sales-analysis

September 2, 2022

1 Adventure-Works-EDA

1.1 Import libraries

[1]:
```
!pip install openpyxl plotly -q
```

[2]:
```
import jovian
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns; sns.set_theme()
import plotly.figure_factory as ff
from itertools import combinations
from collections import Counter
import datetime as dt
import warnings
warnings.filterwarnings('ignore')
```

1.2 Data wrangling

1.2.1 Data gathering

[3]:
```
Customers_data = pd.read_excel('https://github.com/doke93/
↪Data-Analysis-Project-Ineuron/files/8985052/Database.xlsx',
                              'Customers',
                              dtype={'CustomerKey':str},
                              parse_dates=['BirthDate','DateFirstPurchase']
                              )
```

[4]:
```
Product_data = pd.read_excel('https://github.com/doke93/
↪Data-Analysis-Project-Ineuron/files/8985052/Database.xlsx',
                              'Product',
                              dtype={'ProductKey':str},
                              parse_dates=['StartDate']
                              )
```

[5]:
```
Sales_data = pd.read_excel('https://github.com/doke93/
↪Data-Analysis-Project-Ineuron/files/8985052/Database.xlsx',
                           'Sales',
                           dtype={'ProductKey':str,
                                  'CustomerKey':str,
                                  'PromotionKey':str,
                                  'SalesTerritoryKey':str},
                           parse_dates=['OrderDate', 'ShipDate']
                           )
Sales_data['DateKey'] = Sales_data['OrderDate'].astype(str)
```

[6]:
```
Territory_data = pd.read_excel('https://github.com/doke93/
↪Data-Analysis-Project-Ineuron/files/8985052/Database.xlsx',
                           'Territory',
                           dtype={'SalesTerritoryKey':str}
                           )
```

1.2.2 Merging data

[7]:
```
temp_data = pd.merge(Sales_data, Product_data, on='ProductKey', how='inner')
df = pd.merge(temp_data, Customers_data, on='CustomerKey', how='inner')
df = pd.merge(df, Territory_data, on='SalesTerritoryKey', how='inner')
```

1.2.3 Assessing data

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 58189 entries, 0 to 58188
Data columns (total 46 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   ProductKey            58189 non-null  object
 1   OrderDate             58189 non-null  datetime64[ns]
 2   ShipDate              58189 non-null  datetime64[ns]
 3   CustomerKey           58189 non-null  object
 4   PromotionKey          58189 non-null  object
 5   SalesTerritoryKey     58189 non-null  object
 6   SalesOrderNumber      58189 non-null  object
 7   SalesOrderLineNumber  58189 non-null  int64
 8   OrderQuantity         58189 non-null  int64
 9   UnitPrice             58189 non-null  float64
 10  TotalProductCost      58189 non-null  float64
 11  SalesAmount           58189 non-null  float64
 12  TaxAmt                58189 non-null  float64
 13  DateKey               58189 non-null  object
 14  ProductName           58189 non-null  object
```

```
 15  SubCategory           58189 non-null  object
 16  Category              58189 non-null  object
 17  StandardCost          58189 non-null  float64
 18  Color                 30747 non-null  object
 19  ListPrice             58189 non-null  float64
 20  DaysToManufacture     58189 non-null  int64
 21  ProductLine           58189 non-null  object
 22  ModelName             58189 non-null  object
 23  Photo                 58189 non-null  object
 24  ProductDescription    58189 non-null  object
 25  StartDate             58189 non-null  datetime64[ns]
 26  FirstName             58189 non-null  object
 27  LastName              58189 non-null  object
 28  FullName              58189 non-null  object
 29  BirthDate             58189 non-null  datetime64[ns]
 30  MaritalStatus         58189 non-null  object
 31  Gender                58189 non-null  object
 32  YearlyIncome          58189 non-null  int64
 33  TotalChildren         58189 non-null  int64
 34  NumberChildrenAtHome  58189 non-null  int64
 35  Education             58189 non-null  object
 36  Occupation            58189 non-null  object
 37  HouseOwnerFlag        58189 non-null  int64
 38  NumberCarsOwned       58189 non-null  int64
 39  AddressLine1          58189 non-null  object
 40  DateFirstPurchase     58189 non-null  datetime64[ns]
 41  CommuteDistance       58189 non-null  object
 42  Region                58189 non-null  object
 43  Country               58189 non-null  object
 44  Group                 58189 non-null  object
 45  RegionImage           58189 non-null  object
dtypes: datetime64[ns](5), float64(6), int64(8), object(27)
memory usage: 20.9+ MB
```

[9]:
```python
# Check shape of the data after merging
print(f"Number of Rows: {df.shape[0]}")
print(f"Number of Columns: {df.shape[1]} \n")
```

```
Number of Rows: 58189
Number of Columns: 46
```

[10]:
```python
df.describe().transpose()
```

[10]:
```
                       count      mean       std       min  \
SalesOrderLineNumber  58189.0  1.887453  1.018829   1.0000
OrderQuantity         58189.0  1.569386  1.047532   1.0000
```

```
UnitPrice             58189.0    413.888218    833.052938      0.5725
TotalProductCost      58189.0    296.539185    560.171436      0.8565
SalesAmount           58189.0    503.666270    941.462817      2.2900
TaxAmt                58189.0     40.293303     75.317027      0.1832
StandardCost          58189.0    296.539185    560.171436      0.8565
ListPrice             58189.0    503.666270    941.462817      2.2900
DaysToManufacture     58189.0      1.045215      1.757395      0.0000
YearlyIncome          58189.0  59769.887779  33128.041818  10000.0000
TotalChildren         58189.0      1.838921      1.614467      0.0000
NumberChildrenAtHome  58189.0      1.073502      1.580055      0.0000
HouseOwnerFlag        58189.0      0.690560      0.462267      0.0000
NumberCarsOwned       58189.0      1.502466      1.155496      0.0000

                             25%         50%         75%          max
SalesOrderLineNumber      1.0000      2.0000      2.0000       8.0000
OrderQuantity             1.0000      1.0000      2.0000       4.0000
UnitPrice                 4.9900     24.4900    269.9950    3578.2700
TotalProductCost          3.3623     12.1924    343.6496    2171.2942
SalesAmount               8.9900     32.6000    539.9900    3578.2700
TaxAmt                    0.7192      2.6080     43.1992     286.2616
StandardCost              3.3623     12.1924    343.6496    2171.2942
ListPrice                 8.9900     32.6000    539.9900    3578.2700
DaysToManufacture         0.0000      0.0000      4.0000       4.0000
YearlyIncome          30000.0000  60000.0000  80000.0000  170000.0000
TotalChildren             0.0000      2.0000      3.0000       5.0000
NumberChildrenAtHome      0.0000      0.0000      2.0000       5.0000
HouseOwnerFlag            0.0000      1.0000      1.0000       1.0000
NumberCarsOwned           1.0000      2.0000      2.0000       4.0000
```

[11]:
```
# Check for duplicate data
df.duplicated().sum()
```

[11]:
```
0
```

1.2.4 Handling missing data

[12]:
```
def missing_pct(df):
    # Calculate missing value and their percentage for each column
    missing_count_percent = df.isnull().sum() * 100 / df.shape[0]
    df_missing_count_percent = pd.DataFrame(missing_count_percent).round(2)
    df_missing_count_percent = df_missing_count_percent.reset_index().rename(
                    columns={
                            'index':'Column',
                            0:'Missing_Percentage (%)'
                    }
                )
    df_missing_value = df.isnull().sum()
```

