In [93]:
```
!pip install openpyxl plotly -q
!pip install jovian --upgrade
```

```
Requirement already satisfied: jovian in c:\users\prajakta bose\anaconda3\lib\site-pa
ckages (0.2.47)
Requirement already satisfied: pyyaml in c:\users\prajakta bose\anaconda3\lib\site-pa
ckages (from jovian) (6.0)
Requirement already satisfied: requests in c:\users\prajakta bose\anaconda3\lib\site-
packages (from jovian) (2.26.0)
Requirement already satisfied: click in c:\users\prajakta bose\anaconda3\lib\site-pac
kages (from jovian) (8.0.3)
Requirement already satisfied: uuid in c:\users\prajakta bose\anaconda3\lib\site-pack
ages (from jovian) (1.30)
Requirement already satisfied: colorama in c:\users\prajakta bose\anaconda3\lib\site-
packages (from click->jovian) (0.4.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\prajakta bose\anaconda3\lib\s
ite-packages (from requests->jovian) (3.2)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\prajakta bose\anacon
da3\lib\site-packages (from requests->jovian) (1.26.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\prajakta bose\an
aconda3\lib\site-packages (from requests->jovian) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\prajakta bose\anaconda3
\lib\site-packages (from requests->jovian) (2021.10.8)
```

In [6]:
```
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')
```

Data Wrangling

In [14]:
```
Customers_data = pd.read_excel(r'C:\Users\Prajakta Bose\AdventureWorks_Database.xlsx
                               'Customers',
                               dtype={'CustomerKey':str},
                               parse_dates=['BirthDate','DateFirstPurchase']
                               )
```

In [16]:
```
Product_data = pd.read_excel('C:\\Users\\Prajakta Bose\\AdventureWorks_Database.xlsx
                               'Product',
                               dtype={'ProductKey':str},
                               parse_dates=['StartDate']
                               )
```

In [17]:
```
Sales_data = pd.read_excel('C:\\Users\\Prajakta Bose\\AdventureWorks_Database.xlsx',
                               'Sales',
                               dtype={'ProductKey':str,
                                      'CustomerKey':str,
                                      'PromotionKey':str,
                                      'SalesTerritoryKey':str},
```

```
                              parse_dates=['OrderDate', 'ShipDate']
                              )
Sales_data['DateKey'] = Sales_data['OrderDate'].astype(str)
```

In [18]:
```
Territory_data = pd.read_excel('C:\\Users\\Prajakta Bose\\AdventureWorks_Database.xl
                              'Territory',
                              dtype={'SalesTerritoryKey':str}
                              )
```

Merging Data

In [19]:
```
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')
```

Assessing Data

In [20]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 58189 entries, 0 to 58188
Data columns (total 58 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  Unnamed: 13           0 non-null      float64
 14  Unnamed: 14           0 non-null      float64
 15  Unnamed: 15           58189 non-null  float64
 16  Unnamed: 16           58189 non-null  float64
 17  Unnamed: 17           0 non-null      float64
 18  Unnamed: 18           58189 non-null  float64
 19  Unnamed: 19           0 non-null      float64
 20  StandardCost_x        58189 non-null  float64
 21  List Price            58189 non-null  float64
 22  Unnamed: 22           0 non-null      float64
 23  diif std cost         58189 non-null  int64
 24  diff list price       58189 non-null  int64
 25  DateKey               58189 non-null  object
 26  ProductName           58189 non-null  object
 27  SubCategory           58189 non-null  object
 28  Category              58189 non-null  object
 29  StandardCost_y        58189 non-null  float64
 30  Color                 30747 non-null  object
 31  ListPrice             58189 non-null  float64
 32  DaysToManufacture     58189 non-null  int64
 33  ProductLine           58189 non-null  object
 34  ModelName             58189 non-null  object
 35  Photo                 58189 non-null  object
```

```
 36  ProductDescription    58189 non-null  object
 37  StartDate             58189 non-null  datetime64[ns]
 38  FirstName             58189 non-null  object
 39  LastName              58189 non-null  object
 40  FullName              58189 non-null  object
 41  BirthDate             58189 non-null  datetime64[ns]
 42  MaritalStatus         58189 non-null  object
 43  Gender                58189 non-null  object
 44  YearlyIncome          58189 non-null  int64
 45  TotalChildren         58189 non-null  int64
 46  NumberChildrenAtHome  58189 non-null  int64
 47  Education             58189 non-null  object
 48  Occupation            58189 non-null  object
 49  HouseOwnerFlag        58189 non-null  int64
 50  NumberCarsOwned       58189 non-null  int64
 51  AddressLine1          58189 non-null  object
 52  DateFirstPurchase     58189 non-null  datetime64[ns]
 53  CommuteDistance       58189 non-null  object
 54  Region                58189 non-null  object
 55  Country               58189 non-null  object
 56  Group                 58189 non-null  object
 57  RegionImage           58189 non-null  object
dtypes: datetime64[ns](5), float64(16), int64(10), object(27)
memory usage: 26.2+ MB
```

In [21]:

```
# 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: 58
```

In [22]:

```
df.describe().transpose()
```

Out[22]:

	count	mean	std	min	25%	50%
SalesOrderLineNumber	58189.0	1.887453	1.018829	1.0000	1.0000	2.0000
OrderQuantity	58189.0	1.569386	1.047532	1.0000	1.0000	1.0000
UnitPrice	58189.0	413.888218	833.052938	0.5725	4.9900	24.4900
TotalProductCost	58189.0	296.539185	560.171436	0.8565	3.3623	12.1924
SalesAmount	58189.0	503.666270	941.462817	2.2900	8.9900	32.6000
TaxAmt	58189.0	40.293303	75.317027	0.1832	0.7192	2.6080
Unnamed: 13	0.0	NaN	NaN	NaN	NaN	NaN
Unnamed: 14	0.0	NaN	NaN	NaN	NaN	NaN
Unnamed: 15	58189.0	503.666269	941.462815	2.2900	8.9900	32.6000
Unnamed: 16	58189.0	0.000001	0.000014	0.0000	0.0000	0.0000
Unnamed: 17	0.0	NaN	NaN	NaN	NaN	NaN
Unnamed: 18	58189.0	38.398254	667.349417	-5106.9068	1.4335	6.2537
Unnamed: 19	0.0	NaN	NaN	NaN	NaN	NaN
StandardCost_x	58189.0	296.539185	560.171436	0.8565	3.3623	12.1924
List Price	58189.0	503.666270	941.462817	2.2900	8.9900	32.6000

	count	mean	std	min	25%	50%	
Unnamed: 22	0.0	NaN	NaN	NaN	NaN	NaN	
diif std cost	58189.0	0.000000	0.000000	0.0000	0.0000	0.0000	
diff list price	58189.0	0.000000	0.000000	0.0000	0.0000	0.0000	
StandardCost_y	58189.0	296.539185	560.171436	0.8565	3.3623	12.1924	
ListPrice	58189.0	503.666270	941.462817	2.2900	8.9900	32.6000	
DaysToManufacture	58189.0	1.045215	1.757395	0.0000	0.0000	0.0000	
YearlyIncome	58189.0	59769.887779	33128.041818	10000.0000	30000.0000	60000.0000	8(
TotalChildren	58189.0	1.838921	1.614467	0.0000	0.0000	2.0000	
NumberChildrenAtHome	58189.0	1.073502	1.580055	0.0000	0.0000	0.0000	
HouseOwnerFlag	58189.0	0.690560	0.462267	0.0000	0.0000	1.0000	
NumberCarsOwned	58189.0	1.502466	1.155496	0.0000	1.0000	2.0000	

In [23]:

```
# Check for duplicate data
df.duplicated().sum()
```

Out[23]: 0

Handling Missing Data

In [24]:

```
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
    Final = Final.sort_values(by = 'Missing_Percentage (%)',ascending = False)
    return Final
```

In [25]:

```
# Applying the custom function
missing_pct(df)
```

Out[25]:

	Column	Missing_value_count	Missing_Percentage (%)
22	Unnamed: 22	58189	100.00

	Column	Missing_value_count	Missing_Percentage (%)
19	Unnamed: 19	58189	100.00
14	Unnamed: 14	58189	100.00
13	Unnamed: 13	58189	100.00
17	Unnamed: 17	58189	100.00
30	Color	27442	47.16
0	ProductKey	0	0.00
42	MaritalStatus	0	0.00
41	BirthDate	0	0.00
39	LastName	0	0.00
40	FullName	0	0.00
38	FirstName	0	0.00
37	StartDate	0	0.00
36	ProductDescription	0	0.00
35	Photo	0	0.00
34	ModelName	0	0.00
43	Gender	0	0.00
44	YearlyIncome	0	0.00
32	DaysToManufacture	0	0.00
45	TotalChildren	0	0.00
46	NumberChildrenAtHome	0	0.00
47	Education	0	0.00
48	Occupation	0	0.00
49	HouseOwnerFlag	0	0.00
50	NumberCarsOwned	0	0.00
51	AddressLine1	0	0.00
52	DateFirstPurchase	0	0.00
53	CommuteDistance	0	0.00
54	Region	0	0.00
55	Country	0	0.00
56	Group	0	0.00
33	ProductLine	0	0.00
29	StandardCost_y	0	0.00
31	ListPrice	0	0.00
12	TaxAmt	0	0.00
2	ShipDate	0	0.00
3	CustomerKey	0	0.00

	Column	Missing_value_count	Missing_Percentage (%)
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
15	Unnamed: 15	0	0.00
1	OrderDate	0	0.00
16	Unnamed: 16	0	0.00
18	Unnamed: 18	0	0.00
20	StandardCost_x	0	0.00
21	List Price	0	0.00
23	diif std cost	0	0.00
24	diff list price	0	0.00
25	DateKey	0	0.00
26	ProductName	0	0.00
27	SubCategory	0	0.00
28	Category	0	0.00
57	RegionImage	0	0.00

In [26]:

```
#  Drop columns with nan values
df= df.dropna(axis=1)
```

Adding columns

In [27]:

```
# 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
```

Exploration of data

List of product category

In [28]:
```
df['Category'].unique().tolist()
```

Out[28]:
```
['Bikes', 'Accessories', 'Clothing']
```

List of Products Subcategory

In [30]:
```
df['SubCategory'].unique().tolist()
```

Out[30]:
```
['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 Unit price

In [31]:
```
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

In [33]:
```
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.
```

In [34]:
```
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

In [35]:
```
n_salesordernumber = df.groupby(['SalesOrderNumber'])['SalesOrderLineNumber'].transf
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 these three to two products are ordered in a single order

Sales order quatity distribution

In [37]:

```
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', );
```

Age distribution

In [39]:

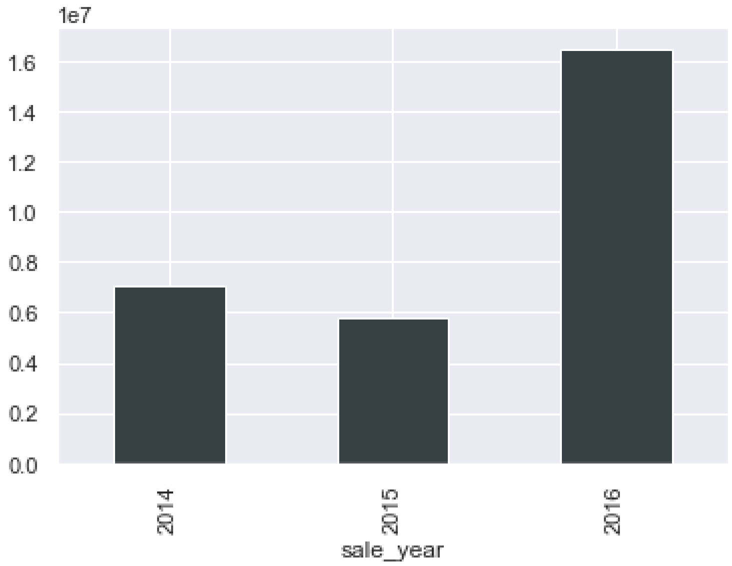
```
bins = [18, 30, 40, 50, 60, 70, 120]
labels = ['18-29', '30-39', '40-49', '50-59', '60-69', '70+']
df['agerange'] = pd.cut(df.Age, bins, labels = labels,include_lowest = True)

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_s
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.

