Empresa de Calzado

Problematica: Inventario alborotado

La empresa no puede tener un control presiso de su inventario debido a la gran variedad de talles y no poder tener un conocimiento exacto de la demanda de cada talle en particular. La empresa necesita saber esto para tener suficiente stock para cubrir la demanda y ,por otro lado, a la vez, no saturar el lugar que alberga su inventario con exceso de talles que superen la misma.

Abordaje al problema

Mediante un Dataset que contiene las ventas realizadas por la empresa en los años 2014, 2015 y 2016, Voy a utilizar estadisticas para determinar demanda y tambien intervalos de confianza para lograr predecir la demanda del 2017.

Importando las librerias a utilizar

In [38]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math
```

Se enlaza el csv de donde extraigo los datos

In [8]:

```
df = pd.read_csv("calzados.csv")
df
```

Out[8]:

	InvoiceNo	Date	Country	ProductID	Shop	Gender	Size (US)	Size (Europe)	Size (UK)	UnitPrice	Discount	SalePrice
0	52389	1/1/2014	United Kingdom	2152	UK2	Male	11.0	44	10.5	$159.00	0%	$159.00
1	52390	1/1/2014	United States	2230	US15	Male	11.5	44-45	11.0	$199.00	20%	$159.20
2	52391	1/1/2014	Canada	2160	CAN7	Male	9.5	42-43	9.0	$149.00	20%	$119.20
3	52392	1/1/2014	United States	2234	US6	Female	9.5	40	7.5	$159.00	0%	$159.00
4	52393	1/1/2014	United Kingdom	2222	UK4	Female	9.0	39-40	7.0	$159.00	0%	$159.00

...	...	...	...	...	...	...	...	...	...	...	...	...
14962	65773	12/31/2016	United Kingdom	2154	UK2	Male	9.5	42-43	9.0	$139.00	0%	$139.00
14963	65774	12/31/2016	United States	2181	US12	Female	12.0	42-43	10.0	$149.00	0%	$149.00
14964	65775	12/31/2016	Canada	2203	CAN6	Male	10.5	43-44	10.0	$179.00	30%	$125.30
14965	65776	12/31/2016	Germany	2231	GER1	Female	9.5	40	7.5	$199.00	0%	$199.00
14966	65777	12/31/2016	Germany	2156	GER1	Female	6.5	37	4.5	$139.00	10%	$125.10

14967 rows × 12 columns

Comienzo a partir e la columna de fecha ("Date")

quiero saber que tipo de dato es...

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14967 entries, 0 to 14966
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   InvoiceNo      14967 non-null  int64  
 1   Date           14967 non-null  object 
 2   Country        14967 non-null  object 
 3   ProductID      14967 non-null  int64  
 4   Shop           14967 non-null  object 
 5   Gender         14967 non-null  object 
 6   Size (US)      14967 non-null  float64
 7   Size (Europe)  14967 non-null  object 
 8   Size (UK)      14967 non-null  float64
 9   UnitPrice      14967 non-null  object 
 10  Discount       14967 non-null  object 
 11  SalePrice      14967 non-null  object 
dtypes: float64(2), int64(2), object(8)
memory usage: 1.4+ MB
```

"Date" es un "object", procedo a usarlo como "datetime"

In [10]:

```
df["Date"] = pd.to_datetime(df["Date"])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14967 entries, 0 to 14966
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   InvoiceNo      14967 non-null  int64         
 1   Date           14967 non-null  datetime64[ns]
 2   Country        14967 non-null  object        
 3   ProductID      14967 non-null  int64         
 4   Shop           14967 non-null  object        
 5   Gender         14967 non-null  object        
 6   Size (US)      14967 non-null  float64       
 7   Size (Europe)  14967 non-null  object        
 8   Size (UK)      14967 non-null  float64       
 9   UnitPrice      14967 non-null  object        
 10  Discount       14967 non-null  object        
 11  SalePrice      14967 non-null  object        
dtypes: datetime64[ns](1), float64(2), int64(2), object(7)
memory usage: 1.4+ MB
```

In [11]:

```
df["Year"] = df["Date"].dt.year
df["Day"] = df["Date"].dt.day
df["Month"] = df["Date"].dt.month
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14967 entries, 0 to 14966
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   InvoiceNo      14967 non-null  int64         
 1   Date           14967 non-null  datetime64[ns]
 2   Country        14967 non-null  object        
 3   ProductID      14967 non-null  int64
 4   Shop           14967 non-null  object
 5   Gender         14967 non-null  object
 6   Size (US)      14967 non-null  float64
 7   Size (Europe)  14967 non-null  object
 8   Size (UK)      14967 non-null  float64
 9   UnitPrice      14967 non-null  object
 10  Discount       14967 non-null  object
 11  SalePrice      14967 non-null  object
 12  Year           14967 non-null  int64
 13  Day            14967 non-null  int64
 14  Month          14967 non-null  int64
dtypes: datetime64[ns](1), float64(2), int64(5), object(7)
memory usage: 1.7+ MB
```

"Unit Price" y "SalePrice" es un "object", procedo a sacarle el signo de dolar para usarlo como flotante

In [12]:

```
df["SalePrice"] = df["SalePrice"].apply(lambda x: float(x[2:]))
df["UnitPrice"] = df["UnitPrice"].apply(lambda x: float(x[2:]))
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14967 entries, 0 to 14966
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   InvoiceNo      14967 non-null  int64
 1   Date           14967 non-null  datetime64[ns]
 2   Country        14967 non-null  object
 3   ProductID      14967 non-null  int64
 4   Shop           14967 non-null  object
 5   Gender         14967 non-null  object
 6   Size (US)      14967 non-null  float64
 7   Size (Europe)  14967 non-null  object
 8   Size (UK)      14967 non-null  float64
 9   UnitPrice      14967 non-null  float64
 10  Discount       14967 non-null  object
 11  SalePrice      14967 non-null  float64
 12  Year           14967 non-null  int64
 13  Day            14967 non-null  int64
 14  Month          14967 non-null  int64
dtypes: datetime64[ns](1), float64(4), int64(5), object(5)
memory usage: 1.7+ MB
```

Ahora voy a ver los talles

In [13]:

```
df.describe()
```

Out[13]:

	InvoiceNo	ProductID	Size (US)	Size (UK)	UnitPrice	SalePrice	Year	Day	Month
count	14967.000000	14967.000000	14967.000000	14967.000000	14967.000000	14967.000000	14967.000000	14967.000000	14967.000000
mean	59050.261509	2195.325115	9.195630	8.089497	164.171377	143.987913	2015.308211	15.745306	6.689517
std	3889.598714	27.633526	1.511719	1.970014	22.940544	35.180799	0.762320	8.719764	3.319909
min	52389.000000	2147.000000	4.500000	2.500000	129.000000	64.500000	2014.000000	1.000000	1.000000
25%	55648.500000	2172.000000	8.000000	6.500000	149.000000	125.100000	2015.000000	8.000000	4.000000
50%	59092.000000	2195.000000	9.000000	8.500000	159.000000	149.000000	2015.000000	16.000000	7.000000
75%	62433.000000	2219.000000	10.000000	9.500000	179.000000	169.000000	2016.000000	23.000000	10.000000
max	65777.000000	2242.000000	15.000000	14.500000	199.000000	199.000000	2016.000000	31.000000	12.000000

Separo variables Categoricas de Numericas

In [14]:

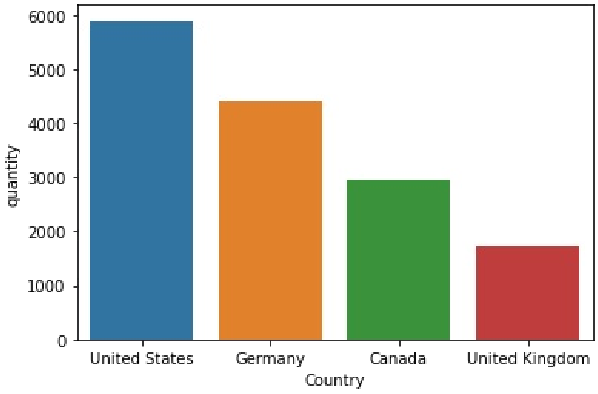
```
categorical_variables = ["Country", "ProductID", "Shop", "Gender", "Size (US)", "Discount", "Year", "Month"]
numerical_variables = ["UnitPrice", "SalePrice"]
```

