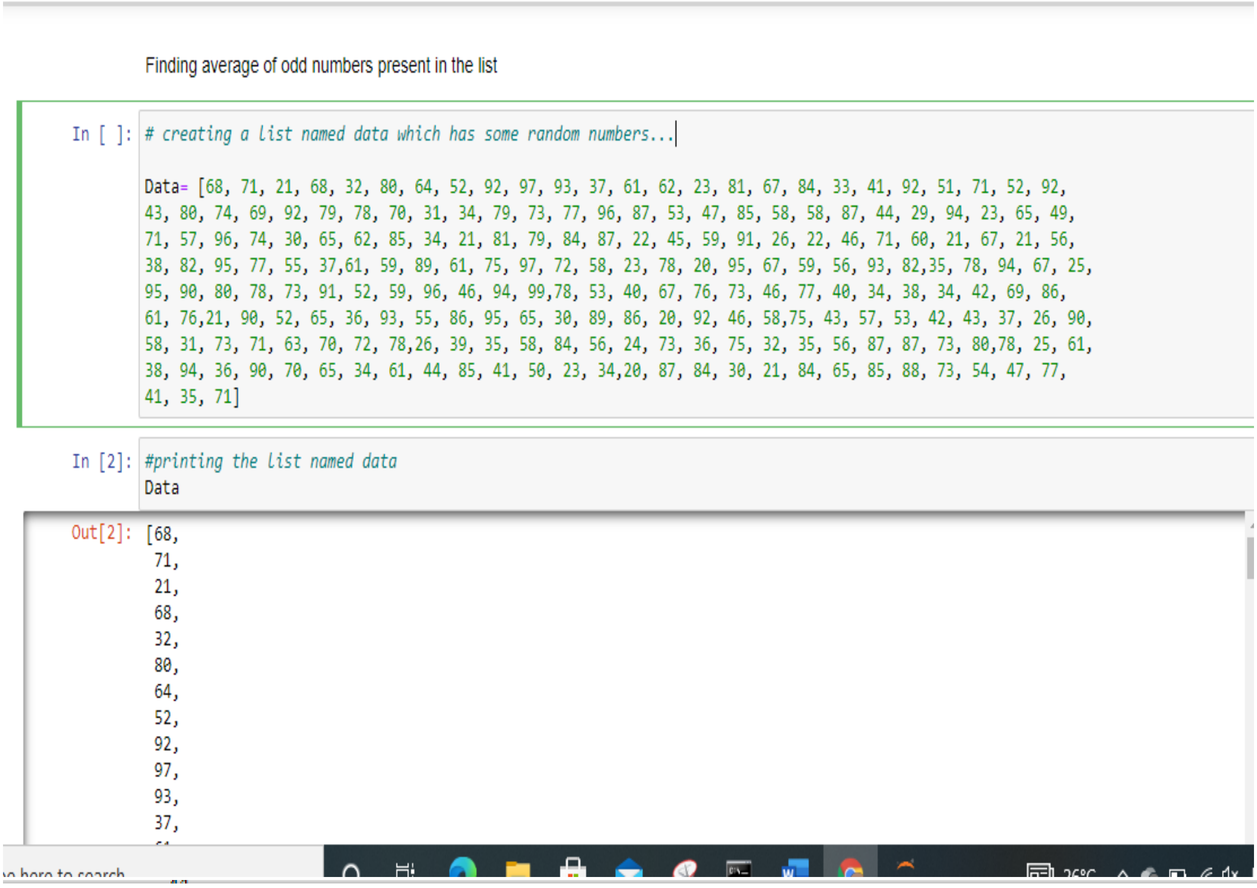
Yashasvi Mehta

Basic Python Programming:

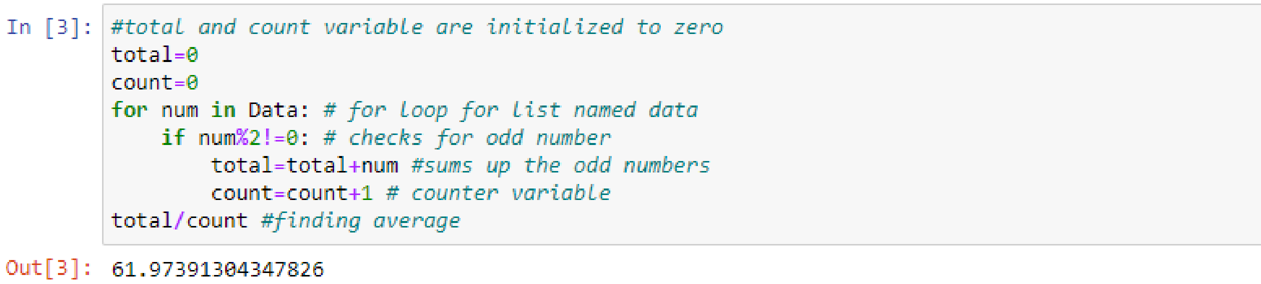
1. **Can you write program to find average of odd numbers in the given list?**

   ➢



Finding average of odd numbers present in the list

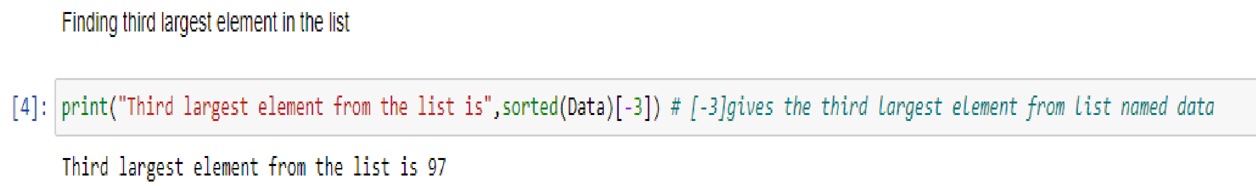
```
In [ ]: # creating a list named data which has some random numbers...

        Data= [68, 71, 21, 68, 32, 80, 64, 52, 92, 97, 93, 37, 61, 62, 23, 81, 67, 84, 33, 41, 92, 51, 71, 52, 92,
        43, 80, 74, 69, 92, 79, 78, 70, 31, 34, 79, 73, 77, 96, 87, 53, 47, 85, 58, 58, 87, 44, 29, 94, 23, 65, 49,
        71, 57, 96, 74, 30, 65, 62, 85, 34, 21, 81, 79, 84, 87, 22, 45, 59, 91, 26, 22, 46, 71, 60, 21, 67, 21, 56,
        38, 82, 95, 77, 55, 37,61, 59, 89, 61, 75, 97, 72, 58, 23, 78, 20, 95, 67, 59, 56, 93, 82,35, 78, 94, 67, 25,
        95, 90, 80, 78, 73, 91, 52, 59, 96, 46, 94, 99,78, 53, 40, 67, 76, 73, 46, 77, 40, 34, 38, 34, 42, 69, 86,
        61, 76,21, 90, 52, 65, 36, 93, 55, 86, 95, 65, 30, 89, 86, 20, 92, 46, 58,75, 43, 57, 53, 42, 43, 37, 26, 90,
        58, 31, 73, 71, 63, 70, 72, 78,26, 39, 35, 58, 84, 56, 24, 73, 36, 75, 32, 35, 56, 87, 87, 73, 80,78, 25, 61,
        38, 94, 36, 90, 70, 65, 34, 61, 44, 85, 41, 50, 23, 34,20, 87, 84, 30, 21, 84, 65, 85, 88, 73, 54, 47, 77,
        41, 35, 71]

In [2]: #printing the list named data
        Data

Out[2]: [68,
         71,
         21,
         68,
         32,
         80,
         64,
         52,
         92,
         97,
         93,
         37,
```



```
In [3]: #total and count variable are initialized to zero
        total=0
        count=0
        for num in Data: # for loop for list named data
            if num%2!=0: # checks for odd number
                total=total+num #sums up the odd numbers
                count=count+1 # counter variable
        total/count #finding average

Out[3]: 61.97391304347826
```

2. **Write a python program to find the third largest number in the given list?**

   ➢



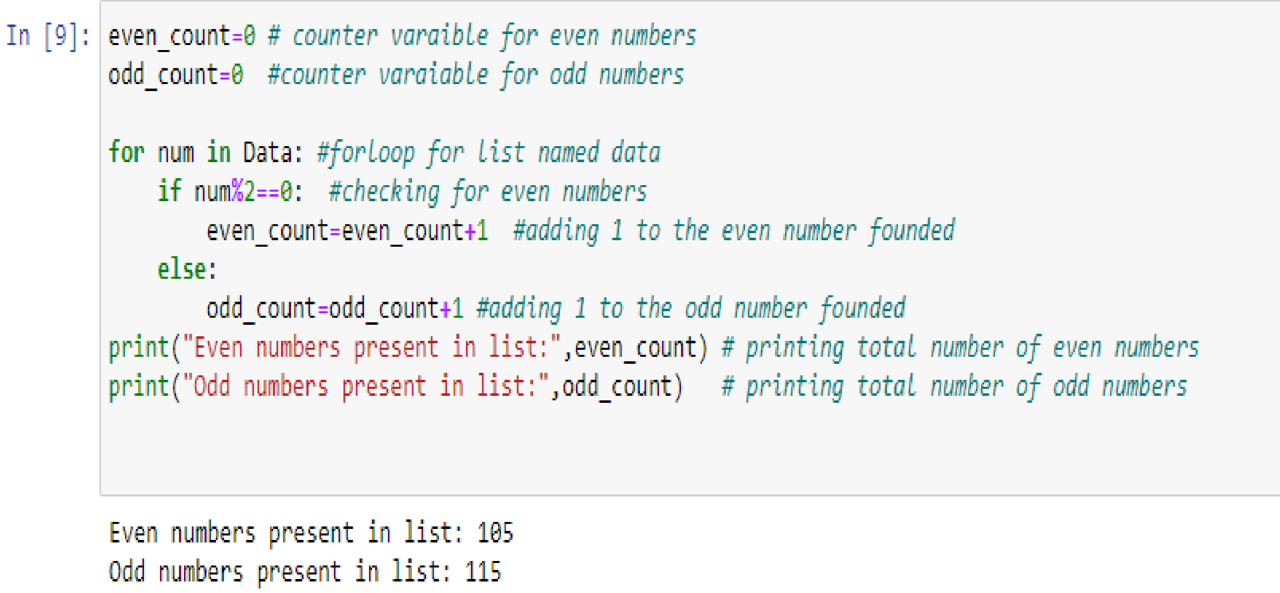
Finding third largest element in the list

```
[4]: print("Third largest element from the list is",sorted(Data)[-3]) # [-3]gives the third largest element from list named data
```

```
Third largest element from the list is 97
```

3. **Write a python program to find the count of even and odd numbers in the given list?**

   ➢



```
In [9]: even_count=0 # counter varaible for even numbers
        odd_count=0  #counter varaiable for odd numbers

        for num in Data: #forloop for list named data
            if num%2==0:  #checking for even numbers
                even_count=even_count+1  #adding 1 to the even number founded
            else:
                odd_count=odd_count+1 #adding 1 to the odd number founded
        print("Even numbers present in list:",even_count) # printing total number of even numbers
        print("Odd numbers present in list:",odd_count)   # printing total number of odd numbers
```

```
Even numbers present in list: 105
Odd numbers present in list: 115
```

Data Analysis for Python Programming

Task: Develop a Sales Analysis by using the dataset set given in the link https://bostonin my.sharepoint.com/:u:/g/personal/krishna_mouli_bostoni ndia_in/Ec_0wGWIsOtNtbghyT9KFwoBQ jtIt8LMYcD_C0GbkxzaUQ?e=MOmLWm

Information of the data Context This Online Retail II data set contains all the transactions occurring for a UK-based and registered, non-store online retail between 01/12/2009 and 09/12/2011.The company mainly sells unique all-occasion gift-ware. Many customers of the company are wholesalers. Content Attribute Information:

InvoiceNo: Invoice number. Nominal. A 6-digit integral number uniquely assigned to each transaction. If this code starts with the letter 'c', it indicates a cancellation.

StockCode: Product (item) code. Nominal. A 5-digit integral number uniquely assigned to each distinct product.

Description: Product (item) name. Nominal.

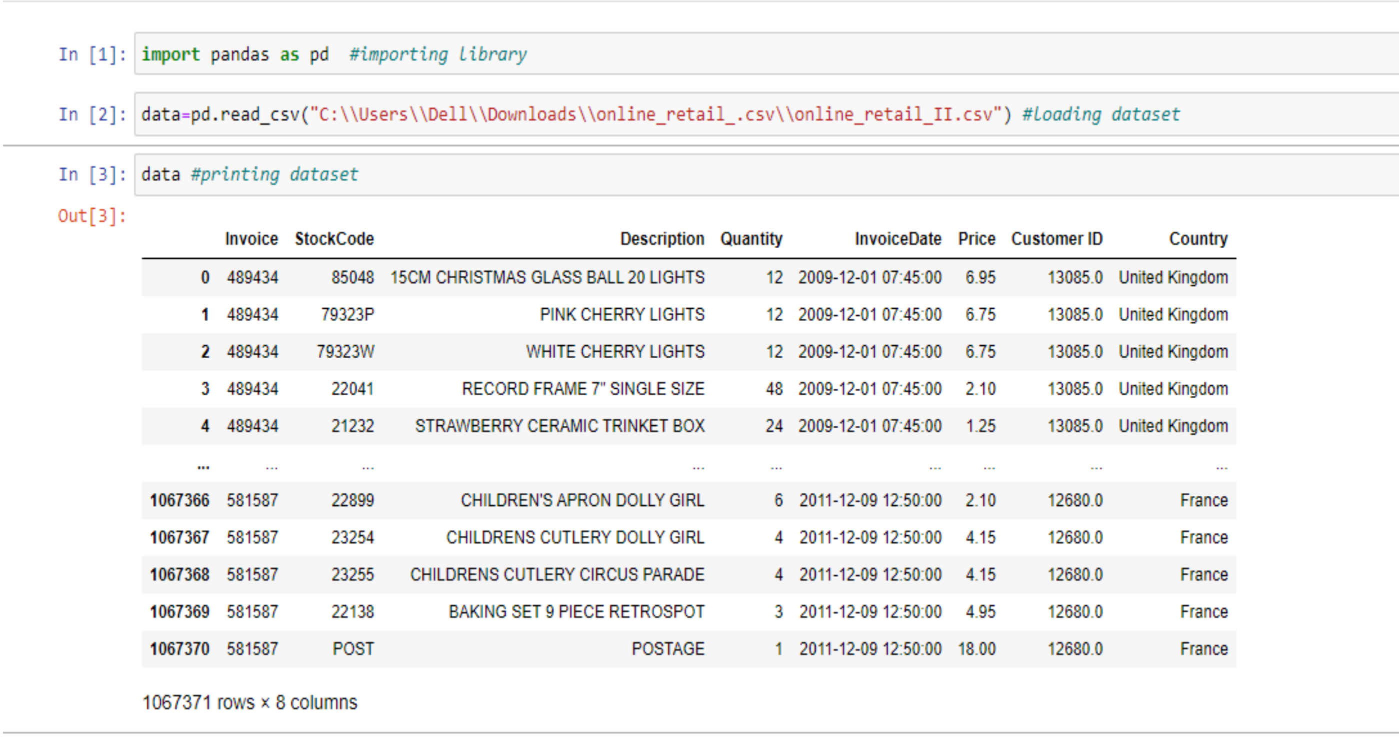
Quantity: The quantities of each product (item) per transaction. Numeric.

InvoiceDate: Invoice date and time. Numeric. The day and time when a transaction was generated. UnitPrice: Unit price. Numeric. Product price per unit in sterling (£).

CustomerID: Customer number. Nominal. A 5-digit integral number uniquely assigned to each customer.

Country: Country name. Nominal. The name of the country where a customer resides. Enter your code below as you answer

- **Answer:**



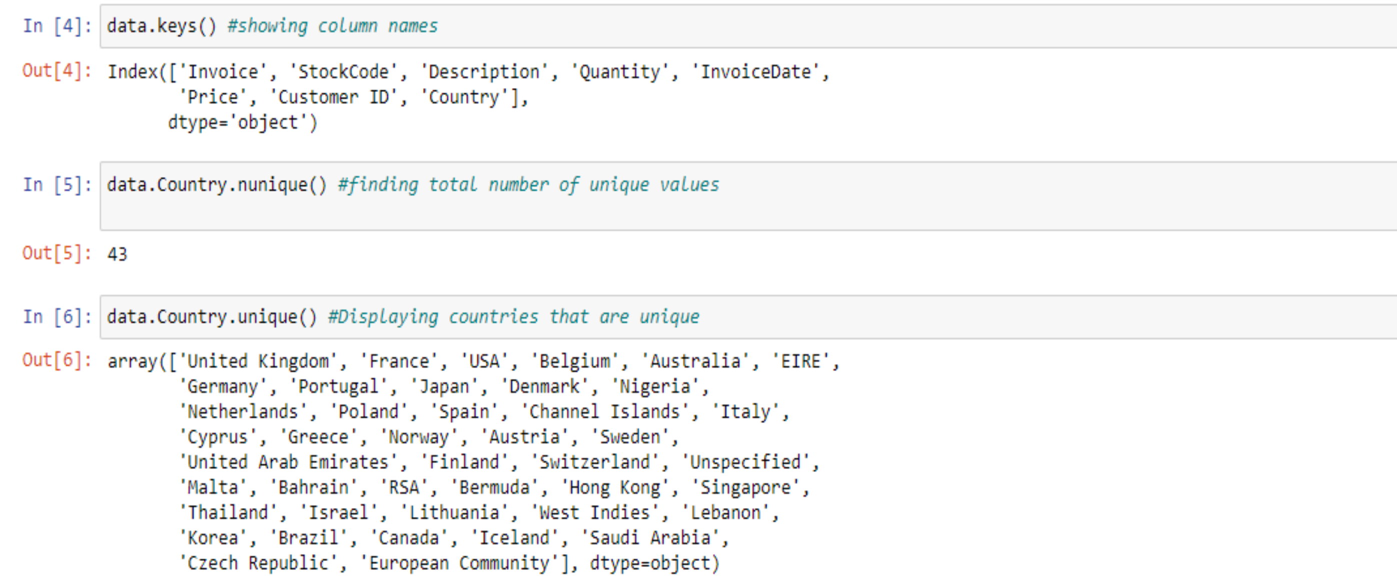
```
In [1]: import pandas as pd  #importing library

In [2]: data=pd.read_csv("C:\\Users\\Dell\\Downloads\\online_retail_.csv\\online_retail_II.csv") #loading dataset

In [3]: data #printing dataset
```

Out[3]:

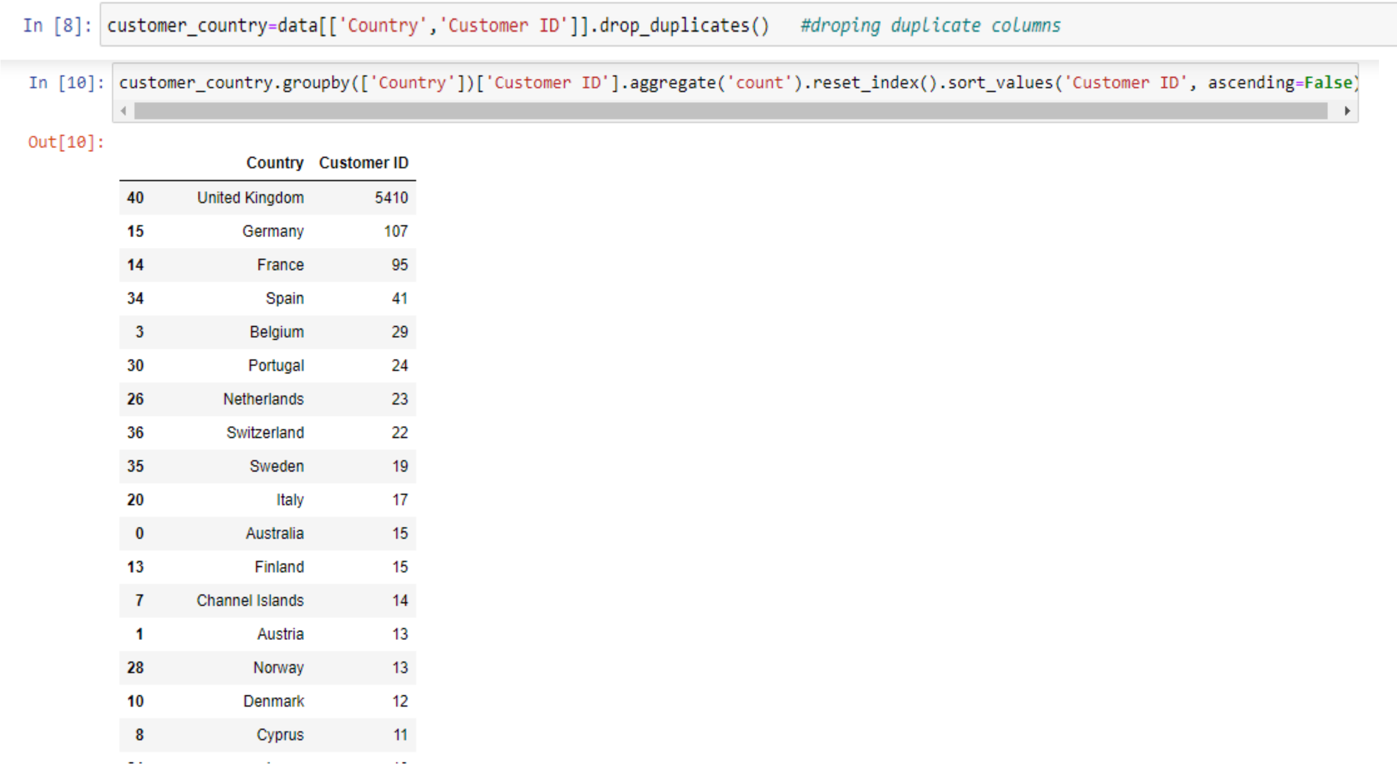
	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom
...	...	...	...	...	...	...	...	...
1067366	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
1067367	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
1067368	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
1067369	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France
1067370	581587	POST	POSTAGE	1	2011-12-09 12:50:00	18.00	12680.0	France

1067371 rows × 8 columns

Yashasvi Mehta



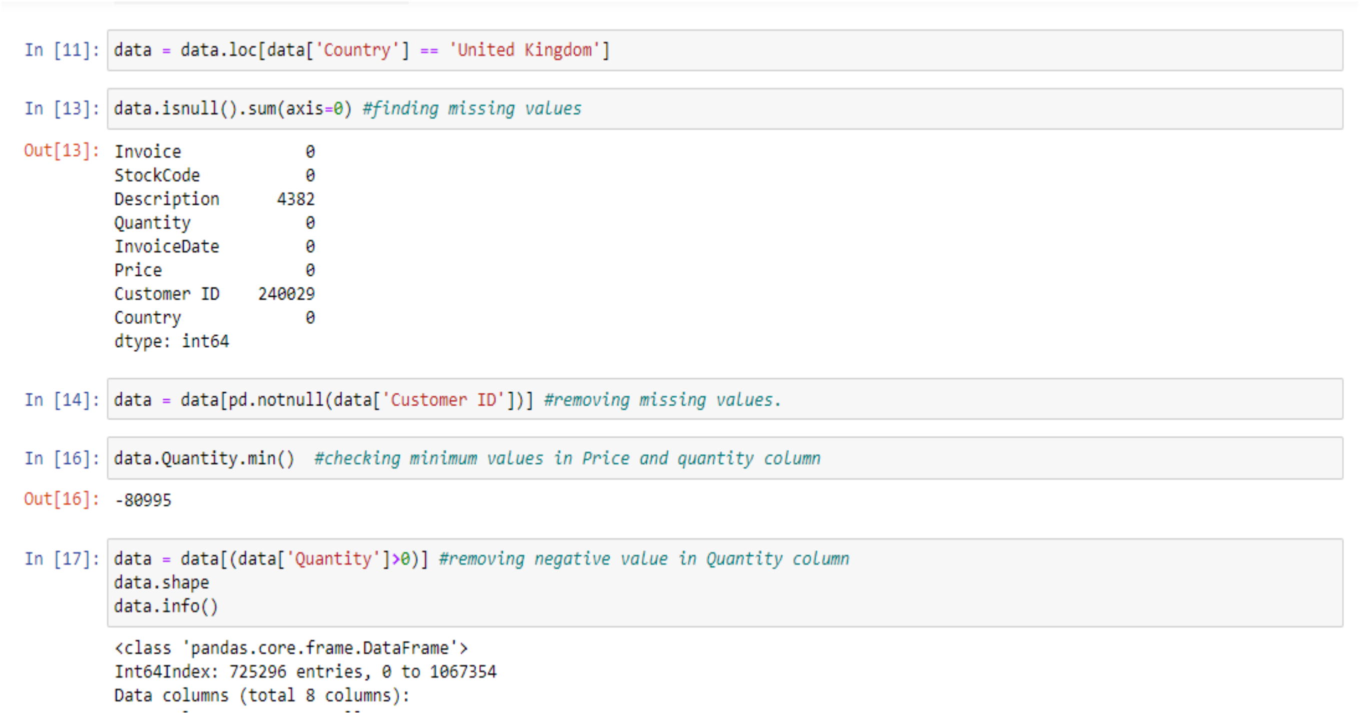
```
In [4]: data.keys() #showing column names

Out[4]: Index(['Invoice', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
               'Price', 'Customer ID', 'Country'],
              dtype='object')

In [5]: data.Country.nunique() #finding total number of unique values

Out[5]: 43

In [6]: data.Country.unique() #Displaying countries that are unique

Out[6]: array(['United Kingdom', 'France', 'USA', 'Belgium', 'Australia', 'EIRE',
               'Germany', 'Portugal', 'Japan', 'Denmark', 'Nigeria',
               'Netherlands', 'Poland', 'Spain', 'Channel Islands', 'Italy',
               'Cyprus', 'Greece', 'Norway', 'Austria', 'Sweden',
               'United Arab Emirates', 'Finland', 'Switzerland', 'Unspecified',
               'Malta', 'Bahrain', 'RSA', 'Bermuda', 'Hong Kong', 'Singapore',
               'Thailand', 'Israel', 'Lithuania', 'West Indies', 'Lebanon',
               'Korea', 'Brazil', 'Canada', 'Iceland', 'Saudi Arabia',
               'Czech Republic', 'European Community'], dtype=object)
```



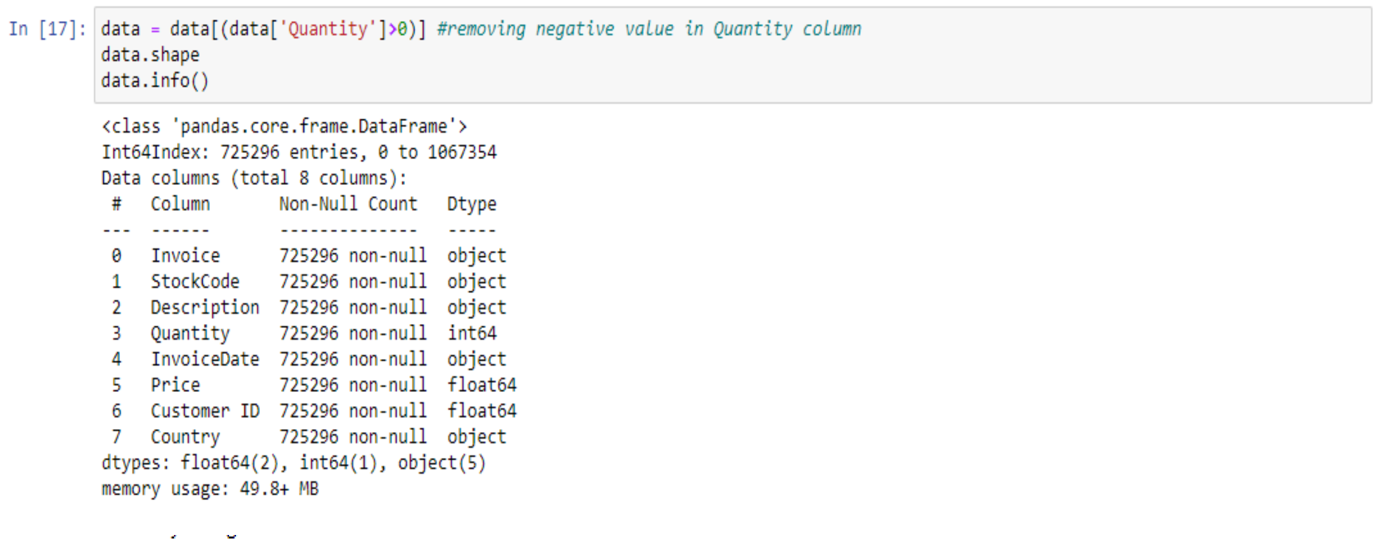
```
In [8]: customer_country=data[['Country','Customer ID']].drop_duplicates()    #droping duplicate columns

In [10]: customer_country.groupby(['Country'])['Customer ID'].aggregate('count').reset_index().sort_values('Customer ID', ascending=False)

Out[10]:
```

	Country	Customer ID
40	United Kingdom	5410
15	Germany	107
14	France	95
34	Spain	41
3	Belgium	29
30	Portugal	24
26	Netherlands	23
36	Switzerland	22
35	Sweden	19
20	Italy	17
0	Australia	15
13	Finland	15
7	Channel Islands	14
1	Austria	13
28	Norway	13
10	Denmark	12
8	Cyprus	11

Yashasvi Mehta

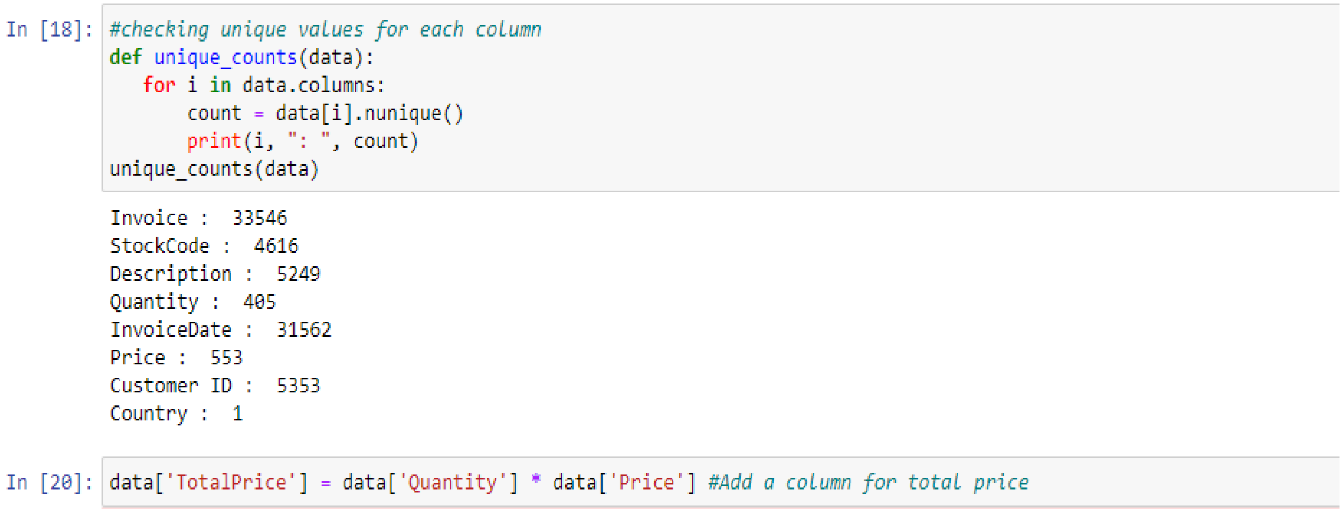


```
In [11]: data = data.loc[data['Country'] == 'United Kingdom']

In [13]: data.isnull().sum(axis=0) #finding missing values

Out[13]: Invoice             0
         StockCode           0
         Description      4382
         Quantity            0
         InvoiceDate         0
         Price               0
         Customer ID    240029
         Country             0
         dtype: int64

In [14]: data = data[pd.notnull(data['Customer ID'])] #removing missing values.

In [16]: data.Quantity.min()  #checking minimum values in Price and quantity column

Out[16]: -80995

In [17]: data = data[(data['Quantity']>0)] #removing negative value in Quantity column
         data.shape
         data.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 725296 entries, 0 to 1067354
         Data columns (total 8 columns):
```

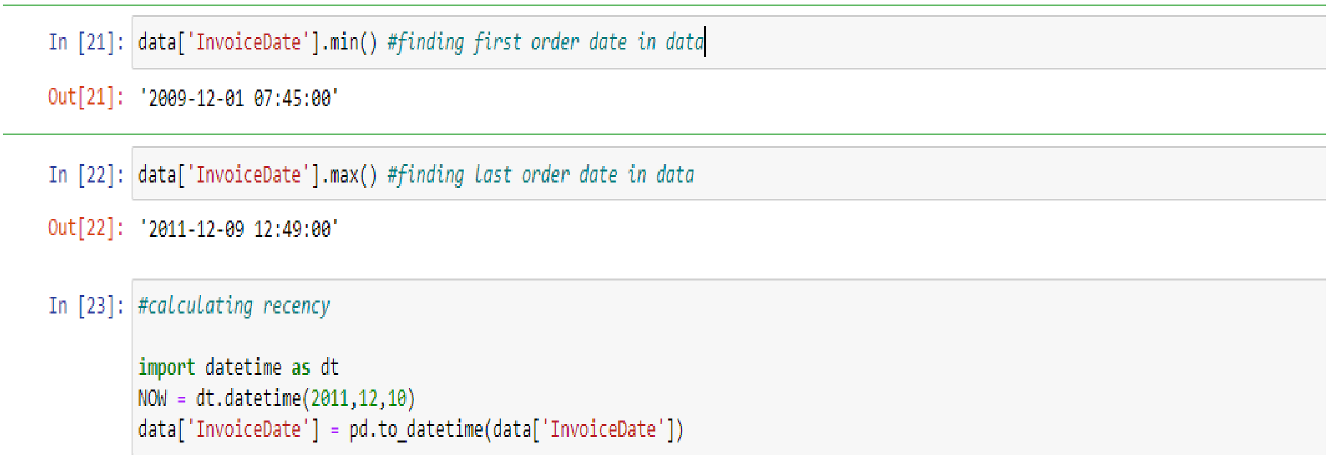


```
In [17]: data = data[(data['Quantity']>0)] #removing negative value in Quantity column
         data.shape
         data.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 725296 entries, 0 to 1067354
         Data columns (total 8 columns):
```

```
          #   Column       Non-Null Count   Dtype  
         ---  ------       --------------   -----  
          0   Invoice      725296 non-null  object 
          1   StockCode    725296 non-null  object 
          2   Description  725296 non-null  object 
          3   Quantity     725296 non-null  int64  
          4   InvoiceDate  725296 non-null  object 
          5   Price        725296 non-null  float64
          6   Customer ID  725296 non-null  float64
          7   Country      725296 non-null  object 
         dtypes: float64(2), int64(1), object(5)
         memory usage: 49.8+ MB
```



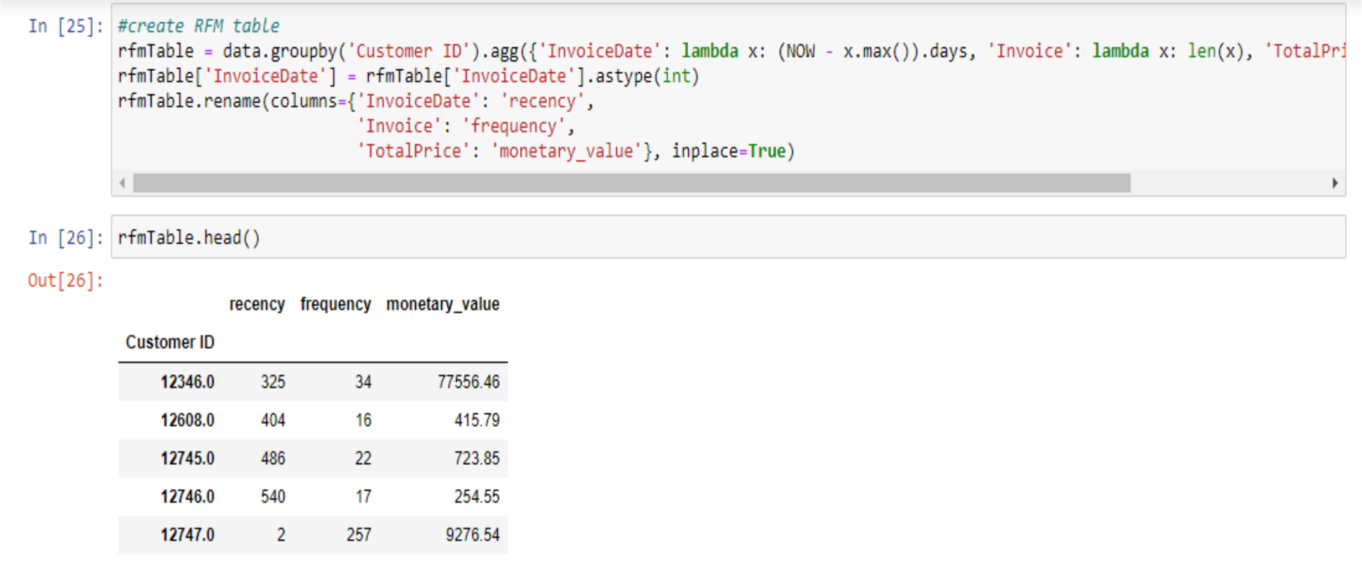
```
In [18]: #checking unique values for each column
         def unique_counts(data):
            for i in data.columns:
                count = data[i].nunique()
                print(i, ": ", count)
         unique_counts(data)

         Invoice :  33546
         StockCode :  4616
         Description :  5249
         Quantity :  405
         InvoiceDate :  31562
         Price :  553
         Customer ID :  5353
         Country :  1

In [20]: data['TotalPrice'] = data['Quantity'] * data['Price'] #Add a column for total price
```



```
In [21]: data['InvoiceDate'].min() #finding first order date in data

Out[21]: '2009-12-01 07:45:00'
```

```
In [22]: data['InvoiceDate'].max() #finding last order date in data

Out[22]: '2011-12-09 12:49:00'
```

```
In [23]: #calculating recency

         import datetime as dt
         NOW = dt.datetime(2011,12,10)
         data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
```

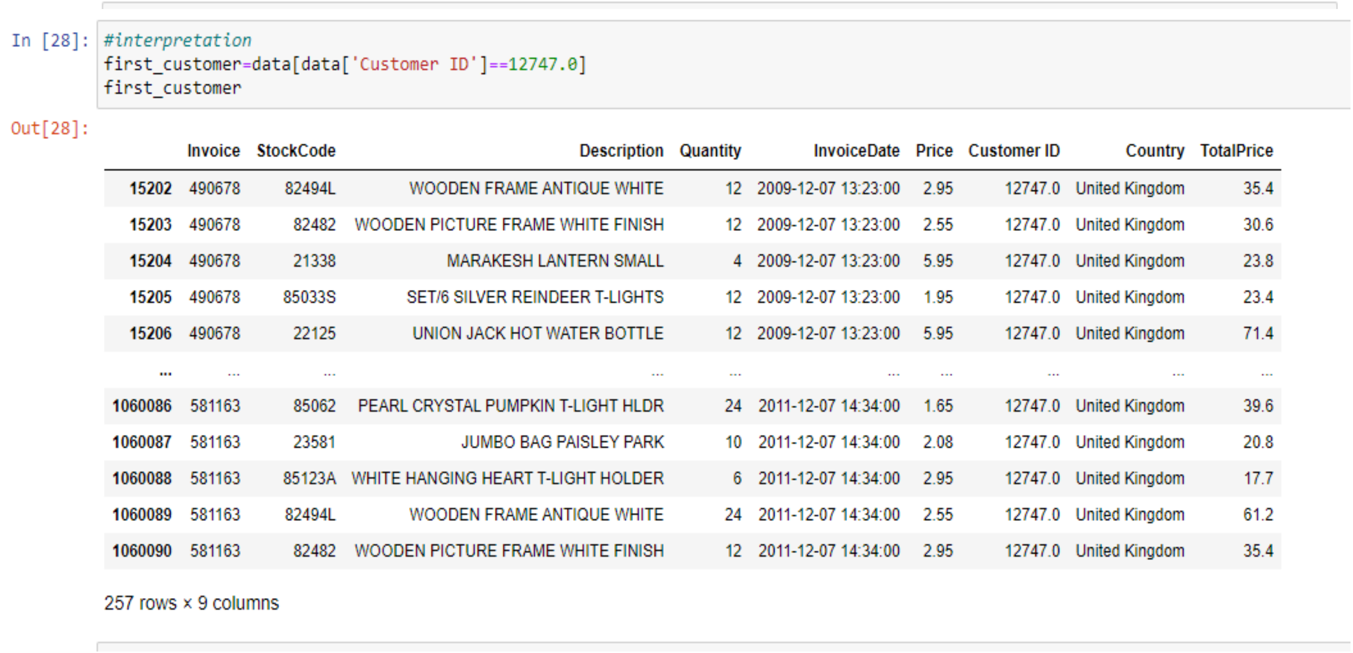


```
In [25]: #create RFM table
         rfmTable = data.groupby('Customer ID').agg({'InvoiceDate': lambda x: (NOW - x.max()).days, 'Invoice': lambda x: len(x), 'TotalPri
         rfmTable['InvoiceDate'] = rfmTable['InvoiceDate'].astype(int)
         rfmTable.rename(columns={'InvoiceDate': 'recency',
                                  'Invoice': 'frequency',
                                  'TotalPrice': 'monetary_value'}, inplace=True)
```

```
In [26]: rfmTable.head()
```

Out[26]:

	recency	frequency	monetary_value
Customer ID			
12346.0	325	34	77556.46
12608.0	404	16	415.79
12745.0	486	22	723.85
12746.0	540	17	254.55
12747.0	2	257	9276.54

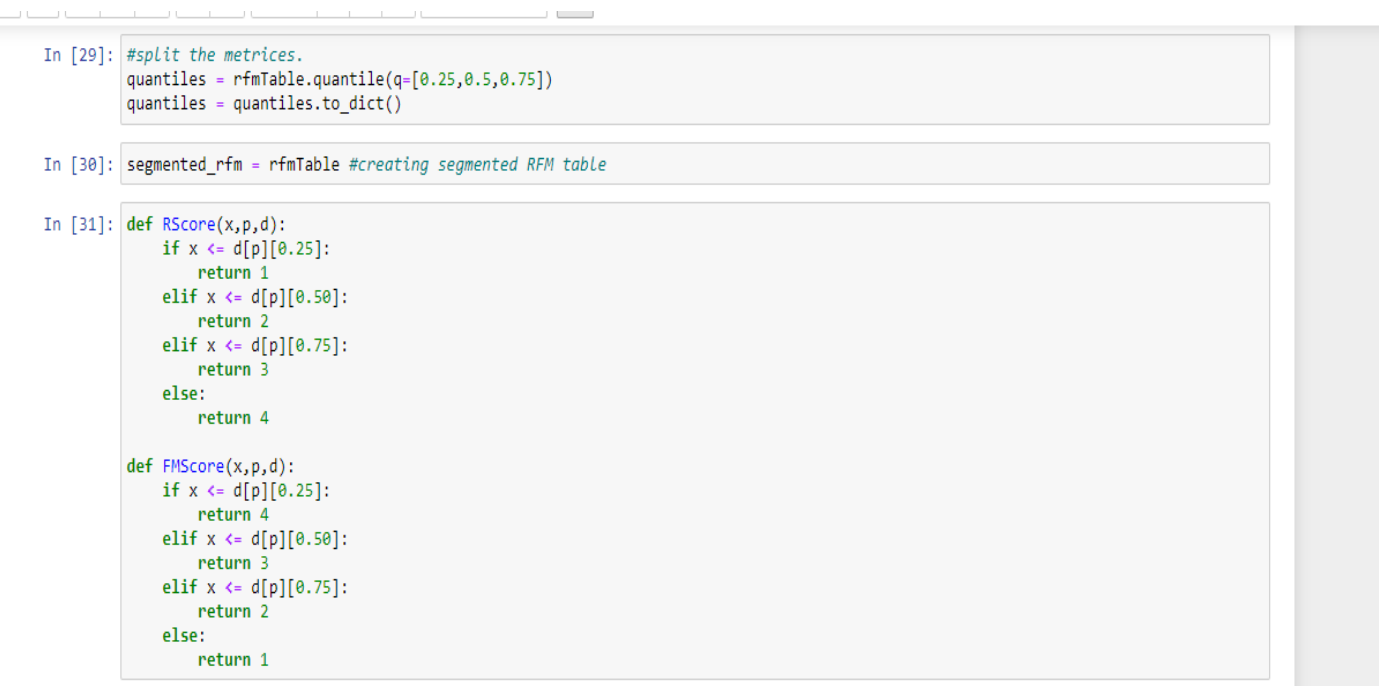


```
In [28]: #interpretation
         first_customer=data[data['Customer ID']==12747.0]
         first_customer
```

Out[28]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	TotalPrice
15202	490678	82494L	WOODEN FRAME ANTIQUE WHITE	12	2009-12-07 13:23:00	2.95	12747.0	United Kingdom	35.4
15203	490678	82482	WOODEN PICTURE FRAME WHITE FINISH	12	2009-12-07 13:23:00	2.55	12747.0	United Kingdom	30.6
15204	490678	21338	MARAKESH LANTERN SMALL	4	2009-12-07 13:23:00	5.95	12747.0	United Kingdom	23.8
15205	490678	85033S	SET/6 SILVER REINDEER T-LIGHTS	12	2009-12-07 13:23:00	1.95	12747.0	United Kingdom	23.4
15206	490678	22125	UNION JACK HOT WATER BOTTLE	12	2009-12-07 13:23:00	5.95	12747.0	United Kingdom	71.4
...	...	...	...	...	...	...	...	...	...
1060086	581163	85062	PEARL CRYSTAL PUMPKIN T-LIGHT HLDR	24	2011-12-07 14:34:00	1.65	12747.0	United Kingdom	39.6
1060087	581163	23581	JUMBO BAG PAISLEY PARK	10	2011-12-07 14:34:00	2.08	12747.0	United Kingdom	20.8
1060088	581163	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2011-12-07 14:34:00	2.95	12747.0	United Kingdom	17.7
1060089	581163	82494L	WOODEN FRAME ANTIQUE WHITE	24	2011-12-07 14:34:00	2.55	12747.0	United Kingdom	61.2
1060090	581163	82482	WOODEN PICTURE FRAME WHITE FINISH	12	2011-12-07 14:34:00	2.95	12747.0	United Kingdom	35.4

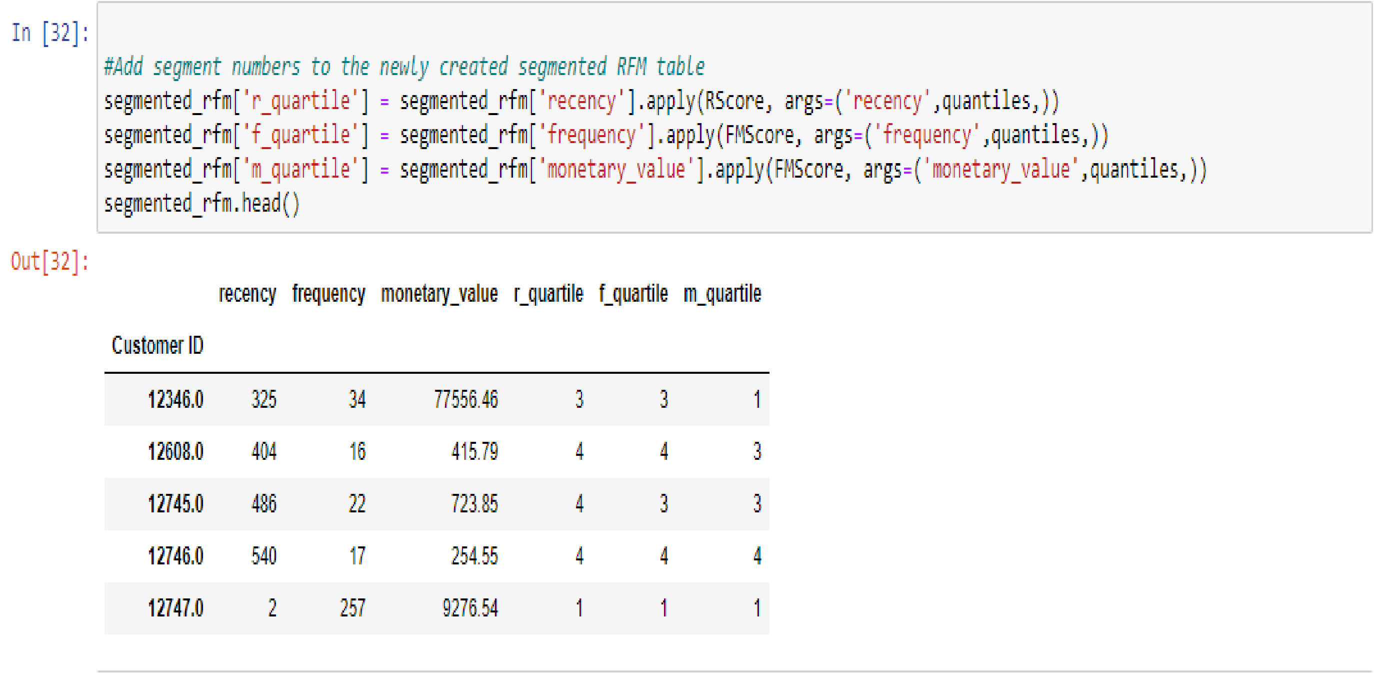
257 rows × 9 columns



In [29]:
```
#split the metrices.
quantiles = rfmTable.quantile(q=[0.25,0.5,0.75])
quantiles = quantiles.to_dict()
```

In [30]:
```
segmented_rfm = rfmTable #creating segmented RFM table
```

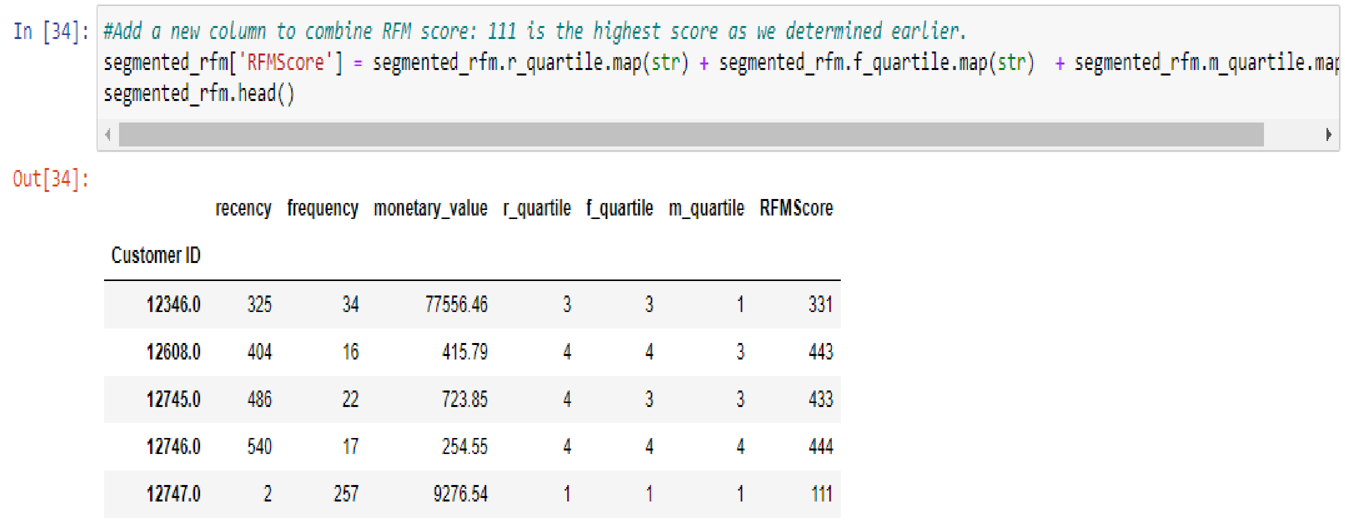
In [31]:
```
def RScore(x,p,d):
    if x <= d[p][0.25]:
        return 1
    elif x <= d[p][0.50]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4

def FMScore(x,p,d):
    if x <= d[p][0.25]:
        return 4
    elif x <= d[p][0.50]:
        return 3
    elif x <= d[p][0.75]:
        return 2
    else:
        return 1
```



In [32]:
```
#Add segment numbers to the newly created segmented RFM table
segmented_rfm['r_quartile'] = segmented_rfm['recency'].apply(RScore, args=('recency',quantiles,))
segmented_rfm['f_quartile'] = segmented_rfm['frequency'].apply(FMScore, args=('frequency',quantiles,))
segmented_rfm['m_quartile'] = segmented_rfm['monetary_value'].apply(FMScore, args=('monetary_value',quantiles,))
segmented_rfm.head()
```

Out[32]:

	recency	frequency	monetary_value	r_quartile	f_quartile	m_quartile
Customer ID						
12346.0	325	34	77556.46	3	3	1
12608.0	404	16	415.79	4	4	3
12745.0	486	22	723.85	4	3	3
12746.0	540	17	254.55	4	4	4
12747.0	2	257	9276.54	1	1	1

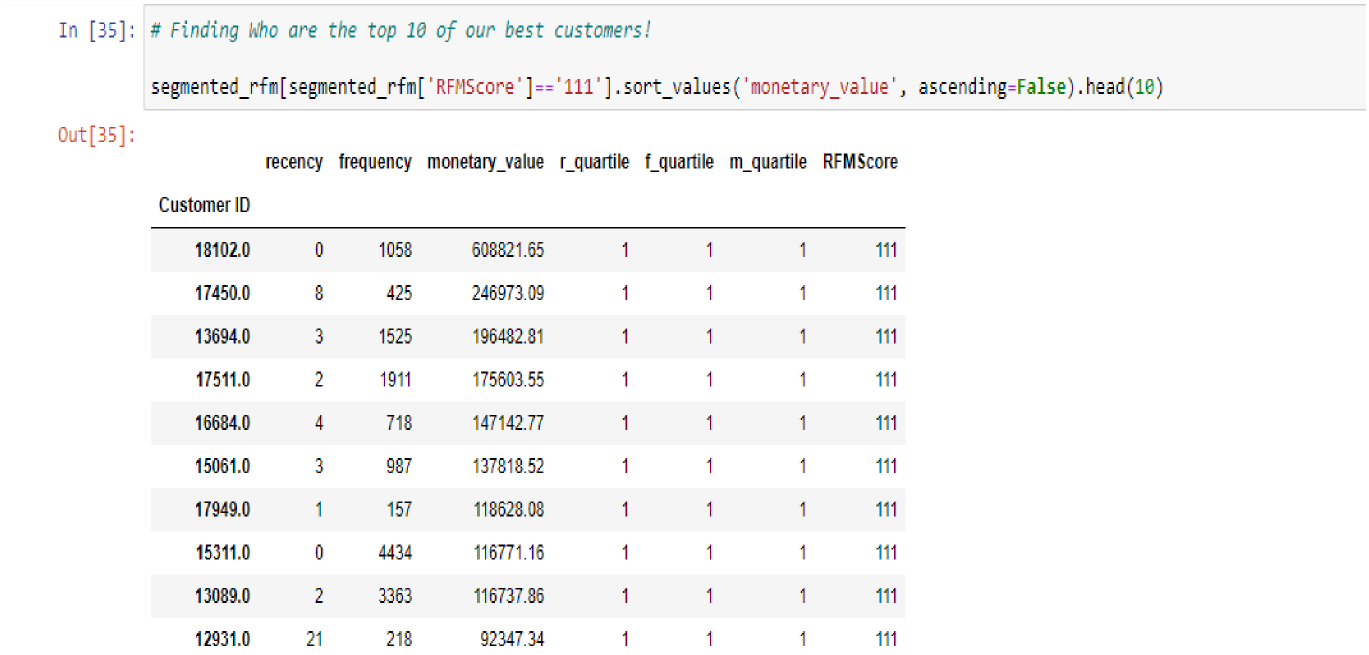


In [34]:

```
#Add a new column to combine RFM score: 111 is the highest score as we determined earlier.
segmented_rfm['RFMScore'] = segmented_rfm.r_quartile.map(str) + segmented_rfm.f_quartile.map(str)  + segmented_rfm.m_quartile.map
segmented_rfm.head()
```

Out[34]:

Customer ID	recency	frequency	monetary_value	r_quartile	f_quartile	m_quartile	RFMScore
12346.0	325	34	77556.46	3	3	1	331
12608.0	404	16	415.79	4	4	3	443
12745.0	486	22	723.85	4	3	3	433
12746.0	540	17	254.55	4	4	4	444
12747.0	2	257	9276.54	1	1	1	111



In [35]:

```
# Finding Who are the top 10 of our best customers!

segmented_rfm[segmented_rfm['RFMScore']=='111'].sort_values('monetary_value', ascending=False).head(10)
```

Out[35]:

Customer ID	recency	frequency	monetary_value	r_quartile	f_quartile	m_quartile	RFMScore
18102.0	0	1058	608821.65	1	1	1	111
17450.0	8	425	246973.09	1	1	1	111
13694.0	3	1525	196482.81	1	1	1	111
17511.0	2	1911	175603.55	1	1	1	111
16684.0	4	718	147142.77	1	1	1	111
15061.0	3	987	137818.52	1	1	1	111
17949.0	1	157	118628.08	1	1	1	111
15311.0	0	4434	116771.16	1	1	1	111
13089.0	2	3363	116737.86	1	1	1	111
12931.0	21	218	92347.34	1	1	1	111