Sales

Year wise sales

In [40]:
```
df.groupby('sale_year')['SalesAmount'].sum().plot(kind='bar', color='#374045');
```



The year 2016 saw an exponential surge in sales

Top 5 Selling Product

In [41]:
```
top_selling_product = df.groupby(['Category', 'SubCategory', 'ProductName'])['OrderQ
top_selling_product
```

Out[41]:

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

In [42]:
```
top_selling_product.reset_index(inplace=True)
fig = px.bar(top_selling_product, x='ProductName', y='OrderQuantity',color_discrete_
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

In [43]:
```
cat_subcat_qty = df.groupby(['sale_year','Category', 'SubCategory'])['OrderQuantity'
cat_subcat_qty = cat_subcat_qty.sort_values(['sale_year', 'Category'], ascending=Tru
cat_subcat_qty.style.bar(subset=['OrderQuantity'], color='#D9B300')
```

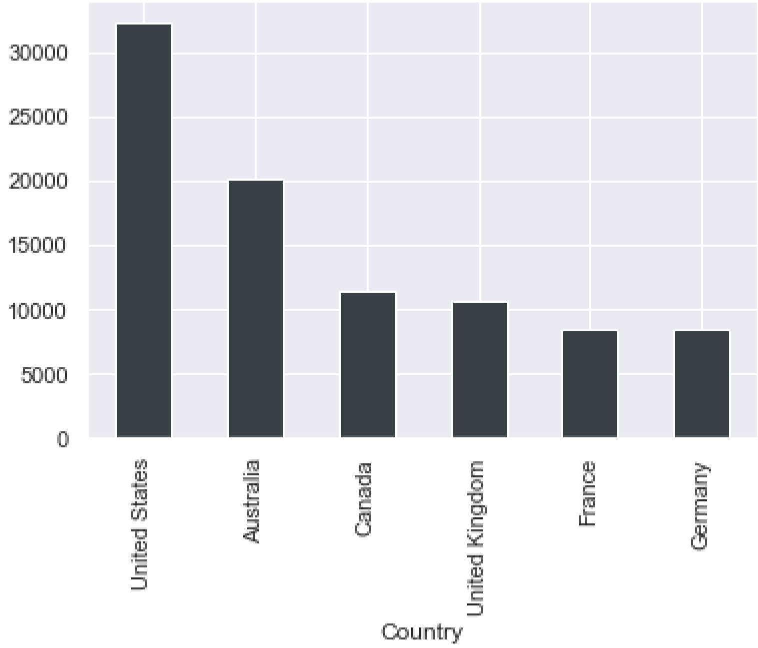
Out[43]:

sale_year	Category	SubCategory	OrderQuantity
2014	Bikes	Mountain Bikes	616
		Road Bikes	2876
2015	Bikes	Mountain Bikes	1661
		Road Bikes	3284
2016	Accessories	Bike Racks	493
		Bike Stands	394
		Bottles and Cages	12055
		Cleaners	1381
		Fenders	3239
		Helmets	9685
		Hydration Packs	1124
		Tires and Tubes	25518
	Bikes	Mountain Bikes	5490
		Road Bikes	6535
		Touring Bikes	3410
	Clothing	Caps	3178
		Gloves	2143
		Jerseys	5068
		Shorts	1491
		Socks	856
		Vests	824

Country wise quantity ordered

In [44]:

```
country_qty_sales = df.groupby('Country')['OrderQuantity'].sum().sort_values(ascendi
country_qty_sales.plot(kind='bar', color='#374045');
```



High quantity of products is ordered from Australia and United States

Profit

Overall profit based on order year, category and subcategory

In [45]:
```
cat_subcat_profit = df.groupby(['sale_year','Category', 'SubCategory'])['profit'].su

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

Out[45]:

sale_year	Category	SubCategory	profit
2014	Bikes	Mountain Bikes	586874.557600
		Road Bikes	2256280.998300
2015	Bikes	Mountain Bikes	1019388.334900
		Road Bikes	1375064.915000
2016	Accessories	Bike Racks	23136.960000
		Bike Stands	23689.092000
		Bottles and Cages	34448.978300
		Cleaners	4299.868800
		Fenders	27711.633000
		Helmets	135167.732700
		Hydration Packs	24303.132200
		Tires and Tubes	144793.083200
	Bikes	Mountain Bikes	2907361.198000
		Road Bikes	1905953.736400
		Touring Bikes	1454872.695900

sale_year	Category	SubCategory	profit
	Clothing	Caps	4331.831500
		Gloves	20895.744100
		Jerseys	37965.228300
		Shorts	41973.524600
		Socks	3055.841100
		Vests	20948.777000

It is observed that major profit is given by the bike category

Low profit contributing product

In [46]:

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

Out[46]:

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

In [47]:

```
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()
```

In [53]:
```
product_sales = df.groupby('Category')['profit'].sum().reset_index()
```

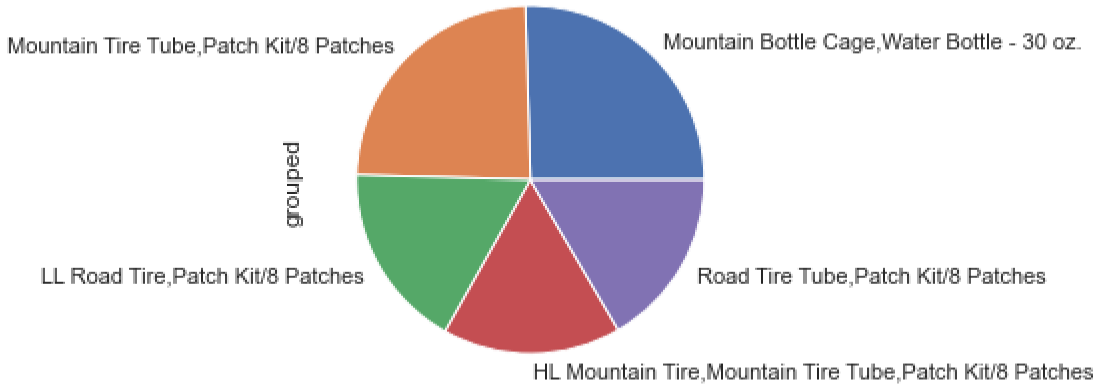
In [59]:
```
plt.figure(figsize=(12,5))
sns.barplot(x='Category', y='profit', data=product_sales)
plt.xticks(rotation=90)
plt.xlabel('Category')
plt.ylabel('Profit')
plt.title('Sales by Product Category')
plt.tight_layout()
plt.savefig('sales_by_category.png')
```

Which products are often sold together?

In [64]:
```
# By setting keep on False, all duplicates are True since we only want repeated orde
dup_order = df[df['SalesOrderNumber'].duplicated(keep=False)]
```

In [65]:
```
# 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(lambd
dup_order = dup_order[['SalesOrderNumber', 'grouped']].drop_duplicates()
```

In [66]:
```
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

In [67]:

```
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)
```

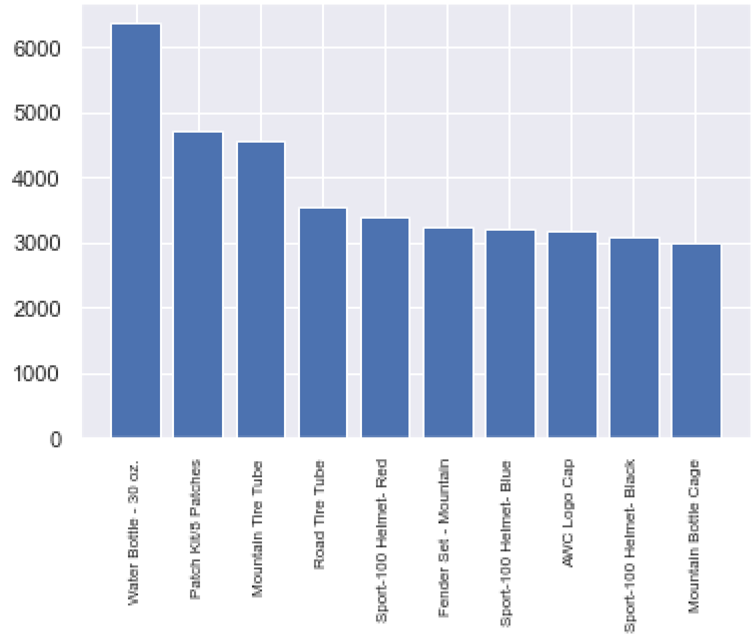
```
('Mountain Bottle Cage', 'Water Bottle - 30 oz.') 1623
('Road Bottle Cage', 'Water Bottle - 30 oz.') 1513
('HL Mountain Tire', 'Mountain Tire Tube') 915
('Touring Tire', 'Touring Tire Tube') 758
('Mountain Tire Tube', 'Patch Kit/8 Patches') 737
('Mountain Tire Tube', 'ML Mountain Tire') 727
('Water 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 products can be sold in bundle

To calculate which product is sold the most

In [68]:

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

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

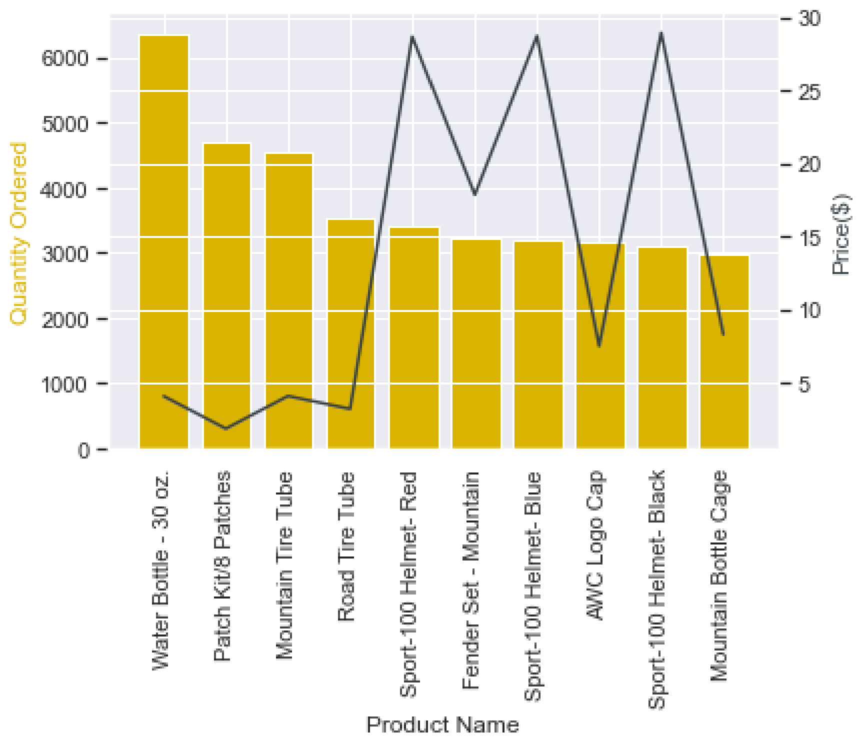


In [69]:

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

In [70]:

```
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();
```



In [71]:
```
prices.corr(quantity_ordered)
```

Out[71]:
```
-0.5333019792658484
```

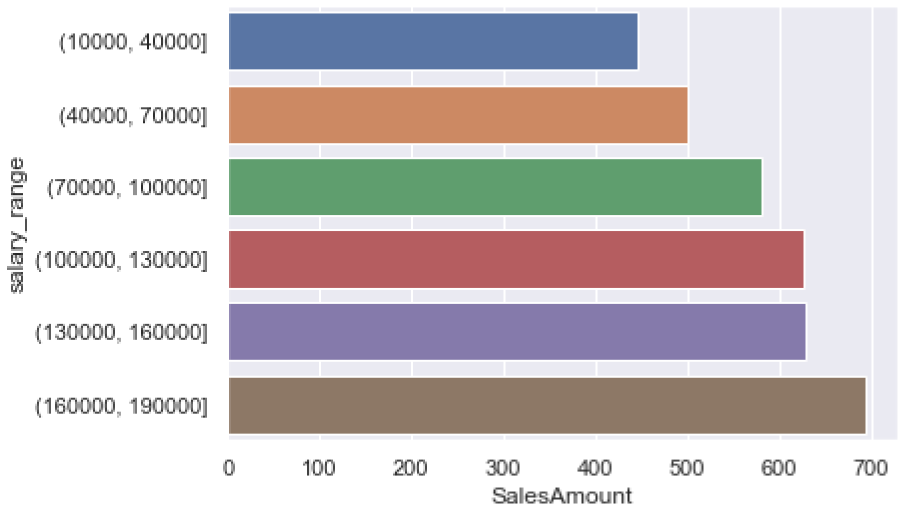
From the above correlation we can conclude that low price product has high demand.

Correlation between yearly income range and purchase

In [72]:
```
def create_bins(lower_bound, width, quantity):
    bins = []
    for low in range(lower_bound,
                     lower_bound + quantity*width + 1, width):
        bins.append((low, low+width))
    return bins
```

In [73]:
```
bins = create_bins(lower_bound=10000,
                   width=30000,
                   quantity=5)
bins2 = pd.IntervalIndex.from_tuples(bins)
df['salary_range'] = pd.cut(df['YearlyIncome'], bins2)
```

In [74]:
```
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);
```



From above we can conclude that higher salary range leads to increase in purchase.

Analyse sales by customer segment

In [76]:
```
# 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.
# 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
```

In [77]:
```
#  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)
```

In [78]:

```
#  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()
```

In [79]:

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

In [80]:

```
# 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()
```

In [81]:

```
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()
```

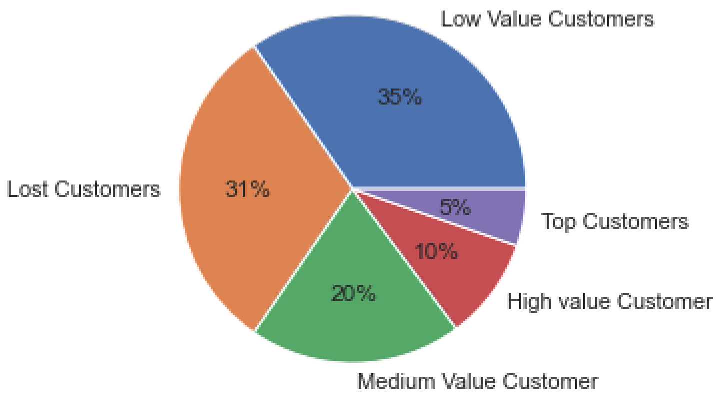
In [82]:

```
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)
```

In [83]:

```
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)
```

In [84]:
```
plt.pie(rfm_df.Customer_segment.value_counts(),
        labels=rfm_df.Customer_segment.value_counts().index,
        autopct='%.0f%%')
plt.show()
```



According to the customer sales segmentation described above, approximately 15% of our clients are high value, whereas the majority of our client are low value and lost customers

Cohort Analysis

In [98]:
```
# 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 aq
df['CohortMonth'] = df.groupby('CustomerKey')['InvoiceMonth'].transform('min')
```

In [99]:
```
# 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
```

In [100…
```
# 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.Seri

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

# change index
```