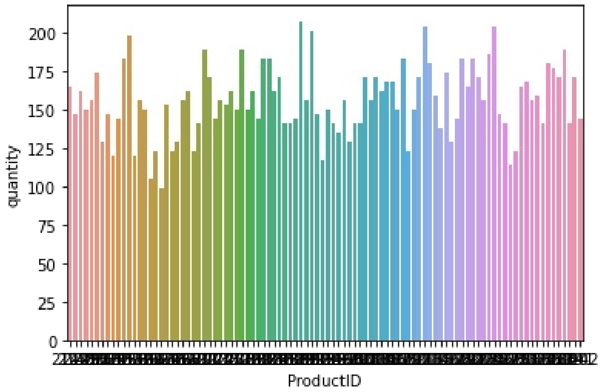
Analizo las variables mediante un loop para graficar

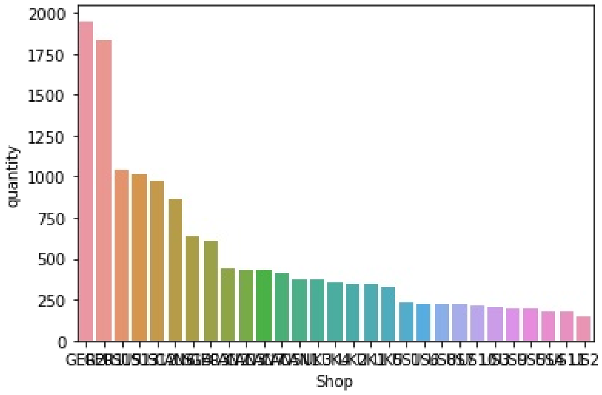
In [16]:

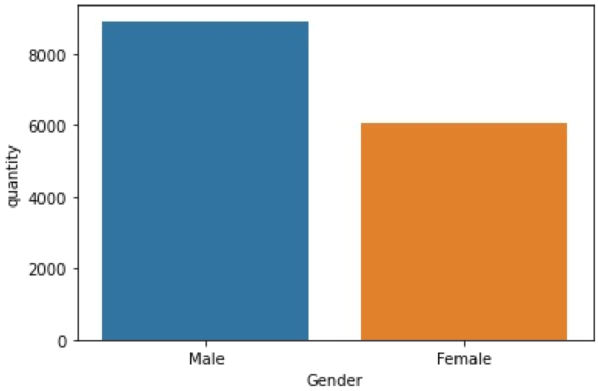
```
for cat_variable in categorical_variables:
```

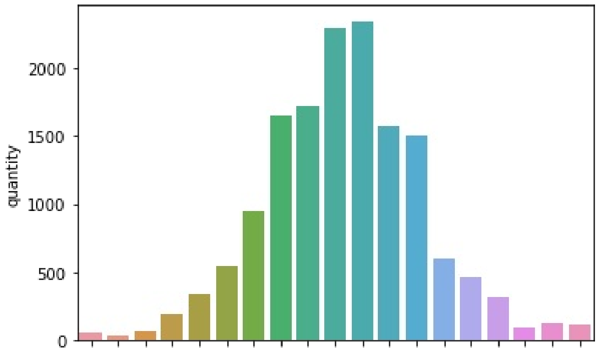
```
for cat_variable in categorical_variables:
    frecuencia = df[cat_variable].value_counts()
    df_frecuencia = pd.DataFrame({cat_variable: frecuencia.index.tolist(), "quantity": frecuencia.tolist()})
    sns.barplot(x=cat_variable, y="quantity", data=df_frecuencia)
    plt.show()
```

