DataMining_Assignment2

October 8, 2021

0.0.1 CISC 520-2021/Fall Data Engineering & Mining - Assignment 2

Student - Anh Hoang Chu

PART 1: CLEAN DATA

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
data = pd.read_excel('Online Retail.xlsx')
print(data.shape)
```

```
(541909, 8)
```

[3]:
```
print(data.columns)
```

```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
```

[4]:
```
dimensions = ['InvoiceNo', 'StockCode',␣
↪'Description','InvoiceDate','CustomerID', 'Country']
measures = ['Quantity', 'UnitPrice']
```

1. NAN CLEAN UP In below analysis, We observe that only category columns have NaN so we will remove rows that have NAs for the dataset

---

[5]:
```
data.isna().sum()
```

[5]:
```
InvoiceNo          0
StockCode          0
Description     1454
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID     135080
Country            0
dtype: int64
```

[6]:
```
data.head(100)
```

[6]:
```
    InvoiceNo StockCode                          Description  Quantity  \
0      536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1      536365     71053                  WHITE METAL LANTERN         6
2      536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3      536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4      536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6
..        …       …                                     …       …
95     536378     22352    LUNCH BOX WITH CUTLERY RETROSPOT          6
96     536378     21212      PACK OF 72 RETROSPOT CAKE CASES       120
97     536378     21975       PACK OF 60 DINOSAUR CAKE CASES        24
98     536378     21977   PACK OF 60 PINK PAISLEY CAKE CASES        24
99     536378     84991           60 TEATIME FAIRY CAKE CASES       24

             InvoiceDate  UnitPrice  CustomerID         Country
0   2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1   2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2   2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3   2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4   2010-12-01 08:26:00       3.39     17850.0  United Kingdom
..                  …        …         …              …
95  2010-12-01 09:37:00       2.55     14688.0  United Kingdom
96  2010-12-01 09:37:00       0.42     14688.0  United Kingdom
97  2010-12-01 09:37:00       0.55     14688.0  United Kingdom
98  2010-12-01 09:37:00       0.55     14688.0  United Kingdom
99  2010-12-01 09:37:00       0.55     14688.0  United Kingdom

[100 rows x 8 columns]
```

[7]:
```
def remove_nan(data):
    # 1. Fill in missing values
    print('**Identify missing data, counts and percentage**')
    print(data.isna().sum())
    print('\n',data.isna().sum()/len(data)*100)
    # Drop missing rows
    df = data.dropna()
    print('\n**Data after dropping missing values**')
    #print('Data Size:', df.shape)
    print(df.isna().sum())
    return df
```

[8]:
```
df = remove_nan(data)
df
```

```
**Identify missing data, counts and percentage**
InvoiceNo           0
StockCode           0
```

```
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64

 InvoiceNo       0.000000
StockCode        0.000000
Description      0.268311
Quantity         0.000000
InvoiceDate      0.000000
UnitPrice        0.000000
CustomerID      24.926694
Country          0.000000
dtype: float64

**Data after dropping missing values**
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

[8]:

	InvoiceNo	StockCode	Description	Quantity	\
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	
1	536365	71053	WHITE METAL LANTERN	6	
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	
…	…	…	…	…	
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	

	InvoiceDate	UnitPrice	CustomerID	Country
0	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

```
4      2010-12-01 08:26:00       3.39     17850.0  United Kingdom
…                      …        …          …               …
541904 2011-12-09 12:50:00       0.85     12680.0          France
541905 2011-12-09 12:50:00       2.10     12680.0          France
541906 2011-12-09 12:50:00       4.15     12680.0          France
541907 2011-12-09 12:50:00       4.15     12680.0          France
541908 2011-12-09 12:50:00       4.95     12680.0          France

[406829 rows x 8 columns]
```

2. SMOOTHING OF CONTINUOUS FIELDS Below codes will perform binning method to smooth data for Continuous fields such as Quantity and Unit Price

