Overview and Requirements

You are a data analyst working in the Data Operations and Quality team. You work regularly with business stakeholders and end-users to investigate issues, create new datasets, code review, and provide confidence in the data.

The data provided in the spreadsheet is for various bookings made in a day for some of the products that the company offers. Please analyse the dataset provided and answer the questions below.

Definitions of the columns have been provided to help support your analysis. Feel free to make any assumptions necessary (you can list any assumptions made).

1. What are your thoughts on the structure of the data?
2. Can you perform some basic data integrity checks on the data? What are your findings? (please make a note of all the checks you carried out) (Note: please only pick a few to carry out and list all the others that you would perform if you had more time)
3. What visualisations would you consider producing for end users and how would you ensure it is of value (no need to create the visualisation, just a description is fine or a rough mock-up)
4. SQL

a. Can you write a SQL statement that calculates the daily insurance attach rate to a car hire?

b. Can you write a SQL statement that would order or rank the data by countries generating the highest margin?

c. Can you write a SQL statement that displays the product(s) with the highest total % margin?

Below are some specific questions which will require you to analyse the data provided and answer at least 2 questions

1. Finance can't link some bookings references when they query the data. Can you highlight any reason why this may be happening and the impact it might have?
2. Our product teams believe that bookings are made equally throughout the day. What evidence can you find to either agree or disagree with the statement.
3. Our marketing team has created specific campaigns ahead of the football World Cup. Special discounts are offered for customers who want to rent in the Middle East. However the ROI (return of investment) report created recently suggests that these should be stopped since the bookings do not give us enough revenue to cover the cost of the campaign. Can you check to see if there is anything wrong with the data?
4. Our Car rental suppliers in Oceania have been complaining about delays with the payments. The cost of bookings represents what suppliers are owed and they state that they are missing payments on some bookings Is there anything wrong with our financial data?

1) Let's import the data and explore

In [1]:
```
#Import relevant libraries
import pandas as pd
import numpy as np
import sqlite3
```

In [2]:
```
#Read the data
data = pd.read_excel('car_data.xlsx', sheet_name='Dataset')
```

In [3]:
```
#Top 5 view of the data
data.head()
```

Out[3]:

	booking_reference	version	product_type	transaction_type	value	value_change	currency	book_date	pic
0	612020763	2	CAR_HIRE	COST	173.980	173.980	AU	2021-04-01 01:14:17	
1	612020763	2	CAR_HIRE	MARGIN	39.285	39.285	AUD	2021-04-01 01:14:17	
2	612020763	2	CAR_HIRE	PAID_BY_CUSTOMER	213.265	213.265	AUD	2021-04-01 01:14:17	
3	612020763	2	CAR_HIRE	PRICE	213.265	213.265	AUD	2021-04-01 01:14:17	
4	612175585	2	CAR_HIRE	COST	222.670	222.670	AU	2021-04-01 04:07:47	

In [4]:
```
#Basic statistical view on the given data
data.describe(include='all').transpose()
```

```
C:\Users\anand\AppData\Local\Temp\ipykernel_21824\414256427.py:2: FutureWarning: Treatin
g datetime data as categorical rather than numeric in `.describe` is deprecated and will
be removed in a future version of pandas. Specify `datetime_is_numeric=True` to silence
this warning and adopt the future behavior now.
  data.describe(include='all').transpose()
```

Out[4]:

	count	unique	top	freq	first	last	mean	std	
booking_reference	17382.0	3199.0	770301455.0	18.0	NaT	NaT	NaN	NaN	N
version	17382.0	NaN	NaN	NaN	NaT	NaT	1.992981	0.436731	
product_type	17382	9	CAR_HIRE	14295	NaT	NaT	NaN	NaN	N
transaction_type	17382	9	MARGIN	4311	NaT	NaT	NaN	NaN	N
value	17382.0	NaN	NaN	NaN	NaT	NaT	700.992414	9026.724511	-18
value_change	17382.0	NaN	NaN	NaN	NaT	NaT	691.683771	9027.955629	-296
currency	17382	38	EUR	8898	NaT	NaT	NaN	NaN	N
book_date	17382	2688	2021-04-01 16:00:00	3683	2021-04-01 00:00:09	2021-04-28 02:51:34	NaN	NaN	N
pick_up_country	17382	88	Australia	3059	NaT	NaT	NaN	NaN	N
customer_origin	17382	71	Australia	3030	NaT	NaT	NaN	NaN	N
contract_type	17382	5	PAY_NOW_PRINCIPLE	11462	NaT	NaT	NaN	NaN	N

In [5]:
```
#Check on the individual data types
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17382 entries, 0 to 17381
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   booking_reference  17382 non-null  object
 1   version            17382 non-null  int64
 2   product_type       17382 non-null  object
 3   transaction_type   17382 non-null  object
 4   value              17382 non-null  float64
 5   value_change       17382 non-null  float64
 6   currency           17382 non-null  object
 7   book_date          17382 non-null  datetime64[ns]
 8   pick_up_country    17382 non-null  object
 9   customer_origin    17382 non-null  object
 10  contract_type      17382 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(7)
memory usage: 1.5+ MB
```

Thoughts on the Structure of the Data

The dataset from the 'Dataset' sheet is well-structured and organized, suitable for analysis. It includes transactional data related to car hire bookings, with columns for booking reference, transaction types, values, and associated metadata like currency and dates. The data types are appropriate for the content, with identifiers as integers, financial figures as floating points, and textual information as strings where as 'Definations' subsheet seems to contain column specifications or metadata.

2) Data Integrity Checks

In [6]:
```
# Checking for null values
null_values = data.isnull().sum()

# Checking for duplicate entries
duplicate_entries = data.duplicated().sum()

print('Null Values in Each Column:\
', null_values)
print('\
Number of Duplicate Entries:', duplicate_entries)
```

```
Null Values in Each Column: booking_reference    0
version              0
product_type         0
transaction_type     0
value                0
value_change         0
currency             0
book_date            0
pick_up_country      0
customer_origin      0
contract_type        0
dtype: int64
Number of Duplicate Entries: 21
```

Findings from Data Integrity Checks

Null Values: There are no null values in any of the columns, which is excellent as it indicates complete data entries for all the fields.

Duplicate Entries: There are 21 duplicate entries in the dataset. This could be an issue if these duplicates represent unintentional repetitions of the same transactions, potentially skewing analysis results.

3) Visualisations for End Users

As per the dataset, here are some visualizations suggestions that could be valuable:

Trend Analysis Over Time: Line graphs showing trends in bookings, costs, and margins over time. This would help in understanding seasonal variations and overall performance.

Geographical Distribution: A map visualization showing the volume of bookings by pick-up country. This could help in identifying key markets and areas for expansion or improvement.

Financial Metrics Breakdown: Bar charts or pie charts showing the distribution of total costs, margins, and payments by customer origin or product type. This would provide insights into profitability and customer payment behaviors. These visualizations would be designed with interactivity in mind, allowing users to filter by date ranges, countries, or other relevant dimensions to get tailored insights.

4) SQL Queries

Since this is a jupyter notebook, I will write and validate queries in python but will also give corresponding SQL prompts in the markdown that can be run in SQL console.

a. Daily Insurance Attach Rate:

```
SELECT book_date, COUNT(DISTINCT booking_reference) AS total_bookings,
  SUM(CASE WHEN product_type = 'INSURANCE' THEN 1 ELSE 0 END) AS
insurance_bookings,
  (SUM(CASE WHEN product_type = 'INSURANCE' THEN 1 ELSE 0 END) * 1.0 /
COUNT(DISTINCT booking_reference)) AS attach_rate
FROM bookings
GROUP BY book_date;
```

b. Order by Countries Generating Highest Margin:

```
SELECT pick_up_country, SUM(value) AS total_margin
FROM bookings
WHERE transaction_type = 'MARGIN'
GROUP BY pick_up_country
ORDER BY total_margin DESC;
```

c. Product with the Highest Total % Margin:

```
SELECT product_type, SUM(value) AS total_margin
FROM bookings
WHERE transaction_type = 'MARGIN'
GROUP BY product_type
ORDER BY total_margin DESC
LIMIT 1;
```

In [7]:
```
data.product_type.value_counts()
```

Out[7]:
```
CAR_HIRE           14295
```

```
INSURANCE           1623
RENTAL_COVER        1204
RENTAL_COVER_DP      210
EXTRAS_ADDDRV         21
EXTRAS_LEGO           15
EXTRAS_CHLDSEAT        6
EXTRAS_BABYSEAT        4
EXTRAS_GPS             4
Name: product_type, dtype: int64
```

In [8]:
```
data.booking_reference.value_counts()
```

Out[8]:
```
770301455    18
666720487    16
634475545    15
754553106    14
721818600    14
             ..
796182386     3
799686760     3
665219106     2
777847018     2
746118208     2
Name: booking_reference, Length: 3199, dtype: int64
```

In [9]:
```
data.transaction_type.value_counts()
```

Out[9]:
```
MARGIN                      4311
PRICE                       4096
COST                        3612
PAID_BY_CUSTOMER            3157
AFFILIATE_COMMISSION        1733
IPT                          452
PAID_BY_CREDIT_AFFILIATE      10
EXCESS                         8
COST_SUPPLIER_CHANGE           3
Name: transaction_type, dtype: int64
```

In [10]:
```
data.contract_type.value_counts()
```

Out[10]:
```
PAY_NOW_PRINCIPLE         11462
PAY_NOW_AGENT_RETAIL       4814
PAY_NOW_NET_RATE_AGENT      648
PAY_LOCAL                   374
#ERR_JAVA_LANG_UTIL          84
Name: contract_type, dtype: int64
```

In [11]:
```
#Corresponding python code for all the mentioned SQL query

a_df = data[["book_date", "booking_reference", "product_type"]]

#Grouping by book_date: The query groups the data by the book_date column.
#Counting total bookings: It counts the number of distinct booking references for each b
#Counting insurance bookings: It calculates the number of bookings where the product_typ
#Calculating attach rate: It calculates the attachment rate of insurance bookings for ea
#It multiplies the count of insurance bookings by 1.0 to ensure floating-point division,
#The attachment rate is the ratio of the number of insurance bookings to the total numbe

# Create SQLite database and insert data
conn = sqlite3.connect(':memory:')
a_df.to_sql('bookings', conn, index=False, if_exists='replace')

# Run the SQL query
query1 = '''
SELECT book_date,
       COUNT(DISTINCT booking_reference) AS total_bookings,
```

```
       SUM(CASE WHEN product_type = 'INSURANCE' THEN 1 ELSE 0 END) AS insurance_bookings
       (SUM(CASE WHEN product_type = 'INSURANCE' THEN 1 ELSE 0 END) * 1.0 / COUNT(DISTIN
FROM bookings
GROUP BY book_date;
'''

# Execute the query
sql_a = pd.read_sql_query(query1, conn)

print(sql_a)

conn.close()
```

```
                book_date  total_bookings  insurance_bookings  attach_rate
0     2021-04-01 00:00:09               1                   0          0.0
1     2021-04-01 00:00:27               1                   0          0.0
2     2021-04-01 00:01:18               1                   0          0.0
3     2021-04-01 00:01:29               1                   0          0.0
4     2021-04-01 00:01:33               1                   0          0.0
...                   ...             ...                 ...          ...
2683  2021-04-23 02:13:50               1                   0          0.0
2684  2021-04-23 02:13:57               1                   0          0.0
2685  2021-04-24 00:36:32               1                   0          0.0
2686  2021-04-24 02:17:02               1                   0          0.0
2687  2021-04-28 02:51:34               1                   0          0.0

[2688 rows x 4 columns]
```

In [12]:
```
#Unique counts for attach_rate
sql_a.attach_rate.value_counts()
```

Out[12]:
```
0.00000    2330
3.00000     179
4.00000     132
2.00000      29
1.50000       9
1.00000       7
0.65312       1
0.50000       1
Name: attach_rate, dtype: int64
```

In [13]:
```
#Show attach_rate > 0
a = sql_a.loc[sql_a['attach_rate']> 0]
a
```

Out[13]:

	book_date	total_bookings	insurance_bookings	attach_rate
36	2021-04-01 00:21:18	1	4	4.0
38	2021-04-01 00:22:26	1	4	4.0
45	2021-04-01 00:28:16	1	3	3.0
70	2021-04-01 00:40:48	1	4	4.0
78	2021-04-01 00:42:45	1	4	4.0
...	...	...	...	...
2635	2021-04-04 18:22:47	1	3	3.0
2665	2021-04-08 18:35:56	1	3	3.0
2670	2021-04-10 11:41:19	1	4	4.0
2672	2021-04-11 11:28:38	1	3	3.0
2675	2021-04-15 08:11:22	1	3	3.0

358 rows × 4 columns

In [14]:
```
b_df = data[["pick_up_country", "value", "transaction_type"]]

#Filtering by transaction type: It selects only those records where the transaction_type
#Grouping by pick_up_country: It groups the filtered data by the pick_up_country column.
#Calculating total margin: It calculates the total margin for each pick_up_country, whic
#Ordering by total margin: It orders the result by the total margin in descending order.

# Create SQLite database and insert data
conn = sqlite3.connect(':memory:')
b_df.to_sql('bookings', conn, index=False, if_exists='replace')

# Run the SQL query
query2 = '''
SELECT pick_up_country, SUM(value) AS total_margin
FROM bookings
WHERE transaction_type = 'MARGIN'
GROUP BY pick_up_country
ORDER BY total_margin DESC;
'''

# Execute the query
sql_b = pd.read_sql_query(query2, conn)

print(sql_b)

conn.close()
```

```
   pick_up_country  total_margin
0            Japan    162002.000
1            Chile    105547.660
2           Russia     67735.570
3           Poland     66011.270
4     South Africa     43302.325
..             ...           ...
83          Latvia         6.105
84            Oman       -14.595
85         Bahrain       -33.860
86           Qatar       -39.060
87    Saudi Arabia     -1132.225

[88 rows x 2 columns]
```

In [15]:
```
c_df = data[["product_type", "value", "transaction_type"]]

#Filtering by transaction type: It selects only those records where the transaction_type
#Grouping by product type: It groups the filtered data by the product_type column.
#Calculating total margin: It calculates the total margin for each product_type, which i
#Ordering by total margin: It orders the result by the total margin in descending order.
#Limiting the result to one row: It limits the result to only the first row, which will

# Create SQLite database and insert data
conn = sqlite3.connect(':memory:')
c_df.to_sql('bookings', conn, index=False, if_exists='replace')

# Run the SQL query
query3 = '''
SELECT product_type, SUM(value) AS total_margin
FROM bookings
WHERE transaction_type = 'MARGIN'
GROUP BY product_type
ORDER BY total_margin DESC
LIMIT 1;
'''
```

```
# Execute the query
sql_c = pd.read_sql_query(query3, conn)

print(sql_c)

conn.close()
```

```
  product_type  total_margin
0     CAR_HIRE     410491.48
```

5) Finance Issue with Booking References

The presence of duplicate entries could be a reason why finance teams are unable to link some booking references. If duplicates represent different versions or updates of the same booking, it might create confusion or mismatches in financial records. This could impact financial reporting and reconciliation processes, leading to potential inaccuracies in financial statements or operational inefficiencies.

6) Analyzed Hourly Booking Distribution

From the data below, it's evident that bookings are not made equally throughout the day. There are significant variations, with peak booking hours in the morning (around 9-11 AM) and late afternoon (around 4 PM). This suggests that the product team's belief that bookings are made equally throughout the day does not hold true according to the data.

In [16]:
```
# Convert book_date to datetime and extract the hour to analyze booking distribution ove
data['book_date'] = pd.to_datetime(data['book_date'])
data['booking_hour'] = data['book_date'].dt.hour
```

In [17]:
```
# Count the number of bookings per hour
df_hourly_distribution = data['booking_hour'].value_counts().sort_index()

# Display the hourly booking distribution
df_hourly_distribution
```

Out[17]:
```
0      578
1      478
2      454
3      407
4      378
5      447
6      378
7      708
8      865
9      952
10     919
11     961
16    4633
17     838
18     858
19     746
20     718
21     839
22     630
23     595
Name: booking_hour, dtype: int64
```

7) Middle Eastern Country Analysis w.r.t ROI

In [18]:
```
sorted(data.pick_up_country.value_counts().index)
```

```
['Australia',
```

Out[18]:
```
'Austria',
'Bahrain',
'Belgium',
'Bolivia',
'Bonaire',
'Bosnia',
'Brazil',
'Bulgaria',
'Canada',
'Chile',
'Colombia',
'Costa Rica',
'Croatia',
'Curacao',
'Cyprus',
'Czech Republic',
'Denmark',
'Dominican Republic',
'Ecuador',
'Egypt',
'El Salvador',
'Finland',
'France - Corsica',
'France - Mainland',
'Georgia',
'Germany',
'Greece',
'Guam',
'Guatemala',
'Holland',
'Honduras',
'Hungary',
'Iceland',
'Ireland',
'Israel',
'Italy',
'Italy - Sardinia',
'Italy - Sicily',
'Jamaica',
'Japan',
'Jordan',
'Kosovo',
'Latvia',
'Lebanon',
'Lithuania',
'Luxembourg',
'Macedonia',
'Madeira',
'Malaysia',
'Malta',
'Martinique',
'Mexico',
'Montenegro',
'Morocco',
'Namibia',
'New Zealand',
'Nicaragua',
'Norway',
'Oman',
'Panama',
'Peru',
'Poland',
'Portugal',
'Portugal - Azores',
'Portugal - Madeira',
'Puerto Rico',
```

```
 'Qatar',
 'Romania',
 'Russia',
 'Saudi Arabia',
 'Serbia',
 'Seychelles',
 'South Africa',
 'South Korea',
 'Spain - Balearic Islands',
 'Spain - Canary Islands',
 'Spain - Mainland',
 'St. Barthelemy',
 'St. Maarten',
 'St. Martin',
 'St. Thomas',
 'Sweden',
 'Switzerland',
 'Thailand',
 'Tunisia',
 'Turkey',
 'U.A.E.']
```

In [19]:
```
data['booking_year'] = data['book_date'].dt.year
data['booking_year'].value_counts()
```

Out[19]:
```
2021    17382
Name: booking_year, dtype: int64
```

Kindly note that the data provided here is of the year 2021 and FIFA World Cup was held in 2022. Besides there are no features that explicitly tells about the campaign cost or any relevant data which creates a blockage to calculate it. Still, if we want to calculate ROI, considering features like 'value' and 'value_change', we observe that the total amount for the transaction type remains constant across all versions of bookings for that middle_eastern_countries. In other words, the financial transactions associated with the bookings have not been modified or adjusted over time for them.

In [20]:
```
## A/T Wikipedia, mentioned is the list of middle_eastern_countries
#
#middle_eastern_countries = ['Akrotiri and Dhekelia','Bahrain','Cyprus','Egypt','Iran','
# 'Lebanon','Oman','Palestine','Qatar','Saudi Arabia','Syria','Turkey','U.A.E','Yemen']
#df_middle_east = data[(data['pick_up_country'].isin(middle_eastern_countries))]
#
## Filter data for the World Cup campaign in the Middle East
#campaign_data = data[(data['customer_origin'].isin(middle_eastern_countries)) &
#                     (data['pick_up_country'].isin(middle_eastern_countries))]
#
##print(campaign_data)
## Calculate total revenue generated
#total_revenue = campaign_data['value'].sum()
#
## Calculate total cost of the campaign
#total_cost = campaign_data['value_change'].sum()
#
## Calculate ROI
#roi = total_revenue - total_cost
#
## Print results
#print("Total Revenue from the World Cup campaign in the Middle East:", total_revenue)
#print("Total Cost of the campaign:", total_cost)
#print("ROI (Return on Investment):", roi)
#
```

8)

In [21]:
```
#oceania_countries=['Australia','Papua New Guinea','New Zealand','Fiji','Solomon Islands
#'Kiribati','Tonga','Marshall Islands','Palau','Nauru','Tuvalu']
#
#df_oceania_countries = data[(data['pick_up_country'].isin(oceania_countries))]
#
## Assuming payment delays can be checked by comparing booking dates with payment dates
#payment_delays = df_oceania_countries[['booking_reference', 'book_date']]
#payment_delays = payment_delays.groupby('booking_reference').agg(
#    min_booking_date=('book_date', 'min'),max_booking_date=('book_date', 'max')
#)
#payment_delays = payment_delays.loc[payment_delays['min_booking_date'] != payment_delay
#payment_delays.reset_index(inplace=True)
#print("Bookings with potential payment delays:")
#print(payment_delays)
```

In [ ]:
```
!jupyter nbconvert --to webpdf --allow-chromium-download Untitled.ipynb
```