```
    df_missing_value = df_missing_value.reset_index().rename(
                    columns={
                            'index':'Column',
                            0:'Missing_value_count'
                    }
                )
    # Sort the data frame
    #df_missing = df_missing.sort_values('Missing_Percentage (%)',␣
↪ascending=False)
    Final = df_missing_value.merge(df_missing_count_percent, how = 'inner',␣
↪left_on = 'Column', right_on = 'Column')
    Final = Final.sort_values(by = 'Missing_Percentage (%)',ascending = False)
    return Final
```

[13]:
```
# Applying the custom function
missing_pct(df)
```

[13]:

	Column	Missing_value_count	Missing_Percentage (%)
18	Color	27442	47.16
0	ProductKey	0	0.00
34	NumberChildrenAtHome	0	0.00
26	FirstName	0	0.00
27	LastName	0	0.00
28	FullName	0	0.00
29	BirthDate	0	0.00
30	MaritalStatus	0	0.00
31	Gender	0	0.00
32	YearlyIncome	0	0.00
33	TotalChildren	0	0.00
35	Education	0	0.00
24	ProductDescription	0	0.00
36	Occupation	0	0.00
37	HouseOwnerFlag	0	0.00
38	NumberCarsOwned	0	0.00
39	AddressLine1	0	0.00
40	DateFirstPurchase	0	0.00
41	CommuteDistance	0	0.00
42	Region	0	0.00
43	Country	0	0.00
44	Group	0	0.00
25	StartDate	0	0.00
23	Photo	0	0.00
1	OrderDate	0	0.00
22	ModelName	0	0.00
2	ShipDate	0	0.00
3	CustomerKey	0	0.00
4	PromotionKey	0	0.00

```
5       SalesTerritoryKey                  0                    0.00
6       SalesOrderNumber                   0                    0.00
7    SalesOrderLineNumber                  0                    0.00
8           OrderQuantity                  0                    0.00
9               UnitPrice                  0                    0.00
10       TotalProductCost                  0                    0.00
11            SalesAmount                  0                    0.00
12                 TaxAmt                  0                    0.00
13                DateKey                  0                    0.00
14            ProductName                  0                    0.00
15            SubCategory                  0                    0.00
16               Category                  0                    0.00
17           StandardCost                  0                    0.00
19              ListPrice                  0                    0.00
20      DaysToManufacture                  0                    0.00
21            ProductLine                  0                    0.00
45            RegionImage                  0                    0.00
```

[14]:
```
#  Drop columns with nan values
df= df.dropna(axis=1)
```

1.2.5 Adding columns

[15]:
```
# Extracting Year from OrderDate
df['sale_year'] = df['OrderDate'].dt.year

# Extracting Month from OrderDate
df['sale_month'] = df['OrderDate'].dt.month

# Extracting day from OrderDate
df['sale_day'] = df['OrderDate'].dt.day

# Extracting dayofweek from OrderDate
df['sale_week'] = df['OrderDate'].dt.dayofweek

# Extracting day_name from OrderDate
df['sale_day_name'] = df['OrderDate'].dt.day_name()

# Extracting Month Year from OrderDate
df['year_month'] = df['OrderDate'].apply(lambda x:x.strftime('%Y-%m'))

# Calculate Total Invoice Amount
df['total_Invoice_amount'] = df['SalesAmount'] + df['TaxAmt']

# Considering only salesamount and total_sales_amount to calculate profit
df['profit'] = (df['UnitPrice']*df['OrderQuantity']) - df['TotalProductCost']
```

```
# Removing extra character from the string
df['ProductName'] = df['ProductName'].str.replace(',','-')

# Calculate Age
df['Age'] = df['OrderDate'].dt.year - df['BirthDate'].dt.year
```

1.3 Exploring data

1.3.1 Basic Overview

List of product's category

[16]:
```
df['Category'].unique().tolist()
```

[16]:
```
['Bikes', 'Accessories', 'Clothing']
```

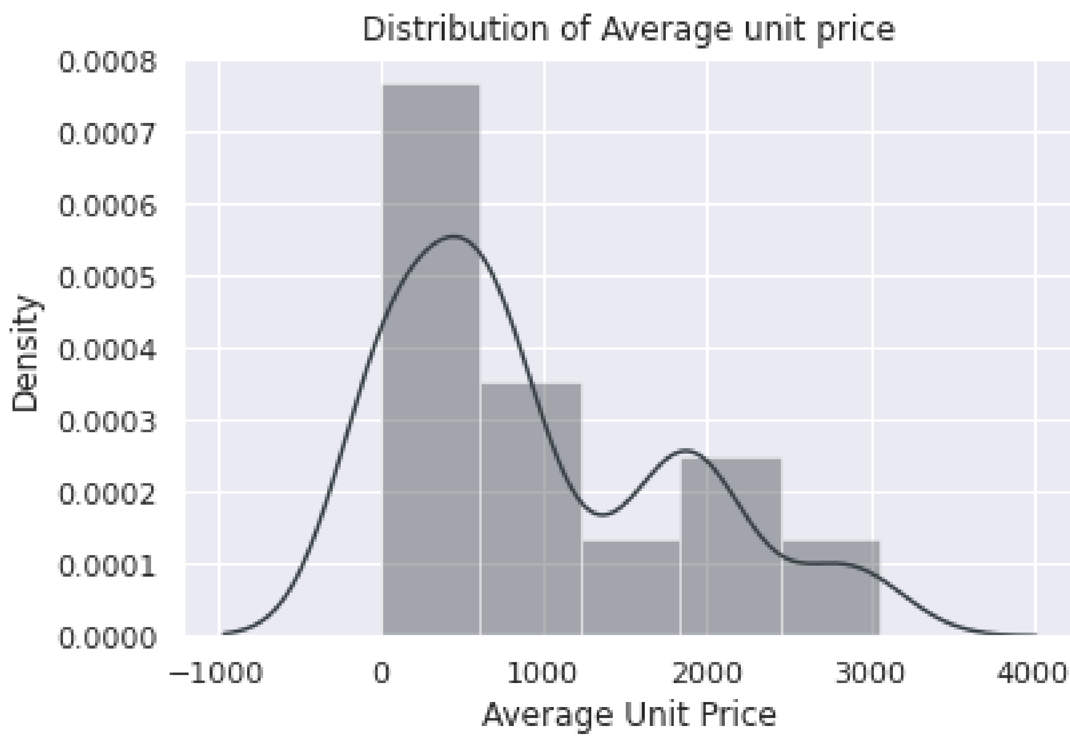
List of product's subcategory

[17]:
```
df['SubCategory'].unique().tolist()
```

[17]:
```
['Road Bikes',
 'Mountain Bikes',
 'Bottles and Cages',
 'Gloves',
 'Tires and Tubes',
 'Helmets',
 'Touring Bikes',
 'Jerseys',
 'Cleaners',
 'Caps',
 'Hydration Packs',
 'Socks',
 'Fenders',
 'Vests',
 'Bike Racks',
 'Bike Stands',
 'Shorts']
```

Analysing UnitPrice

[18]:
```
Avg_unit_price = df.groupby(['ProductKey'])['UnitPrice'].mean()
ax = sns.distplot(Avg_unit_price, kde=True, hist=True, color='#374045')
ax.set(title='Distribution of Average unit price',
       xlabel='Average Unit Price');
```



- Maximum of the product unit price is below $1000

Sales order number distribution

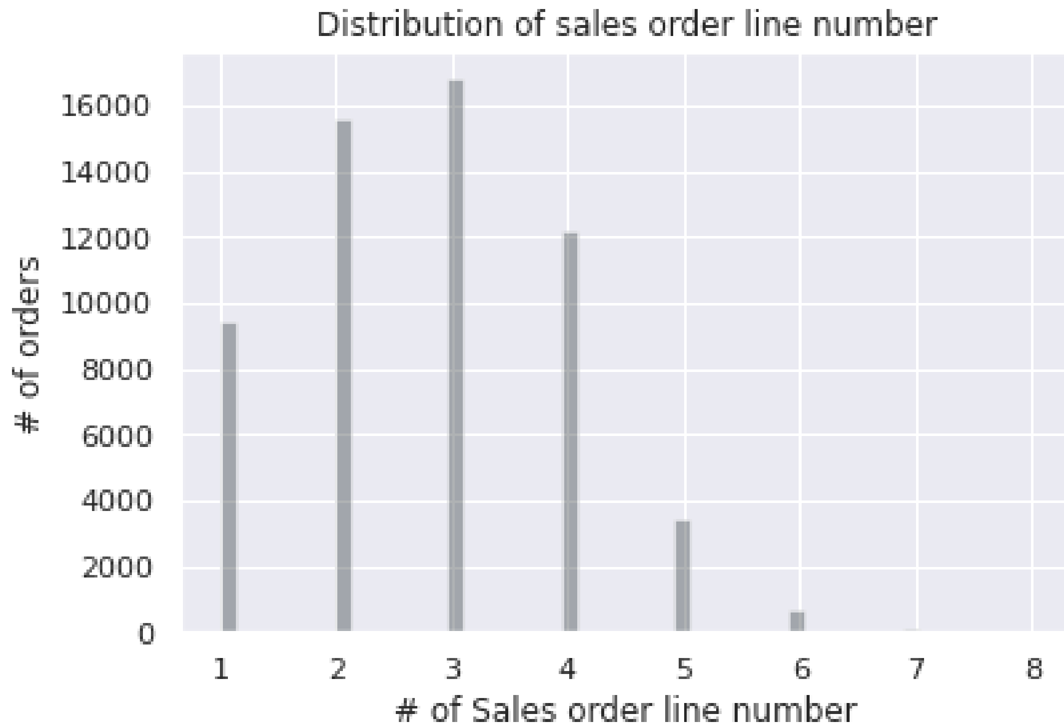
[19]:
```
n_orders = df.groupby(['CustomerKey'])['SalesOrderNumber'].nunique()
multi_orders_perc = np.sum(n_orders > 1)/df['CustomerKey'].nunique()
print(f"{100*multi_orders_perc:.2f}% of customers ordered more than once.")
```

```
36.97% of customers ordered more than once.
```

[20]:
```
ax = sns.distplot(n_orders, kde=False, color='#374045')
ax.set(title='Distribution of number of orders per customer',
       xlabel='# of orders',
       ylabel='# of customers');
```

Sales order line number distribution

[21]:
```
n_salesordernumber = df.groupby(['SalesOrderNumber'])['SalesOrderLineNumber'].
 ↪transform('max')
ax = sns.distplot(n_salesordernumber, kde=False, color='#374045')
ax.set(title='Distribution of sales order line number',
       xlabel='# of Sales order line number',
       ylabel='# of orders');
```



- Most of the time **three to two** products are ordered in a single order

Sales Order Quantity distribution

[22]:
```
n_order_quantity = df.groupby(['SalesOrderNumber'])['OrderQuantity'].sum()
ax = sns.distplot(n_order_quantity, kde=True, hist=True,color='#374045')
ax.set(title='Distribution of order_quantity',
       xlabel='# of order_quantity',
       );
```

- maximum quantity ordered for a product is below 5

Age Distribution

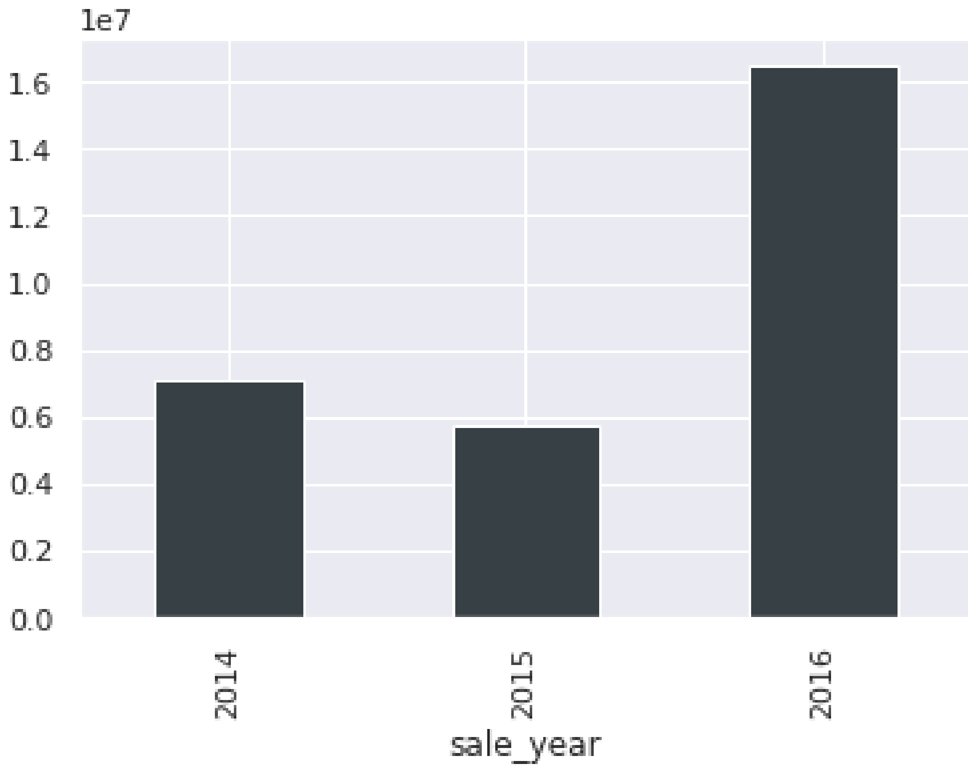
[23]:
```
bins = [18, 30, 40, 50, 60, 70, 120]
labels = ['18-29', '30-39', '40-49', '50-59', '60-69', '70+']
df['agerange'] = pd.cut(df.Age, bins, labels = labels,include_lowest = True)

age_distribution = df['agerange'].value_counts().to_frame().reset_index()

age_distribution.columns = ['Age Range','Population count']

fig = px.bar(age_distribution, x='Age Range', y='Population count',␣
 ↪color_discrete_sequence=['#374045'])
fig.update_layout(
    autosize=True,
    width=500,
    height=500,
    font=dict(size=10))
fig.show()
```

- A sizable portion of the clientele is made up of people between the ages of **40 and 59**.

1.3.2 Sales

Year wise sales

[24]:
```
df.groupby('sale_year')['SalesAmount'].sum().plot(kind='bar', color='#374045');
```



- The year 2016 saw an exponential surge in sales

Top 5 Selling Product

[25]:
```
top_selling_product = df.groupby(['Category', 'SubCategory',␣
↪'ProductName'])['OrderQuantity'].sum().nlargest(5).to_frame()
top_selling_product
```

[25]:

Category	SubCategory	ProductName	OrderQuantity
Accessories	Bottles and Cages	Water Bottle - 30 oz.	6370
	Tires and Tubes	Patch Kit/8 Patches	4705
		Mountain Tire Tube	4551
		Road Tire Tube	3544
	Helmets	Sport-100 Helmet- Red	3398

[26]:
```
top_selling_product.reset_index(inplace=True)
fig = px.bar(top_selling_product, x='ProductName',␣
↪y='OrderQuantity',color_discrete_sequence=['#374045'])
fig.update_layout(
```

