SEDISMART Supply Chain Data Analytics

Problem Statement

SEDISMART is an imaginary office furniture & equipment company based in UK, and its management executives have a meeting soon in which they discuss the business status. The executives would like to see:

- The general KPIs and business metrics of the business (CEO board).
- The performance of the sales-people (HR team).
- The performance of their customers (marketing team).
- The performance of their products (sales team).
- The performance of their suppliers (logistic team).
- The forecast of the revenue for the first quarter of this year - 2017 (finance team).

Data Collection

In [1]:
```
from numpy import log1p,sqrt

import pandas as pd
pd.options.display.float_format = '{:,.2f}'.format

import ipywidgets
import inventorize3 as inv
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
%matplotlib inline

import folium
```

In [2]:
```
sales = pd.read_excel('../data/Sales_Data.xlsx', sheet_name='SLS')
purchases = pd.read_excel('../data/Sales_Data.xlsx',sheet_name='PRCHS')
branches = pd.read_excel('../data/Sales_Data.xlsx',sheet_name='BUDGT')
products = pd.read_excel('../data/Sales_Data.xlsx',sheet_name='PRODS')
staff = pd.read_excel('../data/Sales_Data.xlsx',sheet_name='STFF')
customers = pd.read_excel('../data/Sales_Data.xlsx',sheet_name='CLNTS')
```

In [3]:
```
tables = [(sales,'sales'),(purchases,'purchases'),(branches,'branches'),(products
          (staff,'staff'),(customers,'customers')]

for table, table_name in tables:
    print(f'======{table_name.capitalize()} Table Info======')
    print(table.info())
```

```
======Sales Table Info======
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 279329 entries, 0 to 279328
Data columns (total 12 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   ItemSalesID   279329 non-null  int64
 1   InvoiceID     279329 non-null  int64
 2   ClientID      279329 non-null  int64
 3   StaffID       279329 non-null  int64
 4   Invoice Date  279329 non-null  datetime64[ns]
 5   Payment Date  279329 non-null  datetime64[ns]
 6   Quantity      279329 non-null  int64
 7   ProductID     279329 non-null  int64
 8   Unit Price    279329 non-null  float64
 9   Discount      279329 non-null  float64
 10  Purchase ID   279329 non-null  object
 11  Year          279329 non-null  int64
dtypes: datetime64[ns](2), float64(2), int64(7), object(1)
memory usage: 25.6+ MB
None
======Purchases Table Info======
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20785 entries, 0 to 20784
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ItemPurchasesID  20785 non-null  int64
 1   ProductID        20785 non-null  int64
 2   PurchaseID       20785 non-null  object
 3   Purchase Date    20785 non-null  datetime64[ns]
 4   Payment Date     20785 non-null  datetime64[ns]
 5   Quantity         20785 non-null  int64
 6   Unit Price       20785 non-null  float64
dtypes: datetime64[ns](2), float64(1), int64(3), object(1)
memory usage: 1.1+ MB
None
======Branches Table Info======
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78 entries, 0 to 77
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Branch  78 non-null     object
 1   Year    78 non-null     int64
 2   Budget  78 non-null     float64
dtypes: float64(1), int64(1), object(1)
memory usage: 2.0+ KB
None
======Products Table Info======
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 66 entries, 0 to 65
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   productid      66 non-null     int64  
 1   Product        66 non-null     object 
 2   Product Range  66 non-null     object 
 3   Current RRP    66 non-null     float64
 4   Supplier Name  66 non-null     object 
 5   Supplier Lat   66 non-null     float64
 6   Supplier Long  66 non-null     float64
 7   Complexity     66 non-null     float64
dtypes: float64(4), int64(1), object(3)
memory usage: 4.2+ KB
None
======Staff Table Info======
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 67 entries, 0 to 66
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   StaffID   67 non-null     int64 
 1   Title     67 non-null     object
 2   Name      67 non-null     object
 3   Surname   67 non-null     object
 4   Role      67 non-null     object
 5   Branch    67 non-null     object
 6   Country   67 non-null     object
 7   Username  67 non-null     object
dtypes: int64(1), object(7)
memory usage: 4.3+ KB
None
======Customers Table Info======
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 468 entries, 0 to 467
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   ClientID       468 non-null    int64 
 1   Customer Name  468 non-null    object
 2   Address 1      468 non-null    object
 3   Address 2      149 non-null    object
 4   Address 3      465 non-null    object
 5   Postcode       468 non-null    object
 6   Annual Income  468 non-null    int64 
dtypes: int64(2), object(5)
memory usage: 25.7+ KB
None
```

Data Cleaning

In [4]:
```
# I would like to have one column for the address in the customers table.

# wrtie a function to concatenate the 3 columns of 'Address 1', 'Address 2' and
def address(row):
    address_1 = row[2]
    address_2 = row[3]
    address_3 = row[4]
    if isinstance(address_2, str) & isinstance(address_3,str):
        return (address_1+', '+address_2+', '+address_3)
    elif isinstance(address_2,str):
        return (address_1+', '+address_2)
    elif isinstance(address_3,str):
        return (address_1+', '+address_3)
    else:
        return address_1
```

In [5]:
```
# apply the address function on the customers table
customers['Address'] = customers.apply(address,axis=1)
```

In [6]:
```
# drop the 'Address 1',' Address 2' and 'Address 3' columns from customer
customers.drop(['Address 1','Address 2','Address 3'], axis=1, inplace=True)
```

In [7]:
```
# change the name of the column 'Unit Price' in the purcahses dataframe to 'Unit
purchases.rename(columns={'Unit Price':'Unit Cost'}, inplace=True)
```

In order to merge the branches dataframe with the sales dataframe, I'll first need to merge the staff datafrrame with the sales dataframe, then I'll use the both 'Year' and 'StaffID' to merge the resulted dataframe with the branches dataframe. Therefore, I need first to establish a 'Year' columns in the sales dataframe. I'll use the 'Invoice Date' columns for that. The 'Year' column should be of a period type so it can be used lated for time intelligence analysis.

In [8]:
```
sales['Year'] = sales['Invoice Date'].dt.year
sales['Year'] = pd.to_datetime(sales['Year'], format='%Y')
sales['Year'] = sales['Year'].dt.to_period(freq='Y')
```

In [9]:
```
# change the 'Year' from object to datetime type
branches['Year'] = pd.to_datetime(branches['Year'],format='%Y')
# change the 'Year' column from datetime to period type
branches['Year'] = branches['Year'].dt.to_period(freq='Y')
```

In [10]:
```
sales['Period'] = sales['Invoice Date'].dt.to_period('M')
```

In order to merge the sales dataframe with purchases dataframe later, I need to establish a composite key in both tables. The composite key can be established by joining the product id and the purchase id in one column "Product Purchase".

In [11]:
```
# sales['Product Purchase'] = sales['ProductID'].astype(str) + sales['Purchase ID
# purchases['Product Purchase'] = purchases['ProductID'].astype(str) + purchases[
```

some of the dataframes has similar columns names and when merge them togther suffix 'x' and 'y'

are going to be added to differentiate between the columns. I prefer to add the name of the dataframe before merging to be clearer

In [12]:
```
# remeber the tables list which has the dataframes and their name
for table, table_name in tables:
    table.columns = [table_name.capitalize()+'-'+col for col in table.columns]
```

Cool! Now, I'll merge the dataframes togther.

In [13]:
```
purchases.columns
```

Out[13]:
```
Index(['Purchases-ItemPurchasesID', 'Purchases-ProductID',
       'Purchases-PurchaseID', 'Purchases-Purchase Date',
       'Purchases-Payment Date', 'Purchases-Quantity', 'Purchases-Unit Cost'],
      dtype='object')
```

In [14]:
```
data = sales.merge(purchases, left_on=['Sales-ProductID','Sales-Purchase ID'],
                              right_on=['Purchases-ProductID','Purchases-Purchase
               merge(products,left_on='Sales-ProductID',right_on='Products-product
               merge(customers, left_on='Sales-ClientID',right_on='Customers-Clien
               merge(staff, left_on = 'Sales-StaffID', right_on = 'Staff-StaffID')
               merge(branches, left_on = ['Sales-Year','Staff-Branch'],
                             right_on=['Branches-Year','Branches-Branch'])



data.head()
```

Out[14]:

	Sales-ItemSalesID	Sales-InvoiceID	Sales-ClientID	Sales-StaffID	Sales-Invoice Date	Sales-Payment Date	Sales-Quantity	Sales-ProductID	Sales-Unit Price	Sales Discou
0	547688	3121	133	19	2011-01-03	2011-01-03	3	58	88.75	0.1
1	585858	13601	133	19	2011-11-17	2011-12-18	11	58	88.75	0.2
2	547689	3121	133	19	2011-01-03	2011-01-03	13	66	82.00	0.2
3	581203	12322	133	19	2011-09-26	2011-11-12	2	66	82.00	0.1
4	547690	3121	133	19	2011-01-03	2011-01-03	12	30	55.50	0.1

5 rows × 44 columns

In [15]:
```
data.isna().mean()
```

Out[15]:
```
Sales-ItemSalesID          0.00
Sales-InvoiceID            0.00
Sales-ClientID             0.00
Sales-StaffID              0.00
Sales-Invoice Date         0.00
Sales-Payment Date         0.00
Sales-Quantity             0.00
Sales-ProductID            0.00
Sales-Unit Price           0.00
Sales-Discount             0.00
Sales-Purchase ID          0.00
Sales-Year                 0.00
Sales-Period               0.00
Purchases-ItemPurchasesID  0.00
Purchases-ProductID        0.00
Purchases-PurchaseID       0.00
Purchases-Purchase Date    0.00
Purchases-Payment Date     0.00
Purchases-Quantity         0.00
Purchases-Unit Cost        0.00
Products-productid         0.00
Products-Product           0.00
Products-Product Range     0.00
Products-Current RRP       0.00
Products-Supplier Name     0.00
Products-Supplier Lat      0.00
Products-Supplier Long     0.00
Products-Complexity        0.00
Customers-ClientID         0.00
Customers-Customer Name    0.00
Customers-Postcode         0.00
Customers-Annual Income    0.00
Customers-Address          0.00
Staff-StaffID              0.00
Staff-Title                0.00
Staff-Name                 0.00
Staff-Surname              0.00
Staff-Role                 0.00
Staff-Branch               0.00
Staff-Country              0.00
Staff-Username             0.00
Branches-Branch            0.00
Branches-Year              0.00
Branches-Budget            0.00
dtype: float64
```

Adding the "Revene" Columns

In [16]:
```
data['Revenue'] = data['Sales-Unit Price'] * data['Sales-Quantity'] * (1-data['Sa
```

Adding the "COGS" Column

In [17]:
```
data['COGS'] = data['Sales-Quantity']*data['Purchases-Unit Cost']
```

Adding the Gross "Gross Profit" Column

In [18]:
```
data['Gross Profit'] = data['Revenue'] - data['COGS']
```

In [19]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 279329 entries, 0 to 279328
Data columns (total 47 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   Sales-ItemSalesID           279329 non-null  int64
 1   Sales-InvoiceID             279329 non-null  int64
 2   Sales-ClientID              279329 non-null  int64
 3   Sales-StaffID               279329 non-null  int64
 4   Sales-Invoice Date          279329 non-null  datetime64[ns]
 5   Sales-Payment Date          279329 non-null  datetime64[ns]
 6   Sales-Quantity              279329 non-null  int64
 7   Sales-ProductID             279329 non-null  int64
 8   Sales-Unit Price            279329 non-null  float64
 9   Sales-Discount              279329 non-null  float64
 10  Sales-Purchase ID           279329 non-null  object
 11  Sales-Year                  279329 non-null  period[A-DEC]
 12  Sales-Period                279329 non-null  period[M]
 13  Purchases-ItemPurchasesID   279329 non-null  int64
 14  Purchases-ProductID         279329 non-null  int64
 15  Purchases-PurchaseID        279329 non-null  object
 16  Purchases-Purchase Date     279329 non-null  datetime64[ns]
 17  Purchases-Payment Date      279329 non-null  datetime64[ns]
 18  Purchases-Quantity          279329 non-null  int64
 19  Purchases-Unit Cost         279329 non-null  float64
 20  Products-productid          279329 non-null  int64
 21  Products-Product            279329 non-null  object
 22  Products-Product Range      279329 non-null  object
 23  Products-Current RRP        279329 non-null  float64
 24  Products-Supplier Name      279329 non-null  object
 25  Products-Supplier Lat       279329 non-null  float64
 26  Products-Supplier Long      279329 non-null  float64
 27  Products-Complexity         279329 non-null  float64
 28  Customers-ClientID          279329 non-null  int64
 29  Customers-Customer Name     279329 non-null  object
 30  Customers-Postcode          279329 non-null  object
 31  Customers-Annual Income     279329 non-null  int64
 32  Customers-Address           279329 non-null  object
 33  Staff-StaffID               279329 non-null  int64
 34  Staff-Title                 279329 non-null  object
 35  Staff-Name                  279329 non-null  object
 36  Staff-Surname               279329 non-null  object
 37  Staff-Role                  279329 non-null  object
 38  Staff-Branch                279329 non-null  object
 39  Staff-Country               279329 non-null  object
 40  Staff-Username              279329 non-null  object
 41  Branches-Branch             279329 non-null  object
 42  Branches-Year               279329 non-null  period[A-DEC]
 43  Branches-Budget             279329 non-null  float64
 44  Revenue                     279329 non-null  float64
 45  COGS                        279329 non-null  float64
 46  Gross Profit                279329 non-null  float64
dtypes: datetime64[ns](4), float64(11), int64(13), object(16), period[A-DEC]
(2), period[M](1)
memory usage: 102.3+ MB
```

Data Storage Optimization

In [20]:
```
data.memory_usage().sum()/(1024*1024)
```

Out[20]:
```
102.2933349609375
```

It can be seen that the data dataframe is occupying more than 60 MB in the RAM. To lessen that space, I'll convert the ID columns which are of integer type and the string type columns to a category type columns

In [21]:
```
data.drop(['Purchases-PurchaseID','Sales-Purchase ID'],axis=1,inplace=True)
```

In [22]:
```
id_columns = [col for col in data.columns if 'ID' in col or 'id' in col]
text_columns = [col for col in data.columns if data[col].dtype =='O' if col not i
# text_columns = data.select_dtypes(include='O').columns
```

In [23]:
```
text_columns.extend(id_columns)
```

In [24]:
```
for col in text_columns:
    data[col] = data[col].astype('category')
```

In [25]:
```
data.memory_usage().sum()/(1024*1024)
```

Out[25]:
```
70.02649688720703
```

In [26]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 279329 entries, 0 to 279328
Data columns (total 45 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   Sales-ItemSalesID           279329 non-null  category
 1   Sales-InvoiceID             279329 non-null  category
 2   Sales-ClientID              279329 non-null  category
 3   Sales-StaffID               279329 non-null  category
 4   Sales-Invoice Date          279329 non-null  datetime64[ns]
 5   Sales-Payment Date          279329 non-null  datetime64[ns]
 6   Sales-Quantity              279329 non-null  int64
 7   Sales-ProductID             279329 non-null  category
 8   Sales-Unit Price            279329 non-null  float64
 9   Sales-Discount              279329 non-null  float64
 10  Sales-Year                  279329 non-null  period[A-DEC]
 11  Sales-Period                279329 non-null  period[M]
 12  Purchases-ItemPurchasesID   279329 non-null  category
 13  Purchases-ProductID         279329 non-null  category
 14  Purchases-Purchase Date     279329 non-null  datetime64[ns]
 15  Purchases-Payment Date      279329 non-null  datetime64[ns]
 16  Purchases-Quantity          279329 non-null  int64
 17  Purchases-Unit Cost         279329 non-null  float64
 18  Products-productid          279329 non-null  category
 19  Products-Product            279329 non-null  category
 20  Products-Product Range      279329 non-null  category
 21  Products-Current RRP        279329 non-null  float64
 22  Products-Supplier Name      279329 non-null  category
 23  Products-Supplier Lat       279329 non-null  float64
 24  Products-Supplier Long      279329 non-null  float64
 25  Products-Complexity         279329 non-null  float64
 26  Customers-ClientID          279329 non-null  category
 27  Customers-Customer Name     279329 non-null  category
 28  Customers-Postcode          279329 non-null  category
 29  Customers-Annual Income     279329 non-null  int64
 30  Customers-Address           279329 non-null  category
 31  Staff-StaffID               279329 non-null  category
 32  Staff-Title                 279329 non-null  category
 33  Staff-Name                  279329 non-null  category
 34  Staff-Surname               279329 non-null  category
 35  Staff-Role                  279329 non-null  category
 36  Staff-Branch                279329 non-null  category
 37  Staff-Country               279329 non-null  category
 38  Staff-Username              279329 non-null  category
 39  Branches-Branch             279329 non-null  category
 40  Branches-Year               279329 non-null  period[A-DEC]
 41  Branches-Budget             279329 non-null  float64
 42  Revenue                     279329 non-null  float64
 43  COGS                        279329 non-null  float64
 44  Gross Profit                279329 non-null  float64
dtypes: category(24), datetime64[ns](4), float64(11), int64(3), period[A-DEC]
(2), period[M](1)
memory usage: 70.0 MB
```

Preparing the lists for the widgets of branches, years and product-ranges which I will use in the dynamic analysis

In [27]:
```
# making a list for years to be used for ipywidgets
years_list = sorted(data['Sales-Year'].value_counts().index.tolist())
years_list.insert(0,'All')

# making a list for banches to be used for ipywidgets
branches_list = sorted(data['Staff-Branch'].value_counts().index.tolist())
branches_list.insert(0,'All')

# making a list for product-ranges to be used for ipywidgets
product_ranges_list = sorted(data['Products-Product Range'].value_counts().index.
product_ranges_list.insert(0,'All')

# making a list for products to be used for ipywidgets
products_list = sorted(data['Products-Product'].value_counts().index.tolist())
products_list.insert(0,'All')
```

Cool! Now, it is time for analysis.

General Business Metrics

Total Orders

In [28]:
```
# total_orders = len(data['Sales-InvoiceID'].value_counts().index)
total_orders = len(data['Sales-InvoiceID'].drop_duplicates())
print(f'Total Orders = {total_orders:,} orders')
```

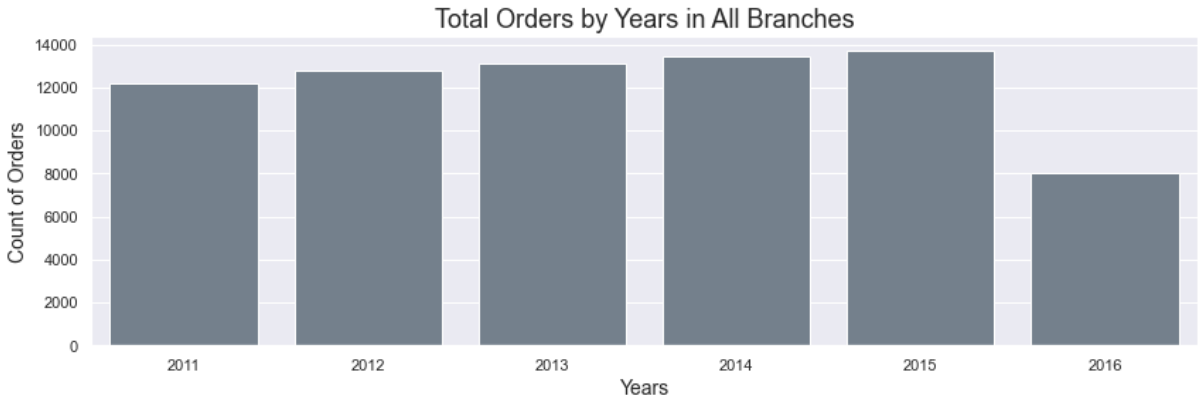
```
Total Orders = 73,203 orders
```

In [29]:

```
slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All'


def select(branch):
    df = data[['Sales-Year','Staff-Branch','Sales-InvoiceID']].copy()
    if branch != 'All':
        df = df[df['Staff-Branch'] == branch]
    orders_years = df.groupby('Sales-Year').\
                    agg({'Sales-InvoiceID':'nunique'}).\
                    rename(columns={'Sales-InvoiceID':'Count of Orders'})
    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = orders_years.index,
                     y = 'Count of Orders',
                     data = orders_years,
                     order=orders_years.index,
                     color='#708090')
    if branch != 'All':
        ax.set_title(f'Total Orders by Years in {branch} Branch',size=18)
    else:
        ax.set_title('Total Orders by Years in All Branches',size=18)
    ax.set_xlabel('Years',size=14)
    ax.set_ylabel('Count of Orders',size=14)
    print(orders_years)
    return ax;

ipywidgets.interact(select, branch = slct_branches_widget);
```

Branches All

```
            Count of Orders
Sales-Year
2011                  12161
2012                  12759
2013                  13103
2014                  13453
2015                  13698
2016                   8029
```




```
<AxesSubplot:title={'center':'Total Orders by Years in All Branches'}, xlabel
='Years', ylabel='Count of Orders'>
```

In [30]:

```
slct_years_widget = ipywidgets.Dropdown(options = years_list,value='All',descr

def select(year):
    df = data[['Sales-Year','Staff-Branch','Sales-InvoiceID']].copy()
    if year != 'All':
        df = df[df['Sales-Year'] == year]

    orders_branches = df.groupby('Staff-Branch').\
                      agg({'Sales-InvoiceID':'nunique'}).\
                      rename(columns={'Sales-InvoiceID':'Count of Orders'}).\
                      sort_values('Count of Orders',ascending=False)

    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = orders_branches.index,
                     y = 'Count of Orders',
                     data = orders_branches,
                     order=orders_branches.index,
                     color='#708090')

    if year != 'All':
        ax.set_title(f'Total Orders by Branches in {year}',size=18)
    else:
        ax.set_title('Total Orders by Branches in All Years',size=18)

    ax.set_xlabel('Branch',size=14)
    ax.set_ylabel('Count of Orders',size=14)
    print(orders_branches)
    return ax;

ipywidgets.interact(select, year = slct_years_widget);
```

Years All

Staff-Branch	Count of Orders
London	7422
Leeds	6160
Reading	5791
Southampton	5600
Liverpool	5583
Glasgow	5572
Cardiff	5510
Banbury	5384
Manchester	5375
Aberdeen	5299
Lisburn	5255
Belfast	5178
Wrexham	5074


```
<AxesSubplot:title={'center':'Total Orders by Branches in All Years'}, xlabel
='Branch', ylabel='Count of Orders'>
```

Total Quantity Sold

In [31]:
```
total_quantity = data['Sales-Quantity'].sum()
print(f'Total Quantity Sold = {total_quantity:,} items')
```

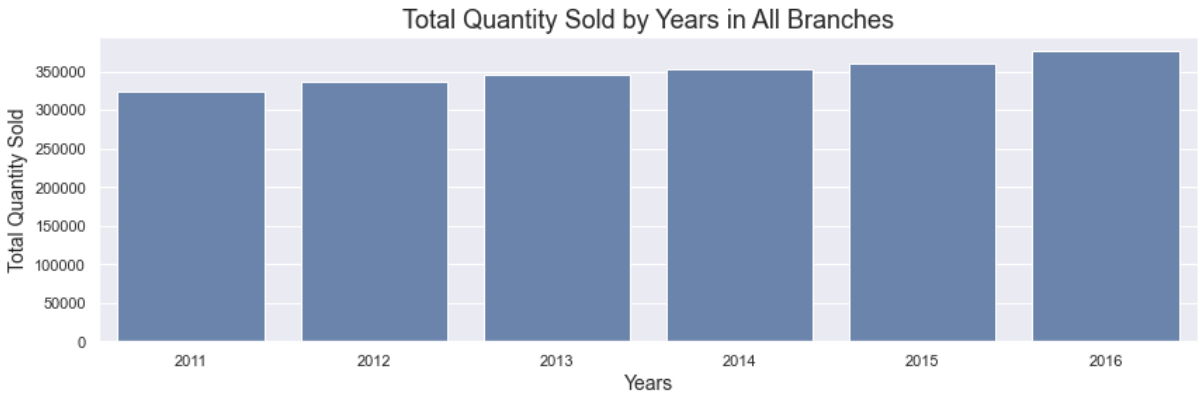
```
Total Quantity Sold = 2,093,762 items
```

In [32]:

```
slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All'


def select(branch):
    df = data[['Sales-Year','Staff-Branch','Sales-Quantity']].copy()
    if branch != 'All':
        df = df[df['Staff-Branch'] == branch]

    quantity_years = df.groupby('Sales-Year').\
                    agg({'Sales-Quantity':'sum'}).\
                    rename(columns={'Sales-Quantity':'Total Quantity Sold'})

    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = quantity_years.index,
                     y = 'Total Quantity Sold',
                     data = quantity_years,
                     order=quantity_years.index,
                     color='#6082B6')

    if branch != 'All':
        ax.set_title(f'Total Quantity Sold by Years in {branch} Branch',size=1
    else:
        ax.set_title('Total Quantity Sold by Years in All Branches',size=18)
    ax.set_xlabel('Years',size=14)
    ax.set_ylabel('Total Quantity Sold',size=14)
    print(quantity_years)
    return ax;

ipywidgets.interact(select, branch = slct_branches_widget);
```

Branches All

```
            Total Quantity Sold
Sales-Year
2011                     323311
2012                     335947
2013                     345281
2014                     352216
2015                     360961
2016                     376046
```




```
<AxesSubplot:title={'center':'Total Quantity Sold by Years in All Branche
```

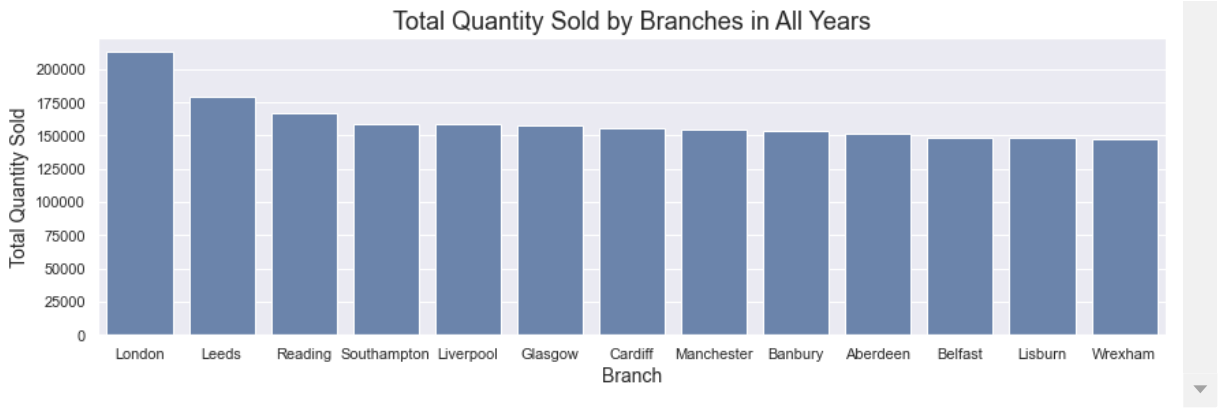
```
s'}, xlabel='Years', ylabel='Total Quantity Sold'>
```

In [33]:

```
slct_years_widget = ipywidgets.Dropdown(options = years_list,value='All',descr

def select(year):
    df = data[['Sales-Year','Staff-Branch','Sales-Quantity']].copy()
    if year != 'All':
        df = df[df['Sales-Year'] == year]

    quantity_branches = df.groupby('Staff-Branch').\
                        agg({'Sales-Quantity':'sum'}).\
                        rename(columns={'Sales-Quantity':'Total Quantity Sold'
                        sort_values('Total Quantity Sold', ascending=False)

    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = quantity_branches.index,
                     y = 'Total Quantity Sold',
                     data = quantity_branches,
                     order=quantity_branches.index,
                     color='#6082B6')

    if year != 'All':
        ax.set_title(f'Total Quantity Sold by Branches in {year}',size=18)
    else:
        ax.set_title('Total Quantity Sold by Branches in All Years',size=18)

    ax.set_xlabel('Branch',size=14)
    ax.set_ylabel('Total Quantity Sold',size=14)
    print(quantity_branches)
    return ax;

ipywidgets.interact(select, year = slct_years_widget);
```

Years All

Staff-Branch	Total Quantity Sold
London	213030
Leeds	179105
Reading	167027
Southampton	158908
Liverpool	158541
Glasgow	157763
Cardiff	155222
Manchester	154217
Banbury	153950
Aberdeen	151553
Belfast	148618
Lisburn	148456
Wrexham	147372




```
<AxesSubplot:title={'center':'Total Quantity Sold by Branches in All Years'},
xlabel='Branch', ylabel='Total Quantity Sold'>
```

Observation:

Although total roders were decreased, the total quantity sold is increassed. That means people are buying larger orders which shows trust in the brand

Total Revenue

In [34]:
```
# data['Revenue'] = data['Sales-Unit Price'] * data['Sales-Quantity'] * (1-data[
total_revenue = data['Revenue'].sum().round(0)
print(f'Total Revenue = £{total_revenue:,}')
```

```
Total Revenue = £133,816,456.0
```

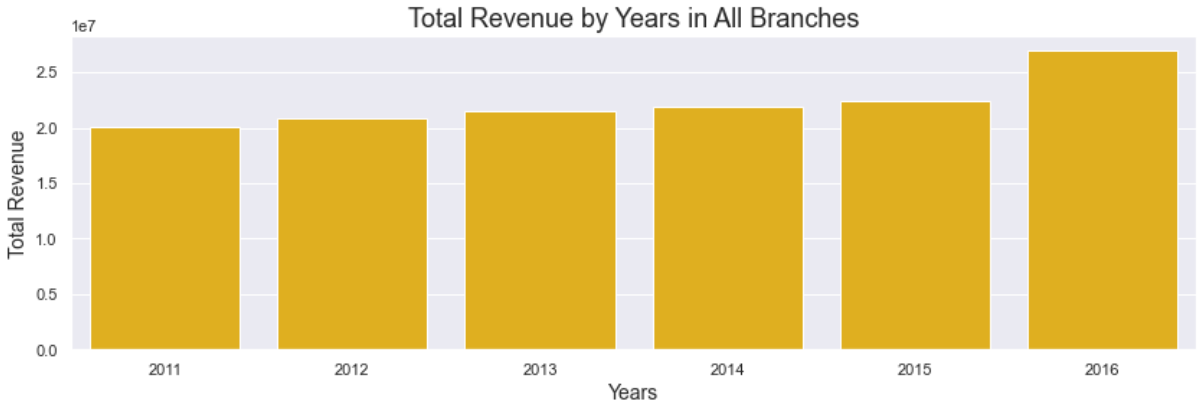
Total Revenue by Years

In [35]:

```
slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All'


def select(branch):
    df = data[['Sales-Year','Staff-Branch','Revenue']].copy()
    if branch != 'All':
        df = df[df['Staff-Branch'] == branch]

    revenue_years = df.groupby('Sales-Year').\
                    agg({'Revenue':'sum'}).\
                    rename(columns={'Revenue':'Total Revenue'})
    revenue_years['Total Revenue'] = revenue_years['Total Revenue'].round(0)

    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = revenue_years.index,
                     y = 'Total Revenue',
                     data = revenue_years,
                     order=revenue_years.index,
                     color='#FFBF00')

    if branch != 'All':
        ax.set_title(f'Total Revenue by Years in {branch} Branch',size=18)
    else:
        ax.set_title('Total Revenue by Years in All Branches',size=18)

    ax.set_xlabel('Years',size=14)
    ax.set_ylabel('Total Revenue',size=14)
    print(revenue_years)
    return ax;

ipywidgets.interact(select, branch = slct_branches_widget);
```

Branches All

Sales-Year	Total Revenue
2011	20,119,650.0000000000000000
2012	20,908,434.0000000000000000
2013	21,501,936.0000000000000000
2014	21,898,533.0000000000000000
2015	22,457,708.0000000000000000
2016	26,930,195.0000000000000000




```
<AxesSubplot:title={'center':'Total Revenue by Years in All Branches'}, xlabel
='Years', ylabel='Total Revenue'>
```

Total Revenue by Bracnhes

In [39]:

```
slct_years_widget = ipywidgets.Dropdown(options = years_list,value='All',descr

def select(year):
    df = data[['Sales-Year','Staff-Branch','Revenue']].copy()
    if year != 'All':
        df = df[df['Sales-Year'] == year]

    revenue_branches = df.groupby('Staff-Branch').\
                        agg({'Revenue':'sum'}).\
                        rename(columns={'Revenue':'Total Revenue'}).\
                        sort_values('Total Revenue', ascending=False)

    revenue_branches = revenue_branches.round(0)

    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = revenue_branches.index,
                     y = 'Total Revenue',
                     data = revenue_branches,
                     order = revenue_branches.index,
                     color='#FFBF00')

    if year != 'All':
        ax.set_title(f'Total Revenue by Branches in {year}',size=18)
    else:
        ax.set_title('Total Revenue by Branches in All Years',size=18)

    ax.set_xlabel('Branch',size=14)
    ax.set_ylabel('Total Revenue',size=14)
    print(revenue_branches)
    return ax;

ipywidgets.interact(select, year = slct_years_widget);
```

Years All

Staff-Branch	Total Revenue
London	13,571,067.0000000000000000
Leeds	11,451,903.0000000000000000
Reading	10,728,925.0000000000000000
Liverpool	10,149,677.0000000000000000
Southampton	10,143,012.0000000000000000
Glasgow	10,079,391.0000000000000000
Cardiff	9,916,720.0000000000000000
Banbury	9,860,388.0000000000000000
Manchester	9,859,955.0000000000000000
Aberdeen	9,683,499.0000000000000000
Lisburn	9,479,553.0000000000000000
Belfast	9,450,684.0000000000000000
Wrexham	9,441,682.0000000000000000


```
<AxesSubplot:title={'center':'Total Revenue by Branches in All Years'}, xlabel
='Branch', ylabel='Total Revenue'>
```

Total Revenue by Year and Product Range

Total COGS

In [37]:
```
# data['Cost'] = data['Sales-Quantity']*data['Purchases-Unit Cost']
total_gocs = data['COGS'].sum().round()
print(f'Total COGS for All Branches through 2011-2016 = £{total_gocs:,}')
```

```
Total COGS for All Branches through 2011-2016 = £88,799,229.0
```

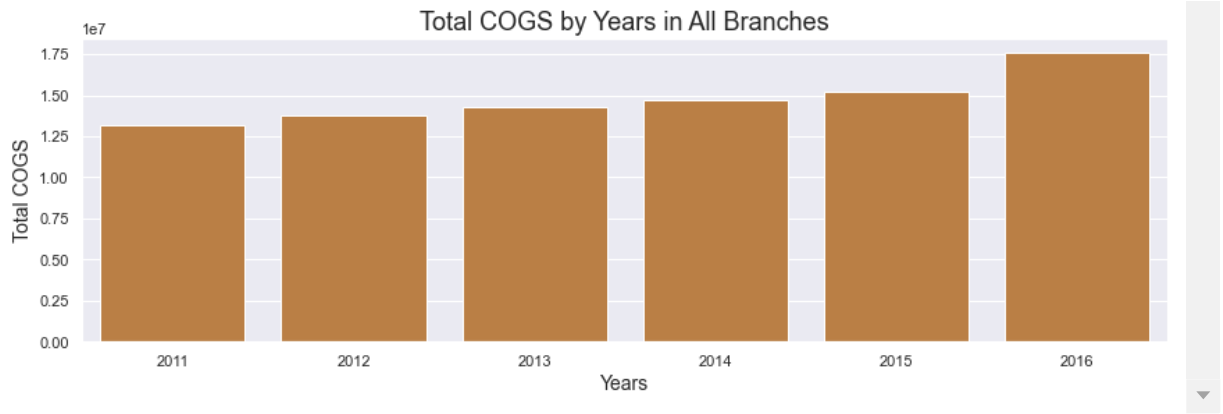
COGS by Years

In [41]:

```
slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All',de


def select(branch):
    df = data[['Sales-Year','Staff-Branch','COGS']].copy()
    if branch != 'All':
        df = df[df['Staff-Branch'] == branch]

    cost_years = df.groupby('Sales-Year').\
                    agg({'COGS':'sum'}).\
                    rename(columns={'COGS':'Total COGS'})
    cost_years =cost_years.round(0)

    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = cost_years.index,
                     y = 'Total COGS',
                     data = cost_years,
                     order= cost_years.index,
                     color='#CD7F32')

    if branch != 'All':
        ax.set_title(f'Total COGS by Years in {branch} Branch',size=18)
    else:
        ax.set_title('Total COGS by Years in All Branches',size=18)

    ax.set_xlabel('Years',size=14)
    ax.set_ylabel('Total COGS',size=14)
    print(cost_years)
    return ax;

ipywidgets.interact(select, branch = slct_branches_widget);
```

Branches All

Sales-Year	Total COGS
2011	13,167,235.00
2012	13,800,918.00
2013	14,324,576.00
2014	14,722,928.00
2015	15,208,279.00
2016	17,575,293.00




```
<AxesSubplot:title={'center':'Total COGS by Years in All Branches'}, xlabel='Y
ears', ylabel='Total COGS'>
```

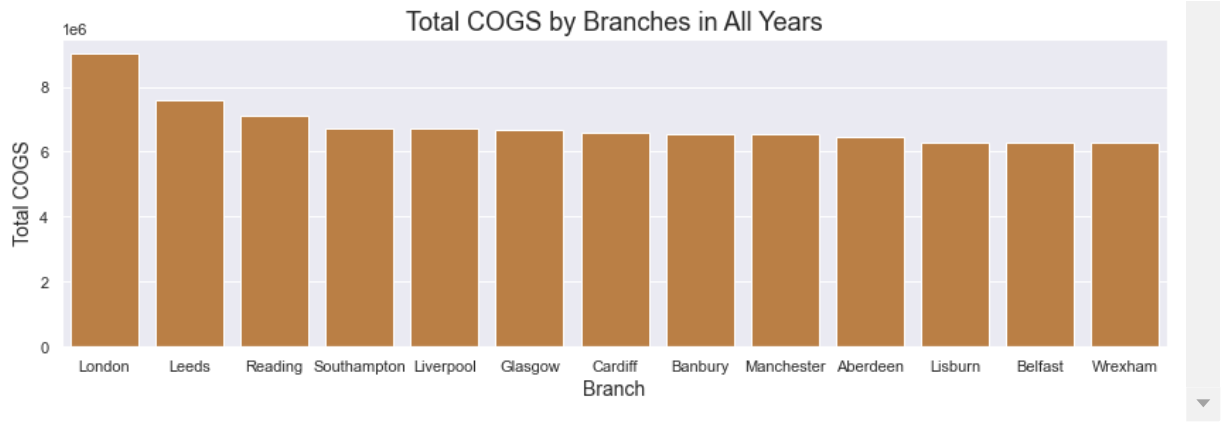
Total COGS by Branches

In [42]:

```
slct_years_widget = ipywidgets.Dropdown(options = years_list,value='All',descript


def select(year):
    df = data[['Sales-Year','Staff-Branch','COGS']].copy()
    if year != 'All':
        df = df[df['Sales-Year'] == year]

    cost_branches = df.groupby('Staff-Branch').\
                        agg({'COGS':'sum'}).\
                        rename(columns={'COGS':'Total COGS'}).\
                        sort_values('Total COGS', ascending=False)
    cost_branches = cost_branches.round(0)


    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = cost_branches.index,
                     y = 'Total COGS',
                     data = cost_branches,
                     order = cost_branches.index,
                     color='#CD7F32')

    if year != 'All':
        ax.set_title(f'Total COGS by Branches in {year}',size=18)
    else:
        ax.set_title('Total COGS by Branches in All Years',size=18)

    ax.set_xlabel('Branch',size=14)
    ax.set_ylabel('Total COGS',size=14)
    print(cost_branches)
    return ax;

ipywidgets.interact(select, year = slct_years_widget);
```

