In [1]:
```
pip install yellowbrick
```

```
Collecting yellowbrick
  Using cached yellowbrick-1.5-py3-none-any.whl (282 kB)
Collecting scikit-learn>=1.0.0
  Downloading scikit_learn-1.1.3-cp39-cp39-win_amd64.whl (7.6 MB)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in c:\users\dristi\anaconda3\lib\site-packages (from yel
lowbrick) (3.4.3)
Requirement already satisfied: numpy>=1.16.0 in c:\users\dristi\anaconda3\lib\site-packages (from yellowbrick)
(1.20.3)
Requirement already satisfied: scipy>=1.0.0 in c:\users\dristi\anaconda3\lib\site-packages (from yellowbrick) (1.
7.1)
Requirement already satisfied: cycler>=0.10.0 in c:\users\dristi\anaconda3\lib\site-packages (from yellowbrick)
(0.10.0)
Requirement already satisfied: six in c:\users\dristi\anaconda3\lib\site-packages (from cycler>=0.10.0->yellowbri
ck) (1.16.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\dristi\anaconda3\lib\site-packages (from matplotl
ib!=3.0.0,>=2.0.2->yellowbrick) (2.7.5)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\dristi\anaconda3\lib\site-packages (from matplotlib!
=3.0.0,>=2.0.2->yellowbrick) (1.3.1)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\dristi\anaconda3\lib\site-packages (from matplotlib!=
```

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
data = pd.read_csv("sales_data_sample.csv", encoding='Latin-1')
data.head()
```

Out[2]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	5	2003
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003

5 rows × 25 columns

In [3]:
```
data.shape
```

Out[3]:
```
(2823, 25)
```

In [4]:
```
data.isnull().sum()
```

Out[4]:
```
ORDERNUMBER           0
QUANTITYORDERED       0
PRICEEACH             0
ORDERLINENUMBER       0
SALES                 0
ORDERDATE             0
STATUS                0
QTR_ID                0
MONTH_ID              0
YEAR_ID               0
PRODUCTLINE           0
MSRP                  0
PRODUCTCODE           0
CUSTOMERNAME          0
PHONE                 0
ADDRESSLINE1          0
ADDRESSLINE2       2521
CITY                  0
STATE              1486
POSTALCODE           76
COUNTRY               0
TERRITORY          1074
CONTACTLASTNAME       0
CONTACTFIRSTNAME      0
DEALSIZE              0
dtype: int64
```

In [5]:
```
data.drop(["ORDERNUMBER", "PRICEEACH", "ORDERDATE", "PHONE", "ADDRESSLINE1", "ADDRESSLINE2", "CITY", "STATE", "TER
```

In [6]:
```
data.head()
```

Out[6]:

	QUANTITYORDERED	ORDERLINENUMBER	SALES	STATUS	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE	MSRP	PRODUCTCODE	CU
0	30	2	2871.00	Shipped	1	2	2003	Motorcycles	95	S10_1678	I
1	34	5	2765.90	Shipped	2	5	2003	Motorcycles	95	S10_1678	Re
2	41	2	3884.34	Shipped	3	7	2003	Motorcycles	95	S10_1678	
3	45	6	3746.70	Shipped	3	8	2003	Motorcycles	95	S10_1678	Toys
4	49	14	5205.27	Shipped	4	10	2003	Motorcycles	95	S10_1678	Cor

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
QUANTITYORDERED    0
ORDERLINENUMBER    0
SALES              0
STATUS             0
QTR_ID             0
MONTH_ID           0
YEAR_ID            0
PRODUCTLINE        0
MSRP               0
PRODUCTCODE        0
CUSTOMERNAME       0
COUNTRY            0
DEALSIZE           0
dtype: int64
```

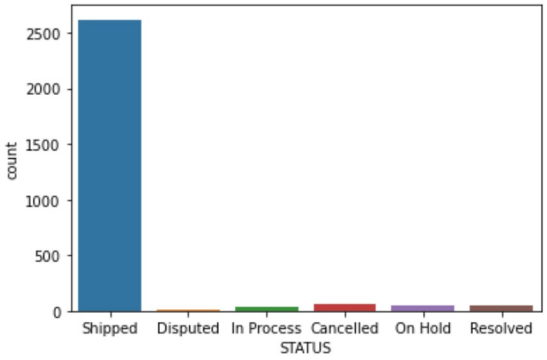
In [8]:
```
data.describe()
```

Out[8]:

	QUANTITYORDERED	ORDERLINENUMBER	SALES	QTR_ID	MONTH_ID	YEAR_ID	MSRP
count	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.00000	2823.000000
mean	35.092809	6.466171	3553.889072	2.717676	7.092455	2003.81509	100.715551
std	9.741443	4.225841	1841.865106	1.203878	3.656633	0.69967	40.187912
min	6.000000	1.000000	482.130000	1.000000	1.000000	2003.00000	33.000000
25%	27.000000	3.000000	2203.430000	2.000000	4.000000	2003.00000	68.000000
50%	35.000000	6.000000	3184.800000	3.000000	8.000000	2004.00000	99.000000
75%	43.000000	9.000000	4508.000000	4.000000	11.000000	2004.00000	124.000000
max	97.000000	18.000000	14082.800000	4.000000	12.000000	2005.00000	214.000000

In [9]:
```
sns.countplot(data = data , x = 'STATUS')
```

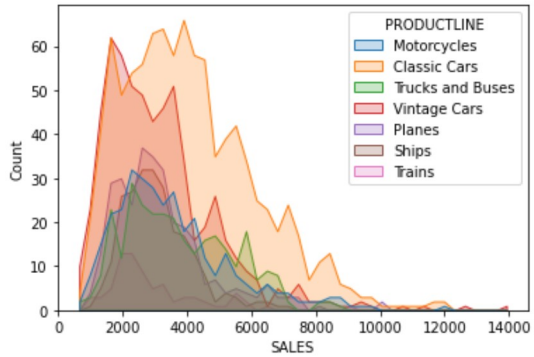
Out[9]:
```
<AxesSubplot:xlabel='STATUS', ylabel='count'>
```



In [10]:
```
import seaborn as sns
```

In [11]:
```
sns.histplot(x = 'SALES' , hue = 'PRODUCTLINE', data = data,
             element="poly")
```

Out[11]:
```
<AxesSubplot:xlabel='SALES', ylabel='Count'>
```



In [12]:
```
data['PRODUCTLINE'].unique()
```

Out[12]:
```
array(['Motorcycles', 'Classic Cars', 'Trucks and Buses', 'Vintage Cars',
       'Planes', 'Ships', 'Trains'], dtype=object)
```

In [13]:
```
#checking the duplicated values
data.drop_duplicates(inplace=True)
```

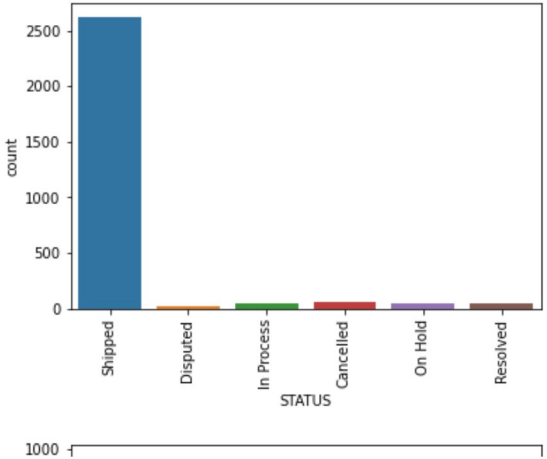
In [14]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2823 entries, 0 to 2822
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   QUANTITYORDERED  2823 non-null   int64  
 1   ORDERLINENUMBER  2823 non-null   int64  
 2   SALES            2823 non-null   float64
 3   STATUS           2823 non-null   object 
 4   QTR_ID           2823 non-null   int64  
 5   MONTH_ID         2823 non-null   int64  
 6   YEAR_ID          2823 non-null   int64  
 7   PRODUCTLINE      2823 non-null   object 
 8   MSRP             2823 non-null   int64  
 9   PRODUCTCODE      2823 non-null   object 
 10  CUSTOMERNAME     2823 non-null   object 
 11  COUNTRY          2823 non-null   object 
 12  DEALSIZE         2823 non-null   object 
dtypes: float64(1), int64(6), object(6)
memory usage: 308.8+ KB
```

In [15]:
```
list_cat = data.select_dtypes(include=['object']).columns.tolist()
list_cat
```

Out[15]:
```
['STATUS', 'PRODUCTLINE', 'PRODUCTCODE', 'CUSTOMERNAME', 'COUNTRY', 'DEALSIZE']
```

In [16]:
```
for i in list_cat:
  sns.countplot(data = data ,x = i)
  plt.xticks(rotation = 90)
  plt.show()
```



In [18]:
```
#dealing with the catagorical features
from sklearn import preprocessing
le = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
for i in list_cat:
  data[i]= le.fit_transform(data[i])
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2823 entries, 0 to 2822
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   QUANTITYORDERED  2823 non-null   int64  
 1   ORDERLINENUMBER  2823 non-null   int64  
 2   SALES            2823 non-null   float64
 3   STATUS           2823 non-null   int64  
 4   QTR_ID           2823 non-null   int64  
 5   MONTH_ID         2823 non-null   int64  
 6   YEAR_ID          2823 non-null   int64  
 7   PRODUCTLINE      2823 non-null   int64  
 8   MSRP             2823 non-null   int64  
 9   PRODUCTCODE      2823 non-null   int64  
 10  CUSTOMERNAME     2823 non-null   int64  
 11  COUNTRY          2823 non-null   int64  
 12  DEALSIZE         2823 non-null   int64  
dtypes: float64(1), int64(12)
memory usage: 373.3 KB
```

In [20]:
```
data['SALES'] = data['SALES'].astype(int)
```

In [21]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2823 entries, 0 to 2822
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   QUANTITYORDERED  2823 non-null   int64
 1   ORDERLINENUMBER  2823 non-null   int64
 2   SALES            2823 non-null   int32
 3   STATUS           2823 non-null   int64
 4   QTR_ID           2823 non-null   int64
 5   MONTH_ID         2823 non-null   int64
 6   YEAR_ID          2823 non-null   int64
 7   PRODUCTLINE      2823 non-null   int64
 8   MSRP             2823 non-null   int64
 9   PRODUCTCODE      2823 non-null   int64
 10  CUSTOMERNAME     2823 non-null   int64
 11  COUNTRY          2823 non-null   int64
 12  DEALSIZE         2823 non-null   int64
dtypes: int32(1), int64(12)
memory usage: 362.3 KB
```

In [22]:
```
data.describe()
```

Out[22]:

	QUANTITYORDERED	ORDERLINENUMBER	SALES	STATUS	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE	MS
count	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.00000	2823.000000	2823.000
mean	35.092809	6.466171	3553.421537	4.782501	2.717676	7.092455	2003.81509	2.515055	100.715
std	9.741443	4.225841	1841.865754	0.879416	1.203878	3.656633	0.69967	2.411665	40.187
min	6.000000	1.000000	482.000000	0.000000	1.000000	1.000000	2003.00000	0.000000	33.000
25%	27.000000	3.000000	2203.000000	5.000000	2.000000	4.000000	2003.00000	0.000000	68.000
50%	35.000000	6.000000	3184.000000	5.000000	3.000000	8.000000	2004.00000	2.000000	99.000
75%	43.000000	9.000000	4508.000000	5.000000	4.000000	11.000000	2004.00000	5.000000	124.000
max	97.000000	18.000000	14082.000000	5.000000	4.000000	12.000000	2005.00000	6.000000	214.000

In [24]:
```
## taget feature are Sales and productline
X = data[['SALES','PRODUCTCODE']]
data.columns
```

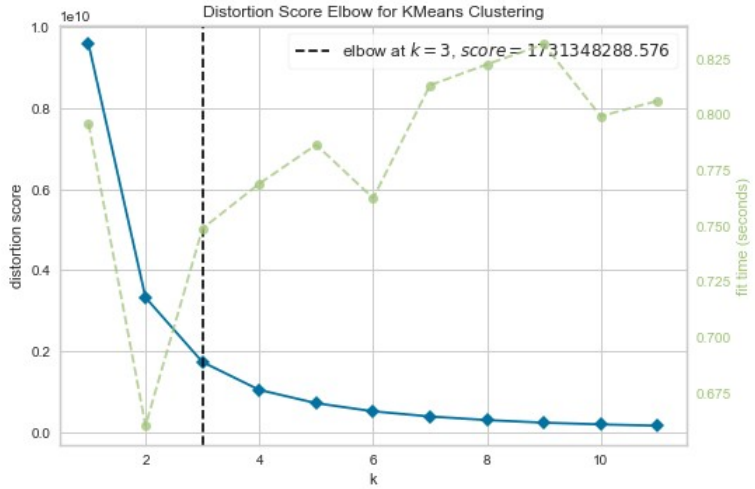
Out[24]:
```
Index(['QUANTITYORDERED', 'ORDERLINENUMBER', 'SALES', 'STATUS', 'QTR_ID',
       'MONTH_ID', 'YEAR_ID', 'PRODUCTLINE', 'MSRP', 'PRODUCTCODE',
       'CUSTOMERNAME', 'COUNTRY', 'DEALSIZE'],
      dtype='object')
```

In [25]:
```
#K Means implementation
```

In [27]:
```
from yellowbrick.cluster import KElbowVisualizer
```

In [29]:
```
from sklearn.cluster import KMeans
```

In [30]:
```
model = KMeans()
visualizer = KElbowVisualizer(model, k=(1,12)).fit(X)
visualizer.show()
```



Out[30]:
```
<AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion scor
e'>
```

In [31]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4, init='k-means++', random_state=0).fit(X)
```

In [32]:
```
kmeans.labels_
```

Out[32]:
```
array([0, 0, 0, ..., 3, 2, 0])
```

In [33]:
```
kmeans.inertia_
```

Out[33]:
```
1042223216.6249839
```

In [34]:
```
kmeans.n_iter_
```

Out[34]:
```
24
```

In [35]:
```
kmeans.cluster_centers_
```

Out[35]:
```
array([[3416.59686888,   56.3072407 ],
       [7983.1758794 ,   28.05025126],
       [1879.28363988,   63.25072604],
       [5289.27065026,   41.01230228]])
```

In [36]:
```
#getting the size of the clusters
from collections import Counter
Counter(kmeans.labels_)
```

Out[36]:
```
Counter({0: 1024, 3: 565, 2: 1035, 1: 199})
```

In [37]:
```
sns.scatterplot(data=X, x="SALES", y="PRODUCTCODE", hue=kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],
            marker="X", c="r", s=80, label="centroids")
plt.legend()
plt.show()
```