Exploratory Data Analysis with Pandas

Whenever you receive a new dataset, you should perform some form of EDA. The main things you should be doing are:

- getting a feel for what the data look like
- establishing what types the data have
- identifying any potential issues that need cleaning
- considering whether there may be any issues when it comes to modelling the dataset

For this example, we haven't given you an explicit modelling task to think of whilst you perform this task. This can happen in real life too - "what can you do with this data?" is a legitimate question you may receive. Think about what you might be able to model from this data whilst you get to know the dataset. How might your model be useful to the dataset's owner?

Module load

Please import the libraries you'll need for data analysis: `numpy` , `pandas` , `matplotlib.pyplot` , and `seaborn`

In [1]:

```
# Import the required modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

The below is not recommended in general, but will clean up this notebook (there are some issues with seaborn plots kicking out warnings which are a bit messy). Feel free to comment this out if you want to see the warnings.

In [2]:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Loading

Load `retail_data.csv` using the relevant pandas method. Make `"CustomerID"` the index of the DataFrame. Print the first few lines to check your import looks ok.

In [3]:

```
# Import the data
pd.read_csv('data/retail_data.csv')
#notice that the customerID is read in so we could use this as an index, to do this instea
#code, we can use the code below
```

Out[3]:

	CustomerID	Country	balance	max_spent	mean_spent	min_spent	n_orders	time_between_or
0	12346	United Kingdom	0.00	77183.60	38591.800000	0.00	2.0	
1	12348	Finland	3874.60	2248.80	1291.533333	478.80	3.0	54.50
2	12350	Norway	294.40	294.40	294.400000	294.40	1.0	
3	12352	Norway	1845.13	1054.10	393.092000	0.00	5.0	11.33
4	12354	Spain	1079.40	1079.40	1079.400000	1079.40	1.0	
5	12356	Portugal	6621.63	5011.34	3310.815000	1610.29	2.0	80.00
6	12358	Austria	404.86	404.86	404.860000	404.86	1.0	
7	12360	Austria	4359.34	2984.60	2179.670000	1374.74	2.0	88.00

In [4]:

```
pd.read_csv('data/retail_data.csv', index_col='CustomerID')
```

Out[4]:

	Country	balance	max_spent	mean_spent	min_spent	n_orders	time_between_orders	t
CustomerID								
12346	United Kingdom	0.00	77183.60	38591.800000	0.00	2.0	NaN	
12348	Finland	3874.60	2248.80	1291.533333	478.80	3.0	54.500000	
12350	Norway	294.40	294.40	294.400000	294.40	1.0	NaN	
12352	Norway	1845.13	1054.10	393.092000	0.00	5.0	11.333333	
12354	Spain	1079.40	1079.40	1079.400000	1079.40	1.0	NaN	
12356	Portugal	6621.63	5011.34	3310.815000	1610.29	2.0	80.000000	
12358	Austria	404.86	404.86	404.860000	404.86	1.0	NaN	

In [5]:

```
#now let us assign the code above to some variables such as
customers = pd.read_csv('data/retail_data.csv', index_col='CustomerID')
```

In [6]:

```
# Print a few rows of the DataFrame
customers.head()
```

Out[6]:

	Country	balance	max_spent	mean_spent	min_spent	n_orders	time_between_o
CustomerID							
12346	United Kingdom	0.00	77183.6	38591.800000	0.0	2.0	
12348	Finland	3874.60	2248.8	1291.533333	478.8	3.0	54.5
12350	Norway	294.40	294.4	294.400000	294.4	1.0	
12352	Norway	1845.13	1054.1	393.092000	0.0	5.0	11.3
12354	Spain	1079.40	1079.4	1079.400000	1079.4	1.0	

Now what do we have - we have some sort of customer data, we have: the country they are from, a balance, how much they spent, number of orders. So, it looks like this might be some sort of aggregated dataset. I've explicitly not really told you much about this because this is what EDA is about

Understanding the data

The first thing to do is to check the import has definitely worked ok and to get an overview:

1. Get the shape of the data
2. Print some basic statistics about the distribution of each feature e.g. mean, standard deviation, or nr unique values etc. (hint: there is a DataFrame method for this)
3. Get the types of all the data. Do they look right? (answer: they're not strictly right, think about why. However, you don't actually need to edit their types for all the following functions to work)
4. Check missing values in the data - can and should they be cleaned? How would this be handled during the modelling stage?

Now that we are beginning to understand our data we want to do things like: 1.get the shape, 2.basic statistics using the describe method, 3.check data types 4.Are there any rows which contain missing data? Print them (hint: use the methods isnull() and any() find all the rows which contain one or more missing values).

In [7]:

```
# shape
customers.shape         #the output shows there are 3254 rows and 11 features
```

Out[7]:

```
(3254, 11)
```

In [8]:

```
# stats
#customers.describe()

#be careful if you use customers.describe().shape
#this will not output all the column, instead we get (8, 10)
#if we use customers.describe(include='all').shape
#we get (11, 11)

#so instead of customers.describe() we should use
customers.describe(include='all')

#there are some columns that are categorical, for example the column is a categorical vari
```

Out[8]:

	Country	balance	max_spent	mean_spent	min_spent	n_orders	time_b
count	3254	3254.000000	3254.000000	3254.000000	3254.000000	3254.000000	
unique	11	NaN	NaN	NaN	NaN	NaN	
top	United Kingdom	NaN	NaN	NaN	NaN	NaN	
freq	3013	NaN	NaN	NaN	NaN	NaN	
mean	NaN	3595.679152	1241.132434	697.714203	336.855298	4.025814	
std	NaN	12287.159793	2765.253795	1105.379255	592.529725	6.330090	
min	NaN	-1192.200000	0.000000	0.000000	0.000000	1.000000	
25%	NaN	330.807500	298.307500	215.720000	0.000000	1.000000	
50%	NaN	948.595000	658.430000	463.260000	150.000000	2.000000	
75%	NaN	3198.430000	1358.170000	852.338786	418.197500	4.000000	
max	NaN	394689.180000	77183.600000	38591.800000	9885.320000	134.000000	

In [9]:

```
# data types as imported
customers.dtypes
```

Out[9]:

```
Country                  object
balance                 float64
max_spent               float64
mean_spent              float64
min_spent               float64
n_orders                float64
time_between_orders     float64
total_items             float64
total_items_returned    float64
total_refunded          float64
total_spent             float64
dtype: object
```

Are there any rows which contain missing data? Print them (hint: use the methods `isnull()` and `any()` find all the rows which contain one or more missing values).

In [10]:

```
# Print missing value rows
#using:
customers.isnull()
#or use below to check across column
customers.isnull().any(axis=1)

#you can check this by returning the number of rows using
#customers.isnull().any(axis=1).shape    #this returns 3254
```

Out[10]:

```
CustomerID
12346     True
12348    False
12350     True
12352    False
12354     True
12356    False
12358     True
12360    False
12361     True
12362    False
12364     True
12373     True
12377    False
12378     True
12379     True
12380     True
12381     True
12383    False
12384     True
12395    False
12397     True
12399    False
12401     True
12402     True
12405     True
12407    False
12408    False
12409     True
12410    False
12413    False
         ...
18233     True
18235    False
18236    False
18237    False
18239    False
18241    False
18242    False
18245    False
18246     True
18248    False
18250     True
18252     True
18256     True
18257    False
```

```
18259     True
18260    False
18262     True
18263    False
18265     True
18268     True
18269     True
18270     True
18272    False
18273     True
18277     True
18280     True
18281     True
18282     True
18283    False
18287     True
Length: 3254, dtype: bool
```

There are actually many...but you'll see that this makes sense given the data. Are there any rows that have missing values apart from this obvious column? Rewrite the cell above and write code to check (tip: you can use `.drop`).

In [11]:

```
#we can further index the dataframe using
customers[customers.isnull().any(axis=1)]
```

Out[11]:

	Country	balance	max_spent	mean_spent	min_spent	n_orders	time_between_orders	t
CustomerID								
12346	United Kingdom	0.00	77183.60	38591.800000	0.00	2.0	NaN	
12350	Norway	294.40	294.40	294.400000	294.40	1.0	NaN	
12354	Spain	1079.40	1079.40	1079.400000	1079.40	1.0	NaN	
12358	Austria	404.86	404.86	404.860000	404.86	1.0	NaN	
12361	Belgium	174.90	174.90	174.900000	174.90	1.0	NaN	
12364	Belgium	1840.52	1840.52	1840.520000	1840.52	1.0	NaN	
12373	Austria	324.60	324.60	324.600000	324.60	1.0	NaN	

In [12]:

```
# Other missing value columns?
#yes, if we look at the column time_between_orders that is a significant one, either becau
#or there is only one order, so we can drop that column
customers.drop('time_between_orders', axis=1).isnull().any(axis=1)
```

Out[12]:

```
CustomerID
12346    False
12348    False
12350    False
12352    False
12354    False
12356    False
12358    False
12360    False
12361    False
12362    False
12364    False
12373    False
12377    False
12378    False
12379    False
12380    False
12381    False
12383    False
12384    False
12395    False
12397    False
12399    False
12401    False
12402    False
12405    False
12407    False
12408    False
12409    False
12410    False
12413    False
         ...
18233    False
18235    False
18236    False
18237    False
18239    False
18241    False
18242    False
18245    False
18246    False
18248    False
18250    False
18252    False
18256    False
18257    False
18259    False
18260    False
18262    False
18263    False
```

```
18265    False
18268    False
18269    False
18270    False
18272    False
18273    False
18277    False
18280    False
18281    False
18282    False
18283    False
18287    False
Length: 3254, dtype: bool
```

In [13]:

```
#again we can index the customer dataset
customers[customers.drop('time_between_orders', axis=1).isnull().any(axis=1)]
```

Out[13]:

	Country	balance	max_spent	mean_spent	min_spent	n_orders	time_between_or
CustomerID							

As a bonus, write some code which checks if a column looks more like an integer than a float value. Skip this if you can't do it quickly (it's not necessary for what comes next).

In [14]:

```
# BONUS: skip if stuck. Find integer columns
for col in customers.columns:
    col_series = customers[col]
    dtype = col_series.dtype
    has_missing = col_series.isnull().any()
    if dtype == np.dtype('float64') and not has_missing:
        if (col_series.astype('int') == col_series).sum() == col_series.shape[0]:
            print('{} looks like an integer'.format(col))
            print('Converting it to an integer')
            customers[col] = customers[col].astype('int')
customers.dtypes
```

```
n_orders looks like an integer
Converting it to an integer
total_items looks like an integer
Converting it to an integer
total_items_returned looks like an integer
Converting it to an integer
```

Out[14]:

```
Country                  object
balance                 float64
max_spent               float64
mean_spent              float64
min_spent               float64
n_orders                  int32
time_between_orders     float64
total_items               int32
total_items_returned      int32
total_refunded          float64
total_spent             float64
dtype: object
```

============================ QUESTIONS ============================

To be good at EDA, you need to be able to quickly answer simple questions about the data. As you become more proficient at pandas, and using `.groupby`, `.apply`, and other methods, you'll become very fast at answering quick questions. [This is a great link (https://pandas.pydata.org/pandas-docs/stable/groupby.html)](https://pandas.pydata.org/pandas-docs/stable/groupby.html) to the pandas docs which outlines methods you'll find useful.

Below are a series of questions which are reasonable first questions to ask about the data. They start simple and increase in difficulty. There are many ways to get the answers and, for that matter, display them. In my sample solutions, I normally opt to plot solutions when it's appropriate and easy to do so. Feel free to print tables, or even write for loops if you prefer!

If you get inspired, ask and answer your own questions.

This is your chance to investigate and get a feel for this dataset.

1) How many unique customers are there? And, therefore, what is the average number of rows per customer?

In [15]:

```
# Nr unique customers
customers.index.nunique()
```

Out[15]:

```
3254
```

In [16]:

```
# Rows per customer - the number of rows in the dataset /
customers.index.nunique()/customers.shape[0]
```

Out[16]:

```
1.0
```

2) What is the total amount spent by customers? (feel free to display numbers in a nice format)

In [17]:

```
# sum of total spent by all the customers
customers.total_spent.sum()  #this returns 11894018.760000002
#to print it out nicely in a scientific format
'{:.2e}'.format(customers.total_spent.sum())
```

Out[17]:

```
'1.19e+07'
```

3) What is the total amount refunded to customers?

In [18]:

```
# Total amount refunded can be returned using  customers.total_refunded.sum(), however thi
#the refunds are show as negative figures in the dataset. To make it output a positive num
-customers.total_refunded.sum()
```

Out[18]:

```
193678.8
```

4) Assuming the company serving these customers had a balance of 0 at the start, what is their balance now?

In [19]:

```
# The balance of the conpany right now
#we can subtract the total refunded from the total spent by customers, but in this case we
#discussed previoudly
customers.total_spent.sum() + customers.total_refunded.sum()
```

Out[19]:

```
11700339.96
```

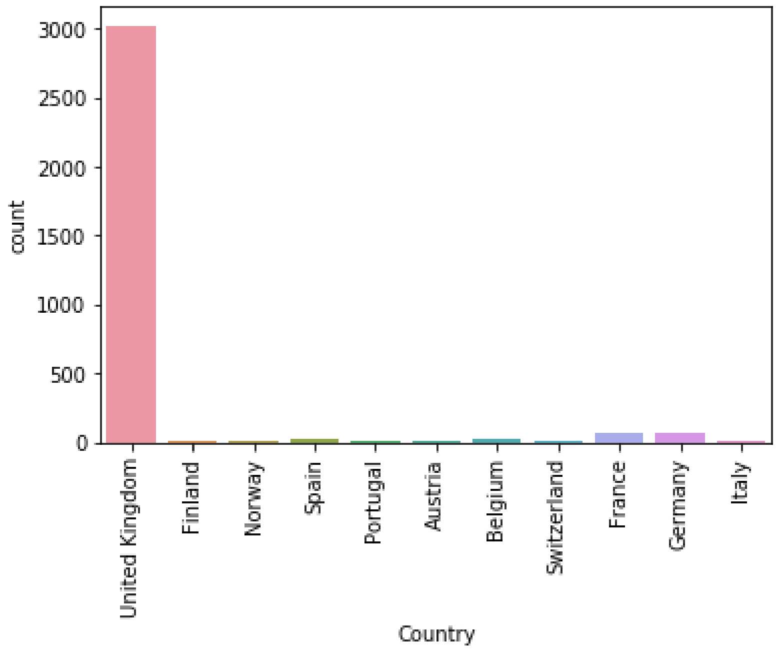
5) Count the number of customers from each country, and all countries excluding the UK

In [20]:

```
# Count of customers by country - use seaborn!
#to do this we can use a countplot, the output on the x axis is messy so rotate the labels
sns.countplot(customers.Country)
plt.xticks(rotation=90)
```

Out[20]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 <a list of 11 Text xticklabel objects>)
```

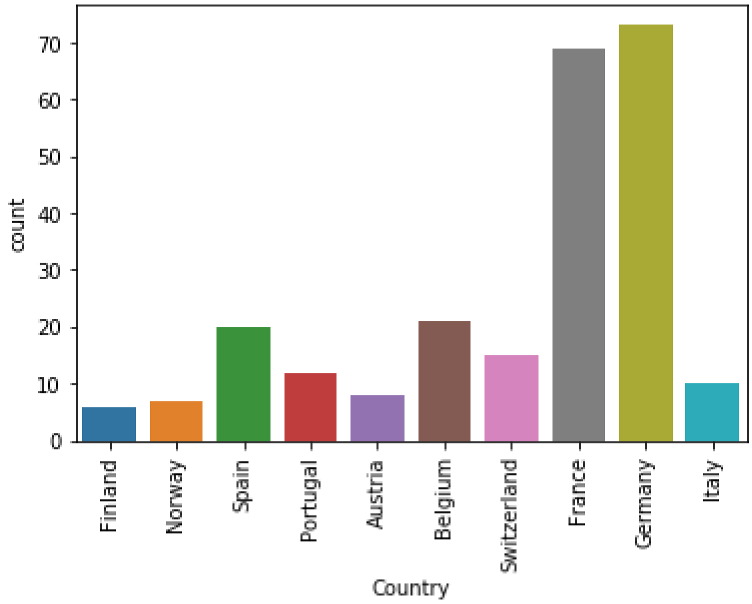


In [21]:

```
# ...excluding UK
sns.countplot(customers.Country[customers.Country != 'United Kingdom'])
plt.xticks(rotation=90)
```

Out[21]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), <a list of 10 Text xticklabel object
s>)
```



6) What is the distribution of the number of orders customers have made? If the distribution is hard to visualise or unclear, try splitting the range up somehow

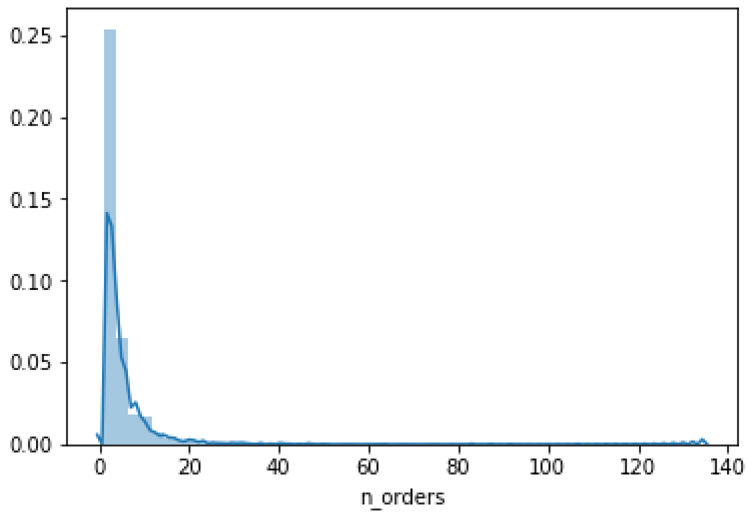
In [22]:

```
# Distribution of number of orders each customer makes - we will use distplot
sns.distplot(customers['n_orders'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b560925cc0>
```



It is not very clear what is going on in that plot. It appears there are some rows with very large values. you might want to have a quick look at all the distribution of all the rows that have some smaller numbers

In [23]:

```
# Distriburtion of the majority of orders
#we will choose a random number 30 and print a subset of the dataset with the rows with or
#as this is small we can use a countplot to examine the data
x = 'n_orders'
query = 'n_orders <= 30'
df = customers.query(query)
df
```

Out[23]:

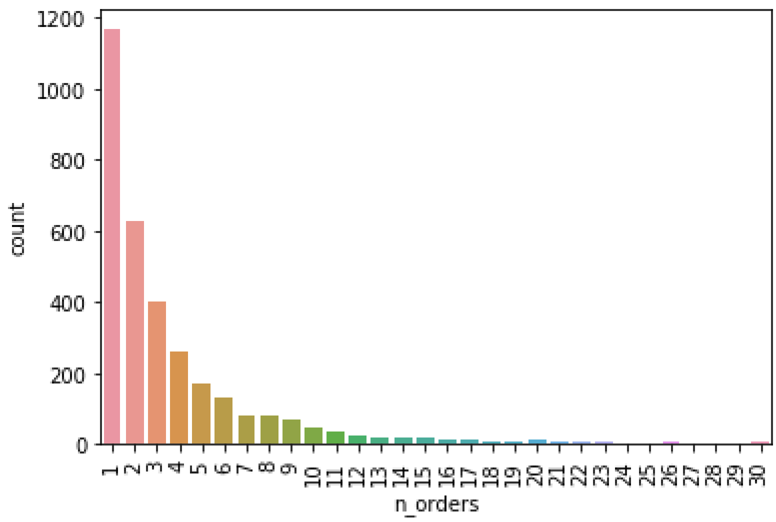
CustomerID	Country	balance	max_spent	mean_spent	min_spent	n_orders	time_between_orders	t
12346	United Kingdom	0.00	77183.60	38591.800000	0.00	2	NaN	
12348	Finland	3874.60	2248.80	1291.533333	478.80	3	54.500000	
12350	Norway	294.40	294.40	294.400000	294.40	1	NaN	
12352	Norway	1845.13	1054.10	393.092000	0.00	5	11.333333	
12354	Spain	1079.40	1079.40	1079.400000	1079.40	1	NaN	
12356	Portugal	6621.63	5011.34	3310.815000	1610.29	2	80.000000	
12358	Austria	404.86	404.86	404.860000	404.86	1	NaN	

In [24]:

```
#using countplot to plot the table above
sns.countplot(df[x])
plt.xticks(rotation=90)
```

Out[24]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]),
 <a list of 30 Text xticklabel objects>)
```



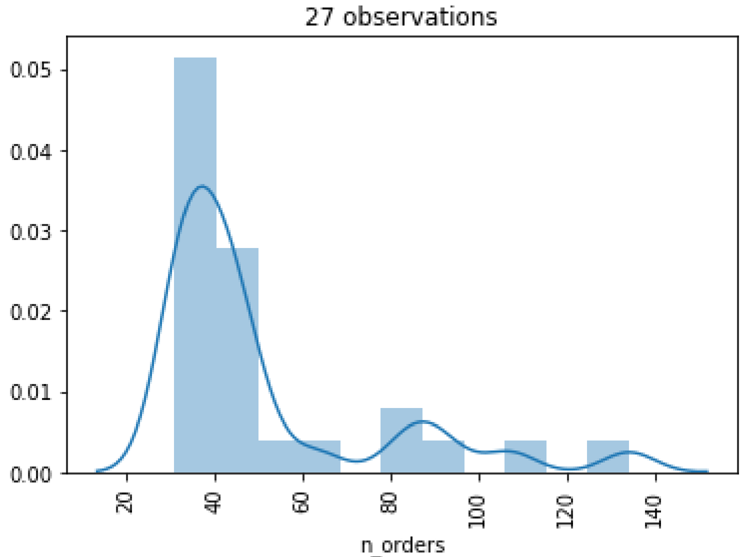
In [25]:

```
# Distribution of the outliers
#Here we could look at orders that are greater than 30 to get the outliers. Instead of a c
#as the data is more disparate. from the plot we see that there are 27 observations that h
x = 'n_orders'
query = 'n_orders > 30'
df = customers.query(query)
sns.distplot(df[x])
plt.xticks(rotation=90)
plt.title('{} observations'.format(df.shape[0]))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[25]:

```
Text(0.5,1,'27 observations')
```



7) What is the distribution of the amount spent by customers? Again, if the distribution is hard to visualise, try splitting the range up somehow. Be careful to show that you understand how many observations you are dealing with in each plot.

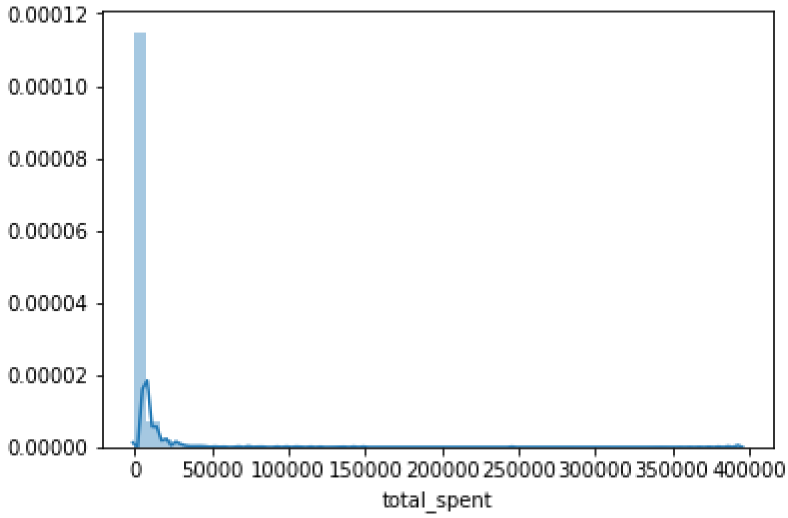
In [26]:

```
# Distribution of the total spent
sns.distplot(customers['total_spent'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b561efa908>
```



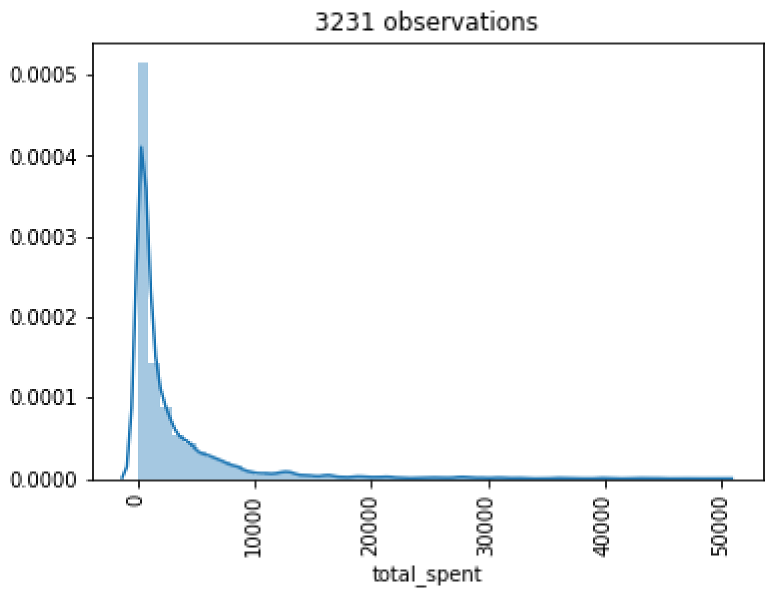
In [27]:

```
# ...the distribution for the majority of companies
x = 'total_spent'
query = 'total_spent <= 50000'
df = customers.query(query)
sns.distplot(df[x])
plt.xticks(rotation=90)
plt.title('{} observations'.format(df.shape[0]))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[27]:

```
Text(0.5,1,'3231 observations')
```



In [28]:

```
#the above plot shows that there are 3231 observations with total spent over 50,000. how m
#dataset. There are 3254 so the lion share is over 500000
customers.shape[0]
```

Out[28]:

```
3254
```

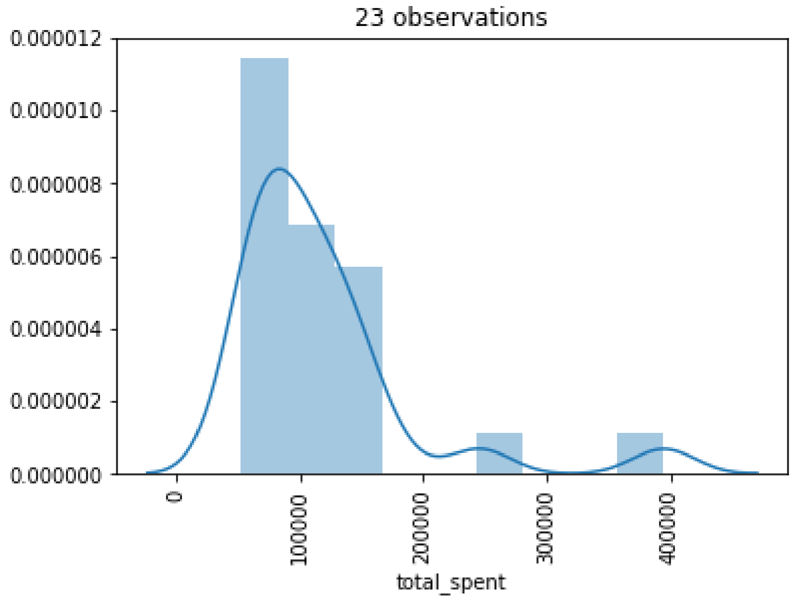
In [29]:

```
# ...the distribution for the outliers
x = 'total_spent'
query = 'total_spent > 50000'
df = customers.query(query)
sns.distplot(df[x])
plt.xticks(rotation=90)
plt.title('{} observations'.format(df.shape[0]))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[29]:

```
Text(0.5,1,'23 observations')
```



8) How about the distribution of refunds? Again, if prevalent values or outliers are making it difficult to visualise the distribution, split the range somehow.

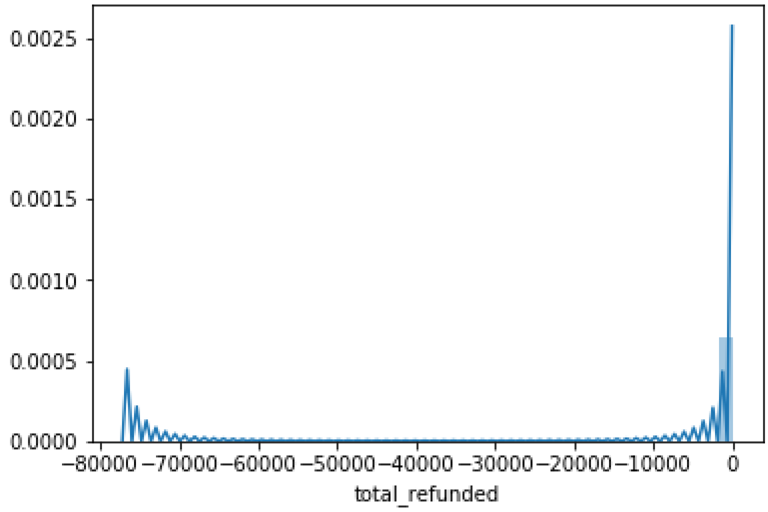
In [40]:

```
# Distribution of refunds
sns.distplot(customers['total_refunded'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[40]:

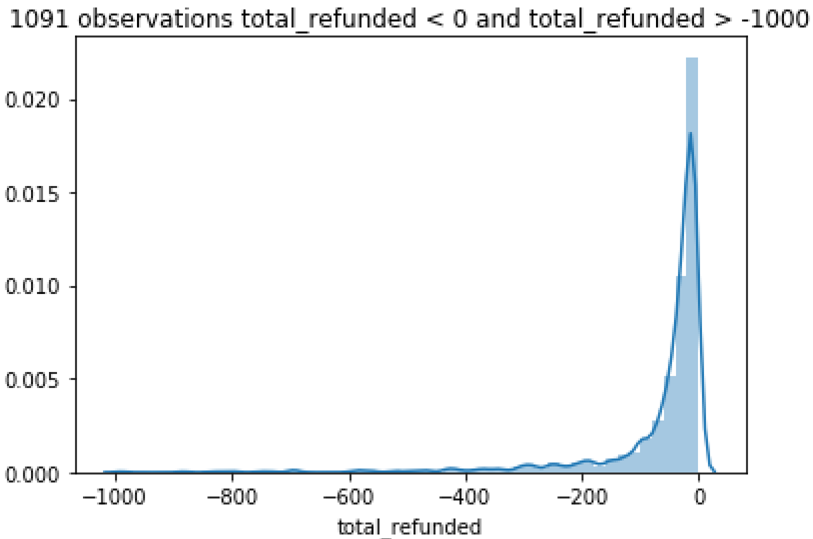
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b562113f98>
```



In [41]:

```
# ...the distribution of the majority
prop_refunded = sum(customers.total_refunded == 0) / customers.shape[0]
print('{:.2f} of customers have never had a refund'.format(prop_refunded))
df = customers.query('total_refunded < 0')
print('Range of the orders with a refund: ({}, {})'.format(
    df['total_refunded'].min(), df['total_refunded'].max()))
query = 'total_refunded < 0 and total_refunded > -1000'
df = customers.query(query)
sns.distplot(df['total_refunded'], rug=False)
plt.title('{} observations {}'.format(df.shape[0], query))
plt.show()
```

```
0.66 of customers have never had a refund
Range of the orders with a refund: (-77183.6, -0.42)

C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



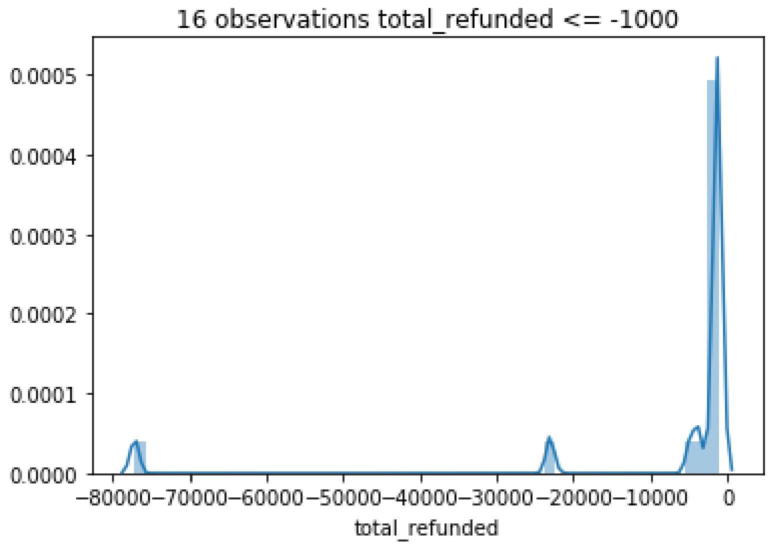
In [42]:

```
# ...the distribution for the outliers
query = 'total_refunded <= -1000'
df = customers.query(query)
sns.distplot(df['total_refunded'], rug=False)
plt.title('{} observations {}'.format(df.shape[0], query))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: Us
erWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'd
ensity' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[42]:

```
Text(0.5,1,'16 observations total_refunded <= -1000')
```



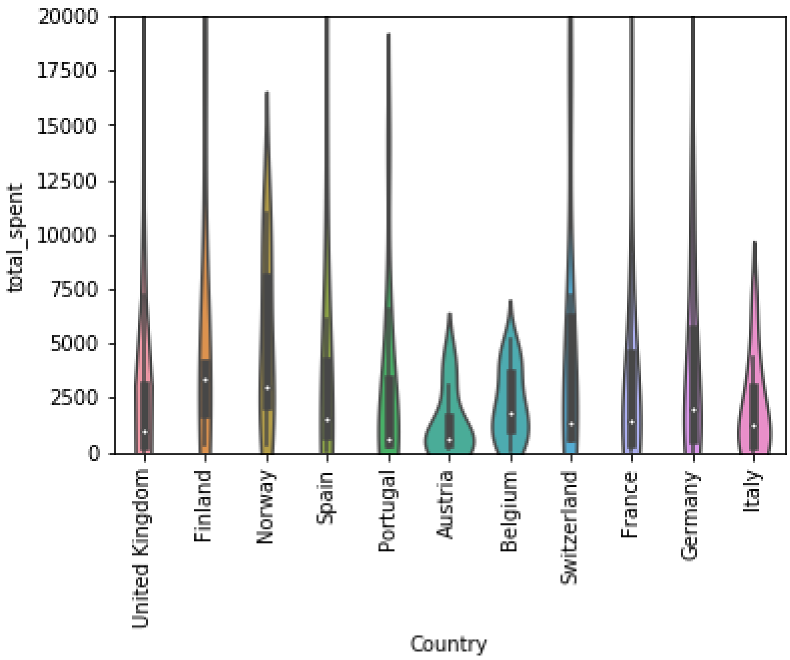
9) Use a [violinplot (https://seaborn.pydata.org/generated/seaborn.violinplot.html?highlight=violin#seaborn.violinplot)](https://seaborn.pydata.org/generated/seaborn.violinplot.html?highlight=violin#seaborn.violinplot) and/or a [boxplot (https://seaborn.pydata.org/generated/seaborn.boxplot.html)](https://seaborn.pydata.org/generated/seaborn.boxplot.html) to plot the distribution of the total spent per country

In [45]:

```
# Violin and box plots of the total spend broken down by country video 8
sns.violinplot(x='Country', y='total_spent', data=customers)
plt.xticks(rotation=90)
plt.ylim(0, 20000)
```

Out[45]:

```
(0, 20000)
```

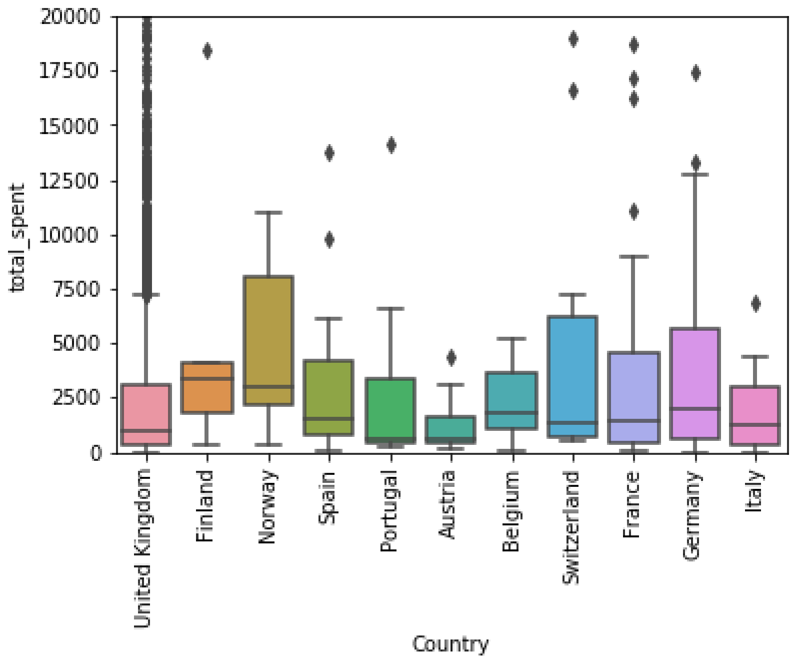


In [47]:

```
sns.boxplot(x='Country', y='total_spent', data=customers)
plt.xticks(rotation=90)
plt.ylim(0, 20000)
```

Out[47]:

```
(0, 20000)
```



10a) What is the total amount spent broken down by country? One option here is to use pandas `groupby` to create a `DataFrame` containing the required information, then use `sns.barplot` to plot the infromation.

In [49]:

```
# Sum of total spent broken down by country
(customers['total_spent']
    .groupby(customers['Country'])
    .sum()
)
#to change the way this is displayed and to add a new column called sum of total spent add
```

Out[49]:

```
Country
Austria              10619.20
Belgium              46252.29
Finland              31091.07
France              272068.47
Germany             325883.91
Italy                20046.71
Norway               34714.18
Portugal             33414.90
Spain                95774.83
Switzerland          67361.13
United Kingdom    10956792.07
Name: total_spent, dtype: float64
```

In [52]:

```
# Sum of total spent broken down by country
(customers['total_spent']
    .groupby(customers['Country'])
    .sum()
    .rename('sum of total spent')
    .reset_index()

)
```

Out[52]:

	Country	sum of total spent
0	Austria	10619.20
1	Belgium	46252.29
2	Finland	31091.07
3	France	272068.47
4	Germany	325883.91
5	Italy	20046.71
6	Norway	34714.18
7	Portugal	33414.90
8	Spain	95774.83
9	Switzerland	67361.13
10	United Kingdom	10956792.07

In [57]:

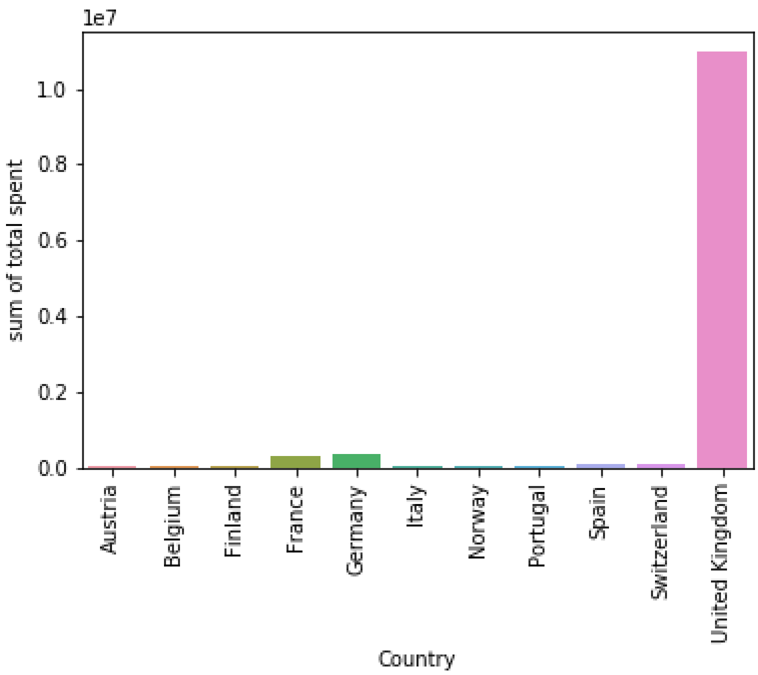
```
#to do the plot below we will assign the code above to a variable
# Sum of total spent broken down by country
sum_tot_spent_by_country = (customers['total_spent']
    .groupby(customers['Country'])
    .sum()
    .rename('sum of total spent')
    .reset_index()

)
```

In [58]:

```
# Plot the data
sns.barplot(x='Country', y='sum of total spent', data=sum_tot_spent_by_country)
plt.xticks(rotation=90)
```

Out[58]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 <a list of 11 Text xticklabel objects>)
```



10b) Again, as might be expected, there is one dominant country. Plot the data again, but exclude the dominant country. In the title of the graph, write explicitly what proportion of the total spent is shown in the plot.

In [61]:

```
# Plot of total spend broken down by country, excluding UK
df = sum_tot_spent_by_country.query('Country != "United Kingdom"')
df
#to calculate the sum and plot this information, see below code
```

Out[61]:

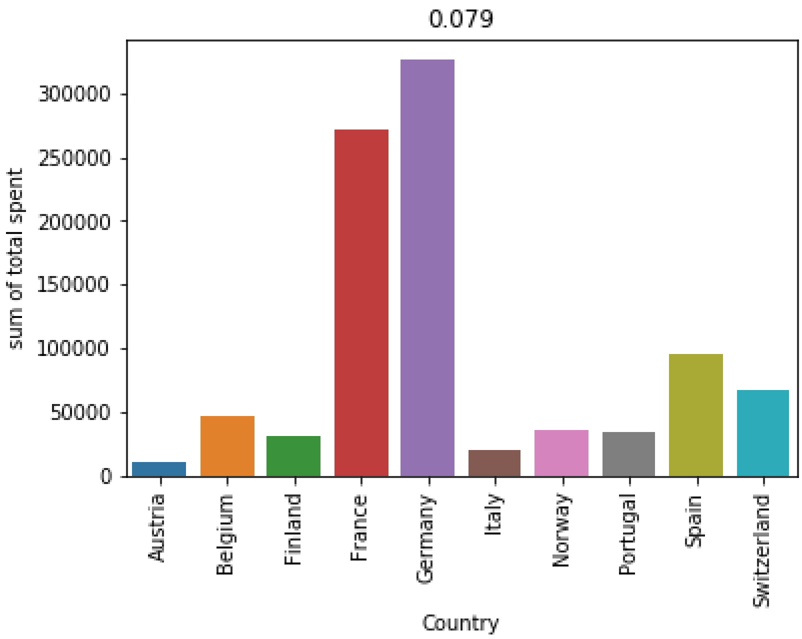
	Country	sum of total spent
0	Austria	10619.20
1	Belgium	46252.29
2	Finland	31091.07
3	France	272068.47
4	Germany	325883.91
5	Italy	20046.71
6	Norway	34714.18
7	Portugal	33414.90
8	Spain	95774.83
9	Switzerland	67361.13

In [62]:

```
# Plot of total spend broken down by country, excluding UK
df = sum_tot_spent_by_country.query('Country != "United Kingdom"')
sns.barplot(x='Country', y='sum of total spent', data=df)
plt.xticks(rotation=90)
prop_uk_spending = customers.query('Country == "United Kingdom"').total_spent.sum() / cust
plt.title('{:.3f}'.format(1-prop_uk_spending))
```

Out[62]:

```
Text(0.5,1,'0.079')
```



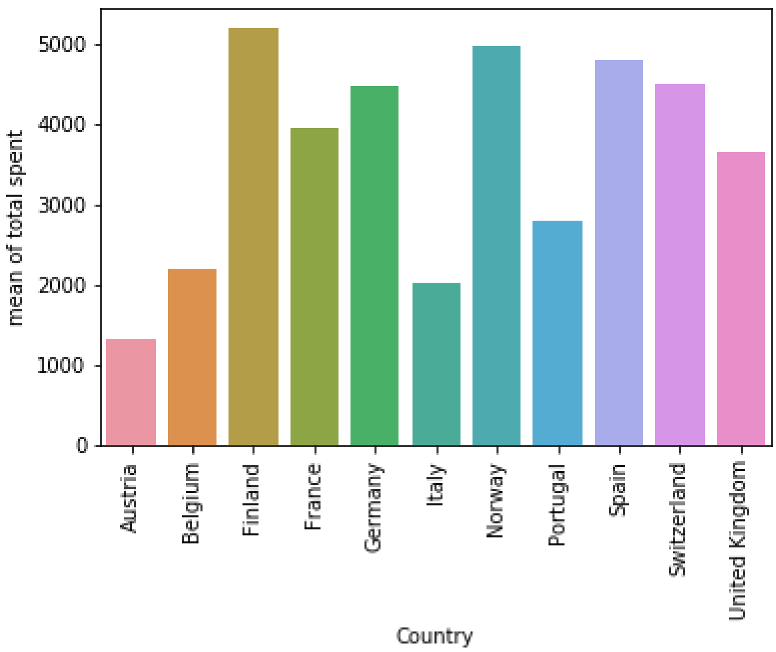
10c) What is the average amount spent per country? (Hint: you'll be able to use very similar code to the analysis summing the total spent)

In [63]:

```
# Mean total_spent by country
mean_tot_spent_by_country = (customers['total_spent']
    .groupby(customers['Country'])
    .mean()
    .rename('mean of total spent')
    .reset_index()

)
sns.barplot(x='Country', y='mean of total spent', data=mean_tot_spent_by_country)
plt.xticks(rotation=90)
```

Out[63]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 <a list of 11 Text xticklabel objects>)
```



10d) Bonus: (move on if not solved in a few mins) -- What is the average amount spent *per order* per country (there is a column `n_orders`)? Hint: be careful...the mean of the column `mean_spent` per country is not the same as the desired quantity e.g. $\frac{\frac{10}{5}+\frac{20}{4}}{2} \neq \frac{10+20}{5+4}$...my solution first makes a function which calculates the mean spend per order for a given dataframe, then I use `apply` to apply this to each country using a `groupby`.

In [64]:

```
# Mean amount spent per order broken down by country -
#You will not get the correct averages if you sum up the totals from the column n_orders a
#you need to do something more complicated which will be shown in the cell below
customers
```

Out[64]:

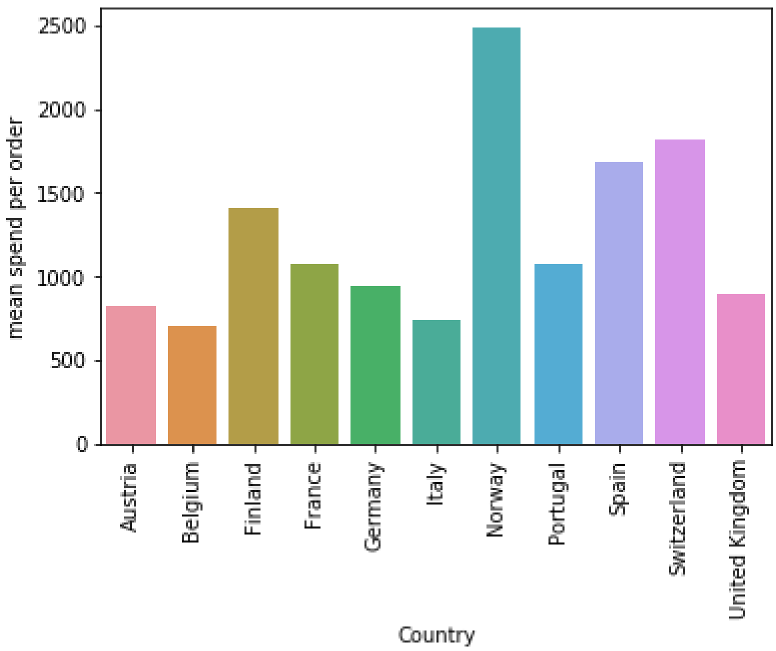
CustomerID	Country	balance	max_spent	mean_spent	min_spent	n_orders	time_between_orders	t
12346	United Kingdom	0.00	77183.60	38591.800000	0.00	2	NaN	
12348	Finland	3874.60	2248.80	1291.533333	478.80	3	54.500000	
12350	Norway	294.40	294.40	294.400000	294.40	1	NaN	
12352	Norway	1845.13	1054.10	393.092000	0.00	5	11.333333	
12354	Spain	1079.40	1079.40	1079.400000	1079.40	1	NaN	
12356	Portugal	6621.63	5011.34	3310.815000	1610.29	2	80.000000	
12358	Austria	404.86	404.86	404.860000	404.86	1	NaN	

In [69]:

```
#we will define a function to do the above
def mean_spent_per_order(df):
    total_orders = df.n_orders.sum()
    total_spent = df.total_spent.sum()
    return total_spent / total_orders     #returns the average per order
mean_spent_per_order(customers)
#breakdown by country
avg_order_spent_by_country = (
    customers
        .groupby('Country')
        .apply(mean_spent_per_order)
        .rename('mean spend per order')
        .reset_index()
)
sns.barplot(x='Country', y='mean spend per order', data=avg_order_spent_by_country)
plt.xticks(rotation=90)
```

Out[69]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 <a list of 11 Text xticklabel objects>)
```



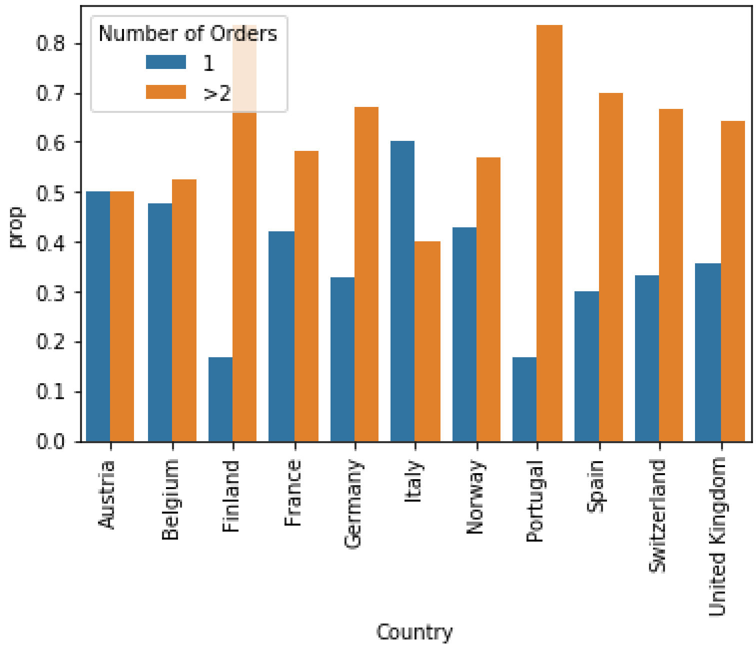
11a) Since the total number of orders is very skewed, when comparing countries, we should compare proportions. Compare the proportion of customers with 1 order vs. 2+ orders for each country. Put this information in one table, or one plot if you can.

In [74]:

```
df = customers
prop_df = (df['n_orders']
     .apply(lambda x: '1' if x==1 else '>2')
     .rename('Number of Orders')
     .groupby(df['Country'])
     .value_counts(normalize=True)
     .rename('prop')
     .reset_index()
)
sns.barplot(x='Country', y='prop', hue='Number of Orders', data=prop_df)
plt.xticks(rotation=90)
```

Out[74]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 <a list of 11 Text xticklabel objects>)
```



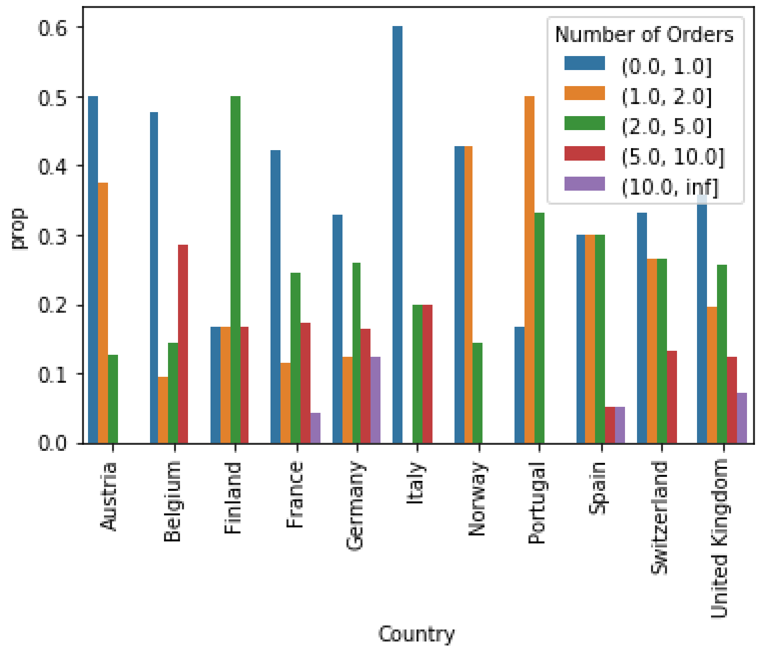
11b) Use `pd.cut` to repeat the above but with some more interesting range bins

In [77]:

```
#we wil copy the function from above and replace lines of it with pd.cut, to see how pd.cu
#cell below and the output
df = customers
prop_df = (pd.cut(df['n_orders'], [0, 1, 2, 5, 10, np.inf])
    .rename('Number of Orders')
    .groupby(df['Country'])
    .value_counts(normalize=True)
    .rename('prop')
    .reset_index()
)
sns.barplot(x='Country', y='prop', hue='Number of Orders', data=prop_df)
plt.xticks(rotation=90)
```

Out[77]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 <a list of 11 Text xticklabel objects>)
```



In [75]:

```
pd.cut(df['n_orders'], [0, 1, 2, 5, 10, np.inf])
```

Out[75]:

```
CustomerID
12346     (1.0, 2.0]
12348     (2.0, 5.0]
12350     (0.0, 1.0]
12352     (2.0, 5.0]
12354     (0.0, 1.0]
12356     (1.0, 2.0]
12358     (0.0, 1.0]
12360     (1.0, 2.0]
12361     (0.0, 1.0]
12362    (5.0, 10.0]
12364     (0.0, 1.0]
12373     (0.0, 1.0]
12377     (1.0, 2.0]
12378     (0.0, 1.0]
12379     (1.0, 2.0]
12380     (0.0, 1.0]
12381     (0.0, 1.0]
12383    (5.0, 10.0]
12384     (0.0, 1.0]
12395    (5.0, 10.0]
12397     (0.0, 1.0]
12399     (2.0, 5.0]
12401     (0.0, 1.0]
12402     (0.0, 1.0]
12405     (0.0, 1.0]
12407     (2.0, 5.0]
12408    (5.0, 10.0]
12409     (1.0, 2.0]
12410     (2.0, 5.0]
12413     (2.0, 5.0]
             ...
18233     (0.0, 1.0]
18235     (1.0, 2.0]
18236     (2.0, 5.0]
18237     (1.0, 2.0]
18239     (2.0, 5.0]
18241    (10.0, inf]
18242     (1.0, 2.0]
18245    (5.0, 10.0]
18246     (0.0, 1.0]
18248     (2.0, 5.0]
18250     (1.0, 2.0]
18252     (0.0, 1.0]
18256     (0.0, 1.0]
18257    (5.0, 10.0]
18259     (0.0, 1.0]
18260    (5.0, 10.0]
18262     (0.0, 1.0]
18263     (1.0, 2.0]
18265     (0.0, 1.0]
18268     (1.0, 2.0]
18269     (1.0, 2.0]
18270     (1.0, 2.0]
```

```
18272     (2.0, 5.0]
18273     (0.0, 1.0]
18277     (0.0, 1.0]
18280     (0.0, 1.0]
18281     (0.0, 1.0]
18282     (1.0, 2.0]
18283    (5.0, 10.0]
18287     (0.0, 1.0]
Name: n_orders, Length: 3254, dtype: category
Categories (5, interval[float64]): [(0.0, 1.0] < (1.0, 2.0] < (2.0, 5.0] <
(5.0, 10.0] < (10.0, inf]]
```

12) In your own words, summarise what you've found out about this dataset from your analysis above

In [39]:

```
# Interpret your findings above!
# Interpret your findings above!
# * This is a dataset about orders customers have made with a business
# * The data are aggregated up to the customer level
# * The features are summary statistics about their order history
# -------
# * There is a total spending of 11 million in this dataset (information about
#   what denomination is not contained in the dataset - it could be cents!)
# * There is comparatively little refunded
# * The customers are mostly from the UK
# * Of customers excluding the UK, France and Germany are most common
# * From simple distribution plots of spending and n_order variables, we can
#   see we have some companies that are big outliers
# * The pattern is roughly the same regarding the total amount spent
# * The distribution of total spent by customers from each region, the
#   distributions aren't radically different...but there are a few outliers
#   with very large spends
# * The mean order is about £1000 (and the mean item cost is a mere £1.69...
#   my bet is that this dataset is about selling office stationary at this
#   point)
# * For each country, between 30 and 50 percent of customers have submitted
#   only one order (Bonus: germany seems to have loyal customers, over 10%
#   of the customers have ordered 10 times or more!)
# * ...this is by no means an exhaustive list!!!
```