Years All

Staff-Branch	Total COGS
London	9,016,641.00
Leeds	7,589,467.00
Reading	7,113,384.00
Southampton	6,738,046.00
Liverpool	6,716,717.00
Glasgow	6,693,870.00
Cardiff	6,576,500.00
Banbury	6,542,034.00
Manchester	6,538,424.00
Aberdeen	6,442,288.00
Lisburn	6,294,805.00
Belfast	6,272,573.00
Wrexham	6,264,479.00




```
<AxesSubplot:title={'center':'Total COGS by Branches in All Years'}, xlabel='B
ranch', ylabel='Total COGS'>
```

Total Gross Profit

In [43]:
```
total_gross = data['Gross Profit'].sum().round(0)
print(f'Total Gross Profit = £{total_gross:,}')
```

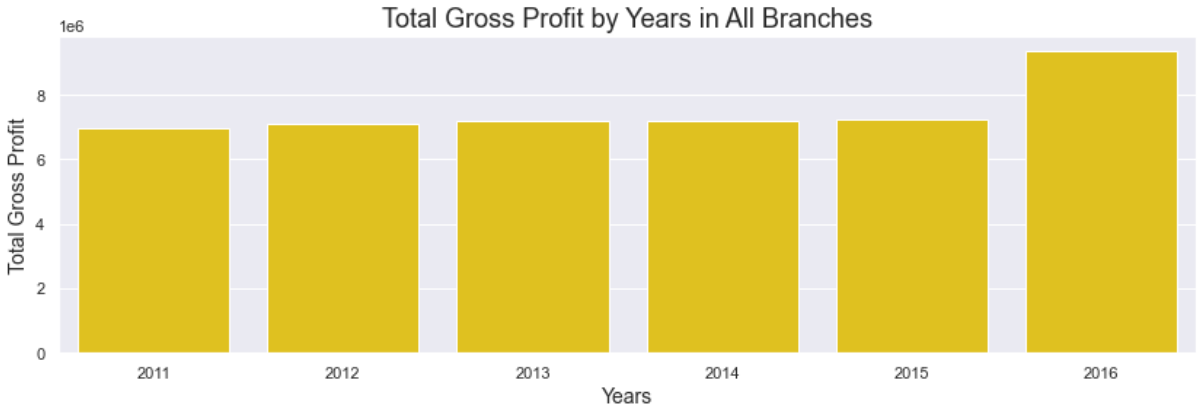
```
Total Gross Profit = £45,017,227.0
```

Total Gross Profit by Years

In [44]:
```
#slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All',d
def select(branch):
    df = data[['Sales-Year','Staff-Branch','Gross Profit']].copy()
    if branch != 'All':
        df = df[df['Staff-Branch'] == branch]

    gross_years = df.groupby('Sales-Year').\
                    agg({'Gross Profit':'sum'}).\
                    rename(columns={'Gross Profit':'Total Gross Profit'})

    gross_years = gross_years.round(0)

    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = gross_years.index,
                     y = 'Total Gross Profit',
                     data = gross_years,
                     order=gross_years.index,
                     color='#FFD700')

    if branch != 'All':
        ax.set_title(f'Total Gross Profit by Years in {branch} Branch',size=18)
    else:
        ax.set_title('Total Gross Profit by Years in All Branches',size=18)

    ax.set_xlabel('Years',size=14)
    ax.set_ylabel('Total Gross Profit',size=14)
    print(gross_years)
    return ax;

ipywidgets.interact(select, branch = slct_branches_widget);
```

Branches All

Sales-Year	Total Gross Profit
2011	6,952,414.00
2012	7,107,516.00
2013	7,177,360.00
2014	7,175,605.00
2015	7,249,429.00
2016	9,354,903.00




```
<AxesSubplot:title={'center':'Total Gross Profit by Years in All Branches'}, x
label='Years', ylabel='Total Gross Profit'>
```

Total Gross Profit by Branches

In [45]:
```
slct_years_widget = ipywidgets.Dropdown(options = years_list,value='All',descript
def select(year):
    df = data[['Sales-Year','Staff-Branch','Gross Profit']].copy()
    if year != 'All':
        df = df[df['Sales-Year'] == year]

    gross_branches = df.groupby('Staff-Branch').\
                        agg({'Gross Profit':'sum'}).\
                        rename(columns={'Gross Profit':'Total Gross Profit'}).\
                        sort_values('Total Gross Profit', ascending=False)

    gross_branches = gross_branches.round(0)

    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = gross_branches.index,
                     y = 'Total Gross Profit',
                     data = gross_branches,
                     order = gross_branches.index,
                     color='#FFD700')

    if year != 'All':
        ax.set_title(f'Total Gross Profit by Branches in {year}',size=18)
    else:
        ax.set_title('Total Gross Profit by Branches in All Years',size=18)

    ax.set_xlabel('Branch',size=14)
    ax.set_ylabel('Total Gross Profit',size=14)
    print(gross_branches)
    return ax;

ipywidgets.interact(select, year = slct_years_widget);
```

Years All

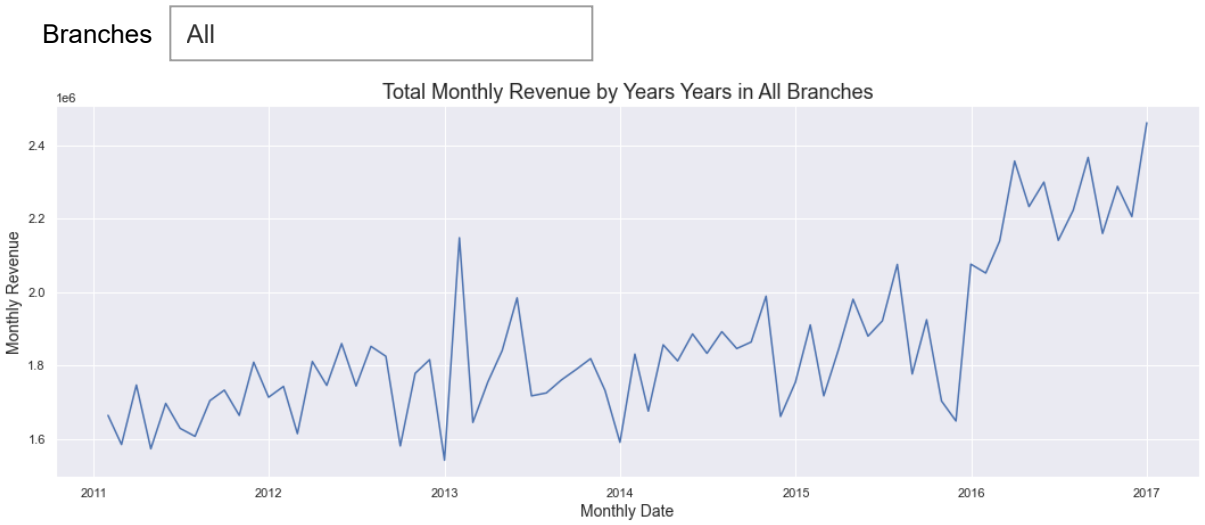
Staff-Branch	Total Gross Profit
London	4,554,427.00
Leeds	3,862,436.00
Reading	3,615,541.00
Liverpool	3,432,960.00
Southampton	3,404,966.00
Glasgow	3,385,521.00
Cardiff	3,340,220.00
Manchester	3,321,531.00
Banbury	3,318,353.00
Aberdeen	3,241,211.00
Lisburn	3,184,748.00
Belfast	3,178,111.00
Wrexham	3,177,203.00


```
<AxesSubplot:title={'center':'Total Gross Profit by Branches in All Years'}, x
label='Branch', ylabel='Total Gross Profit'>
```

Revenue Time-series Analysis

Monthly Revenue

In [46]:
```
slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All'
def select_monthly_revenue(branch):
    tsa = data[['Staff-Branch','Sales-Invoice Date','Sales-Period','Revenue']]

    if branch != 'All':
        tsa = tsa[tsa['Staff-Branch'] == branch]

    tsa = tsa.sort_values('Sales-Invoice Date').\
            set_index('Sales-Invoice Date').\
            resample('M')['Revenue'].sum().\
            to_frame().\
            rename(columns={'Revenue':'Monthly Revenue'})

    plt.figure(figsize=(18,6))
    rev_timeline = sns.lineplot(x= tsa.index,y= 'Monthly Revenue',data=tsa)
    rev_timeline.set_xlabel('Monthly Date',size=14)
    rev_timeline.set_ylabel('Monthly Revenue',size=14)


    if branch != 'All':
        rev_timeline.set_title(f'Total Monthly Revenue by Years in {branch}',s
    else:
        rev_timeline.set_title('Total Monthly Revenue by Years Years in All Bra

    return rev_timeline;

ipywidgets.interact(select_monthly_revenue, branch = slct_branches_widget);
```

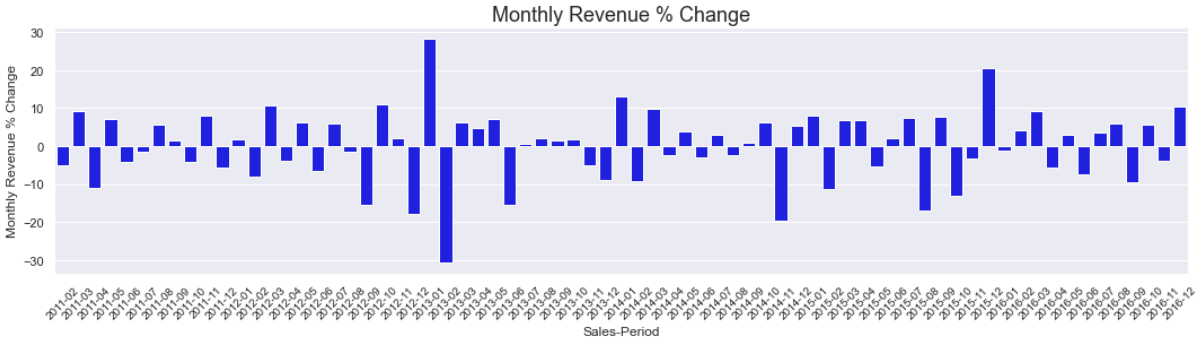



```
<AxesSubplot:title={'center':'Total Monthly Revenue by Years Years in All Bran
ches'}, xlabel='Monthly Date', ylabel='Monthly Revenue'>
```

Monthly Revenue % Change

In [47]:
```
slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All',de

def select_monthly_revenue(branch):
    tsa = data[['Staff-Branch','Sales-Invoice Date','Sales-Period','Revenue']].co

    if branch != 'All':
        tsa = tsa[tsa['Staff-Branch'] == branch]



    tsa = tsa.sort_values('Sales-Invoice Date').set_index('Sales-Invoice Date').\
            groupby('Sales-Period')['Revenue'].sum().\
            to_frame().rename(columns={'Revenue':'Monthly Revenue'})


    # tsa = tsa.sort_values('Sales-Invoice Date').set_index('Sales-Invoice Date')
    #         resample('M')['Revenue'].sum().\
    #         to_frame().rename(columns={'Revenue':'Monthly Revenue'})

    tsa['Monthly Revenue Change'] = tsa['Monthly Revenue'].diff(1)
    tsa['Monthly Revenue % Change'] = tsa['Monthly Revenue Change']/tsa['Monthly
    tsa = tsa[tsa['Monthly Revenue Change'].notna()]

    plt.figure(figsize=(18,4))
    ax = sns.barplot(x= tsa.index,y='Monthly Revenue % Change',data=tsa, color='b
    plt.xticks(rotation=45,size=10)

    if branch != 'All':
        ax.set_title(f'Monthly Revenue % Change in  {branch}',size=18)
    else:
        ax.set_title('Monthly Revenue % Change',size=18)


    return ax;

ipywidgets.interact(select_monthly_revenue, branch = slct_branches_widget);
```

Branches All

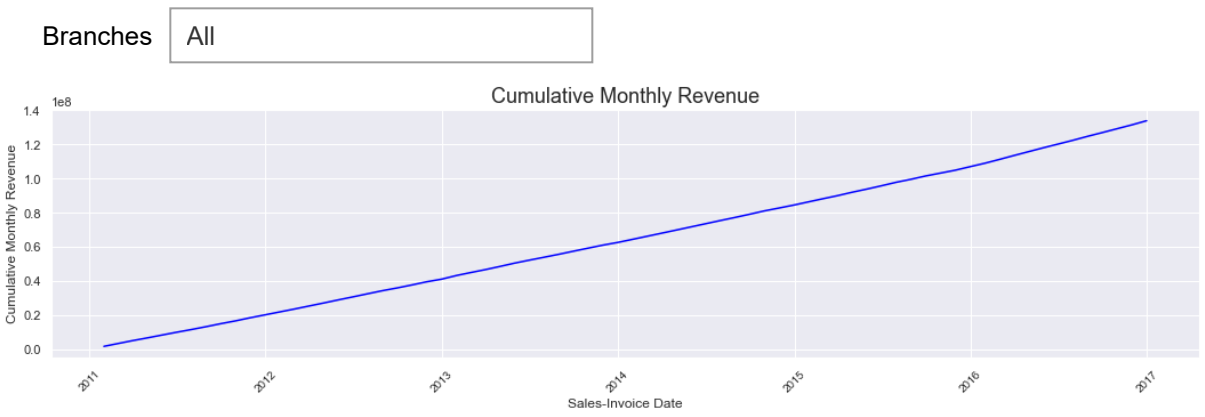



```
<AxesSubplot:title={'center':'Monthly Revenue % Change'}, xlabel='Sales-Peri
```

```
od', ylabel='Monthly Revenue % Change'>
```

Cumulative Monthly Revenue

In [48]:
```
slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All',de

def select_monthly_revenue(branch):
    tsa = data[['Staff-Branch','Sales-Invoice Date','Sales-Period','Revenue']].co

    if branch != 'All':
        tsa = tsa[tsa['Staff-Branch'] == branch]



    tsa = tsa.sort_values('Sales-Invoice Date').\
            set_index('Sales-Invoice Date').\
            resample('M')['Revenue'].sum().\
            to_frame().\
            rename(columns={'Revenue':'Monthly Revenue'})



    tsa['Cumulative Monthly Revenue'] = tsa['Monthly Revenue'].cumsum()

    plt.figure(figsize=(18,4))
    ax = sns.lineplot(x= tsa.index,y='Cumulative Monthly Revenue',data=tsa, color
    plt.xticks(rotation=45,size=10)

    if branch != 'All':
        ax.set_title(f'Cumulative Monthly Revenue in {branch}',size=18)
    else:
        ax.set_title('Cumulative Monthly Revenue',size=18)


    return ax;

ipywidgets.interact(select_monthly_revenue, branch = slct_branches_widget);
```




```
<AxesSubplot:title={'center':'Cumulative Monthly Revenue'}, xlabel='Sales-Invo
ice Date', ylabel='Cumulative Monthly Revenue'>
```

Staff Analysis

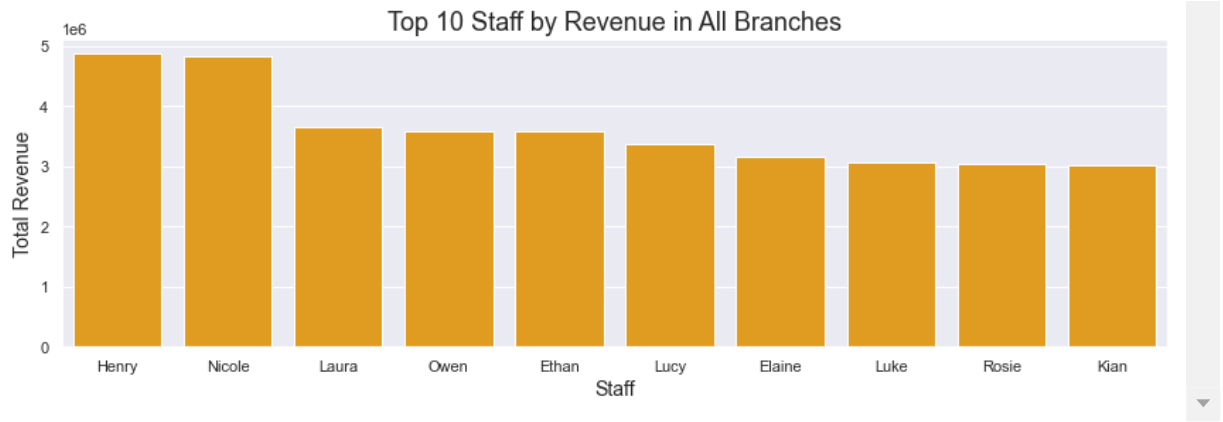
Staff Performance by Revenue

In [49]:
```
slct_years_widget = ipywidgets.Dropdown(options = years_list,value='All',descript
slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All',de

def select(branch,year):
    df = data[['Staff-Name','Staff-Branch','Sales-Year','Revenue']].copy()
    if branch != 'All':
        df = df[df['Staff-Branch'] == branch]
    if year != 'All':
        df = df[df['Sales-Year']==year]

    top_10_staff = df.groupby(['Staff-Name']).agg(TotalRevenue = ('Revenue','sum'
                        sort_values('TotalRevenue', ascending=False).nlargest(10,
                        reset_index()
    top_10_staff['TotalRevenue'] = top_10_staff['TotalRevenue'].round(0)


    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x='Staff-Name', y='TotalRevenue',
                     data=top_10_staff,
                     color = 'orange',
                     order = top_10_staff['Staff-Name'])

    if branch != 'All':
        ax.set_title(f'Top 10 Staff by Revenue in {branch} Branch',size=18)
    else:
        ax.set_title('Top 10 Staff by Revenue in All Branches',size=18)

    ax.set_xlabel('Staff',size=14)
    ax.set_ylabel('Total Revenue',size=14)
    print(top_10_staff)
    return ax;

ipywidgets.interact(select, branch = slct_branches_widget,year=slct_years_widget)
```

Branches All

Years All

	Staff-Name	TotalRevenue
0	Henry	4,876,364.00
1	Nicole	4,836,661.00
2	Laura	3,647,738.00
3	Owen	3,595,039.00
4	Ethan	3,582,930.00
5	Lucy	3,379,237.00
6	Elaine	3,162,133.00
7	Luke	3,075,458.00
8	Rosie	3,034,970.00
9	Kian	3,031,298.00




```
<AxesSubplot:title={'center':'Top 10 Staff by Revenue in All Branches'}, xlabe
l='Staff', ylabel='Total Revenue'>
```

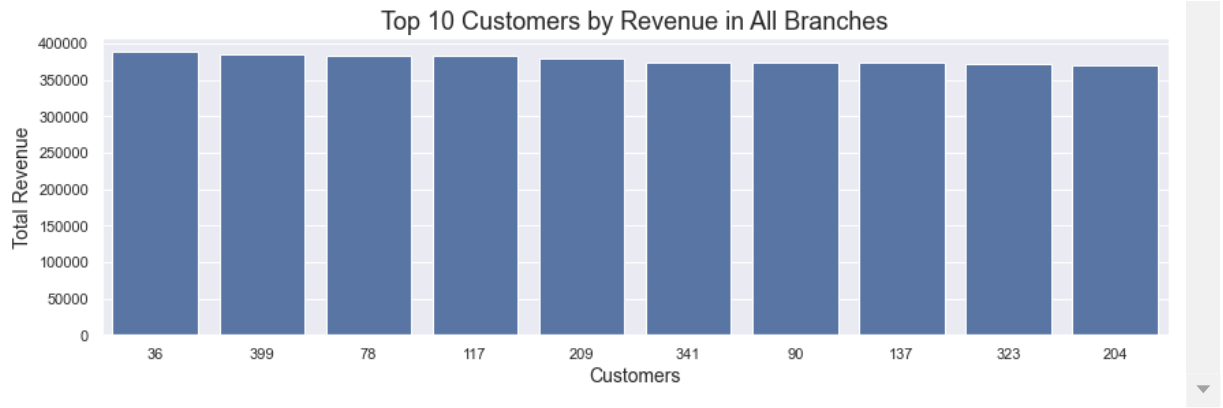
Customer Analysis

In [50]:
```
slct_years_widget = ipywidgets.Dropdown(options = years_list,value='All',descript
slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All',de

def select(branch,year):
    df = data[['Customers-ClientID','Staff-Branch','Sales-Year','Revenue']].copy(
    if branch != 'All':
        df = df[df['Staff-Branch'] == branch]
    if year != 'All':
        df = df[df['Sales-Year']==year]

    top_10_customer = df.groupby(['Customers-ClientID']).agg(TotalRevenue = ('Rev
                        sort_values('TotalRevenue', ascending=False).nlargest(10,
                        reset_index()

    top_10_customer['TotalRevenue'] = top_10_customer['TotalRevenue'].round(0)
    top_10_customer['Customers-ClientID'] = top_10_customer['Customers-ClientID']

    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x='Customers-ClientID', y='TotalRevenue',data=top_10_custome
                     color = 'b', order = top_10_customer['Customers-ClientID'])

    if branch != 'All':
        ax.set_title(f'Top 10 Customers by Revenue in {branch} Branch',size=18)
    else:
        ax.set_title('Top 10 Customers by Revenue in All Branches',size=18)

    ax.set_xlabel('Customers',size=14)
    ax.set_ylabel('Total Revenue',size=14)
    print(top_10_customer)
    return ax;

ipywidgets.interact(select, branch = slct_branches_widget,year=slct_years_widget)
```

Branches All

Years All

	Customers-ClientID	TotalRevenue
0	36	388,459.00
1	399	384,620.00
2	78	383,480.00
3	117	382,538.00
4	209	380,176.00
5	341	374,133.00
6	90	373,212.00
7	137	372,921.00
8	323	371,195.00
9	204	370,590.00

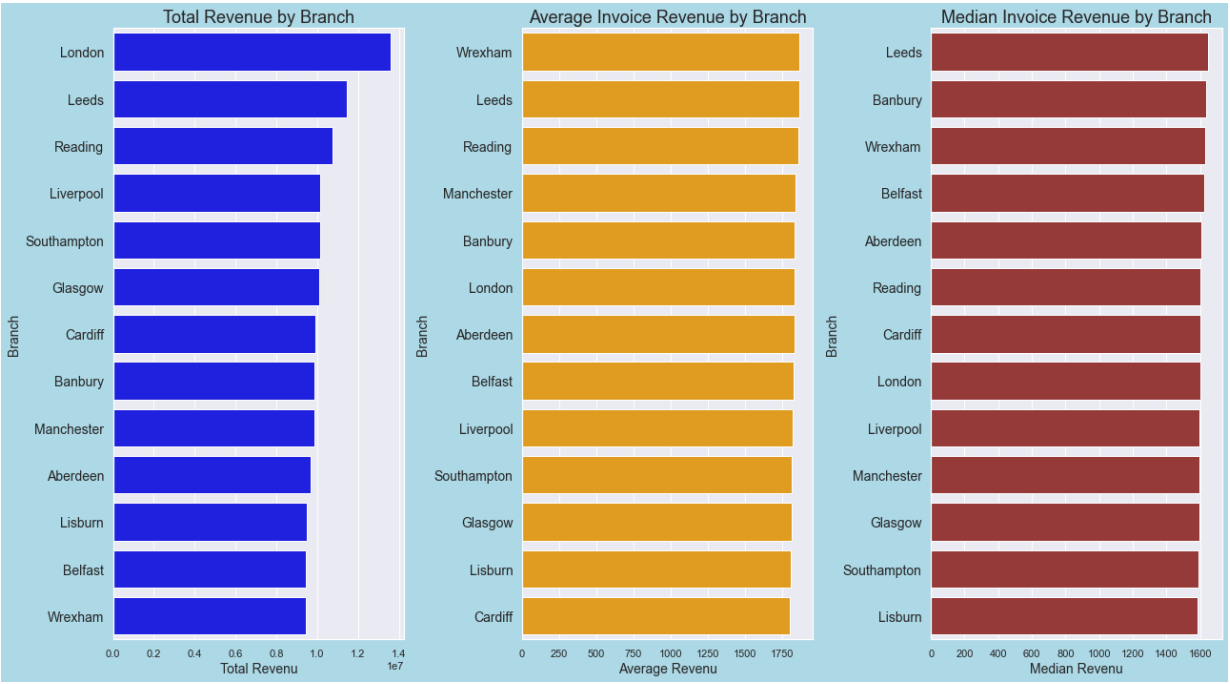



```
<AxesSubplot:title={'center':'Top 10 Customers by Revenue in All Branches'}, x
label='Customers', ylabel='Total Revenue'>
```

Show the Total, Average Revenue per Invoice and Median Revenue per Invoice by Branches

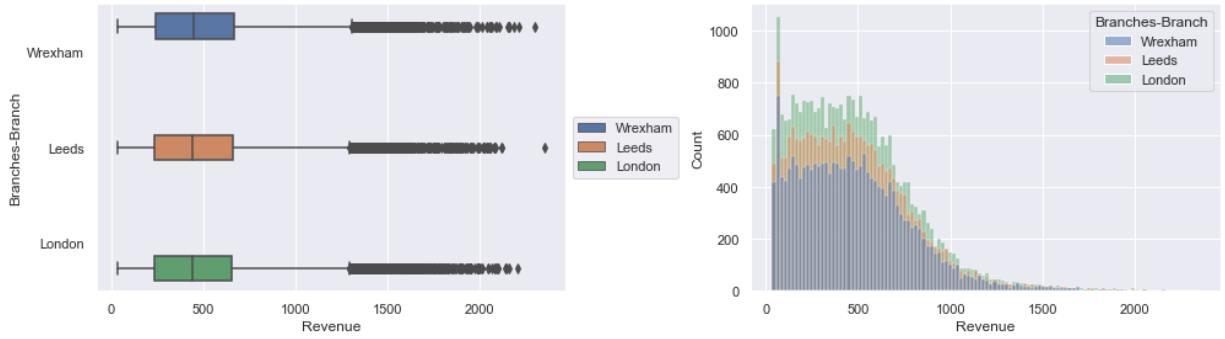
In [51]:
```
branch_invoice_df = data.groupby(['Branches-Branch','Sales-InvoiceID'])['Revenue'

branch_invoice_df = branch_invoice_df[branch_invoice_df['Revenue']!=0]

branch_df = branch_invoice_df.groupby('Branches-Branch').agg({'Revenue':['sum','m

branch_df = branch_df['Revenue'].rename(columns = {'sum':'Total Revenue','mean':'

branch_df = branch_df.sort_values('Total Revenue', ascending = False)

branch_totals_df, branch_averages_df, branch_medians_df = (branch_df['Total Rever
                                                           branch_df['Average Reve
                                                           branch_df['Median Rever

branch_totals_df,branch_averages_df , branch_medians_df = (branch_totals_df.sort_
                                                            branch_averages_df.sor
                                                            branch_medians_df.sort
```

In [52]:
```
fig = plt.figure(figsize=(18,10), facecolor='lightblue')

plt.subplot(1,3,1)
sns.barplot(y = branch_totals_df.index, x = 'Total Revenue', data = branch_totals
            order = branch_df.index, orient = 'h',color='blue')
plt.title('Total Revenue by Branch', size=18)
plt.xlabel('Total Revenu', size=14)
plt.ylabel('Branch', size=14)
plt.yticks(size=14)


plt.subplot(1,3,2)
sns.barplot(y = branch_averages_df.index, x = 'Average Revenue', data = branch_av
            order = branch_averages_df.index, orient = 'h',color='orange')
plt.title('Average Invoice Revenue by Branch', size=18)
plt.xlabel('Average Revenu', size=14)
plt.ylabel('Branch', size=14)
plt.yticks(size=14)


plt.subplot(1,3,3)
sns.barplot(y = branch_medians_df.index, x = 'Median Revenue', data = branch_medi
            order = branch_medians_df.index,orient = 'h', color = 'brown')
plt.title('Median Invoice Revenue by Branch', size=18)
plt.xlabel('Median Revenu', size=14)
plt.ylabel('Branch', size=14)
plt.yticks(size=14)

plt.tight_layout()
plt.show()
```



In [53]:
```
Lon_Lee_Wre_data = data[data['Branches-Branch'].isin(['London','Leeds','Wrexham']

Lon_Lee_Wre_data.loc[:,'Branches-Branch'] = Lon_Lee_Wre_data.loc[:,'Branches-Brar
```

In [54]:
```
plt.figure(figsize=(14,4))

plt.subplot(1,2,1)
ax1 = sns.boxplot(x='Revenue',y='Branches-Branch',data=Lon_Lee_Wre_data, hue='Bra
ax1.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.subplot(1,2,2)
ax2 = sns.histplot(data=Lon_Lee_Wre_data,x='Revenue', hue='Branches-Branch')


plt.tight_layout()
plt.show()
```



Observation:

Due to outliers, Leeds and Wrexham has larger average and median revenue respectively than London

Customer Cohort Analysis

In [55]:
```
# cohort_df = (data[data['Sales-Invoice Date'].dt.year == 2016]
#              [['Sales-Invoice Date','Customers-ClientID','Sales-Quantity']]
#             )

cohort_df = data[['Sales-Invoice Date','Customers-ClientID','Sales-Quantity']]


cohort_df.head()
```

Out[55]:

	Sales-Invoice Date	Customers-ClientID	Sales-Quantity
0	2011-01-03	133	3
1	2011-11-17	133	11
2	2011-01-03	133	13
3	2011-09-26	133	2
4	2011-01-03	133	12

In [56]:
```
# examine monthly total quantity
print(cohort_df.set_index('Sales-Invoice Date')['Sales-Quantity'].resample('M').s
```

```
Sales-Invoice Date
2011-01-31    26913
2011-02-28    25410
2011-03-31    27961
2011-04-30    25253
2011-05-31    27442
              ...
2016-08-31    32708
2016-09-30    29493
2016-10-31    31389
2016-11-30    30447
2016-12-31    33870
Freq: M, Name: Sales-Quantity, Length: 72, dtype: int64
```

In [57]:
```
# examine monthly total number of distinct customers
print(cohort_df.set_index('Sales-Invoice Date')['Customers-ClientID'].resample('M
```

```
Sales-Invoice Date
2011-01-31    379
2011-02-28    377
2011-03-31    387
2011-04-30    385
2011-05-31    393
             ...
2016-08-31    361
2016-09-30    338
2016-10-31    358
2016-11-30    346
2016-12-31    378
Freq: M, Name: Customers-ClientID, Length: 72, dtype: int64
```

From the numbers of both series, I can conclude that there is nothing not normal with data (there

is no outlier value in the totals). So, it is ready for the cohort analysis.

In [58]:
```
cohort = cohort_df.groupby('Customers-ClientID')['Sales-Invoice Date'].min().dt.t
cohort = cohort.to_frame()
cohort.columns = ['Cohort']
```

In [59]:
```
cohort['Cohort'].value_counts()
```

Out[59]:
```
2011-01    379
2016-01     36
2011-02     35
2011-03      8
2016-02      8
2016-03      2
Freq: M, Name: Cohort, dtype: int64
```

In [60]:
```
# join the cohort_df dataframe to cohort dataframe
cohort_df = cohort_df.set_index('Customers-ClientID').join(cohort)

cohort_df.reset_index(inplace=True)

# make a new column for 'SalesPeriod'
cohort_df['SalesPeriod'] = cohort_df['Sales-Invoice Date'].dt.to_period('m')
```

In [61]:
```
# check the each cohort's retention over months
cohort_df.groupby(['Cohort','SalesPeriod'])['Customers-ClientID'].nunique()
```

Out[61]:
```
Cohort   SalesPeriod
2011-01  2011-01        379
         2011-02        342
         2011-03        346
         2011-04        345
         2011-05        351
                        ...
2016-03  2016-08          2
         2016-09          2
         2016-10          1
         2016-11          2
         2016-12          1
Name: Customers-ClientID, Length: 245, dtype: int64
```

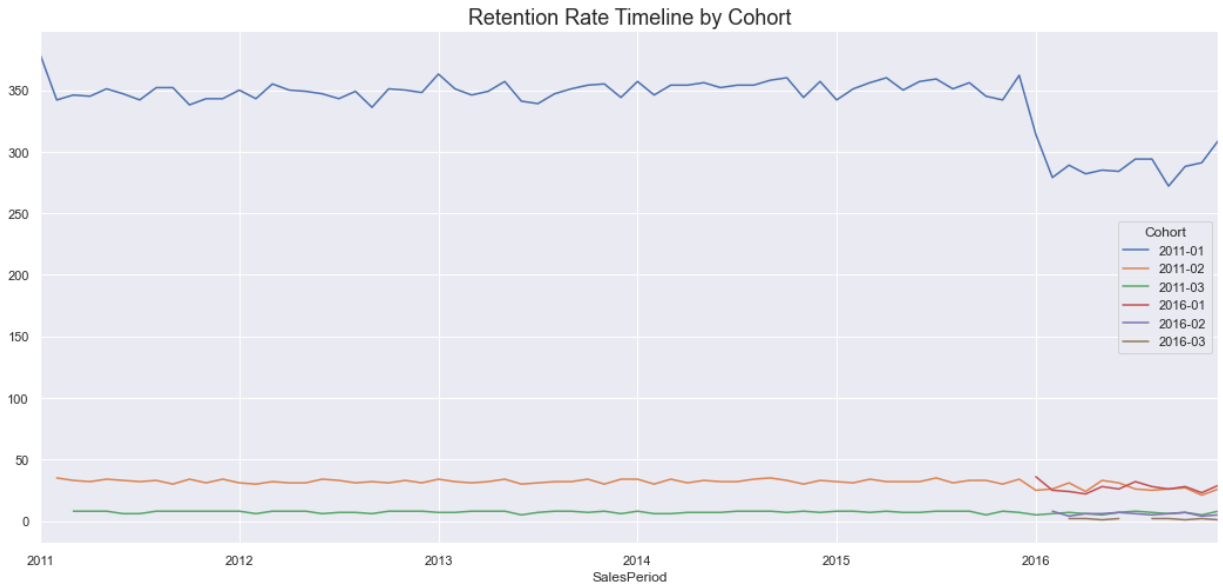
In [62]:
```
retention = cohort_df.groupby(['Cohort','SalesPeriod'])['Customers-ClientID'].nur
            reset_index()

retention = retention.pivot(index='SalesPeriod',columns='Cohort', values='Custome

retention
```

Out[62]:

SalesPeriod	2011-01	2011-02	2011-03	2016-01	2016-02	2016-03
2011-01	379.00	NaN	NaN	NaN	NaN	NaN
2011-02	342.00	35.00	NaN	NaN	NaN	NaN
2011-03	346.00	33.00	8.00	NaN	NaN	NaN
2011-04	345.00	32.00	8.00	NaN	NaN	NaN
2011-05	351.00	34.00	8.00	NaN	NaN	NaN
...	...	...	...	...	...	...
2016-08	294.00	25.00	7.00	28.00	5.00	2.00
2016-09	272.00	26.00	6.00	26.00	6.00	2.00
2016-10	288.00	27.00	7.00	28.00	7.00	1.00
2016-11	291.00	21.00	5.00	23.00	4.00	2.00
2016-12	309.00	26.00	8.00	29.00	5.00	1.00

72 rows × 6 columns

In [63]:
```
retention.plot(figsize=(18,8))
plt.title('Retention Rate Timeline by Cohort', size=18)
plt.show()
```



Observations:

- Customers registered only in 6 periods which are the first 3 months of 2011 and 2016.
- Most of the customers registered in 2011-01.
- 2011-01 Cohort are the most loyal customers with the highest retention rate, but their retention rate droped in the late months of 2015 and most of 2016 and stayed that way till last date available.
- Cohort 2016-03 has the smallest retention rate which stoped in mid 2016 before continuing few month later.
- Marketing team should investigate the reasons for those drop of retention rate of Cohort 2011-01 and the reasons why few customers registered after that Cohort.

I am going to investigate if Cohort 2011-01 customers are from the same region!

In [64]:
```
# get the IDs of Cohort 2011-01 customers
cohort_2011_01_customersIDs = cohort[cohort['Cohort']=='2011-01'].index
```

In [65]:
```
# filter the data dataframe to Cohort 2011-01 customers and get their address inf
cohort_2011_01_customersInfo = (data[data['Customers-ClientID'].
                                     isin(cohort_2011_01_customersIDs)]
                                [['Customers-Address','Staff-Branch']]
                               )
cohort_2011_01_customersInfo.head()
```

Out[65]:

	Customers-Address	Staff-Branch
0	52-55 Thompsons Lane, Medstead	Southampton
1	52-55 Thompsons Lane, Medstead	Southampton
2	52-55 Thompsons Lane, Medstead	Southampton
3	52-55 Thompsons Lane, Medstead	Southampton
4	52-55 Thompsons Lane, Medstead	Southampton

In [66]:
```
# examine the regions for Cohort 2011-01
cohort_2011_01_customersInfo['Staff-Branch'].value_counts()
```

Out[66]:
```
London         26092
Leeds          21510
Southampton    20349
Glasgow        19409
Liverpool      18522
Cardiff        18254
Aberdeen       18173
Banbury        18138
Lisburn        18023
Manchester     17810
Belfast        17784
Wrexham        17579
Reading        17219
Name: Staff-Branch, dtype: int64
```

There is no indication that Cohort 201-01 customers are from the same region!

