KPMG

July 27, 2020

[1]:
```
import pandas as pd
import datetime as DT
import numpy as np
```

1 Transactions

Upon loading the first sheet of our dataset (Transactions) and reading its head, we see that the first row is just a notice. Hence, we skip it while reading. Then, we check its shape to see the number of rows (transaction entries) and columns (features).

Next, we check the number of null values in all columns. We observe that the column online_order has 360 null values. In order to ensure data completness and quality, we can drop the rows with these null values. Depending on how we want to use our data later, we can also filter out rows on the basis of whether the order was online or not.

Similarly, we drop all rows that have null values for the brand name. After this, when we check null values again, we see that all rows with any null values have been dropped.

Finally, we change the product_first_sold_date from a vague string to a date format.

[2]:
```
df = pd.read_excel("KPMG_VI_New_raw_data_update_final.xlsx", sheet_name=1,
 ↪skiprows=1)

rows = df.shape[0]
cols = df.shape[1]
print("Number of Customer Id Entries: ", rows-1)
print("Number of Features: ", cols, "\n" )

# counting null values
print("Number of Initial Null Values: ")
print(df.isnull().sum(axis = 0),"\n")
# print(df.head(3))

# filtering out null values & offline orders
df = df[df['online_order'].notnull()]
df = df[df['brand'].notnull()]
# print(df.isnull().sum(axis = 0))
```

```python
# converting string to date format
df['product_first_sold_date'] = pd.
 ↪to_datetime(df['product_first_sold_date'],unit='d',origin='1900-01-01')
df = df.reset_index(drop = True)

print(df.head(3))
```

```
Number of Customer Id Entries:  19999
Number of Features:  13

Number of Initial Null Values:
transaction_id               0
product_id                   0
customer_id                  0
transaction_date             0
online_order               360
order_status                 0
brand                      197
product_line               197
product_class              197
product_size               197
list_price                   0
standard_cost              197
product_first_sold_date    197
dtype: int64

   transaction_id  product_id  customer_id transaction_date  online_order  \
0               1           2         2950       2017-02-25           0.0
1               2           3         3120       2017-05-21           1.0
2               3          37          402       2017-10-16           0.0

  order_status          brand product_line product_class product_size  \
0     Approved          Solex     Standard        medium       medium
1     Approved  Trek Bicycles     Standard        medium        large
2     Approved     OHM Cycles     Standard           low       medium

   list_price  standard_cost product_first_sold_date
0       71.49          53.62              2012-12-04
1     2091.47         388.92              2014-03-05
2     1793.43         248.82              1999-07-22
```


1.1 Checking for Duplicates

In order to ensure uniquness of data, we must ensure that there are no unncessary duplicates. In this case, transaction ids should not be repeated over rows. Hence, we check for duplicate transaction ids and if any are found, we drop rows as required. Luckily, in this case, there are no duplicates.

[3]:
```python
print(len(df['transaction_id'])-len(df['transaction_id'].drop_duplicates()))
```

```
0
```

1.2 Net Profits

Since our ultimate goal is to help Sprocket Central Pty Ltd grow its business, we must consider their profits. For this, we can add a new column for net profits. This can help us achieve relevancy (data items with value meta data).

[4]:
```python
df['net_profit'] = df['list_price'] - df['standard_cost']
print(df.net_profit.head(3))
```

```
0      17.87
1    1702.55
2    1544.61
Name: net_profit, dtype: float64
```

Now that our dataset appears to be in a good shape, let's save our dataset into a dictionary that we will use later to generate an Excel file.

[5]:
```python
writer = pd.ExcelWriter('KPMG_TASK1.xlsx', engine='xlsxwriter')
dataFrames = {'Transactions': df}
```


2 Customer Demographics

Similar to the Transactions dataset, upon loading the third sheet (Customer Demographics), we see that the first row is just a notice. Hence, we skip it while reading it. The default columns seems quite absurd and useless. It's best to drop it in the early stage.

Then, we check its shape to see the number of rows and columns (features). Next, we check the number of null values in all columns. We observe that there are around 125 N/A values for the last name. However, since last names might not have any impact on our business strategy, we can choose to ignore that column. However, DOB is a valuable asset for us. Hence, we drop the rows which have null values for the DOB.

Similarly, job titles could also be useful for us. Hence, we drop the rows which have null values for the job title as well.

[6]:
```python
df = pd.read_excel("KPMG_VI_New_raw_data_update_final.xlsx", sheet_name=3,␣
 ↪skiprows=1)
df = df.drop(['default'], axis=1)

rows = df.shape[0]
cols = df.shape[1]
print("Number of Customer Id Entries: ", rows-1)
print("Number of Features: ", cols, "\n" )
```

```python
print("Number of Initial Null Values: ")
print(df.isnull().sum(axis = 0),"\n")
df = df[df['DOB'].notnull()]
df = df[df['job_title'].notnull()]

print(df.head())
```

```
Number of Customer Id Entries:  3999
Number of Features:  12

Number of Initial Null Values:
customer_id                              0
first_name                               0
last_name                              125
gender                                   0
past_3_years_bike_related_purchases      0
DOB                                     87
job_title                              506
job_industry_category                  656
wealth_segment                           0
deceased_indicator                       0
owns_car                                 0
tenure                                  87
dtype: int64

   customer_id      first_name  last_name  gender  \
0            1         Laraine  Medendorp       F
1            2             Eli    Bockman    Male
2            3           Arlin     Dearle    Male
4            5  Sheila-kathryn     Calton  Female
7            8             Rod      Inder    Male

   past_3_years_bike_related_purchases        DOB               job_title  \
0                                   93 1953-10-12     Executive Secretary
1                                   81 1980-12-16  Administrative Officer
2                                   61 1954-01-20      Recruiting Manager
4                                   56 1977-05-13           Senior Editor
7                                   31 1962-03-30         Media Manager I

  job_industry_category     wealth_segment deceased_indicator owns_car  tenure
0                Health      Mass Customer                  N      Yes    11.0
1    Financial Services      Mass Customer                  N      Yes    16.0
2              Property      Mass Customer                  N      Yes    15.0
4                   NaN  Affluent Customer                  N      Yes     8.0
7                   NaN      Mass Customer                  N       No     7.0
```

2.1 Contradiction in Genders

We notice that the values in the gender column are not consistent. By looking at the value counts, we observe that F and Female both represent the same thing. Similarly, M and Male represent the both thing. Additionally, terms U and Femal are also present in the dataset.

To fix this, we can replace Female/Femal with F and Male with M everywhere. Then, we can get rid of the row where gender is undefined (U).

[7]:
```
print(df.gender.value_counts(),"\n")
df["gender"].replace({"Female": "F", "Male": "M", "Femal": "F"}, inplace=True)
df = df[df['gender'] != "U"]
```

```
Female    1769
Male      1643
U            1
F            1
Femal        1
M            1
Name: gender, dtype: int64
```

2.2 Customer Ages

When preparing business and marketing strategies, targeting the right age bracket is quite essential. Hence, we need to know the exact ages of our customers. For this, we can add a new column for customer ages. We can calculate ages by subtracting the DOB from the current date. After this, the dataset seems to look good to go.

[8]:
```
now = pd.Timestamp('now')
df['age'] = (now - df['DOB']).astype('<m8[Y]')
df['age'] = df['age'].astype(np.int64)
print(df.head(3))
```

```
   customer_id first_name  last_name gender  \
0            1    Laraine  Medendorp      F
1            2        Eli    Bockman      M
2            3      Arlin     Dearle      M

   past_3_years_bike_related_purchases        DOB               job_title  \
0                                   93 1953-10-12     Executive Secretary
1                                   81 1980-12-16  Administrative Officer
2                                   61 1954-01-20      Recruiting Manager

  job_industry_category wealth_segment deceased_indicator owns_car  tenure  \
0                Health  Mass Customer                  N      Yes    11.0
1    Financial Services  Mass Customer                  N      Yes    16.0
2              Property  Mass Customer                  N      Yes    15.0
```

```
   age
0   66
1   39
2   66
```

[9]:
```
dataFrames['CustomerDemographic'] = df
```

3 Customer Addresses

Just like the previous two datasets, this one also has a notice in the first row. Hence, we skip it while loading our dataset. Then, we check its shape of our dataset to see the number of rows and columns (features).

Next, we check the number of null values in all columns. Luckily, there are none.

[10]:
```
df = pd.read_excel("KPMG_VI_New_raw_data_update_final.xlsx", sheet_name=4,␣
 ↪skiprows=1)

rows = df.shape[0]
cols = df.shape[1]
print("Number of Customer Id Entries: ", rows-1)
print("Number of Features: ", cols, "\n" )

print("Number of Initial Null Values: ")
print(df.isnull().sum(axis = 0),"\n")
print(df.head())
```

```
Number of Customer Id Entries:  3998
Number of Features:  6

Number of Initial Null Values:
customer_id           0
address               0
postcode              0
state                 0
country               0
property_valuation    0
dtype: int64

   customer_id             address  postcode            state    country  \
0            1   060 Morning Avenue      2016  New South Wales  Australia
1            2  6 Meadow Vale Court      2153  New South Wales  Australia
2            4   0 Holy Cross Court      4211              QLD  Australia
3            5  17979 Del Mar Point      2448  New South Wales  Australia
4            6     9 Oakridge Court      3216              VIC  Australia
```

```
   property_valuation
0                  10
1                  10
2                   9
3                   4
4                   9
```

3.1 Contradiction in States

Upon seeing the head of our dataset, we observe something strange. Some state names are written full while others are abbreviations. To investigate further, we can check the value counts. After checking the value counts, we realize that there's a contradiction in state names. For instance, New South Wales and NSW is the same but are written separately. Same is the case with Victoria and VIC. Hence, we make the necessary replacements.

[11]:
```python
print(df.state.value_counts(),"\n")
df["state"].replace({"New South Wales": "NSW", "Victoria": "VIC"}, inplace=True)
print(df.head())
```

```
NSW                  2054
VIC                   939
QLD                   838
New South Wales        86
Victoria               82
Name: state, dtype: int64

   customer_id               address  postcode state    country  \
0            1    060 Morning Avenue      2016   NSW  Australia
1            2  6 Meadow Vale Court      2153   NSW  Australia
2            4    0 Holy Cross Court      4211   QLD  Australia
3            5  17979 Del Mar Point      2448   NSW  Australia
4            6      9 Oakridge Court      3216   VIC  Australia

   property_valuation
0                  10
1                  10
2                   9
3                   4
4                   9
```


Everything appears to be better now. As the final step, let's merge the datasets into a single Excel sheet.

[12]:
```python
dataFrames['CustomerAddress'] = df
for sheet, frame in dataFrames.items():
    frame.to_excel(writer, sheet_name = sheet, index=False)
writer.save()
```