eda_online_retail

May 31, 2021

2.) Explanatory Data Analysis

As pointed out in the introduction of the project, the first task would be to perform an Explanatory Data Analysis of the data set along with Data Cleaning and Data Preparation. An investigation of all features is conducted leading to the final data set that is used in the second part of the project "Online Retail"

The first step would be to import the necessary Python Libraries used to tackle the project's tasks.

[1]:
```
# importing necessary Python libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import itertools
warnings.filterwarnings("ignore")
plt.style.use("fivethirtyeight")
import statsmodels.api as sm
import matplotlib
matplotlib.rcParams["axes.labelsize"] = 14
matplotlib.rcParams["xtick.labelsize"] = 12
matplotlib.rcParams["ytick.labelsize"] = 12
matplotlib.rcParams["text.color"] = "k"
import sklearn as sk
import fbprophet
from fbprophet import Prophet
from matplotlib import pylab
from pylab import *
from pylab import rcParams
```

After importing the necessary Libraries, the online-retail data set is loaded, stored and presented.

[2]:
```
# import the online_retail_data set
data = pd.read_csv("online_retail_II.csv")
# print the dataset
print(data)
```

```
        Invoice StockCode                          Description  Quantity  \
0        489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
```

```
1        489434   79323P                  PINK CHERRY LIGHTS        12
2        489434   79323W                 WHITE CHERRY LIGHTS        12
3        489434    22041         RECORD FRAME 7" SINGLE SIZE        48
4        489434    21232       STRAWBERRY CERAMIC TRINKET BOX       24
...         ...      ...                                 ...       ...
1067366  581587    22899         CHILDREN'S APRON DOLLY GIRL         6
1067367  581587    23254        CHILDRENS CUTLERY DOLLY GIRL         4
1067368  581587    23255     CHILDRENS CUTLERY CIRCUS PARADE         4
1067369  581587    22138        BAKING SET 9 PIECE RETROSPOT         3
1067370  581587     POST                             POSTAGE         1

                 InvoiceDate  Price  Customer ID         Country
0        2009-12-01 07:45:00   6.95      13085.0  United Kingdom
1        2009-12-01 07:45:00   6.75      13085.0  United Kingdom
2        2009-12-01 07:45:00   6.75      13085.0  United Kingdom
3        2009-12-01 07:45:00   2.10      13085.0  United Kingdom
4        2009-12-01 07:45:00   1.25      13085.0  United Kingdom
...                      ...    ...          ...             ...
1067366  2011-12-09 12:50:00   2.10      12680.0          France
1067367  2011-12-09 12:50:00   4.15      12680.0          France
1067368  2011-12-09 12:50:00   4.15      12680.0          France
1067369  2011-12-09 12:50:00   4.95      12680.0          France
1067370  2011-12-09 12:50:00  18.00      12680.0          France

[1067371 rows x 8 columns]
```

For convenience purposes, the variable “Customer ID” is renamed to “CustomerID”.

[3]:
```python
# rename the variable "Customer ID" to "CustomerID"
data.rename(columns={"Customer ID": "CustomerID"}, inplace=True)
```

[4]:
```python
# overview : head of the dataset
data.head()
```

[4]:
```
  Invoice StockCode                          Description  Quantity  \
0  489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1  489434    79323P                   PINK CHERRY LIGHTS        12
2  489434    79323W                  WHITE CHERRY LIGHTS        12
3  489434     22041         RECORD FRAME 7" SINGLE SIZE         48
4  489434     21232       STRAWBERRY CERAMIC TRINKET BOX        24

          InvoiceDate  Price  CustomerID         Country
0 2009-12-01 07:45:00   6.95     13085.0  United Kingdom
1 2009-12-01 07:45:00   6.75     13085.0  United Kingdom
2 2009-12-01 07:45:00   6.75     13085.0  United Kingdom
3 2009-12-01 07:45:00   2.10     13085.0  United Kingdom
4 2009-12-01 07:45:00   1.25     13085.0  United Kingdom
```

Having a look at the information of the data set, it can be seen that there are NULL values for features “Description” and “CustomerID”.

[5]:
```
# obtain information on the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1067371 entries, 0 to 1067370
Data columns (total 8 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   Invoice      1067371 non-null  object
 1   StockCode    1067371 non-null  object
 2   Description  1062989 non-null  object
 3   Quantity     1067371 non-null  int64
 4   InvoiceDate  1067371 non-null  object
 5   Price        1067371 non-null  float64
 6   CustomerID   824364 non-null   float64
 7   Country      1067371 non-null  object
dtypes: float64(2), int64(1), object(5)
memory usage: 65.1+ MB
```

With the following ‘unique_counts’ function, the number of unique values of each variable in the data set is obtained.

[6]:
```
# check unique values/counts for each column/variable
def unique_counts(df):
   for i in df.columns:
       count = df[i].nunique()
       print(i, ": ", count)
unique_counts(data)
```

```
Invoice :  53628
StockCode :  5305
Description :  5698
Quantity :  1057
InvoiceDate :  47635
Price :  2807
CustomerID :  5942
Country :  43
```

Before we proceed with the exploration of NULL values in the data, the duplicates(same observation entries) are removed from the data set.

[7]:
```
# drop duplicate rows/observations/entries
data = data.drop_duplicates()
```

Going back to the NULL values, 4.382 observations (0.41%) of “Description” and 243.007 observations (22.77%) of “CustomerID” are NULL entries.

[8]:
```python
# check for NULL values
missing_values = data.isnull().sum()
print (missing_values)
# check the percentage of NaN values for all variables
missing_percentage = data.isnull().sum() / data.shape[0] * 100
print(missing_percentage)
## 0.41% for "Description" and 22.77% for "Customer ID"
# total NaN entries
print(data.isnull().sum().sum())
```

```
Invoice             0
StockCode           0
Description      4275
Quantity            0
InvoiceDate         0
Price               0
CustomerID     235151
Country             0
dtype: int64
Invoice         0.000000
StockCode       0.000000
Description     0.413829
Quantity        0.000000
InvoiceDate     0.000000
Price           0.000000
CustomerID     22.763098
Country         0.000000
dtype: float64
239426
```

Having an insight at the first few NaN values in “Description”, it seems that “CustomerID” is also NaN, “Price” is 0 and “Quntity” has and negative values.