```
    autosize=True,
    width=500,
    height=300,
    margin=dict(
        l=25,
        r=25,
        b=10,
        t=10,
    ),
    font=dict(size=8))
fig.show()
```

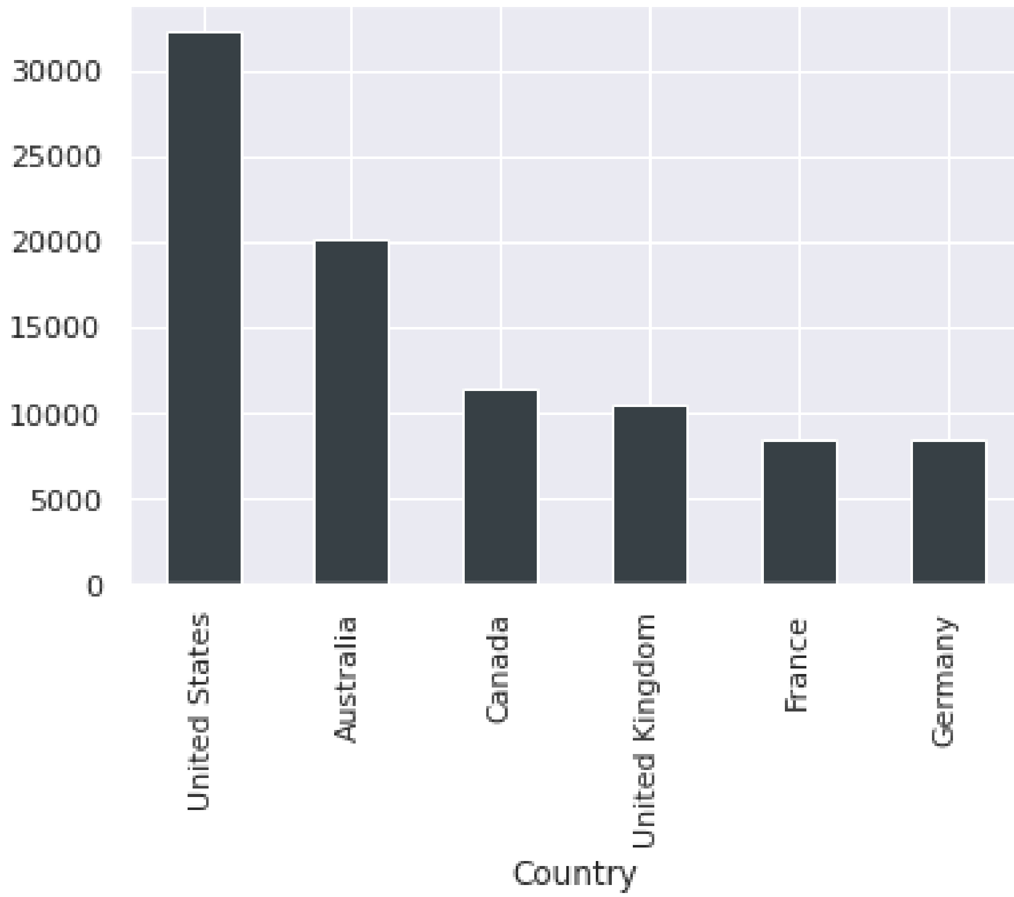
Quantity ordered based on category and subcategory from 2014 to 2016

[27]:
```
cat_subcat_qty = df.groupby(['sale_year','Category',␣
 ↪'SubCategory'])['OrderQuantity'].sum().to_frame()
cat_subcat_qty = cat_subcat_qty.sort_values(['sale_year', 'Category'],␣
 ↪ascending=True)
cat_subcat_qty.style.bar(subset=['OrderQuantity'], color='#D9B300')
```

[27]: <pandas.io.formats.style.Styler at 0x7f6487132dc0>

Country wise quantity ordered

[28]:
```
country_qty_sales = df.groupby('Country')['OrderQuantity'].sum().
 ↪sort_values(ascending=False)
country_qty_sales.plot(kind='bar', color='#374045');
```



- High quantity of products is ordered from **Australia and United States**

1.3.3 Profit

Overall profit based on order year, category and subcategory

[29]:
```
cat_subcat_profit = df.groupby(['sale_year','Category',␣
↪'SubCategory'])['profit'].sum().to_frame()

#Sorting the results
cat_subcat_profit = cat_subcat_profit.sort_values(['sale_year', 'Category'],␣
↪ascending=True)
cat_subcat_profit.style.bar(subset=['profit'], color='#D9B300')
```

[29]:
```
<pandas.io.formats.style.Styler at 0x7f6487c4e190>
```

- Major Profit is contributed by the Bike Category

Low profit contributing product

[30]:
```
df.groupby(['Category', 'SubCategory','ProductName'])['profit'].sum().
↪nsmallest(10).to_frame()
```

[30]:

Category	SubCategory	ProductName	profit
Clothing	Socks	Racing Socks- L	1474.4574
		Racing Socks- M	1581.3837
Accessories	Cleaners	Bike Wash - Dissolver	4299.8688
	Tires and Tubes	Patch Kit/8 Patches	4314.8350
Clothing	Caps	AWC Logo Cap	4331.8315
Accessories	Tires and Tubes	Touring Tire Tube	4363.8089
Clothing	Jerseys	Long-Sleeve Logo Jersey- XL	4495.6007
		Short-Sleeve Classic Jersey- L	4544.8782
		Long-Sleeve Logo Jersey- S	4610.5777
		Short-Sleeve Classic Jersey- M	4793.2322

Profitability by country

[31]:
```
country_sales = pd.DataFrame(df.groupby('Country').sum()[['SalesAmount',␣
↪'profit']])
country_sales.reset_index(inplace=True)

fig = px.bar(country_sales, x='Country', y='profit',text_auto='.2s',
             color='SalesAmount',
             height=400)
fig.show()
```

- High volume of profit is earned from **Australia and United States**

1.3.4 Question and Answers

How efficient are the logistics?

[32]:
```
# Adding manufacturing days to the order received date
df['OrderreadyDate'] = df['OrderDate'] + pd.
↪to_timedelta(df['DaysToManufacture'], unit='D')

# Check the delay between order shipment date and order ready to supply
df['shipping_efficiency'] = (df['ShipDate'] - df['OrderreadyDate']).dt.days

fig = px.histogram(df, x="shipping_efficiency",␣
↪color_discrete_sequence=['#374045'])
fig.update_layout(
    autosize=True,
    width=300,
    height=300,
    margin=dict(
        l=25,
```

