Super Market Sales Prediction By Team SVM

In [119…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import scipy as sp
import warnings
import datetime
import xgboost as xgb


warnings.filterwarnings("ignore")
%matplotlib inline
```

In [73]:

```
pip install  xgboost
```

```
Requirement already satisfied: xgboost in c:\users\darisa supriya\anaconda3\lib\site
-packages (1.6.1)Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: scipy in c:\users\darisa supriya\anaconda3\lib\site-p
ackages (from xgboost) (1.7.1)
Requirement already satisfied: numpy in c:\users\darisa supriya\anaconda3\lib\site-p
ackages (from xgboost) (1.20.3)
```

Reading the dataset

In [75]:

```
data = pd.read_csv("supermarket_sales - Sheet1.csv")
```

interpreting the dataset

prints first five rows from dataset

In [76]:

```
data.head()
```

Out[76]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	

prints last five rows from dataset

```
In [77]: data.tail()
```

Out[77]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Tota
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	2.0175	42.367
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	48.6900	1022.490
997	727-02-1313	A	Yangon	Member	Male	Food and beverages	31.84	1	1.5920	33.432
998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	65.82	1	3.2910	69.111
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7	30.9190	649.299

Describe method is used to return the description of the data in the dataset

```
In [78]: data.describe()
```

Out[78]:

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1.000000e+03	1000.000000
mean	55.672130	5.510000	15.379369	322.966749	307.58738	4.761905e+00	15.379369
std	26.494628	2.923431	11.708825	245.885335	234.17651	6.131498e-14	11.708825
			0	10.678500	10.17000	4.761905e+00	0.508500

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income
25%	32.875000	3.000000	5.924875	124.422375	118.49750	4.761905e+00	5.924875
50%	55.230000	5.000000	12.088000	253.848000	241.76000	4.761905e+00	12.088000
75%	77.935000	8.000000	22.445250	471.350250	448.90500	4.761905e+00	22.445250
max	99.960000	10.000000	49.650000	1042.650000	993.00000	4.761905e+00	49.650000

info method is used to print the information about dataset

In [79]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Invoice ID               1000 non-null   object 
 1   Branch                   1000 non-null   object 
 2   City                     1000 non-null   object 
 3   Customer type            1000 non-null   object 
 4   Gender                   1000 non-null   object 
 5   Product line             1000 non-null   object 
 6   Unit price               1000 non-null   float64
 7   Quantity                 1000 non-null   int64  
 8   Tax 5%                   1000 non-null   float64
 9   Total                    1000 non-null   float64
 10  Date                     1000 non-null   object 
 11  Time                     1000 non-null   object 
 12  Payment                  1000 non-null   object 
 13  cogs                     1000 non-null   float64
 14  gross margin percentage  1000 non-null   float64
 15  gross income             1000 non-null   float64
 16  Rating                   1000 non-null   float64
dtypes: float64(7), int64(1), object(9)
memory usage: 132.9+ KB
```

Returns object containing counts of unique values

In [80]:

```
data.nunique()
```

Out[80]:

```
Invoice ID                 1000
Branch                        3
City                          3
Customer type                 2
Gender                        2
Product line                  6
Unit price                  943
Quantity                     10
Tax 5%                      990
Total                       990
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Date                        89
Time                       506
Payment                      3
cogs                       990
gross margin percentage      1
gross income               990
Rating                      61
dtype: int64
```

In [81]:

```
data
```

Out[81]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Tota
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.971
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.220
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.525
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.048
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.378
...	...	...	...	...	...	...	...	...	...	.
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	2.0175	42.367
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	48.6900	1022.490
997	727-02-1313	A	Yangon	Member	Male	Food and beverages	31.84	1	1.5920	33.432
998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	65.82	1	3.2910	69.111
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7	30.9190	649.299

1000 rows × 17 columns

returns total no.of rows and columns

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [82]:

```
data.shape
```

Out[82]:
```
(1000, 17)
```

returns datatypes present in the dataset

In [83]:
```
data.dtypes
```

Out[83]:
```
Invoice ID                  object
Branch                      object
City                        object
Customer type               object
Gender                      object
Product line                object
Unit price                 float64
Quantity                     int64
Tax 5%                     float64
Total                      float64
Date                        object
Time                        object
Payment                     object
cogs                       float64
gross margin percentage    float64
gross income               float64
Rating                     float64
dtype: object
```

prints columns names present in the dataset

In [84]:
```
data.columns
```

Out[84]:
```
Index(['Invoice ID', 'Branch', 'City', 'Customer type', 'Gender',
       'Product line', 'Unit price', 'Quantity', 'Tax 5%', 'Total', 'Date',
       'Time', 'Payment', 'cogs', 'gross margin percentage', 'gross income',
       'Rating'],
      dtype='object')
```

cleaning the dataset

return the count of NULLs/NaN values in each column

In [85]:
```
data.isnull().sum()
```

Out[85]:
```
Invoice ID                 0
Branch                     0
City                       0
Customer type              0
Gender                     0
Product line               0
Unit price                 0
Quantity                   0
Tax 5%                     0
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Date                       0
```

```
Time                       0
Payment                    0
cogs                       0
gross margin percentage    0
gross income               0
Rating                     0
dtype: int64
```

In [86]:

```
data.isnull()
```

Out[86]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time
0	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...
995	False	False	False	False	False	False	False	False	False	False	False	False
996	False	False	False	False	False	False	False	False	False	False	False	False
997	False	False	False	False	False	False	False	False	False	False	False	False
998	False	False	False	False	False	False	False	False	False	False	False	False
999	False	False	False	False	False	False	False	False	False	False	False	False

1000 rows × 17 columns

In [87]:

```
data.notnull()
```

Out[87]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time
0	True	True	True	True	True	True	True	True	True	True	True	True
1	True	True	True	True	True	True	True	True	True	True	True	True
2	True	True	True	True	True	True	True	True	True	True	True	True
3	True	True	True	True	True	True	True	True	True	True	True	True
4	True	True	True	True	True	True	True	True	True	True	True	True
...	...	...	...	...	...	...	...	...	...	...	...	...
995	True	True	True	True	True	True	True	True	True	True	True	True
996	True	True	True	True	True	True	True	True	True	True	True	True
997	True	True	True	True	True	True	True	True	True	True	True	True
998	True	True	True	True	True	True	True	True	True	True	True	True
999	True	True	True	True	True	True	True	True	True	True	True	True

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

1000 rows × 17 columns

In [88]:

```
data.notnull().head()
```

Out[88]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Pa
0	True	True	True	True	True	True	True	True	True	True	True	True	
1	True	True	True	True	True	True	True	True	True	True	True	True	
2	True	True	True	True	True	True	True	True	True	True	True	True	
3	True	True	True	True	True	True	True	True	True	True	True	True	
4	True	True	True	True	True	True	True	True	True	True	True	True	

In [89]:

```
data.notnull().tail()
```

Out[89]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time
995	True	True	True	True	True	True	True	True	True	True	True	True
996	True	True	True	True	True	True	True	True	True	True	True	True
997	True	True	True	True	True	True	True	True	True	True	True	True
998	True	True	True	True	True	True	True	True	True	True	True	True
999	True	True	True	True	True	True	True	True	True	True	True	True

it will check if a dataset contains NaN/None values in any cell This method returns True if it finds NaN/None...

In [90]:

```
data.isnull().any()
```

Out[90]:

```
Invoice ID                 False
Branch                     False
City                       False
Customer type              False
Gender                     False
Product line               False
Unit price                 False
Quantity                   False
Tax 5%                     False
Total                      False
Date                       False
Time                       False
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
cogs                       False
gross margin percentage    False
gross income               False
Rating                     False
dtype: bool
```

Exploratory Data Analysis

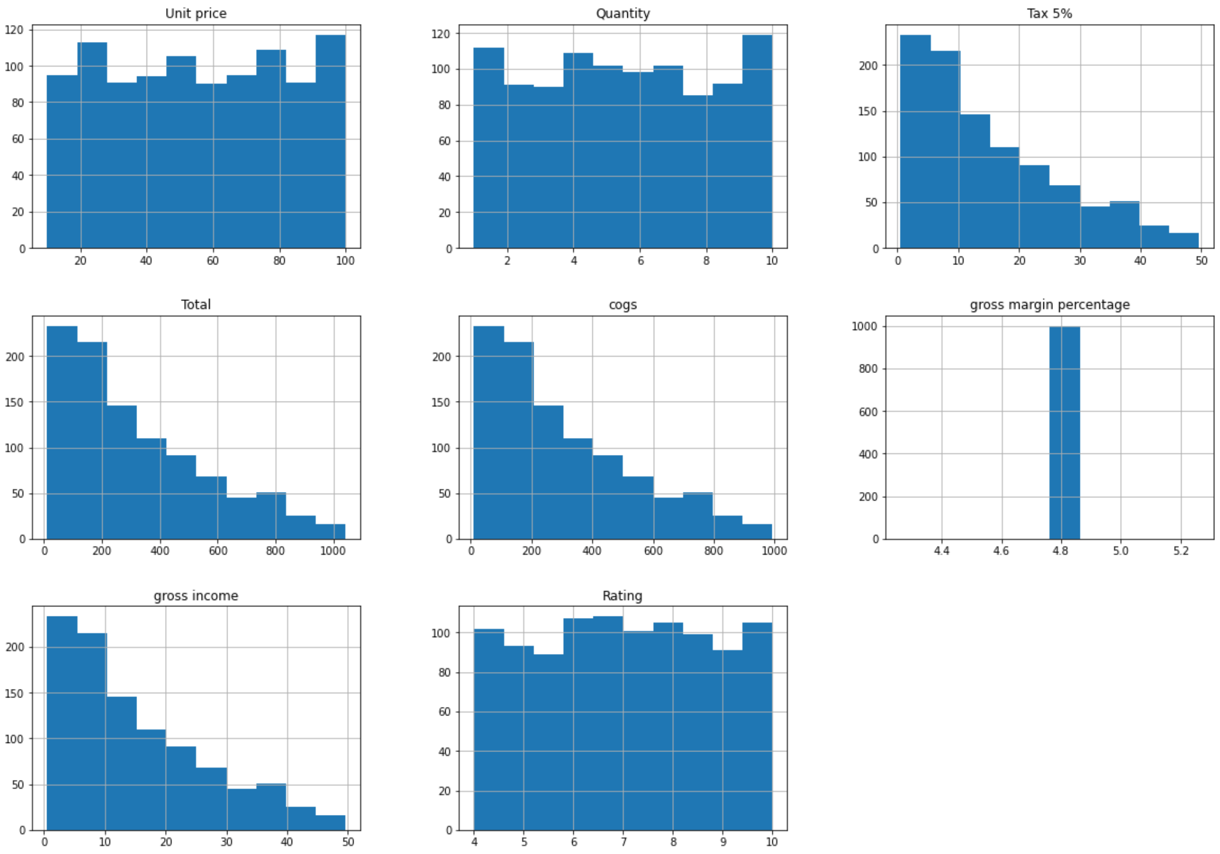
EDA is an approach to analyze the data using visual techniques.

A histogram is a graphical display of data using bars of different heights. It is a graphical representation of numerical data

figsize : tuple (width, height) - The size of the output image(graph)

In [91]:

```
data.hist(figsize=(20,14))
plt.show()
```



corr() calculates the relationship between each column in data set

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [92]:
```
data.corr()
```

Out[92]:

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rating
Unit price	1.000000	0.010778	0.633962	0.633962	0.633962	NaN	0.633962	-0.008778
Quantity	0.010778	1.000000	0.705510	0.705510	0.705510	NaN	0.705510	-0.015815
Tax 5%	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.036442
Total	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.036442
cogs	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.036442
gross margin percentage	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gross income	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.036442
Rating	-0.008778	-0.015815	-0.036442	-0.036442	-0.036442	NaN	-0.036442	1.000000

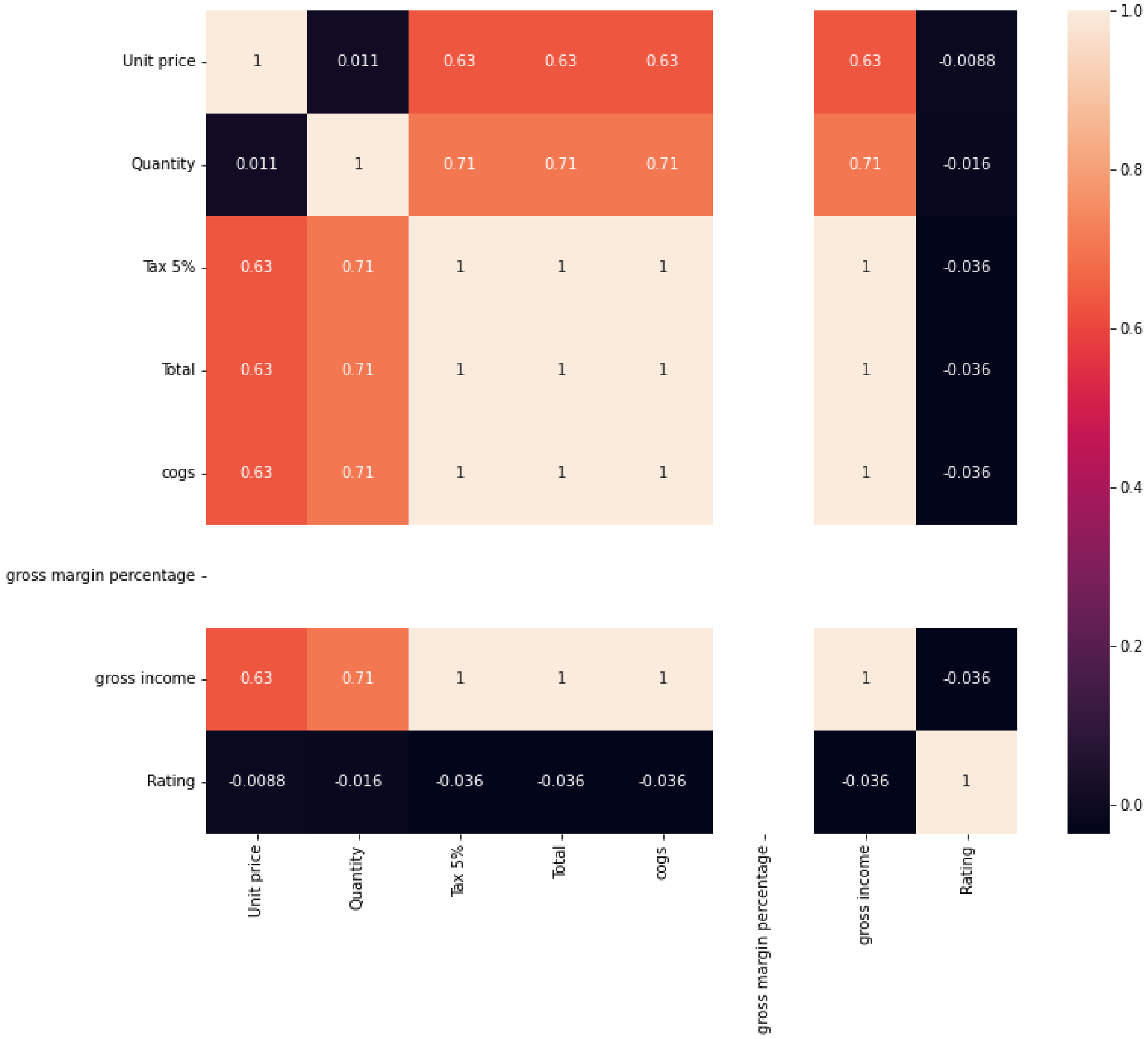
HEATMAP : It is a graphical representation of data that uses a system of color-coding to represent different values

In [93]:
```
plt.figure(figsize = (12,10))

sns.heatmap(data.corr(), annot =True)
```

Out[93]:
```
<AxesSubplot:>
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

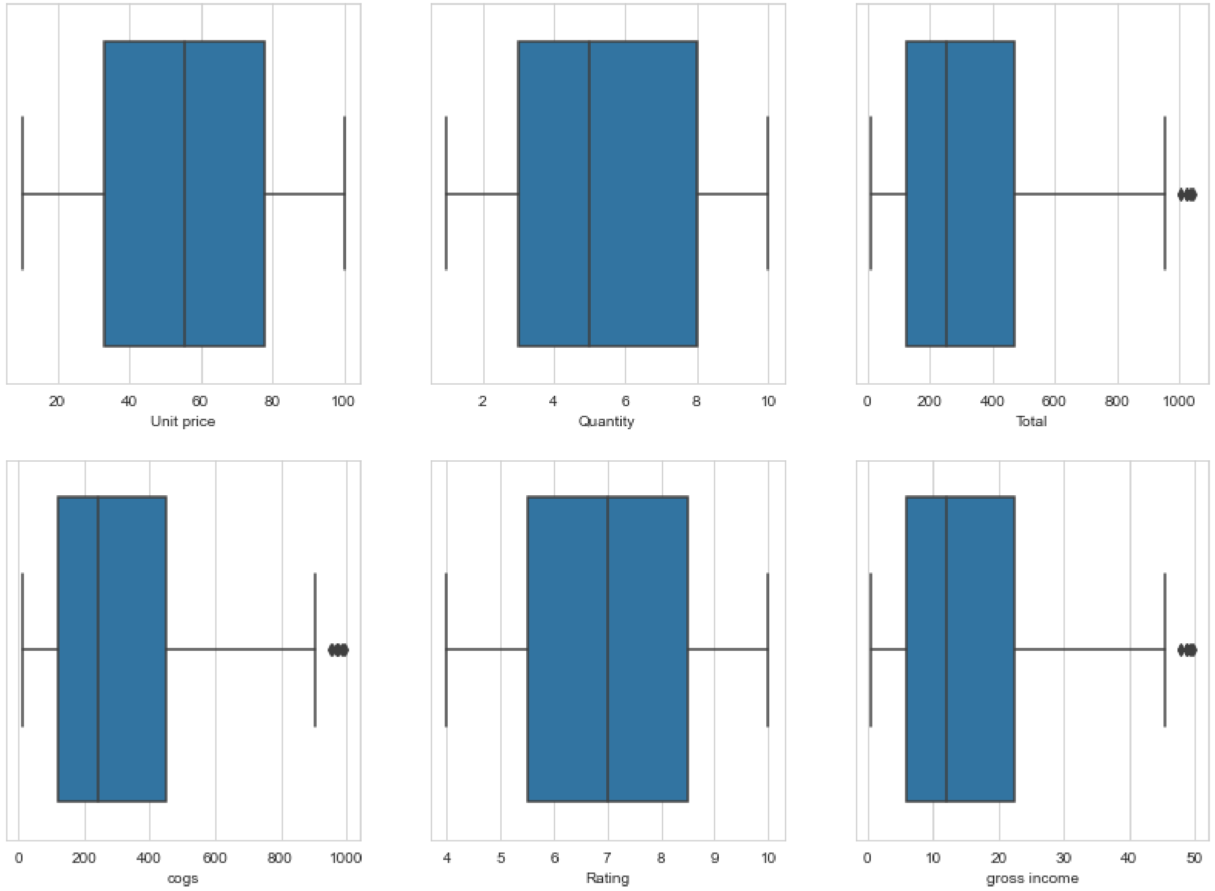


Boxplot : It is a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”).

In [94]:
```
plt.figure(figsize=(14,10))
sns.set_style(style='whitegrid')
plt.subplot(2,3,1)
sns.boxplot(x='Unit price',data=data)
plt.subplot(2,3,2)
sns.boxplot(x='Quantity',data=data)
plt.subplot(2,3,3)
sns.boxplot(x='Total',data=data)
plt.subplot(2,3,4)
sns.boxplot(x='cogs',data=data)
plt.subplot(2,3,5)
sns.boxplot(x='Rating',data=data)
plt.subplot(2,3,6)
sns.boxplot(x='gross income',data=data)
```

```
<AxesSubplot:xlabel='gross income'>
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



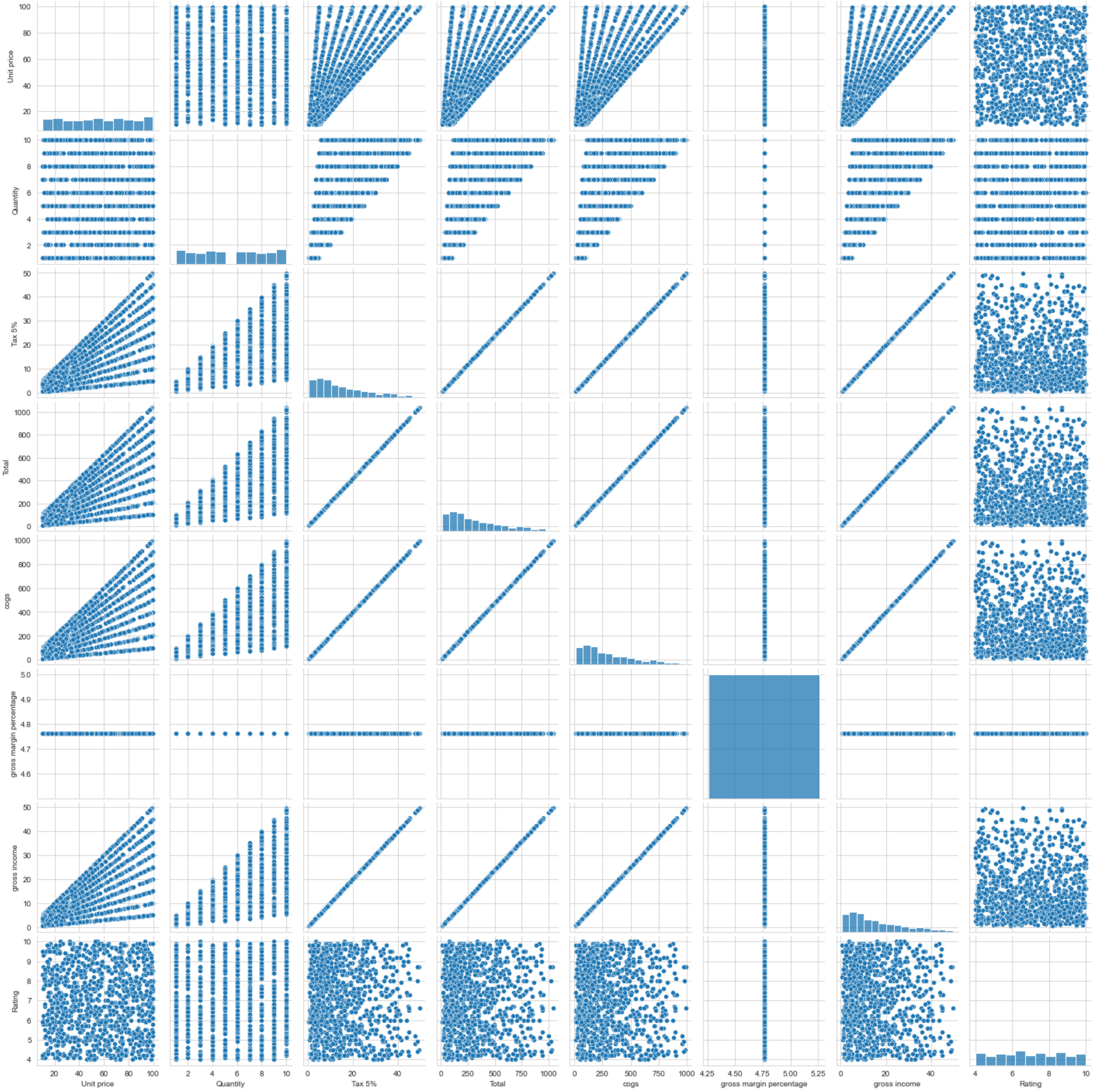
Pairplot : It is used to understand the best set of features to explain a relationship between two variables or to form the most separated clusters.

In [95]:

```
sns.pairplot(data=data)
```

Out[95]:

```
<seaborn.axisgrid.PairGrid at 0x13c5ab5cb20>
```



Regplot is used to plot data and a linear regression model fit

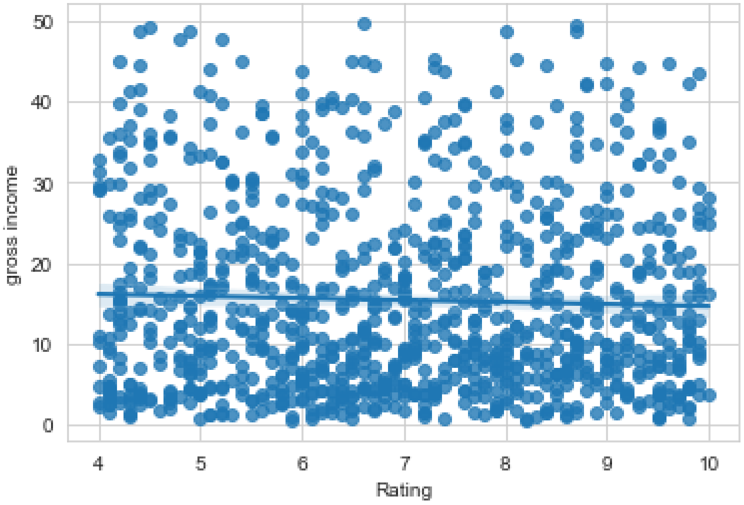
In [96]:

```
sns.regplot(x='Rating', y= 'gross income', data=data)
```

Out[96]:

```
<AxesSubplot:xlabel='Rating', ylabel='gross income'>
```

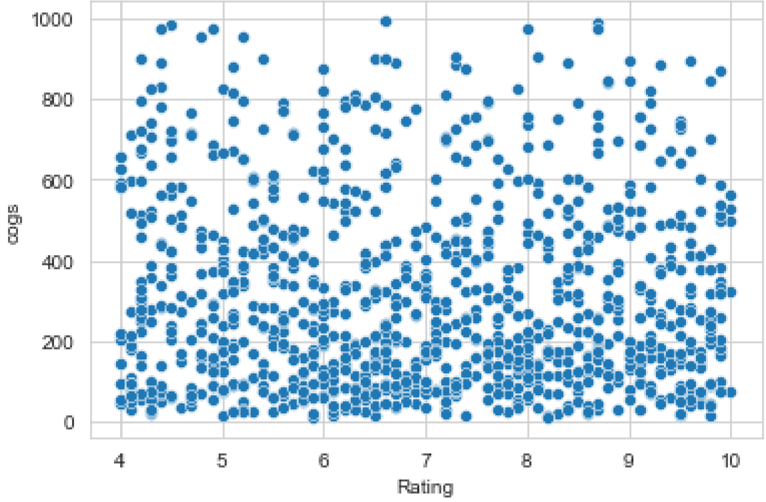
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



Scatterplot : Scatter plots are the graphs that present the relationship between two variables in a data-set. It represents data points on a two-dimensional plane or on a Cartesian system. The independent variable or attribute is plotted on the X-axis, while the dependent variable is plotted on the Y-axis.

```
In [97]: sns.scatterplot(x='Rating', y= 'cogs', data=data)
```

```
Out[97]: <AxesSubplot:xlabel='Rating', ylabel='cogs'>
```

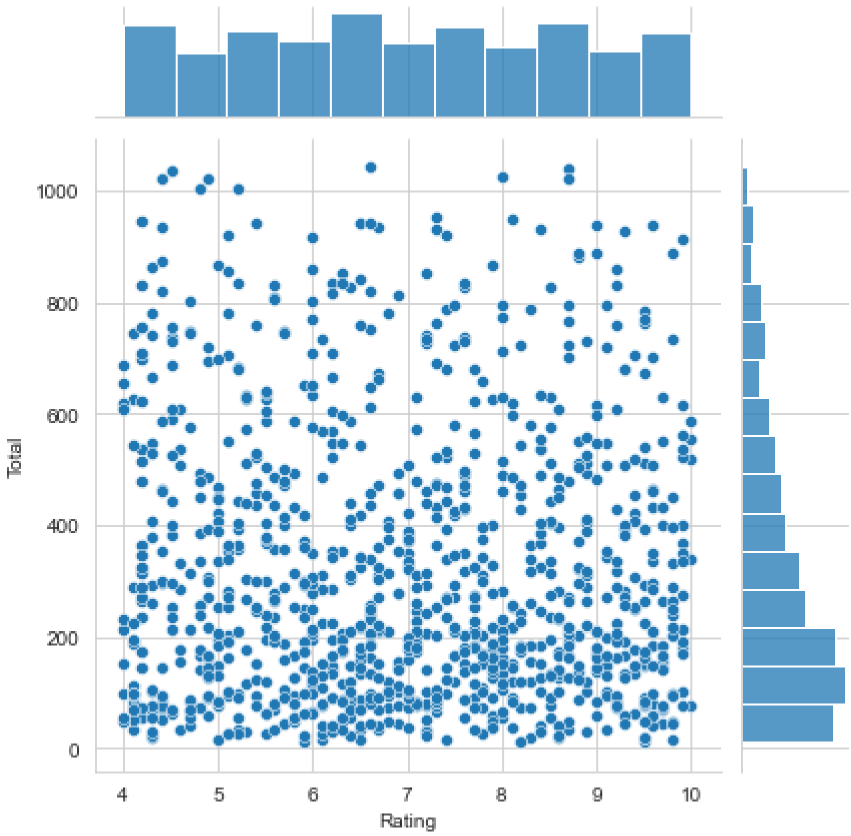


Jointplot : it displays a relationship between 2 variables (bivariate) as well as 1D profiles (univariate) in the margins. This plot is a convenience class that wraps JointGrid.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
In [98]: sns.jointplot(x='Rating', y= 'Total', data=data)
```

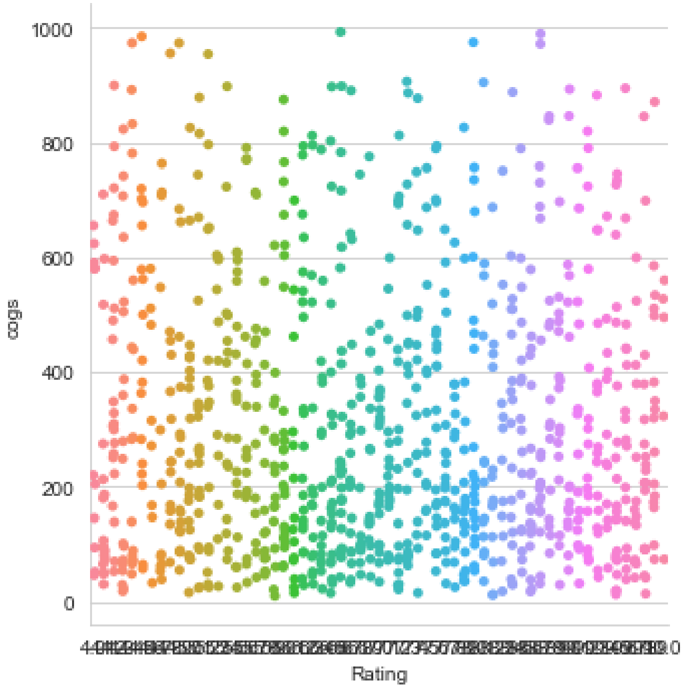
```
Out[98]: <seaborn.axisgrid.JointGrid at 0x13c5e490490>
```



Catplot : Catplot is a relatively new addition to Seaborn that simplifies plotting that involves categorical variables

```
In [99]: sns.catplot(x='Rating', y= 'cogs', data=data)
```

```
Out[99]: <seaborn.axisgrid.FacetGrid at 0x13c5e5830a0>
```



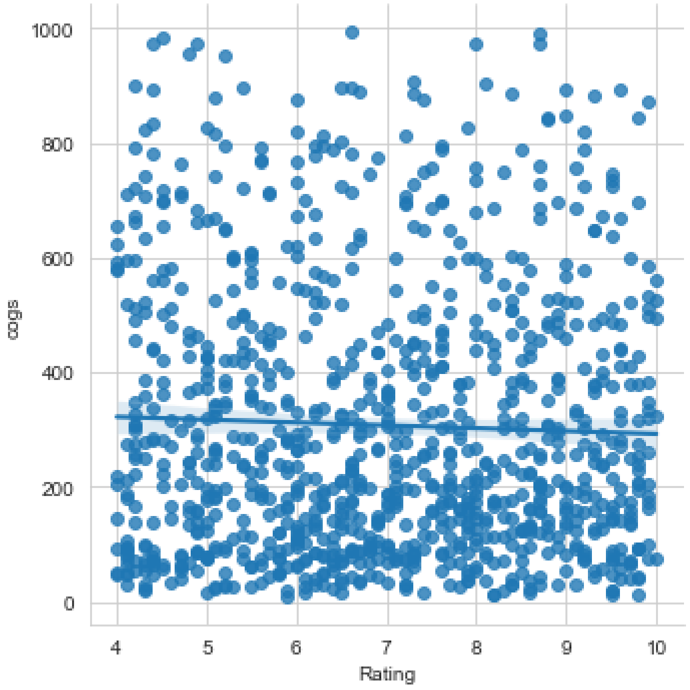
LMPLOT : It shows a line on a 2 dimensional plane. You can plot it with seaborn or matlotlib depending on your preference. The examples below use seaborn to create the plots, but matplotlib to show.

In [100…

```
sns.lmplot(x='Rating', y= 'cogs', data=data)
```

Out[100…

```
<seaborn.axisgrid.FacetGrid at 0x13c5e5df040>
```



KDE PLOT (DENSITY PLOT) : KDE Plot

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

described as Kernel Density Estimate is used for visualizing the Probability Density of a continuous variable. It depicts the probability density at different values in a continuous variable. We can also plot a single graph for multiple samples which helps in more efficient data visualization.

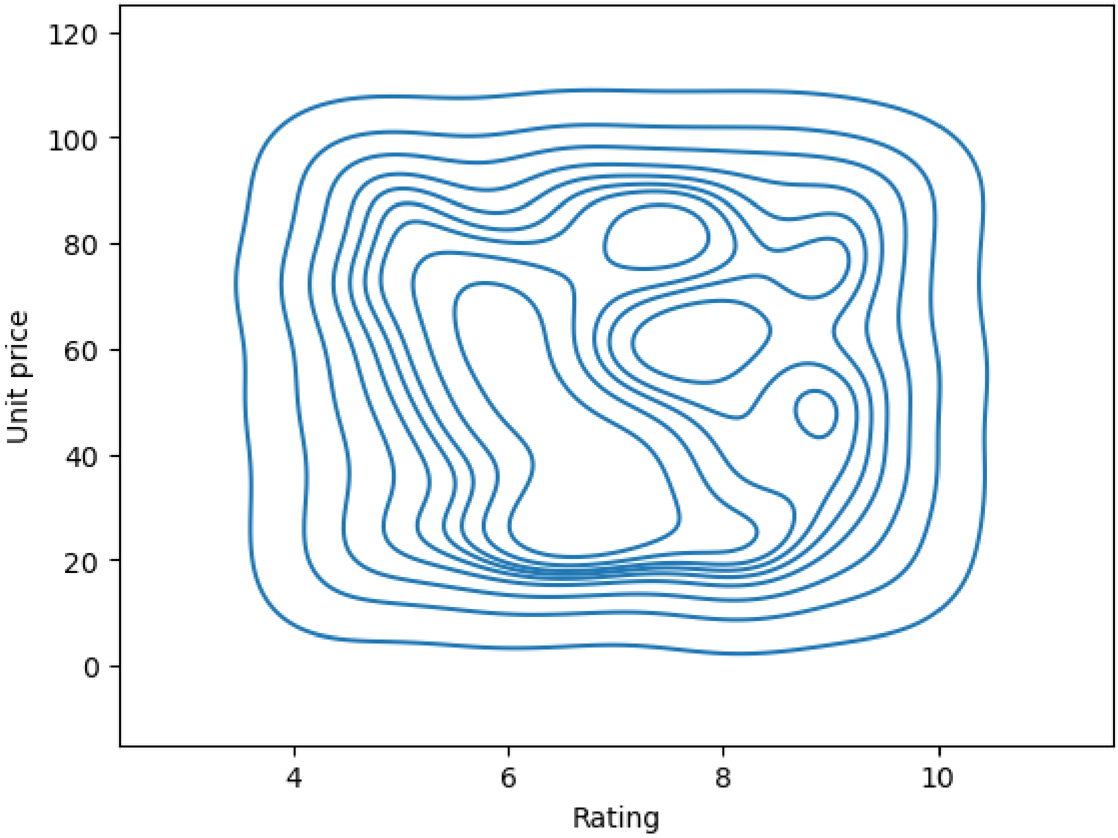
In [101…

```
plt.style.use("default")

sns.kdeplot(x='Rating', y= 'Unit price', data=data)
```

Out[101…

```
<AxesSubplot:xlabel='Rating', ylabel='Unit price'>
```



LINEPLOT : It as a graph that displays data as points or check marks above a number line, showing the frequency of each value.

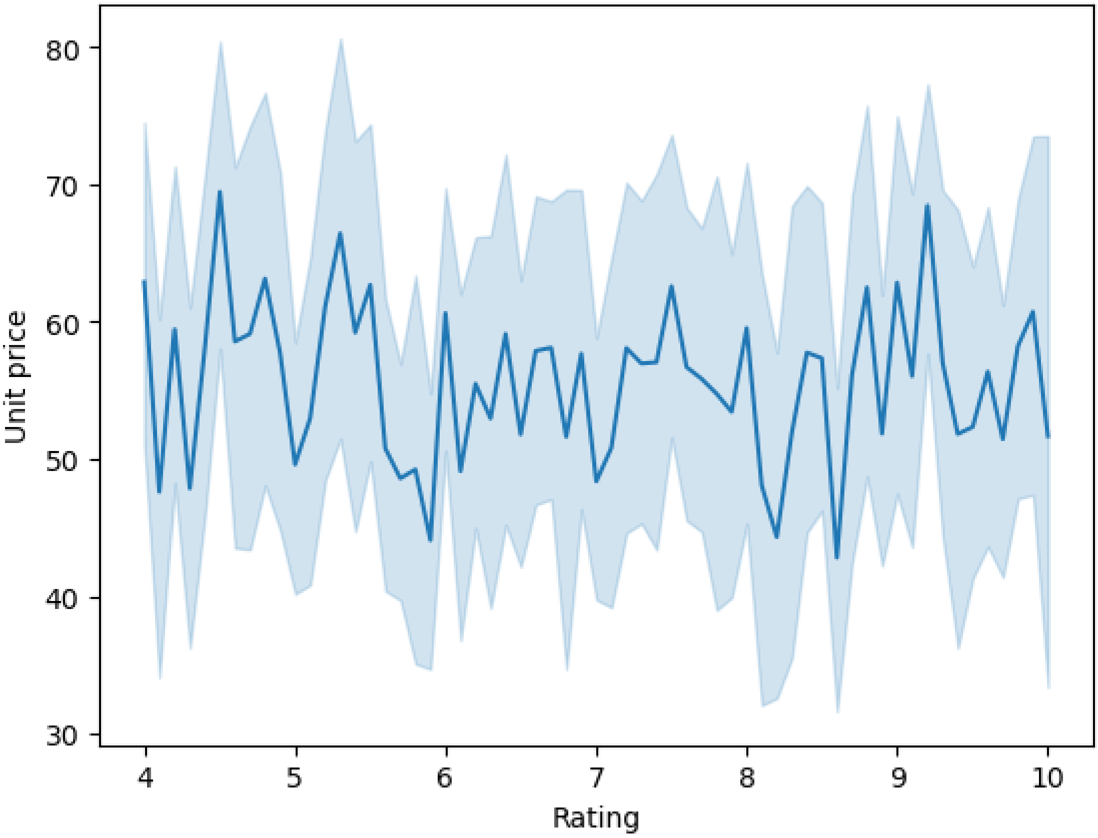
In [102…

```
sns.lineplot(x='Rating', y= 'Unit price', data=data)
```

Out[102…

```
<AxesSubplot:xlabel='Rating', ylabel='Unit price'>
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

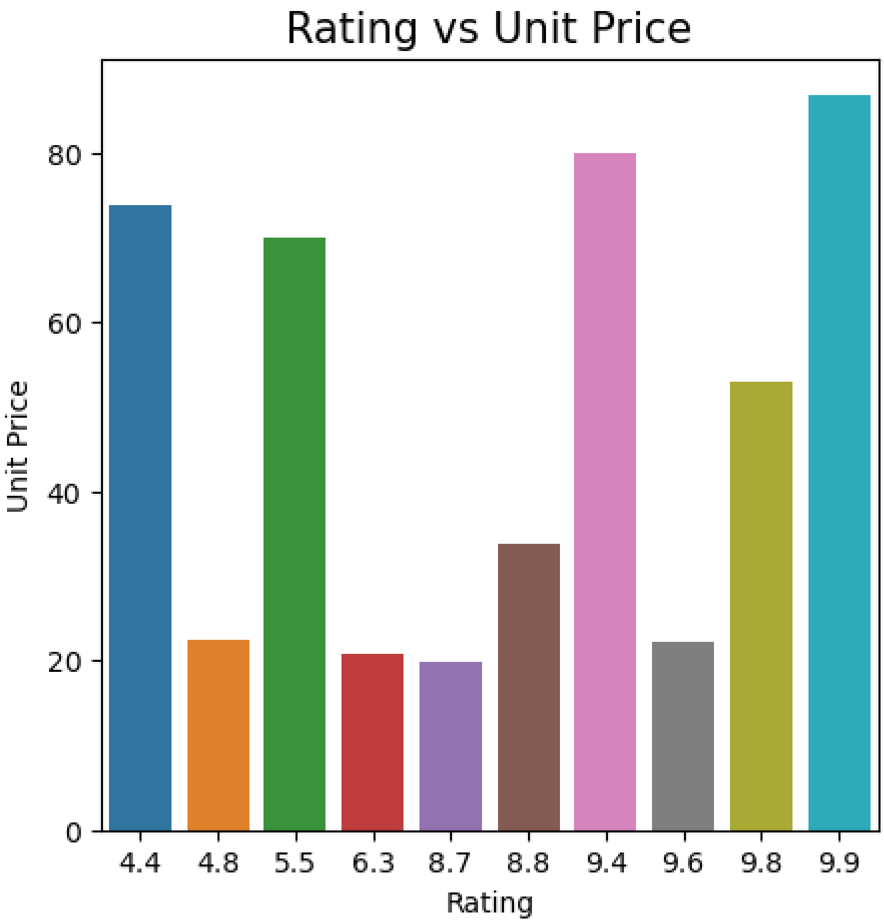


BARPLOT : It shows the relationship between a numeric and a categoric variable. Each entity of the categoric variable is represented as a bar. The size of the bar represents its numeric value.

In [103…

```
plt.style.use("default")
plt.figure(figsize=(5,5))
sns.barplot(x="Rating", y="Unit price", data=data[170:180])
plt.title("Rating vs Unit Price",fontsize=15)
plt.xlabel("Rating")
plt.ylabel("Unit Price")
plt.show()
```

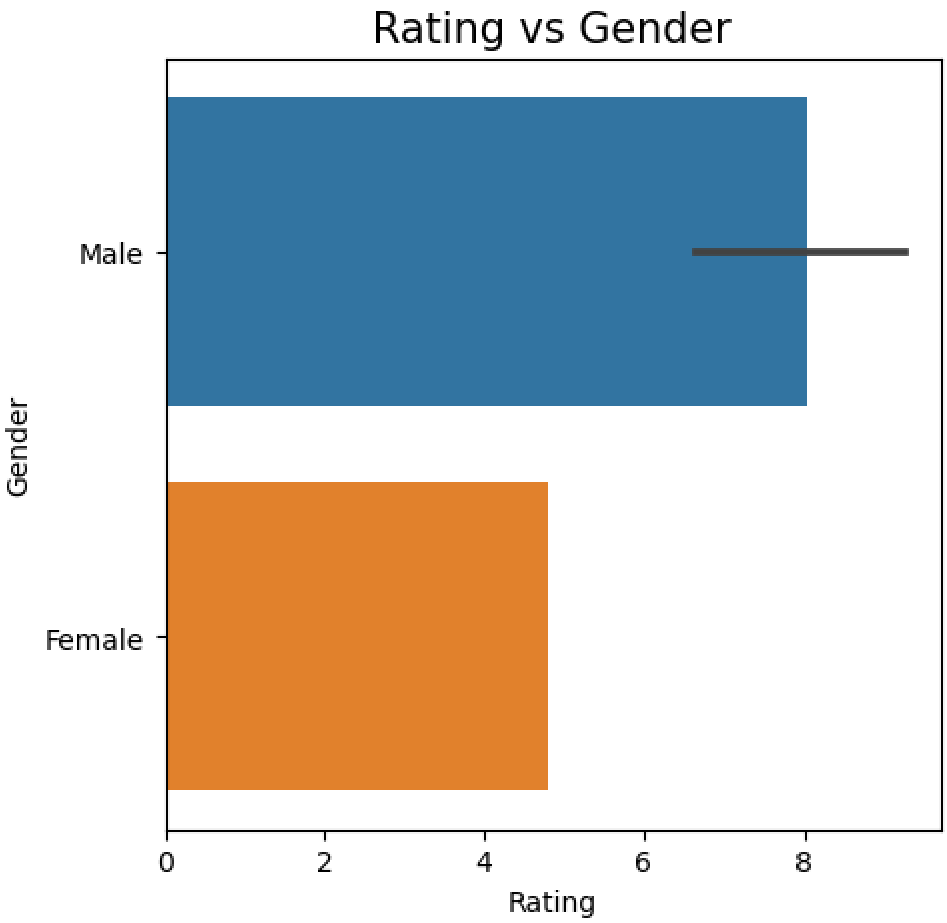
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [104…

```
plt.style.use("default")
plt.figure(figsize=(5,5))
sns.barplot(x="Rating", y="Gender", data=data[170:180])
plt.title("Rating vs Gender",fontsize=15)
plt.xlabel("Rating")
plt.ylabel("Gender")
plt.show()
```

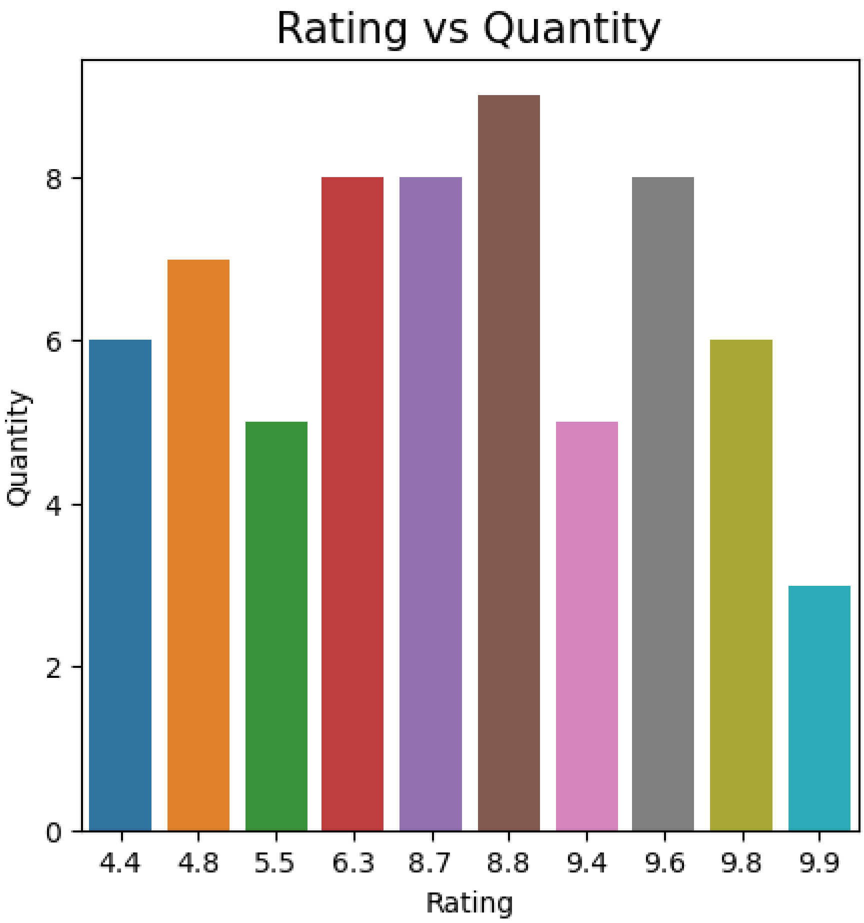
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [105…

```
plt.style.use("default")
plt.figure(figsize=(5,5))
sns.barplot(x="Rating", y="Quantity", data=data[170:180])
plt.title("Rating vs Quantity",fontsize=15)
plt.xlabel("Rating")
plt.ylabel("Quantity")
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



finding categorialfeatures

In [106…

```
list_1=list(data.columns)
```

In [107…

```
list_cate=[]
for i in list_1:
    if data[i].dtype=='object':
        list_cate.append(i)
```

In [108…

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [109…

```
for i in list_cate:
    data[i]=le.fit_transform(data[i])
```

In [110…

```
y=data['Gender']
x=data.drop('Gender',axis=1)
```

Training and testing the dataset

In [111…

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0,test_size=0.2)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [112…

```
print(len(x_train))
print(len(x_test))
print(len(y_train))
print(len(y_test))
```

```
800
200
800
200
```

Models

KNeighborsClassifier

In [113…

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=7)

knn.fit(x_train,y_train)
```

Out[113…

```
KNeighborsClassifier(n_neighbors=7)
```

In [114…

```
y_pred=knn.predict(x_test)
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
print("Classification Report is:\n",classification_report(y_test,y_pred))
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
print("Training Score:\n",knn.score(x_train,y_train)*100)
```

```
Classification Report is:
               precision    recall  f1-score   support

           0       0.47      0.49      0.48       100
           1       0.47      0.45      0.46       100

    accuracy                           0.47       200
   macro avg       0.47      0.47      0.47       200
weighted avg       0.47      0.47      0.47       200

Confusion Matrix:
 [[49 51]
 [55 45]]
Training Score:
 64.75
```

SVC

Support Vector Systems

In [115…

```
from sklearn.svm import SVC

svc = SVC()
svc.fit(x_train, y_train)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
SVC()
```

Out[115…

In [116…

```
y_pred=svc.predict(x_test)
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
print("Classification Report is:\n",classification_report(y_test,y_pred)
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
print("Training Score:\n",svc.score(x_train,y_train)*100)
```

```
  File "C:\Users\DARISA~1\AppData\Local\Temp/ipykernel_900/1704706032.py", line 6
    print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
    ^
SyntaxError: invalid syntax
```

Naive Bayes

In [ ]:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train,y_train)
```

In [ ]:

```
y_pred=gnb.predict(x_test)
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
print("Classification Report is:\n",classification_report(y_test,y_pred))
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
print("Training Score:\n",gnb.score(x_train,y_train)*100)
```

DECISION TREE CLASSIFIER

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(max_depth=6, random_state=123,criterion='entropy')

dtree.fit(x_train,y_train)
```

In [ ]:

```
y_pred=dtree.predict(x_test)
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
print("Classification Report is:\n",classification_report(y_test,y_pred))
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
print("Training Score:\n",dtree.score(x_train,y_train)*100)
```

Random Forest Classifier

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
y_pred=rfc.predict(x_test)
```

```
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
print("Classification Report is:\n",classification_report(y_test,y_pred))
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
print("Training Score:\n",rfc.score(x_train,y_train)*100)
```

AdaBoostClassifier

In [ ]:

```
from sklearn.ensemble import AdaBoostClassifier
adb = AdaBoostClassifier(base_estimator = None)
adb.fit(x_train,y_train)
```

In [ ]:

```
y_pred=adb.predict(x_test)
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
print("Classification Report is:\n",classification_report(y_test,y_pred))
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
print("Training Score:\n",adb.score(x_train,y_train)*100)
```

Gradient Boosting Classifier

In [ ]:

```
from sklearn.ensemble import GradientBoostingClassifier
gbc=GradientBoostingClassifier()
gbc.fit(x_train,y_train)
```

In [ ]:

```
y_pred=gbc.predict(x_test)
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
print("Classification Report is:\n",classification_report(y_test,y_pred))
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
print("Training Score:\n",gbc.score(x_train,y_train)*100)
```

In [ ]:

```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
data
```

XGBClassifier

In [ ]:

```
from xgboost import XGBClassifier

xgb =XGBClassifier(objective ='reg:linear', colsample_bytree = 0.3, learning_rate =
                max_depth = 5, alpha = 10, n_estimators = 10)

xgb.fit(x_train, y_train)
```

In [ ]:

```
y_pred=xgb.predict(x_test)
                                          _score,classification_report,confusion_matrix
from sklearn.metrics import r2_score
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
from sklearn.metrics import mean_squared_error
print("Classification Report is:\n",classification_report(y_test,y_pred))
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
print("Training Score:\n",xgb.score(x_train,y_train)*100)
```

ExtraTreesClassifier

In [ ]:

```
from sklearn.ensemble import ExtraTreesClassifier
etc = ExtraTreesClassifier(n_estimators=100, random_state=0)
etc.fit(x_train,y_train)
```

In [ ]:

```
y_pred=etc.predict(x_test)
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
print("Classification Report is:\n",classification_report(y_test,y_pred))
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
print("Training Score:\n",etc.score(x_train,y_train)*100)
```

Bagging Classifier

In [ ]:

```
from sklearn.ensemble import BaggingClassifier
from sklearn import tree
model = BaggingClassifier(tree.DecisionTreeClassifier(random_state=1))
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

In [ ]:

```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
data
```

We got a good accuracy of about 100 % using Random Forest Classifier and Extra Trees Classifier which is quite well for the given dataset.

Thank You

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js