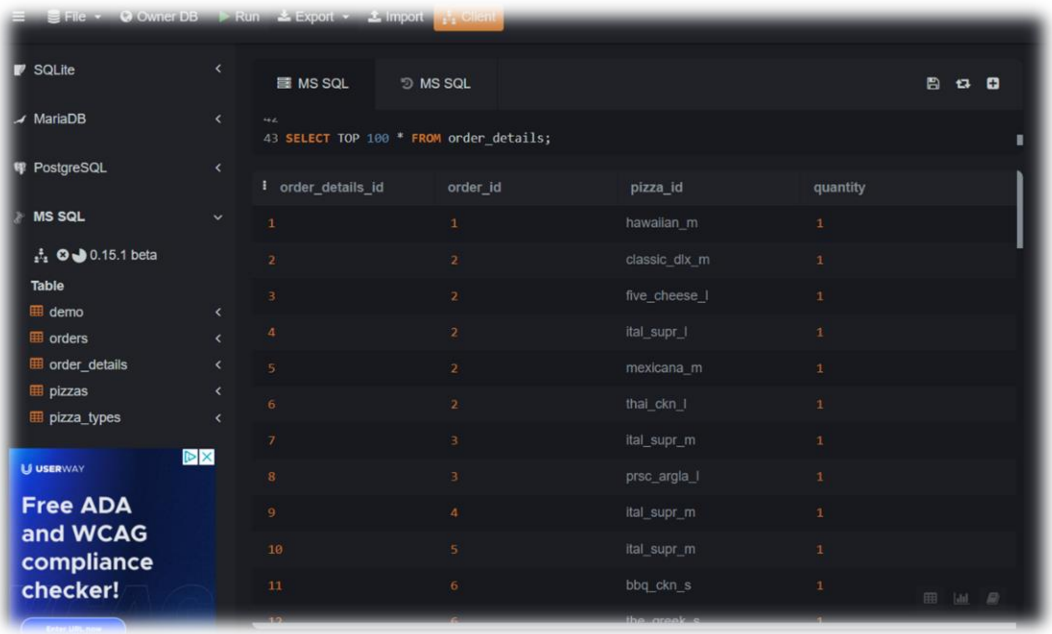
pizzas table



```sql
SELECT TOP 100 * FROM orders;
```

order_id	date	time
1	1/1/2015	11:38:36
2	1/1/2015	11:57:40
3	1/1/2015	12:12:28
4	1/1/2015	12:16:31
5	1/1/2015	12:21:30
6	1/1/2015	12:29:36
7	1/1/2015	12:50:37
8	1/1/2015	12:51:37
9	1/1/2015	12:52:01
10	1/1/2015	13:00:15
11	1/1/2015	13:02:59
12	1/1/2015	13:04:41

orders table



```sql
SELECT TOP 100 * FROM order_details;
```

order_details_id	order_id	pizza_id	quantity
1	1	hawaiian_m	1
2	2	classic_dlx_m	1
3	2	five_cheese_l	1
4	2	ital_supr_l	1
5	2	mexicana_m	1
6	2	thai_ckn_l	1
7	3	ital_supr_m	1
8	3	prsc_argla_l	1
9	4	ital_supr_m	1
10	5	ital_supr_m	1
11	6	bbq_ckn_s	1
12	6	the_greek_s	1

order_details table

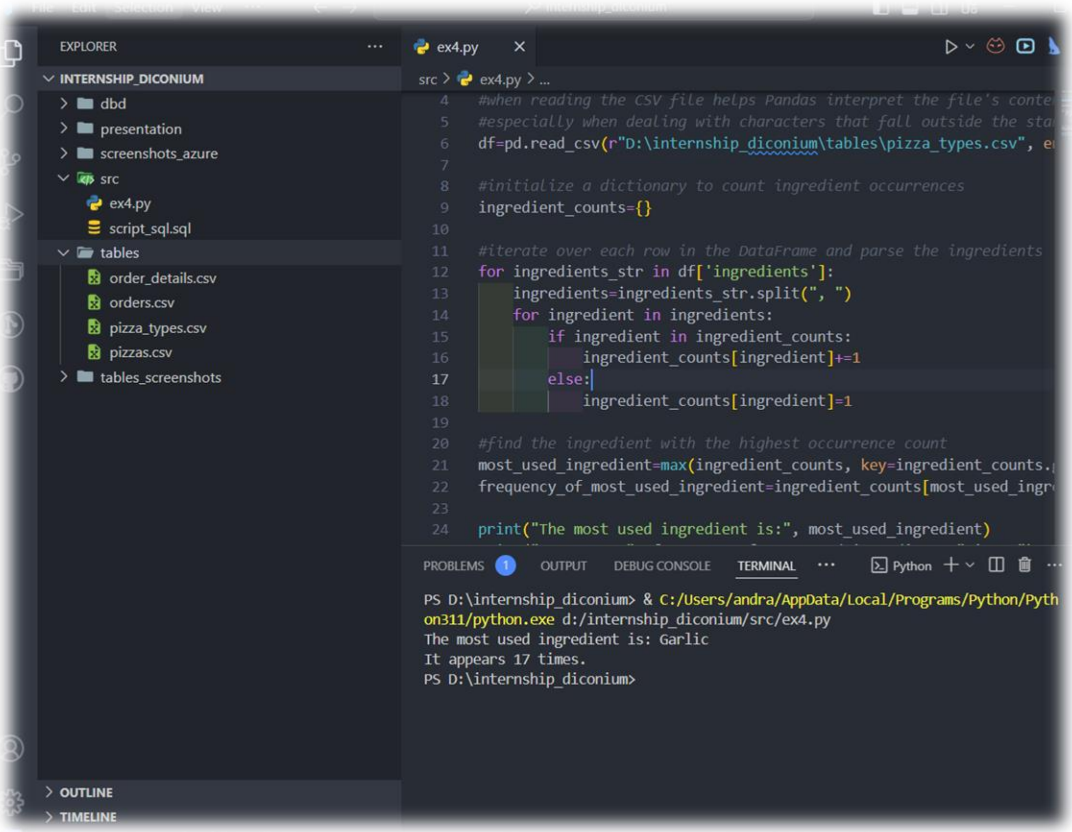
4. I used python

- The code:

```python
import pandas as pd

#load the CSV file specifying the encoding(specifying the encoding parameter
#when reading the CSV file helps Pandas interpret the file's contents correctly,
#especially when dealing with characters that fall outside the standard ASCII range.)
df=pd.read_csv(r"D:\internship_diconium\tables\pizza_types.csv", encoding='latin1')

#initialize a dictionary to count ingredient occurrences
ingredient_counts={}

#iterate over each row in the DataFrame and parse the ingredients
for ingredients_str in df['ingredients']:
    ingredients=ingredients_str.split(", ")
    for ingredient in ingredients:
        if ingredient in ingredient_counts:
            ingredient_counts[ingredient]+=1
        else:
            ingredient_counts[ingredient]=1
```

```
#find the ingredient with the highest occurrence count
most_used_ingredient=max(ingredient_counts, key=ingredient_counts.get)
frequency_of_most_used_ingredient=ingredient_counts[most_used_ingredient]

print("The most used ingredient is:", most_used_ingredient)
print("It appears", frequency_of_most_used_ingredient, "times.")
```

- The result:



```
#when reading the CSV file helps Pandas interpret the file's conte
#especially when dealing with characters that fall outside the sta
df=pd.read_csv(r"D:\internship_diconium\tables\pizza_types.csv", e

#initialize a dictionary to count ingredient occurrences
ingredient_counts={}

#iterate over each row in the DataFrame and parse the ingredients
for ingredients_str in df['ingredients']:
    ingredients=ingredients_str.split(", ")
    for ingredient in ingredients:
        if ingredient in ingredient_counts:
            ingredient_counts[ingredient]+=1
        else:
            ingredient_counts[ingredient]=1

#find the ingredient with the highest occurrence count
most_used_ingredient=max(ingredient_counts, key=ingredient_counts.
frequency_of_most_used_ingredient=ingredient_counts[most_used_ingr

print("The most used ingredient is:", most_used_ingredient)
```