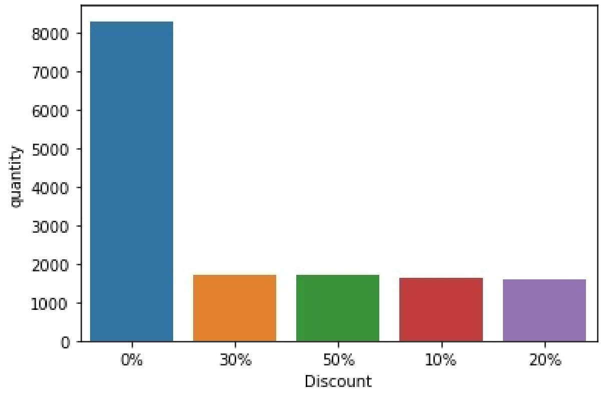


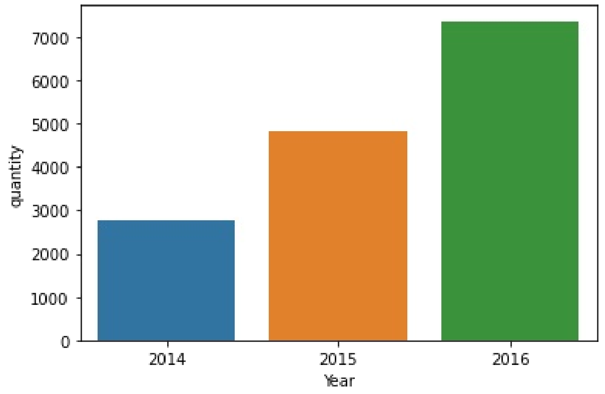


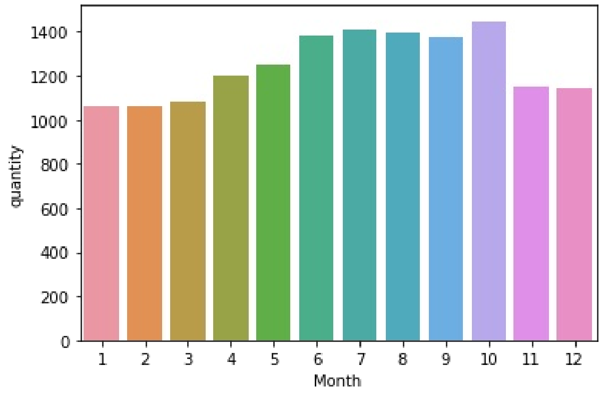






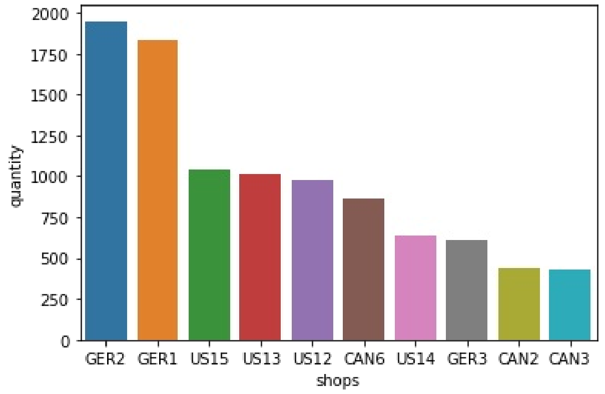






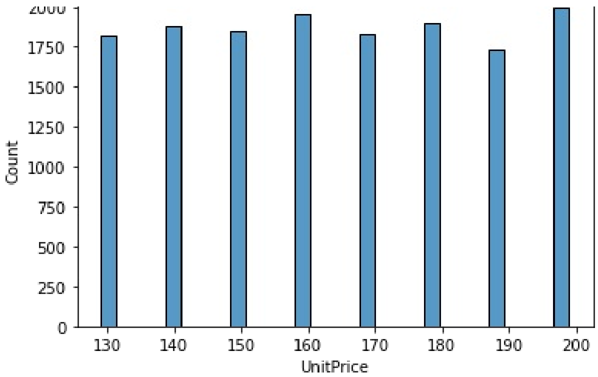
In [26]:

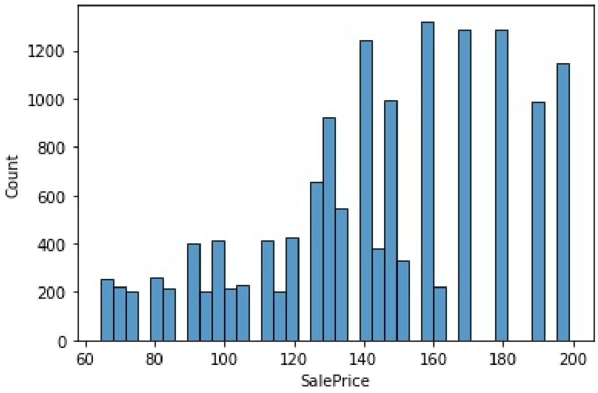
```
frequencia_shops = df['Shop'].value_counts().head(10)
df_frequencia_shops = pd.DataFrame({'shops': frequencia_shops.index.tolist(), 'quantity': frequencia_shops.tolist
sns.barplot(x='shops', y='quantity', data=df_frequencia_shops)
plt.show()
```



In [17]:

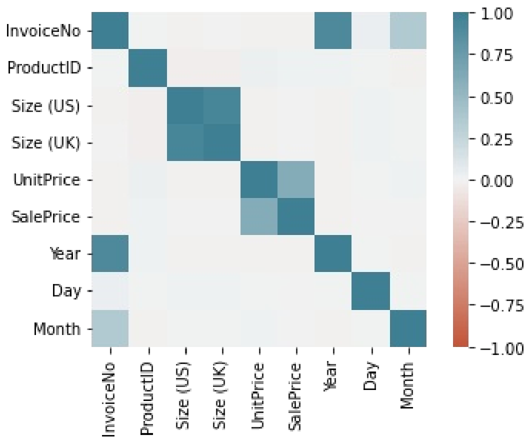
```
for num_variable in numerical_variables:
    sns.histplot(df[num_variable], bins="auto")
    plt.show()
```





In [24]:

```
correlacion = df.corr()
sns.heatmap(correlacion, vmin=-1, vmax=1, center=0,
            cmap=sns.diverging_palette(20, 220, n=200),
            square=True)
plt.show()
```



Uso un intervalo de confianza para predecir demanda del 2017

Empiezo agrupando los datos

In [27]:

```
grouped = df[(df['Year'] != 2014) & (df['Gender'] == 'Male') & (df['Country'] == 'United States')]\
    .groupby(['Size (US)', 'Year', 'Month']).size().unstack(level=0).fillna(value=0)
```

Calculo la media de cada columna (cuanto min y cuanto max se venden los calzados)

Creo dos array: uno con la media y otro con la desviacion std

In [29]:

```
media = []
desviacion = []
for column in grouped.columns:
    media.append(grouped[column].mean())
    desviacion.append(grouped[column].sem())
```

Lo meto en un dataframe

In [36]:

```
d = {'means': means, 'std_error': desviacion}
df_calculations = pd.DataFrame(data=d, index=grouped.columns)
```

Calculo margen de error

In [43]:

```
df_calculations['error_margin'] = df_calculations['std_error'].apply(lambda x: x * 2.07)
df_calculations['low_margin'] = df_calculations.apply(lambda x: x['means'] - x['error_margin'], axis=1)
df_calculations['up_margin'] = df_calculations.apply(lambda x: x['means'] + x['error_margin'], axis=1)
```

Redondeo hacia arriba

In [44]:

```
df_calculations['math_round_up'] = df_calculations.apply(lambda x: math.ceil(x['up_margin']), axis=1)
```

In [46]:

```
df_calculations
```

Out[46]:

	means	std_error	error_margin	low_margin	up_margin	math_round_up
Size (US)						
6.0	2.166667	0.393179	0.813880	1.352787	2.980546	3
6.5	1.583333	0.340059	0.703922	0.879411	2.287255	3
7.0	1.333333	0.338725	0.701160	0.632174	2.034493	3
7.5	2.333333	0.411196	0.851176	1.482158	3.184509	4
8.0	4.791667	0.598849	1.239618	3.552049	6.031284	7
8.5	7.875000	0.944689	1.955505	5.919495	9.830505	10
9.0	16.333333	1.262139	2.612628	13.720705	18.945961	19
9.5	25.583333	1.766144	3.655917	21.927416	29.239251	30
10.0	18.791667	1.325583	2.743957	16.047709	21.535624	22
10.5	14.958333	1.020584	2.112608	12.845725	17.070942	18
11.0	7.541667	0.719750	1.489883	6.051784	9.031550	10
11.5	5.333333	0.582556	1.205892	4.127442	6.539225	7
12.0	3.083333	0.580219	1.201054	1.882279	4.284388	5
13.0	1.208333	0.255229	0.528324	0.680009	1.736658	2
14.0	1.958333	0.297813	0.616473	1.341860	2.574806	3
15.0	0.541667	0.190148	0.393607	0.148060	0.935274	1

Conclusiones:

Una de las muchas conclusiones es que Los calzados de talle 9.5 son los mas demandados a nivel mundial y la empresa cuenta con pocos pares del mismo. y con este metodo se pueden resolver muchas mas cuestiones relacionadas con el stock