```
        r=25,
        b=10,
        t=10,
    ),
    font=dict(size=10))
fig.show()
```

- The average order has a gap of 7 days between the day the order is ready for export from the factory and the date it was shipped
- Management must work to reduce this gap toward 3 days.

What was the best month for sales? How much was earned that month ?

[33]:
```
month_sales = df.groupby('sale_month').sum()[['SalesAmount', 'profit']]
month_sales.reset_index(inplace=True)
fig = px.bar(month_sales, x='sale_month', y='SalesAmount',text_auto='.2s',
             hover_data=['sale_month', 'SalesAmount'], color='profit',
             height=400)
fig.show()
```

- There are large profit transactions in the months of **June, November, and December**

What time should we display advertisement to maximize likelihood of customerls buying product?

[34]:
```
sales_by_week = df.groupby(['sale_day_name']).count()['SalesAmount'].
 ↪reset_index().sort_values('SalesAmount', ascending=False)

fig = px.line(sales_by_week, x='sale_day_name', y='SalesAmount', title='Sales␣
 ↪Frequency by week')
fig.update_layout(
    autosize=True,
    width=300,
    height=300,
    margin=dict(
        l=25,
        r=25,
        b=10,
        t=10,
    ),
    font=dict(size=7))
fig.show()
```

- High sales orders are seen on **Wednesday and Saturday**, therefore we can promote our product during these workweek

Which products are most often sold together?

[35]:

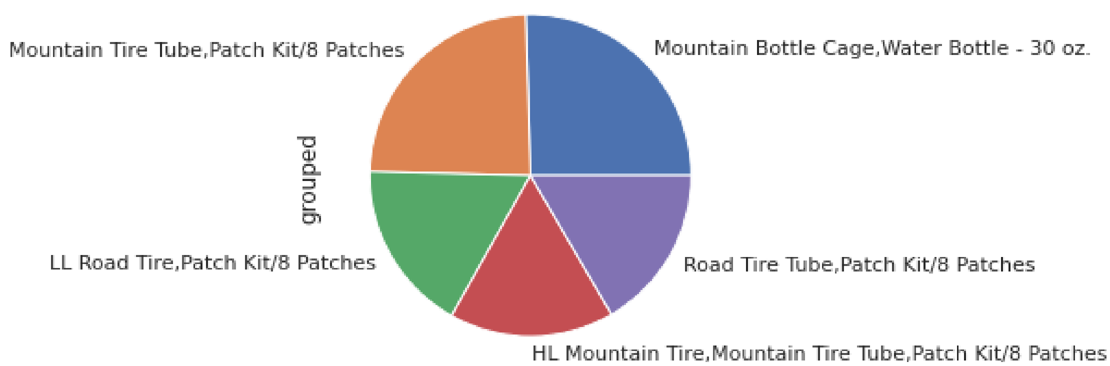
```
# By setting keep on False, all duplicates are True since we only want repeated␣
↪order number
dup_order = df[df['SalesOrderNumber'].duplicated(keep=False)]
```

[36]:

```
# Group the data based on sales order number and product name because the␣
↪products
# that bought together will have share same order number
dup_order['grouped'] = df.groupby('SalesOrderNumber')['ProductName'].
↪transform(lambda x: ','.join(x))
dup_order = dup_order[['SalesOrderNumber', 'grouped']].drop_duplicates()
```

[37]:

```
count = dup_order['grouped'].value_counts()[0:5].plot.pie()
```



- From the above pie diagram we can draw a conclusion that these products are mostly Purchased together

[38]:

```
count = Counter()

for row in dup_order['grouped']:
    row_list  = row.split(',')
    count.update(Counter(combinations(row_list, 2)))

for key, value in count.most_common(10):
    print(key, value)
```

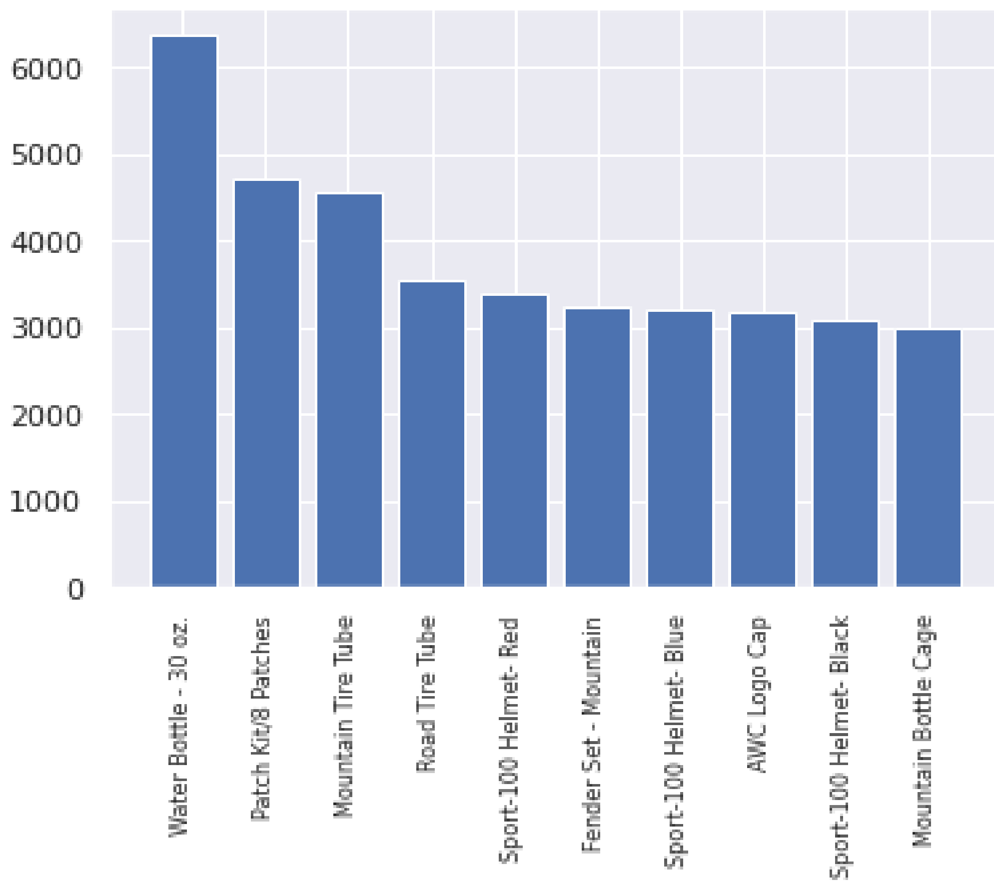
```
('Mountain Bottle Cage', 'Water Bottle - 30 oz.') 1623
('Road Bottle Cage', 'Water Bottle - 30 oz.') 1513
('HL Mountain Tire', 'Mountain Tire Tube') 915
('Touring Tire', 'Touring Tire Tube') 758
('Mountain Tire Tube', 'Patch Kit/8 Patches') 737
('Mountain Tire Tube', 'ML Mountain Tire') 727
('Water Bottle - 30 oz.', 'AWC Logo Cap') 599
('Road Tire Tube', 'ML Road Tire') 580
```

```
('Road Tire Tube', 'Patch Kit/8 Patches') 556
('HL Road Tire', 'Road Tire Tube') 552
```

- The above product can be sold in a bundle or a combined package for discount

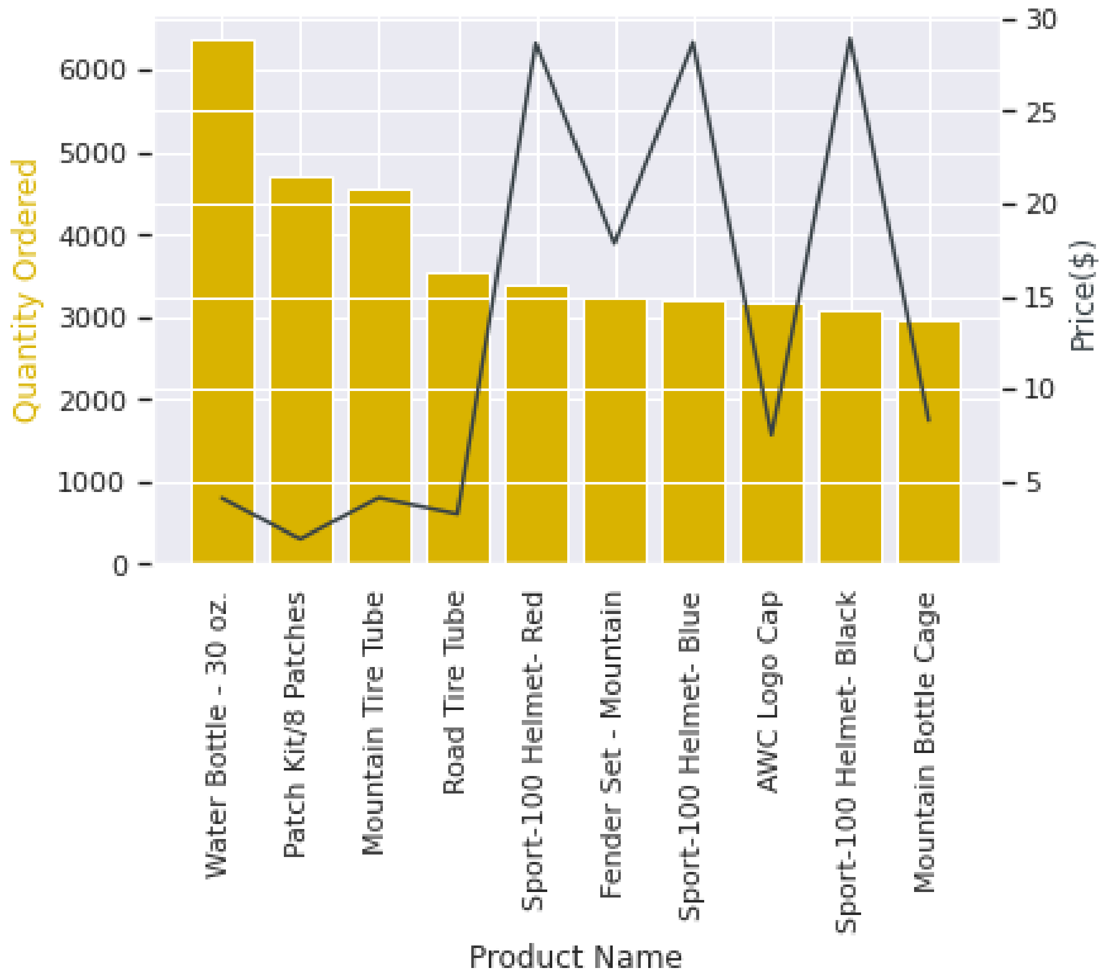
Which product sold the most? why do you think it sold the most?

[39]:
```
product_group = df.groupby('ProductName')
quantity_ordered = product_group['OrderQuantity'].sum().
↪sort_values(ascending=False)[:10]
products = quantity_ordered.index.tolist()

plt.bar(products, quantity_ordered, )
plt.xticks(products, rotation='vertical', size=8)
plt.show()
```



[40]:
```
prices = df.groupby('ProductName').mean()['UnitPrice']
prices = prices[products]
```

[41]:
```
fig, ax1 = plt.subplots()

ax2 = ax1.twinx()
ax1.bar(products, quantity_ordered, color='#D9B300')
ax2.plot(products, prices, '#374045')

ax1.set_xlabel('Product Name')
ax1.set_ylabel('Quantity Ordered', color='#D9B300')
ax2.set_ylabel('Price($)', color='#374045')
ax1.set_xticklabels(products, rotation='vertical')

plt.show();
```



[42]:
```
prices.corr(quantity_ordered)
```

[42]: -0.5333019792658484

- There is a **high negative correlation** between **Price and number of Quantity ordered**
- we can conclude that **low price product has high demand**

Compare most ordered product by gender

[43]:
```
male = df[df["Gender"]=="M"]
female = df[df["Gender"]=="F"]
```

[44]:
```
male_ord_qty = male.groupby(['ProductName'],as_index=False)['OrderQuantity'].
 ↪sum().nlargest(5,'OrderQuantity').sort_values('ProductName')
male_ord_qty.columns=['ProductName','Order_Qty_Male']

female_ord_qty = female.
 ↪groupby(['ProductName'],as_index=False)['OrderQuantity'].sum().
 ↪nlargest(5,'OrderQuantity').sort_values('ProductName')
female_ord_qty.columns=['ProductName','Order_Qty_Female']

df_merge = pd.merge(male_ord_qty, female_ord_qty, on='ProductName')
```

[45]:
```
fig = px.line(df_merge, x="ProductName",␣
 ↪y=["Order_Qty_Male","Order_Qty_Female"])
fig.update_layout(
    autosize=True,
    width=800,
    height=400)
fig.show()
```

Does Gender and home ownership matter in order purchasing

[46]:
```
fig = px.imshow(df.groupby(["Gender", "HouseOwnerFlag"])["SalesAmount"].mean().
 ↪unstack(),
                labels=dict(color="Average Purchase"))
fig.show()
```

- It's interesting to note that the average amount spent by men without permanent addresses is low, whilst the average amount spent by women without permanent addresses is higher.

Number of childer and Purchase correlation

[47]:
```
df_1 = df.groupby(["NumberChildrenAtHome"])["SalesAmount"].mean().to_frame()
df_1.reset_index(inplace=True)
fig = px.bar(df_1, x='NumberChildrenAtHome',␣
 ↪y='SalesAmount',color_discrete_sequence=['#374045'])
fig.update_layout(
    autosize=False,
    width=300,
    height=300,
    margin=dict(
        l=25,
```

```
        r=25,
        b=10,
        t=10,
    ))
fig.show()
```

Education, Occupation and Purchase correlation

[48]:
```
fig = px.imshow(df.groupby(["Education", "Occupation"])["SalesAmount"].mean().
↪unstack(),
                labels=dict(color="Average Purchase"))
fig.show()
```

Maritial Status single and above 50 age purchase

[49]:
```
df_2 = df[(df['MaritalStatus']=='S')&(df['Age']>50)]
```

[50]:
```
df_2 = df_2.groupby('agerange')['SalesAmount'].mean().to_frame().dropna()
df_2.reset_index(inplace=True)
fig = px.bar(df_2, x='agerange', y='SalesAmount',␣
↪color_discrete_sequence=['#374045'])
fig.update_layout(
    autosize=False,
    width=300,
    height=300,
    margin=dict(
        l=25,
        r=25,
        b=10,
        t=10,
    ))
fig.show()
```

Which age group has produced the most revenue?

[51]:
```
df_3 = df.groupby('agerange')['SalesAmount'].mean().to_frame().dropna()
df_3.reset_index(inplace=True)
fig = px.bar(df_3, x='agerange', y='SalesAmount',␣
↪color_discrete_sequence=['#374045'])
fig.update_layout(
    autosize=False,
    width=300,
    height=300,
    margin=dict(
        l=25,
        r=25,
        b=10,
```

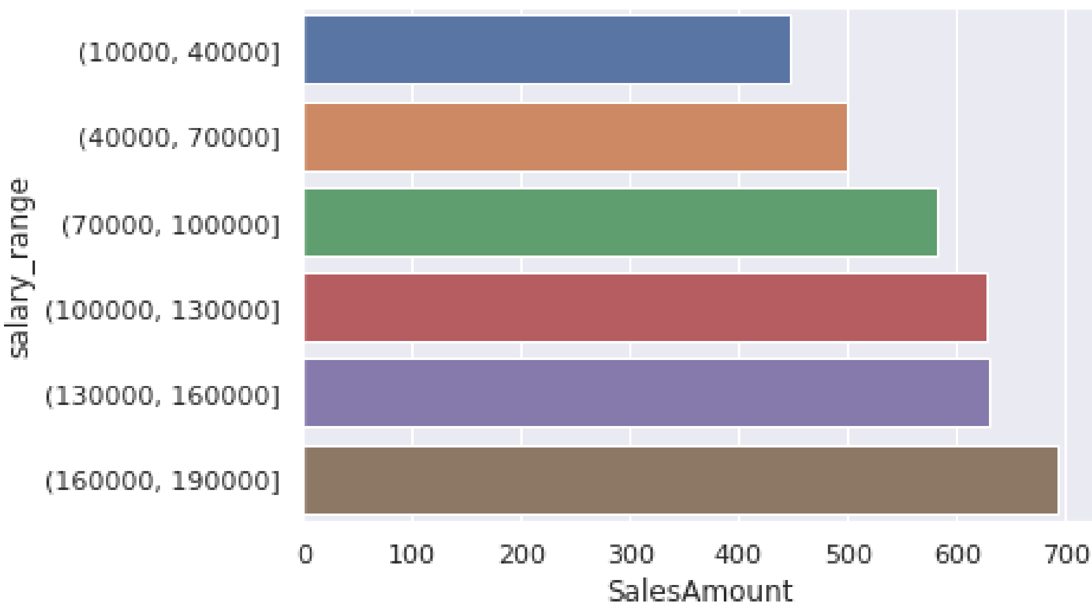
```
        t=10,
    ))
fig.show()
```

Yearly income range and purchase correlation

[52]:
```
def create_bins(lower_bound, width, quantity):
    """ create_bins returns an equal-width (distance) partitioning.
        It returns an ascending list of tuples, representing the intervals.
        A tuple bins[i], i.e. (bins[i][0], bins[i][1])  with i > 0
        and i < quantity, satisfies the following conditions:
            (1) bins[i][0] + width == bins[i][1]
            (2) bins[i-1][0] + width == bins[i][0] and
                bins[i-1][1] + width == bins[i][1]
    """


    bins = []
    for low in range(lower_bound,
                     lower_bound + quantity*width + 1, width):
        bins.append((low, low+width))
    return bins
```

[53]:
```
bins = create_bins(lower_bound=10000,
                   width=30000,
                   quantity=5)
bins2 = pd.IntervalIndex.from_tuples(bins)
df['salary_range'] = pd.cut(df['YearlyIncome'], bins2)
```

[54]:
```
df_4 = df.groupby('salary_range')['SalesAmount'].mean().to_frame()
df_4.reset_index(inplace=True)
sns.barplot(x="SalesAmount", y="salary_range", data=df_4);
```



- High salary range leads to increase in purchase

Paritial high school vs bachlors income mean and most ordered product

[55]:
```
df_6 = df[(df['Education']=='Partial High␣
↪School')|(df['Education']=='Bachelors')].
↪groupby('Education')['YearlyIncome'].mean().to_frame()
```

[56]:
```
df_6.reset_index(inplace=True)
fig = px.bar(df_6, x='Education', y='YearlyIncome')
fig.update_layout(
    autosize=False,
    width=300,
    height=300,
    margin=dict(
        l=25,
        r=25,
        b=10,
        t=10,
    ))
fig.show()
```

[57]:
```
df_7 = df[(df['Education']=='Partial High␣
↪School')|(df['Education']=='Bachelors')]
df_7 = df_7.groupby(['Education','ProductName'])['OrderQuantity'].mean().
↪to_frame().sort_values('OrderQuantity', ascending=False)[:10]
df_7.reset_index(inplace=True)
```

```
fig = px.bar(df_7, x="Education",
             y="OrderQuantity", color="ProductName",
             title="Paritial high school vs bachlors expense analysis",
             barmode="group")
fig.show()
```

- Customers with a **high school diploma and modest annual income buy more products** than people with bachelor's degrees

1.3.5 Customer Segmentation

[58]:
```
# RFM stands for recency, frequency, monetary value.
#  In business analytics, we often use this concept to divide
#  customers into different segments, like high-value customers,
#  medium value customers or low-value customers, and similarly many others.
```

[59]:
```
# Recency: How recently has the customer made a transaction with us
# Frequency: How frequent is the customer in ordering/buying some product from␣
 ↪us
# Monetary: How much does the customer spend on purchasing products from us
```

[60]:
```
#  calculating recency for customers who had made a purchase with a company

df_recency = df.groupby(by='FullName',
                        as_index=False)['OrderDate'].max()
df_recency.columns = ['CustomerName', 'LastPurchaseDate']
recent_date = df_recency['LastPurchaseDate'].max()
df_recency['Recency'] = df_recency['LastPurchaseDate'].apply(
    lambda x: (recent_date - x).days)
```

[61]:
```
#  calculating the frequency of frequent transactions of the
#  customer in ordering/buying some product from the company.


frequency_df = df.drop_duplicates().groupby(
    by=['FullName'], as_index=False)['OrderDate'].count()
frequency_df.columns = ['CustomerName', 'Frequency']
# frequency_df.head()
```

[62]:
```
monetary_df = df.groupby(by='FullName', as_index=False)['SalesAmount'].sum()
monetary_df.columns = ['CustomerName', 'Monetary']
# monetary_df.head()
```

[63]:
```
# merging dataset
rf_df = df_recency.merge(frequency_df, on='CustomerName')
rfm_df = rf_df.merge(monetary_df, on='CustomerName').drop(
    columns='LastPurchaseDate')
```

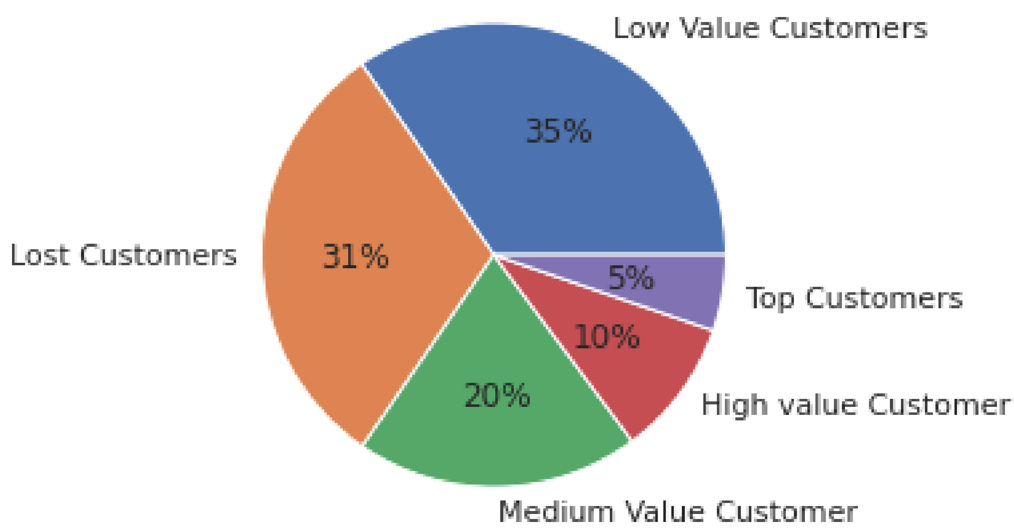
```
# rfm_df.head()
```

```
[64]: rfm_df['R_rank'] = rfm_df['Recency'].rank(ascending=False)
rfm_df['F_rank'] = rfm_df['Frequency'].rank(ascending=True)
rfm_df['M_rank'] = rfm_df['Monetary'].rank(ascending=True)

# normalizing the rank of the customers
rfm_df['R_rank_norm'] = (rfm_df['R_rank']/rfm_df['R_rank'].max())*100
rfm_df['F_rank_norm'] = (rfm_df['F_rank']/rfm_df['F_rank'].max())*100
rfm_df['M_rank_norm'] = (rfm_df['F_rank']/rfm_df['M_rank'].max())*100

rfm_df.drop(columns=['R_rank', 'F_rank', 'M_rank'], inplace=True)

# rfm_df.head()
```

```
[65]: rfm_df['RFM_Score'] = 0.15*rfm_df['R_rank_norm']+0.28 * \
    rfm_df['F_rank_norm']+0.57*rfm_df['M_rank_norm']
rfm_df['RFM_Score'] *= 0.05
rfm_df = rfm_df.round(2)
# rfm_df[['CustomerName', 'RFM_Score']].head(7)
```

```
[66]: rfm_df["Customer_segment"] = np.where(rfm_df['RFM_Score'] >
                                      4.5, "Top Customers",
                                      (np.where(
                                        rfm_df['RFM_Score'] > 4,
                                        "High value Customer",
                                        (np.where(
    rfm_df['RFM_Score'] > 3,
                             "Medium Value Customer",
                             np.where(rfm_df['RFM_Score'] > 1.6,
                            'Low Value Customers', 'Lost Customers'))))))
# rfm_df[['CustomerName', 'RFM_Score', 'Customer_segment']].head(20)
```

```
[67]: plt.pie(rfm_df.Customer_segment.value_counts(),
        labels=rfm_df.Customer_segment.value_counts().index,
        autopct='%.0f%%')
plt.show()
```



- According to the customer segmentation described above, approximately **15% of our clients are high value clients**, whereas the **majority of our clientele are low value and lost clients**

1.3.6 Cohort Analysis

[68]:
```
# create an invoice month

# Function for month
def get_month(x):
  return dt.datetime(x.year, x.month,1)

# apply the function
df['InvoiceMonth'] = df['OrderDate'].apply(get_month)
# create a column index with the minimum invoice date aka first time customer␣
↪was aquired
df['CohortMonth'] = df.groupby('CustomerKey')['InvoiceMonth'].transform('min')
```

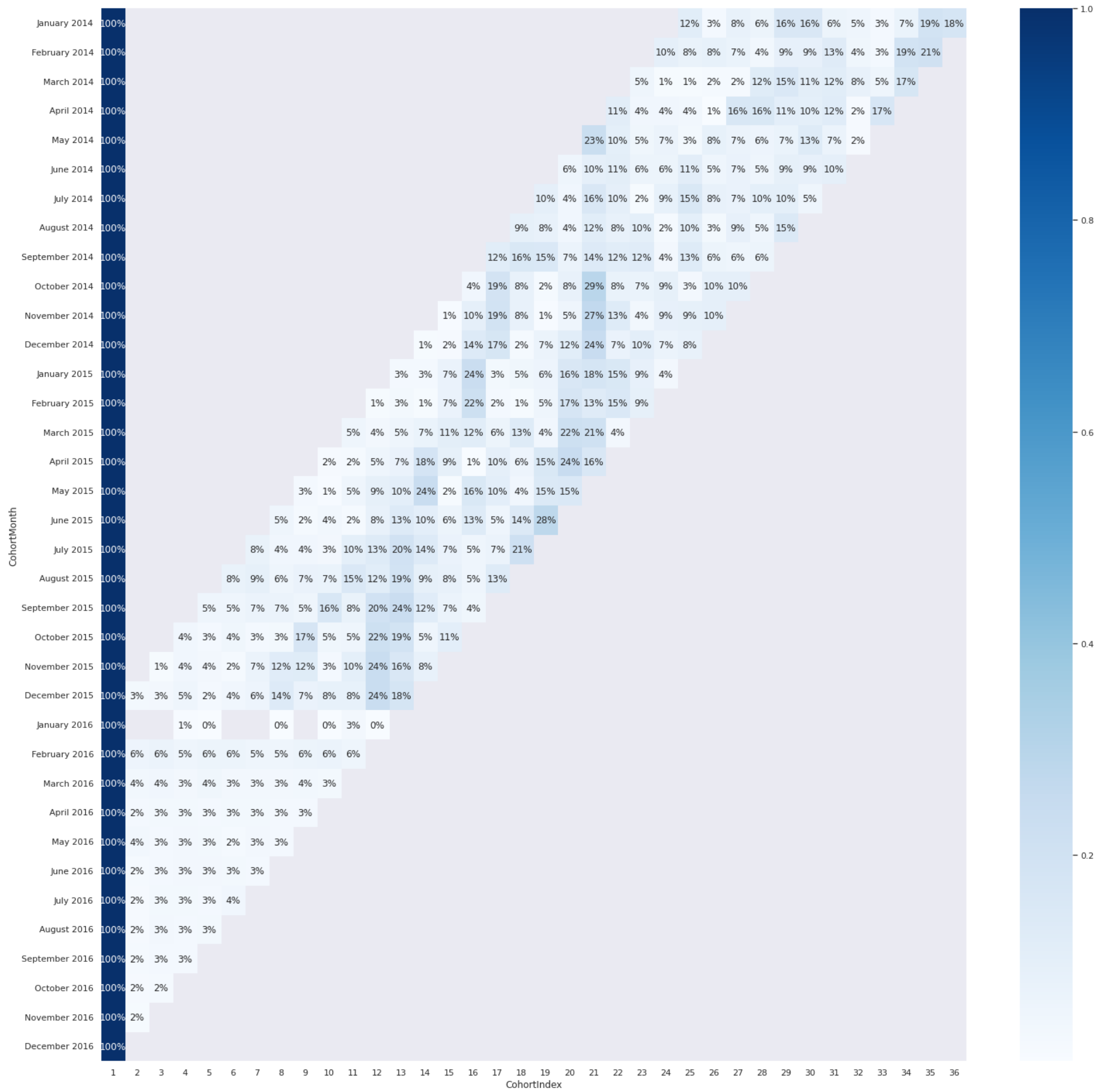
[69]:
```
# create a date element function to get a series for subtranction
def get_date_elements(data,column):
  day = data[column].dt.day
  month = data[column].dt.month
  year = data[column].dt.year
  return day, month, year
```

[70]:
```
# get date elements for our cohort and invoice columns(one dimentional Series)
_, Invoice_month, Invoice_year = get_date_elements(df, 'InvoiceMonth')
```

```
_, Cohort_month, Cohort_year = get_date_elements(df, 'CohortMonth')

# create a cohort index
year_diff = Invoice_year - Cohort_year
month_diff = Invoice_month - Cohort_month
df['CohortIndex'] = year_diff*12+month_diff+1

# count the customer ID by grouping by Cohort Month and Cohort index
cohort_data = df.groupby(['CohortMonth','CohortIndex'])['CustomerKey'].apply(pd.
↪Series.nunique).reset_index()

# create pivot table
cohort_table = cohort_data.pivot(index='CohortMonth',␣
↪columns=['CohortIndex'],values='CustomerKey')

# change index
cohort_table.index = cohort_table.index.strftime('%B %Y')

# cohort table for percentage
new_cohort_table = cohort_table.divide(cohort_table.iloc[:,0],axis=0)
```

[71]:
```
# create percentages
plt.figure(figsize=(25,25))
sns.heatmap(new_cohort_table, annot=True, cmap='Blues',fmt='.0%')
```

[71]: <AxesSubplot:xlabel='CohortIndex', ylabel='CohortMonth'>



- We can infer from the heatmap above that client retention in 2014 was subpar
- Since August of 2015, we have noticed some customers returning, though not in large numbers
- 2016 brought about a slight improvement in retention