---

[9]:
```python
# Self-written code
def equal_width_binning(df, column, N):
    # Smoothing by bin means: each value in a bin is replaced by the mean value␣
 ↪of the bin.
    s = np.array(df[column])
    s.sort()
    #print(s)

    w = (s.max() - s.min())/N
    print(f'**Equal-width partition with {N} intervals with equal size of␣
 ↪{w}**')
    bins = []
    val = round(s.min(),0)
    while val <= s.max():
        bins.append(val)
        val = round(val + w)
    bins.append(s.max()+1)
    bins = np.array(bins)
    #print(bins)

    inds = np.digitize(s, bins, right=False)
    #print(inds)

    means_binning = pd.DataFrame({'Bin_Indx': np.array(inds), 'Val': s},␣
 ↪columns=['Bin_Indx', 'Val'])
    #print(means_binning.shape)

    #'Smoothing by Bin Means'
    means = means_binning.groupby('Bin_Indx').mean().reset_index()

    #print(means)
```

```
    #bin2 = means_binning.loc[means_binning['Bin_Indx'] == 2]
    means_binning = means_binning.merge(means, on='Bin_Indx', how='left')
    means_binning.rename(columns={'Val_x':'Val', 'Val_y':'Means'}, inplace=True)
    return means_binning
```

Binning for Unit Price

[10]:
```
print('Price range:', df['UnitPrice'].min(), df['UnitPrice'].max())
df.loc[df['UnitPrice'] == 0].head()
```

```
Price range: 0.0 38970.0
```

[10]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
9302	537197	22841	ROUND CAKE TIN VINTAGE GREEN	1	2010-12-05 14:02:00	0.0	12647.0	Germany
33576	539263	22580	ADVENT CALENDAR GINGHAM SACK	4	2010-12-16 14:36:00	0.0	16560.0	United Kingdom
40089	539722	22423	REGENCY CAKESTAND 3 TIER	10	2010-12-21 13:45:00	0.0	14911.0	EIRE
47068	540372	22090	PAPER BUNTING RETROSPOT	24	2011-01-06 16:41:00	0.0	13081.0	United Kingdom
47070	540372	22553	PLASTERS IN TIN SKULLS	24	2011-01-06 16:41:00	0.0	13081.0	United Kingdom

[11]:
```
ew_means_100 = equal_width_binning(df, 'UnitPrice', 100)
#ew_means.tail(10)
ew_means_group_100 = ew_means_100.groupby('Bin_Indx').count().reset_index()
ew_means_group_100.head(10)
```

Equal-width partition with 100 intervals with equal size of 389.7

[11]:

	Bin_Indx	Val	Means
0	1	406729	406729
1	2	44	44
2	3	16	16
3	4	9	9
4	5	10	10
5	6	5	5
6	7	3	3
7	8	1	1
8	9	2	2
9	11	6	6

[12]:
```
ew_means_1000 = equal_width_binning(df, 'UnitPrice', 1000)
#print(ew_means.tail(10))
```

```
ew_means_group_1000 = ew_means_1000.groupby('Bin_Indx').count().reset_index()
ew_means_group_1000.head(10)
```

Equal-width partition with 1000 intervals with equal size of 38.97

[12]:

	Bin_Indx	Val	Means
0	1	405985	405985
1	2	499	499
2	3	54	54
3	4	93	93
4	5	33	33
5	6	21	21
6	7	10	10
7	8	21	21
8	9	8	8
9	10	5	5

[13]:
```
# Equal Width Binning - built in function
print('Bins Interval: 100')
UnitPrice_ew_binning = pd.cut(df['UnitPrice'], bins = 100)
UnitPrice_ew_binning.value_counts().sort_index()
```

```
Bins Interval: 100
```

[13]:
```
(-38.97, 389.7]       406729
(389.7, 779.4]            44
(779.4, 1169.1]           16
(1169.1, 1558.8]           9
(1558.8, 1948.5]          10
                          ...
(37021.5, 37411.2]         0
(37411.2, 37800.9]         0
(37800.9, 38190.6]         0
(38190.6, 38580.3]         0
(38580.3, 38970.0]         1
Name: UnitPrice, Length: 100, dtype: int64
```

[14]:
```
print('Bins Interval: 1000')
UnitPrice_ew_binning = pd.cut(df['UnitPrice'], bins = 1000)
UnitPrice_ew_binning.value_counts().sort_index()
```

```
Bins Interval: 1000
```

