DSC680_CPJP_Data_Preprocessing

January 8, 2022

1 Import Packages

[18]:
```
import pandas as pd
from pprint import pprint
import numpy as np
pd.options.mode.chained_assignment = None
from datetime import date
import datetime as dt
```

2 Data preparation

[19]:
```
col_names=['transaction_dt', 'customer_id', 'age_group', 'pin_code',␣
 ↪'product_subclass', 'product_id','amount','asset','sales_price']
```

Import Dataset

[20]:
```
#Import data using ISO-8859-1 seems to work, seperation is semi-colon
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.colheader_justify', 'center')
pd.set_option('display.precision', 3)

# November'2020 data import

K2_nov=pd.read_csv('C:\BU\DSC680\project1\data\D11',sep=';',names=col_names,␣
 ↪encoding = 'ISO-8859-1',low_memory=False)
# the first row in the raw data that is garbage
K2_nov=K2_nov.drop(K2_nov.index[0])
#modify dates column to datetime
K2_nov['transaction_dt'] = pd.to_datetime(K2_nov['transaction_dt'])
#remove empty spaces from values from the columns
K2_nov['age_group'] = K2_nov['age_group'].str.strip()
K2_nov['pin_code'] = K2_nov['pin_code'].str.strip()
K2_nov['customer_id'] = K2_nov['customer_id'].str.strip()
print ("\n\n*********"+"\033[1m"+" November 2000 data"+␣
 ↪"\033[0m"+"**************** \n\n")
```

```
display (K2_nov.head(3))

# December'2020 data import
K2_dec=pd.read_csv('C:\BU\DSC680\project1\data\D12',sep=';',names=col_names,␣
 ↪encoding = 'ISO-8859-1',low_memory=False)
K2_dec=K2_dec.drop(K2_dec.index[0])
K2_dec['transaction_dt'] = pd.to_datetime(K2_dec['transaction_dt'])
#remove empty spaces from values from the columns
K2_dec['age_group'] = K2_dec['age_group'].str.strip()
K2_dec['pin_code'] = K2_dec['pin_code'].str.strip()
K2_dec['customer_id'] = K2_dec['customer_id'].str.strip()
print ("\n\n*********"+"\033[1m"+" December 2000 data"+␣
 ↪"\033[0m"+"**************** \n\n")
display( K2_dec.head(3))

# January'2021 data import
jan_2k1=pd.read_csv('C:\BU\DSC680\project1\data\D01',sep=';',names=col_names,␣
 ↪encoding = 'ISO-8859-1',low_memory=False)
jan_2k1=jan_2k1.drop(jan_2k1.index[0])
jan_2k1['transaction_dt'] = pd.to_datetime(jan_2k1['transaction_dt'])
#remove empty spaces from values from the columns
jan_2k1['age_group'] = jan_2k1['age_group'].str.strip()
jan_2k1['pin_code'] = jan_2k1['pin_code'].str.strip()
jan_2k1['customer_id'] = jan_2k1['customer_id'].str.strip()
print ("\n\n*********"+"\033[1m"+" January 2021 data"+␣
 ↪"\033[0m"+"**************** \n\n")
display (jan_2k1.head(3))

# feb'2021 data import
feb_2k1=pd.read_csv('C:\BU\DSC680\project1\data\D02',sep=';',names=col_names,␣
 ↪encoding = 'ISO-8859-1',low_memory=False)
feb_2k1=feb_2k1.drop(feb_2k1.index[0])
feb_2k1['transaction_dt'] = pd.to_datetime(feb_2k1['transaction_dt'])
#remove empty spaces from values from the columns
feb_2k1['age_group'] = feb_2k1['age_group'].str.strip()
feb_2k1['pin_code'] = feb_2k1['pin_code'].str.strip()
feb_2k1['customer_id'] = feb_2k1['customer_id'].str.strip()
print ("\n\n*********"+"\033[1m"+" Feb 2021 data"+ "\033[0m"+"****************␣
 ↪\n\n")
display (feb_2k1.head(3))
```

```
********* November 2000 data****************
```

```
  transaction_dt customer_id age_group pin_code product_subclass   product_id  ␣
 ↪amount asset sales_price
1   2000-11-01     00046855       D        E         110411       4710085120468 ␣
 ↪  3      51       57
2   2000-11-01     00539166       E        E         130315       4714981010038 ␣
 ↪  2      56       48
3   2000-11-01     00663373       F        E         110217       4710265847666 ␣
 ↪  1     180      135
```

********* December 2000 data****************

```
  transaction_dt customer_id age_group pin_code product_subclass   product_id  ␣
 ↪amount asset sales_price
1   2000-12-01     00207423       C        E         530101       4710054134403 ␣
 ↪  1     92       99
2   2000-12-01     00329002       F        E         590514       4710049000973 ␣
 ↪  1     41       49
3   2000-12-01     01657951       E        E         120103       4710011401135 ␣
 ↪  1     23       29
```

