Emirates Case Study

Marco Massetti

Scenario

Emirates Airline is a global carrier operating out of the UAE. The Airline sells three different seat types/products:

- **Economy Class**
  - Average Retail price: 4,000 AED
  - Operations Cost: 2,000 AED
- **Premium Economy Class**
  - Average Retail price: 5,000 AED
  - Cost: 2,500 AED
- **Business Class**
  - Retail price: 8,000 AED (increased to 9,000 AED from January 2019)
  - Cost: 4,000 AED

purchases.csv

```
customer_id,booking_date,product_booked,Booking_Price,Discount
1,01/12/2016,Premium_Economy,5000,190
4,01/12/2016,Premium_Economy,5000,450
5,01/12/2016,Business,8000,130
6,01/12/2016,Economy,4000,380
8,01/12/2016,Economy,4000,200
10,01/12/2016,Business,8000,580
```

leads.csv

```
customer_id,lead_date,product_searched,gender,nationality,province,dob
1,01/12/2016,Premium_Economy,Male,India,Dubai,31/03/1982
2,01/12/2016,Economy,Male,U.A.E.,Sharjah,19/05/1967
3,01/12/2016,Economy,Male,Pakistan,Sharjah,08/11/1958
4,01/12/2016,Premium_Economy,Male,India,Dubai,16/07/1970
5,01/12/2016,Business,Female,U.A.E.,Fujairah,05/10/1984
6,01/12/2016,Economy,Male,Sri Lanka,Abu Dhabi,16/02/1989
7,01/12/2016,Premium_Economy,Male,India,Dubai,03/06/1973
8,01/12/2016,Economy,Female,Algeria,Dubai,14/01/1985
9,01/12/2016,Premium_Economy,Male,India,Dubai,19/12/1985
10,01/12/2016,Business,Male,Sudan,Dubai,07/07/1967
```

Over the last three years, the company has seen a drop in sales and subsequently profit. To help combat the change, the company trialled two initiatives:

- From January 2018, the Airline decided to up the maximum potential discounts from 10% to 20%
- From January 2019, the Airline decided to increase the price of the Business Class seats from 8,000 to 9,000

It is unclear what impact each of these changes had on business performance.

You have been provided with two sets of data:

- **Leads (leads.csv)** - A list of enquiries and associated customer information
- **Purchases (purchases.csv)** - A list of purchases and their associated retail price and discount amount

Data Cleaning

In Python, an easy way to handle CSV is through Pandas.

Pandas has a lot of methods that can be applied to a Dataframe

(the logical version of the CSV or a table).

In our case, a combination of **isnull()** and a **sum** of the results

will give us the total number of missing values per column.

To backfill the null data some ad-hoc logic must be applied:

- **Gender**: Inferred from the mode or can be decided by the name*
- **Nationality** & **Province**: Inferred from the mode.
  Maybe can be fixed with the information in the ticket*
- **DOB**: Use a synthetic date from the median of the customer ages (*refer to a wider datasets)

```python
from datetime import datetime, timedelta
import pandas as pd
import numpy as np
df_purchases = pd.read_csv("purchases.csv")
df_leads = pd.read_csv("leads.csv")
df_joined = df_leads.set_index("customer_id").join(df_purchases.set_index("customer_id"),
on="customer_id", how="left", )
# Check nulls
null_purchase_values = df_purchases.isnull()
null_purchase_counts = null_purchase_values.sum()

print("Null values found in purchases:")
print(null_purchase_counts)

null_leads_values = df_leads.isnull()
null_leads_counts = null_leads_values.sum()

print("Null values found in leads:")
print(null_leads_counts)

## Fill missing gender
filling_gender = df_joined['gender'].mode()[0]
df_joined['gender'].fillna(filling_gender, inplace=True)

## Fill missing nationality and province
filling_nationality = df_joined['nationality'].mode()[0]
filling_province = df_joined['province'].mode()[0]
df_joined['nationality'].fillna(filling_nationality, inplace=True)
df_joined['province'].fillna(filling_province, inplace=True)
df_joined['dob'] = pd.to_datetime(df_joined['dob'], dayfirst=True, format="mixed")
df_joined.head()

# Fill missing dob
df_joined['today'] = datetime.today().strftime("%d/%m/%Y")
df_joined['today'] = pd.to_datetime(df_joined['today'], dayfirst=True, errors='ignore')

df_joined = df_joined.assign(Age=lambda x: round((x['today'] - x['dob']).dt.days))
filling_dob = (datetime.today() - timedelta(days=df_joined['Age'].median())).date()
df_joined['dob'].fillna(filling_dob, inplace=True)
df_joined.drop(columns=['today','Age'], inplace=True)
```

```
Most common gender: Male
Most common nationality: U.A.E.
Most common province: Dubai
Most common age: 44
DOB used: 1980-02-29
```

```
Null values found in leads:
customer_id             0
lead_date               0
product_searched        0
gender                 78
nationality            78
province             2489
dob                  4856
dtype: int64
```

Data Manipulation

Add a new column in the dataset that represent the duration between a lead to convert into a purchase.

↓

The approach is to calculate the difference between the booking date and the lead date.

```
from datetime import datetime
import pandas as pd
df_purchases = pd.read_csv("purchases.csv")
df_leads = pd.read_csv("leads.csv")

df_joined = df_leads.set_index("customer_id").join(df_purchases.set_index("customer_id"),
on="customer_id", how="left", )
df_joined.head()
df_joined['booking_date'] = pd.to_datetime(df_joined['booking_date'], dayfirst=True,
errors='ignore')
df_joined['lead_date'] = pd.to_datetime(df_joined['lead_date'], dayfirst=True,
errors='ignore')
df_joined_modified = df_joined.assign(ConvertedInDays=lambda x: (x['booking_date'] -
x['lead_date']).dt.days)

df_joined_modified[df_joined_modified['ConvertedInDays'] > 0].head()
```

nationality	province	dob	booking_date	product_booked	Booking_Price	Discount	ConvertedInDays
India	Dubai	23/10/1996	2018-10-15	Premium_Economy	5000.0	400.0	1.0
China	Dubai	NaN	2018-12-01	Premium_Economy	5000.0	970.0	1.0
India	Dubai	24/02/1969	2018-12-03	Economy	4000.0	680.0	1.0
U.A.E.	Ajman	08/08/1980	2018-12-05	Premium_Economy	5000.0	900.0	1.0
Croatia	Abu Dhabi	14/06/1989	2018-12-07	Premium_Economy	5000.0	480.0	1.0

```
Converted customers:
18
```

Data Transformation

- Create a flag for customers who searched for Economy Class but ended up booking premium economy or business class ticket (comparing searched and booked)
- Create a target variable called “Converted” for the analytics team
- Calculate the final booking price after applying the discount for each customer who booked a flight

```
from datetime import datetime
import pandas as pd
import numpy as np

df_purchases = pd.read_csv("purchases.csv")
df_leads = pd.read_csv("leads.csv")
df_joined = df_leads.set_index("customer_id").join(df_purchases.set_index("customer_id"),
on="customer_id", how="left", )

# Add flag for converted customers
df_joined['IsEconomy'] = df_joined['product_searched'].isin(['Economy'])
df_joined['BoughtPremiumOrBusiness'] =
df_joined['product_booked'].isin(['Premium_Economy','Business'])
df_joined = df_joined.assign(ConvertedFlag=lambda x: x['IsEconomy'] &
x['BoughtPremiumOrBusiness'])
#df_joined.assign(IsUpgraded=lambda x: True if x['product_searched'].str.contains("Economy")
else False)

# df_joined.head()
df_joined.drop(columns=['IsEconomy','BoughtPremiumOrBusiness'], inplace=True)
df_joined[df_joined['ConvertedFlag']].head()

# Add Converted target variables
df_joined['Converted'] = df_joined['ConvertedFlag']
df_joined = df_joined.assign(ConvertedFlag=lambda x: "Converted" if x['Converted'].bool else
None)

# Final booking price
df_joined["FinalPrice"] = df_joined["Booking_Price"] - df_joined["Discount"]
df_joined.head()

df_joined.to_csv("CSV/3_da
```

ice	Discount	ClassConvertedFlag	ClassConverted	ConvertedInDays	PurchaseConverted	Converted	FinalPrice
0.0	470.0	Converted	True	0.0	False	False	7530.0
0.0	120.0	Converted	True	0.0	False	False	4880.0
0.0	510.0	Converted	True	0.0	False	False	7490.0
0.0	830.0	Converted	True	0.0	False	False	7170.0
	20.0	Converted	True	0.0	False	False	4080.0

```
Customers whom upgraded from Economy to a superior class:
1301
```

Data Storage

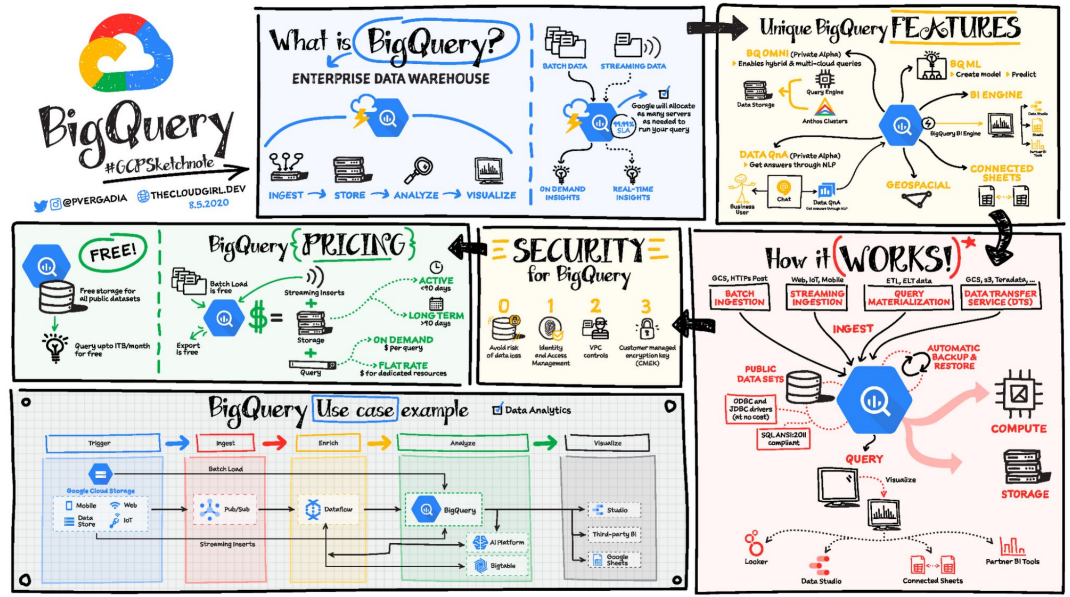
The best choice we have in GCP is BigQuery for analysis and querying of the data shown before. Scalability, speed, serveless are just few of the perks that the service can offer: integrated

with the Google Cloud Ecosystem,

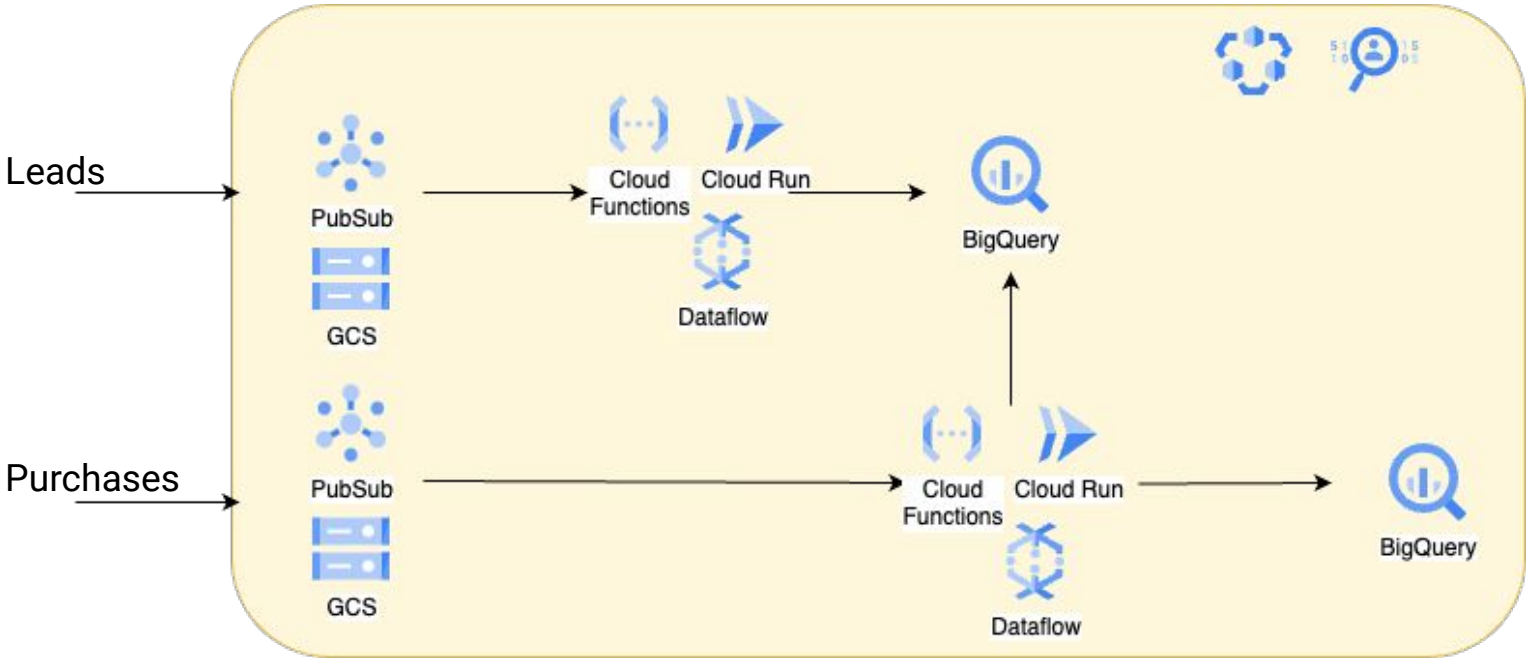
Data Warehousing capabilities,

ML integration, Data Federation

and security.



Data Pipeline



Reliability: Forced BigQuery schema and/or relying on Data Catalog to ensure data quality
Scalability: BigQuery is transparent regarding increasing size or schema, while data processing services can be scaled up as needed

Extra: Raw Json Event Transformation

```
Results:
Event Processed: 500
Event Accepted: 500
```

```
Table null values:
date                  0
device_category       0
country               0
event_name            0
item_name           464
item_id             464
page_title          466
content             347
trigger               0
content_type        464
type                 81
session_id            0
ga_session_number     6
skywards_tier       101
event_count           0
operating_system      0
downloads           494
dtype: int64
Check 'flat_table.csv' for final results
```

```
LOOKUP_TABLE = {
    "event_date": {"date": parser.parse},
    "device_category": {"device_category": str},
    "geo_country": {"country": str},
    "event_name": {"event_name": str},
    "item_name": {"item_name": str},
    "item_id": {"item_id": str},
    "screen_name": {"page_title": str},
    "firebase_screen": {"content": str},
    "firebase_event_origin": {"trigger": str},
    "content_type": {"content_type": str},
    "firebase_screen_class": {"type": str},
    "ga_session_id": {"session_id": int},
    "ga_session_number": {"ga_session_number": int},
    "skywards_tier": {"skywards_tier": str},
    "event_bundle_sequence_id": {"event_count": int},
    "device_operating_system": {"operating_system": str}
}
```

```
def flatten(root, nested_dict):
    new_dict = { f"{root}_{key}": nested_dict[key] for key in nested_dict.keys()}
    return new_dict

def extract(nested_list):
    new_dict = {elem['key']: [x for x in elem['value'].values() if x is not None][0]  for elem
in nested_list}
    return new_dict
```