[14]:
```
(-38.97, 38.97]         405985
(38.97, 77.94]             499
(77.94, 116.91]             54
(116.91, 155.88]            93
(155.88, 194.85]            33
                           ...
```

```
(38775.15, 38814.12]         0
(38814.12, 38853.09]         0
(38853.09, 38892.06]         0
(38892.06, 38931.03]         0
(38931.03, 38970.0]          1
Name: UnitPrice, Length: 1000, dtype: int64
```

—- OBSERVATION OF equal-width binning AND means smoothing method for UNIT PRICE —-

- Using equal-width binning (N=100), we can see most of the data (406729 records) stay in the bin with width in range (-38.97, 38.97] , which means most records in our dataset has UnitPrice < 390
- If we increase the interval to 1000, most of the data (405985 records) stay in the first bin with width in range (-38.97, 38.97] , which means most records in our dataset has UnitPrice < 39.
- We also see outliers in the bin with range (38931.03, 38970.0]

Binning for Quantity

[15]:
```
print('Quantity range:', df['Quantity'].min(), df['Quantity'].max())
print('Negative quantity record count:', len(df.loc[df['Quantity']<0]))
df.loc[df['Quantity'].isin([-80995,80995])]
```

```
Quantity range: -80995 80995
Negative quantity record count: 8905
```

[15]:
```
       InvoiceNo StockCode                  Description  Quantity  \
540421    581483     23843  PAPER CRAFT , LITTLE BIRDIE     80995
540422   C581484     23843  PAPER CRAFT , LITTLE BIRDIE    -80995

                InvoiceDate  UnitPrice  CustomerID         Country
540421 2011-12-09 09:15:00       2.08     16446.0  United Kingdom
540422 2011-12-09 09:27:00       2.08     16446.0  United Kingdom
```

— From above, we notice negative values in Quantity field —

[16]:
```
# Equal Width Binning - built in function
print('Bins Interval: 100')
Quantity_ew_binning = pd.cut(df['Quantity'], bins = 10)
Quantity_ew_binning.value_counts().sort_index()
```

```
Bins Interval: 100
```

[16]:
```
(-81156.99, -64796.0]         2
(-64796.0, -48597.0]          0
(-48597.0, -32398.0]          0
(-32398.0, -16199.0]          0
(-16199.0, 0.0]            8903
(0.0, 16199.0]           397922
```

```
(16199.0, 32398.0]            0
(32398.0, 48597.0]            0
(48597.0, 64796.0]            0
(64796.0, 80995.0]            2
Name: Quantity, dtype: int64
```

[17]:
```
print('Bins Interval: 1000')
Quantity_ew_binning = pd.cut(df['Quantity'], bins = 1000)
Quantity_ew_binning.value_counts()
Quantity_ew_binning.value_counts()
```

```
Bins Interval: 1000
```

[17]:
```
(0.0, 161.99]              395385
(-161.99, 0.0]               8807
(161.99, 323.98]             1715
(323.98, 485.97]              385
(485.97, 647.96]              202
                              ...
(-22030.64, -21868.65]          0
(-21868.65, -21706.66]          0
(-21706.66, -21544.67]          0
(-21544.67, -21382.68]          0
(-79861.07, -79699.08]          0
Name: Quantity, Length: 1000, dtype: int64
```

—- OBSERVATION OF equal-width binning AND means smoothing method for QUANTITY —-

- Using equal-width binning (N=100), we can see most of the data (397922 records) stay in the bin with width in range (0.0, 16199.0] , which means most records in our dataset has Quantity $<$ 16199.
- If we increase the interval to 1000, most of the data (405985 records) stay in the bin with width in range (0.0, 161.99], which means most records in our dataset has Quantity $<$ 162
- We also notice outlier in range (-81156.99, -80833.01] and (80833.01, 80995.0]

3. CORRECT INCONSISTENT DATA In above analysis we see 8905 records have negative quantity and we see 2 records example with same entries except for InvoiceNo and InvoiceDate, this looks like a return when customer bought 80995 units and then return them.

[ ]: