Retail - Capstone project

Project Task: Week 1

Data Cleaning:

1. Perform a preliminary data inspection and data cleaning.

In [4]:

```
# importing necessary libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.colors as mcolors
import matplotlib.pyplot as plt
import datetime as dt
from scipy.stats import skewnorm
import scipy.stats as stats
```

In [5]:

```
# Reading and saving the date in a dataframe

file_path = 'Retail Capstone project/Online Retail.xlsx'
df = pd.read_excel('Retail Capstone project/Online Retail.xlsx')
df.head()
```

Out[5]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

In [6]:

```
df.shape
```

Out[6]:

```
(541909, 8)
```

a. Check for missing data and formulate an apt strategy to treat them.

In [7]:

```
df.isnull().sum()
```

Out[7]:

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

In [8]:

```
df.dropna(inplace=True)
```

In [9]:

```
df.isnull().sum()
```

Out[9]:

```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

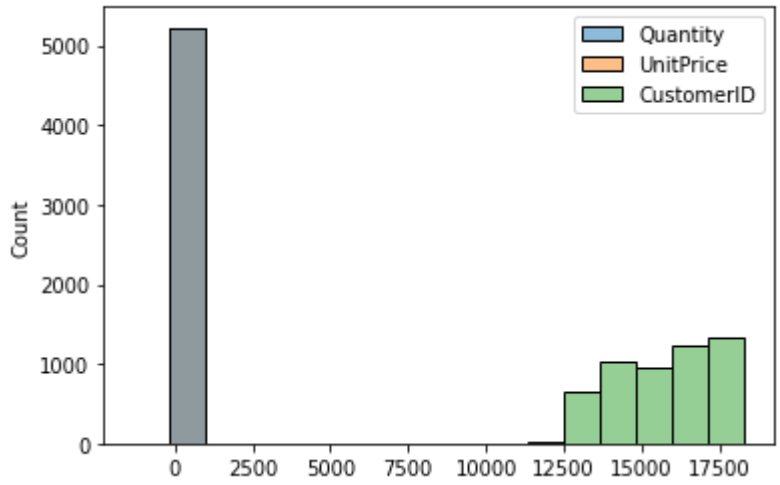
b. Remove duplicate data records.

In [10]:

```
duplicates = df[df.duplicated()]
sns.histplot(data=duplicates)
```

Out[10]:

```
<AxesSubplot:ylabel='Count'>
```



In [11]:

```
df.duplicated().sum()
```

Out[11]:

```
5225
```

In [12]:

```
df.drop_duplicates(inplace=True)
```

In [13]:

```
df.duplicated().sum()
```

Out[13]:

```
0
```

c. Perform descriptive analytics on the given data.

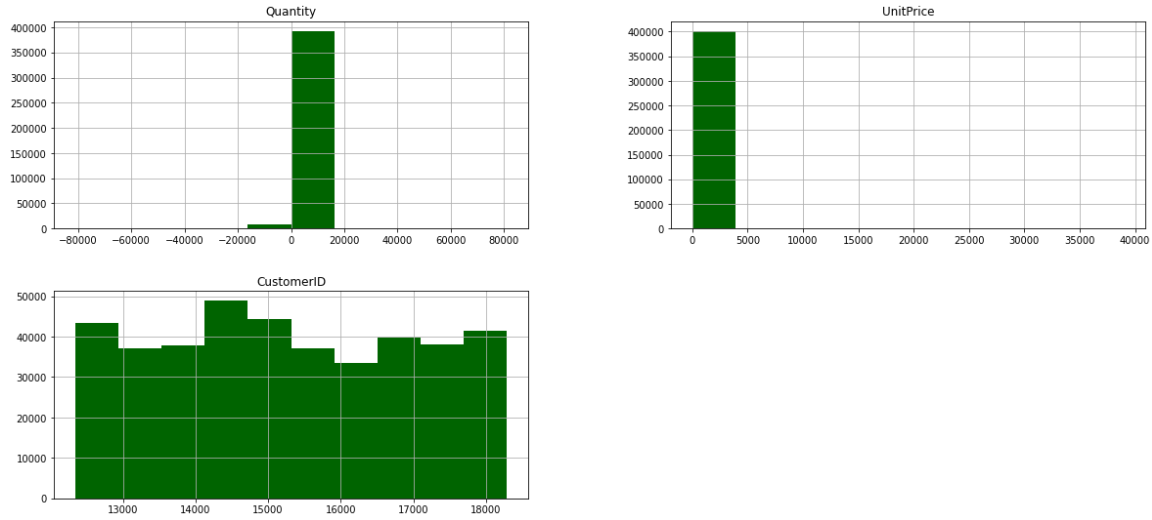
In [14]:

```
# summary statistics
print(df.describe())

df.hist(color='#006400',figsize=(20, 9))
```

```
            Quantity      UnitPrice     CustomerID
count  401604.000000  401604.000000  401604.000000
mean       12.183273       3.474064   15281.160818
std       250.283037      69.764035    1714.006089
min    -80995.000000       0.000000   12346.000000
25%         2.000000       1.250000   13939.000000
50%         5.000000       1.950000   15145.000000
75%        12.000000       3.750000   16784.000000
max     80995.000000   38970.000000   18287.000000
```

Out[14]:

```
array([[<AxesSubplot:title={'center':'Quantity'}>,
        <AxesSubplot:title={'center':'UnitPrice'}>],
       [<AxesSubplot:title={'center':'CustomerID'}>, <AxesSubplot:>]],
      dtype=object)
```



In [130]:

```
# unique values in the Country column

pd.DataFrame(df['Country'].unique())
```

Out[130]:

	0
0	United Kingdom
1	France
2	Australia
3	Netherlands
4	Germany
5	Norway
6	EIRE
7	Switzerland
8	Spain
9	Poland
10	Portugal
11	Italy
12	Belgium
13	Lithuania
14	Japan
15	Iceland
16	Channel Islands
17	Denmark
18	Cyprus
19	Sweden
20	Austria
21	Israel
22	Finland
23	Greece
24	Singapore
25	Lebanon
26	United Arab Emirates
27	Saudi Arabia
28	Czech Republic
29	Canada
30	Unspecified
31	Brazil
32	USA
33	European Community
34	Bahrain
35	Malta

	0
36	RSA

In [15]:

```
# find the number of unique customers
num_customers = df['CustomerID'].nunique()

print(num_customers)
```

```
4372
```

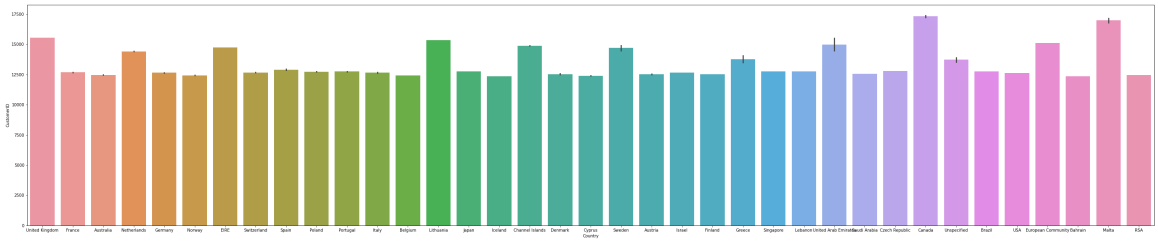
In [16]:

```
# find the number of customers from each country
customer_count_by_country = df.groupby('Country')['CustomerID'].nunique()
customer_count_by_country = customer_count_by_country.sort_values(ascending=False)

print(customer_count_by_country)

fig = plt.figure(figsize=(50,10))
sns.barplot(x='Country', y='CustomerID', data=df, ax=fig.add_subplot(1,1,1))
```

```
Country
United Kingdom          3950
Germany                   95
France                    87
Spain                     31
Belgium                   25
Switzerland               21
Portugal                  19
Italy                     15
Finland                   12
Austria                   11
Norway                    10
Channel Islands            9
Denmark                    9
Australia                  9
Netherlands                9
Cyprus                     8
Japan                      8
Sweden                     8
Poland                     6
Unspecified                4
Israel                     4
Greece                     4
USA                        4
Canada                     4
EIRE                       3
United Arab Emirates       2
Bahrain                    2
Malta                      2
Czech Republic             1
Singapore                  1
Lithuania                  1
Saudi Arabia               1
Brazil                     1
RSA                        1
Iceland                    1
Lebanon                    1
European Community         1
Name: CustomerID, dtype: int64
```

Out[16]:

```
<AxesSubplot:xlabel='Country', ylabel='CustomerID'>
```

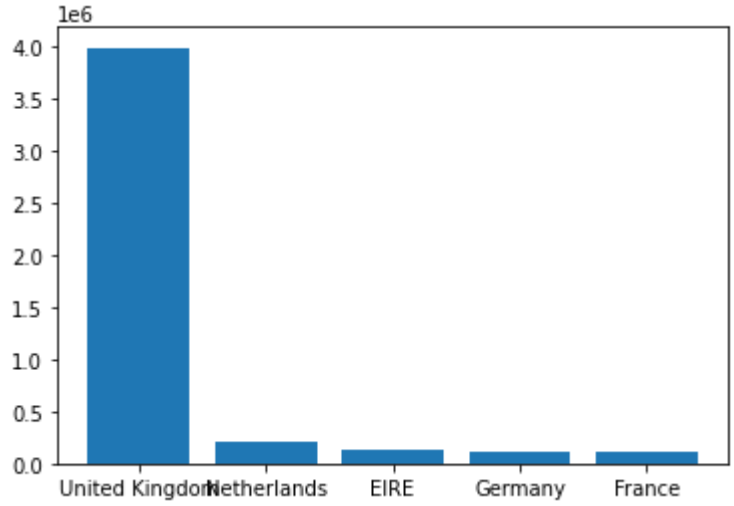


In [18]:

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)

# group the data by country and sum the total purchases in each country
country_totals = df.groupby('Country')['Quantity'].sum()

# sort the country totals in descending order and select the top 5 countries
top_countries = country_totals.sort_values(ascending=False)[:5]
ax.bar(top_countries.index, top_countries.values)
```

Out[18]:

```
<BarContainer object of 5 artists>
```

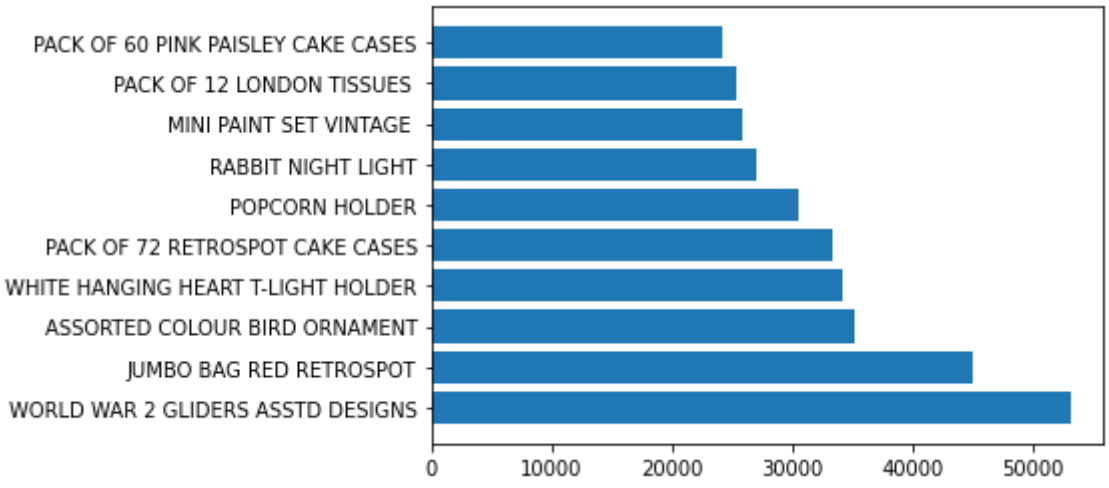


In [19]:

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)

# group the data by item description and sum the total purchases for each item
item_totals = df.groupby('Description')['Quantity'].sum()

# sort the item totals in descending order and select the top 10 items
top_items = item_totals.sort_values(ascending=False)[:10]
ax.barh(top_items.index, top_items.values)
```

Out[19]:

```
<BarContainer object of 10 artists>
```



2. Perform cohort analysis (a cohort is a group of subjects that share a defining characteristic). Observe how a cohort behaves across time and compare it to other cohorts.

a. Create month cohorts and analyze active customers for each cohort.

In [21]:

```
# cohort analysis of customer retention
# period_number column indicates the number of months since the customer's cohort month

from operator import attrgetter

df['order_month'] = df['InvoiceDate'].dt.to_period('M')
df['cohort'] = df.groupby('CustomerID')['InvoiceDate'].transform('min').dt.to_period
('M')
df_cohort=pd.DataFrame(df.groupby(['cohort', 'order_month']).agg(n_customers=('Customer
ID', 'nunique')).reset_index(drop=False))
df_cohort['period_number'] = (df_cohort.order_month - df_cohort.cohort).apply(attrgette
r('n'))
df_cohort
```

Out[21]:

	cohort	order_month	n_customers	period_number
0	2010-12	2010-12	948	0
1	2010-12	2011-01	362	1
2	2010-12	2011-02	317	2
3	2010-12	2011-03	367	3
4	2010-12	2011-04	341	4
...	...	...	...	...
86	2011-10	2011-11	93	1
87	2011-10	2011-12	46	2
88	2011-11	2011-11	321	0
89	2011-11	2011-12	43	1
90	2011-12	2011-12	41	0

91 rows × 4 columns

In [22]:

```
# to create a pivot table that summarizes the number of unique customers in each cohort
-period combination

cohort_pivot = df_cohort.pivot_table(index = 'cohort',columns = 'period_number',values
= 'n_customers')
cohort_pivot
```

Out[22]:

period_number	0	1	2	3	4	5	6	7	8	9	10	
cohort												
2010-12	948.0	362.0	317.0	367.0	341.0	376.0	360.0	336.0	336.0	374.0	354.0	47
2011-01	421.0	101.0	119.0	102.0	138.0	126.0	110.0	108.0	131.0	146.0	155.0	6
2011-02	380.0	94.0	73.0	106.0	102.0	94.0	97.0	107.0	98.0	119.0	35.0	N
2011-03	440.0	84.0	112.0	96.0	102.0	78.0	116.0	105.0	127.0	39.0	NaN	N
2011-04	299.0	68.0	66.0	63.0	62.0	71.0	69.0	78.0	25.0	NaN	NaN	N
2011-05	279.0	66.0	48.0	48.0	60.0	68.0	74.0	29.0	NaN	NaN	NaN	N
2011-06	235.0	49.0	44.0	64.0	58.0	79.0	24.0	NaN	NaN	NaN	NaN	N
2011-07	191.0	40.0	39.0	44.0	52.0	22.0	NaN	NaN	NaN	NaN	NaN	N
2011-08	167.0	42.0	42.0	42.0	23.0	NaN	NaN	NaN	NaN	NaN	NaN	N
2011-09	298.0	89.0	97.0	36.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2011-10	352.0	93.0	46.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2011-11	321.0	43.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2011-12	41.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

b. Analyze the retention rate of customers.

In [23]:

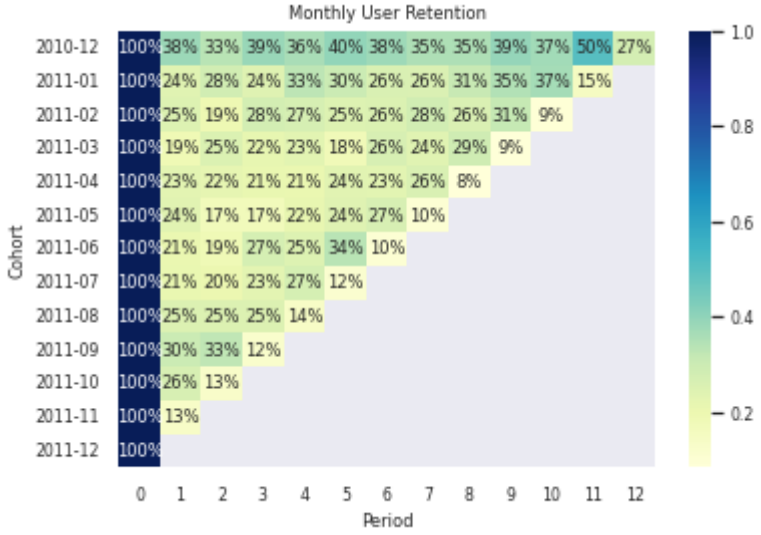
```
# to create a retention matrix that shows the percentage of each cohort that is still a
ctive in each period

cohort_size = cohort_pivot.iloc[:,0]
retention_matrix = cohort_pivot.divide(cohort_size, axis = 0)
retention_matrix
```

Out[23]:

period_number	0	1	2	3	4	5	6	7	
cohort									
2010-12	1.0	0.381857	0.334388	0.387131	0.359705	0.396624	0.379747	0.354430	0.3
2011-01	1.0	0.239905	0.282660	0.242280	0.327791	0.299287	0.261283	0.256532	0.3
2011-02	1.0	0.247368	0.192105	0.278947	0.268421	0.247368	0.255263	0.281579	0.2
2011-03	1.0	0.190909	0.254545	0.218182	0.231818	0.177273	0.263636	0.238636	0.2
2011-04	1.0	0.227425	0.220736	0.210702	0.207358	0.237458	0.230769	0.260870	0.(
2011-05	1.0	0.236559	0.172043	0.172043	0.215054	0.243728	0.265233	0.103943	
2011-06	1.0	0.208511	0.187234	0.272340	0.246809	0.336170	0.102128	NaN	
2011-07	1.0	0.209424	0.204188	0.230366	0.272251	0.115183	NaN	NaN	
2011-08	1.0	0.251497	0.251497	0.251497	0.137725	NaN	NaN	NaN	
2011-09	1.0	0.298658	0.325503	0.120805	NaN	NaN	NaN	NaN	
2011-10	1.0	0.264205	0.130682	NaN	NaN	NaN	NaN	NaN	
2011-11	1.0	0.133956	NaN	NaN	NaN	NaN	NaN	NaN	
2011-12	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

In [25]:

```
# heatmap visualization of the retention_matrix data
# the colors indicate the percentage of each cohort that is still active in each period

sns.set(font_scale=0.7)
sns.heatmap(data=retention_matrix, mask=retention_matrix.isnull(),annot=True, fmt='.
0%', cmap='YlGnBu')
plt.title("Monthly User Retention")
plt.xlabel("Period")
plt.ylabel("Cohort")
plt.show()
```



Findings

50% is the highest retention rate observed

50% of custermer who were active on december 2010 are active 12 months later

Project Task: Week 2

Data Modeling :

1. Build a RFM (Recency Frequency Monetary) model.

2. Calculate RFM metrics.

In [27]:

```
# calculate recency, frequency, and monetary value for each customer
rfm = df.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (pd.Timestamp('now') - x.max()).days,
    'InvoiceNo': 'count',
    'UnitPrice': 'sum'
})

# rename columns
rfm.rename(columns={'InvoiceDate': 'Recency', 'InvoiceNo': 'Frequency', 'UnitPrice': 'M
onetaryValue'}, inplace=True)

# print RFM metrics for each customer
print(rfm)
```

```
            Recency  Frequency  MonetaryValue
CustomerID
12346.0        4364          2           2.08
12347.0        4041        182         481.21
12348.0        4114         31         178.71
12349.0        4057         73         605.10
12350.0        4348         17          65.30
...             ...        ...            ...
18280.0        4316         10          47.65
18281.0        4219          7          39.36
18282.0        4046         13          62.68
18283.0        4042        721        1174.33
18287.0        4081         70         104.55

[4372 rows x 3 columns]
```

3. Build RFM Segments. Give recency, frequency, and monetary scores individually by dividing them into quartiles.

In [28]:

```
# Calculate RFM groups, labels and quartiles with the qcut function

r_labels = list(range(4, 0, -1))
f_labels = list(range(1, 5))
m_labels = list(range(1, 5))

# Assign these labels to 4 equal percentile groups
r_groups = pd.qcut(rfm['Recency'], q=4, labels=r_labels)

# Assign these labels to 4 equal percentile groups
f_groups = pd.qcut(rfm['Frequency'], q=4, labels=f_labels)

# Assign these labels to four equal percentile groups
m_groups = pd.qcut(rfm['MonetaryValue'], q=4, labels=m_labels)
```

In [141]:

```
print(r_groups)
print(f_groups)
print(m_groups)
```

```
CustomerID
12346.0    1
12347.0    4
12348.0    2
12349.0    3
12350.0    1
          ..
18280.0    1
18281.0    1
18282.0    4
18283.0    4
18287.0    3
Name: Recency, Length: 4372, dtype: category
Categories (4, int64): [4 < 3 < 2 < 1]
CustomerID
12346.0    1
12347.0    4
12348.0    2
12349.0    3
12350.0    1
          ..
18280.0    1
18281.0    1
18282.0    1
18283.0    4
18287.0    3
Name: Frequency, Length: 4372, dtype: category
Categories (4, int64): [1 < 2 < 3 < 4]
CustomerID
12346.0    1
12347.0    4
12348.0    3
12349.0    4
12350.0    2
          ..
18280.0    1
18281.0    1
18282.0    2
18283.0    4
18287.0    2
Name: MonetaryValue, Length: 4372, dtype: category
Categories (4, int64): [1 < 2 < 3 < 4]
```

In [29]:

```
# Adding the new columns to original rmf

rfm = rfm.assign(R = r_groups.values, F = f_groups.values, M = m_groups.values)
rfm.head()
```

Out[29]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M
12346.0	4364	2	2.08	1	1	1
12347.0	4041	182	481.21	4	4	4
12348.0	4114	31	178.71	2	2	3
12349.0	4057	73	605.10	3	3	4
12350.0	4348	17	65.30	1	1	2

b1. Combine three ratings to get a RFM segment (as strings).

In [31]:

```
rfm['RFM_segment'] = rfm.apply(lambda x: (str(x['R']) + str(x['F']) + str(x['M'])), axis=1)
rfm.head()
```

Out[31]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_Score	RFM_segment
12346.0	4364	2	2.08	1	1	1	3	1.01.01.0
12347.0	4041	182	481.21	4	4	4	12	4.04.04.0
12348.0	4114	31	178.71	2	2	3	7	2.02.03.0
12349.0	4057	73	605.10	3	3	4	10	3.03.04.0
12350.0	4348	17	65.30	1	1	2	4	1.01.02.0

b2. Get the RFM score by adding up the three ratings.

In [32]:

```
# the sum of the values in the R, F, and M columns

rfm['RFM_Score'] = rfm[['R','F','M']].sum(axis = 1)
rfm['RFM_Score'] = rfm['RFM_Score'].apply(lambda x : int(x))
rfm.head()
```

Out[32]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_Score	RFM_segment
12346.0	4364	2	2.08	1	1	1	3	1.01.01.0
12347.0	4041	182	481.21	4	4	4	12	4.04.04.0
12348.0	4114	31	178.71	2	2	3	7	2.02.03.0
12349.0	4057	73	605.10	3	3	4	10	3.03.04.0
12350.0	4348	17	65.30	1	1	2	4	1.01.02.0

In [33]:

```
# Creating a function to define rfm_level function on the basis of importance

def rfm_level(df):
    if df['RFM_Score'] >= 9:
        return 'Important'
    elif ((df['RFM_Score'] >= 8) and (df['RFM_Score'] < 9)):
        return 'Good'
    elif ((df['RFM_Score'] >= 7) and (df['RFM_Score'] < 8)):
        return 'Okay'
    elif ((df['RFM_Score'] >= 6) and (df['RFM_Score'] < 7)):
        return 'Neutral'
    elif ((df['RFM_Score'] >= 5) and (df['RFM_Score'] < 6)):
        return 'Might'
    elif ((df['RFM_Score'] >= 4) and (df['RFM_Score'] < 5)):
        return 'Needs Attention'
    else:
        return 'Activate'

# Create a new variable RFM_Level
rfm['RFM_Level'] = rfm.apply(rfm_level, axis=1)
rfm.head()
```

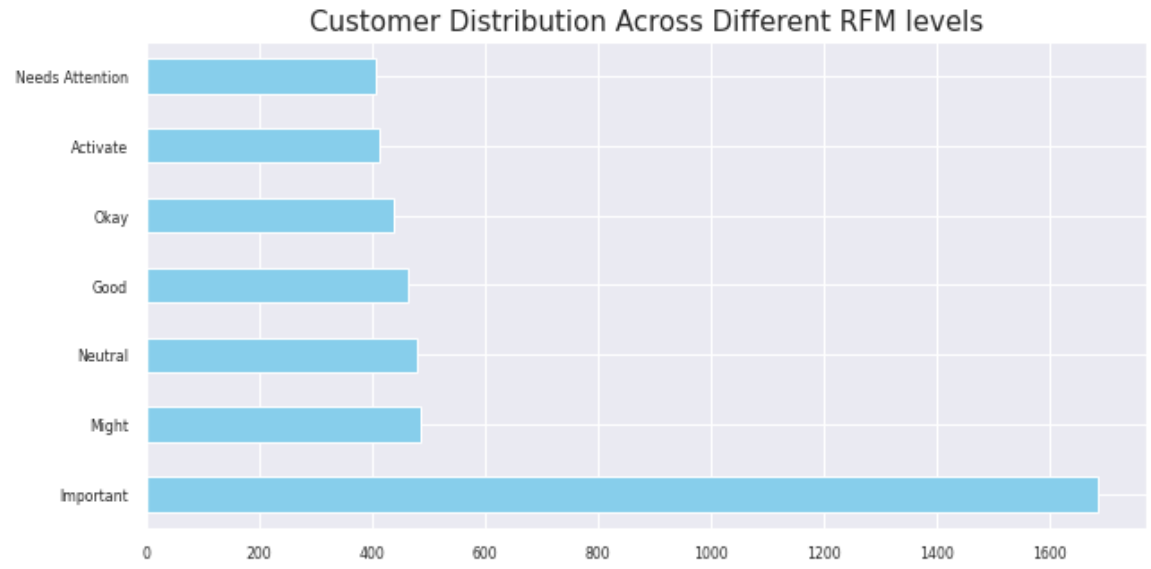
Out[33]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_Score	RFM_segment	RFM_
12346.0	4364	2	2.08	1	1	1	3	1.01.01.0	A
12347.0	4041	182	481.21	4	4	4	12	4.04.04.0	Imp
12348.0	4114	31	178.71	2	2	3	7	2.02.03.0	
12349.0	4057	73	605.10	3	3	4	10	3.03.04.0	Imp
12350.0	4348	17	65.30	1	1	2	4	1.01.02.0	I Att

Analyze the RFM segments by summarizing them and comment on the findings.

In [35]:

```
# The RFM levels are based on the values in the R, F, and M columns of the rfm DataFram
e
# the chart allows you to quickly and easily see how many customers fall into each leve
l

rfm['RFM_Level'].value_counts().plot(kind = 'barh', figsize = (10, 5), color = 'skyblu
e')
plt.title('Customer Distribution Across Different RFM levels', fontsize = 15)
plt.show()
```



Findings :

Most of the customers 1600 plus are in the important RFM_segment

Data Modeling :

1. Create clusters using k-means clustering algorithm.

a. Prepare the data for the algorithm. If the data is asymmetrically distributed, manage the skewness with appropriate transformation. Standardize the data.

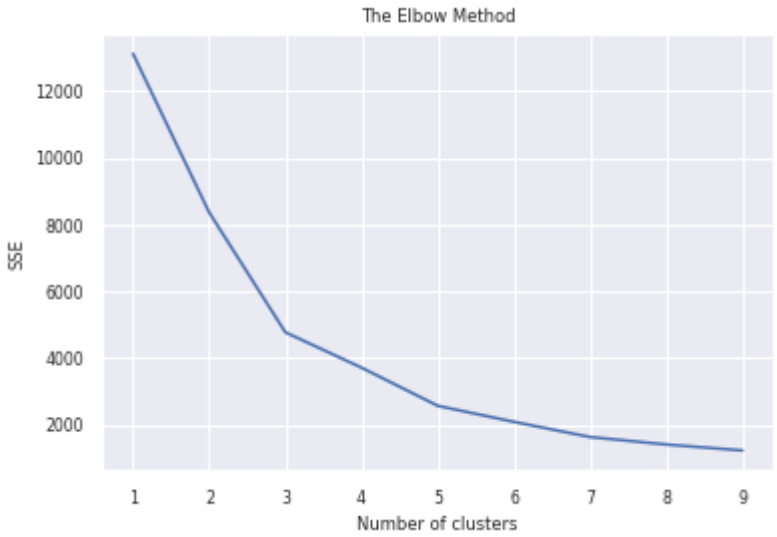
In [36]:

```
# importing necessary libraries

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# transform and standardize the data
scaler = StandardScaler()
X = scaler.fit_transform(rfm[['Recency', 'Frequency', 'MonetaryValue']])
```

b. Decide the optimum number of clusters to be formed.

In [37]:

```
# determine the optimum number of clusters using the elbow method

sse = []
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(X)
    sse.append(kmeans.inertia_)
plt.plot(range(1, 10), sse)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
plt.show()
```



The optimum number of cluster are 3.

c. Analyze these clusters and comment on the results.

In [39]:

```
# Fit the K-means algorithm with 3 clusters
kmeans = KMeans(n_clusters = 3)
kmeans.fit(X)

# Assign cluster labels to each customer
rfm['cluster'] = kmeans.labels_
centroids = kmeans.cluster_centers_

print(rfm.groupby('cluster')['Recency', 'Frequency', 'MonetaryValue'].mean())
```

```
              Recency    Frequency  MonetaryValue
cluster
0         4078.664833   106.322640     333.283832
1         4285.974359    27.297619      98.730972
2         4078.285714  3400.142857   28077.950000

/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:9: FutureWarn
ing: Indexing with multiple keys (implicitly converted to a tuple of keys)
will be deprecated, use a list instead.
  if __name__ == '__main__':
```

In [44]:

```
kmeans = KMeans(n_clusters = 3, init = 'k-means++')
y_kmeans = kmeans.fit_predict(X)
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 5, c = 'red', label = 'Lost/i
nactive Customer')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 5, c = 'blue', label = 'Loyal
customer')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 5, c = 'green', label = 'Aver
age Customers')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 20, c =
'black', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Total Spending')
plt.ylabel('Buying Frequency')
plt.legend()
plt.show()
```