Data Wrangling in Python

This is a notebook by Ogechi Anoliefo.

1. Importing the library used

In [1]:
```
#In this case, I needed/used only pandas
import pandas as pd
```

2. Importing the datasets and merging them

In [2]:
```
#importing the sales dataset for 2018
data1 = pd.read_csv("C:/Users/ETIABA CHAMBER'S/Documents/Data_Analytics/My_portfolio/Onlir
data1.head()
```

Out[2]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	Region
0	493410	TEST001	This is a test product.	5	1/4/2018 9:24	4.50	12346.0	United Kingdom	Europe
1	C493411	21539	RETRO SPOTS BUTTER DISH	-1	1/4/2018 9:43	4.25	14590.0	United Kingdom	Europe
2	493412	TEST001	This is a test product.	5	1/4/2018 9:53	4.50	12346.0	United Kingdom	Europe
3	493413	21724	PANDA AND BUNNIES STICKER SHEET	1	1/4/2018 9:54	0.85	NaN	United Kingdom	Europe
4	493413	84578	ELEPHANT TOY WITH BLUE T-SHIRT	1	1/4/2018 9:54	3.75	NaN	United Kingdom	Europe

In [3]:
```
#importing the sales dataset for 2019
data2 = pd.read_csv("C:/Users/ETIABA CHAMBER'S/Documents/Data_Analytics/My_portfolio/Onlir
data2.head()
```

Out[3]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Region
0	539993	22386	JUMBO BAG PINK POLKADOT	10	1/4/2019 10:00	1.95	13313.0	United Kingdom	Europe
1	539993	21499	BLUE POLKADOT WRAP	25	1/4/2019 10:00	0.42	13313.0	United Kingdom	Europe
2	539993	21498	RED RETROSPOT WRAP	25	1/4/2019 10:00	0.42	13313.0	United Kingdom	Europe
3	539993	22379	RECYCLING BAG RETROSPOT	5	1/4/2019 10:00	2.10	13313.0	United Kingdom	Europe
4	539993	20718	RED RETROSPOT SHOPPER BAG	10	1/4/2019 10:00	1.25	13313.0	United Kingdom	Europe

Having studied the datasets, I observed that both datasets had similar columns, but different column names. This could also be seen from the last two outputs above. So I renamed the columns in both datasets to have the same names.

In [4]:
```
#renaming data1 and data2 columns
data1.columns = ['Invoice_No', 'Stock_Code', 'Description', 'Quantity',
                 'Invoice_Date', 'Unit_Price', 'Customer_ID', 'Country', 'Region']
data2.columns = ['Invoice_No', 'Stock_Code', 'Description', 'Quantity',
                 'Invoice_Date', 'Unit_Price', 'Customer_ID', 'Country', 'Region']
```

In [5]:
```
#checking that the column names had been properly renamed
data1.info()
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 503277 entries, 0 to 503276
Data columns (total 9 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   Invoice_No    503277 non-null  object 
 1   Stock_Code    503277 non-null  object 
 2   Description   500396 non-null  object 
 3   Quantity      503277 non-null  int64  
 4   Invoice_Date  503277 non-null  object 
 5   Unit_Price    503277 non-null  float64
 6   Customer_ID   401010 non-null  float64
 7   Country       503277 non-null  object 
 8   Region        503277 non-null  object 
dtypes: float64(2), int64(1), object(6)
memory usage: 34.6+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 519386 entries, 0 to 519385
Data columns (total 9 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   Invoice_No    519386 non-null  object 
 1   Stock_Code    519386 non-null  object 
 2   Description   518039 non-null  object 
 3   Quantity      519386 non-null  int64  
 4   Invoice_Date  519386 non-null  object 
 5   Unit_Price    519386 non-null  float64
 6   Customer_ID   392028 non-null  float64
 7   Country       519386 non-null  object 
 8   Region        519386 non-null  object 
dtypes: float64(2), int64(1), object(6)
memory usage: 35.7+ MB
```

Then I combined the two datasets into one.

In [6]:
```
#combining both datasets
df = pd.concat([data1, data2], ignore_index = True)
```

In [7]:
```
#checking that the datasets were properly combined
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1022663 entries, 0 to 1022662
Data columns (total 9 columns):
 #   Column        Non-Null Count    Dtype  
---  ------        --------------    -----  
 0   Invoice_No    1022663 non-null  object 
 1   Stock_Code    1022663 non-null  object 
 2   Description   1018435 non-null  object 
 3   Quantity      1022663 non-null  int64  
 4   Invoice_Date  1022663 non-null  object 
 5   Unit_Price    1022663 non-null  float64
 6   Customer_ID   793038 non-null   float64
 7   Country       1022663 non-null  object 
 8   Region        1022663 non-null  object 
dtypes: float64(2), int64(1), object(6)
memory usage: 70.2+ MB
```

Out[7]:

	Invoice_No	Stock_Code	Description	Quantity	Invoice_Date	Unit_Price	Customer_ID	Country	Regio
0	493410	TEST001	This is a test product.	5	1/4/2018 9:24	4.50	12346.0	United Kingdom	Europ
1	C493411	21539	RETRO SPOTS BUTTER DISH	-1	1/4/2018 9:43	4.25	14590.0	United Kingdom	Europ
2	493412	TEST001	This is a test product.	5	1/4/2018 9:53	4.50	12346.0	United Kingdom	Europ
3	493413	21724	PANDA AND BUNNIES STICKER SHEET	1	1/4/2018 9:54	0.85	NaN	United Kingdom	Europ
4	493413	84578	ELEPHANT TOY WITH BLUE T-SHIRT	1	1/4/2018 9:54	3.75	NaN	United Kingdom	Europ

The combined dataset has 1,022,663 entries and 9 columns.
After successfully combining the datasets, it was time for me to clean the data.

3. Data Cleaning

~ Changing some column types

The first data cleaning step I took was to change some of the column types to the appropriate format. It can be seen from the last output that the Customer_ID column is stored as float type instead of object (string) type. I had to change this. Also, Invoice_Date column is stored as object type instead of datetime; I changed this too.

In [8]:
```
#changing Customer_ID column to object type
df.Customer_ID = df.Customer_ID.astype('Int32').astype(object)
```

In [9]:
```
#changing Invoice_Date column to datetime
df['Invoice_Date'] = pd.to_datetime(df['Invoice_Date'], infer_datetime_format=True)
```

In [10]:
```
#checking to see that my columns were properly changed
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1022663 entries, 0 to 1022662
Data columns (total 9 columns):
 #   Column        Non-Null Count    Dtype
---  ------        --------------    -----
 0   Invoice_No    1022663 non-null  object
 1   Stock_Code    1022663 non-null  object
 2   Description   1018435 non-null  object
 3   Quantity      1022663 non-null  int64
 4   Invoice_Date  1022663 non-null  datetime64[ns]
 5   Unit_Price    1022663 non-null  float64
 6   Customer_ID   793038 non-null   object
 7   Country       1022663 non-null  object
 8   Region        1022663 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(1), object(6)
memory usage: 70.2+ MB
```

~ Dropping bad data

In the dataset, there were test entries where IT staff made faux purchases to check that the web store was running smoothly. These faux purchases had the stock codes 'TEST001' or 'TEST002'. I counted the number of entries having such stock codes and dropped them.

In [11]:
```
#to get the number of test entries in the dataset
df['Stock_Code'].str.contains('TEST', na=False).sum()
```

Out[11]:
```
17
```

In [12]:
```
#the output above shows there are 17 test entries in the dataset
#drop the test entries
df = df[~df.Stock_Code.str.contains("TEST")]
df.reset_index(drop=True, inplace=True)
df.head()
```

Out[12]:

	Invoice_No	Stock_Code	Description	Quantity	Invoice_Date	Unit_Price	Customer_ID	Country	Regio
0	C493411	21539	RETRO SPOTS BUTTER DISH	-1	2018-01-04 09:43:00	4.25	14590	United Kingdom	Europ
1	493413	21724	PANDA AND BUNNIES STICKER SHEET	1	2018-01-04 09:54:00	0.85	<NA>	United Kingdom	Europ
2	493413	84578	ELEPHANT TOY WITH BLUE T-SHIRT	1	2018-01-04 09:54:00	3.75	<NA>	United Kingdom	Europ
3	493413	21723	ALPHABET HEARTS STICKER SHEET	1	2018-01-04 09:54:00	0.85	<NA>	United Kingdom	Europ
4	493414	21844	RETRO SPOT MUG	36	2018-01-04 10:28:00	2.55	14590	United Kingdom	Europ

In [13]:
```
#double-checking that all 17 test entries were successfully dropped
df['Stock_Code'].str.contains('TEST', na=False).sum()
```

Out[13]:
```
0
```

In [14]:
```
#checking that the number of rows left correspond
df.shape
```

Out[14]:
```
(1022646, 9)
```

The outputs above show that all 17 test entries were successfully dropped.

I also want to point out that in the dataset, there were entries that had negative number of quantities in the 'Quantity' column, with invoice numbers starting with 'C'. These were for cancelled orders. I did not take them out as they represented either orders that were cancelled before shipment, and money refunded to the client, or orders that were returned, so the negative quanities would cancel out where the actual orders were made.

~ Removing Irrelevant Columns

For my analysis I did not need the 'Description' column as I could easily refer to the items using their stock code. So I dropped the description column.

In [15]:
```
#dropping the description column and confirming it's been dropped
df.drop('Description', axis=1, inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1022646 entries, 0 to 1022645
Data columns (total 8 columns):
 #   Column        Non-Null Count    Dtype
---  ------        --------------    -----
 0   Invoice_No    1022646 non-null  object
 1   Stock_Code    1022646 non-null  object
 2   Quantity      1022646 non-null  int64
 3   Invoice_Date  1022646 non-null  datetime64[ns]
 4   Unit_Price    1022646 non-null  float64
 5   Customer_ID   793022 non-null   object
 6   Country       1022646 non-null  object
 7   Region        1022646 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(1), object(5)
memory usage: 62.4+ MB
```

~ Checking for missing values

In [16]:
```
#to get the number of missing values (if any) in each column
df.isnull().sum()
```

Out[16]:
```
Invoice_No           0
Stock_Code           0
Quantity             0
Invoice_Date         0
Unit_Price           0
Customer_ID     229624
Country              0
Region               0
dtype: int64
```

There were 229624 missing values, all from the 'Customer_ID' column. For the purpose of this analysis, I replaced the missing customer ids with 'Unknown'

In [17]:
```
#replace all 229624 missing values for Customer_ID
df.Customer_ID = df.Customer_ID.fillna('Unknown')
```

In [18]:
```
#double-checking that there are no more missing values
df.isnull().sum()
```

Out[18]:
```
Invoice_No      0
Stock_Code      0
Quantity        0
Invoice_Date    0
Unit_Price      0
Customer_ID     0
Country         0
Region          0
dtype: int64
```

~ Checking for, and removing duplicate entries

The nature of this dataset was such that all columns had valid duplicate entries, view the head below:

In [19]:
```
df.head(8)
```

Out[19]:

	Invoice_No	Stock_Code	Quantity	Invoice_Date	Unit_Price	Customer_ID	Country	Region
0	C493411	21539	-1	2018-01-04 09:43:00	4.25	14590	United Kingdom	Europe
1	493413	21724	1	2018-01-04 09:54:00	0.85	Unknown	United Kingdom	Europe
2	493413	84578	1	2018-01-04 09:54:00	3.75	Unknown	United Kingdom	Europe
3	493413	21723	1	2018-01-04 09:54:00	0.85	Unknown	United Kingdom	Europe
4	493414	21844	36	2018-01-04 10:28:00	2.55	14590	United Kingdom	Europe
5	493414	21533	12	2018-01-04 10:28:00	4.25	14590	United Kingdom	Europe
6	493414	37508	2	2018-01-04 10:28:00	2.55	14590	United Kingdom	Europe
7	493414	35001G	2	2018-01-04 10:28:00	4.25	14590	United Kingdom	Europe

However, the trick was that no two same combinations of Invoice_No. and Stock_Code could occur, beacuse the same invoice could not have more than one entry of the same stock code. If the same purchase had more than one item with a specific stock code, the "Quantity' should have increased, rather than have another entry of same stock code on same invoice. So to check for duplicate entries, I checked for duplicate combinations of Invoice_ID & Stock_Code.

In [20]:
```
#creating a subset of the dataframe to view the duplicate entries
df1 = df[df.duplicated(subset = ['Invoice_No', 'Stock_Code'], keep=False)]
df1.head()
```

Out[20]:

	Invoice_No	Stock_Code	Quantity	Invoice_Date	Unit_Price	Customer_ID	Country	Region
158	493435	21678	2	2018-01-04 12:57:00	0.85	13206	United Kingdom	Europe
171	493435	21678	2	2018-01-04 12:57:00	0.85	13206	United Kingdom	Europe
257	493442	20751	1	2018-01-04 13:36:00	2.10	13821	United Kingdom	Europe
258	493442	20751	1	2018-01-04 13:36:00	2.10	13821	United Kingdom	Europe
267	493442	20712	1	2018-01-04 13:36:00	1.95	13821	United Kingdom	Europe

In [21]:
```
#to get the number of duplicate entries
df.duplicated(subset = ['Invoice_No', 'Stock_Code']).sum()
```

Out[21]:
```
22963
```

In [22]:
```
#there were 22963 duplicate entries
#dropping them
df2 = df.drop_duplicates(subset = ['Invoice_No', 'Stock_Code'], keep = 'first').reset_inde
```

In [23]:
```
#checking that the duplicate entries were successfully dropped ie. that the number of rows
df2.shape
```

Out[23]:
```
(999683, 8)
```

999,683 entries left. The duplicate entries were successfully dropped.

4. Adding new columns

After the data cleaning step, I proceeded to add a few new columns in the datsset, which would help me have better descriptive analysis of the data. I added a new calculated column 'Item_Price_Total' to get the total amount for each item type on an invoice. I also added columns 'Month' and 'Year', where I extracted the month and year respectively from the Invoice_Date column.

In [24]:
```
#adding Item_Price_Total column
df2['Item_Price_Total'] =  df2['Unit_Price'] * df2['Quantity']

#adding 'Month' column
df2['Month'] = df2['Invoice_Date'].dt.strftime('%b')

#adding 'Year' column
df2['Year'] = df2['Invoice_Date'].dt.year

df2.info()
df2.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999683 entries, 0 to 999682
Data columns (total 11 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Invoice_No        999683 non-null  object
 1   Stock_Code        999683 non-null  object
 2   Quantity          999683 non-null  int64
 3   Invoice_Date      999683 non-null  datetime64[ns]
 4   Unit_Price        999683 non-null  float64
 5   Customer_ID       999683 non-null  object
 6   Country           999683 non-null  object
 7   Region            999683 non-null  object
 8   Item_Price_Total  999683 non-null  float64
 9   Month             999683 non-null  object
 10  Year              999683 non-null  int64
dtypes: datetime64[ns](1), float64(2), int64(2), object(6)
memory usage: 83.9+ MB
```

Out[24]:

	Invoice_No	Stock_Code	Quantity	Invoice_Date	Unit_Price	Customer_ID	Country	Region	Item_Price
0	C493411	21539	-1	2018-01-04 09:43:00	4.25	14590	United Kingdom	Europe	
1	493413	21724	1	2018-01-04 09:54:00	0.85	Unknown	United Kingdom	Europe	
2	493413	84578	1	2018-01-04 09:54:00	3.75	Unknown	United Kingdom	Europe	
3	493413	21723	1	2018-01-04 09:54:00	0.85	Unknown	United Kingdom	Europe	
4	493414	21844	36	2018-01-04 10:28:00	2.55	14590	United Kingdom	Europe	

5. Running some summaries for descriptive analysis

I did some initial summary of the data, to get some initial findings.

In [25]:
```
#calculating total order amount for each order, ie. grouping the entries by Invoice_No and
Total_Order_Amount = df2.groupby('Invoice_No')['Item_Price_Total'].sum()
Total_Order_Amount.head()
```

Out[25]:
```
Invoice_No
491148    3882.66
491149     289.92
491150     435.60
491151     661.16
491152     436.80
Name: Item_Price_Total, dtype: float64
```

In [26]:
```
#calculating annual customer spend for each customer
Annual_Customer_Spend = df2.groupby(['Year', 'Customer_ID'])['Item_Price_Total'].sum()
Annual_Customer_Spend.head()
```

Out[26]:
```
Year  Customer_ID
2018  12346           59.68
      12347         1323.32
      12348          222.16
      12349         2671.14
      12351          300.93
Name: Item_Price_Total, dtype: float64
```

In [27]:
```
#calculating annual sales for each country
Annual_Sale_per_Country = df2.groupby(['Country', 'Year'])['Item_Price_Total'].sum()
Annual_Sale_per_Country.head(10)
```

Out[27]:
```
Country    Year
Australia  2018     30076.55
           2019    136359.42
Austria    2018     11585.20
           2019     10154.32
Bahrain    2018      2313.15
           2019       548.40
Belgium    2018     22204.93
           2019     40564.86
Bermuda    2018      1253.14
Brazil     2018       266.32
Name: Item_Price_Total, dtype: float64
```

In [28]:
```
#calculating annual sales for each region
Annual_Sale_per_Region = df2.groupby(['Region', 'Year'])['Item_Price_Total'].sum()
Annual_Sale_per_Region.head(10)
```

Out[28]:
```
Region        Year
Asia-Pacific  2018      50701.650
              2019     186426.990
Europe        2018    8991371.724
              2019    9098321.913
MEA           2018      16198.060
              2019      13175.860
The Americas  2018       7594.260
              2019       6540.900
Unspecified   2018       4937.530
              2019       4697.480
Name: Item_Price_Total, dtype: float64
```

In [29]:
```
#calculating total annual sales
Total_Annual_Sales = df2.groupby(['Year'])['Item_Price_Total'].sum()
Total_Annual_Sales
```

Out[29]:
```
Year
2018    9070803.224
2019    9309163.143
Name: Item_Price_Total, dtype: float64
```

6. Exporting my cleaned dataset

After I was done with the data wrangling process, I exported my cleaned dataset to CSV.

In [30]:
```
#write to CSV
df2.to_csv("C:/Users/ETIABA CHAMBER'S/Documents/Data_Analytics/My_portfolio/Online_store_s
```

Ps.: I could have continued with data visualization using Python, but I decided to use Power BI for data visualization and further analysis.