[9]:
```python
# an overview on the NaN "Description" in relation with other features
print(data[data.Description.isnull()].head())
```

```
     Invoice StockCode Description  Quantity          InvoiceDate  Price  \
470   489521     21646         NaN       -50  2009-12-01 11:44:00    0.0
3114  489655     20683         NaN       -44  2009-12-01 17:26:00    0.0
3161  489659     21350         NaN       230  2009-12-01 17:39:00    0.0
3731  489781     84292         NaN        17  2009-12-02 11:45:00    0.0
4296  489806     18010         NaN      -770  2009-12-02 12:42:00    0.0

      CustomerID          Country
470          NaN   United Kingdom
3114         NaN   United Kingdom
3161         NaN   United Kingdom
3731         NaN   United Kingdom
```

```
4296         NaN  United Kingdom
```

Let’s investigate this bit further. How often is “Price” 0 and how often is “CustomerID” missing when “Description” is missing?

[10]:
```
# how often is "Price" 0 when "Description" is missing ?
# what about "Quantity" when "Description" is missing ?
data.loc[data.Description.isnull(), ["Price", "Quantity"]].describe()
```

[10]:
```
        Price     Quantity
count  4275.0  4275.000000
mean      0.0   -17.366784
std       0.0   515.938734
min       0.0 -9600.000000
25%       0.0   -28.000000
50%       0.0    -4.000000
75%       0.0     3.000000
max       0.0  9600.000000
```

[11]:
```
# how often is "CustomerID" missing when "Description" is missing ?
print(data[data.Description.isnull()].CustomerID.isnull().value_counts())
```

```
True    4275
Name: CustomerID, dtype: int64
```

It can therefore be concluded that, when “Description” is missing, “Price” is always 0 and “CustomerID” is also NaN. As far as the feature “Quantity” is concerned, it mostly consists of negative values. It is also noticed that the min and max values are ±9.600, which is interesting and probably indicates a cancelled transaction. Considering the aim of the second part of the project(forecasting of aggregated sales), these observations are not of interest so they are removed from the set.

[12]:
```
data = data.dropna(subset=["Description", "CustomerID"])
print(data.head())
```

```
  Invoice StockCode                          Description  Quantity  \
0  489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1  489434    79323P                   PINK CHERRY LIGHTS        12
2  489434    79323W                  WHITE CHERRY LIGHTS        12
3  489434     22041         RECORD FRAME 7" SINGLE SIZE         48
4  489434     21232       STRAWBERRY CERAMIC TRINKET BOX        24

          InvoiceDate  Price  CustomerID         Country
0  2009-12-01 07:45:00   6.95     13085.0  United Kingdom
1  2009-12-01 07:45:00   6.75     13085.0  United Kingdom
2  2009-12-01 07:45:00   6.75     13085.0  United Kingdom
3  2009-12-01 07:45:00   2.10     13085.0  United Kingdom
4  2009-12-01 07:45:00   1.25     13085.0  United Kingdom
```

Before we move on to the exploration of othe features, it might be useful to investigate bit more the feature “Description”. Since it describes the product’s name (nominal variable), it might contain hidden NaN entries or empty entries or other forms.

[13]:
```
# investigate further the varibale "Description" for 'lowercase NaN' or 'hidden␣
↪NaN'
# introduce a new variable "Lowercase_Description"
data.loc[data.Description.isnull()==False, "Lowercase_Description"] = data.loc[
    data.Description.isnull()==False,"Description"].apply(lambda l: l.lower())

data.Lowercase_Description.dropna().apply(
    lambda l: np.where("nan" in l, True, False)).value_counts()
## 916 nan in "Lowercase_Description"
```

[13]:
```
False    796969
True        916
Name: Lowercase_Description, dtype: int64
```

[14]:
```
# investigate variable "Description" for empty "" strings
data.Lowercase_Description.dropna().apply(
    lambda l: np.where("" == l, True, False)).value_counts()
## no empty entries in "Lowercase_Description"
```

[14]:
```
False    797885
Name: Lowercase_Description, dtype: int64
```

There are eventually additional, hidden NaN entries (916) in a string "nan" form instead of a NaN-value that are going to be removed. In addition, no "empty" sting entries found in "Description".

[15]:
```
# transform the string "NaN" to "NaN" value
data.loc[data.Lowercase_Description.isnull()==False, "Lowercase_Description"] =␣
↪data.loc[
    data.Lowercase_Description.isnull()==False, "Lowercase_Description"].
↪apply(lambda l: np.where("nan" in l, None, l))
```

[16]:
```
# drop the rows/observations having "NaN" entries in "Description"
data = data.loc[(data.CustomerID.isnull()==False) & (data.Lowercase_Description.
↪isnull()==False)].copy()
# remove variable "Lowercase_Description" from the dataset
data = data.drop(["Lowercase_Description"], axis=1)
```

Make a final check for NaN values in the data set.

[17]:
```
# check again for NaN entries
data.isnull().sum().sum()
## no more NaN entries in the data set
```

[17]:
```
0
```

We are now ready to proceed. There were previously noticed negative values in "Quantity". For the purposes of this task, these values are not of interest. Let's see then if there are still negative values in the set after the removal of some observations.

[18]:
```
data[(data.Quantity<1)].value_counts().sum()
```

[18]:
```
18371
```

[19]:
```
# check for negative quanititities related to cancellations
data[(data.Quantity<1) & (~data.Invoice.str.startswith('C'))]
```

[19]:
```
Empty DataFrame
Columns: [Invoice, StockCode, Description, Quantity, InvoiceDate, Price,
CustomerID, Country]
Index: []
```

It seems that there are still 18.371 negative values (2.31%) in "Quantity" which are also associated with "Cancellations".