One explanation for having cohort 2011-01 as the best cohort is that the customers in the cohort are living near the stores/selling centers of SEDISMART, hence they visit the stores regularly

Customer RFM Analysis

In [67]:
```
ref_date = data['Sales-Invoice Date'].max()
```

In [68]:
```
recency_df = data.groupby('Customers-ClientID')['Sales-Invoice Date'].\
                apply(lambda x: (ref_date - x.max()).days)
recency_df.head()
```

Out[68]:
```
Customers-ClientID
1     7
2    10
3     5
4    11
5     8
Name: Sales-Invoice Date, dtype: int64
```

In [69]:
```
frequency_df = data.groupby('Customers-ClientID')['Sales-InvoiceID'].nunique()
frequency_df.head()
```

Out[69]:
```
Customers-ClientID
1    178
2    178
3    164
4    175
5    166
Name: Sales-InvoiceID, dtype: int64
```

In [70]:
```
monetary_df =  data.groupby('Customers-ClientID')['Revenue'].sum().round(2)
monetary_df.head()
```

Out[70]:
```
Customers-ClientID
1   315,448.73
2   327,974.08
3   319,774.47
4   324,201.66
5   321,278.41
Name: Revenue, dtype: float64
```

In [71]:
```
rf_df = recency_df.to_frame(name='Recency').\
        join(frequency_df.to_frame(name='Frequency'))
rfm_df = rf_df.join(monetary_df.to_frame(name='Monetary'))
rfm_df.head()
```

Out[71]:

Customers-ClientID	Recency	Frequency	Monetary
1	7	178	315,448.73
2	10	178	327,974.08
3	5	164	319,774.47
4	11	175	324,201.66
5	8	166	321,278.41

In [72]:
```
rfm_stats = rfm_df.describe()
rfm_stats
```

Out[72]:

	Recency	Frequency	Monetary
count	468.00	468.00	468.00
mean	20.00	156.42	285,932.60
std	20.69	48.05	81,164.83
min	0.00	9.00	27,631.12
25%	6.00	159.00	282,634.65
50%	14.00	169.00	306,077.32
75%	26.00	179.00	328,723.52
max	154.00	212.00	388,459.10

In [73]:
```
quantiles = rfm_df.quantile(q=[0.2,0.4,0.6,0.8])
quantiles
```

Out[73]:

	Recency	Frequency	Monetary
0.20	5.00	157.00	277,588.54
0.40	11.00	166.00	297,688.79
0.60	18.00	173.00	313,943.10
0.80	30.00	183.00	333,843.23

In [74]:
```
# write a function for the recency score (the later the purchase the larger the s
def Rscore(x,q,df):
    if x <= df[q][0.2]:
        return 5
    elif x <= df[q][0.4]:
        return 4
    elif x <= df[q][0.6]:
        return 3
    elif x <= df[q][0.8]:
        return 2
    else:
        return 1
```

In [75]:
```
# write a function for the frequecny and monetary scores
# score (the larger the value the larger the score)
def FMscore(x,q,df):
    if x <= df[q][0.2]:
        return 1
    elif x <= df[q][0.4]:
        return 2
    elif x <= df[q][0.6]:
        return 3
    elif x <= df[q][0.8]:
        return 4
    else:
        return 1
```

In [76]:
```
rfm_copy = rfm_df.copy()

rfm_copy['Rscore'] = rfm_copy['Recency'].apply(Rscore, args = ('Recency',quantile

rfm_copy['Fscore'] = rfm_copy['Frequency'].apply(FMscore, args = ('Frequency',qua

rfm_copy['Mscore'] = rfm_copy['Monetary'].apply(FMscore, args = ('Monetary',quant

rfm_copy['RFMscore'] = rfm_copy[['Rscore','Fscore','Mscore']].mean(axis=1).round(


rfm_copy.head()
```

Out[76]:

Customers-ClientID	Recency	Frequency	Monetary	Rscore	Fscore	Mscore	RFMscore
1	7	178	315,448.73	4	4	4	4.00
2	10	178	327,974.08	4	4	4	4.00
3	5	164	319,774.47	5	2	4	3.67
4	11	175	324,201.66	4	4	4	4.00
5	8	166	321,278.41	4	2	4	3.33

In [77]:
```
def segment(df):
    if df.RFMscore <= 1:
        return 'Basic'
    elif df.RFMscore <= 2:
        return 'Bronze'
    elif df.RFMscore <= 3:
        return 'Sliver'
    elif df.RFMscore <= 4:
        return 'Gold'
    elif df.RFMscore <= 4.5:
        return 'Platinum'
    else:
        return 'Diamond'
```

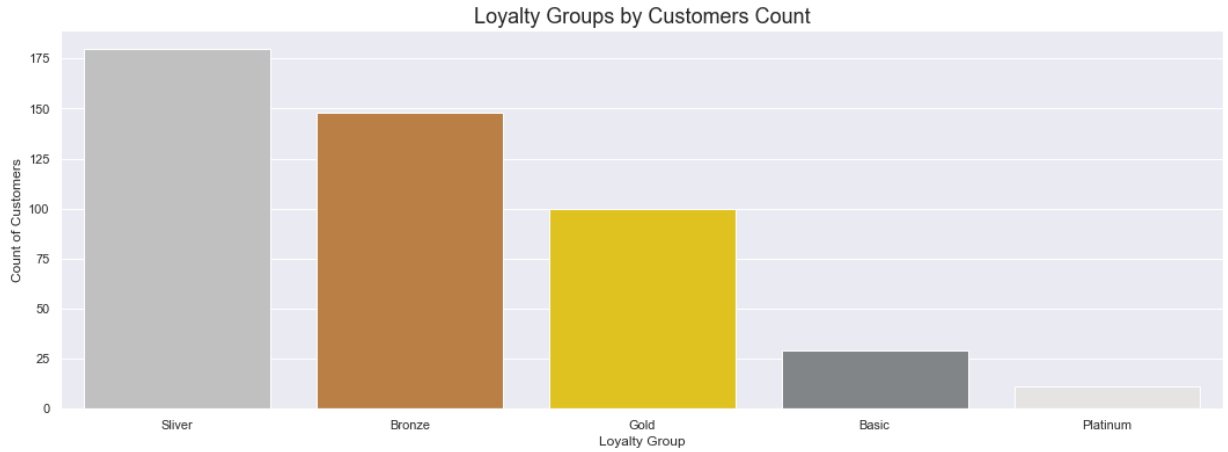
In [78]:
```
rfm_copy['LoyaltyGroup'] = rfm_copy.apply(segment, axis=1)
rfm_copy.head()
```

Out[78]:

Customers-ClientID	Recency	Frequency	Monetary	Rscore	Fscore	Mscore	RFMscore	LoyaltyGroup
1	7	178	315,448.73	4	4	4	4.00	Gold
2	10	178	327,974.08	4	4	4	4.00	Gold
3	5	164	319,774.47	5	2	4	3.67	Gold
4	11	175	324,201.66	4	4	4	4.00	Gold
5	8	166	321,278.41	4	2	4	3.33	Gold

In [79]:
```
print(rfm_copy.LoyaltyGroup.value_counts())

colors = ['#C0C0C0','#CD7F32','#FFD700','#818589','#E5E4E2']

plt.figure(figsize=(18,6))
sns.barplot(x = rfm_copy.LoyaltyGroup.value_counts().to_frame().index,
            y = rfm_copy.LoyaltyGroup.value_counts().to_frame()['LoyaltyGroup'],
           data = rfm_copy.LoyaltyGroup.value_counts().to_frame(), palette=colors
plt.xlabel('Loyalty Group')
plt.ylabel('Count of Customers')

plt.title('Loyalty Groups by Customers Count', size=18);
```

```
Sliver      180
Bronze      148
Gold        100
Basic        29
Platinum     11
Name: LoyaltyGroup, dtype: int64
```



Customers Clustering Based on Annaul Income and RFMscore

In this sub-section I will use Kmeans algorithm to cluster customers by their annual incomes and their RFM scores.

In [80]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

In [81]:
```
annual_income = data.groupby('Customers-ClientID')['Customers-Annual Income'].mea
annual_income.head()
```

Out[81]:

Customers-ClientID	Customers-Annual Income
1	55,400.00
2	183,600.00
3	385,900.00
4	412,000.00
5	372,600.00

In [82]:
```
RFMscore_income = rfm_copy['RFMscore'].to_frame().join(annual_income)
RFMscore_income.head()
```

Out[82]:

Customers-ClientID	RFMscore	Customers-Annual Income
1	4.00	55,400.00
2	4.00	183,600.00
3	3.67	385,900.00
4	4.00	412,000.00
5	3.33	372,600.00

In [83]:
```
scaler = StandardScaler()

RFMscore_income_log = log1p(RFMscore_income)

RFMscore_income_log_scaled = scaler.fit_transform(RFMscore_income_log)

RFMscore_income_log_scaled_df = pd.DataFrame(data = RFMscore_income_log_scaled,
                                             index = RFMscore_income.index,
                                             columns = RFMscore_income.columns
                                             )
RFMscore_income_log_scaled_df.head()
```

Out[83]:

Customers-ClientID	RFMscore	Customers-Annual Income
1	1.55	-2.28
2	1.55	-0.34
3	1.28	0.87
4	1.55	0.97
5	0.98	0.81

In [84]:
```
inertia1 = []

for n_cluster in range(1,10):
    kmeans1 = KMeans(n_clusters = n_cluster, random_state=42)
    kmeans1.fit(RFMscore_income_log_scaled_df)
    inertia1.append(kmeans1.inertia_)


df_inertia1 = pd.DataFrame({'Clusters':range(1,10),'Inertia':inertia1})

plt.figure(figsize=(8,4))
plt.plot(df_inertia1.Clusters, df_inertia1.Inertia, '-o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Clusters vs. Inertia')
```

Out[84]:
```
Text(0.5, 1.0, 'Clusters vs. Inertia')
```

In [85]:
```
n_cluster = 4

kmeans1 = KMeans(n_clusters = n_cluster, random_state=42)
clusters1 = kmeans1.fit_transform(RFMscore_income_log_scaled_df)

RFMscore_income_log_scaled_df['Label'] = kmeans1.labels_
```

In [86]:
```
RFMscore_income_log_scaled_df.head()
```

Out[86]:

Customers-ClientID	RFMscore	Customers-Annual Income	Label
1	1.55	-2.28	3
2	1.55	-0.34	0
3	1.28	0.87	0
4	1.55	0.97	0
5	0.98	0.81	0

In [87]:
```
fig = plt.figure(figsize=(14,6))
ax = sns.scatterplot(x = 'RFMscore',
            y = 'Customers-Annual Income',
            data = RFMscore_income_log_scaled_df,
            hue = 'Label',
            palette= ['#C0C0C0','#CD7F32','#FFD700','#818589']
           )

ax.set_xlabel('RFMscore')
ax.set_ylabel('Customers-Annual Incomey')

ax.set_title('Clusters of Customers', fontsize=18)

plt.show()
```

Customers can be segmented into the following segments:

- Customers with large purchasing power and large annaul income.
- Customers with large purcahsing power and low annaul income.
- Customers with low purchasing power and large income.

- Customers with low purchasing power and low income.

The marketing team should customize the promotions and discounts according to those clusters. For example, customers with large purchasing power and low income should be given big promotions for their loyalty to the brand, while customers with large income level and large purchasing power should be given promotions for expensive items.

Supplier Analysis

Suppliers by Total Revenue

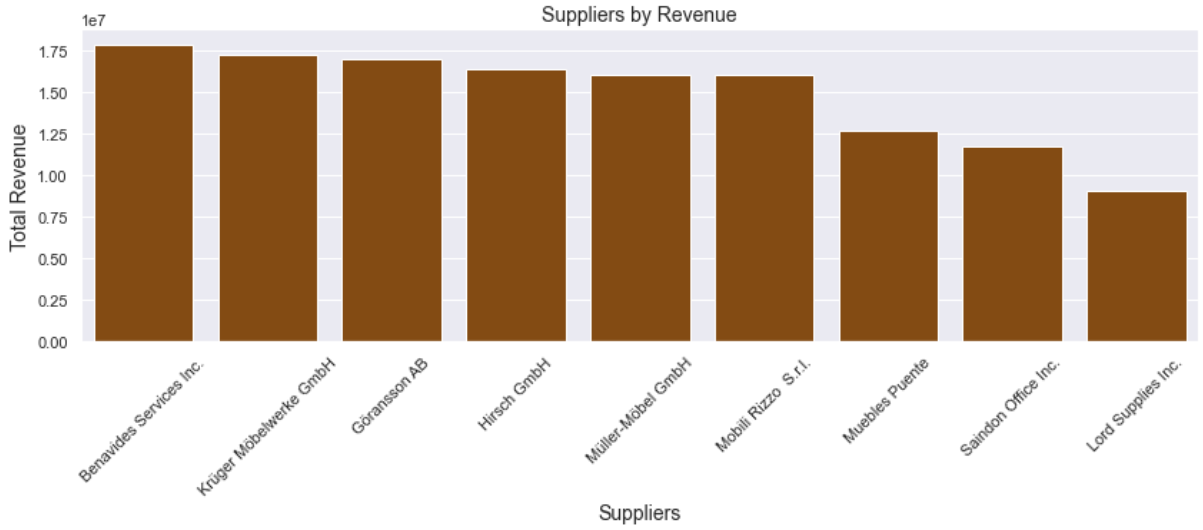
In [89]:
```
slct_products_widget = ipywidgets.Dropdown(options = products_list,value='All',de

slct_years_widget = ipywidgets.Dropdown(options = years_list,value='All',descript



def select(product, year):
    df = data[['Products-Supplier Name','Products-Product','Revenue','Sales-Year'
    if product!= 'All':
        df = df[df['Products-Product'] == product]
    if year != 'All':
        df = df[df['Sales-Year'] == year]


    suppliers_revenue = df.groupby('Products-Supplier Name').\
                           agg(TotalRevenue = ('Revenue','sum')).\
                           sort_values('TotalRevenue',ascending = False)

    suppliers_revenue['TotalRevenue'] = suppliers_revenue['TotalRevenue'].round(0

    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = suppliers_revenue.index,
                     y = 'TotalRevenue',
                     data = suppliers_revenue,
                     order= suppliers_revenue.index,
                     color='#964B00')
    plt.xticks(rotation=45)



    plt.title(f'Suppliers by Revenue',size=14)



    ax.set_xlabel('Suppliers',size=14)
    ax.set_ylabel('Total Revenue',size=14)
    print(suppliers_revenue)
    return ax;

ipywidgets.interact(select, product=slct_products_widget,
                    year=slct_years_widget
                   );
```

Product All

Years All

```
                          TotalRevenue
Products-Supplier Name
Benavides Services Inc.  17,854,352.00
Krüger Möbelwerke GmbH   17,221,771.00
Göransson AB             16,957,145.00
Hirsch GmbH              16,324,840.00
Müller-Möbel GmbH        16,033,960.00
Mobili Rizzo  S.r.l.     16,016,761.00
Muebles Puente           12,685,573.00
```

```
Saindon Office Inc.     11,712,461.00
Lord Supplies Inc.       9,009,593.00
```




```
<AxesSubplot:title={'center':'Suppliers by Revenue'}, xlabel='Suppliers', ylab
el='Total Revenue'>
```

Suppliers Segmentation

In [90]:
```
#preparing the suppliers dataframe
suppliers = data.\
        groupby(['Products-Supplier Name','Products-Supplier Lat','Products-Suppl
        agg(total_revenue = ('Revenue','sum'),complexity = ('Products-Complexity'
        reset_index()
suppliers = suppliers.loc[suppliers['complexity'].notna()]

suppliers['complexity'] = suppliers['complexity'].round(3)

suppliers = suppliers.rename(
                    columns ={'Products-Supplier Name':'Name','Products-Supplier
```

In [91]:
```
rev_mean_value = suppliers['total_revenue'].mean()
```

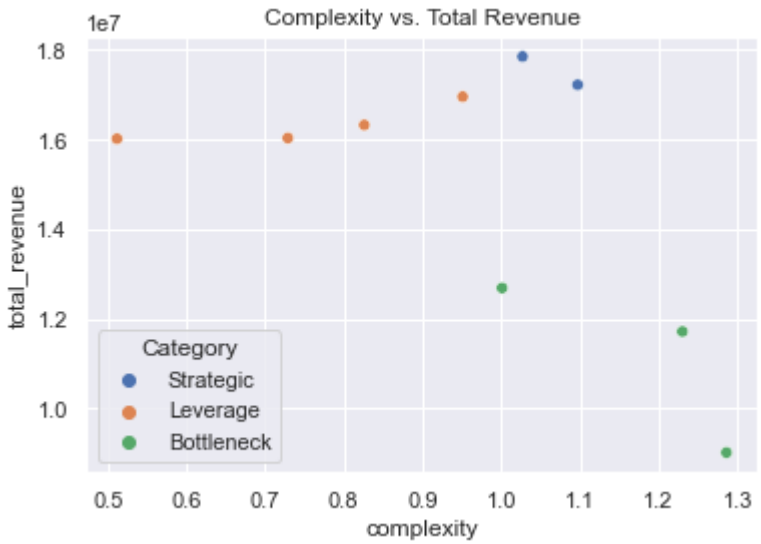
In [92]:
```
# write a function to segment suppliers
def supplier_seg(row):
    if (row['total_revenue'] >= rev_mean_value) & (row['complexity'] >=1):
        return 'Strategic'
    elif (row['total_revenue'] >= rev_mean_value) & (row['complexity'] < 1):
        return 'Leverage'
    elif (row['total_revenue'] < rev_mean_value) & (row['complexity'] >= 1):
        return 'Bottleneck'
    elif (row['total_revenue'] < rev_mean_value) & (row['complexity'] < 1):
        return 'Non-critical'
suppliers['Category'] = suppliers.apply(supplier_seg,axis=1)
```

In [93]:
```
suppliers
```

Out[93]:

	Name	Lat	Long	total_revenue	complexity	Category
0	Benavides Services Inc.	33.45	-112.11	17,854,352.14	1.03	Strategic
161	Göransson AB	59.40	18.03	16,957,145.36	0.95	Leverage
221	Hirsch GmbH	48.20	11.50	16,324,839.98	0.83	Leverage
310	Krüger Möbelwerke GmbH	48.72	9.22	17,221,770.71	1.10	Strategic
334	Lord Supplies Inc.	35.36	-97.59	9,009,593.26	1.29	Bottleneck
439	Mobili Rizzo S.r.l.	40.76	14.15	16,016,760.70	0.51	Leverage
507	Muebles Puente	40.38	-3.68	12,685,573.12	1.00	Bottleneck
618	Müller-Möbel GmbH	48.14	11.55	16,033,959.60	0.73	Leverage
686	Saindon Office Inc.	43.75	-79.45	11,712,461.28	1.23	Bottleneck

In [94]:
```
sns.scatterplot(x='complexity',y='total_revenue',hue='Category',data=suppliers)
plt.title('Complexity vs. Total Revenue')
plt.show()
```



Suppliers by Location and Total Revenue

In [95]:
```
#preparing the suppliers dataframe
suppliers = data.\
        groupby(['Products-Supplier Name','Products-Supplier Lat','Products-Suppl
        agg(total_revenue = ('Revenue','sum'),complexity = ('Products-Complexity'
        reset_index()
suppliers = suppliers.loc[suppliers['complexity'].notna()]

suppliers['complexity'] = suppliers['complexity'].round(3)

suppliers = suppliers.rename(
                    columns ={'Products-Supplier Name':'Name','Products-Supplier




map_suppliers = folium.Map(location = [40.4508829999999975,-85.1051750000000027],

for (index, row) in suppliers.iterrows():
    folium.Marker(location=[row.loc['Lat'],row.loc['Long']],
                  popup = 'Revenue: ' + str(row.loc['total_revenue'])  ,
                  tooltip = row.loc['Name']
                 ).add_to(map_suppliers)

map_suppliers
```

Out[95]:

Product Analysis

Top 3 Products by Total Quantity Sold

In [96]:
```
slct_product_ranges_widget = ipywidgets.Dropdown(options = product_ranges_list,
                                                 value='All',
                                                 description='Product Range',
                                                 disabled=False)

slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All',de

slct_years_widget = ipywidgets.Dropdown(options = years_list,value='All',descript



def select(product_range, branch, year):
    df = data[['Products-Product Range','Sales-Year','Staff-Branch','Products-Pro
    if product_range != 'All':
        df = df[df['Products-Product Range'] == product_range]
    if branch != 'All':
        df = df[df['Staff-Branch'] == branch]
    if year != 'All':
        df = df[df['Sales-Year'] == year]


    top_products_by_qty = df.groupby('Products-Product').\
                             agg({'Sales-Quantity':'sum'}).\
                             rename(columns={'Sales-Quantity':'Total Quantity Sol
                             sort_values('Total Quantity Sold',ascending=False).h

    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = top_products_by_qty.index,
                     y = 'Total Quantity Sold',
                     data = top_products_by_qty,
                     order= top_products_by_qty.index,
                     color='#FFC000')

    if product_range == 'All':
        product_range_title = 'All Product Ranges'
    else:
        product_range_title = product_range+' Product Range'
    if branch == 'All':
        branch_title = 'All Branches'
    else:
        branch_title = branch
    if year == 'All':
        year_title = 'All Years'
    else:
        year_title = year

    plt.title(f'Top 3 Products in {branch_title} in {year_title} in {product_rang



    ax.set_xlabel('Products',size=14)
    ax.set_ylabel('Quantity Sold',size=14)
    print(top_products_by_qty)
    return ax;

ipywidgets.interact(select, product_range=slct_product_ranges_widget,
```

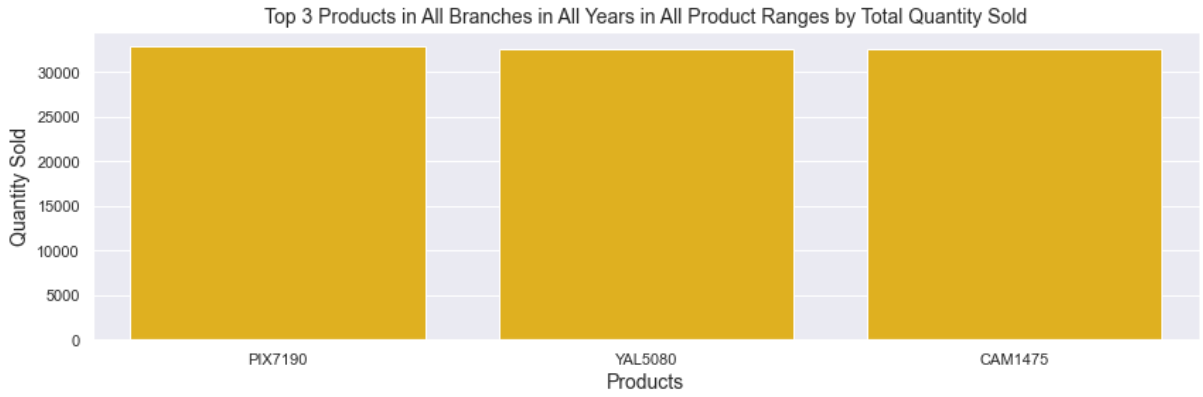
```
                branch=slct_branches_widget,
                year=slct_years_widget
               );
```

Product Ra… All

Branches All

Years All

Products-Product	Total Quantity Sold
PIX7190	32873
YAL5080	32529
CAM1475	32511




```
<AxesSubplot:title={'center':'Top 3 Products in All Branches in All Years in A
ll Product Ranges by Total Quantity Sold'}, xlabel='Products', ylabel='Quantit
y Sold'>
```

Top 3 Product by Revenue

In [98]:
```
slct_product_ranges_widget = ipywidgets.Dropdown(options = product_ranges_list,
                                                 value='All',
                                                 description='Product Range',
                                                 disabled=False)

slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All',de

slct_years_widget = ipywidgets.Dropdown(options = years_list,value='All',descript



def select(product_range, branch, year):
    df = data[['Products-Product Range','Sales-Year','Staff-Branch','Products-Pro
    if product_range != 'All':
        df = df[df['Products-Product Range'] == product_range]
    if branch != 'All':
        df = df[df['Staff-Branch'] == branch]
    if year != 'All':
        df = df[df['Sales-Year'] == year]


    top_products_by_rev = df.groupby('Products-Product').\
                             agg({'Revenue':'sum'}).\
                             rename(columns={'Revenue':'Total Revenue'}).\
                             sort_values('Total Revenue',ascending=False).head(3)

    top_products_by_rev = top_products_by_rev.round(0)

    fig, ax = plt.subplots(figsize=(14,4))
    ax = sns.barplot(x = top_products_by_rev.index,
                     y = 'Total Revenue',
                     data = top_products_by_rev,
                     order= top_products_by_rev.index,
                     color='#FFC000')

    if product_range == 'All':
        product_range_title = 'All Product Ranges'
    else:
        product_range_title = product_range+' Product Range'
    if branch == 'All':
        branch_title = 'All Branches'
    else:
        branch_title = branch
    if year == 'All':
        year_title = 'All Years'
    else:
        year_title = year

    plt.title(f'Top 3 Products in {branch_title} in {year_title} in {product_rang



    ax.set_xlabel('Products',size=14)
    ax.set_ylabel('Quantity Sold',size=14)
    print(top_products_by_rev)
    return ax;
```

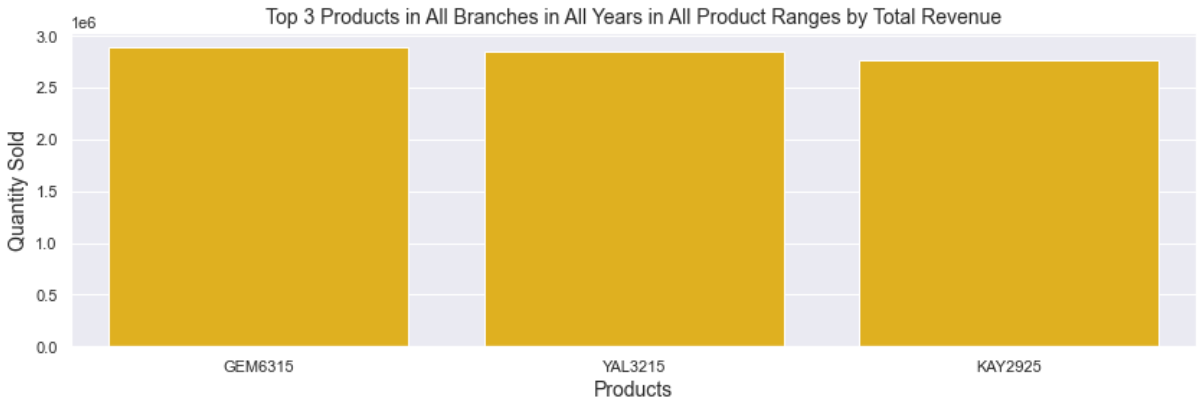
```
ipywidgets.interact(select, product_range=slct_product_ranges_widget,
                    branch=slct_branches_widget,
                    year=slct_years_widget
                   );
```

Product Ra… All

Branches All

Years All

Products-Product	Total Revenue
GEM6315	2,888,053.00
YAL3215	2,847,076.00
KAY2925	2,772,239.00




```
<AxesSubplot:title={'center':'Top 3 Products in All Branches in All Years in A
ll Product Ranges by Total Revenue'}, xlabel='Products', ylabel='Quantity Sol
d'>
```

Multi-criteria ABC Analysis

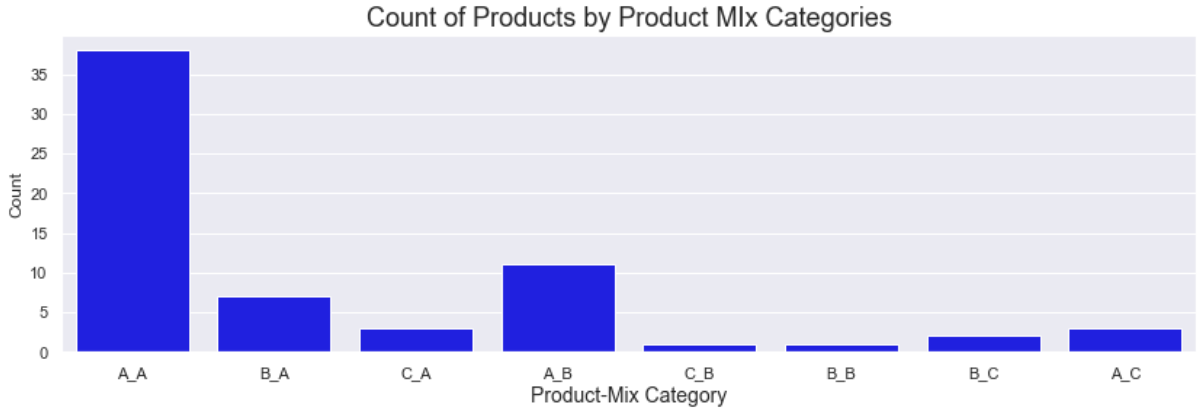
In [99]:
```
slct_product_ranges_widget = ipywidgets.Dropdown(options = product_ranges_list,
                                                 value='All',
                                                 description='Product Range',
                                                 disabled=False)

slct_branches_widget = ipywidgets.Dropdown(options = branches_list,value='All',de

slct_years_widget = ipywidgets.Dropdown(options = years_list,value='All',descript


def select_product_mix(product_range, branch, year):
    df = data[['Products-Product Range','Sales-Year','Staff-Branch','Products-Pro
    if product_range != 'All':
        df = df[df['Products-Product Range'] == product_range]
    if branch != 'All':
        df = df[df['Staff-Branch'] == branch]
    if year != 'All':
        df = df[df['Sales-Year'] == year]

    products_rev_qty = df.groupby('Products-Product').\
                          agg(total_sales=('Sales-Quantity','sum'),total_revenue=('
                          reset_index().\
                          rename(columns={'Products-Product':'Product'})


    abc_analysis = inv.productmix(products_rev_qty['Product'],products_rev_qty['t

    columns_names = {'skus':'Product','sales':'Total Quantity Sold','revenue':'To
                     'revenue_category':'Revenue Category','product_mix':'Product
                    }

    abc_analysis_df = abc_analysis.rename(columns = columns_names)

    req_columns = ['Product','Total Quantity Sold','Total Revenue','Quantity Cate

    abc_analysis_df = abc_analysis_df[req_columns]
    # abc_analysis_df.head()
    print(abc_analysis_df['Product Mix Category'].value_counts())

    plt.figure(figsize=(14,4))
    ax = sns.countplot(x='Product Mix Category',data=abc_analysis_df, color='blue
    ax.set_xlabel('Product-Mix Category',size=14)
    ax.set_ylabel('Count')
    ax.set_title('Count of Products by Product MIx Categories',size=18)
    plt.show()
    return ax

ipywidgets.interact(select_product_mix, product_range=slct_product_ranges_widget,
                    branch=slct_branches_widget,
                    year=slct_years_widget
                   );
```

Product Ra… All

Branches All

Years | All

```
A_A    38
A_B    11
B_A     7
C_A     3
A_C     3
B_C     2
C_B     1
B_B     1
Name: Product Mix Category, dtype: int64
```




```
<AxesSubplot:title={'center':'Count of Products by Product MIx Categories'}, x
label='Product-Mix Category', ylabel='Count'>
```

Product Recommendation List via Collaborative Filtering

Customer-Based Approach

In [100]:
```
from sklearn.metrics.pairwise import cosine_similarity
```

In [101]:
```
customer_item = data.pivot_table(index='Customers-ClientID', columns = 'Products-
                                 values='Sales-Quantity',aggfunc='sum')

customer_item = customer_item.applymap(lambda x: 1 if x > 0  else 0)
customer_item.index.name = 'customer_ID'
customer_item.columns.name = 'product_ID'
customer_item.head()
```

Out[101]:

product_ID	1	2	3	4	5	6	7	8	9	10	...	57	58	59	60	61	62	63	64	65	66
customer_ID																					
1	1	1	1	1	1	1	1	1	1	1	...	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	...	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1	...	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1	...	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1	...	1	1	1	1	1	1	1	1	1	1

5 rows × 66 columns

In [102]:
```
customer_customer_similarity = pd.DataFrame(cosine_similarity(customer_item))
customer_customer_similarity.columns = customer_item.index
customer_customer_similarity['customer_ID'] = customer_item.index
customer_customer_similarity = customer_customer_similarity.set_index('customer_I

pd.options.display.float_format = '{:,.16f}'.format

for i in range(len(customer_customer_similarity)):
    customer_customer_similarity.loc[i,i] = 1

customer_customer_similarity.head()
```

Out[102]:

customer_ID	1	2	3	4	
customer_ID					
1	1.0000000000000000	0.9999999999999992	0.9999999999999992	0.9999999999999992	0.9
2	0.9999999999999992	1.0000000000000000	0.9999999999999992	0.9999999999999992	0.9
3	0.9999999999999992	0.9999999999999992	1.0000000000000000	0.9999999999999992	0.9
4	0.9999999999999992	0.9999999999999992	0.9999999999999992	1.0000000000000000	0.9
5	0.9999999999999992	0.9999999999999992	0.9999999999999992	0.9999999999999992	1.0

5 rows × 469 columns

There are several customers with a similarity score of 0.9999999999999992. Those are customers who bought the same items. This of course is a synthesized data.

In [104]:
```
customer_3 = customer_customer_similarity.loc[3][customer_customer_similarity.loc
customer_3.sort_values(ascending=False).head(10)
```

Out[104]:
```
customer_ID
314   0.9923953268977457
423   0.8961195807649243
435   0.8961195807649243
428   0.8876253645985944
441   0.8876253645985944
459   0.8876253645985944
466   0.8876253645985944
460   0.8790490729915316
449   0.8790490729915316
446   0.8790490729915316
Name: 3, dtype: float64
```

Excluding the customers who bought the same items as customer number 3, those are the top 10 customers that are more similar to customer number 3.

I pick customer 459 and discuss how we can recommend products using these results.

In [105]:
```
customer_customer_similarity.loc[3,459]
```

Out[105]:
```
0.8876253645985944
```

The similarity between customer 3 and customer and customer 459 is 0.887625

The strategy is as follows. First, we need to identify the items that the customers 3 and 459 have already bought. Then, we are going to find the products that the target customer 459 has not purchased, but customer 3 has. Since these two customers have bought similar items in the past, we are going to assume that the target customer 459 has a high chance of purchasing the items that he or she has not bought, but customer 3 has bought. Lastly, we are going to use this list of items and recommend them to the target customer 459.

In [106]:
```
customer_3_items = set(customer_item.loc[3].iloc[customer_item.loc[3].to_numpy().
customer_459_items = set(customer_item.loc[459].iloc[customer_item.loc[459].to_nu
```

In [107]:
```
items_to_recommend_to_459 = customer_3_items - customer_459_items


items_to_recommend_to_459 = products.\
                 loc[products['Products-productid'].isin(items_to_recommend_to_459
print('Items to be recommended to customer with the ID 459')
items_to_recommend_to_459
```

```
Items to be recommended to customer with the ID 459
```

Out[107]:

	Products-productid	Products-Product
0	58	DIG1180
1	66	BRN1395
2	30	DIG6320
7	10	PIX2715
10	34	CAM7465
11	26	DIV2710
13	16	MYT2345
16	29	PIX4910
18	61	PIX1905
20	23	BRN3550
35	39	BRN3810
42	63	YAL4135
45	33	DIV4580
56	7	MYT2590

So this process can be repeated for ALL the top customers that are most similar to customer 3. Next the whole process shoudl be repeated for ALL customers of interest by replacing the customer 3 with their customer IDs

Item-Based Approach

In [108]:
```
item_item_similarity = pd.DataFrame(cosine_similarity(customer_item.T))
item_item_similarity.columns = customer_item.T.index
item_item_similarity['product_ID'] = customer_item.T.index
item_item_similarity = item_item_similarity.set_index('product_ID')

for i in range(len(item_item_similarity)):
    item_item_similarity.loc[i,i] = 1
item_item_similarity.head()
```

Out[108]:

product_ID	1	2	3	4	
product_ID					
1	1.0000000000000000	0.9846177626348285	0.9801857225104139	0.9857305609394474	0.98
2	0.9846177626348285	1.0000000000000000	0.9757198166771567	0.9812987236176437	0.98
3	0.9801857225104139	0.9757198166771567	1.0000000000000000	0.9812622584993598	0.97
4	0.9857305609394474	0.9812987236176437	0.9812622584993598	1.0000000000000000	0.97
5	0.9834548334956920	0.9811889063713322	0.9789177460265108	0.9712603312692165	1.00

5 rows × 67 columns

In [109]:
```
product_2 = item_item_similarity.loc[2][item_item_similarity.loc[2] < 1]
product_2.sort_values(ascending=False).head(10)
```

Out[109]:
```
product_ID
64   0.9857760092955804
36   0.9857760092955804
13   0.9856407745742413
61   0.9856186531269588
51   0.9846585936377441
1    0.9846177626348285
32   0.9845814977973557
28   0.9834989449710465
9    0.9834037797491455
41   0.9823423493695356
Name: 2, dtype: float64
```

I will pick product 32

In [110]:
```
top_10_similar_products_to_product_2 = list(product_2.sort_values(ascending=False
items_to_recommend_with_product_2 = products.\
                    loc[products['Products-productid'].isin(top_10_similar_produc
                    [['Products-productid','Products-Product']]

print('Products to be recommended with product of the ID 2')
items_to_recommend_with_product_2
```

```
Products to be recommended with product of the ID 2
```

Out[110]:

	Products-productid	Products-Product
14	28	DIG5630
18	61	PIX1905
26	13	CAM4175
27	36	GEM3940
37	41	DIG4505
43	51	DIG2995
44	64	YAL5320
55	9	CAM2410
62	1	YAL1940
63	32	YAL6285

This process can be repeated for ALL the products of interest

Forecasting Revenue for the first Quarter of 2017

In [111]:
```
import datetime as dt
import itertools
import statsmodels.graphics.tsaplots as sgt
import statsmodels.tsa.stattools as sts
from statsmodels.tsa.ar_model import AR
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from pmdarima.arima.utils import ndiffs
import statsmodels.api as sm
import pmdarima as pm
```

In [112]:
```
# !conda install -c saravji pmdarima
```

In [113]:
```
monthly_series = data[['Staff-Branch','Sales-Invoice Date','Sales-Period','Revenu
monthly_series = monthly_series.sort_values('Sales-Invoice Date').\
            set_index('Sales-Invoice Date').\
            resample('M')['Revenue'].sum().\
            to_frame().\
            rename(columns={'Revenue':'Monthly Revenue'})
monthly_series.head()
```

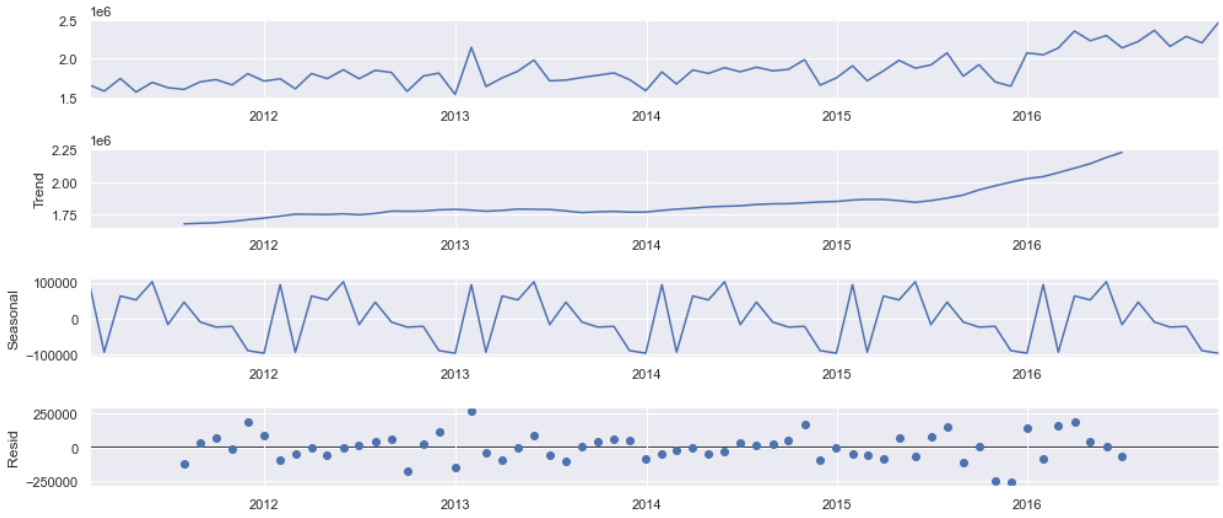
Out[113]:

Sales-Invoice Date	Monthly Revenue
2011-01-31	1,663,631.2775000000838190
2011-02-28	1,584,083.1774999999906868
2011-03-31	1,746,423.7775000000838190
2011-04-30	1,572,619.9225000001024455
2011-05-31	1,696,315.7549999998882413

Determing the Model Type

Checking the Seasonality and Time Components of the monthly Series

In [114]:
```
plt.rcParams["figure.figsize"] = (14,6)
components = seasonal_decompose(monthly_series)
components.plot();
```

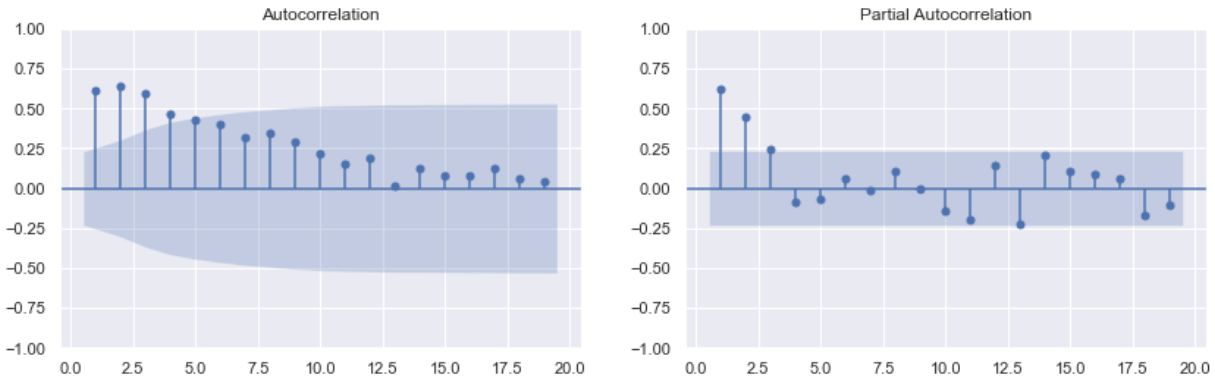


It can be seen from the graph that this series is not stationary due to seasonality and trend

Checking the Autocorrelatin and Partial Autocrrelation Functions Plots

In [115]:
```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14,4))
sgt.plot_acf(monthly_series['Monthly Revenue'],zero=False, ax=ax1)
sgt.plot_pacf(monthly_series['Monthly Revenue'],zero=False, ax=ax2)
plt.show()
plt.close()
```

```
C:\Users\ADMIN\miniconda3\envs\data_analysis\lib\site-packages\statsmodels\graphics\tsaplots.py:348: FutureWarning: The default method 'yw' can produce PACF values outside of the [-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker ('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```



From the graphs of acf and pacf, it can been that this is an ARIMA mode, and sine the series shows seasonality I will use SARIMAX model to include the seasonality effect

Determing p,d,q

Checking the Stationarity of the monthly_series

In [116]:
```
t_stat,p_value,lags, observation_num,t_critical_vals,ic = sts.adfuller(monthly_se
print('P_value: ',p_value)
if p_value <= 0.05:
    print('There is a significant evidence that this time series is stationary')
else:
    print('There is no significant evidence that this time series is stationary')
```

```
P_value:  0.9238628198728739
There is no significant evidence that this time series is stationary
```

Differencing

In [117]:
```
new_monthly_series = monthly_series['Monthly Revenue'].diff().dropna().to_frame()
new_monthly_series.head()
```

Out[117]:

Sales-Invoice Date	Monthly Revenue
2011-02-28	-79,548.1000000000931323
2011-03-31	162,340.6000000000931323
2011-04-30	-173,803.8549999999813735
2011-05-31	123,695.8324999997857958
2011-06-30	-68,061.2199999999720603

In [118]:
```
t_stat,p_value,lags, observation_num,t_critical_vals,ic = sts.adfuller(new_monthl
print('P_value: ',p_value)
if p_value <= 0.05:
    print('There is a significant evidence that this time series is stationary')
else:
    print('There is no significant evidence that this time series is stationary')
```

```
P_value:  7.351421662166509e-19
There is a significant evidence that this time series is stationary
```

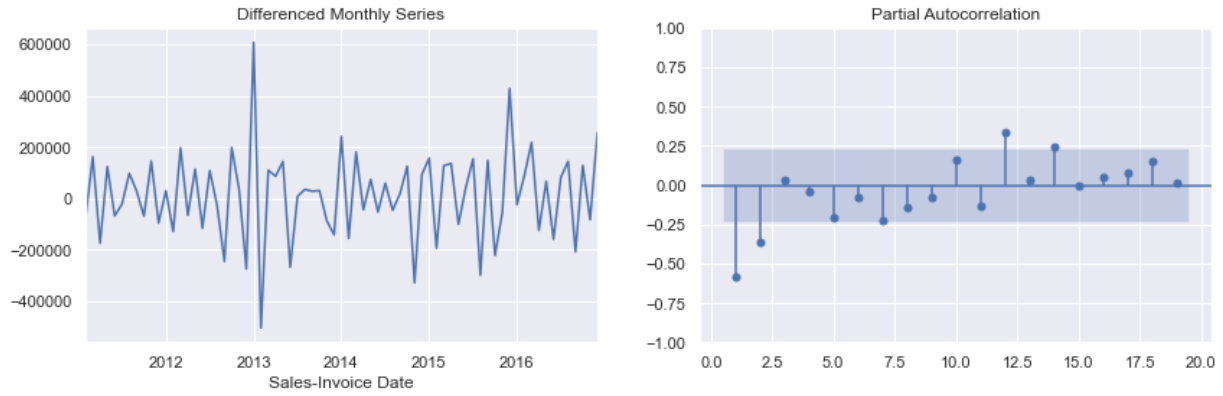
In [119]:
```
diff_num = ndiffs(monthly_series['Monthly Revenue'], test='adf')
diff_num
```

Out[119]: 1

It can be seen that d = 1 based on the order of differencing and the value of the ndiffs function

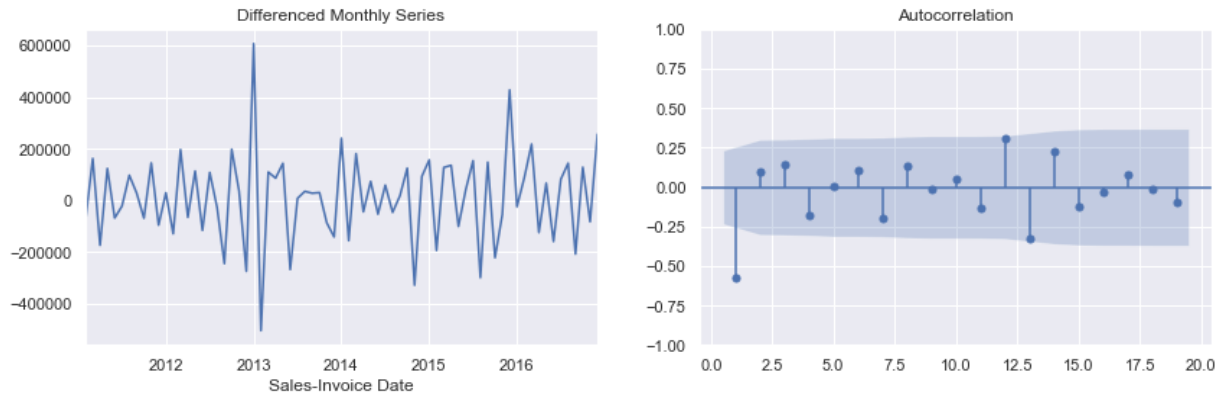
Determing p from the Partial-Autocorrelation Functions Plot

In [122]:
```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14,4))
new_monthly_series.plot(ax=ax1, title='Differenced Monthly Series', legend=False)
sgt.plot_pacf(new_monthly_series['Monthly Revenue'],zero=False, ax=ax2)
plt.show()
plt.close()
```



Determing q from the Autocorrelation Functions Plot

In [123]:
```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14,4))
new_monthly_series.plot(ax=ax1, title='Differenced Monthly Series', legend=False)
sgt.plot_acf(new_monthly_series['Monthly Revenue'],zero=False, ax=ax2)
plt.show()
plt.close()
```



It can be seen from the plots that p = 2 and q = 1

Splitting the Data

In [124]:
```
limit = int(len(monthly_series)*0.8)
train_data = monthly_series[:limit]
test_data = monthly_series[limit:]
```

Modelling Based on the Obtained p,d,q

In [125]:
```
first_model = sm.tsa.statespace.SARIMAX(train_data, order = (2,1,1),seasonal_orde
first_model_result = first_model.fit()
```

```
C:\Users\ADMIN\miniconda3\envs\data_analysis\lib\site-packages\statsmodels\tsa
\statespace\sarimax.py:997: UserWarning: Non-stationary starting seasonal autor
egressive Using zeros as starting parameters.
  warn('Non-stationary starting seasonal autoregressive'
C:\Users\ADMIN\miniconda3\envs\data_analysis\lib\site-packages\statsmodels\tsa
\statespace\sarimax.py:1009: UserWarning: Non-invertible starting seasonal movi
ng average Using zeros as starting parameters.
  warn('Non-invertible starting seasonal moving average'
```

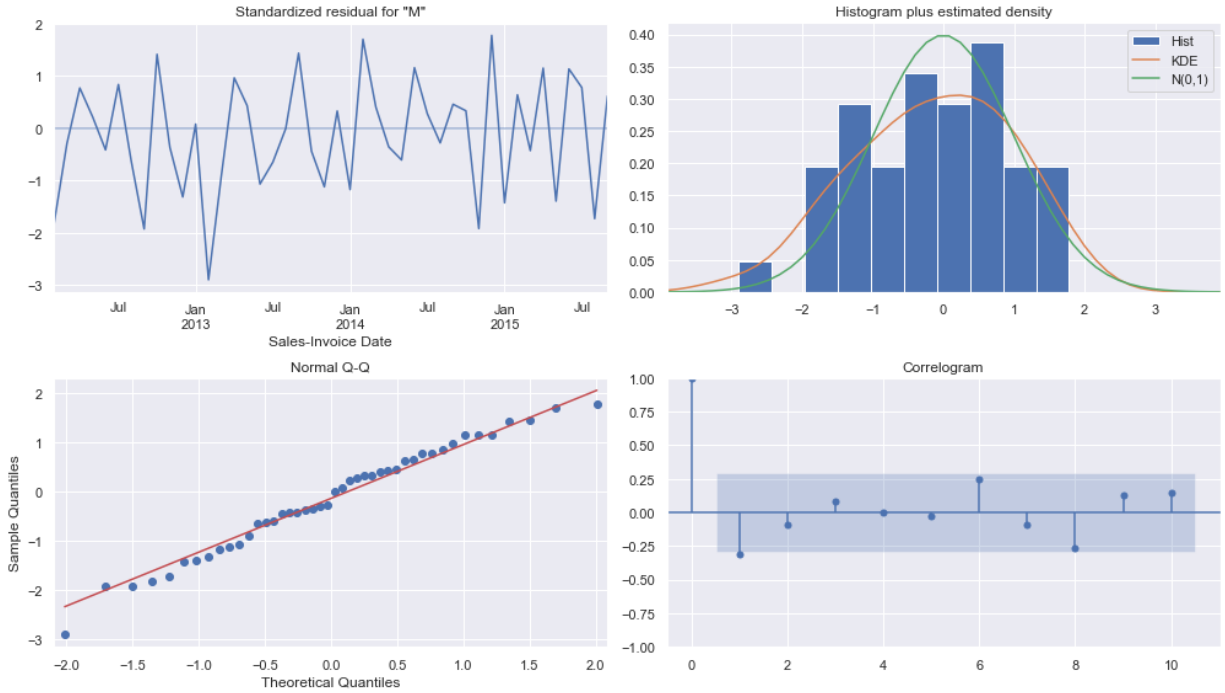
In [126]:
```
print(first_model_result.summary())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                     Monthly Revenue   No. Observations:
57
Model:             SARIMAX(2, 1, 1)x(2, 1, 1, 12)   Log Likelihood
-586.984
Date:                            Thu, 01 Sep 2022   AIC
1187.967
Time:                                    17:33:59   BIC
1200.457
Sample:                                01-31-2011   HQIC
1192.599
                                     - 09-30-2015
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.5309      0.767     -0.692      0.489      -2.034       0.972
ar.L2         -0.1595      0.127     -1.259      0.208      -0.408       0.089
ma.L1          0.3942      0.767      0.514      0.608      -1.110       1.898
ar.S.L12      -0.1891      3.352     -0.056      0.955      -6.758       6.380
ar.S.L24       0.0556      1.070      0.052      0.959      -2.041       2.153
ma.S.L12      -0.1034      3.366     -0.031      0.975      -6.702       6.495
sigma2      1.835e+10   8.72e-10    2.1e+19      0.000    1.83e+10    1.83e+10
==============================================================================
====
Ljung-Box (L1) (Q):                   4.55   Jarque-Bera (JB):
1.20
Prob(Q):                              0.03   Prob(JB):
0.55
Heteroskedasticity (H):               0.80   Skew:                            -
0.30
Prob(H) (two-sided):                  0.67   Kurtosis:
2.47
==============================================================================
====

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
[2] Covariance matrix is singular or near-singular, with condition number 1.99e
+35. Standard errors may be unstable.
```

It can be ssen from the p-values that non of the coefficients is of significane

plot_diagnostics of first_model

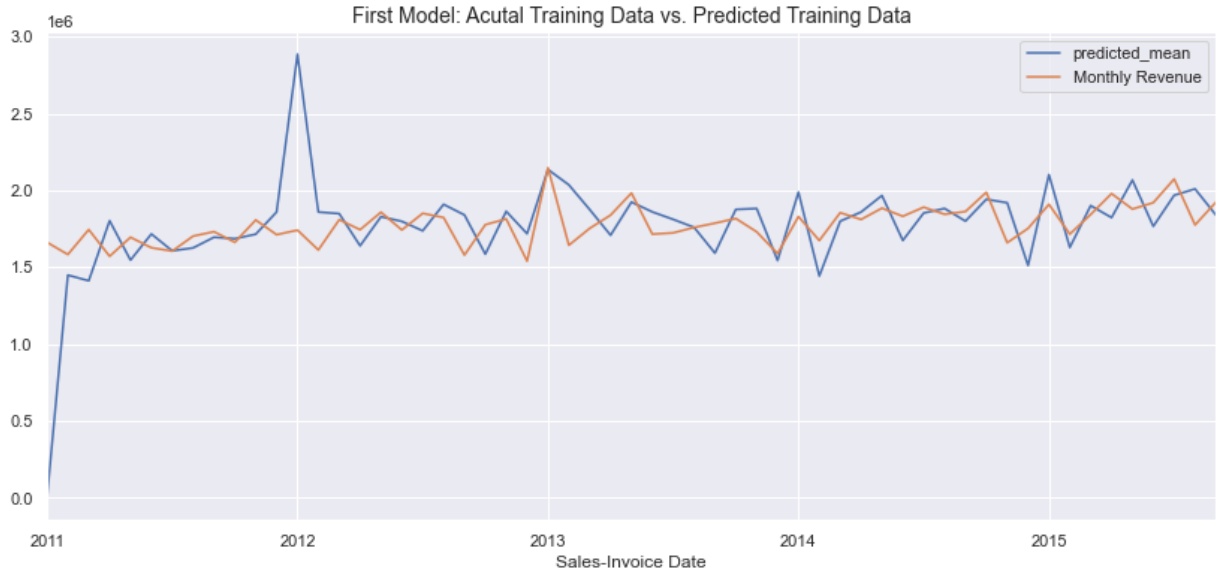
In [127]:
```
first_model_result.plot_diagnostics(figsize=(14, 8))
plt.tight_layout()
plt.show()
```



Comparing between first_model predicted train_data and acutal train_data

Plotting the first_model Predicted Mean vs. train_data

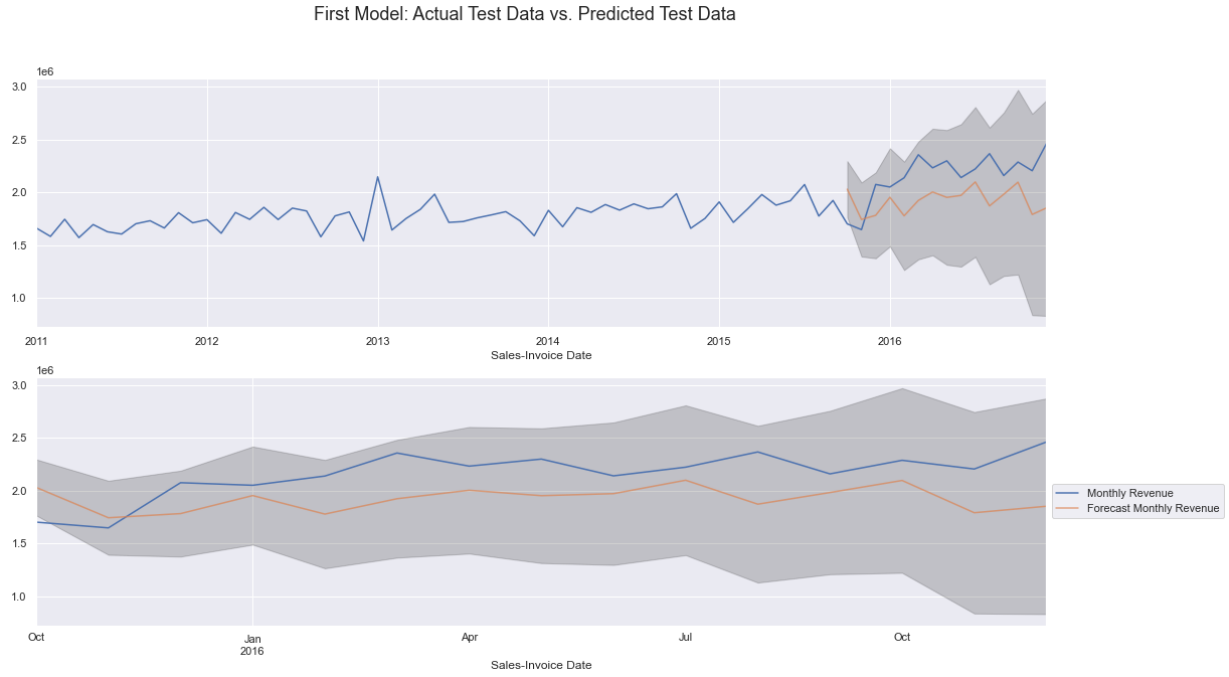
In [128]:
```
plt.figure(figsize=(14,6))
plt.title('First Model: Acutal Training Data vs. Predicted Training Data',size=14
ax1 = first_model_result.predict().plot()
monthly_series[:limit].plot(ax=ax1)
plt.legend()
plt.show()
```



Comparing between first_model predcited test_data and actual test_data

Plotting test_data vs Predcited Mean of test_data for the first_model

In [129]:
```
first_model_pred = first_model_result.get_prediction(start=test_data.index[0],enc
first_model_pred_ci = first_model_pred.conf_int()

fig, (ax1,ax2) = plt.subplots(2,1,figsize=(18,10))
plt.suptitle('First Model: Actual Test Data vs. Predicted Test Data',size=18)

monthly_series.plot(ax=ax1)
first_model_pred.predicted_mean.plot(ax=ax1, alpha=0.7)
ax1.fill_between(test_data.index, first_model_pred_ci.iloc[:,0], first_model_pred
ax1.get_legend().remove()

test_data.plot(ax=ax2,label='Actual Monthly Revenue')
first_model_pred.predicted_mean.plot(ax=ax2, label='Forecast Monthly Revenue', al
ax2.fill_between(test_data.index, first_model_pred_ci.iloc[:,0], first_model_pred

plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.show()
```



In [130]:
```
first_model_comparison_df = pd.concat([pd.DataFrame(first_model_pred.predicted_me
first_model_comparison_df.head()
```

Out[130]:

	predicted_mean	Monthly Revenue
2015-10-31	2,031,484.2652592037338763	1,702,833.2350000001024455
2015-11-30	1,744,298.3608001039829105	1,648,332.8274999998975545
2015-12-31	1,783,667.7946303046774119	2,076,207.7150000000838190
2016-01-31	1,954,434.4320674475748092	2,051,915.3525000000372529
2016-02-29	1,779,056.7350235355552286	2,139,425.0300000002607703

In [131]:
```
first_model_rmse = sqrt(mean_squared_error(test_data, first_model_pred.predicted_
print('First Mdoel RMSE: ', first_model_rmse)
```

```
First Mdoel RMSE:  326586.6077884129
```

In [132]:
```
first_model_mae =  mean_absolute_error(test_data, first_model_pred.predicted_mean
print('First Model Mean Absolute Error: ',first_model_mae)
```

```
First Model Mean Absolute Error:  290679.1835714649
```

Using Grid Search for Catching the Best Model based Based on AIC

Since there is a seasonality in the series, I will use SARIMAX model with S = 12

In [133]:
```
import itertools
```

In [134]:
```
p = P = range(0,3)
d = D = range(0,2)
q = Q = range(0,2)
S = 12

combination = list(itertools.product(p,d,q,P,D,Q))

arima_order = [(x[0],x[1],x[2]) for x in combination]
seasonal_order = [(x[3],x[4],x[5],S) for x in combination]

results_data = pd.DataFrame(columns = ['p','d','q','P','D','Q','AIC'])
```

In [135]:
```
for i in range(len(combination)):
    try:
        model = sm.tsa.statespace.SARIMAX(train_data,
                                          order = arima_order[i],
                                          seasonal_order = seasonal_order[i]
                                         )
        results = model.fit()

        results_data.loc[i,'p'] = arima_order[i][0]
        results_data.loc[i,'d'] = arima_order[i][1]
        results_data.loc[i,'q'] = arima_order[i][2]
        results_data.loc[i,'P'] = seasonal_order[i][0]
        results_data.loc[i,'D'] = seasonal_order[i][1]
        results_data.loc[i,'Q'] = seasonal_order[i][2]
        results_data.loc[i,'AIC'] = results.aic
    except:
        continue
```

. . .

In [136]:
```
best_parameters = results_data.sort_values('AIC').head(1)
best_parameters
```

Out[136]:

	p	d	q	P	D	Q	AIC
42	0	1	1	1	1	0	1,179.9764384796626473

In [137]:
```
p,d,q = int(best_parameters['p']),int(best_parameters['d']),int(best_parameters['
P,D,Q = int(best_parameters['P']),int(best_parameters['D']),int(best_parameters['
```

In [138]:
```
p,d,q,P,D,Q
```

Out[138]: (0, 1, 1, 1, 1, 0)

In [139]:
```
second_model = sm.tsa.statespace.SARIMAX(train_data,
                                         order = (p,d,q),
                                         seasonal_order = (P,D,Q,S)

                                        )

second_model_result = second_model.fit()
```

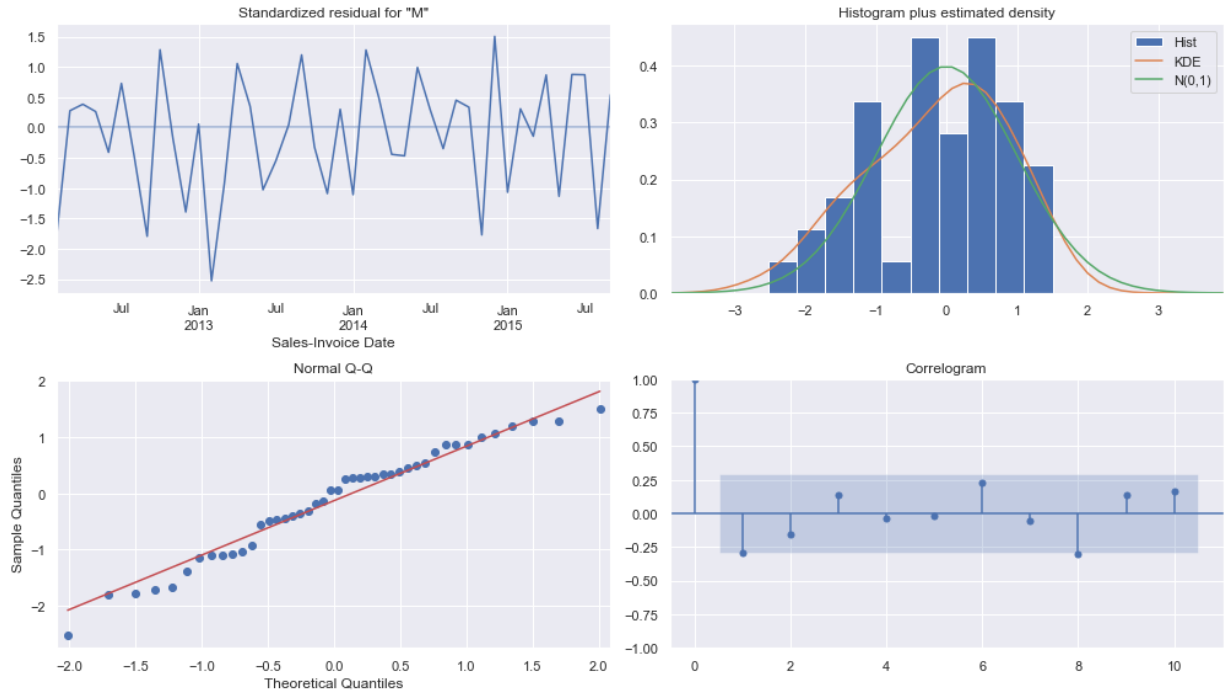
In [140]:
```
print(second_model_result.summary())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                      Monthly Revenue   No. Observations:
57
Model:             SARIMAX(0, 1, 1)x(1, 1, [], 12)   Log Likelihood
-586.988
Date:                             Thu, 01 Sep 2022   AIC
1179.976
Time:                                     17:35:17   BIC
1185.329
Sample:                                 01-31-2011   HQIC
1181.961
                                      - 09-30-2015
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.1690      0.074     -2.276      0.023      -0.314      -0.023
ar.S.L12      -0.3258      0.117     -2.784      0.005      -0.555      -0.096
sigma2      2.371e+10   1.19e-12   1.99e+22      0.000    2.37e+10    2.37e+10
===================================================================================
====
Ljung-Box (L1) (Q):                   4.02   Jarque-Bera (JB):
1.93
Prob(Q):                              0.04   Prob(JB):
0.38
Heteroskedasticity (H):               0.71   Skew:                             -
0.42
Prob(H) (two-sided):                  0.52   Kurtosis:
2.41
===================================================================================
====

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
[2] Covariance matrix is singular or near-singular, with condition number 1.55e
+38. Standard errors may be unstable.
```

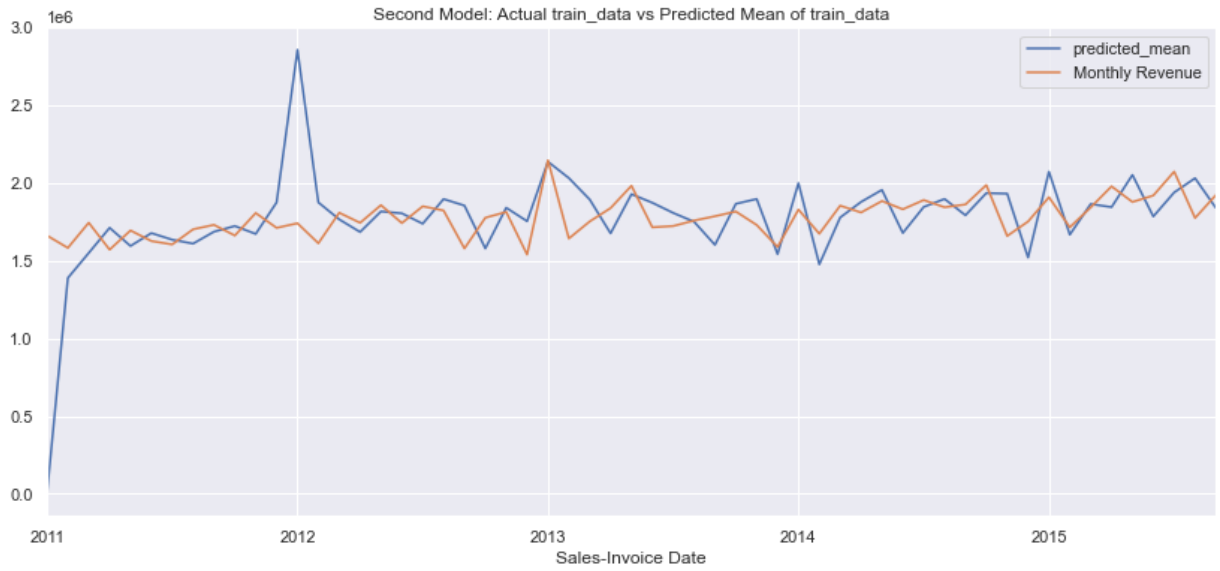
It can be seen from the p-values that the coeffiencts are of a significance

In [141]:
```
# second_model_resid_df = pd.DataFrame(second_model_results.resid)

# fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14,4))

# ax1.plot(second_model_resid_df)
# ax1.set_title('Residuals')
# ax2.hist(second_model_resid_df, bins = 30)
# ax2.set_title('Residuals Distribution')
# plt.show()
```

In [142]:
```
second_model_result.plot_diagnostics(figsize=(14, 8))
plt.tight_layout()
plt.show()
```



In [143]:
```
plt.figure(figsize=(14,6))
plt.title('Second Model: Actual train_data vs Predicted Mean of train_data')
ax1 = second_model_result.predict().plot()
monthly_series[:limit].plot(ax=ax1)
plt.legend()
plt.show()
```

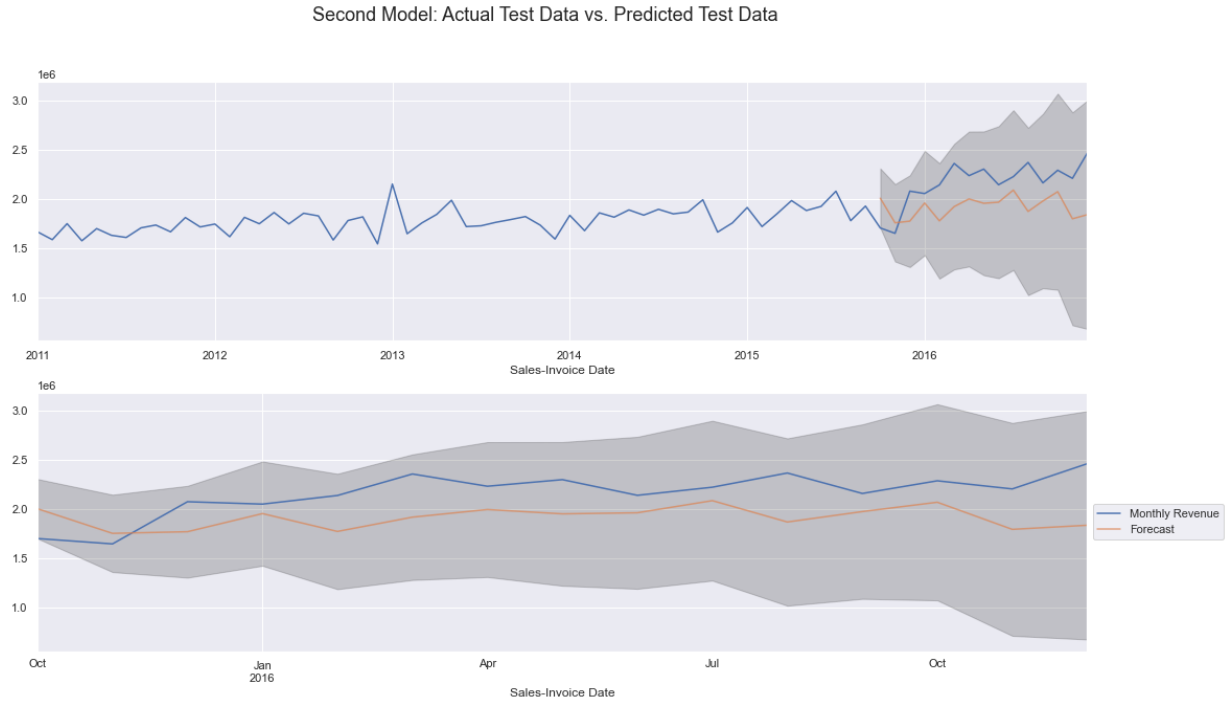


Forecasting on Test Data

In [144]:
```
# pred = best_model_results.get_prediction(start=test_data.index[0],end=test_data
# pred_ci = pred.conf_int()

# ax = monthly_series.plot(label='Actual', figsize = (14,4))
# pred.predicted_mean.plot(ax=ax, label='Forecast', alpha=0.7)
# ax.fill_between(test_data.index, pred_ci.iloc[:,0], pred_ci.iloc[:,1], color='k
# plt.legend(loc='upper left')

# plt.show()
```

In [145]:
```
# ax = test_data.plot(label='Actual', figsize = (14,4))
# pred.predicted_mean.plot(ax=ax, label='Forecast', alpha=0.7)
# ax.fill_between(test_data.index, pred_ci.iloc[:,0], pred_ci.iloc[:,1], color='k
# plt.legend(loc='upper left')

# plt.show()
```

In [146]:
```
second_model_pred = second_model_result.get_prediction(start=test_data.index[0],e
second_model_pred_ci = second_model_pred.conf_int()

fig, (ax1,ax2) = plt.subplots(2,1,figsize=(18,10))
plt.suptitle('Second Model: Actual Test Data vs. Predicted Test Data',size=18)

monthly_series.plot(ax=ax1,label='Actual')
second_model_pred.predicted_mean.plot(ax=ax1, label='Forecast', alpha=0.7)
ax1.fill_between(test_data.index, second_model_pred_ci.iloc[:,0], second_model_pr
ax1.get_legend().remove()

test_data.plot(ax=ax2,label='Actual')
second_model_pred.predicted_mean.plot(ax=ax2, label='Forecast', alpha=0.7)
ax2.fill_between(test_data.index, second_model_pred_ci.iloc[:,0], second_model_pr

plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```



In [147]:
```
second_model_comparison_df = pd.concat([pd.DataFrame(second_model_pred.predicted_
second_model_comparison_df.head()
```

Out[147]:

	predicted_mean	Monthly Revenue
2015-10-31	2,004,698.2028852326329798	1,702,833.2350000001024455
2015-11-30	1,755,196.0832696699071676	1,648,332.8274999998975545
2015-12-31	1,772,022.3054752915631980	2,076,207.7150000000838190
2016-01-31	1,955,879.4633571482263505	2,051,915.3525000000372529
2016-02-29	1,774,702.3875373133923858	2,139,425.0300000002607703

In [148]:
```
second_model_rmse = sqrt(mean_squared_error(test_data, second_model_pred.predicte
print('second Mdoel RMSE: ', second_model_rmse)
```

```
second Mdoel RMSE:  330770.44100480655
```

In [149]:
```
second_model_mae =  mean_absolute_error(test_data, second_model_pred.predicted_me
print('Second Model Mean Absolute Error: ',second_model_mae)
```

```
Second Model Mean Absolute Error:  296044.92027644836
```

From the analysis fo the results of the two models, it can be seen that the first model has slightly better accuracy. I will use it for the forecast

Final Model

In [150]:
```
final_model = sm.tsa.statespace.SARIMAX(monthly_series, order = (2,1,1),seasonal_
final_model_result = final_model.fit()
```

In [151]:
```
final_model_pred = final_model_result.get_prediction(start=pd.to_datetime('2017-0
                                                     end=pd.to_datetime('2017-03-
```

In [152]:
```
lower_limit = final_model_pred.conf_int().iloc[:,0]
prediction = final_model_pred.predicted_mean
upper_limit = final_model_pred.conf_int().iloc[:,1]
df_dict = {'Predicted Lower Limit':lower_limit,'Predicted Rev.':prediction,'Predi
```

In [153]:
```
final_model_result_df = pd.DataFrame(df_dict)
display(final_model_result_df.round(0))
final_model_result_df.plot(color=['red','blue','red'],figsize=(8,4))
plt.title('2017 First Quarter Predicted Reveneu')
plt.legend()
plt.show()
```

	Predicted Lower Limit	Predicted Rev.	Predicted Upper Limit
2017-01-31	2,234,851.0000000000000000	2,555,185.0000000000000000	2,875,519.0000000000000000
2017-02-28	2,105,221.0000000000000000	2,510,097.0000000000000000	2,914,973.0000000000000000
2017-03-31	2,250,983.0000000000000000	2,710,693.0000000000000000	3,170,402.0000000000000000