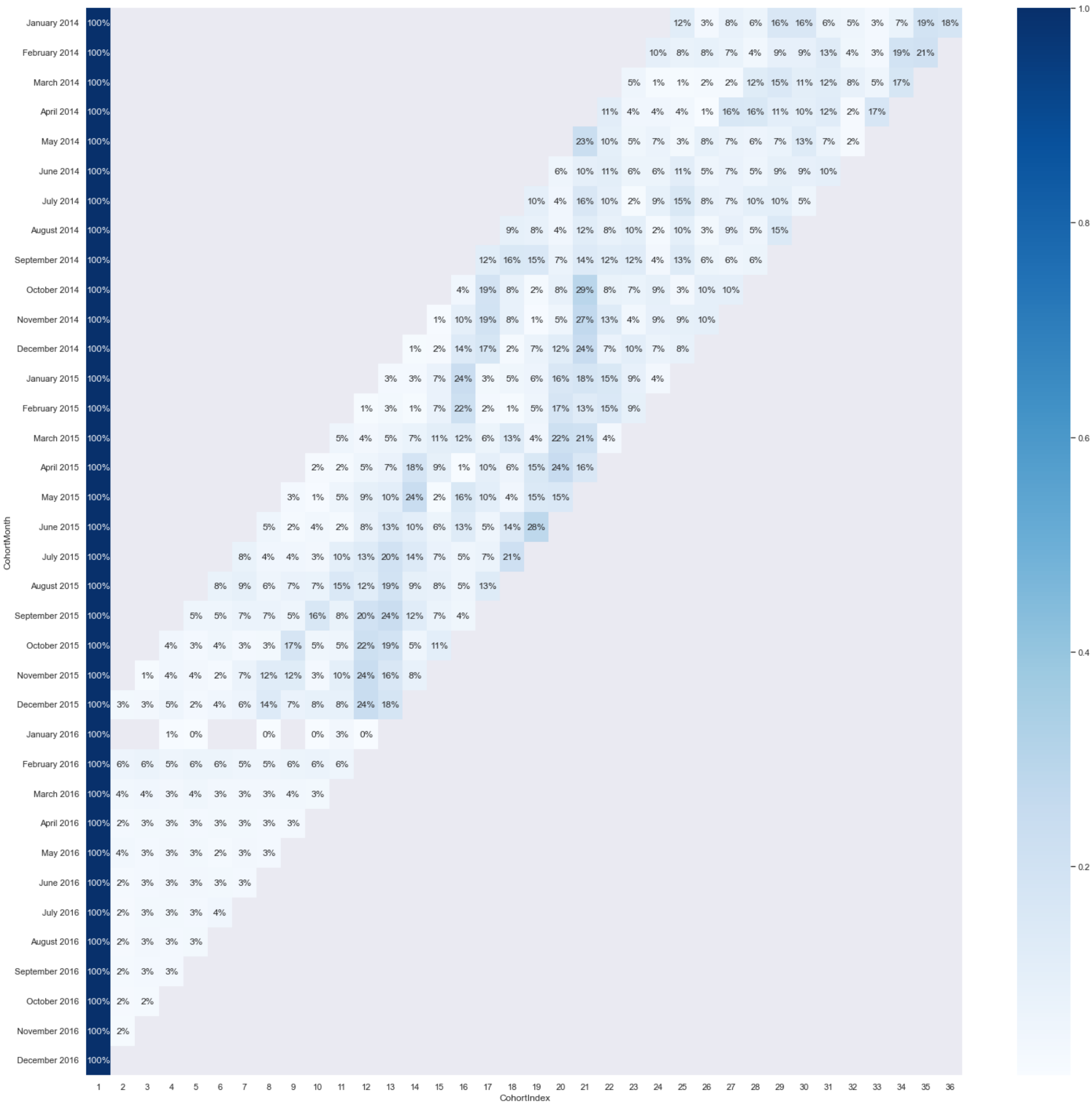
```python
cohort_table.index = cohort_table.index.strftime('%B %Y')

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

In [101…

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

Out[101…

```
<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

In [ ]: