Data Engineer Case Interview

By: Daryl Ong Chun Kit

Section 1: Pipeline

Language: Python3

Tools used: VisualStudioCode, Apache Airflow, Astro CLI, Docker

Dag object
and
Task operator

```python
python_task = PythonOperator(
    task_id='Section1',
    python_callable=process_datasets,
    dag=dag,
)
```

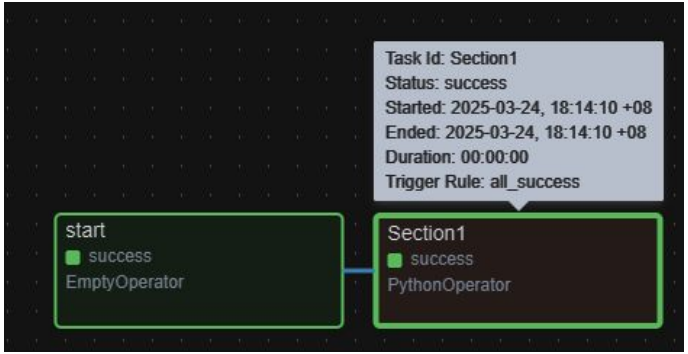
```python
# Define the DAG
dag = DAG(
    'Section1',  # DAG name
    default_args=default_args,
    description='process and output data files on daily interval',
    schedule_interval='0 1 * * *',  # Run every day at 1 AM
    catchup=False,  # Don't backfill past DAG runs
)
```

Results

Dataset processing function

```python
def process_datasets():
    # Read Datasets--------------------------------------------
    df1 = pd.read_csv('/opt/airflow/csv_files/dataset1.csv')
    df2 = pd.read_csv('/opt/airflow/csv_files/dataset2.csv')

    print(df1.head())
    print(df2.head())

    # Process dataset1.csv-------------------------------------------
    # Clean NaN or empty names and price = 0
    df1_cleaned = df1[(df1['name'].notna()) & (df1['name'] != '') & (df1['price'] != 0)]

    # Split name into first_name and last_name
    df1_cleaned[['first_name', 'last_name']] = df1_cleaned['name'].str.split(' ', expand=True)

    # Remove name column
    df1_cleaned = df1_cleaned.drop(columns=['name'])

    # Reorder columns
    df1_cleaned = df1_cleaned[['first_name', 'last_name', 'price']]
    df1_cleaned['above_100'] = df1_cleaned['price'] > 100

    print(df1_cleaned.head(10))
```