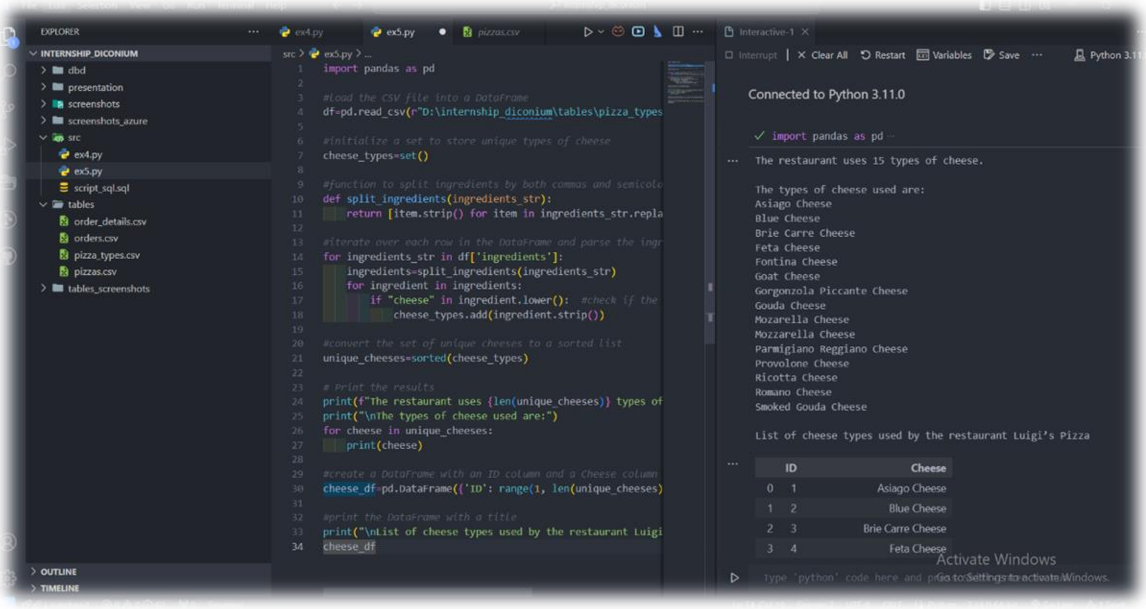
```
PS D:\internship_diconium> & C:/Users/andra/AppData/Local/Programs/Python/Pyth
on311/python.exe d:/internship_diconium/src/ex4.py
The most used ingredient is: Garlic
It appears 17 times.
PS D:\internship_diconium>
```

5. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd

#load the CSV file into a DataFrame
df=pd.read_csv(r"D:\internship_diconium\tables\pizza_types.csv", encoding='latin1')

#initialize a set to store unique types of cheese
cheese_types=set()

#function to split ingredients by both commas and semicolons
def split_ingredients(ingredients_str):
    return [item.strip() for item in ingredients_str.replace(';', ',').split(',')]

#iterate over each row in the DataFrame and parse the ingredients
for ingredients_str in df['ingredients']:
    ingredients=split_ingredients(ingredients_str)
    for ingredient in ingredients:
        if "cheese" in ingredient.lower():  #check if the word 'cheese' is in the ingredient
            cheese_types.add(ingredient.strip())

#convert the set of unique cheeses to a sorted list
unique_cheeses=sorted(cheese_types)

# Print the results
print(f"The restaurant uses {len(unique_cheeses)} types of cheese.")
print("The types of cheese used are:")
for cheese in unique_cheeses:
    print(cheese)

#create a DataFrame with an ID column and a Cheese column
cheese_df=pd.DataFrame({'ID': range(1, len(unique_cheeses)+1), 'Cheese': unique_cheeses})

#print the DataFrame with a title
print("List of Cheese Types Used by the Restaurant")
cheese_df
```

- The result:



```python
import pandas as pd

#load the CSV file into a DataFrame
df=pd.read_csv(r"D:\internship_diconium\tables\pizza_types

#initialize a set to store unique types of cheese
cheese_types=set()

#function to split ingredients by both commas and semicolo
def split_ingredients(ingredients_str):
    return [item.strip() for item in ingredients_str.repla

#iterate over each row in the DataFrame and parse the ingr
for ingredients_str in df['ingredients']:
    ingredients=split_ingredients(ingredients_str)
    for ingredient in ingredients:
        if "cheese" in ingredient.lower():  #check if the
            cheese_types.add(ingredient.strip())

#convert the set of unique cheeses to a sorted list
unique_cheeses=sorted(cheese_types)

# Print the results
print(f"The restaurant uses {len(unique_cheeses)} types of
print("\nThe types of cheese used are:")
for cheese in unique_cheeses:
    print(cheese)

#create a DataFrame with an ID column and a Cheese column
cheese_df=pd.DataFrame({'ID': range(1, len(unique_cheeses)

#print the DataFrame with a title
print("\nList of cheese types used by the restaurant Luigi
cheese_df
```

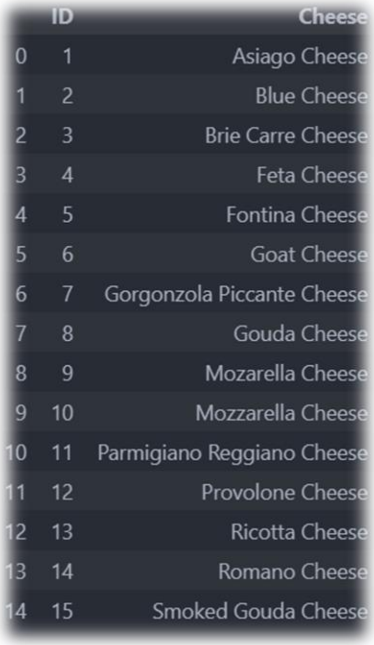
Connected to Python 3.11.0

```
✓ import pandas as pd ···
```

```
The restaurant uses 15 types of cheese.

The types of cheese used are:
Asiago Cheese
Blue Cheese
Brie Carre Cheese
Feta Cheese
Fontina Cheese
Goat Cheese
Gorgonzola Piccante Cheese
Gouda Cheese
Mozarella Cheese
Mozzarella Cheese
Parmigiano Reggiano Cheese
Provolone Cheese
Ricotta Cheese
Romano Cheese
Smoked Gouda Cheese

List of cheese types used by the restaurant Luigi’s Pizza
```

	ID	Cheese
0	1	Asiago Cheese
1	2	Blue Cheese
2	3	Brie Carre Cheese
3	4	Feta Cheese

```
Type 'python' code here and pr
```



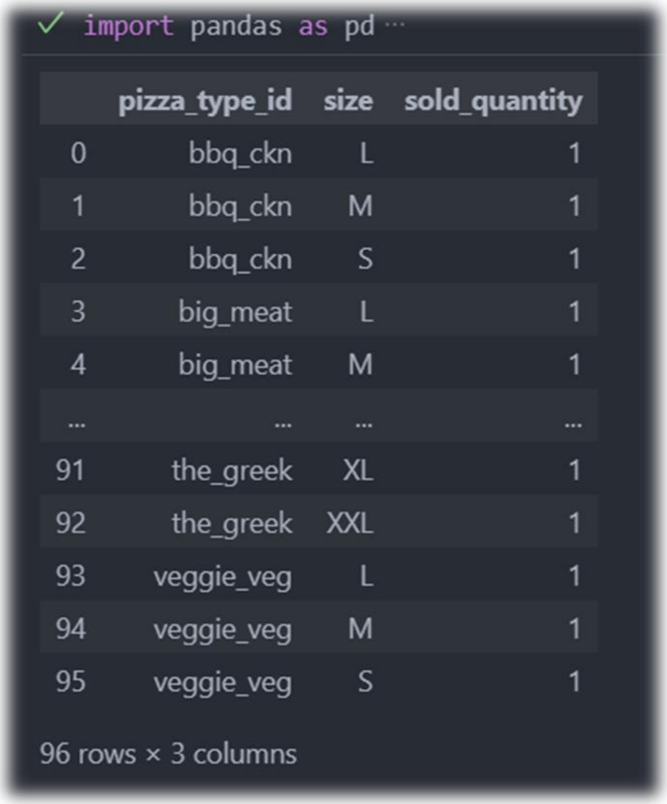
	ID	Cheese
0	1	Asiago Cheese
1	2	Blue Cheese
2	3	Brie Carre Cheese
3	4	Feta Cheese
4	5	Fontina Cheese
5	6	Goat Cheese
6	7	Gorgonzola Piccante Cheese
7	8	Gouda Cheese
8	9	Mozarella Cheese
9	10	Mozzarella Cheese
10	11	Parmigiano Reggiano Cheese
11	12	Provolone Cheese
12	13	Ricotta Cheese
13	14	Romano Cheese
14	15	Smoked Gouda Cheese

6. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd

#load the CSV file into a pandas DataFrame
pizzas_df=pd.read_csv(r"D:\internship_diconium\tables\pizzas.csv")

#group by pizza_type_id and size, then count the number of pizzas sold for each group
sales_summary=pizzas_df.groupby(['pizza_type_id', 'size']).size().reset_index(name='sold_quantity')

#display the sales summary DataFrame as a table
sales_summary
```

- The result:



```
✓ import pandas as pd ···
```

	pizza_type_id	size	sold_quantity
0	bbq_ckn	L	1
1	bbq_ckn	M	1
2	bbq_ckn	S	1
3	big_meat	L	1
4	big_meat	M	1
...	...	...	...
91	the_greek	XL	1
92	the_greek	XXL	1
93	veggie_veg	L	1
94	veggie_veg	M	1
95	veggie_veg	S	1

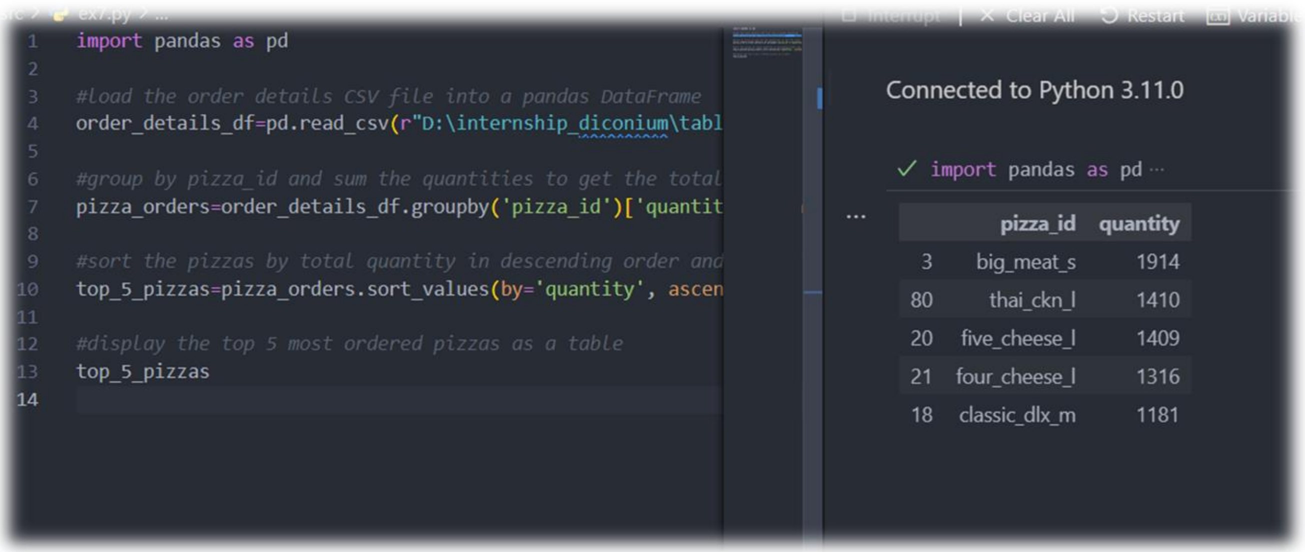
96 rows × 3 columns

7. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd

#load the order details CSV file into a pandas DataFrame
order_details_df=pd.read_csv(r"D:\internship_diconium\tables\order_details.csv")

#group by pizza_id and sum the quantities to get the total quantity of each pizza ordered
pizza_orders=order_details_df.groupby('pizza_id')['quantity'].sum().reset_index()

#sort the pizzas by total quantity in descending order and select the top 5
top_5_pizzas=pizza_orders.sort_values(by='quantity', ascending=False).head(5)

#display the top 5 most ordered pizzas as a table
top_5_pizzas
```

- The result:



```
import pandas as pd

#load the order details CSV file into a pandas DataFrame
order_details_df=pd.read_csv(r"D:\internship_diconium\tabl

#group by pizza_id and sum the quantities to get the total
pizza_orders=order_details_df.groupby('pizza_id')['quantit

#sort the pizzas by total quantity in descending order and
top_5_pizzas=pizza_orders.sort_values(by='quantity', ascen

#display the top 5 most ordered pizzas as a table
top_5_pizzas
```

Connected to Python 3.11.0

✓ import pandas as pd ...

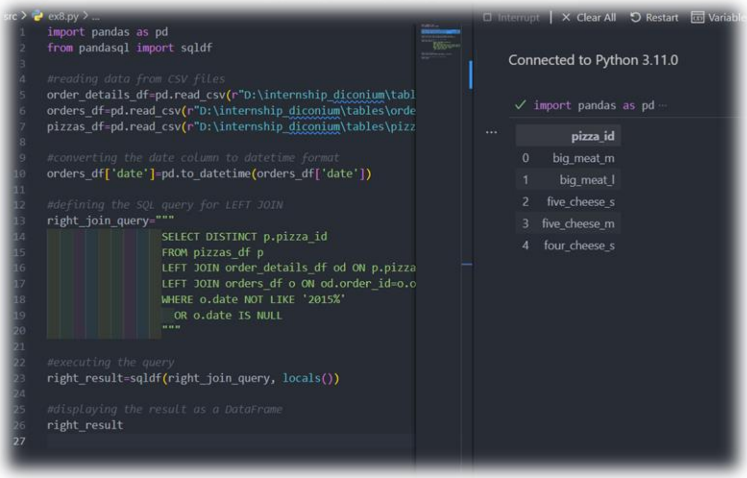
	pizza_id	quantity
3	big_meat_s	1914
80	thai_ckn_l	1410
20	five_cheese_l	1409
21	four_cheese_l	1316
18	classic_dlx_m	1181

8. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd
from pandasql import sqldf

#reading data from CSV files
order_details_df=pd.read_csv(r"D:\internship_diconium\tables\order_details.csv")
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orders.csv")
pizzas_df=pd.read_csv(r"D:\internship_diconium\tables\pizzas.csv")

#converting the date column to datetime format
orders_df['date']=pd.to_datetime(orders_df['date'])

#defining the SQL query for LEFT JOIN
right_join_query="""
            SELECT DISTINCT p.pizza_id
            FROM pizzas_df p
            LEFT JOIN order_details_df od ON p.pizza_id=od.pizza_id
            LEFT JOIN orders_df o ON od.order_id=o.order_id
            WHERE o.date NOT LIKE '2015%'
             OR o.date IS NULL
            """

#executing the query
right_result=sqldf(right_join_query, locals())

#displaying the result as a DataFrame
right_result
```

- The result:



```
import pandas as pd
from pandasql import sqldf

#reading data from CSV files
order_details_df=pd.read_csv(r"D:\internship_diconium\tabl
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orde
pizzas_df=pd.read_csv(r"D:\internship_diconium\tables\pizz

#converting the date column to datetime format
orders_df['date']=pd.to_datetime(orders_df['date'])

#defining the SQL query for LEFT JOIN
right_join_query="""
                  SELECT DISTINCT p.pizza_id
                  FROM pizzas_df p
                  LEFT JOIN order_details_df od ON p.pizza
                  LEFT JOIN orders_df o ON od.order_id=o.o
                  WHERE o.date NOT LIKE '2015%'
                    OR o.date IS NULL
                  """

#executing the query
right_result=sqldf(right_join_query, locals())

#displaying the result as a DataFrame
right_result
```

Connected to Python 3.11.0

✓ import pandas as pd ...

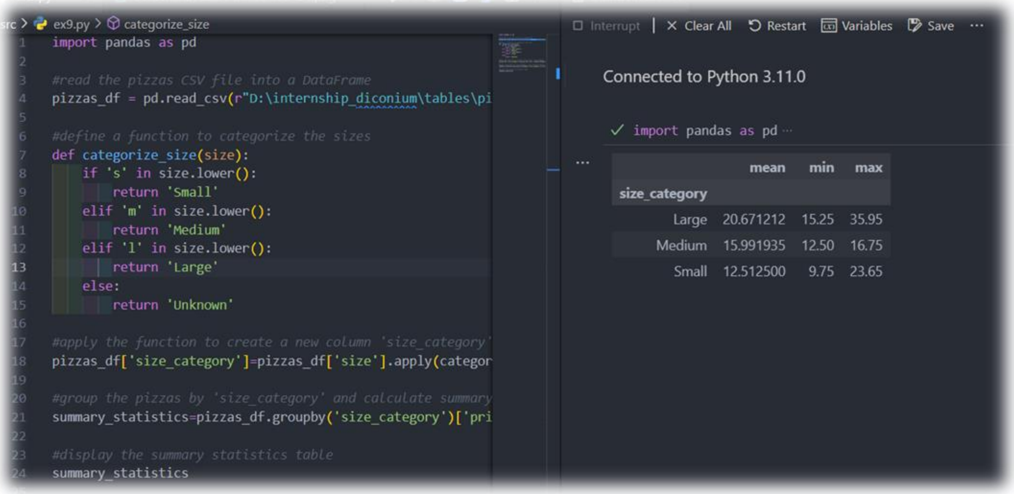
```
        pizza_id
0     big_meat_m
1     big_meat_l
2   five_cheese_s
3   five_cheese_m
4   four_cheese_s
```

9. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd

#read the pizzas CSV file into a DataFrame
pizzas_df = pd.read_csv(r"D:\internship_diconium\tables\pizzas.csv")

#define a function to categorize the sizes
def categorize_size(size):
    if 's' in size.lower():
        return 'Small'
    elif 'm' in size.lower():
        return 'Medium'
    elif 'l' in size.lower():
        return 'Large'
    else:
        return 'Unknown'

#apply the function to create a new column 'size_category'
pizzas_df['size_category']=pizzas_df['size'].apply(categorize_size)

#group the pizzas by 'size_category' and calculate summary statistics for 'price'
summary_statistics=pizzas_df.groupby('size_category')['price'].agg(['mean', 'min', 'max'])

#display the summary statistics table
summary_statistics
```

- The result:



src > ex9.py > categorize_size
□ Interrupt | ✕ Clear All ⟲ Restart Variables Save ···

```
import pandas as pd

#read the pizzas CSV file into a DataFrame
pizzas_df = pd.read_csv(r"D:\internship_diconium\tables\pi

#define a function to categorize the sizes
def categorize_size(size):
    if 's' in size.lower():
        return 'Small'
    elif 'm' in size.lower():
        return 'Medium'
    elif 'l' in size.lower():
        return 'Large'
    else:
        return 'Unknown'

#apply the function to create a new column 'size_category'
pizzas_df['size_category']=pizzas_df['size'].apply(categor

#group the pizzas by 'size_category' and calculate summary
summary_statistics=pizzas_df.groupby('size_category')['pri

#display the summary statistics table
summary_statistics
```

Connected to Python 3.11.0

✓ import pandas as pd ···

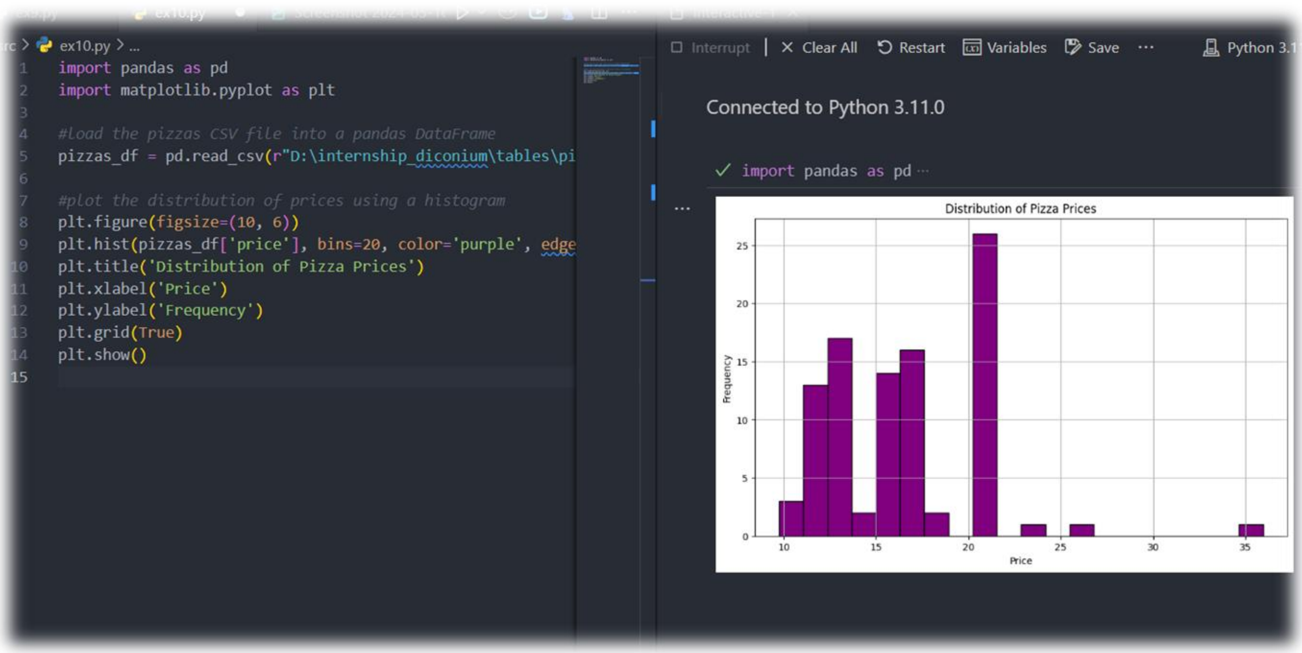
```
                    mean    min    max
size_category
Large          20.671212  15.25  35.95
Medium         15.991935  12.50  16.75
Small          12.512500   9.75  23.65
```

10. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd
import matplotlib.pyplot as plt

#load the pizzas CSV file into a pandas DataFrame
pizzas_df = pd.read_csv(r"D:\internship_diconium\tables\pizzas.csv")

#plot the distribution of prices using a histogram
plt.figure(figsize=(10, 6))
plt.hist(pizzas_df['price'], bins=20, color='purple', edgecolor='black')
plt.title('Distribution of Pizza Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

- The result:



```
import pandas as pd
import matplotlib.pyplot as plt

#load the pizzas CSV file into a pandas DataFrame
pizzas_df = pd.read_csv(r"D:\internship_diconium\tables\pi

#plot the distribution of prices using a histogram
plt.figure(figsize=(10, 6))
plt.hist(pizzas_df['price'], bins=20, color='purple', edge
plt.title('Distribution of Pizza Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

Connected to Python 3.11.0

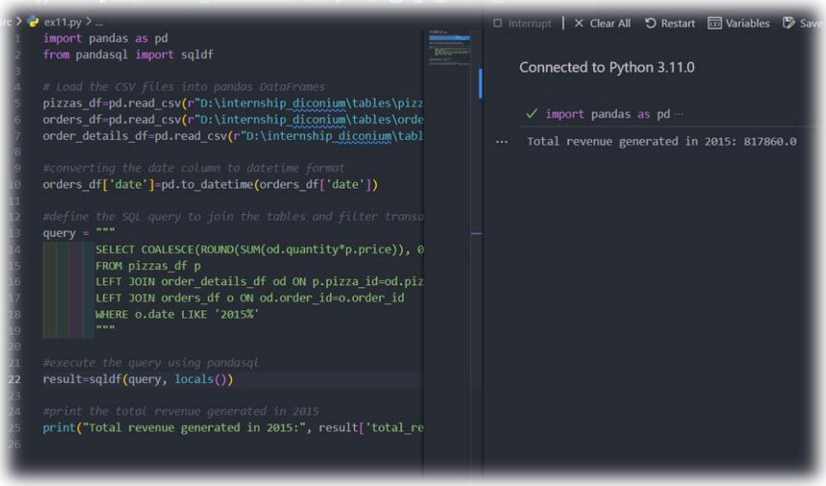
✓ import pandas as pd ···

11. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd
from pandasql import sqldf

# Load the CSV files into pandas DataFrames
pizzas_df=pd.read_csv(r"D:\internship_diconium\tables\pizzas.csv")
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orders.csv")
order_details_df=pd.read_csv(r"D:\internship_diconium\tables\order_details.csv")

#converting the date column to datetime format
orders_df['date']=pd.to_datetime(orders_df['date'])

#define the SQL query to join the tables and filter transactions in 2015
query = """
    SELECT COALESCE(ROUND(SUM(od.quantity*p.price)), 0) AS total_revenue_2015
    FROM pizzas_df p
    LEFT JOIN order_details_df od ON p.pizza_id=od.pizza_id
    LEFT JOIN orders_df o ON od.order_id=o.order_id
    WHERE o.date LIKE '2015%'
    """

#execute the query using pandasql
result=sqldf(query, locals())

#print the total revenue generated in 2015
print("Total revenue generated in 2015:", result['total_revenue_2015'].values[0])
```

- The result:



```
import pandas as pd
from pandasql import sqldf

# Load the CSV files into pandas DataFrames
pizzas_df=pd.read_csv(r"D:\internship_diconium\tables\pizz
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orde
order_details_df=pd.read_csv(r"D:\internship_diconium\tabl

#converting the date column to datetime format
orders_df['date']=pd.to_datetime(orders_df['date'])

#define the SQL query to join the tables and filter transa
query = """
        SELECT COALESCE(ROUND(SUM(od.quantity*p.price)), 0
        FROM pizzas_df p
        LEFT JOIN order_details_df od ON p.pizza_id=od.piz
        LEFT JOIN orders_df o ON od.order_id=o.order_id
        WHERE o.date LIKE '2015%'
        """

#execute the query using pandasql
result=sqldf(query, locals())

#print the total revenue generated in 2015
print("Total revenue generated in 2015:", result['total_re
```

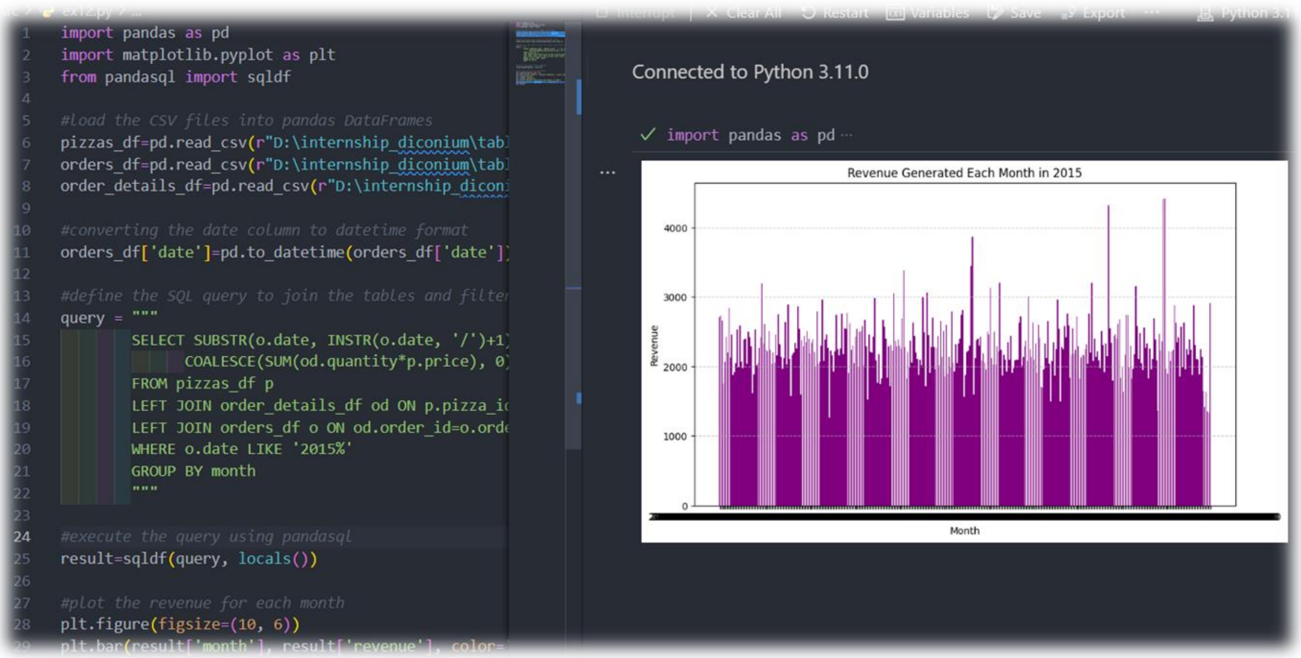
```
Connected to Python 3.11.0

✓ import pandas as pd ···

··· Total revenue generated in 2015: 817860.0
```

12. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd
import matplotlib.pyplot as plt
from pandasql import sqldf

#load the CSV files into pandas DataFrames
pizzas_df=pd.read_csv(r"D:\internship_diconium\tables\pizzas.csv")
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orders.csv")
order_details_df=pd.read_csv(r"D:\internship_diconium\tables\order_details.csv")

#converting the date column to datetime format
orders_df['date']=pd.to_datetime(orders_df['date'])

#define the SQL query to join the tables and filter transactions in 2015, grouped by month
query = """
    SELECT SUBSTR(o.date, INSTR(o.date, '/')+1) AS month,
        COALESCE(SUM(od.quantity*p.price), 0) AS revenue
    FROM pizzas_df p
    LEFT JOIN order_details_df od ON p.pizza_id=od.pizza_id
    LEFT JOIN orders_df o ON od.order_id=o.order_id
    WHERE o.date LIKE '2015%'
    GROUP BY month
    """

#execute the query using pandasql
result=sqldf(query, locals())

#plot the revenue for each month
plt.figure(figsize=(10, 6))
plt.bar(result['month'], result['revenue'], color='purple')
plt.xlabel('Month')
plt.ylabel('Revenue')
plt.title('Revenue Generated Each Month in 2015')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

- The result:



```
import pandas as pd
import matplotlib.pyplot as plt
from pandasql import sqldf

#load the CSV files into pandas DataFrames
pizzas_df=pd.read_csv(r"D:\internship_diconium\tabl
orders_df=pd.read_csv(r"D:\internship_diconium\tabl
order_details_df=pd.read_csv(r"D:\internship_diconi

#converting the date column to datetime format
orders_df['date']=pd.to_datetime(orders_df['date']

#define the SQL query to join the tables and filter
query = """
        SELECT SUBSTR(o.date, INSTR(o.date, '/')+1)
               COALESCE(SUM(od.quantity*p.price), 0)
        FROM pizzas_df p
        LEFT JOIN order_details_df od ON p.pizza_id
        LEFT JOIN orders_df o ON od.order_id=o.orde
        WHERE o.date LIKE '2015%'
        GROUP BY month
        """

#execute the query using pandasql
result=sqldf(query, locals())

#plot the revenue for each month
plt.figure(figsize=(10, 6))
plt.bar(result['month'], result['revenue'], color=
```

Connected to Python 3.11.0

```
✓ import pandas as pd ···
```

- Conclusion: I observe that revenues seem to vary depending on the month. For example, we can see that revenues appear to be higher during the warm months and lower during the cold months. This variation may be influenced by seasonal factors such as vacations or holidays.

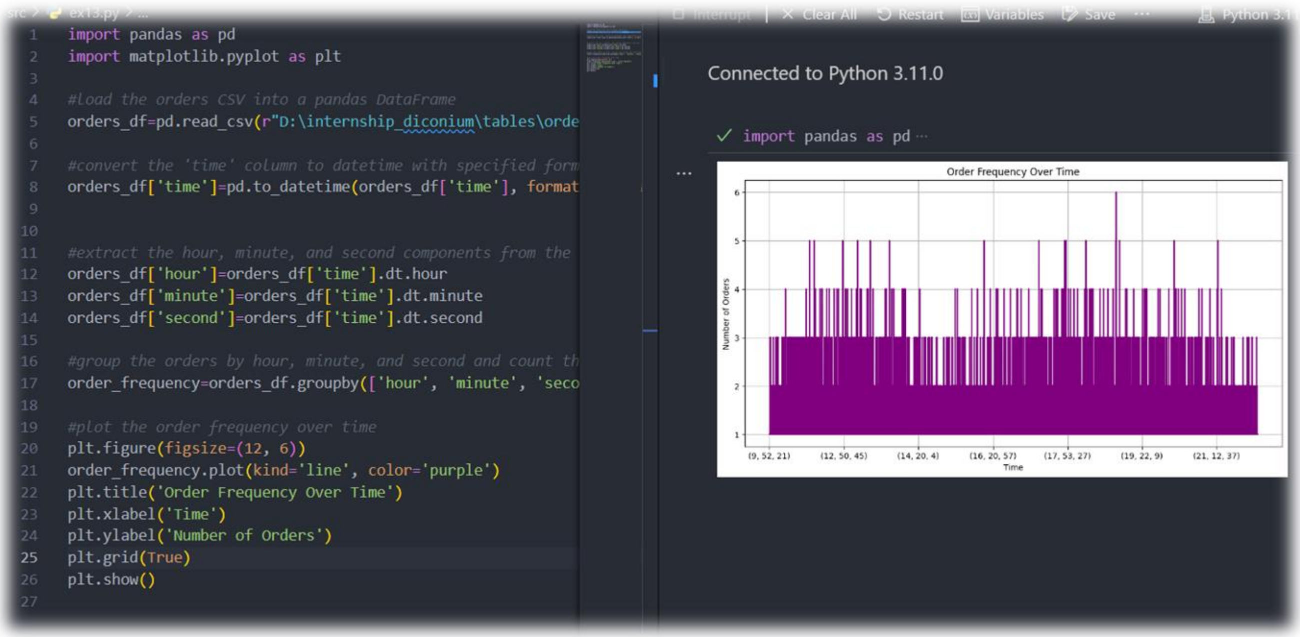
13. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd
import matplotlib.pyplot as plt

#load the orders CSV into a pandas DataFrame
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orders.csv")

#convert the 'time' column to datetime with specified format
orders_df['time']=pd.to_datetime(orders_df['time'], format='%H:%M:%S')
```

```
#extract the hour, minute, and second components from the 'time' column
orders_df['hour']=orders_df['time'].dt.hour
orders_df['minute']=orders_df['time'].dt.minute
orders_df['second']=orders_df['time'].dt.second

#group the orders by hour, minute, and second and count the number of orders for each timestamp
order_frequency=orders_df.groupby(['hour', 'minute', 'second']).size()

#plot the order frequency over time
plt.figure(figsize=(12, 6))
order_frequency.plot(kind='line', color='purple')
plt.title('Order Frequency Over Time')
plt.xlabel('Time')
plt.ylabel('Number of Orders')
plt.grid(True)
plt.show()
```

- The result:



```
import pandas as pd
import matplotlib.pyplot as plt

#load the orders CSV into a pandas DataFrame
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orde

#convert the 'time' column to datetime with specified form
orders_df['time']=pd.to_datetime(orders_df['time'], format


#extract the hour, minute, and second components from the
orders_df['hour']=orders_df['time'].dt.hour
orders_df['minute']=orders_df['time'].dt.minute
orders_df['second']=orders_df['time'].dt.second

#group the orders by hour, minute, and second and count th
order_frequency=orders_df.groupby(['hour', 'minute', 'seco

#plot the order frequency over time
plt.figure(figsize=(12, 6))
order_frequency.plot(kind='line', color='purple')
plt.title('Order Frequency Over Time')
plt.xlabel('Time')
plt.ylabel('Number of Orders')
plt.grid(True)
plt.show()

```

```
Connected to Python 3.11.0

✓ import pandas as pd ···
```

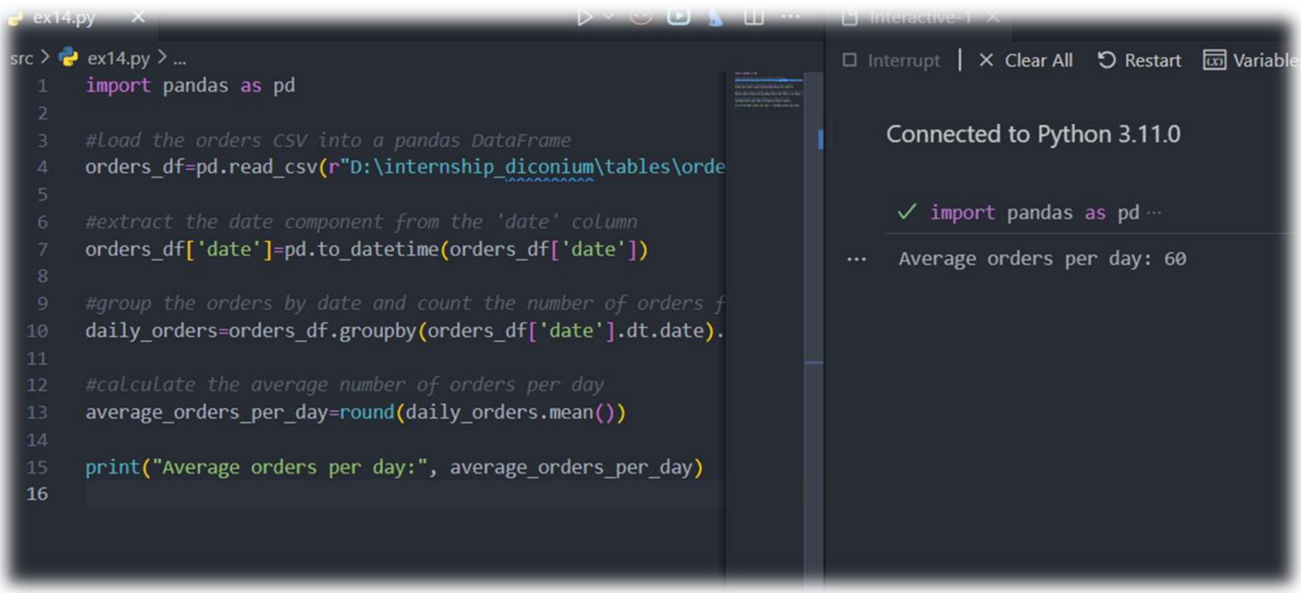
- Conclusion: The graph shows order frequency throughout the day, revealing busy peak hours and quieter times. This insight aids in resource planning for better customer service.

14. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd

#load the orders CSV into a pandas DataFrame
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orders.csv")

#extract the date component from the 'date' column
orders_df['date']=pd.to_datetime(orders_df['date'])

#group the orders by date and count the number of orders for each date
daily_orders=orders_df.groupby(orders_df['date'].dt.date).size()

#calculate the average number of orders per day
average_orders_per_day=round(daily_orders.mean())

print("Average orders per day:", average_orders_per_day)
```

- The result:



```
import pandas as pd

#load the orders CSV into a pandas DataFrame
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orde

#extract the date component from the 'date' column
orders_df['date']=pd.to_datetime(orders_df['date'])

#group the orders by date and count the number of orders f
daily_orders=orders_df.groupby(orders_df['date'].dt.date).

#calculate the average number of orders per day
average_orders_per_day=round(daily_orders.mean())

print("Average orders per day:", average_orders_per_day)
```

Connected to Python 3.11.0

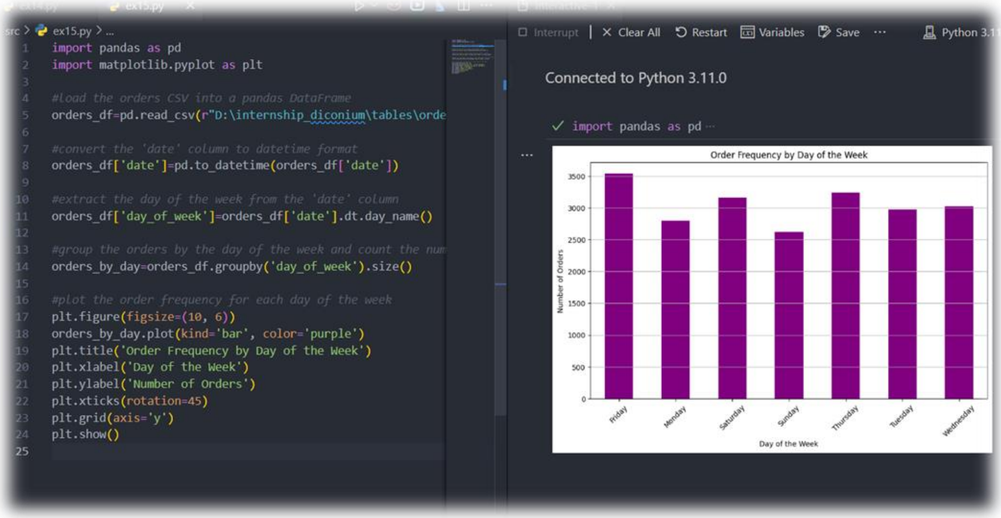
```
✓ import pandas as pd ···
Average orders per day: 60
```

15. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd
import matplotlib.pyplot as plt

#load the orders CSV into a pandas DataFrame
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orders.csv")

#convert the 'date' column to datetime format
orders_df['date']=pd.to_datetime(orders_df['date'])

#extract the day of the week from the 'date' column
orders_df['day_of_week']=orders_df['date'].dt.day_name()

#group the orders by the day of the week and count the number of orders for each day
orders_by_day=orders_df.groupby('day_of_week').size()

#plot the order frequency for each day of the week
plt.figure(figsize=(10, 6))
orders_by_day.plot(kind='bar', color='purple')
plt.title('Order Frequency by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Orders')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()
```

- The result:



```
import pandas as pd
import matplotlib.pyplot as plt

#load the orders CSV into a pandas DataFrame
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orde

#convert the 'date' column to datetime format
orders_df['date']=pd.to_datetime(orders_df['date'])

#extract the day of the week from the 'date' column
orders_df['day_of_week']=orders_df['date'].dt.day_name()

#group the orders by the day of the week and count the num
orders_by_day=orders_df.groupby('day_of_week').size()

#plot the order frequency for each day of the week
plt.figure(figsize=(10, 6))
orders_by_day.plot(kind='bar', color='purple')
plt.title('Order Frequency by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Orders')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()
```

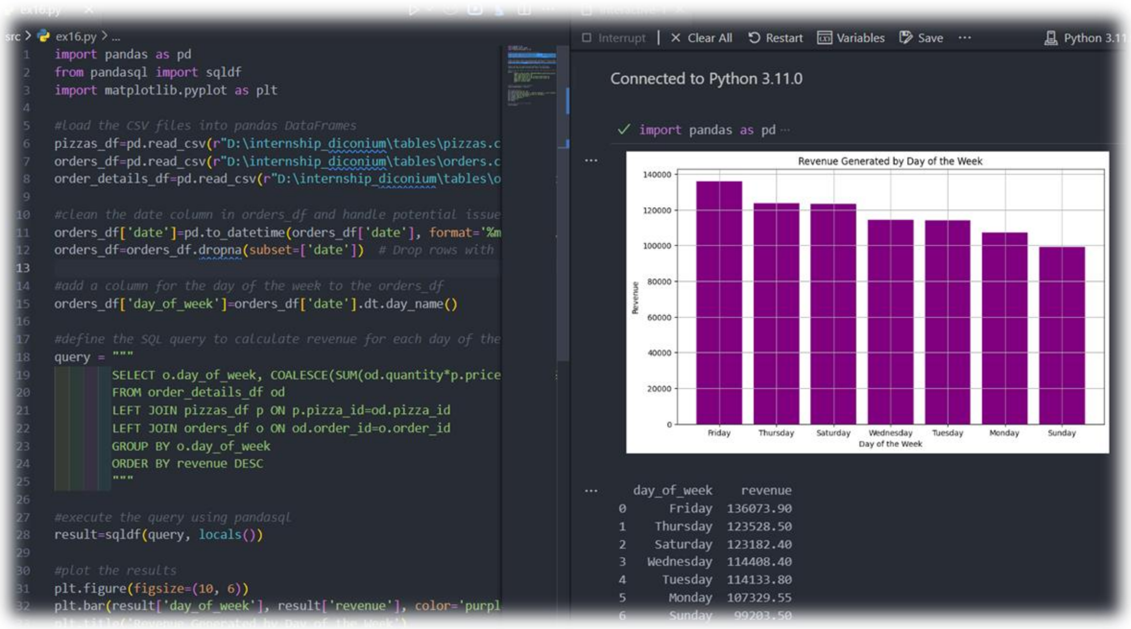
- Conclusion: Based on the graph, it seems like Fridays are the most crowded days of the week. This could be because people tend to dine out more often as the workweek comes to a close, or perhaps it's influenced by social customs and traditions.

16. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd
from pandasql import sqldf
import matplotlib.pyplot as plt

#load the CSV files into pandas DataFrames
pizzas_df=pd.read_csv(r"D:\internship_diconium\tables\pizzas.csv")
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orders.csv")
order_details_df=pd.read_csv(r"D:\internship_diconium\tables\order_details.csv")

#clean the date column in orders_df and handle potential issues
orders_df['date']=pd.to_datetime(orders_df['date'], format='%m/%d/%Y', errors='coerce')
orders_df=orders_df.dropna(subset=['date'])  # Drop rows with invalid dates

#add a column for the day of the week to the orders_df
orders_df['day_of_week']=orders_df['date'].dt.day_name()

#define the SQL query to calculate revenue for each day of the week
query = """
    SELECT o.day_of_week, COALESCE(SUM(od.quantity*p.price), 0) AS revenue
    FROM order_details_df od
    LEFT JOIN pizzas_df p ON p.pizza_id=od.pizza_id
    LEFT JOIN orders_df o ON od.order_id=o.order_id
    GROUP BY o.day_of_week
    ORDER BY revenue DESC
    """

#execute the query using pandasql
result=sqldf(query, locals())

#plot the results
plt.figure(figsize=(10, 6))
plt.bar(result['day_of_week'], result['revenue'], color='purple')
plt.title('Revenue Generated by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Revenue')
plt.grid(True)
plt.show()

#print the result for reference
print(result)
```

- The result:



src > ex16.py > ...

```
import pandas as pd
from pandasql import sqldf
import matplotlib.pyplot as plt

#load the CSV files into pandas DataFrames
pizzas_df=pd.read_csv(r"D:\internship_diconium\tables\pizzas.c
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orders.c
order_details_df=pd.read_csv(r"D:\internship_diconium\tables\o

#clean the date column in orders_df and handle potential issue
orders_df['date']=pd.to_datetime(orders_df['date'], format='%m
orders_df=orders_df.dropna(subset=['date'])  # Drop rows with

#add a column for the day of the week to the orders_df
orders_df['day_of_week']=orders_df['date'].dt.day_name()

#define the SQL query to calculate revenue for each day of the
query = """
        SELECT o.day_of_week, COALESCE(SUM(od.quantity*p.price
        FROM order_details_df od
        LEFT JOIN pizzas_df p ON p.pizza_id=od.pizza_id
        LEFT JOIN orders_df o ON od.order_id=o.order_id
        GROUP BY o.day_of_week
        ORDER BY revenue DESC
        """

#execute the query using pandasql
result=sqldf(query, locals())

#plot the results
plt.figure(figsize=(10, 6))
plt.bar(result['day_of_week'], result['revenue'], color='purpl
```

Connected to Python 3.11.0

```
import pandas as pd ···
```

```
  day_of_week    revenue
0      Friday  136073.90
1    Thursday  123528.50
2    Saturday  123182.40
3   Wednesday  114408.40
4     Tuesday  114133.80
5      Monday  107329.55
6      Sunday   99203.50
```

- Conclusion: Based on the graph, you can observe which day of the week generates the highest revenue. This insight helps understand customer behavior and optimize restaurant operations. For instance, if Fridays show the highest revenue, it could be due to increased dining out as people celebrate the end of the workweek.

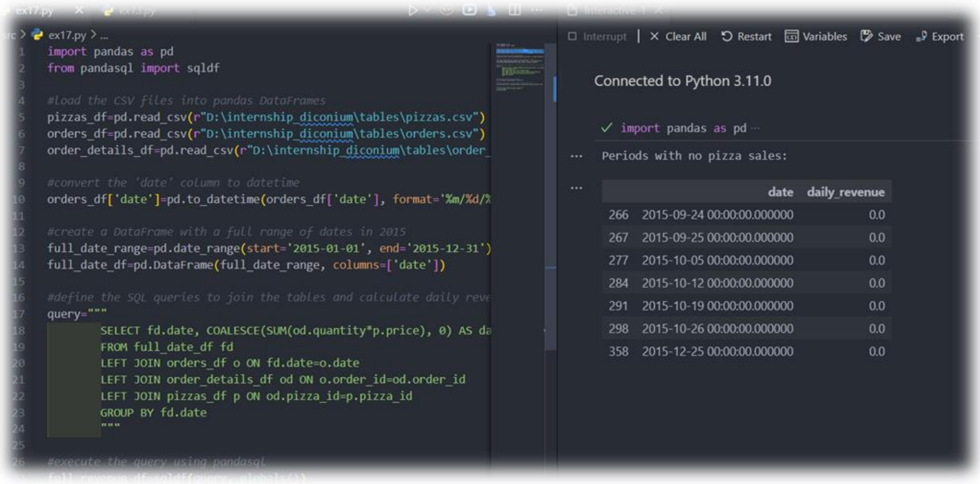
17. I used python and the jupyter notebook extension to display the result more clearly

- The code:

```
import pandas as pd
from pandasql import sqldf

#load the CSV files into pandas DataFrames
pizzas_df=pd.read_csv(r"D:\internship_diconium\tables\pizzas.csv")
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orders.csv")
order_details_df=pd.read_csv(r"D:\internship_diconium\tables\order_details.csv")
```

```
#convert the 'date' column to datetime
orders_df['date']=pd.to_datetime(orders_df['date'], format='%m/%d/%Y')

#create a DataFrame with a full range of dates in 2015
full_date_range=pd.date_range(start='2015-01-01', end='2015-12-31')
full_date_df=pd.DataFrame(full_date_range, columns=['date'])

#define the SQL queries to join the tables and calculate daily revenue
query="""
        SELECT fd.date, COALESCE(SUM(od.quantity*p.price), 0) AS daily_revenue
        FROM full_date_df fd
        LEFT JOIN orders_df o ON fd.date=o.date
        LEFT JOIN order_details_df od ON o.order_id=od.order_id
        LEFT JOIN pizzas_df p ON od.pizza_id=p.pizza_id
        GROUP BY fd.date
        """

#execute the query using pandasql
full_revenue_df=sqldf(query, globals())

#identify the dates with no sales (daily_revenue is 0)
no_sales_dates=full_revenue_df[full_revenue_df['daily_revenue']==0]

#display the periods with no pizza sales
print("Periods with no pizza sales:")
no_sales_dates
```

- The result:



```
import pandas as pd
from pandasql import sqldf

#load the CSV files into pandas DataFrames
pizzas_df=pd.read_csv(r"D:\internship_diconium\tables\pizzas.csv")
orders_df=pd.read_csv(r"D:\internship_diconium\tables\orders.csv")
order_details_df=pd.read_csv(r"D:\internship_diconium\tables\order_

#convert the 'date' column to datetime
orders_df['date']=pd.to_datetime(orders_df['date'], format='%m/%d/%

#create a DataFrame with a full range of dates in 2015
full_date_range=pd.date_range(start='2015-01-01', end='2015-12-31')
full_date_df=pd.DataFrame(full_date_range, columns=['date'])

#define the SQL queries to join the tables and calculate daily reve
query="""
        SELECT fd.date, COALESCE(SUM(od.quantity*p.price), 0) AS da
        FROM full_date_df fd
        LEFT JOIN orders_df o ON fd.date=o.date
        LEFT JOIN order_details_df od ON o.order_id=od.order_id
        LEFT JOIN pizzas_df p ON od.pizza_id=p.pizza_id
        GROUP BY fd.date
        """

#execute the query using pandasql
```

```
Connected to Python 3.11.0

✓ import pandas as pd ···

Periods with no pizza sales:

                          date  daily_revenue
266 2015-09-24 00:00:00.000000            0.0
267 2015-09-25 00:00:00.000000            0.0
277 2015-10-05 00:00:00.000000            0.0
284 2015-10-12 00:00:00.000000            0.0
291 2015-10-19 00:00:00.000000            0.0
298 2015-10-26 00:00:00.000000            0.0
358 2015-12-25 00:00:00.000000            0.0
```

- Conclusion: The absence of pizza sales during certain periods could be attributed to factors like restaurant closures, changes in customer behavior, or external influences like holidays or events.