[20]:
```
# "Invoice" starting with "C" indicates cancellation
# investiagate these entries
data["Cancellation"]=np.where(data.Invoice.apply(lambda l: l[0]=="C"), True,␣
 ↪False)
print (data.Cancellation.value_counts())
print (data.Cancellation.value_counts() / data.shape[0] * 100)
```

```
False    778598
True      18371
Name: Cancellation, dtype: int64
False    97.694892
True      2.305108
Name: Cancellation, dtype: float64
```

[21]:
```
# summary statistics about the cancelled transactions/"Invoice"
data.loc[data.Cancellation==True,["Quantity", "Price"]].describe()
```

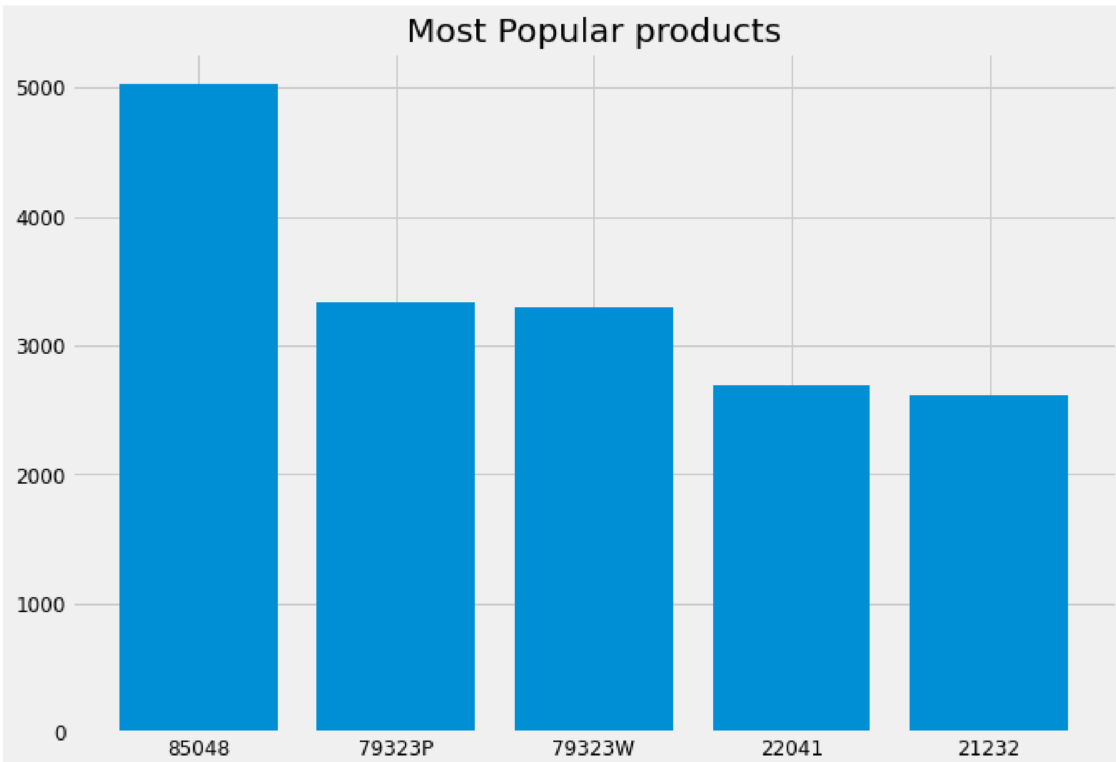
[21]:

	Quantity	Price
count	18371.000000	18371.000000
mean	-25.737902	24.263801
std	826.406149	428.462306
min	-80995.000000	0.010000
25%	-6.000000	1.450000
50%	-2.000000	2.950000
75%	-1.000000	5.950000
max	-1.000000	38970.000000

[22]:
```
# drop the cancelled transactions
data = data.loc[data.Cancellation==False].copy()
data = data.drop("Cancellation", axis=1)
```

As a reminder, the "StockCode" feature represents the product's code. Among all products in the data set, which are the most popular? The following Figure represents the 5 most popular product codes with more than 2.000 counts in the data set.

[23]:
```
# indicate the most popular products
StockCode = data["StockCode"]
Counts = data["StockCode"].value_counts().loc[lambda x : x >= 2000]
fig = plt.figure(figsize =(10, 7))
plt.bar(StockCode[0:5], Counts[0:5])
plt.title("Most Popular products")
plt.show()
```



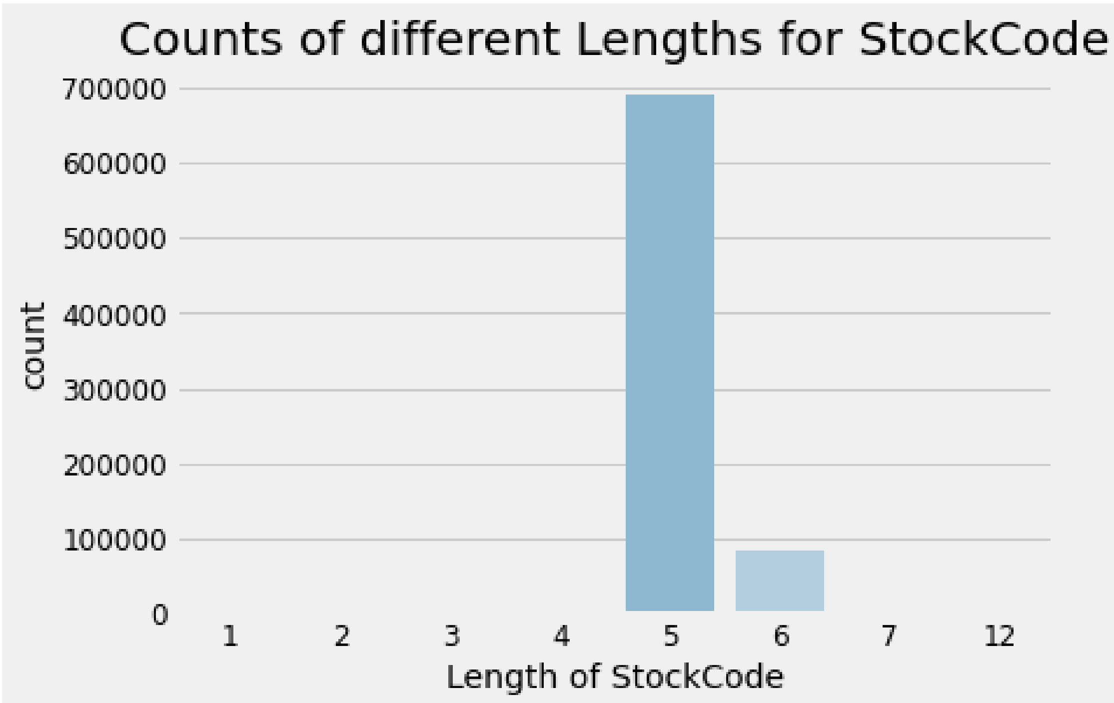
In the description of the data set, "StockCode" is a 5-digit integral number representing the product's code. Let us see if there are entries of "StockCode" of larger or smaller length.

[24]:
```
# check the length of the StockCode
def check_length_chars(l):
    return sum(1 for c in l if c.isdigit())

data["StockCodeLength"] = data.StockCode.apply(lambda l: len(l))
```

[25]:
```
fig, ax = plt.subplots()
sns.countplot(data["StockCodeLength"], palette="Blues_r", ax=ax)
ax.set_title("Counts of different Lengths for StockCode")
```

```
ax.set_xlabel("Length of StockCode")
```

[25]: Text(0.5, 0, 'Length of StockCode')



[26]:
```
# summary statistics of "StockCodeLength"
data["StockCodeLength"].describe()
```

[26]:
```
count    778598.000000
mean          5.105237
std           0.353487
min           1.000000
25%           5.000000
50%           5.000000
75%           5.000000
max          12.000000
Name: StockCodeLength, dtype: float64
```

The vast majority of "StockCode" is of length 5, even though there are entries ranging from 1 to 12. The second most prominent length is the one of 6 digits. Before we make any decision on "StockCode" feature, let's see what happens in the cases where length is 5.

[27]:
```
# investigate the "StockCode" with length = 6
data[(data.StockCode.str.len())==6].sort_values(by='StockCode').head(100)
```

[27]:
```
        Invoice StockCode          Description  Quantity  \
508389  536863    10123C  HEARTS WRAPPING TAPE         1
95127   498364    10123C  HEARTS WRAPPING TAPE        12
63607   495053    10123C  HEARTS WRAPPING TAPE        12
319911  520564    10123C  HEARTS WRAPPING TAPE         5
25602   491622    10123C  HEARTS WRAPPING TAPE         1
...        ...       ...                   ...       ...
276378  516263    15044A     PINK PAPER PARASOL         1
191895  507608    15044A     PINK PAPER PARASOL         6
233673  512046    15044A     PINK PAPER PARASOL         1
799682  560892    15044A     PINK PAPER PARASOL         6
254125  513938    15044A     PINK PAPER PARASOL         6

                InvoiceDate  Price  CustomerID         Country  \
508389  2010-12-03 11:19:00   0.65     17967.0  United Kingdom
95127   2010-02-18 13:50:00   0.65     16170.0  United Kingdom
63607   2010-01-20 14:36:00   0.65     17351.0  United Kingdom
319911  2010-08-26 17:11:00   0.65     17402.0  United Kingdom
25602   2009-12-11 14:20:00   0.65     13415.0  United Kingdom
...                     ...    ...         ...             ...
276378  2010-07-19 11:59:00   2.95     17841.0  United Kingdom
191895  2010-05-10 13:51:00   2.95     15141.0  United Kingdom
233673  2010-06-11 17:18:00   2.95     17091.0  United Kingdom
799682  2011-07-21 17:08:00   2.95     13089.0  United Kingdom
254125  2010-06-29 12:45:00   2.95     13089.0  United Kingdom

        StockCodeLength
508389                6
95127                 6
63607                 6
319911                6
25602                 6
...                 ...
276378                6
191895                6
233673                6
799682                6
254125                6

[100 rows x 9 columns]
```

[28]:
```python
# investigate the "StockCode" with length < 5
data[(data.StockCode.str.len())<5].sort_values(by='StockCode').head(1000)
```

[28]:
```
        Invoice StockCode Description  Quantity          InvoiceDate  Price  \
754378  556969        C2    CARRIAGE         1  2011-06-16 10:11:00   25.0
800004  560922        C2    CARRIAGE         1  2011-07-21 19:34:00   50.0
```

```
922706  571129      C2    CARRIAGE        1  2011-10-14 10:03:00   50.0
404559  528114      C2    CARRIAGE        1  2010-10-20 14:22:00   50.0
763043  557872      C2    CARRIAGE        1  2011-06-23 12:42:00   50.0
...        ...     ...         ...     ...                  ...    ...
706778  552464    POST     POSTAGE        1  2011-05-09 15:12:00   18.0
708598  552649    POST     POSTAGE        1  2011-05-10 13:44:00   18.0
818898  562600    POST     POSTAGE        1  2011-08-08 10:01:00   40.0
720418  553682    POST     POSTAGE        4  2011-05-18 13:20:00   15.0
825299  563172    POST     POSTAGE        1  2011-08-12 13:15:00   15.0

        CustomerID         Country  StockCodeLength
754378     16257.0  United Kingdom                2
800004     14911.0            EIRE                2
922706     14156.0            EIRE                2
404559     14911.0            EIRE                2
763043     14911.0            EIRE                2
...            ...             ...              ...
706778     12726.0          France                4
708598     12592.0         Germany                4
818898     12676.0          Sweden                4
720418     14646.0     Netherlands                4
825299     12399.0         Belgium                4

[1000 rows x 9 columns]
```

The "StockCodes" of length 6 do not seem problematic since the 6th digit, following the 5-digit number, is a capital letter indicating probably some sort of specific type of the product and therefore will not be discarded from the data set. On the other hand, the "StockCodes" with length < 5, seem to be related with some other parties or other additional costs (shipping charges,discounts,others). It is not clear if those observations are of interest. There should be some sort of separation between those different transactions. In the first place, the observations with lenght of "StockCode" < 5 will be removed.

[29]:
```python
# drop observations with "StockCode" length smaller than 5
data = data.drop(data[(data.StockCode.str.len())<5].index).reset_index(drop=True)
data = data.drop(["StockCodeLength"], axis=1)
```

As mentioned above, the "StockCodes" with length < 5 represent some sort of special transactions and their "Description" describes this special transaction. Thus, let's check the unique values of "StockCode" and "Description".

[30]:
```python
unique_counts(data[["StockCode","Description"]])
```

```
StockCode :  4623
Description :  5275
```

[31]:
```
print(data.groupby("StockCode").Description.nunique().
 ↪sort_values(ascending=False).iloc[0:10])
print(data.loc[data.StockCode == "22384"].Description.value_counts())
## product "22384" has "4" different entries
```

```
StockCode
22345    4
22346    4
23196    4
20685    4
22384    4
23236    4
21955    4
22344    4
21243    3
23126    3
Name: Description, dtype: int64
LUNCH BAG PINK POLKADOT     1116
LUNCH BAG PINK RETROSPOT     744
LUNCHBAG PINK RETROSPOT       13
LUNCH BAG PINK POLKADOTS       1
Name: Description, dtype: int64
```

It is noticed that there are more "Descriptions" than "StockCodes". Exploring this a bit more, we can see that a specific "StockCode" can be associated with more than one specific "Description". For instance, product "22384" is associated with 4 different descriptions. Having a closer look, this does not seem problematic since this can be due to either misspeling and connecting words or just a characteristic of the product.

Next, the duration of the data set is presented. The "InvoiceDate" is converted to a 'datetime64' object and "CustomerID" is converted to 'int64' object for better representation.

[32]:
```
# investigate time of the dataset
data["InvoiceDate"] = pd.to_datetime(data.InvoiceDate, cache=True)
# duration
data.InvoiceDate.max() - data.InvoiceDate.min()
# start and end point
print("Datafile starts with timepoint {}".format(data.InvoiceDate.min()))
print("Datafile ends with timepoint {}".format(data.InvoiceDate.max()))
```

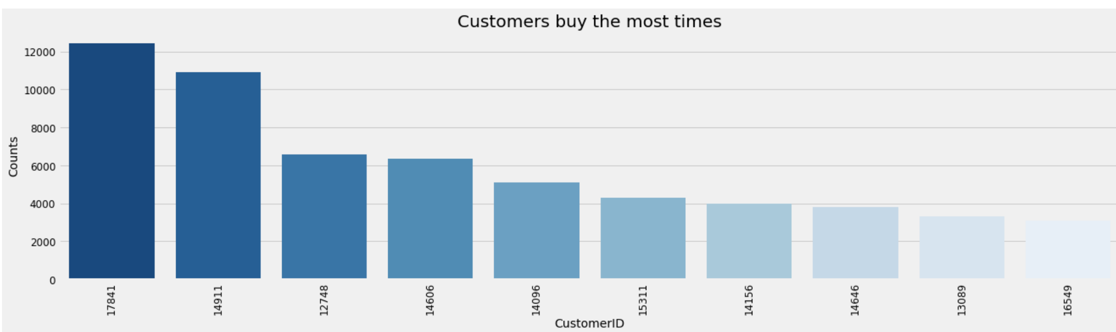
```
Datafile starts with timepoint 2009-12-01 07:45:00
Datafile ends with timepoint 2011-12-09 12:50:00
```

[33]:
```
# convert the type of variable "CustomerID" to integer
data["CustomerID"] = data["CustomerID"].astype("int64")
data.dtypes
```

[33]:
```
Invoice                object
StockCode              object
Description            object
Quantity                int64
InvoiceDate    datetime64[ns]
Price                 float64
CustomerID              int64
Country                object
dtype: object
```

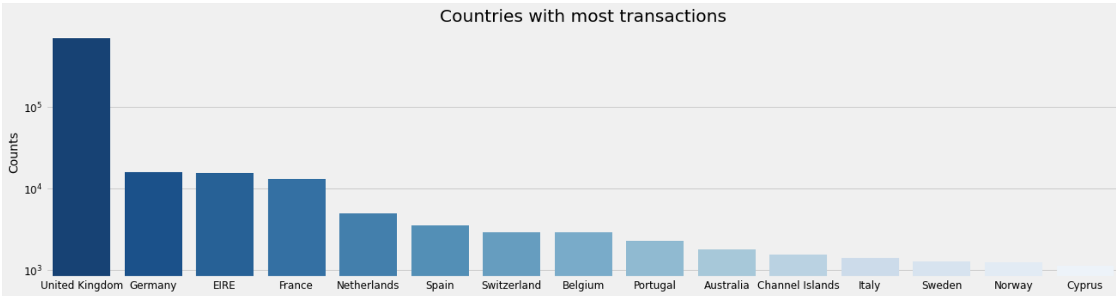
Previously, we had a look on the best products in terms of frequeny. Now, in the following plot, the top-10 customers in terms of requency are indicated.

[34]:
```
data.CustomerID.nunique()
customer_counts = data.CustomerID.value_counts().sort_values(ascending=False).
 ↪iloc[0:10]
plt.figure(figsize=(20,5))
sns.barplot(customer_counts.index, customer_counts.values, order=customer_counts.
 ↪index,palette="Blues_r")
plt.ylabel("Counts")
plt.xlabel("CustomerID")
plt.title("Customers buy the most times");
plt.xticks(rotation=90);
```



Something similar with respect to the countries of buyers.

[35]:
```
country_counts = data.Country.value_counts().sort_values(ascending=False).
 ↪loc[lambda x : x >= 1000]
plt.figure(figsize=(20,5))
sns.barplot(country_counts.index, country_counts.values, palette="Blues_r")
plt.ylabel("Counts")
plt.title("Countries with most transactions");
plt.yscale("log")
```



[36]:
```
# percentages of countries with most transactions
print(data.loc[data.Country=="United Kingdom"].shape[0] / data.shape[0] * 100)
print(data.loc[data.Country=="Germany"].shape[0] / data.shape[0] * 100)
print(data.loc[data.Country=="EIRE"].shape[0] / data.shape[0] * 100)
```

```
90.08030213193781
2.033074682271661
1.979325101183264
```

The three top countries with most transcactions, United Kingdom, Germany and Republic of Ireland consist of around 94% of the company's transactions. In terms of "Quantity" on average, northern European countries, such as Denmark, Netherlands and Sweden, buy the most.

[37]:
```
# countries with the higher numbers of quantities
data.groupby("Country")["Quantity"].mean().sort_values(ascending=False).iloc[0:
↪10]
```

[37]:
```
Country
Denmark           314.029101
Netherlands        77.093317
Sweden             69.817536
Japan              67.285408
Australia          58.270588
Thailand           33.578947
Czech Republic     27.916667
Singapore          21.045181
EIRE               20.706629
France             20.640927
Name: Quantity, dtype: float64
```

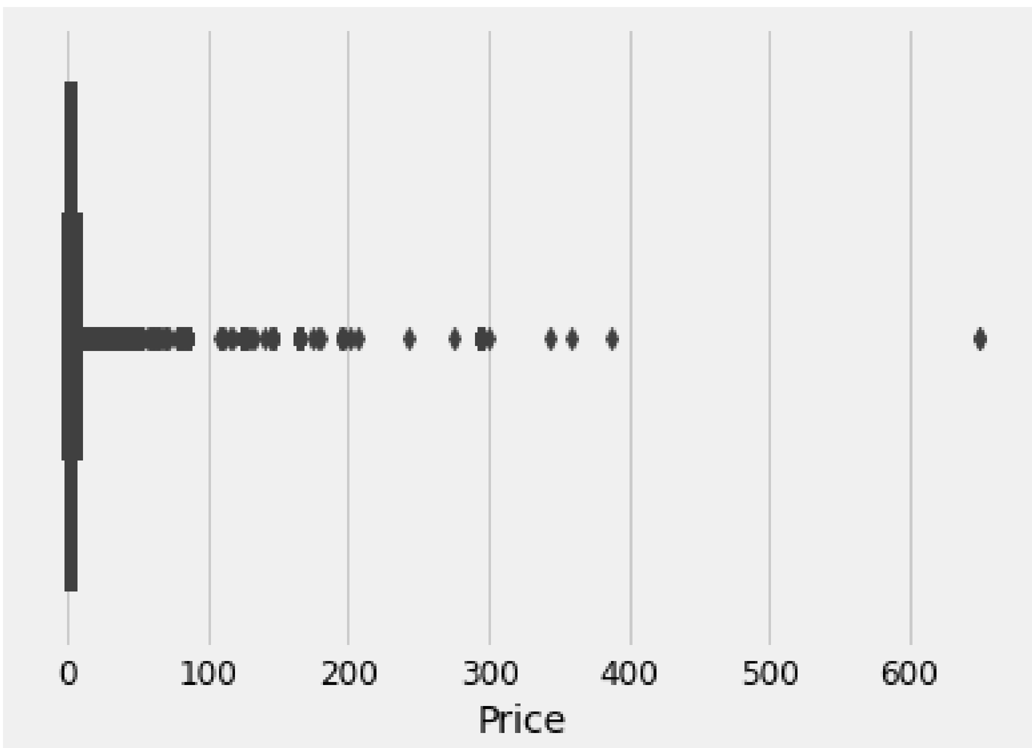
At the beginning, it was noticed that "Price" has also 0 values. After removing unnecassary for the task's purpose observations, are there still 0 values in feature "Price"?

[38]:
```
# summary statistics for "Price" variable
data.Price.describe()
## min   0.000000; occurence of 0 "Price" values
```

[38]:
```
count    775820.000000
mean          2.941663
std           4.427213
min           0.000000
25%           1.250000
50%           1.950000
75%           3.750000
max         649.500000
Name: Price, dtype: float64
```

[39]:
```
# boxplot of variable "Price"
sns.boxplot(x=data["Price"])
```

[39]:
```
<AxesSubplot:xlabel='Price'>
```



By the summary statistics and the boxplot of the feature “Price”, not only 0 values are noticed, but also some outliers.

[40]:
```
data.loc[data.Price == 0].sort_values(by="Quantity", ascending=False).head()
```

[40]:
```
        Invoice StockCode                          Description  Quantity  \
749043   578841     84826        ASSTD DESIGN 3D PAPER STICKERS     12540
272423   524181    46000M         POLYESTER FILLER PAD 45x45cm       648
590765   562973     23157          SET OF 6 NATIVITY MAGNETS       240
```

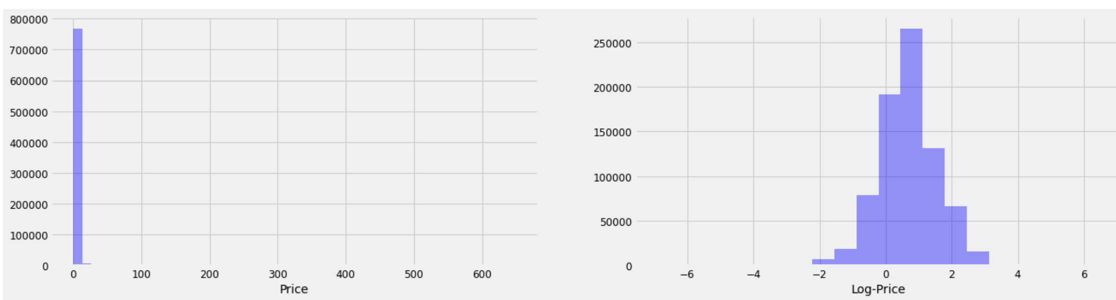
```
699794  574138     23234        BISCUIT TIN VINTAGE CHRISTMAS       216
605566  564651     23268   SET OF 2 CERAMIC CHRISTMAS REINDEER       192

                InvoiceDate  Price  CustomerID         Country
749043 2011-11-25 15:57:00    0.0       13256  United Kingdom
272423 2010-09-27 16:59:00    0.0       17450  United Kingdom
590765 2011-08-11 11:42:00    0.0       14911            EIRE
699794 2011-11-03 11:26:00    0.0       12415       Australia
605566 2011-08-26 14:19:00    0.0       14646     Netherlands
```

[ ]:

The left plot below demonstrates the distribution of the "Price > 0" observations, while the right plot the same but on a log scale.

[41]:

```python
data = data.loc[data.Price > 0].copy()
fig, ax = plt.subplots(1,2,figsize=(20,5))
sns.distplot(data.Price, ax=ax[0], kde=False, color="blue")
sns.distplot(np.log(data.Price), ax=ax[1], bins=20, color="blue", kde=False)
ax[1].set_xlabel("Log-Price");
```



The 95% quantile of "Price" is equal to 8.5, meaning that this number is greater than 95% of the numbers in the data set. The question that arises here is if we should consider removing the outliers, since it is often the case that outliers are pretty informative. In the first place, we are not going to remove them.

[42]:

```python
print(np.exp(-2))
print(np.exp(2))
print(np.quantile(data.Price, 0.95))
```

```
0.1353352832366127
7.38905609893065
8.5
```

[43]:

```python
# in case we want to remove outliers
# data = data.loc[(data.Price > 0.1) & (data.Price < 8.6)].copy()
```

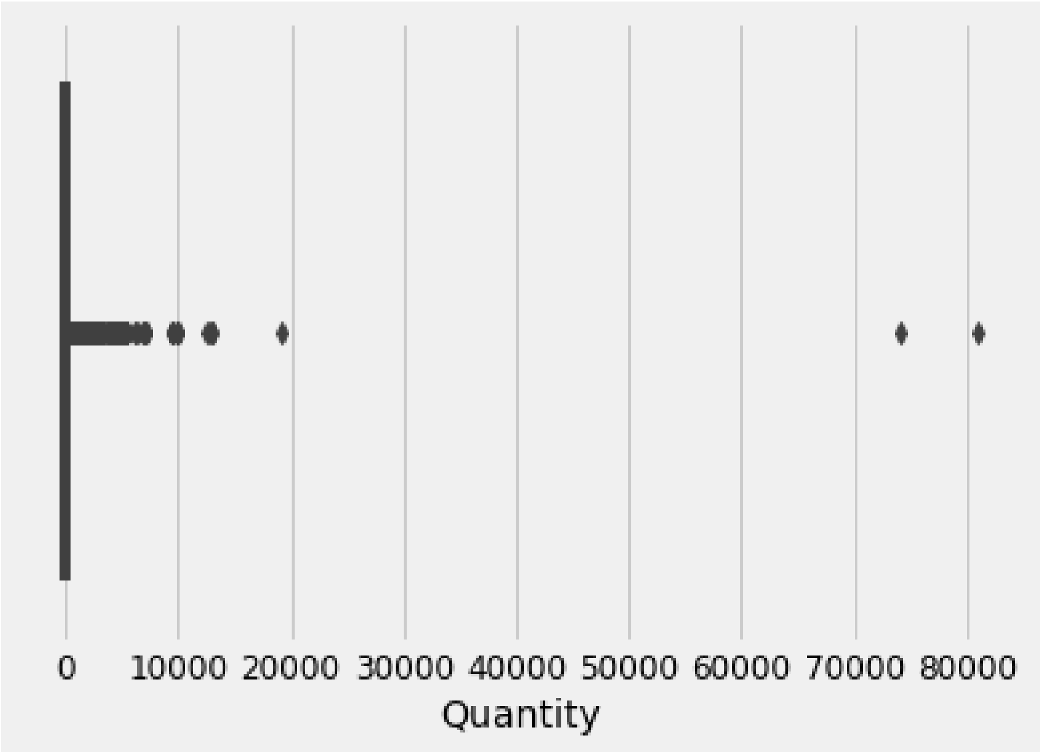
Let's now investigate the feature "Quantity" in a similar manner.

[44]:
```
data.Quantity.describe()
```

[44]:
```
count    775758.000000
mean         13.521066
std         146.173137
min           1.000000
25%           2.000000
50%           6.000000
75%          12.000000
max       80995.000000
Name: Quantity, dtype: float64
```

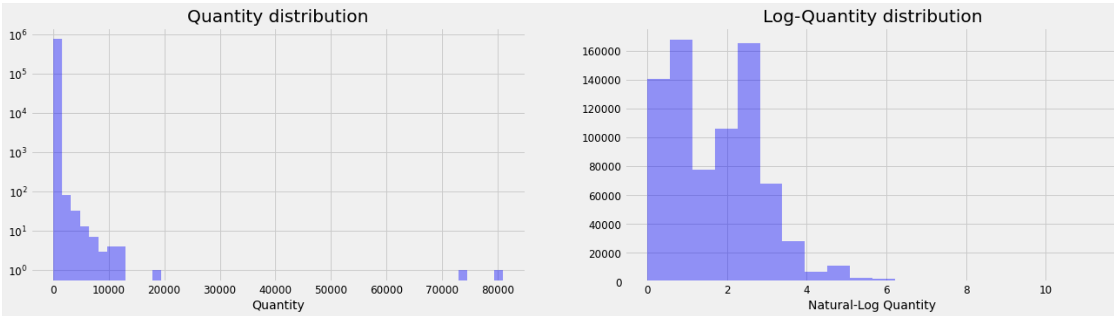
[45]:
```
# boxplot of variable "Quantity"
sns.boxplot(x=data["Quantity"])
```

[45]: <AxesSubplot:xlabel='Quantity'>



[46]:
```
fig, ax = plt.subplots(1,2,figsize=(20,5))
sns.distplot(data.Quantity, ax=ax[0], kde=False, color="blue");
sns.distplot(np.log(data.Quantity), ax=ax[1], bins=20, kde=False, color="blue");
ax[0].set_title("Quantity distribution")
ax[0].set_yscale("log")
```

```python
ax[1].set_title("Log-Quantity distribution")
ax[1].set_xlabel("Natural-Log Quantity");
```



[47]:
```python
print(np.exp(4))
np.quantile(data.Quantity, 0.95)
```

```
54.598150033144236
```

[47]:
```
36.0
```

There are no 0 values in “Quantity”, although some outliers are detected. The 95% quantile of “Quantity” is equal to 36. Again, a choice about the outliers has to be made.

[48]:
```python
# data = data.loc[data.Quantity < 37].copy()
```

By sales of a company is either meant the quantities of a product/item sold (volume sales) or the income/revenue earned (sales revenue). The “Revenue” is simply calculated by multiplying the “Quantity” with the “Price”.

[49]:
```python
data["Revenue"] = data.Quantity * data.Price
data.head()
```

[49]:
```
  Invoice StockCode                          Description  Quantity  \
0  489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1  489434    79323P                   PINK CHERRY LIGHTS        12
2  489434    79323W                  WHITE CHERRY LIGHTS        12
3  489434     22041         RECORD FRAME 7" SINGLE SIZE         48
4  489434     21232       STRAWBERRY CERAMIC TRINKET BOX        24

          InvoiceDate  Price  CustomerID         Country  Revenue
0 2009-12-01 07:45:00   6.95       13085  United Kingdom     83.4
1 2009-12-01 07:45:00   6.75       13085  United Kingdom     81.0
2 2009-12-01 07:45:00   6.75       13085  United Kingdom     81.0
3 2009-12-01 07:45:00   2.10       13085  United Kingdom    100.8
4 2009-12-01 07:45:00   1.25       13085  United Kingdom     30.0
```

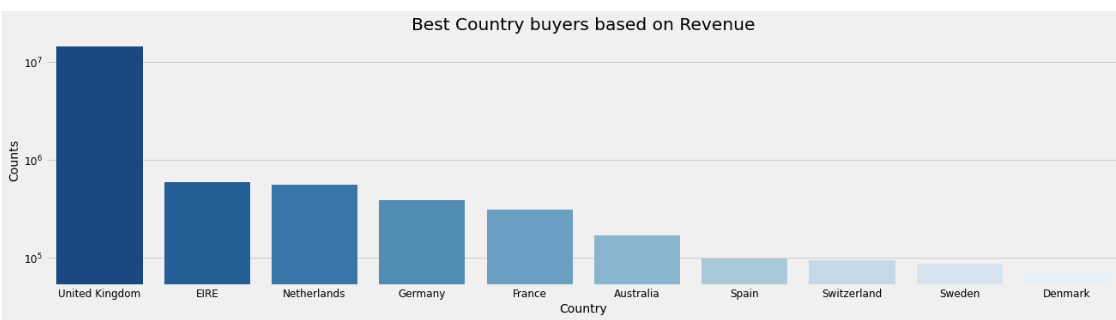
Which countries give the higher “Revenue” to the company?

As expected, UK is at the first place.

[50]:
```
# countries that lead to the highest "Revenue"
data.groupby("Country")["Revenue"].sum().sort_values(ascending=False).iloc[0:10]
```

[50]:
```
Country
United Kingdom    1.427773e+07
EIRE              5.878934e+05
Netherlands       5.496152e+05
Germany           3.828885e+05
France            3.093681e+05
Australia         1.677304e+05
Spain             9.795730e+04
Switzerland       9.336134e+04
Sweden            8.602774e+04
Denmark           6.742269e+04
Name: Revenue, dtype: float64
```

[51]:
```
# plot of the countries that lead to the highest "Revenue"
country_counts = data.groupby("Country")["Revenue"].sum().
↪sort_values(ascending=False).iloc[0:10]
plt.figure(figsize=(20,5))
sns.barplot(country_counts.index, country_counts.values, palette="Blues_r")
plt.ylabel("Counts")
plt.title("Best Country buyers based on Revenue");
plt.yscale("log")
```



Which products give the higher “Revenue” to the store?

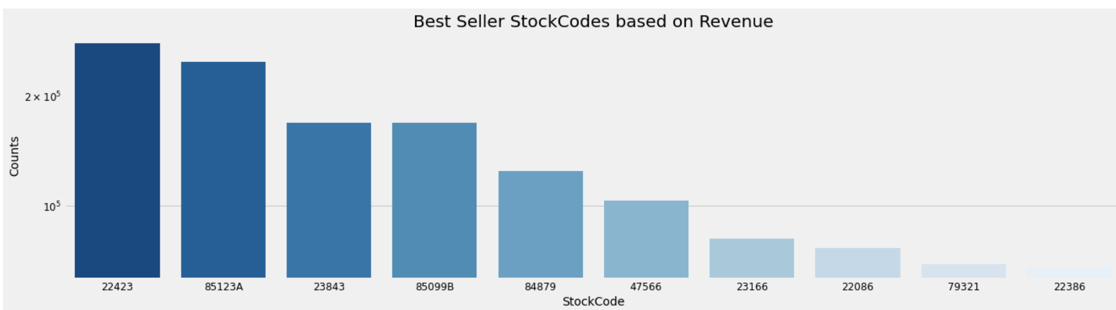
[52]:
```
data.groupby("StockCode")["Revenue"].sum().sort_values(ascending=False).iloc[0:
↪10]
```

[52]:
```
StockCode
22423      277656.25
```

```
85123A    247203.36
23843     168469.60
85099B    167920.64
84879     124351.86
47566     103283.38
23166      81416.73
22086      76598.18
79321      69084.30
22386      67769.76
Name: Revenue, dtype: float64
```

[53]:

```python
# plot of the countries that lead to the highest "Revenue"
stockcode_counts = data.groupby("StockCode")["Revenue"].sum().
 ↪sort_values(ascending=False).iloc[0:10]
plt.figure(figsize=(20,5))
sns.barplot(stockcode_counts.index, stockcode_counts.values, palette="Blues_r")
plt.ylabel("Counts")
plt.title("Best Seller StockCodes based on Revenue");
plt.yscale("log")
```



[54]:

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 775758 entries, 0 to 775819
Data columns (total 9 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Invoice      775758 non-null  object
 1   StockCode    775758 non-null  object
 2   Description  775758 non-null  object
 3   Quantity     775758 non-null  int64
 4   InvoiceDate  775758 non-null  datetime64[ns]
 5   Price        775758 non-null  float64
 6   CustomerID   775758 non-null  int64
 7   Country      775758 non-null  object
 8   Revenue      775758 non-null  float64
```

```
dtypes: datetime64[ns](1), float64(2), int64(2), object(4)
memory usage: 59.2+ MB
```

Let's have another overview in the data set by group of "InvoiceDate" and "StockCode".

[55]:
```
data["InvoiceDate"] = pd.to_datetime(data["InvoiceDate"])

data["Year-Week"] = data["InvoiceDate"].apply(lambda x: '{0}-{1}'.format(x.year, x.isocalendar()[1]))
```

[56]:
```
data.head()
```

[56]:
```
   Invoice StockCode                          Description  Quantity  \
0   489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1   489434    79323P                   PINK CHERRY LIGHTS        12
2   489434    79323W                  WHITE CHERRY LIGHTS        12
3   489434     22041         RECORD FRAME 7" SINGLE SIZE         48
4   489434     21232       STRAWBERRY CERAMIC TRINKET BOX        24

          InvoiceDate  Price  CustomerID         Country  Revenue Year-Week
0 2009-12-01 07:45:00   6.95       13085  United Kingdom     83.4   2009-49
1 2009-12-01 07:45:00   6.75       13085  United Kingdom     81.0   2009-49
2 2009-12-01 07:45:00   6.75       13085  United Kingdom     81.0   2009-49
3 2009-12-01 07:45:00   2.10       13085  United Kingdom    100.8   2009-49
4 2009-12-01 07:45:00   1.25       13085  United Kingdom     30.0   2009-49
```

[57]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 775758 entries, 0 to 775819
Data columns (total 10 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Invoice      775758 non-null  object
 1   StockCode    775758 non-null  object
 2   Description  775758 non-null  object
 3   Quantity     775758 non-null  int64
 4   InvoiceDate  775758 non-null  datetime64[ns]
 5   Price        775758 non-null  float64
 6   CustomerID   775758 non-null  int64
 7   Country      775758 non-null  object
 8   Revenue      775758 non-null  float64
 9   Year-Week    775758 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(2), object(5)
memory usage: 65.1+ MB
```

Before we proceed to the second part of the project, we save the data set to be able to reload it.

[58]:
```
data.to_csv("data.csv")
```