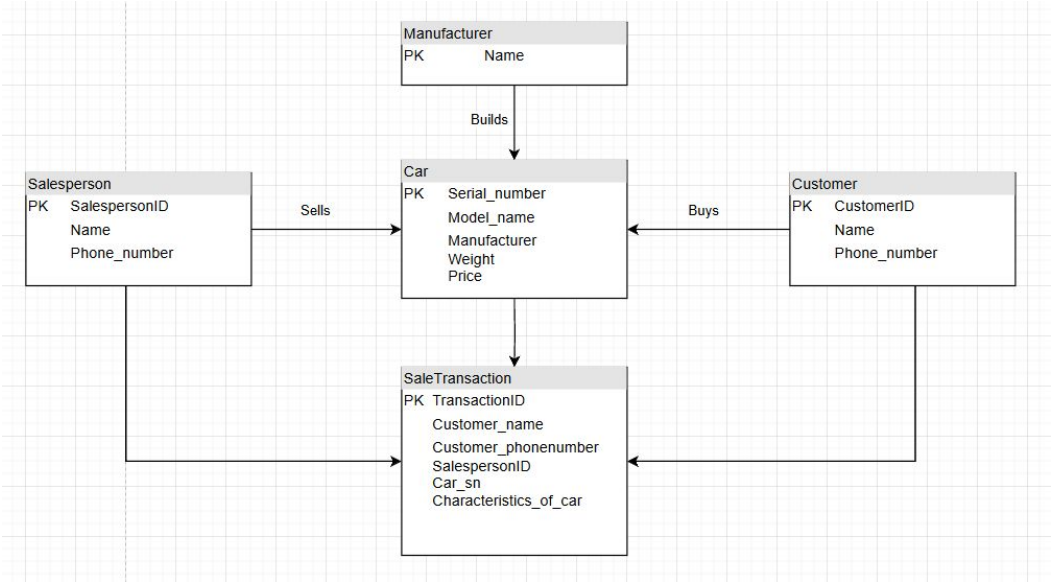
Successful execution of the dag



Section 2: Database

Language: PosgreSQL

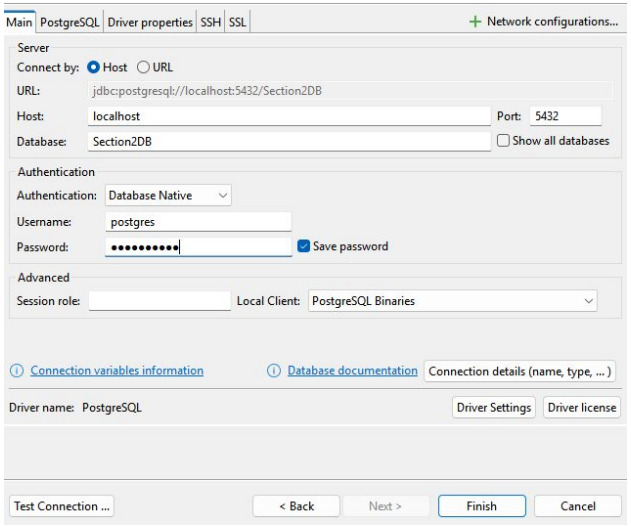
Tools used: Docker Image, DBeaver, Draw.io



Results

Building the postgresql with docker image

```
C:\Users\daryl\OneDrive\Desktop\Job Applications 2025\SkillsFuture\Section2>docker build -t section2_postgres_image .
[+] Building 0.4s (7/7) FINISHED                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                0.0s
 => => transferring dockerfile: 340B                                                                                0.0s
 => WARN: SecretsUsedInArgOrEnv: Do not use ARG or ENV instructions for sensitive data (ENV "POSTGRES_PASSWORD")   0.0s
 => [internal] load metadata for docker.io/library/postgres:latest                                                  0.0s
 => [internal] load .dockerignore                                                                                   0.0s
 => => transferring context: 2B                                                                                     0.0s
 => [internal] load build context                                                                                   0.0s
 => => transferring context: 37B                                                                                    0.0s
 => [1/2] FROM docker.io/library/postgres:latest                                                                    0.0s
 => CACHED [2/2] COPY tableScript.sql /docker-entrypoint-initdb.d/                                                  0.0s
 => exporting to image                                                                                              0.1s
 => => exporting layers                                                                                             0.0s
 => => writing image sha256:17bf387356d2951fa1b00ca47e0ac048d62ef7af790b8d97ff3fcee847d69ada                        0.0s
 => => naming to docker.io/library/section2_postgres_image                                                          0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/uv6r4ceweo0gvljegd4eih7j2

 1 warning found (use docker --debug to expand):
 - SecretsUsedInArgOrEnv: Do not use ARG or ENV instructions for sensitive data (ENV "POSTGRES_PASSWORD") (line 5)

C:\Users\daryl\OneDrive\Desktop\Job Applications 2025\SkillsFuture\Section2>docker run --name section2_postgres_containe
r -p 5432:5432 -d section2_postgres_image
73f7d7e1abad41b3c8d20c646294fd5a6643fe86aa0a1106cfb1f7c75f3ff89d
```



Results

Query 1 and 2 outputs

```sql
-- Output list of cust and their spending
select c.customerID, c.name as customer_name, SUM(car.price) as total_spending
from transaction t
join customer c on t.customerID = c.customerID
join car on t.car_sn = car.serial_number
group by c.customerID, c.name;
```

manufacturer 1 (2) | customer 1 (3) | salesperson 1 (4) | transaction 1 (5)

ct c.customerID, c.name as customer_name, ! Enter a SQL expression to filter results (use Ctrl+Space)

customerid	customer_name	total_spending
C002	Jane Smith	68,000
C001	John Doe	60,000
C004	Emily Davis	60,000
C003	Alex Johnson	57,000
C005	Michael Brown	74,000

```sql
-- Output top 3 car manufactures by sales
select m.name as manufacturer_name, count(t.car_sn) as cars_sold
from transaction t
join car on t.car_sn = car.serial_number
join manufacturer m on car.manufacturer = m.name
group by m.name
order by cars_sold desc
limit 3;
```

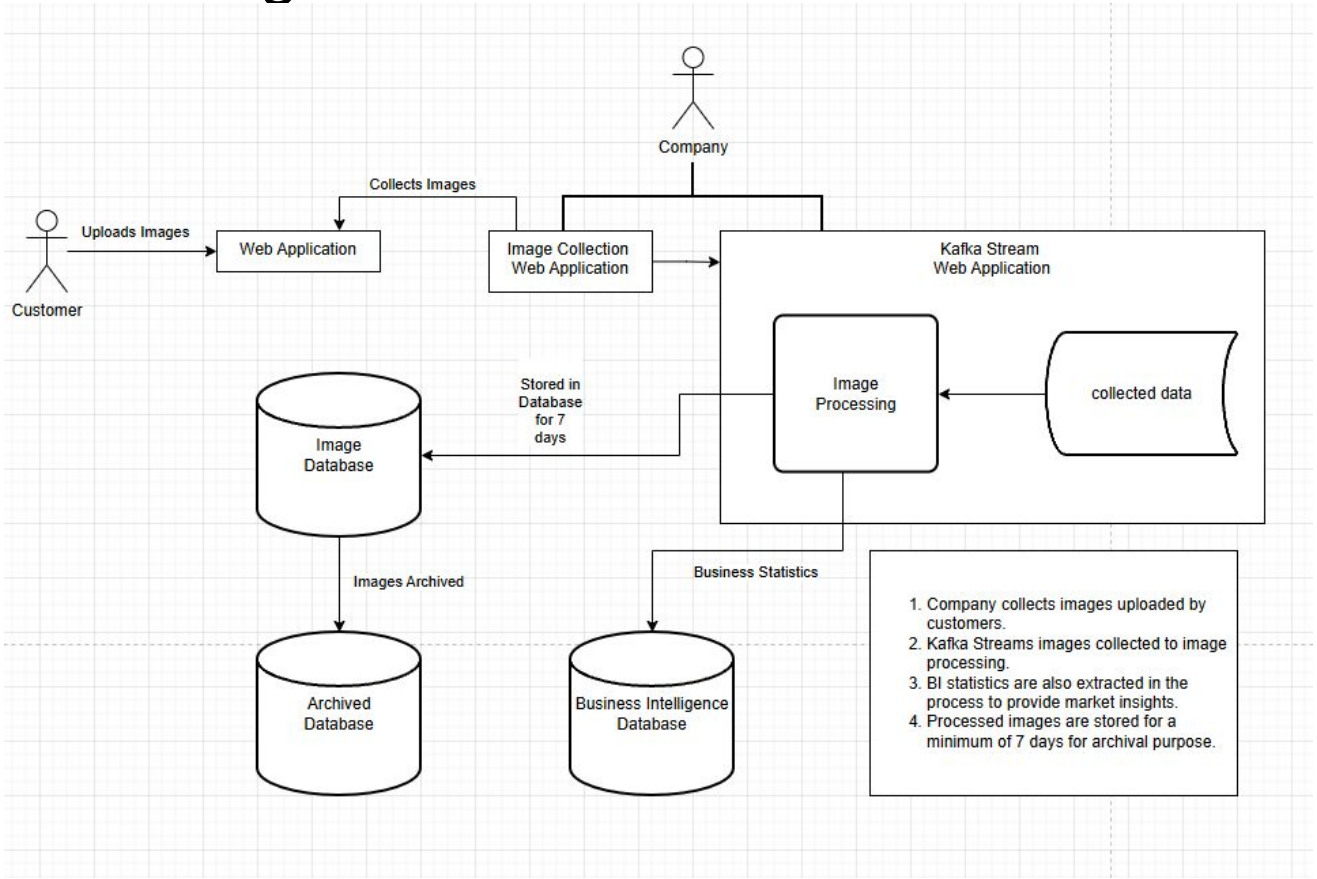
manufacturer 1 (2) | customer 1 (3) | salesperson 1 (4) | transa

ct m.name as manufacturer_name, count(t.c Enter a SQL expression to filter results (use

manufacturer_name	cars_sold
Ford	2
Honda	2
Mercedes	2

Section 3: System Design

Tools used: Draw.io



Section 4: API and Graph

Language: Python3

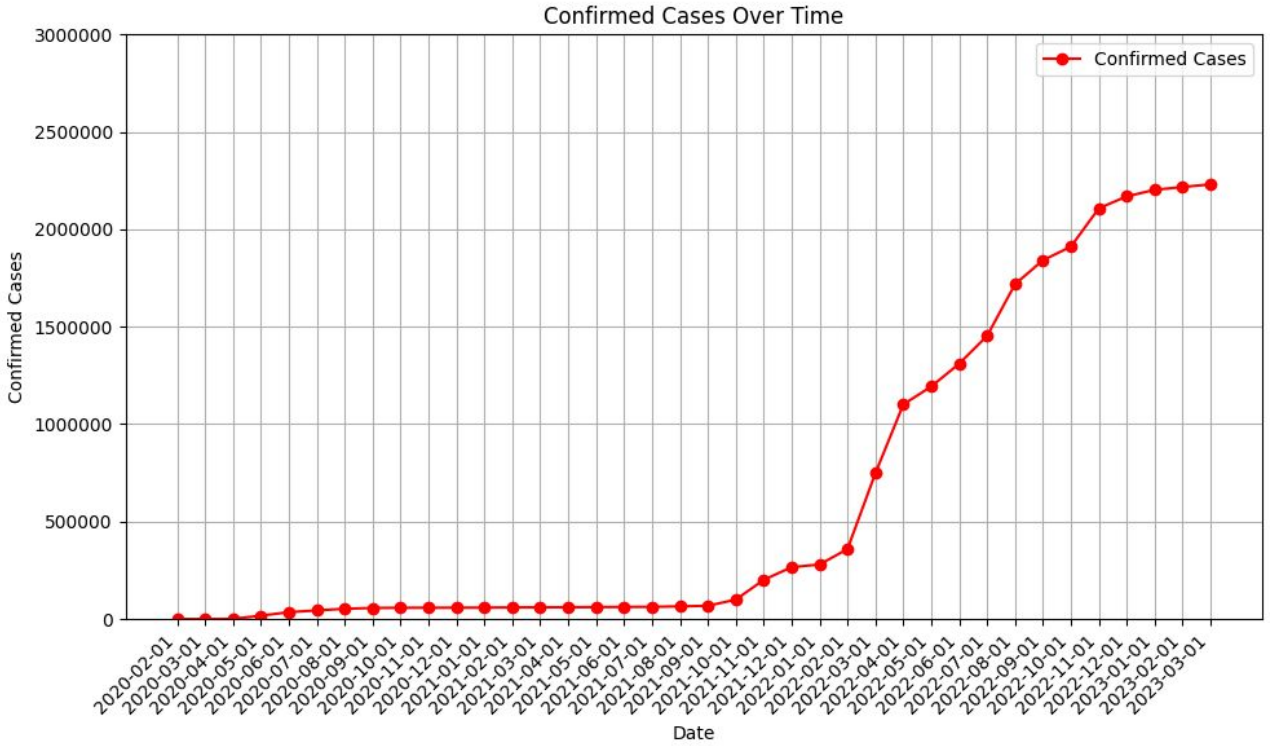
Tools used: requests, datetime, dateutil, matplotlib

```python
import requests
from datetime import datetime, timedelta
from dateutil.relativedelta import relativedelta

def get_covid_data(start_date, end_date):
    covid_data = []

    # Lop over range of date
    current_date = start_date

    while (current_date <= end_date):
        formatted_date = current_date.strftime('%Y-%m-%d')

        # API URL
        url = "https://covid-api.com/api/reports/total"

        # Parameters for the API request
        params = {
            "date": formatted_date,
            "iso": "SGP"
        }

        # Make the request to the API
        response = requests.get(url, params=params)

        if response.status_code == 200:
            data = response.json().get("data", {})
            confirmed = data.get("confirmed", "0")
            covid_data.append({"date": formatted_date, "confirmed": confirmed})
            print(formatted_date, confirmed)
        else:
            print(f"Error: {response.status_code}")
            print(response.text)

        current_date += relativedelta(months=1)

    return covid_data
```

```python
start_date = datetime(2020, 2, 1)
end_date = datetime(2023, 3, 1)

data = get_covid_data(start_date, end_date)

# for entry in data:
#     print(entry)


# import plot lib
import matplotlib.pyplot as plt

# extract X and Y
dates = [entry["date"] for entry in data]
values = [entry["confirmed"] for entry in data]

# Plotting the graph
plt.figure(figsize=(10, 6))
plt.plot(dates, values, marker='o', color='r', linestyle='-', label="Confirmed Cases")

# Customizing the plot
plt.title('Confirmed Cases Over Time')
plt.xlabel('Date')
plt.ylabel('Confirmed Cases')
plt.xticks(rotation=45, ha='right')  # Rotate the date labels for better readability
plt.ylim(0, 3000000)
plt.ticklabel_format(style="plain", axis="y")
plt.grid(True)
plt.tight_layout()

# Show the plot
plt.legend()
plt.show()
```

Results

Displayed Graph of Number of cases overtime in singapore in months

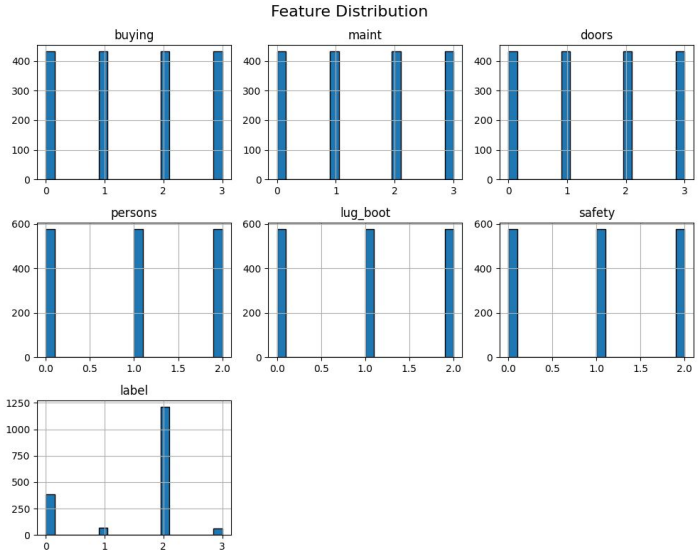


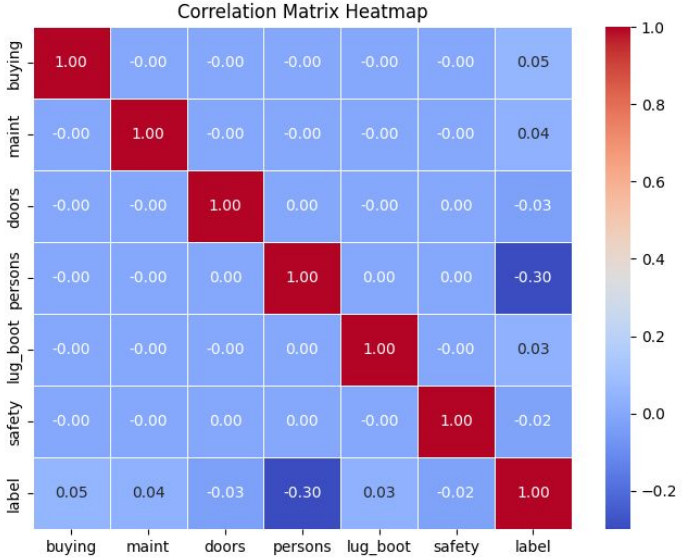
Section 5: ML

Language: Python3

Tools used: SciKit-Learn, Jupyter, Visual Studio Code

Visualisation to better understand dataset





Results

Label Encoding to numerical values

```
Click to add a breakpoint                al data to numerical data
    le = LabelEncoder()

    # encoding to each categorical column
    for column in df.columns:
        df[column] = le.fit_transform(df[column])

    # printing df
    print(df.head())
    print(df.tail())
```

✓ 0.0s

```
   buying  maint  doors  persons  lug_boot  safety  label
0       3      3      0        0         2       1      2
1       3      3      0        0         2       2      2
2       3      3      0        0         2       0      2
3       3      3      0        0         1       1      2
4       3      3      0        0         1       2      2
      buying  maint  doors  persons  lug_boot  safety  label
1723       1      1      3        2         1       2      1
1724       1      1      3        2         1       0      3
1725       1      1      3        2         0       1      2
1726       1      1      3        2         0       2      1
1727       1      1      3        2         0       0      3
```

Selecting features and data splitting

```
# import libs for model training
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score

# Split data
X = df.drop(columns=["buying", "persons"])
y = df["buying"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
X
```

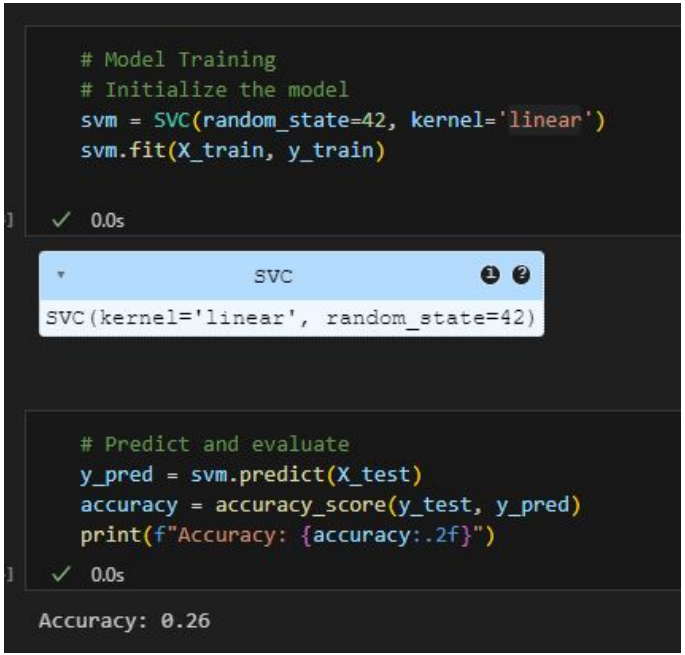
✓ 0.0s

	maint	doors	lug_boot	safety	label
0	3	0	2	1	2
1	3	0	2	2	2
2	3	0	2	0	2
3	3	0	1	1	2
4	3	0	1	2	2
...	...	...	...	...	...
1723	1	3	1	2	1
1724	1	3	1	0	3
1725	1	3	0	1	2
1726	1	3	0	2	1

Results

Training model and testing

Model used: Support Vector Machine (SVM) has best performing model compared to other models



Results

Predicting buying price

- Maintenance = High
- Number of doors = 4
- Lug Boot Size = Big
- Safety = High
- Class Value = Good

```
# Predict buying price
# Load the model
svm_loaded = joblib.load("random_forest_model.pkl")
```

✓ 0.0s

```
# Data for testing
# test: maint:high, doors:4, lug_boot:big, safety:high
test = ['high', '4', 'big', 'high', 'good']

# Convert to DataFrame
enc = le.fit_transform(test)
enc_test = pd.DataFrame([enc], columns=['maint', 'doors','lug_boot', 'safety', 'label'])
enc_test
```

✓ 0.0s

	maint	doors	lug_boot	safety	label
0	3	0	1	3	2

```
pred = svm_loaded.predict(enc_test)
pred_convert = unique_values[pred]
print(pred)
print(pred_convert)
```

✓ 0.0s

```
[3]
['vhigh']
```