********* January 2021 data****************

```
  transaction_dt customer_id age_group pin_code product_subclass   product_id  ␣
 ↪amount asset sales_price
1   2001-01-01     00141833       F        F         130207       4710105011011 ␣
 ↪  2      44       52
2   2001-01-01     01376753       E        E         110217       4710265849066 ␣
 ↪  1     150      129
3   2001-01-01     01603071       E        G         100201       4712019100607 ␣
 ↪  1      35       39
```

********* Feb 2021 data****************

```
  transaction_dt customer_id age_group pin_code product_subclass   product_id  ␣
 ↪amount asset sales_price
1   2001-02-01     00557818       H        E         500210       4710114105046 ␣
 ↪  1     123      135
```

```
2   2001-02-01      01677683        C       B       711310      4902520163103
 ↪ 6     840     894
3   2001-02-01      01900910        A       D       500206      4710036003598
 ↪ 1     26      33
```

3 Label Assignment and Merging datasets

[21]:
```
#Age Grouping labels
age_dict_class = {'A':'<25', 'B':'25-29', 'C':'30-34', 'D':'35-39', 'E':
 ↪'40-44', \
        'F':'45-49', 'G':'50-54', 'H':'55-59', 'I':'60-64', 'J':'+65', 'K':'NA'}

inv_age_dict_class = {v: k for k, v in age_dict_class.items()}

#Age Grouping labels into integer values
age_dict_int = {'A':1, 'B':2, 'C':3, 'D':4, 'E':5, \
        'F':6, 'G':7, 'H':8, 'I':9, 'J':10, 'K':11}

inv_age_dict_int = {v: k for k, v in age_dict_int.items()}

#pin code values into integers
pin_code_dict_int = {'A':1, 'B':2, 'C':3, 'D':4, 'E':5, \
        'F':6, 'G':7, 'H':8}

# Merge Nov2000,Dec 2000,Jan 2021,Feb 2021 Datasets to create a single dataset
cust_data=K2_nov.append(K2_dec).append(jan_2k1).append(feb_2k1)
#Apply value labels for dataframe
cust_data['age_label'] = cust_data['age_group'].apply(lambda x:
 ↪age_dict_class[x])
cust_data['age_int'] = cust_data['age_group'].apply(lambda x: age_dict_int[x])
cust_data['pin_code_int'] = cust_data['pin_code'].apply(lambda x:
 ↪pin_code_dict_int[x])

# Sort by Customer ID and Date
cust_data=cust_data.sort_values(by=['customer_id','transaction_dt'])
cust_data=cust_data.reset_index(drop=True)
cust_data.head(3)
```

[21]:
```
  transaction_dt customer_id age_group pin_code product_subclass   product_id
amount asset sales_price age_label  age_int  pin_code_int
0   2000-11-13     00001069       K        E         100314       4710176008699
1      78       98        NA       11           5
1   2000-11-13     00001069       K        E         100205       9556439880610
1      80       89        NA       11           5
2   2001-01-21     00001069       K        E         110333       4710320224661
1     361      425        NA       11           5
```

[22]:
```
display (cust_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 817741 entries, 0 to 817740
Data columns (total 12 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   transaction_dt    817741 non-null  datetime64[ns]
 1   customer_id       817741 non-null  object
 2   age_group         817741 non-null  object
 3   pin_code          817741 non-null  object
 4   product_subclass  817741 non-null  object
 5   product_id        817741 non-null  object
 6   amount            817741 non-null  object
 7   asset             817741 non-null  object
 8   sales_price       817741 non-null  object
 9   age_label         817741 non-null  object
 10  age_int           817741 non-null  int64
 11  pin_code_int      817741 non-null  int64
dtypes: datetime64[ns](1), int64(2), object(9)
memory usage: 74.9+ MB
```

```
None
```

[23]:
```
# below function will create dummy columns with suffix _A,_B .. and so on...
def suf_cols(df,column):
    temp_df=pd.DataFrame(df[column])
    for x in temp_df[column].unique():
        temp_df[str(column+'_')+x]=(temp_df[column]==x).astype(int)
    return temp_df

#Create dummy variables for age groupings
temp_age_group=suf_cols(cust_data,'age_group')
comparison=sorted(temp_age_group.columns)
temp_age_group=temp_age_group[comparison]
#concatenate the previously created dataframe with the created dummies
temp_age_group=pd.concat([cust_data, temp_age_group], axis=1)
```

[24]:
```
#Same as above but creating dummies of pin code
temp_pin_code=suf_cols(temp_age_group,'pin_code')
comparison=sorted(temp_pin_code.columns)
temp_pin_code=temp_pin_code[comparison]

temp_pin_code=pd.concat([temp_age_group, temp_pin_code], axis=1)
cust_data=temp_pin_code.drop(['age_group','pin_code'],axis=1)
cust_data.head(3).style.hide_index().set_caption('Total Raw Data w/ Dummies')
```

[24]: <pandas.io.formats.style.Styler at 0x1b0def4bb20>

Convert Columns to Integers to make modeling and plotting easier

[25]:
```
#convert variables to integers

cust_data['product_subclass']=cust_data.product_subclass.astype(np.int64)
cust_data['product_id']=cust_data.product_id.astype(np.int64)
cust_data['amount']=cust_data.amount.astype(np.int64)
cust_data['asset']=cust_data.asset.astype(np.int64)
cust_data['sales_price']=cust_data.sales_price.astype(np.int64)
# age map
age_map=cust_data.reset_index()
age_map=cust_data[['customer_id','age_int']]
age_map=dict(zip(list(age_map.customer_id),list(age_map.age_int)))

# pin code map
pin_map=cust_data.reset_index()
pin_map=cust_data[['customer_id','pin_code_int']]
pin_map=dict(zip(list(pin_map.customer_id),list(pin_map.pin_code_int)))

cust_data.head(3).style.hide_index().set_caption('Integer Modified Data')
```

[25]:
```
<pandas.io.formats.style.Styler at 0x1b0ded29a90>
```

[26]:
```
display (cust_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 817741 entries, 0 to 817740
Data columns (total 29 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   transaction_dt    817741 non-null  datetime64[ns]
 1   customer_id       817741 non-null  object
 2   product_subclass  817741 non-null  int64
 3   product_id        817741 non-null  int64
 4   amount            817741 non-null  int64
 5   asset             817741 non-null  int64
 6   sales_price       817741 non-null  int64
 7   age_label         817741 non-null  object
 8   age_int           817741 non-null  int64
 9   pin_code_int      817741 non-null  int64
 10  age_group_A       817741 non-null  int32
 11  age_group_B       817741 non-null  int32
 12  age_group_C       817741 non-null  int32
 13  age_group_D       817741 non-null  int32
 14  age_group_E       817741 non-null  int32
 15  age_group_F       817741 non-null  int32
 16  age_group_G       817741 non-null  int32
 17  age_group_H       817741 non-null  int32
```

```
 18  age_group_I       817741 non-null  int32
 19  age_group_J       817741 non-null  int32
 20  age_group_K       817741 non-null  int32
 21  pin_code_A        817741 non-null  int32
 22  pin_code_B        817741 non-null  int32
 23  pin_code_C        817741 non-null  int32
 24  pin_code_D        817741 non-null  int32
 25  pin_code_E        817741 non-null  int32
 26  pin_code_F        817741 non-null  int32
 27  pin_code_G        817741 non-null  int32
 28  pin_code_H        817741 non-null  int32
dtypes: datetime64[ns](1), int32(19), int64(7), object(2)
memory usage: 121.7+ MB
```

None

[27]:
```
cust_data_subset=cust_data[['transaction_dt','customer_id','age_int','product_subclass','produ
cust_data_subset=cust_data_subset.
 ↪sort_values(by='transaction_dt',ascending=True)
display(cust_data_subset.head().style.hide_index().set_caption('Customer␣
 ↪transactions subset data'))
```

<pandas.io.formats.style.Styler at 0x1b088a35070>

[28]:
```
#helper function for calculating the number of months from the first transaction
def get_date_int(df,column):
    '''
    Returns year,month,week, and day units.
    '''
    year=df[column].dt.year
    month=df[column].dt.month
    week=df[column].dt.isocalendar().week
    day=df[column].dt.day
    return year,month,week,day
```

[29]:
```
cust_data_subset['year'],cust_data_subset['month'],cust_data_subset['week'],cust_data_subset['
```

[30]:
```
cust_data_subset.to_csv(r'C:\BU\DSC680\project1\data\Final_Dataset.csv', index␣
 ↪= False)
```

[31]:
```
#Preprocess data for Recency, Frequency, Monetary (RFM) Segmentation
cust_data_subset['unit_price']=cust_data_subset['sales_price']/
 ↪cust_data_subset['amount']
rmf_cust_data_subset=cust_data_subset[['product_id','amount','transaction_dt','sales_price','c
rmf_cust_data_subset['total_sum']=rmf_cust_data_subset['unit_price']*cust_data_subset['amount'
```

[32]:
```
display(rmf_cust_data_subset.head().style.hide_index().set_caption('Customer␣
 ↪transactions subset data'))
```

```
<pandas.io.formats.style.Styler at 0x1b088b96370>
```

[33]:
```
rmf_cust_data_subset.head()
```

[33]:

	product_id	amount	transaction_dt	sales_price	customer_id	unit_price	total_sum
753166	4714981010038	1	2000-11-01	24	02101750	24.0	24.0
787527	4710088410382	1	2000-11-01	55	02144511	55.0	55.0
787526	37000445111	1	2000-11-01	47	02144511	47.0	47.0
787525	4711372660094	1	2000-11-01	76	02144511	76.0	76.0
787524	4710008290032	1	2000-11-01	57	02144511	57.0	57.0

[ ]:
```

```