Retail_Sales_Analysis

August 25, 2021

1 Retail Sales Profit Analysis

[1]:
```python
import pandas as pd
```

[3]:
```python
df=pd.read_csv('Retail_SalesMarketing_ProfitCost.csv')
df.shape
```

[3]:
```
(84672, 16)
```

[4]:
```python
df.head()
```

[4]:
```
   Year       Product line  Product type             Product  \
0  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
1  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
2  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
3  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
4  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag

  Order method type Retailer country   Revenue   Planned revenue   \
0         Telephone    United States  315,044          437,477
1         Telephone           Canada   13,445           14,313
2         Telephone           Mexico       NaN              NaN
3         Telephone           Brazil       NaN              NaN
4         Telephone            Japan  181,120          235,237

   Product cost   Quantity    Unit cost   Unit price   Gross profit   \
0      158,372     66,385           3.0            7        156,673
1        6,299      2,172           3.0            7          7,146
2          NaN        NaN           NaN          NaN            NaN
3          NaN        NaN           NaN          NaN            NaN
4       89,413     35,696           3.0            7         91,707

   Unit sale price   Unnamed: 14  Unnamed: 15
0                5           NaN          NaN
1                6           NaN          NaN
2              NaN           NaN          NaN
3              NaN           NaN          NaN
4                5           NaN          NaN
```

[5]:
```
df.drop(columns=['Unnamed: 14', 'Unnamed: 15'] ,inplace=True)
```

[6]:
```
df.columns
```

[6]:
```
Index(['Year', 'Product line', 'Product type', 'Product', 'Order method type',
       'Retailer country', ' Revenue ', ' Planned revenue ', ' Product cost ',
       ' Quantity ', ' Unit cost ', ' Unit price ', ' Gross profit ',
       ' Unit sale price '],
      dtype='object')
```

1.0.1 Problem: Examine whether or not the retail store outdoor products are growing, maturing or declining. Compare your prospects for future sales with past performance by analyzing the industry average.

1.1 Data Cleaning

1.1.1 Lets clean up the data !!

Let's remove the spaces from the column names to remove ambiguity and better efficiency using strip function

[8]:
```
df.columns = [cl.strip() for cl in df.columns.tolist()]

df.columns= df.columns.str.replace(' ','_')
df.columns.tolist()
```

[8]:
```
['Year',
 'Product_line',
 'Product_type',
 'Product',
 'Order_method_type',
 'Retailer_country',
 'Revenue',
 'Planned_revenue',
 'Product_cost',
 'Quantity',
 'Unit_cost',
 'Unit_price',
 'Gross_profit',
 'Unit_sale_price']
```

[9]:
```
df.dtypes
```

[9]:
```
Year                  int64
Product_line         object
Product_type         object
Product              object
Order_method_type    object
Retailer_country     object
```

```
Revenue               object
Planned_revenue       object
Product_cost          object
Quantity              object
Unit_cost            float64
Unit_price            object
Gross_profit          object
Unit_sale_price       object
dtype: object
```

Since we have , in between 315,044 etc, it takes it as an object...so let us remove the commas

[10]:
```
df.replace(',','', regex=True, inplace=True)
```

[11]:
```
df.head(3)
```

[11]:
```
   Year        Product_line  Product_type             Product  \
0  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
1  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
2  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag

  Order_method_type Retailer_country   Revenue Planned_revenue Product_cost  \
0         Telephone    United States   315044          437477       158372
1         Telephone           Canada    13445           14313         6299
2         Telephone           Mexico      NaN             NaN          NaN

  Quantity  Unit_cost Unit_price Gross_profit Unit_sale_price
0   66385        3.0          7       156673               5
1    2172        3.0          7         7146               6
2      NaN        NaN        NaN          NaN             NaN
```

In Pandas missing data is represented by two value:

- None: None is a Python singleton object that is often used for missing data in Python code.
- NaN : NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation (NaN is numpy type)

We find using isnull() and not notnull()

[13]:
```
null_series=df.isnull().sum()
# type(null_df)
null_series
```

[13]:
```
Year                   0
Product_line           0
Product_type           0
Product                0
Order_method_type      0
Retailer_country       0
```

```
Revenue               59929
Planned_revenue       59929
Product_cost          59929
Quantity              59929
Unit_cost             59929
Unit_price            59929
Gross_profit          59929
Unit_sale_price       59929
dtype: int64
```

These are the null/missing values in the dataset

[14]:
```
notnull_series=df.notnull().sum()
# type(notnull_df)
notnull_series
```

[14]:
```
Year                  84672
Product_line          84672
Product_type          84672
Product               84672
Order_method_type     84672
Retailer_country      84672
Revenue               24743
Planned_revenue       24743
Product_cost          24743
Quantity              24743
Unit_cost             24743
Unit_price            24743
Gross_profit          24743
Unit_sale_price       24743
dtype: int64
```

These are the values that are Not-Null. Let us combine the null and not null to find percentage of missing values in each column

We are concatenating two series null & not null ,into a dataframe vertically (columns) by setting axis =1 and naming them . Missing_df.columns[list] we are assigning the names

[15]:
```
missing_df=pd.concat([null_series,notnull_series], axis=1).reset_index()
missing_df.columns=['Column_name','Null_count','Not_Null_Count']
```

[16]:
```
missing_df['Total_count']= missing_df['Null_count'] +␣
 ↪missing_df['Not_Null_Count']

missing_df['Missing_values_percent']= (missing_df['Null_count']/␣
 ↪missing_df['Total_count']) *100
missing_df
# Null_count/ (Null_count+Not_Null_Count)* 100
```

[16]:
```
           Column_name  Null_count  Not_Null_Count  Total_count  \
0                 Year           0           84672        84672
1         Product_line           0           84672        84672
2         Product_type           0           84672        84672
3              Product           0           84672        84672
4    Order_method_type           0           84672        84672
5     Retailer_country           0           84672        84672
6              Revenue       59929           24743        84672
7      Planned_revenue       59929           24743        84672
8         Product_cost       59929           24743        84672
9             Quantity       59929           24743        84672
10           Unit_cost       59929           24743        84672
11          Unit_price       59929           24743        84672
12        Gross_profit       59929           24743        84672
13     Unit_sale_price       59929           24743        84672

    Missing_values_percent
0                 0.000000
1                 0.000000
2                 0.000000
3                 0.000000
4                 0.000000
5                 0.000000
6                70.777825
7                70.777825
8                70.777825
9                70.777825
10               70.777825
11               70.777825
12               70.777825
13               70.777825
```

We can find that we have a 70% missing data from column 6 to column 13…which is a lot. We have to eventually delete the data,but lets see which subgroup has maximum no. of missing values

[17]:
```
df.groupby(['Product_line'])['Revenue','Year'].count()
```

```
<ipython-input-17-753fef286776>:1: FutureWarning: Indexing with multiple keys
(implicitly converted to a tuple of keys) will be deprecated, use a list
instead.
  df.groupby(['Product_line'])['Revenue','Year'].count()
```

[17]:
```
                          Revenue   Year
Product_line
Camping Equipment            8375  24108
Golf Equipment               2763   8820
Mountaineering Equipment     2947  12348
Outdoor Protection           2944   8820
```

```
Personal Accessories          7714  30576
```

We can see that the null count is distributed across all the prroduct lines, (we narrowed down to check if a particular sub-granular group like product line had lots of missing data wrt only one). We can proceed with deleting them.

There was a **70%** data loss happening across all metrics which cannot be manipulated by filling values using **Fillna**.

[18]:
```
df=df.dropna()
```

[19]:
```
df.head(3)
```

[19]:
```
   Year        Product_line  Product_type              Product  \
0  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
1  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
4  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag

  Order_method_type Retailer_country   Revenue Planned_revenue Product_cost  \
0         Telephone    United States   315044          437477       158372
1         Telephone           Canada    13445           14313         6299
4         Telephone            Japan   181120          235237        89413

  Quantity  Unit_cost Unit_price Gross_profit Unit_sale_price
0    66385        3.0          7       156673               5
1     2172        3.0          7         7146               6
4    35696        3.0          7        91707               5
```

[20]:
```
df.reset_index(drop= True, inplace=True)

# Drop= true as we dont want to carry forward that index. Index was 0,1,4
```

[21]:
```
df.head(3)
```

[21]:
```
   Year        Product_line  Product_type              Product  \
0  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
1  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
2  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag

  Order_method_type Retailer_country   Revenue Planned_revenue Product_cost  \
0         Telephone    United States   315044          437477       158372
1         Telephone           Canada    13445           14313         6299
2         Telephone            Japan   181120          235237        89413

  Quantity  Unit_cost Unit_price Gross_profit Unit_sale_price
0    66385        3.0          7       156673               5
1     2172        3.0          7         7146               6
2    35696        3.0          7        91707               5
```

[22]:
```
df.shape
```

[22]:
```
(24743, 14)
```

[23]:
```
df.dtypes
```

[23]:
```
Year                    int64
Product_line           object
Product_type           object
Product                object
Order_method_type      object
Retailer_country       object
Revenue                object
Planned_revenue        object
Product_cost           object
Quantity               object
Unit_cost             float64
Unit_price             object
Gross_profit           object
Unit_sale_price        object
dtype: object
```

Let us strip extra spaces, - and such noises in the columns...

[24]:
```
df=df.assign(Revenue= df['Revenue'].str.strip(),
             Planned_revenue= df['Planned_revenue'].str.strip(),
             Product_cost= df['Product_cost'].str.strip(),
             Quantity= df['Quantity'].str.strip(),
             Unit_price= df['Unit_price'].str.strip(),
             Gross_profit= df['Gross_profit'].str.strip(),
             Unit_sale_price= df['Unit_sale_price'].str.strip()
            )
# df.iloc[100]
# df['Revenue'].iloc[100]
```

[25]:
```
df=df.assign(Revenue= df['Revenue'].str.replace('-',''),
             Planned_revenue= df['Planned_revenue'].str.replace('-',''),
             Product_cost= df['Product_cost'].str.replace('-',''),
             Quantity= df['Quantity'].str.replace('-',''),
             Unit_price= df['Unit_price'].str.replace('-',''),
             Gross_profit= df['Gross_profit'].str.replace('-',''),
             Unit_sale_price= df['Unit_sale_price'].str.replace('-','')
            )
```

Converting to numeric types....

[26]:
```
df= df.assign(
    Revenue= pd.to_numeric(df['Revenue']),
```

```
    Planned_revenue= pd.to_numeric(df['Planned_revenue']),
    Product_cost= pd.to_numeric(df['Product_cost']),
    Quantity= pd.to_numeric(df['Quantity']),
    Unit_price= pd.to_numeric(df['Unit_price']),
#     Gross_profit= pd.to_numeric(df['Gross_profit']),
    Unit_sale_price= pd.to_numeric(df['Unit_sale_price'])
)
```

[27]:
```
df.Gross_profit.iloc[301]
```

[27]: '(1119)'

Since it has paranthesis in place for negative values, lets convert them and change data type

[28]:
```
df['Gross_profit']= (df['Gross_profit'].replace( '[\$,)]','', regex=True )
               .replace( '[(]','-',   regex=True ).astype(float))
```

[29]:
```
df.dtypes
```

[29]:
```
Year                  int64
Product_line         object
Product_type         object
Product              object
Order_method_type    object
Retailer_country     object
Revenue             float64
Planned_revenue       int64
Product_cost          int64
Quantity              int64
Unit_cost           float64
Unit_price            int64
Gross_profit        float64
Unit_sale_price     float64
dtype: object
```

[30]:
```
df.isnull().sum()
```

[30]:
```
Year                 0
Product_line         0
Product_type         0
Product              0
Order_method_type    0
Retailer_country     0
Revenue             76
Planned_revenue      0
Product_cost         0
Quantity             0
Unit_cost            0
```

```
Unit_price           0
Gross_profit         0
Unit_sale_price     76
dtype: int64
```

Revenue and Unit sale price still have 76 null values...lets fill them with 0 as they are not missing. They have no revenue and no sale price

[31]:
```
df=df.fillna(0)
```

[32]:
```
df.isnull().sum()
```

[32]:
```
Year                0
Product_line        0
Product_type        0
Product             0
Order_method_type   0
Retailer_country    0
Revenue             0
Planned_revenue     0
Product_cost        0
Quantity            0
Unit_cost           0
Unit_price          0
Gross_profit        0
Unit_sale_price     0
dtype: int64
```

[33]:
```
df.columns
```

[33]:
```
Index(['Year', 'Product_line', 'Product_type', 'Product', 'Order_method_type',
       'Retailer_country', 'Revenue', 'Planned_revenue', 'Product_cost',
       'Quantity', 'Unit_cost', 'Unit_price', 'Gross_profit',
       'Unit_sale_price'],
      dtype='object')
```

[37]:
```
df.Product_type.unique()
```

[37]:
```
array(['Cooking Gear', 'Tents', 'Sleeping Bags', 'Packs', 'Lanterns',
       'Watches', 'Eyewear', 'Knives', 'Binoculars', 'Navigation',
       'Insect Repellents', 'Sunscreen', 'First Aid', 'Irons', 'Woods',
       'Putters', 'Golf Accessories', 'Rope', 'Safety',
       'Climbing Accessories', 'Tools'], dtype=object)
```

[38]:
```
df.Product.unique()
```

[38]:
```
array(['TrailChef Water Bag', 'TrailChef Canteen',
       'TrailChef Kitchen Kit', 'TrailChef Cup', 'TrailChef Cook Set',
```

```
'TrailChef Deluxe Cook Set', 'TrailChef Single Flame',
'TrailChef Double Flame', 'TrailChef Kettle', 'TrailChef Utensils',
'Star Lite', 'Star Dome', 'Star Gazer 2', 'Star Gazer 3',
'Star Gazer 6', 'Star Peg', 'Hibernator Lite', 'Hibernator',
'Hibernator Extreme', 'Hibernator Self - Inflating Mat',
'Hibernator Pad', 'Hibernator Pillow', 'Hibernator Camp Cot',
'Canyon Mule Climber Backpack', 'Canyon Mule Weekender Backpack',
'Canyon Mule Journey Backpack', 'Canyon Mule Extreme Backpack',
'Canyon Mule Cooler', 'Canyon Mule Carryall', 'Firefly Lite',
'Firefly Mapreader', 'Firefly 2', 'Firefly 4', 'Firefly Extreme',
'Firefly Multi-light', 'EverGlow Single', 'EverGlow Double',
'EverGlow Kerosene', 'EverGlow Butane', 'EverGlow Lamp',
'Flicker Lantern', 'Mountain Man Analog', 'Mountain Man Digital',
'Mountain Man Deluxe', 'Mountain Man Combination',
'Mountain Man Extreme', 'Venue', 'Infinity', 'Lux', 'Sam', 'TX',
'Legend', 'Kodiak', 'Polar Sun', 'Polar Ice', 'Polar Sports',
'Polar Wave', 'Polar Extreme', 'Bella', 'Capri', 'Cat Eye',
'Dante', 'Fairway', 'Inferno', 'Maximus', 'Trendi', 'Zone',
'Hawk Eye', 'Single Edge', 'Double Edge', 'Edge Extreme',
'Bear Edge', 'Bear Survival Edge', 'Max Gizmo', 'Pocket Gizmo',
'Seeker 35', 'Seeker 50', 'Seeker Extreme', 'Seeker Mini',
'Opera Vision', 'Ranger Vision', 'Glacier Basic', 'Glacier Deluxe',
'Glacier GPS', 'Glacier GPS Extreme', 'Trail Master',
'Trail Scout', 'Trail Star', 'BugShield Natural',
'BugShield Spray', 'BugShield Lotion Lite', 'BugShield Lotion',
'BugShield Extreme', 'Sun Blocker', 'Sun Shelter Stick',
'Sun Shelter 15', 'Sun Shelter 30', 'Sun Shield',
'Compact Relief Kit', 'Deluxe Family Relief Kit',
'Calamine Relief', 'Aloe Relief', 'Insect Bite Relief',
'Hailstorm Steel Irons', 'Hailstorm Titanium Irons',
'Lady Hailstorm Steel Irons', 'Lady Hailstorm Titanium Irons',
'Hailstorm Titanium Woods Set', 'Hailstorm Steel Woods Set',
'Lady Hailstorm Titanium Woods Set',
'Lady Hailstorm Steel Woods Set', 'Course Pro Putter',
'Blue Steel Putter', 'Blue Steel Max Putter',
'Course Pro Golf and Tee Set', 'Course Pro Umbrella',
'Course Pro Golf Bag', 'Course Pro Gloves', 'Zodiak', 'Retro',
'Astro Pilot', 'Sky Pilot', 'Husky Rope 50', 'Husky Rope 60',
'Husky Rope 100', 'Husky Rope 200', 'Granite Climbing Helmet',
'Husky Harness', 'Husky Harness Extreme', 'Granite Signal Mirror',
'Granite Carabiner', 'Granite Belay', 'Granite Pulley',
'Firefly Climbing Lamp', 'Firefly Charger',
'Firefly Rechargeable Battery', 'Granite Chalk Bag', 'Granite Ice',
'Granite Hammer', 'Granite Shovel', 'Granite Grip', 'Granite Axe',
'Granite Extreme', 'Auto Pilot'], dtype=object)
```

[36]:
```
df.Product_line.unique()
```

[36]:
```
array(['Camping Equipment', 'Personal Accessories', 'Outdoor Protection',
       'Golf Equipment', 'Mountaineering Equipment'], dtype=object)
```

[39]:
```
df.head(3)
```

[39]:
```
   Year       Product_line  Product_type             Product  \
0  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
1  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
2  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag

  Order_method_type Retailer_country   Revenue  Planned_revenue  Product_cost  \
0         Telephone    United States  315044.0           437477        158372
1         Telephone           Canada   13445.0            14313          6299
2         Telephone            Japan  181120.0           235237         89413

   Quantity  Unit_cost  Unit_price  Gross_profit  Unit_sale_price
0     66385        3.0           7      156673.0              5.0
1      2172        3.0           7        7146.0              6.0
2     35696        3.0           7       91707.0              5.0
```

1.1.2 The entire data set is cleaned…let us do the exploratory analysis

1.2 Exploratory Analysis

- Summary statistics
- Distribution of Records across product lines
- Distribution of data across countries
- Feature selection Conducting correlation to find which is the best parameter that contri butees to sales profits/ highest revenue

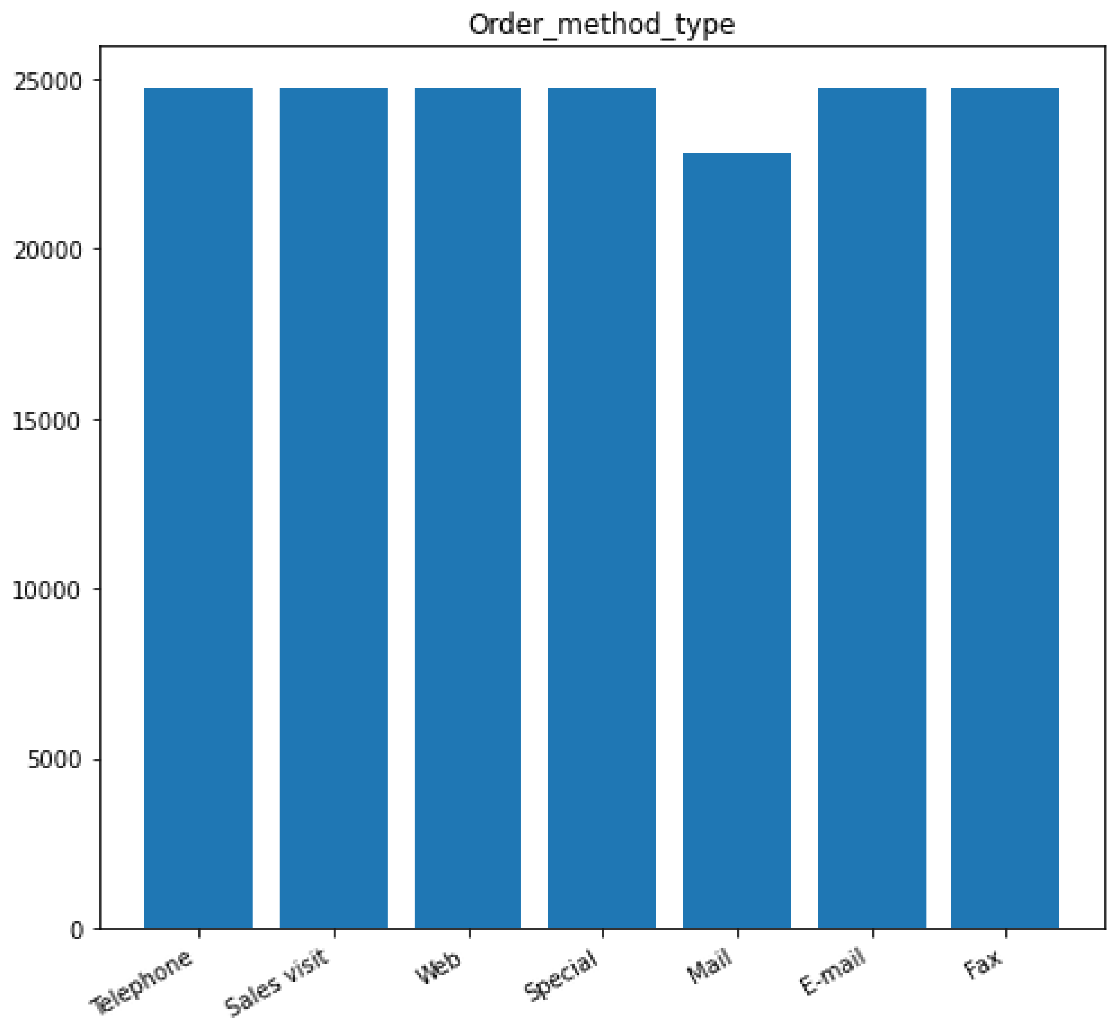
[292]:
```
# Summary statistics

df.describe()
```

[292]:
```
               Year       Revenue  Planned_revenue  Product_cost  \
count  24743.000000  2.474300e+04     2.474300e+04  2.474300e+04
mean    2016.345067  1.894183e+05     1.988176e+05  1.116252e+05
std        1.073106  3.907509e+05     4.025355e+05  2.384156e+05
min     2015.000000  0.000000e+00     1.600000e+01  6.000000e+00
25%     2015.000000  1.857900e+04     1.955700e+04  9.431500e+03
50%     2016.000000  5.986700e+04     6.390700e+04  3.278400e+04
75%     2017.000000  1.901930e+05     2.039955e+05  1.113710e+05
max     2018.000000  1.005429e+07     1.005429e+07  6.756853e+06

           Quantity     Unit_cost    Unit_price  Gross_profit  \
count  24743.000000  24743.000000  24743.000000  2.474300e+04
mean    3606.559067     84.946530    156.056541  7.779312e+04
std     8777.721091    131.108962    246.805361  1.581223e+05
```

```
min          1.000000      1.000000      2.000000 -1.816000e+04
25%        328.000000     11.000000     23.000000  8.333000e+03
50%       1043.000000     37.000000     67.000000  2.579400e+04
75%       3288.000000     80.000000    148.000000  7.825400e+04
max     313628.000000    690.000000   1360.000000  3.521098e+06

       Unit_sale_price
count     24743.000000
mean        147.254900
std         232.045043
min           0.000000
25%          20.000000
50%          63.000000
75%         141.000000
max        1308.000000
```

[51]:

```
import matplotlib.pyplot as plt
fig, axs = plt.subplots(figsize=(8,8))
axs.bar(df['Order_method_type'], df.index)
axs.set_title('Order_method_type')
fig.autofmt_xdate()
```



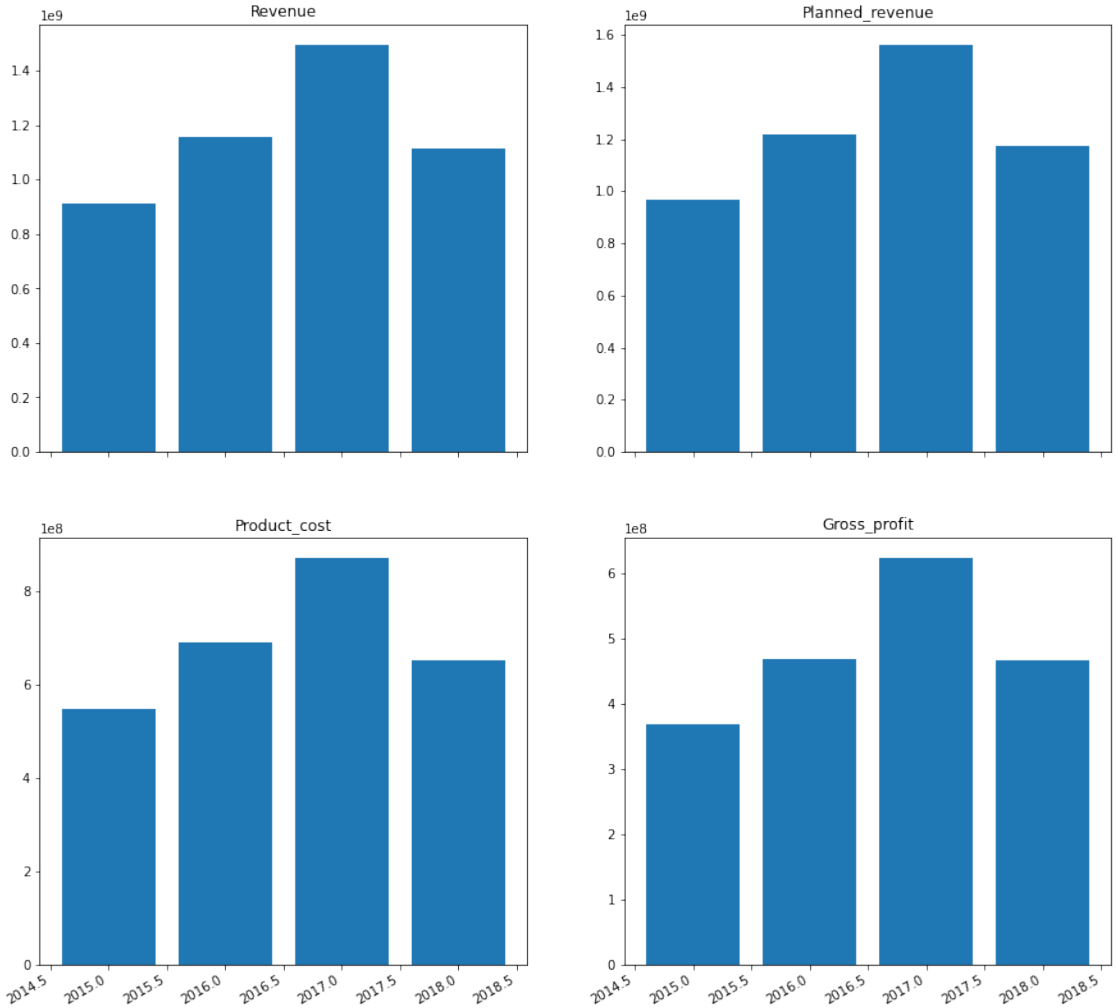
Revenue over the years

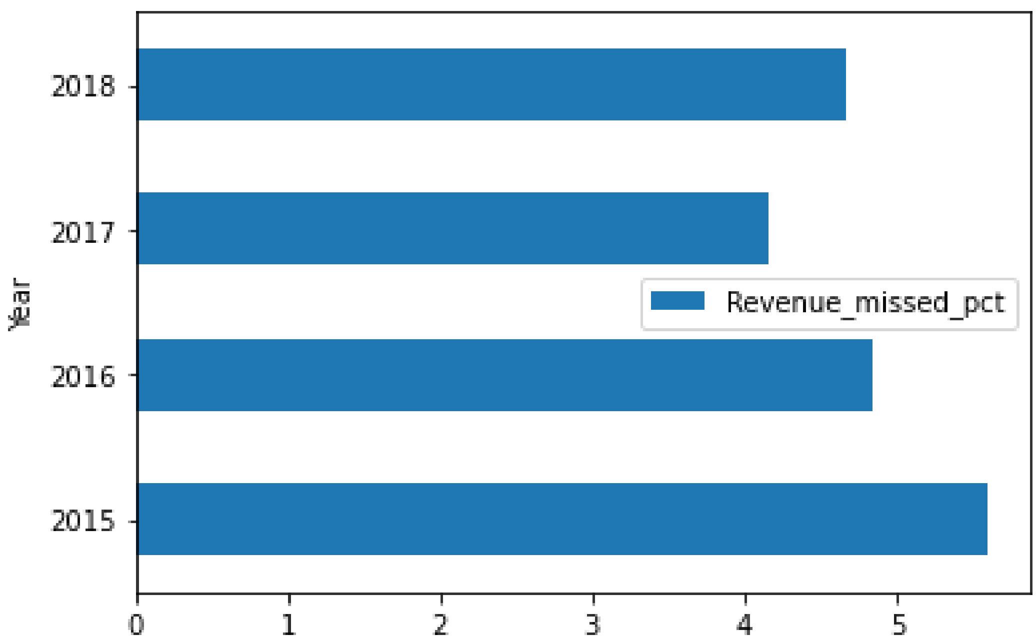
[62]:
```
import matplotlib.pyplot as plt
def group_analysis(column_name):
    # Generate the results based on the given column_name
    results= df.groupby(column_name).sum()
    results['Revenue_missed_pct'] = ␣
↪((results['Planned_revenue']-results['Revenue'])/
↪(results['Planned_revenue']))*100.0

    # Then plot the KPI (Key Performance Indicies) for the considered␣
↪column_name
    fig, axs = plt.subplots(2, 2, figsize=(15,15))
    axs[0,0].bar(results.index,results['Revenue'])
    axs[0,0].set_title('Revenue')
```

```python
    fig.autofmt_xdate()

    axs[0,1].bar(results.index,results['Planned_revenue'])
    axs[0,1].set_title('Planned_revenue')
    fig.autofmt_xdate()

    axs[1,0].bar(results.index,results['Product_cost'])
    axs[1,0].set_title('Product_cost')
    fig.autofmt_xdate()

    axs[1,1].bar(results.index,results['Gross_profit'])
    axs[1,1].set_title('Gross_profit')
    fig.autofmt_xdate()

#     Plot for missed revenue and the column name
    results.plot.barh(y='Revenue_missed_pct')
    plt.ylabel=('Revenue_missed_pct')
    plt.xlabel=(results.index)

    #     ax = results.plot.barh(x='Revenue_missed_pct', y=results.index)
    return results
```

[63]:
```python
group_analysis('Year')
```

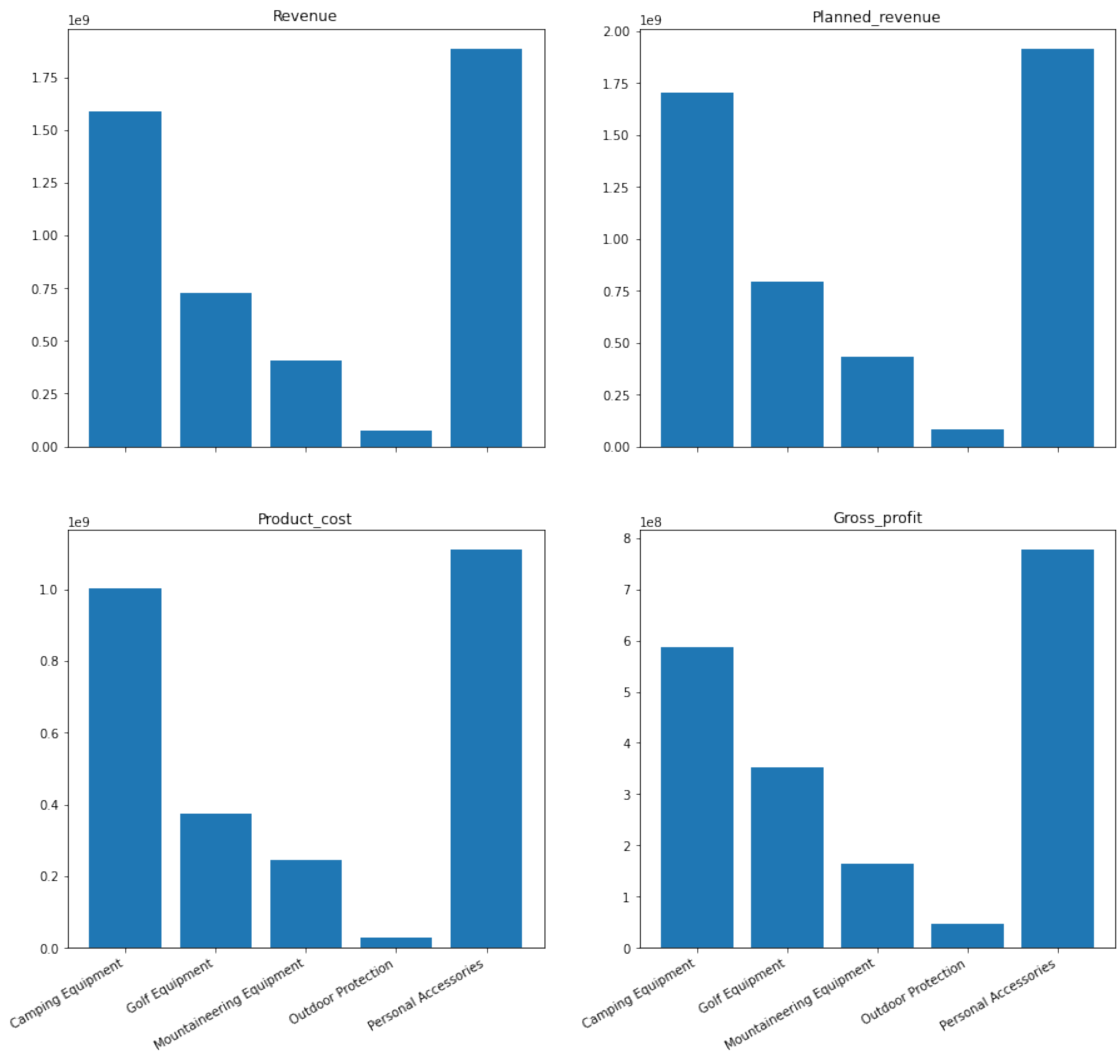
[63]:
```
           Revenue  Planned_revenue  Product_cost  Quantity  Unit_cost  \
Year
2015  9.143529e+08        968475273     546735041  20174730   605714.0
2016  1.159196e+09       1218177535     690512038  23524685   605797.0
2017  1.495891e+09       1560738939     872630604  25941790   495691.0
2018  1.117336e+09       1171950964     652063489  19595886   394630.0

      Unit_price  Gross_profit  Unit_sale_price  Revenue_missed_pct
Year
2015     1115154   367617948.0        1042079.0            5.588411
2016     1092246   468683778.0        1031078.0            4.841811
2017      921617   623260635.0         877109.0            4.154940
2018      732290   465272886.0         693262.0            4.660142
```

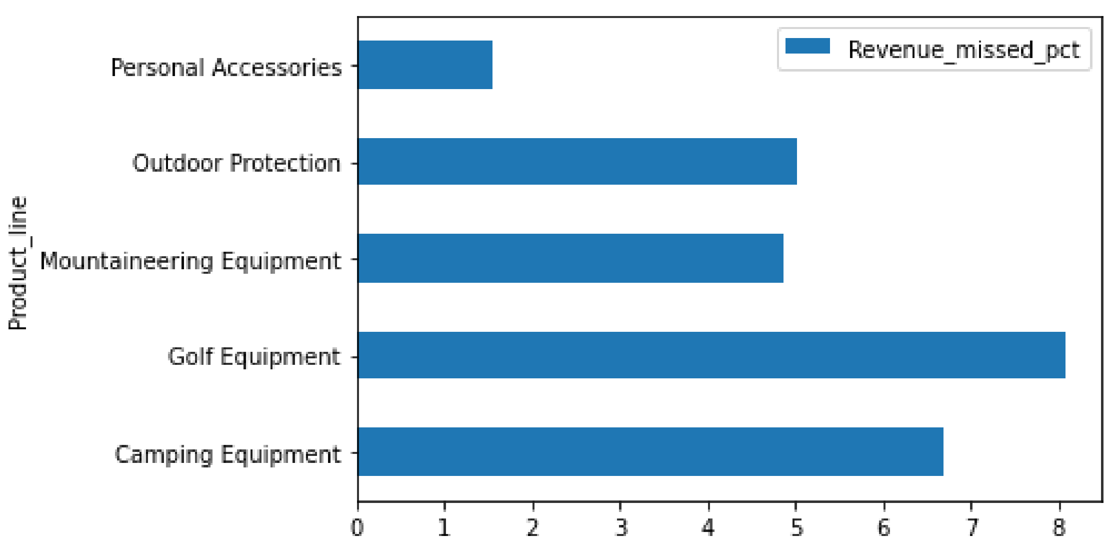

Revenue
1e9
Planned_revenue
1e9
Product_cost
1e8
Gross_profit
1e8
2014.5
2015.0
2015.5
2016.0
2016.5
2017.0
2017.5
2018.0
2018.5



[64]:
```
group_analysis('Product_line')
```

[64]:
```
                          Year       Revenue  Planned_revenue  \
Product_line
Camping Equipment     16886376  1.589037e+09       1703124790
Golf Equipment         5571060  7.264114e+08        790261053
Mountaineering Equipment  5943678  4.096602e+08    430568075
Outdoor Protection     5935833  7.599434e+07         80005280
Personal Accessories  15553479  1.885673e+09       1915383513

                      Product_cost  Quantity  Unit_cost  Unit_price  \
Product_line
Camping Equipment       1002237787  27301149   754776.0     1268118
Golf Equipment           374217745   5113701   693534.0     1419337
Mountaineering Equipment  246384239  9900091   185392.0      302861
Outdoor Protection        30011034  12014445    10472.0       25602
Personal Accessories    1109090367  34907705   457658.0      845389

                      Gross_profit  Unit_sale_price  Revenue_missed_pct
Product_line
Camping Equipment      586799155.0        1191877.0            6.698751
Golf Equipment         352193680.0        1322223.0            8.079568
Mountaineering Equipment 163275949.0       287501.0            4.855878
Outdoor Protection      45983327.0          24348.0            5.013339
Personal Accessories   776583136.0         817579.0            1.551127
```


Revenue
1e9
Planned_revenue
1e9
Product_cost
1e9
Gross_profit
1e8
Camping Equipment
Golf Equipment
Mountaineering Equipment
Outdoor Protection
Personal Accessories


Revenue_missed_pct
Product_line
Personal Accessories
Outdoor Protection
Mountaineering Equipment
Golf Equipment
Camping Equipment

[65]:

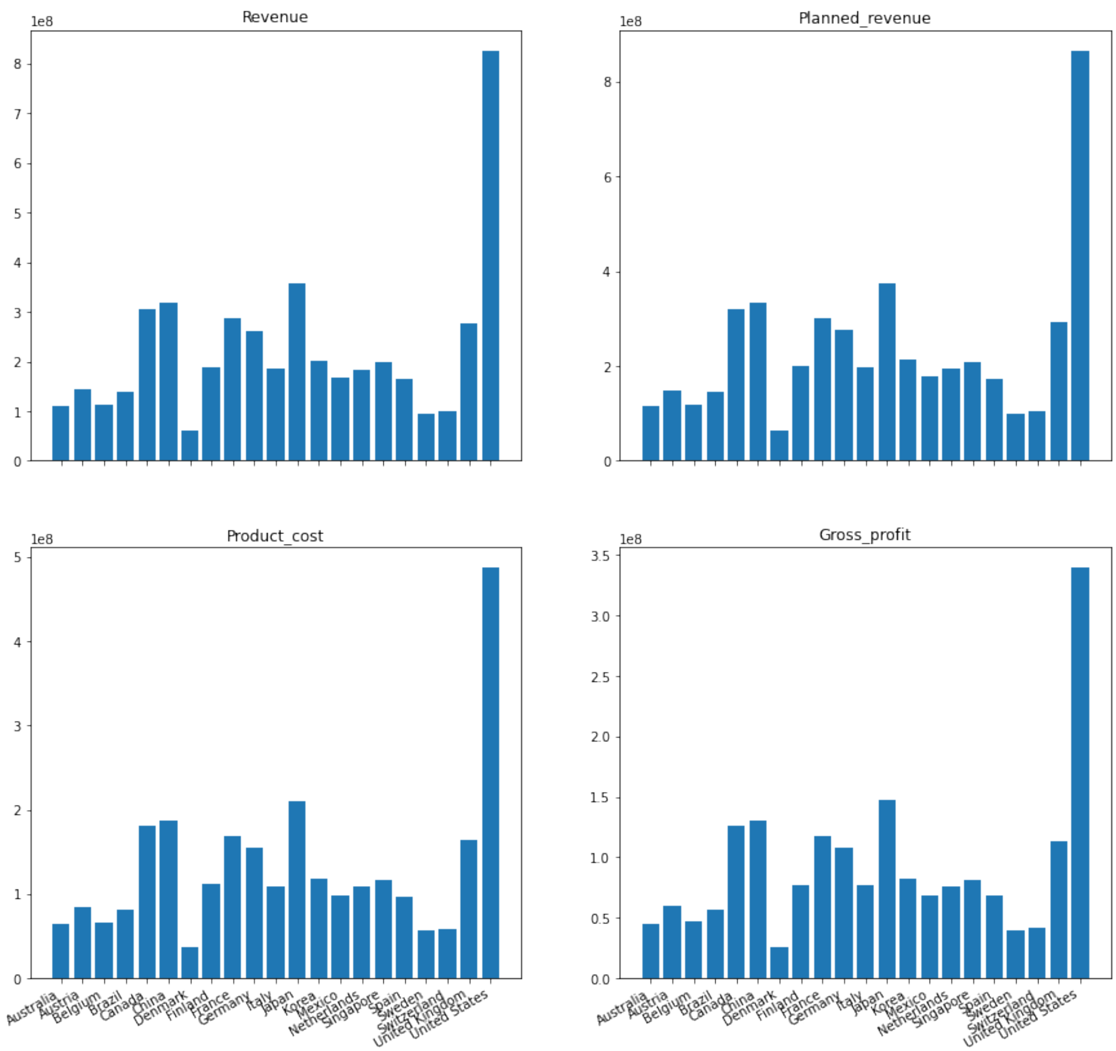
```
group_analysis('Retailer_country')
```

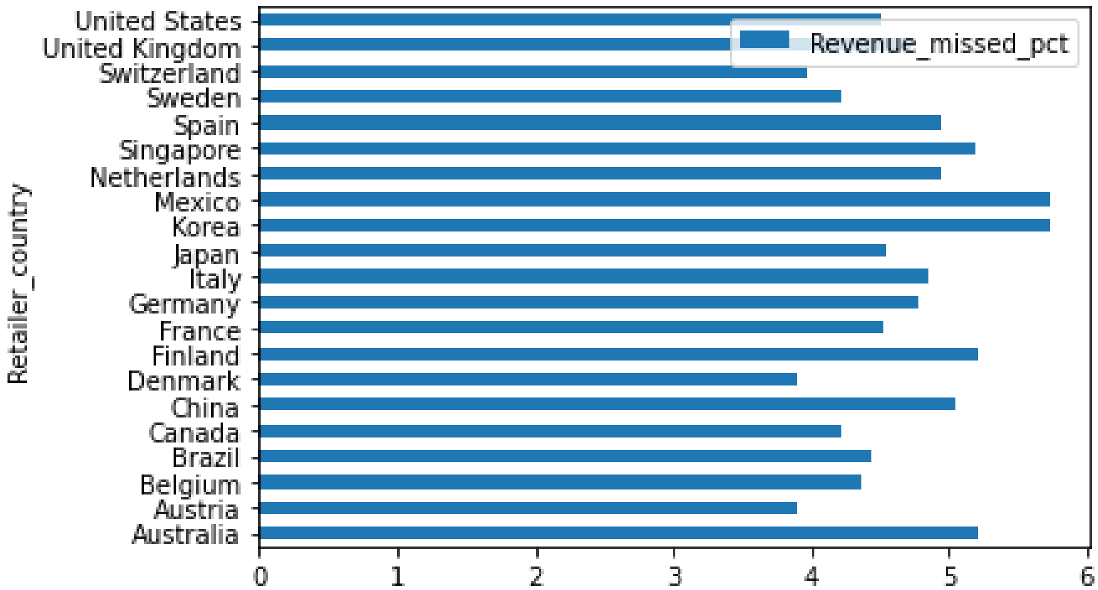
[65]:

```
                     Year      Revenue  Planned_revenue  Product_cost  \
Retailer_country
Australia         1857486  109299974.0        115311829      64254527
Austria           2240195  143709418.0        149549365      84147249
Belgium           1951807  113031619.0        118195712      66372307
Brazil            1504085  138276680.0        144697428      81387153
Canada            3441944  306159372.0        319653077     180416413
China             1842903  317244917.0        334082002     187417612
Denmark           1602886   62013037.0         64526127      36803620
Finland           1669339  188575323.0        198950287     111502578
France            3211876  286569519.0        300166336     168780985
Germany           3302882  262313078.0        275476051     154658565
Italy             2514362  186648117.0        196181889     109579819
Japan             3365197  357446635.0        374437913     210315931
Korea             2074717  200725320.0        212939596     118797376
Mexico            2165367  167187026.0        177346704      98849705
Netherlands       2435565  184321687.0        193917160     108704507
Singapore         2381155  197622402.0        208460742     116855753
Spain             2312759  165066493.0        173636549      96906541
Sweden            2093248   95411467.0         99623254      56122289
Switzerland       1623527  100731878.0        104890676      58855881
United Kingdom    2419810  277509546.0        291298415     163835204
United States     3879316  826912620.0        866001599     487377157

                  Quantity  Unit_cost  Unit_price  Gross_profit  \
Retailer_country
Australia          2000781    76472.0      139500    45045456.0
Austria            2742824    95182.0      174512    59562183.0
Belgium            2124791    75948.0      138325    46659336.0
Brazil             2591989    65678.0      120706    56889537.0
Canada             5722733   138890.0      252253   125742997.0
China              6110945    89506.0      166923   129827307.0
Denmark            1301136    58199.0      104951    25209421.0
Finland            3603492    66496.0      120998    77072755.0
France             5529613   135531.0      249099   117788568.0
Germany            5084611   136106.0      249054   107654509.0
Italy              3545695   104284.0      192412    77068308.0
Japan              6787127   147334.0      273881   147130706.0
Korea              3902092    90867.0      167671    81927974.0
Mexico             3175752    91625.0      168281    68337351.0
Netherlands        3448760   104459.0      192132    75617204.0
Singapore          3788595   100989.0      186702    80766665.0
```

Spain	3171715	100201.0	186009	68159942.0
Sweden	1681811	87679.0	159907	39289182.0
Switzerland	1822191	73297.0	137480	41876018.0
United Kingdom	5378361	101947.0	185999	113674387.0
United States	15722077	161142.0	294512	339535441.0

	Unit_sale_price	Revenue_missed_pct
Retailer_country		
Australia	132200.0	5.213563
Austria	164684.0	3.905030
Belgium	131558.0	4.369104
Brazil	114105.0	4.437362
Canada	239257.0	4.221359
China	157188.0	5.039806
Denmark	99282.0	3.894686
Finland	113849.0	5.214852
France	234453.0	4.529761
Germany	234939.0	4.778264
Italy	181504.0	4.859660
Japan	258109.0	4.537809
Korea	157702.0	5.736029
Mexico	158549.0	5.728710
Netherlands	180166.0	4.948233
Singapore	175607.0	5.199224
Spain	175661.0	4.935629
Sweden	151123.0	4.227715
Switzerland	130090.0	3.964888
United Kingdom	175439.0	4.733589
United States	278063.0	4.513731


Revenue
1e8
Planned_revenue
1e8
Product_cost
1e8
Gross_profit
1e8
Australia
Austria
Belgium
Brazil
Canada
China
Denmark
Finland
France
Germany
Italy
Japan
Korea
Mexico
Netherlands
Singapore
Spain
Sweden
Switzerland
United Kingdom
United States



- Which country had highest number of sales?
- Which product sold the most and. which product line had more revenue loss?
-

[297]:
```
df.head(3)
```

[297]:
```
   Year       Product_line  Product_type             Product  \
0  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
1  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag
2  2015  Camping Equipment  Cooking Gear  TrailChef Water Bag

  Order_method_type Retailer_country   Revenue  Planned_revenue  Product_cost  \
0         Telephone    United States  315044.0           437477        158372
1         Telephone           Canada   13445.0            14313          6299
2         Telephone            Japan  181120.0           235237         89413

   Quantity  Unit_cost  Unit_price  Gross_profit  Unit_sale_price
0     66385        3.0           7      156673.0              5.0
1      2172        3.0           7        7146.0              6.0
2     35696        3.0           7       91707.0              5.0
```

1.3 Feature selection

Feature selection huggely impacts performance of model. It : - reduces overfitting - improves accuracy - reduces training time

There are 3 methods of feature selection: - Univariate selection - Feature importance - Correlation matrix with Heatmap

[298]:
```python
# Adding more features

df['Missed_Revenue']=df['Planned_revenue']-df['Revenue']
df['revenue_to_cost_rate']=(df['Revenue']/df['Product_cost'])
```

[299]:
```python
df['revenue_to_cost_rate'].head(3)
```

[299]:
```
0    1.989266
1    2.134466
2    2.025656
Name: revenue_to_cost_rate, dtype: float64
```

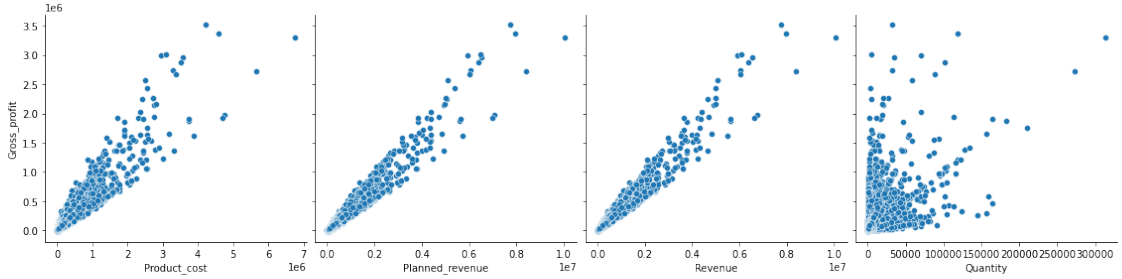
'revenue_to_cost_rate' of 1.989266 can be interpreted as - if you spend $1, you can yield a revenue of $1.989266. The more.. the better..

[300]:
```python
# Univariate selection

import numpy as np
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2


x = df.iloc[:,6:10]  #independent columns
y = df.iloc[:,-2]    #target column i.e Gross_profit

#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k='all')
fit = bestfeatures.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)

#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)

featureScores.columns = ['Feature_Name','Score']  #naming the dataframe columns
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
      Feature_Name         Score
1  Planned_revenue  1.131347e+10
0          Revenue  1.065375e+10
2     Product_cost  7.025373e+09
3         Quantity  2.488430e+08
```

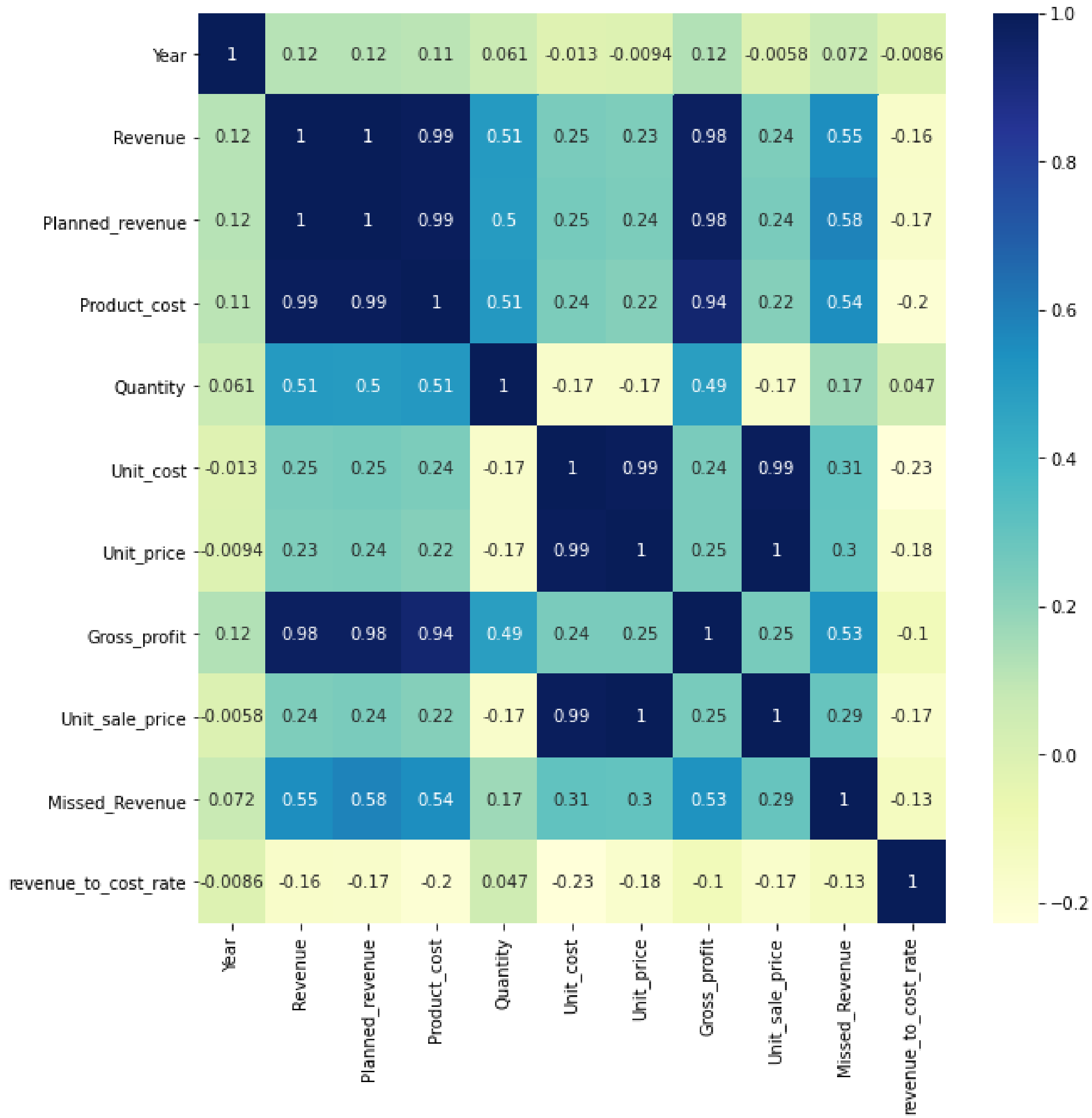
[301]:
```python
# Let's see how Gross Profit is related with other variables using scatter plot.
import seaborn as sns
sns.pairplot(df, x_vars=['Product_cost', 'Planned_revenue', 'Revenue',␣
 ↪'Quantity'],
             y_vars='Gross_profit', height=4, aspect=1, kind='scatter')
```

```
plt.show()
```



Top three features that are selected by the algorithm does not have any outliers. However Quantity is not a linearly increasing metrics with Gross_Profit We can ignore considering **Quantity** for the model, since we are taking the overall Revenue and Cost for determining the profits and not Unit_Cost or Unit_Sale

[302]:
```
# Let's see the correlation between different variables.

fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(df.corr(), cmap="YlGnBu", annot = True, ax=ax)
plt.show()
```



This further proves that Revenue, Planned Revenue and Product Cost are the best and most correlated metrics to grossprofit

1.4 Predictive models

- Partition the data into 80% training and 20% validation.
- Explain how using validation set helps to avoid overfitting/underfitting
- At least build two models using decision tree, logistic regression, linear regression, neural network, and knn.
- Assess, analyze, and compare the performance of your models

1.4.1 Model 1: Multi-Variable Regression

[303]:
```
X = df[['Revenue','Planned_revenue','Product_cost']]
y = df['Gross_profit']
```

[304]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8,␣
↪test_size = 0.2, random_state = 100)
```

[305]:
```
X_train.head()
```

[305]:

	Revenue	Planned_revenue	Product_cost
13586	71571.0	73497	31013
14657	186462.0	196276	132296
2130	33813.0	35219	23594
23741	0.0	2840	1318
2486	13409.0	13822	8931

[306]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(19794, 3)
(4949, 3)
(19794,)
(4949,)
```

[307]:
```
import statsmodels.api as sm

# Add a constant to get an intercept
X_train_sm = sm.add_constant(X_train)

# Fit the resgression line using 'OLS'
lr = sm.OLS(y_train, X_train_sm).fit()
```

[308]:
```
lr.params
```

[308]:
```
const              1.355763e-02
Revenue            9.999998e-01
Planned_revenue    6.130563e-08
Product_cost      -9.999999e-01
dtype: float64
```

[309]:
```
# Performing a summary operation lists out all the different parameters of the␣
↪regression line fitted
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Gross_profit   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 9.382e+14
Date:                Tue, 10 Aug 2021   Prob (F-statistic):               0.00
Time:                        23:10:53   Log-Likelihood:                -10872.
No. Observations:               19794   AIC:                         2.175e+04
Df Residuals:                   19790   BIC:                         2.178e+04
Df Model:                           3
Covariance Type:            nonrobust
================================================================================
===
                     coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
---
const              0.0136      0.003      4.010      0.000       0.007
0.020
Revenue            1.0000   1.81e-07   5.51e+06      0.000       1.000
1.000
Planned_revenue  6.131e-08    1.7e-07      0.360      0.719   -2.73e-07
3.95e-07
Product_cost      -1.0000   9.02e-08  -1.11e+07      0.000      -1.000
-1.000
==============================================================================
Omnibus:                     1271.077   Durbin-Watson:                   1.995
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5952.968
Skew:                           0.074   Prob(JB):                         0.00
Kurtosis:                       5.683   Cond. No.                     7.67e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 7.67e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpreting the model using OLS regression results:

- The coefficients and significance (p-values)
- R-squared
- F statistic and its significance

1. **Coefficient Interpretation:**

- Revenue The coefficient for Revenue is 1.0000, with a very low p value (0.00<0.05) The coefficient is positive, so it is positively correlated with Profit (Independent/ target variable) p value is statistically significant.

- Planned_revenue The coefficient for Planned_revenue is 6.131e-08, with a high p value (0.719 >0.05) The coefficient is positive, so it is positively correlated with Profit (Independent/ target variable) p value is not statistically significant.
- Product_cost The coefficient for Product_cost is -1.0000, with a very low p value (0.00<0.05) It makes sense, as if there is increase in cost, the profits will decrease. SO it is negatively correlated. p value is statistically significant.

2. **R - squared is 1:**

- Meaning that 100% of the variance in Profit is explained by the 3 factors mentioned above
- This is a perfect R-squared value, which implies that independent variables andd ddependent variables are having a strong correlation.

3. **F statistic has a very low p value (practically low)**

- Meaning that the model fit is statistically significant, and the explained variance isn't purely by chance.

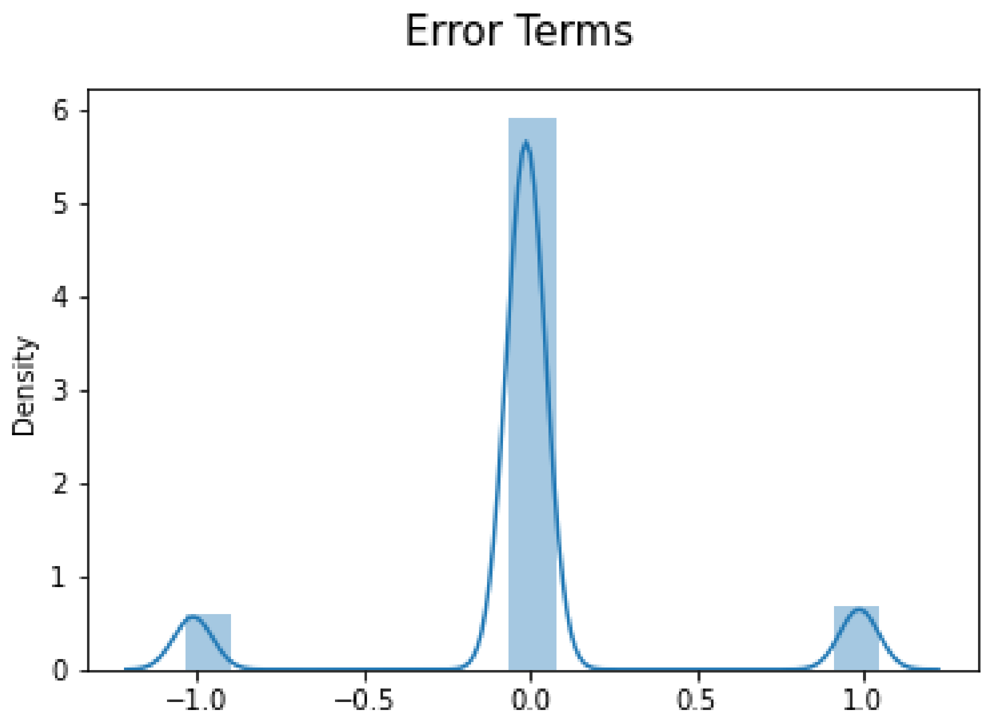
The fit is significant. Let us visualize how well the model fit the data. From our parameters, the linear regression equation is:

Gross_Profit= 0.0136 + 1.0000 * Revenue + 6.131e-08 * Planned_revenue -1.0000 *Product_cost

[310]:
```
y_train_pred = lr.predict(X_train_sm)
res = (y_train - y_train_pred)
```

[311]:
```
fig = plt.figure()
sns.distplot(res, bins = 15)
fig.suptitle('Error Terms', fontsize = 15)                  # Plot heading
plt.show()
```

```
/Users/Gaya/opt/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



The residuals are following the normally distributed with a mean 0. All good!

[312]:
```
X_test_sm = sm.add_constant(X_test)

# Predict the y values corresponding to X_test_sm
y_pred = lr.predict(X_test_sm)
```

[313]:
```
y_pred.head()
```

[313]:
```
11845     2358.013430
3353      3093.013437
8863     77959.011071
23037     4774.013349
4364      2893.013698
dtype: float64
```

[314]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

[315]:
```
#Returns the mean squared error; we'll take a square root
np.sqrt(mean_squared_error(y_test, y_pred))
r_squared = r2_score(y_test, y_pred)
r_squared
```

[315]: 0.999999999992928

1.4.2 Model Summary

We can say that as there is an increase in the Cost, we are seeing good profits. Which means that whatever strategy that is being used for sales is clearly working good

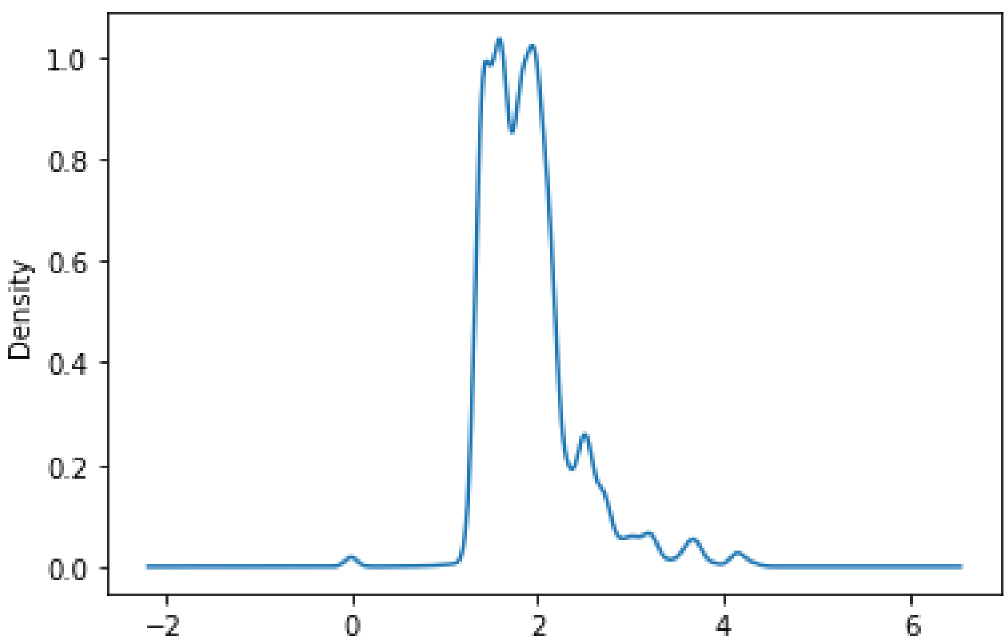
1.5 Model 2: Logistic Regressionn

[316]:
```
df.revenue_to_cost_rate.describe()
```

[316]:
```
count    24743.000000
mean         1.908928
std          0.497976
min          0.000000
25%          1.573688
50%          1.829897
75%          2.092941
max          4.358086
Name: revenue_to_cost_rate, dtype: float64
```

[317]:
```
df['revenue_to_cost_rate'].plot.kde()
```

[317]: <AxesSubplot:ylabel='Density'>



Let us build a boolean value called 'profit_satisfied' with 'true' or 'false' values

Our threshold should be on the mean value here (1.9)

[318]:
```
df['profit_satisfied'] = np.where(df['revenue_to_cost_rate']>1.9, 1, 0)
```

[319]:
```
df.groupby('profit_satisfied')['profit_satisfied'].count()
```

[319]:
```
profit_satisfied
0    13810
1    10933
Name: profit_satisfied, dtype: int64
```

Now the values are assigned, let us run a logistic regression to see if we can predict the sales satisfaction boolean We can create training and test dataset (80:20) as the previous model

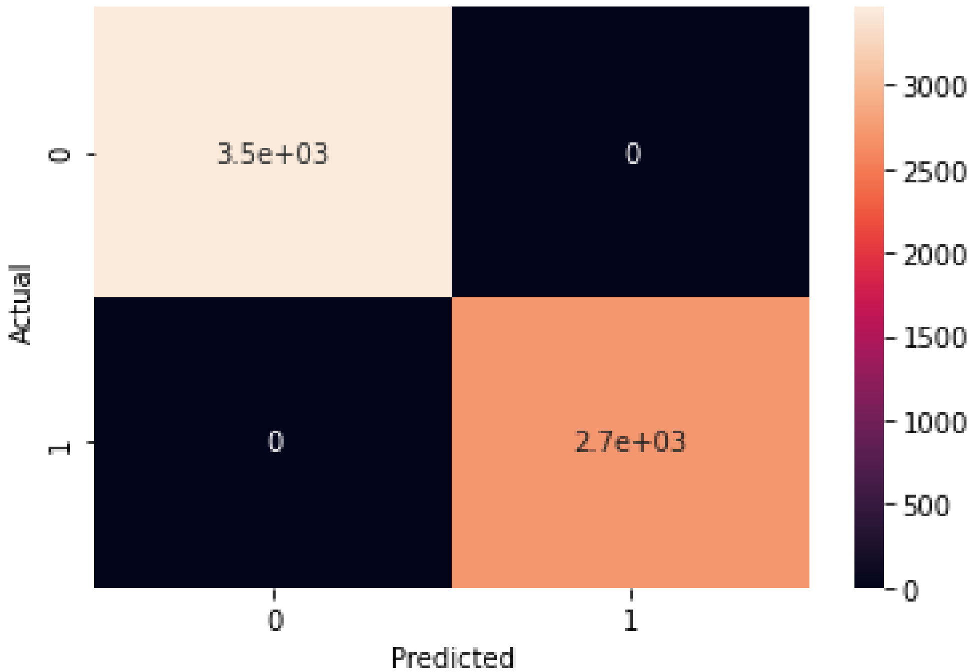
[324]:
```
X = df[['Revenue', 'Planned_revenue','Product_cost']]
y = df['profit_satisfied']
```

[325]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
↪25,random_state=0)
```

[336]:
```
from sklearn.linear_model import LogisticRegression
import seaborn as sn
from sklearn import metrics

logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)

y_pred = logmodel.predict(X_test)
```

[337]:
```
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'],␣
↪colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
```

[337]:
```
<AxesSubplot:xlabel='Predicted', ylabel='Actual'>
```



[338]:
```
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
plt.show()
```

```
Accuracy:  1.0
```

1.5.1 Model Summary

Accuracy of this logistic regression model is 100% meaning that `profit_satisfied` category completely depends on [‘Revenue’,‘Planned Revenue’,‘Product Cost’]

1.6 Future Steps & Conclusions

After detailed analysis we can say that the sales of outdoor products world wide is growing with the increase in the investments and some of the shortcomings of this analysis are

- Too many missing values
- More features on the data like customer related info, months, day, seasonality information
- Breakdown of cost and sales into multiple factors (operational cost, labour cost and so on)
- With more transaction-level customer metrics, we can build recommendation systems

If we can gather these details, we can expand our analysis to different dimensions