Data Engineer Case Interview

Luah Jun Yang

Section 1:
Data
Pipelines

Section 1 - Processing Data files on a regular interval

1

Airflow as the scheduling Tool

I selected Airflow as the scheduling tool as I have had prior experience with it. I set a session-based Airflow UI using ngrok via a Google Colab Notebook which is the python Notebook I submitted

2

Data processing and output – Python Operator

I wrote a function to perform the necessary Data processing and output the new CSV files using pandas. The function was then run using 2 task instances of PythonOperator which takes in the dataset1.csv and dataset2.csv as input arguments

Section 1 =Processing Data files on a regular interval

```
default_args = {
    'start_date': datetime(2024, 1, 22),
    'schedule_interval': "10 1 * * *",
    'retries': 3,
    'retry_delay': timedelta(minutes=5),
    'depends_on_past': False,
    'email_on_failure': False,
    'email_on_retry': False,
    'email': 'example@admin.com'
}
```

Cron Schedule set for 0110hrs everyday, as I am expecting the new data file to come in at 1am everyday

Section 1- Processing Data files on a regular interval

```python
def process_data(input_file_location):
  df = pd.read_csv(input_file_location)
  #To remove all rows with no name column
  df = df.dropna(subset=['name'])
  #To split name column up into first_name and last_name columns
  df['name'] = df['name'].astype(str)
  df[['first_name', 'last_name']] = df['name'].str.split(' ',n=1, expand = True)
  #To remove prepended zeros from price column, and convert back to numeric type from string
  df['price'] = df['price'].astype(str).str.lstrip('0')
  df['price'] = pd.to_numeric(df['price'], errors = 'coerce')
  #To create the above_100 column
  df['above_100'] = df['price'] > 100
  #To drop the original name column
  df.drop(columns = ['name'], inplace = True)
  original_file_name = os.path.basename(input_file_location)
  output_file_name = f'processed_{original_file_name}'
  output_path = os.path.join('/content/airflow/dags', output_file_name)
  #To output processed CSV files
  df.to_csv(output_path, index = False)
```

process_data function:
- Performs processing steps as specified by comments in Green
- Drops original name column
- Outputs CSV files with new name ‘processed_{original_file_name}.csv’

Section 1- Processing Data files on a regular interval

```
40  with DAG('section_1_data_pipelines_dag', default_args = default_args) as dag:
41
42      process_dataset1_task = PythonOperator(
43          task_id = 'process_dataset1_task',
44          python_callable = process_data,
45          op_kwargs = {'input_file_location': '/content/airflow/dags/dataset1.csv'}
46      )
47
48      process_dataset2_task = PythonOperator(
49          task_id = 'process_dataset2_task',
50          python_callable = process_data,
51          op_kwargs = {'input_file_location': '/content/airflow/dags/dataset2.csv'}
52      )
53
54  process_dataset1_task >> process_dataset2_task
```

- 2 instances of PythonOperator calling the process_data() function, with the input files being dataset1.csv and dataset2.csv

Section 1- Output data files

	A	B	C	D
1	price	first_name	last_name	above_100
2	111	Gertrude	Hale	TRUE
3	67	Kaitlan	Smart	FALSE
4	92	Adelle	Oconnell	FALSE
5	139	Shay	Oneal	TRUE
6	55	Finley	Oneill	FALSE
7	65	Harri	Williamson	FALSE
8	44	Eira	Hackett	FALSE
9	49	Bentley	Larsen	FALSE
10	44	Justine	Kouma	FALSE
11	97	Aaron	Calderon	FALSE
12	54	Kye	Werner	FALSE
13	59	Riaan	Hood	FALSE
14	173	Margie	Thatcher	TRUE
15	4	Susanna	O'Ryan	FALSE
16	148	Lilith	Petersen	TRUE
17	176	Carina	Ayers	TRUE
18	30	Gracie	Frye	FALSE
19	29	Alayna	Weeks	FALSE
20	142	Paige	Bowes	TRUE
21	121	Sufyan	Field	TRUE
22	77	Kirstie	Wharton	FALSE
23	182	Diane	Greig	TRUE
24	161	Sophia	Dunlap	TRUE
25	180	Shiloh	Forster	TRUE
26	131	Sidrah	Roberts	TRUE
27	109	Safiyah	Burch	TRUE
28	80	Jannat	Hewitt	FALSE
29	73	Blaine	Nairn	FALSE
30	131	Amir	Pemberton	TRUE
31	4	Solomon	Mcgregor	FALSE
32	102	Celia	Hodges	TRUE

A	B	C	D
price	first_name	last_name	above_100
117	Lily-Rose	Crowther	TRUE
198	Teagan	Lynn	TRUE
39	Jerome	Gibbs	FALSE
157	Bobby	O'Moore	TRUE
	Eduardo	Whitfield	FALSE
120	Khalil	Kavanagh	TRUE
880	Huw	Weber	TRUE
172	Eathan	Bishop	TRUE
98	Alysha	Barnard	FALSE
122	Hettie	Santos	TRUE
143	Blaine	Drummond	TRUE
178	Mercy	Ramirez	TRUE
53	Merryn	Rutledge	FALSE
22	Homer	Johns	FALSE
20	Larry	Bender	FALSE
178	Amal	Haynes	TRUE
193	Kevin	Scott	TRUE
87	Yasir	Medrano	FALSE
67	Ksawery	Glass	FALSE
178	Santino	Macgregor	TRUE
41	Laurence	Wicks	FALSE
34	Luka	Baird	FALSE
111	Haniya	Schmidt	TRUE
189	Suhail	Humphrey	TRUE
48	Ira	Clayton	FALSE
170	Kaya	Odonnell	TRUE
12	Angelo	Santana	FALSE
197	Riley	Lloyd	TRUE
112	Kendrick	Raymond	TRUE
39	Ann	Appleton	FALSE
149	Sienna	Kendall	TRUE
	Dylan	Goff	FALSE
96	Tayyibah	Kidd	FALSE
61	Dale	Rodriguez	FALSE

Section 2: Databases

Section 2 - Database Infrastructure

Deliverables

1. Dockerfile to pull from provided link
2. DDL statements – create_db.sql and ER diagram
3. Queries folder – containing the 2 .sql files for the 2 queries

Section 2 - Database Infrastructure

Dockerfile + DDL

```
FROM postgres:latest

# Set environment variables for the PostgreSQL instance
ENV POSTGRES_USER=dealership
ENV POSTGRES_PASSWORD=yourpassword
ENV POSTGRES_DB=dealership_db

# Copy the DDL script that creates the tables
COPY create_db.sql /docker-entrypoint-initdb.d/
```

Dockerfile:

- Pulls the latest postgres image
- Sets env variables
- Copies the DDL script that will create the tables

```
CREATE TABLE manufacturers (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE
);

CREATE TABLE salespersons (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    phone VARCHAR(20)
);

CREATE TABLE cars (
    id INT PRIMARY KEY,
    manufacturer INT REFERENCES manufacturers(id),
    model_name VARCHAR(100) NOT NULL,
    serial_number VARCHAR(100) NOT NULL UNIQUE,
    weight DECIMAL(10, 2),
    price DECIMAL(10,2)
);

CREATE TABLE customers (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    phone VARCHAR(20)
);

CREATE TABLE transactions (
    id INT PRIMARY KEY,
    customer_id INT REFERENCES customers(id),
    salesperson_id INT REFERENCES salespersons(id),
    car_id INT REFERENCES car(id),
    transaction_date datetime
);
```

Section 2 - Database Infrastructure

Dockerfile + DDL

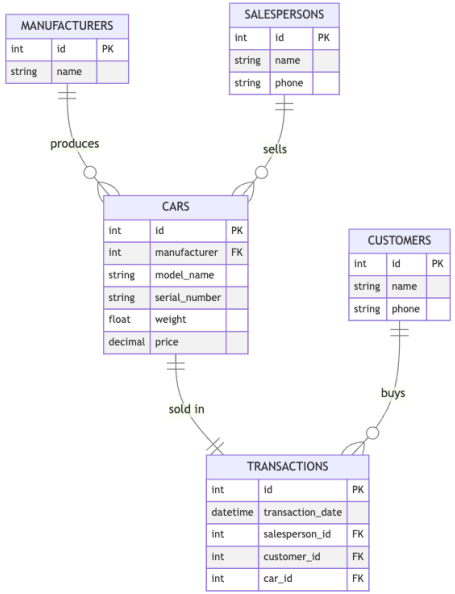
```sql
CREATE TABLE manufacturers (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE
);

CREATE TABLE salespersons (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    phone VARCHAR(20)
);

CREATE TABLE cars (
    id INT PRIMARY KEY,
    manufacturer INT REFERENCES manufacturers(id),
    model_name VARCHAR(100) NOT NULL,
    serial_number VARCHAR(100) NOT NULL UNIQUE,
    weight DECIMAL(10, 2),
    price DECIMAL(10,2)
);

CREATE TABLE customers (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    phone VARCHAR(20)
);

CREATE TABLE transactions (
    id INT PRIMARY KEY,
    customer_id INT REFERENCES customers(id),
    salesperson_id INT REFERENCES salespersons(id),
    car_id INT REFERENCES car(id),
    transaction_date datetime
);
```



DDL statement and ER diagram for the 5 tables to be created:

1. Manufacturers
2. Salespersons
3. Cars
4. Customers
5. Transactions

Section 2 - Database Infrastructure

Steps to Run

Step 2:

- **docker build** builds a new Docker image using the Dockerfile in the current directory and **–t dealership-db** tags the image with the name dealership-db.

Step 3:

- **docker run** creates and starts a new container from the dealership-db image created in the previous step
- **-d** runs the container in detached mode
- **--name dealership-db** names the container dealership-db
- **-p 5432:5432** maps port 5432 on your machine to port 5432 on the container which is PostgreSQL's default port

Section 2 - Database Infrastructure

Steps to Run

Steps to setup the PostgreSQL Database

1. Navigate to directory containing the Dockerfile and create_db.sql DDL script

```
cd/path/to/project
```

2. Build the image

```
docker build -t dealership-db .
```

3. Run the Container

```
docker run -d --name dealership-db -p 5432:5432 dealership-db
```

4. Verify the container is running

```
docker ps -a | grep dealership-db
```

5. Check the logs for any initialization error, there should be 5 `CREATE TABLE` statements that showed the 5 tables were created successfully by create_db.sql

```
docker logs dealership-db
```

6. Connect to the PostgreSQL database

```
docker exec -it dealership-db psql -U dealership -d dealership_db
```

Step 4:

- **docker ps –a** lists all containers
- **grep dealership-db** filters for containers only containing dealership-db

Step 5:

- **Displays logs**

Step 6:

- **docker exec –it** allocates an interactive terminal
- **psql –U dealership –d dealership-db** runs the PostgreSQL CLI client as the dealership user and connects to the dealership-db database

Section 2 - Database Infrastructure

Query 1

```sql
SELECT
    customers.name,
    SUM(cars.price) as total_spending
FROM customers
JOIN transactions
ON customers.id = transactions.customer_id
JOIN cars
ON transactions.car_id = car.id
GROUP BY customers.id;
```

- Selects customers' name and SUM() on their price to get their aggregated total spending
- Joins customers table with transactions table on customer_id
- Joins transactions table on cars table on car_id
- Group by customer_id to SUM their total_spending

Section 2 - Database Infrastructure

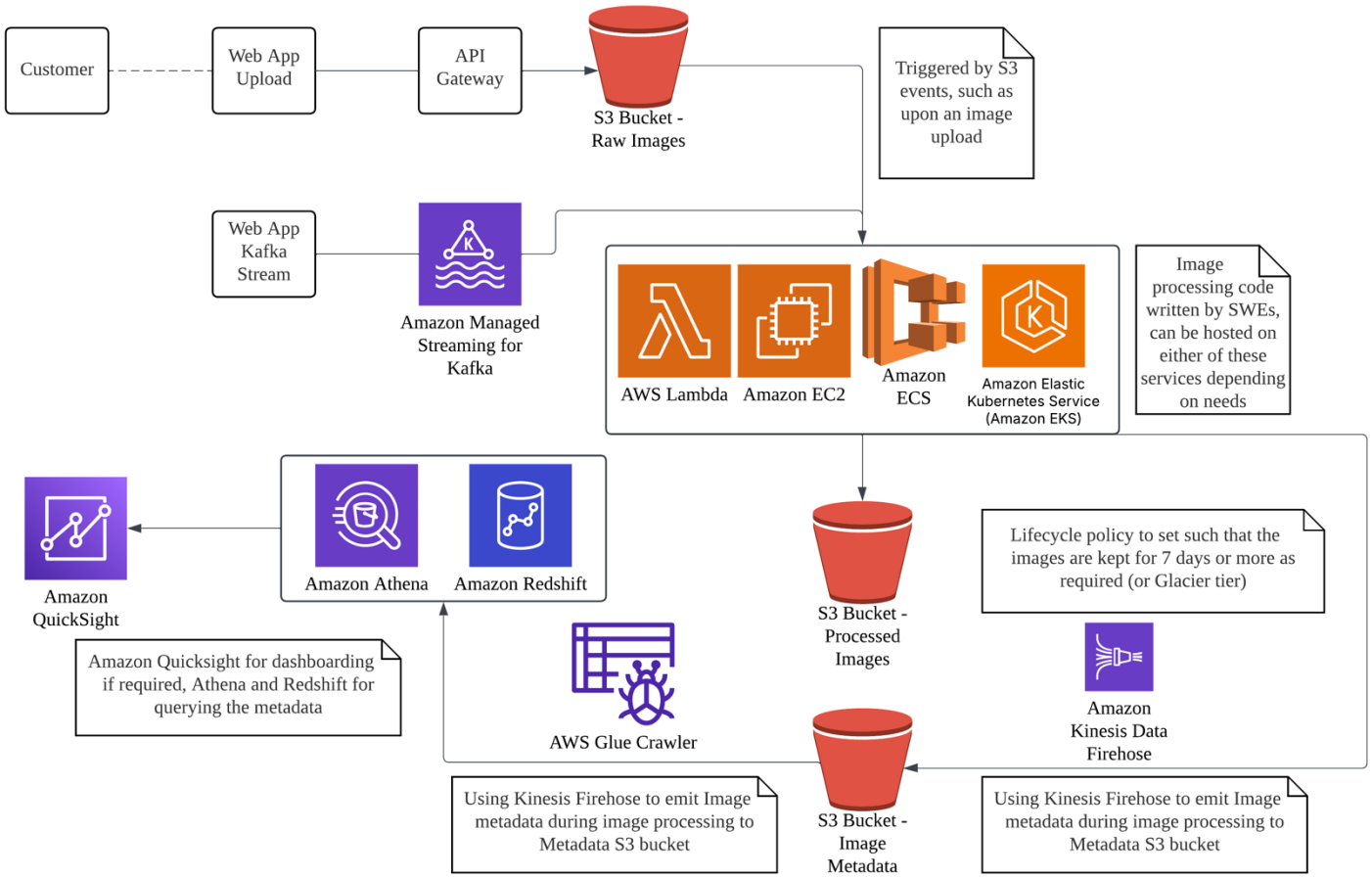
Query 2

```sql
SELECT
    manufacturers.name,
    COUNT(*) AS quantity_sold
FROM manufacturers
JOIN cars ON manufacturers.id = cars.manufacturer
JOIN transactions ON car.id = transactions.car_id
WHERE date_trunc('month', transactions.transaction_date) = date_trunc('month', CURRENT_DATE())
GROUP by manufacturers.name
ORDER BY quantity_sold DESC
LIMIT 3;
```

- Select Manufacturer's name and COUNT(*) to get the total number of cars sold by each manufacturer
- Joins manufacturers table with cars table on manufacturer_id
- Joins cars table with transactions table on car_id
- Use date_trunc('month') and current_date() to get just the rows of transactions for the current month
- Group by manufacturer's name for the COUNT(*)
- Order by quantity_sold DESC and LIMIT 3 to get the top 3 manufacturers in terms of number of cars sold

Section 3:
Cloud

Section 3 – Cloud Infrastructure (AWS)



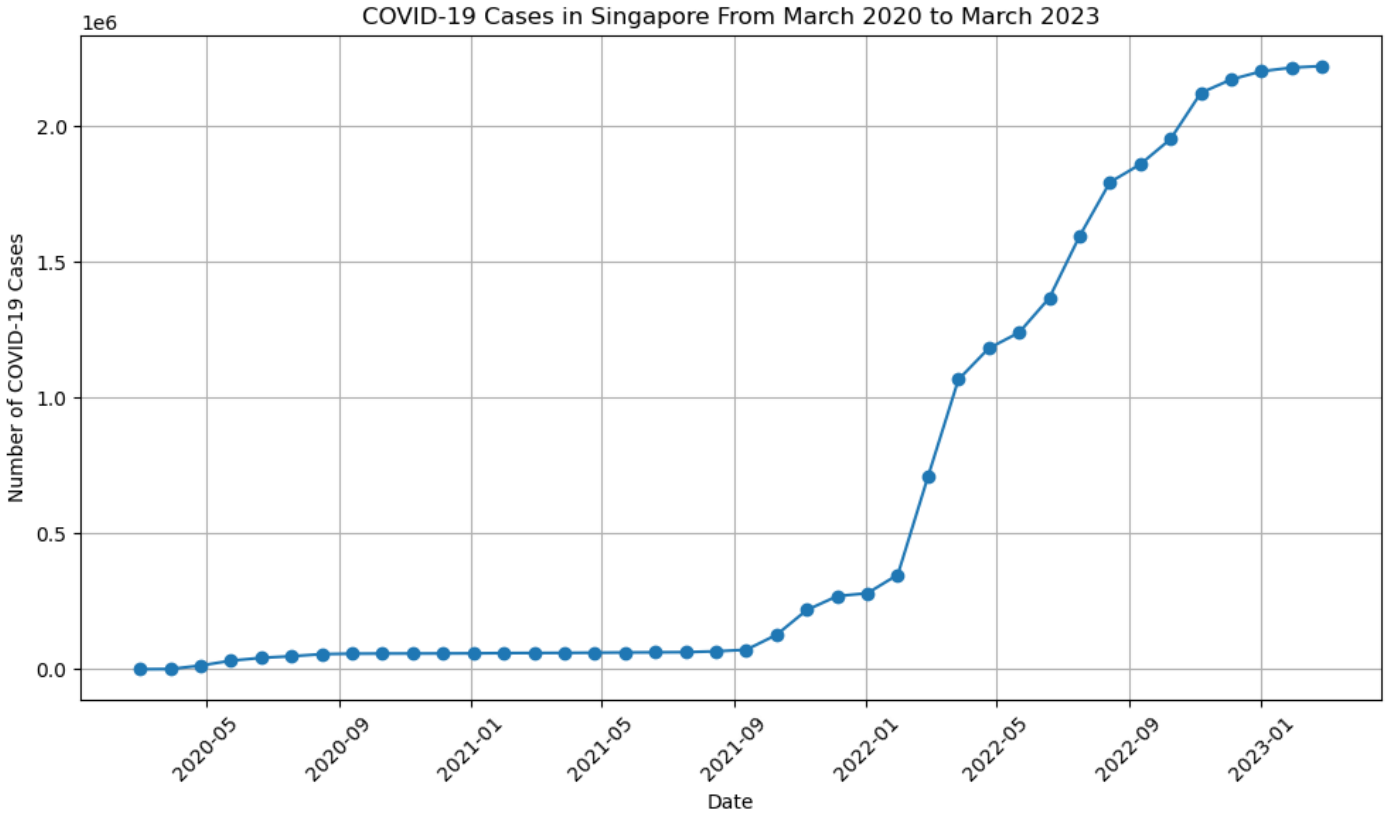
Section 3 – Cloud Infrastructure (AWS)

Assumptions

- The existing image processing code is cloud-ready and can be run in a containerized or serverless state (AWS Lambda, ECS, EKS, EC2 instance)
- Web apps are assumed to be integrated with the AWS Ecosystem, which has them working with API Gateway and Amazon MSK
- The existing image processing code is able to output the metadata of the images for Kinesis firehose to ingest the data into the metadata S3 bucket
- Amazon Quicksight is sufficient as a Business Intelligence solution for the company's needs

Section 4: Charts and API

Section 4 – Charts and API



Section 4 – Charts and API


```
{
  "data": {
    "date": "2023-03-08",
    "last_update": "2023-03-09 04:20:56",
    "confirmed": 2235294,
    "confirmed_diff": 4426,
    "deaths": 1722,
    "deaths_diff": 0,
    "recovered": 0,
    "recovered_diff": 0,
    "active": 2233572,
    "active_diff": 4426,
    "fatality_rate": 0.0008
  }
}
```


Returned object from the API is of the above format, we are only interested in the ”date” and the “confirmed” fields to allow us to plot the number of COVID-19 cases over time

```
#Setting date range for data to be plotted
start_date = datetime(2020, 3, 1)
end_date = datetime(2023, 3, 1) #Data stops at 2023-03

dates = []
confirmed_cases = []

current_date = start_date
while current_date <= end_date:
    #Format date as string in YYYY-MM-DD format
    date_str = current_date.strftime('%Y-%m-%d')
    url = f"https://covid-api.com/api/reports/total?date={date_str}&iso=SGP"

    response = requests.get(url)
    #Status code 200 indicates a successful request
    if response.status_code == 200:
        json_data = response.json()
        #Checking that data was obtained from the request appropriately
        if "data" in json_data and json_data["data"]:
            confirmed = json_data["data"].get("confirmed", None)
            #Checks if the "confirmed" value is not None - whether there is a valid number of COVID-19 cases
            if confirmed is not None:
                dates.append(current_date)
                confirmed_cases.append(confirmed)
    else:
        print(f"Failed to retrieve data")

    current_date += timedelta(weeks=4)
```

Using requests.get() to and response.json() to get the json object and checking that the json object is not empty.

The date and confirmed numbers are appended to the dates and confirmed_cases lists which are then subsequently used to plot the chart, iterate through dates from start till end to get number of cases for each month.

Section 4 – Charts and API

```
#Create a DataFrame from the collected data
df = pd.DataFrame({'Date': dates, 'Confirmed': confirmed_cases})
df = df.sort_values('Date')

#Plot the data
plt.figure(figsize=(10, 6))
plt.plot(df['Date'], df['Confirmed'], marker='o', linestyle='-')
plt.title("COVID-19 Cases in Singapore From March 2020 to March 2023")
plt.xlabel("Date")
plt.ylabel("Number of COVID-19 Cases")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

A pandas dataframe is created from the dates and confirmed_cases data obtained from iterating through the dates in the previous code chunk.

Matpotlib is then used to plot the final chart of number of covid cases for each month from March 2020 to March 2023

Section 5: Machine Learning

Section 5 – Machine Learning

Given the requirements, we will not be using the 'persons' column as a feature for our model.

```
features = ["maint", "doors", "lug_boot", "safety", "class"]
X = df[features]
y = df["buying"]
```

0] ✓ 0.0s Python

Using only the Maintenance, Number of doors, Lug boot size, safety, and class value of the car as features for the model.

```
print(y.value_counts())
```

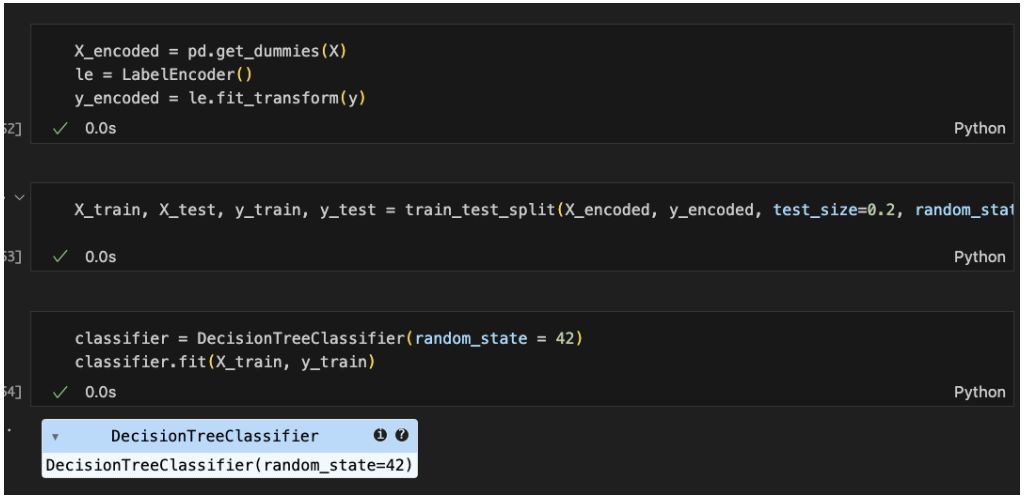
1] ✓ 0.0s Python

```
buying
vhigh    432
high     432
med      432
low      432
Name: count, dtype: int64
```

There are no class imbalances.

Checking if there is any class imbalance in the buying price of the car which could lead to poorer performance of the model.

Section 5 – Machine Learning



- Using get_dummies() from pandas to one-hot encode the features and LabelEncoder to encode the prediction class (buying price) to numeric values
- Use sklearn's train_test_split to split the data up with a 80/20 split
- Fitting the DecisionTreeClassifier from sklearn to the training data

Section 5 – Machine Learning

- Given the poor initial performance of the model, I will run hyperparameter tuning, more specifically GridSearch as I aim to improve the performance of the DecisionTreeClassifier

Section 5 – Machine Learning

```
param_grid = {
    "criterion": ["gini", "entropy", "log_loss"],
    "max_depth": [3, 5, 7, 9, 11, 13],
    "min_samples_split": [2, 5, 10, 20],
    "min_samples_leaf": [1, 2, 4, 8],
    "max_features": [None, "sqrt", "log2"]
}

from sklearn.model_selection import GridSearchCV

grid_search = GridSearchCV(
    estimator = classifier,
    param_grid = param_grid,
    scoring = 'accuracy',
    cv = 20,
    n_jobs =-1
)

grid_search.fit(X_train, y_train)

print(grid_search.best_params_)
print(grid_search.best_score_)

best_model = grid_search.best_estimator_
```

1] ✓ 14.0s Python

```
{'criterion': 'gini', 'max_depth': 5, 'max_features': None, 'min_samples_leaf': 8, 'min_samples_split':
0.31259834368530015
```

```
new_y_pred = best_model.predict(X_test)
print(classification_report(y_test, new_y_pred))
```

2] ✓ 0.0s Python

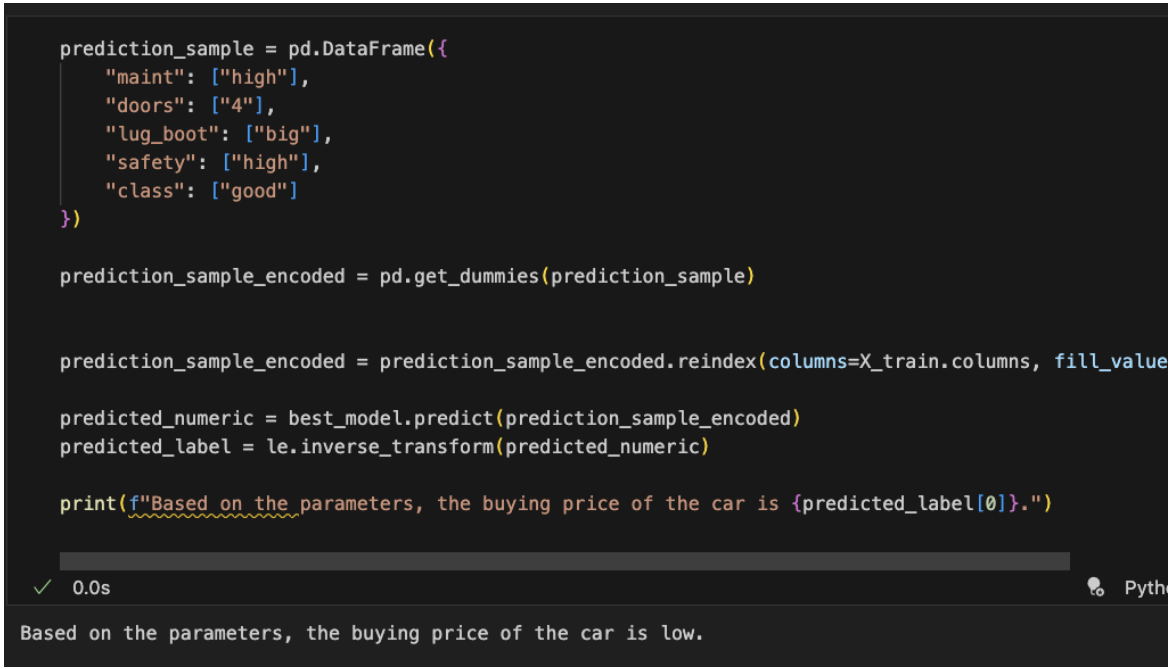
```
              precision    recall  f1-score   support

           0       0.29      0.46      0.35        92
           1       0.22      0.25      0.23        83
           2       0.19      0.19      0.19        77
           3       0.46      0.13      0.20        94

    accuracy                           0.26       346
   macro avg       0.29      0.26      0.25       346
weighted avg       0.30      0.26      0.25       346
```

- While the performance is still not ideal, there is an improvement after running GridSearch

Section 5 – Machine Learning



```
prediction_sample = pd.DataFrame({
    "maint": ["high"],
    "doors": ["4"],
    "lug_boot": ["big"],
    "safety": ["high"],
    "class": ["good"]
})

prediction_sample_encoded = pd.get_dummies(prediction_sample)


prediction_sample_encoded = prediction_sample_encoded.reindex(columns=X_train.columns, fill_value

predicted_numeric = best_model.predict(prediction_sample_encoded)
predicted_label = le.inverse_transform(predicted_numeric)

print(f"Based on the parameters, the buying price of the car is {predicted_label[0]}.")
```

✓ 0.0s

```
Based on the parameters, the buying price of the car is low.
```

- The final predicted buying price of the car based on the provided parameters is **low.**