Sales Prediction

The growth of supermarkets in most populated cities are increasing and market competitions are also high. The dataset is one of the historical sales of supermarket company.

IMPORTING THE LIBRARIES

In [1]:
```
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
warnings.filterwarnings("ignore")
%matplotlib inline
```

LOADING THE DATASET

In [2]:
```
data = pd.read_csv(r"D:/Supermarket Sales Prediction.csv")
```

In [3]:
```
data
```

Out[3]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	To
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.97
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.22
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.52
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.04
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.37
...	...	...	...	...	...	...	...	...	...	
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	2.0175	42.36
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	48.6900	1022.49
997	727-02-1313	A	Yangon	Member	Male	Food and beverages	31.84	1	1.5920	33.43
998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	65.82	1	3.2910	69.11
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7	30.9190	649.29

1000 rows × 17 columns

In [4]:
```
data.head()
```

Out[4]:

	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
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785

In [5]:
```
data.describe()
```

Out[5]:

	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
min	10.080000	1.000000	0.508500	10.678500	10.17000	4.761905e+00	0.508500
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

In [6]:
```
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
```

In [7]:
```
data.value_counts()
```

Out[7]:
```
Invoice ID   Branch  City       Customer type  Gender  Product line         Unit p
rice  Quantity  Tax 5%   Total     Date       Time   Payment      cogs    gross ma
rgin percentage  gross income  Rating
101-17-6199  A       Yangon     Normal         Male    Food and beverages   45.79
7         16.0265  336.5565  3/13/2019  19:44  Credit card  320.53  4.761905
16.0265       7.0       1
641-62-7288  B       Mandalay   Normal         Male    Home and lifestyle   99.92
6         29.9760  629.4960  3/24/2019  13:33  Ewallet      599.52  4.761905
29.9760       7.1       1
633-91-1052  A       Yangon     Normal         Female  Home and lifestyle   12.03
2         1.2030   25.2630   1/27/2019  15:51  Cash         24.06   4.761905
1.2030        5.1       1
634-97-8956  A       Yangon     Normal         Male    Food and beverages   32.90
3         4.9350   103.6350  2/17/2019  17:27  Credit card  98.70   4.761905
4.9350        9.1       1
635-28-5728  A       Yangon     Normal         Male    Health and beauty    56.00
3         8.4000   176.4000  2/28/2019  19:33  Ewallet      168.00  4.761905
8.4000        4.8       1


..
373-14-0504  A       Yangon     Member         Female  Sports and travel    71.63
2         7.1630   150.4230  2/12/2019  14:33  Ewallet      143.26  4.761905
7.1630        8.8       1
373-73-7910  A       Yangon     Normal         Male    Sports and travel    86.31
7         30.2085  634.3785  2/8/2019   10:37  Ewallet      604.17  4.761905
30.2085       5.3       1
373-88-1424  C       Naypyitaw  Member         Male    Home and lifestyle   35.81
5         8.9525   188.0025  2/6/2019   18:44  Ewallet      179.05  4.761905
8.9525        7.9       1
374-17-3652  B       Mandalay   Member         Female  Food and beverages   42.82
9         19.2690  404.6490  2/5/2019   15:26  Credit card  385.38  4.761905
19.2690       8.9       1
898-04-2717  A       Yangon     Normal         Male    Fashion accessories  76.40
9         34.3800  721.9800  3/19/2019  15:49  Ewallet      687.60  4.761905
34.3800       7.5       1
Length: 1000, dtype: int64
```

In [8]:
```
data.shape
```

Out[8]:
```
(1000, 17)
```

In [9]:
```
data.dtypes
```

Out[9]:
```
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
```

In [10]:
```
data.columns
```

Out[10]:
```
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')
```

Checking Null Value

In [11]:
```
data.isnull().sum()
```

Out[11]:
```
Invoice ID                0
Branch                    0
City                      0
Customer type             0
Gender                    0
Product line              0
Unit price                0
Quantity                  0
Tax 5%                    0
Total                     0
Date                      0
Time                      0
Payment                   0
cogs                      0
gross margin percentage   0
gross income              0
Rating                    0
dtype: int64
```

In [12]:
```
data.isnull().any()
```

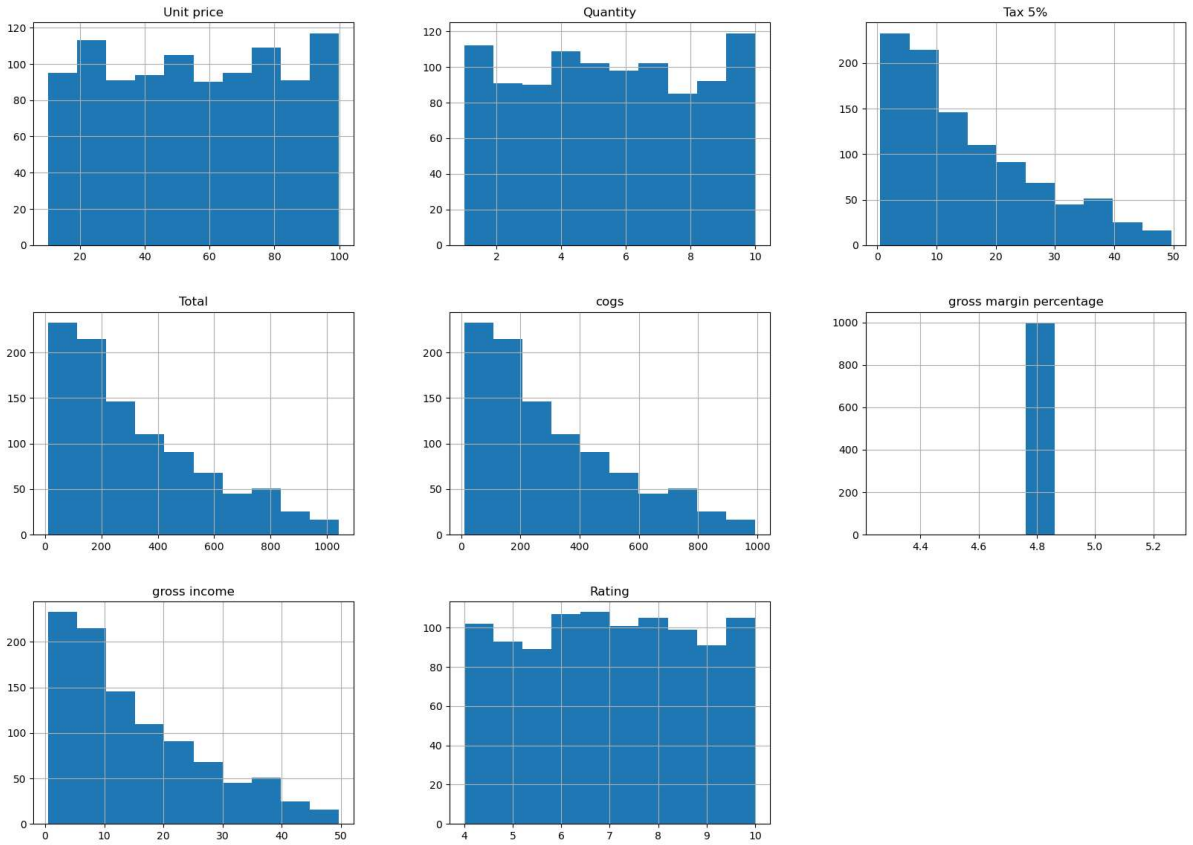
Out[12]:
```
Invoice ID                False
Branch                    False
City                      False
Customer type             False
Gender                    False
Product line              False
Unit price                False
Quantity                  False
Tax 5%                    False
Total                     False
Date                      False
Time                      False
Payment                   False
cogs                      False
gross margin percentage   False
gross income              False
Rating                    False
dtype: bool
```

Exploratory Data Analysis

HISTOGRAM

A histogram is basically used to represent data provided in a form of some groups.It is accurate method for the graphical representation of numerical data distribution.It is a type of bar plot where X-axis represents the bin ranges while Y-axis gives information about frequency.

In [13]:
```
data.hist(figsize=(20,14))
plt.show()
```



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

Out[14]:

	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	-0.00877
Quantity	0.010778	1.000000	0.705510	0.705510	0.705510	NaN	0.705510	-0.01581
Tax 5%	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.03644
Total	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.03644
cogs	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000	-0.03644
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	-0.03644
Rating	-0.008778	-0.015815	-0.036442	-0.036442	-0.036442	NaN	-0.036442	1.00000

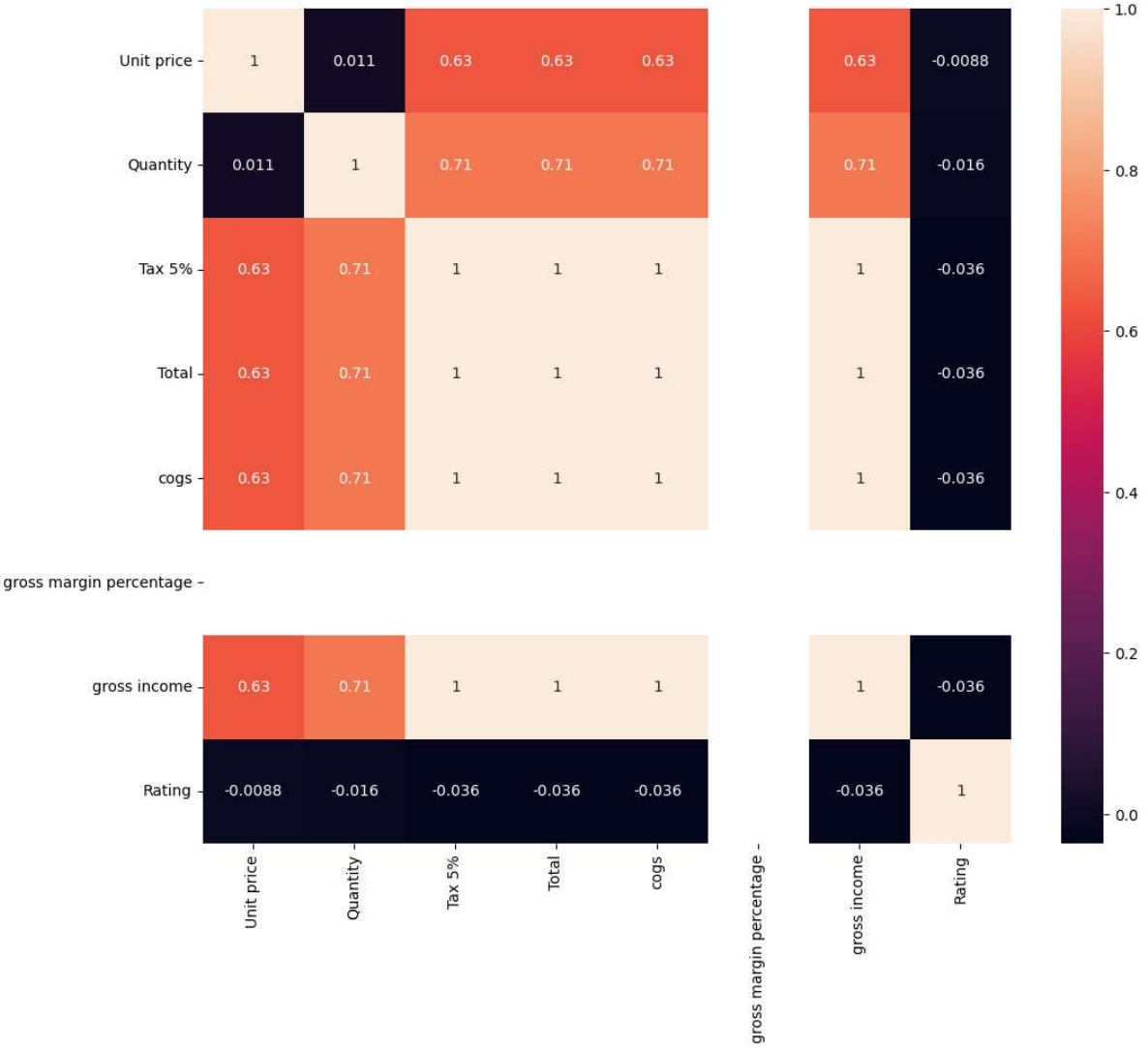
HEATMAP

A heatmap is a graphical representation of data that uses a system of color-coding to represent different values. Heatmaps are used in various forms of analytics but are most commonly used to show user behaviour on specific webpages or webpage templates.

In [15]:
```
plt.figure(figsize = (12,10))

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

Out[15]:
```
<Axes: >
```



In [16]:
```
data.columns
```

Out[16]:
```
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')
```

BOXPLOT

A boxplot 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”). ... It can also tell you if your data is symmetrical, how tightly your data is grouped, and if and how your data is skewed.

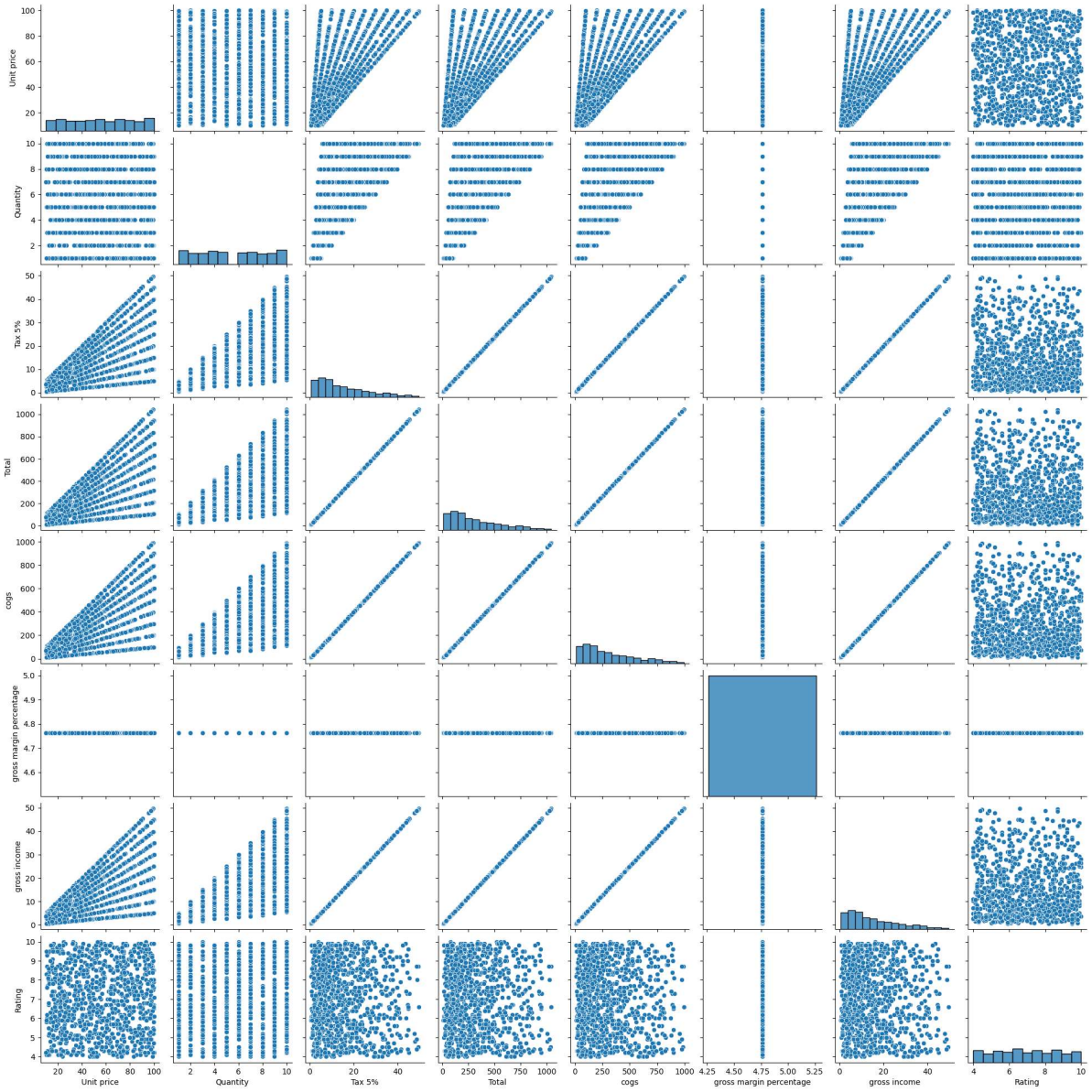
plt.figure(figsize=(14,10)) sns.set_style(style='whitegrid') plt.subplot(2,3,1) sns.boxplot(x='Unit price',data=data) plt.subplot(2,3,2) sns.boxplot(x='Quantity',data=data) plt.subplot(2,3,3) sns.boxplot(x='Total',data=data) plt.subplot(2,3,4) sns.boxplot(x='cogs',data=data) plt.subplot(2,3,5) sns.boxplot(x='Rating',data=data)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

PAIRPLOT

A pairplot plot a pairwise relationships in a dataset. The pairplot function creates a grid of Axes such that each variable in data will by shared in the y-axis across a single row and in the x-axis across a single column.

In [17]:
```
sns.pairplot(data=data)
```

Out[17]:
```
<seaborn.axisgrid.PairGrid at 0x2e615663dc0>
```

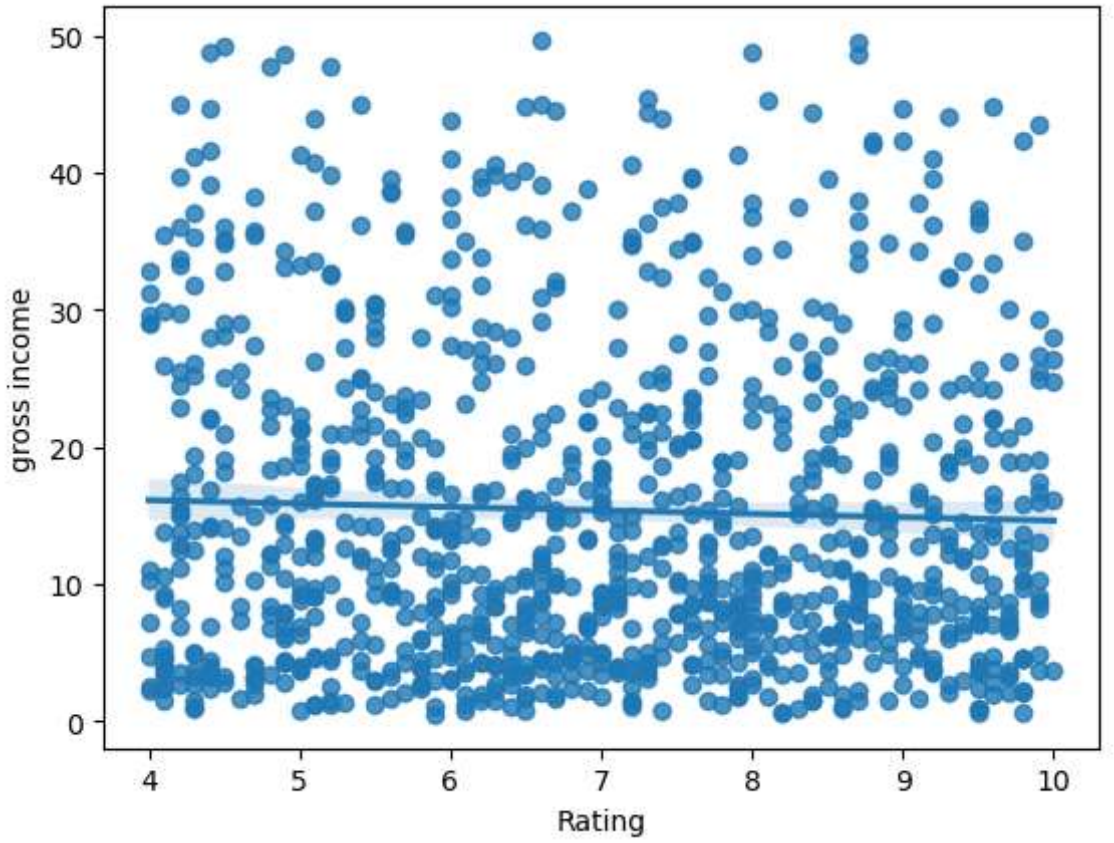


REGPLOT

This method is used to plot data and a linear regression model fit. ... If strings, these should correspond with column names in “data”. When pandas objects are used, axes will be labeled with the series name. data: This is dataframe where each column is a variable and each row is an observation.

In [18]:
```
sns.regplot(x='Rating', y= 'gross income', data=data)
```

Out[18]:
```
<Axes: xlabel='Rating', ylabel='gross income'>
```

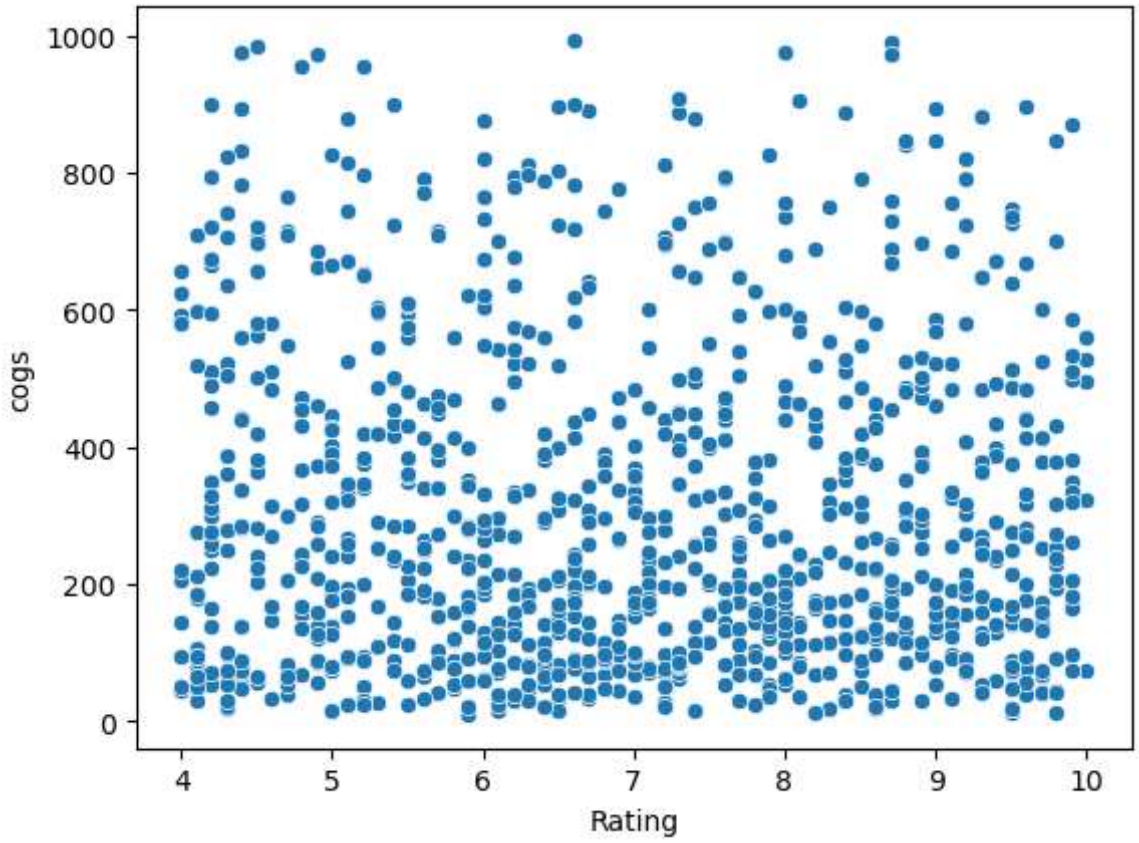


SCATTER PLOT

A scatter plot (aka scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables.

In [19]:
```
sns.scatterplot(x='Rating', y= 'cogs', data=data)
```

Out[19]:
```
<Axes: xlabel='Rating', ylabel='cogs'>
```

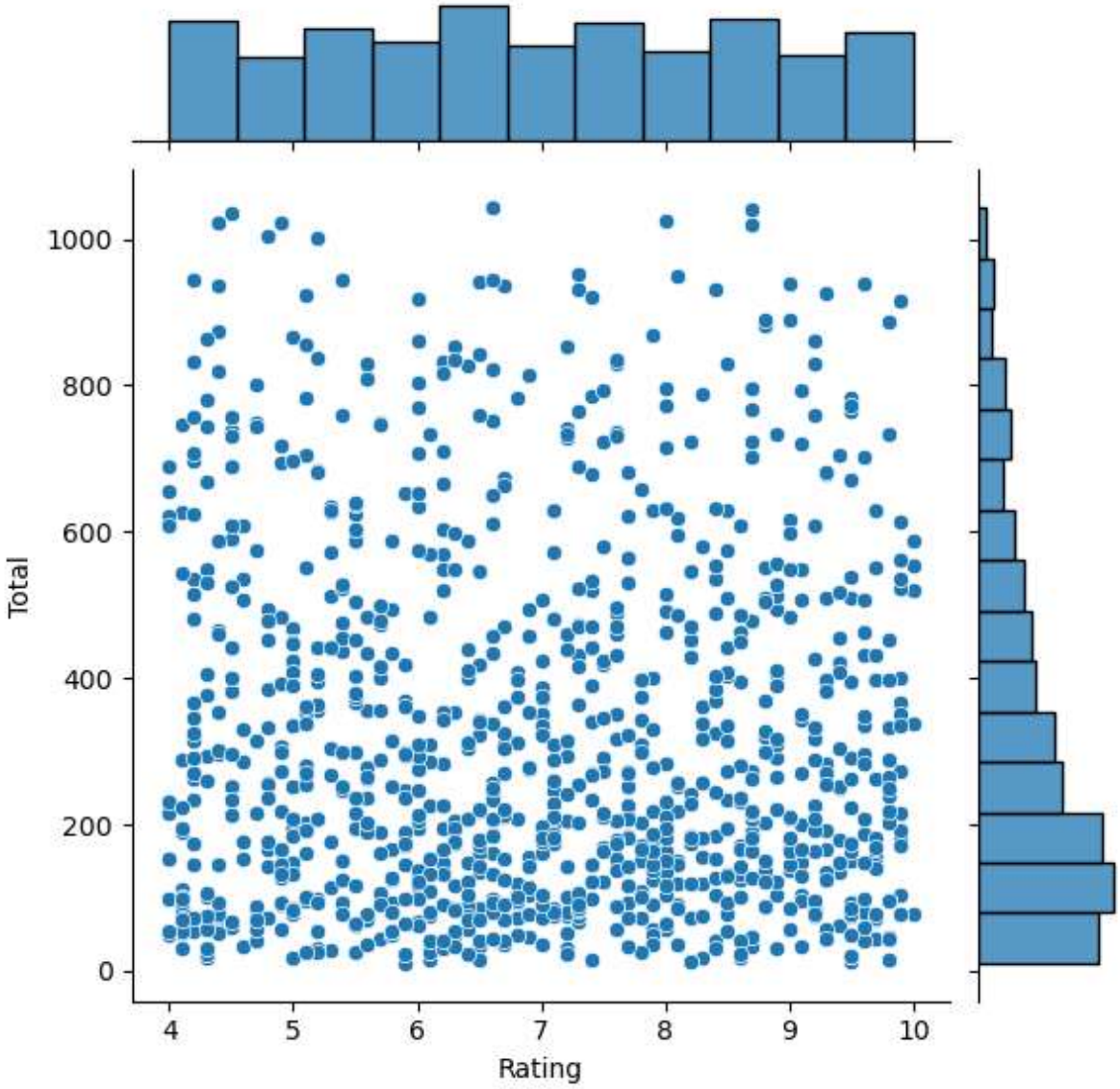


JOINTPLOT

Seaborn's jointplot 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.

In [20]:
```
sns.jointplot(x='Rating', y= 'Total', data=data)
```

Out[20]:
```
<seaborn.axisgrid.JointGrid at 0x2e61a04a5f0>
```

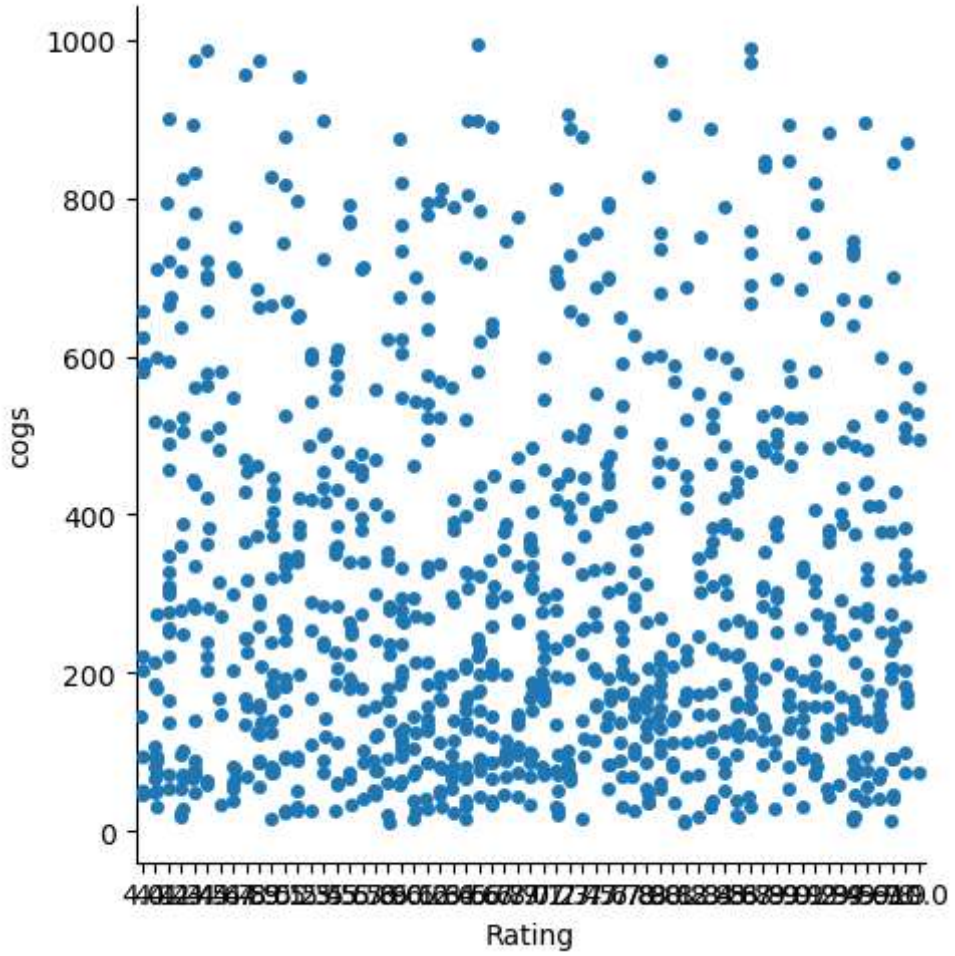


CATPLOT

Catplot is a relatively new addition to Seaborn that simplifies plotting that involves categorical variables. In Seaborn version v0. 9.0 that came out in July 2018, changed the older factor plot to catplot to make it more consistent with terminology in pandas and in seaborn.

In [21]:
```
sns.catplot(x='Rating', y= 'cogs', data=data)
```

Out[21]:
```
<seaborn.axisgrid.FacetGrid at 0x2e61a103370>
```

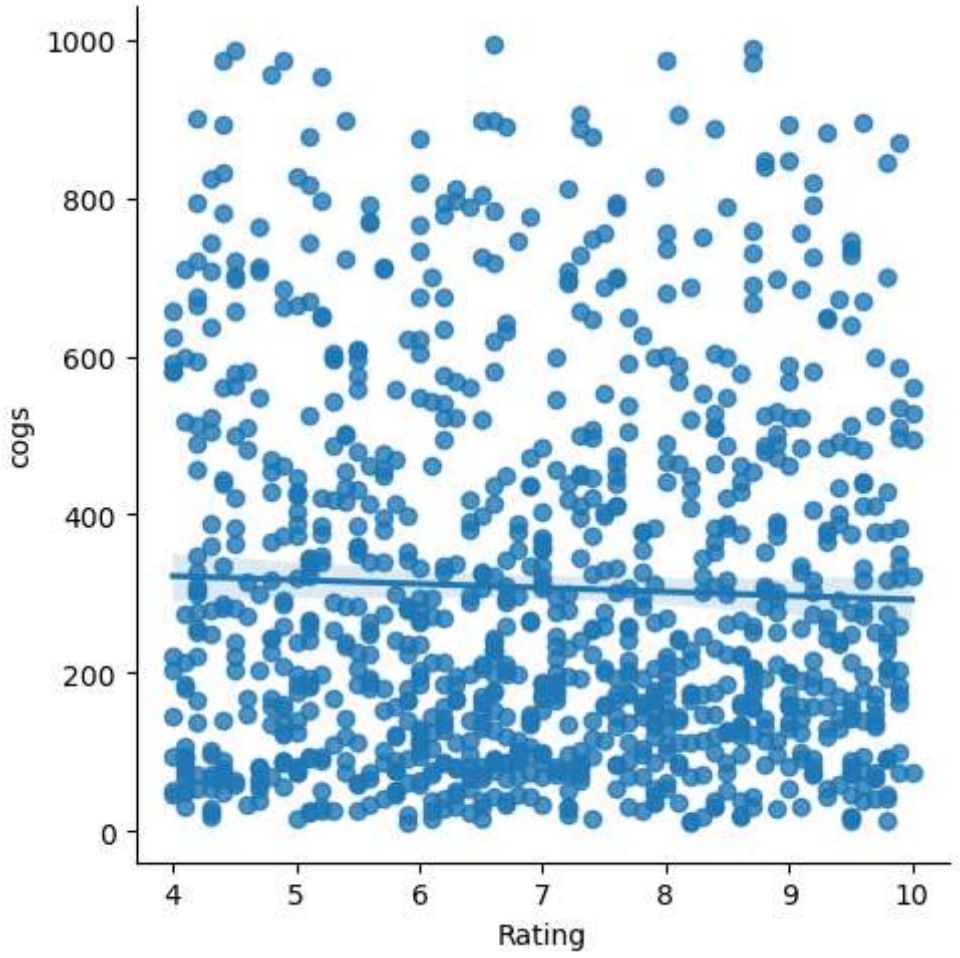


LMPLOT

The lineplot (lmplot) is one of the most basic plots. 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 [22]:
```
sns.lmplot(x='Rating', y= 'cogs', data=data)
```

Out[22]:
```
<seaborn.axisgrid.FacetGrid at 0x2e61a14f640>
```



In [23]:
```
data.columns
```

Out[23]:
```
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')
```

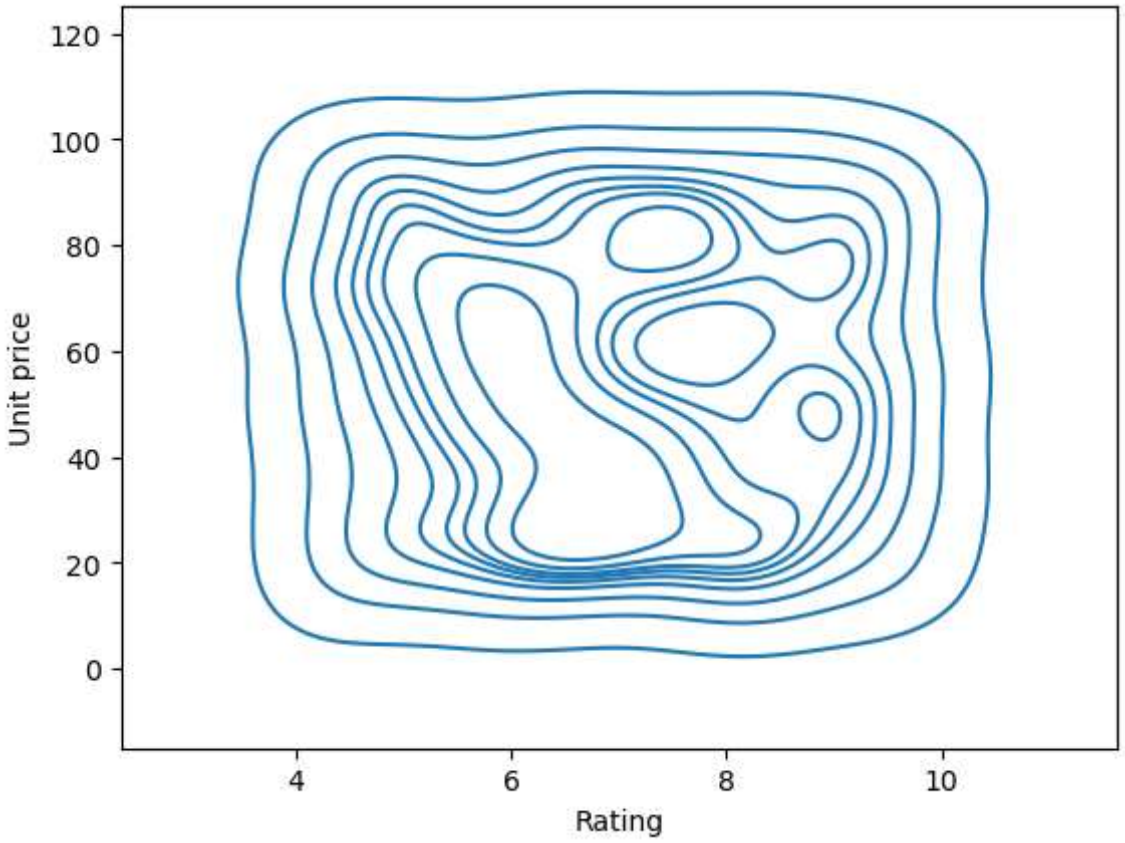
KDE PLOT (DENSITY PLOT)

KDE Plot 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 [24]:
```
plt.style.use("default")

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

Out[24]:
```
<Axes: xlabel='Rating', ylabel='Unit price'>
```

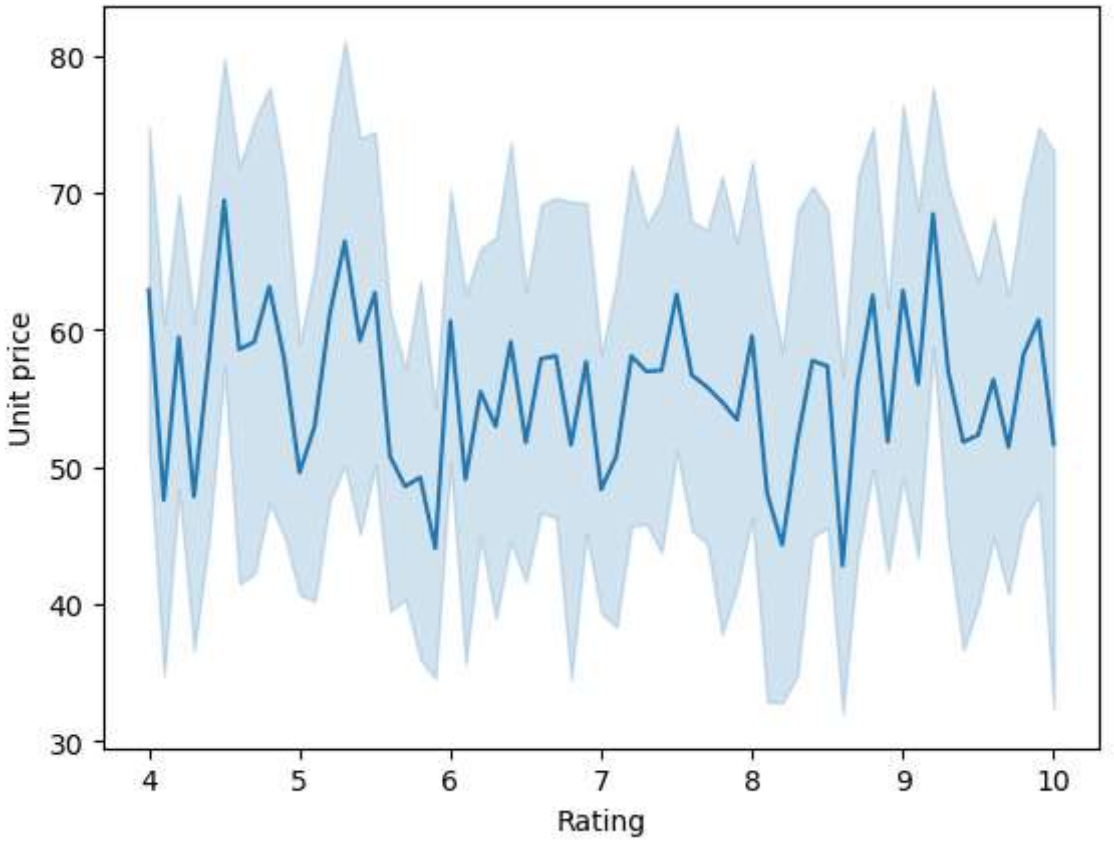


LINEPLOT

A Line plot can be defined as a graph that displays data as points or check marks above a number line, showing the frequency of each value.

In [25]:
```
sns.lineplot(x='Rating', y= 'Unit price', data=data)
```

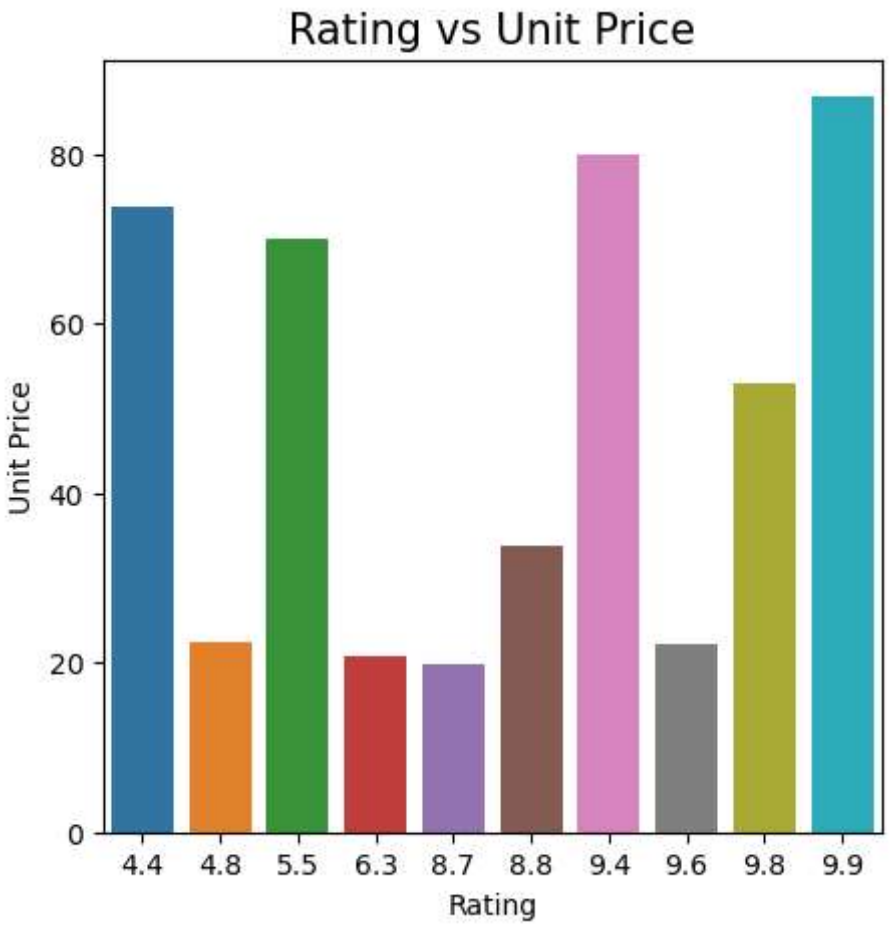
Out[25]:
```
<Axes: xlabel='Rating', ylabel='Unit price'>
```



BARPLOT

A barplot (or barchart) is one of the most common types of graphic. 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.

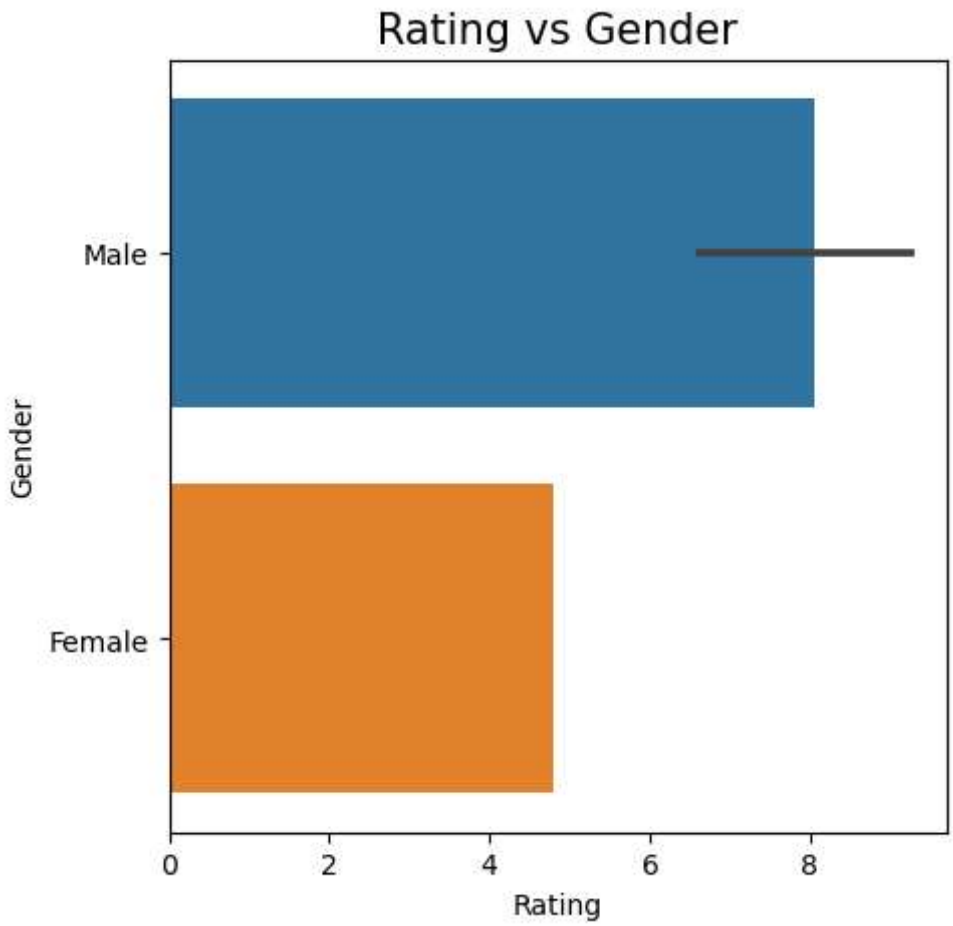
In [26]:
```
plt.style.use("default")
plt.figure(figsize=(5,5))
sns.barplot(x="Rating", y="Unit price", data=data[170:180])
plt.title("Rating vs Unit Price",fontsize=15)
plt.xlabel("Rating")
plt.ylabel("Unit Price")
plt.show()
```



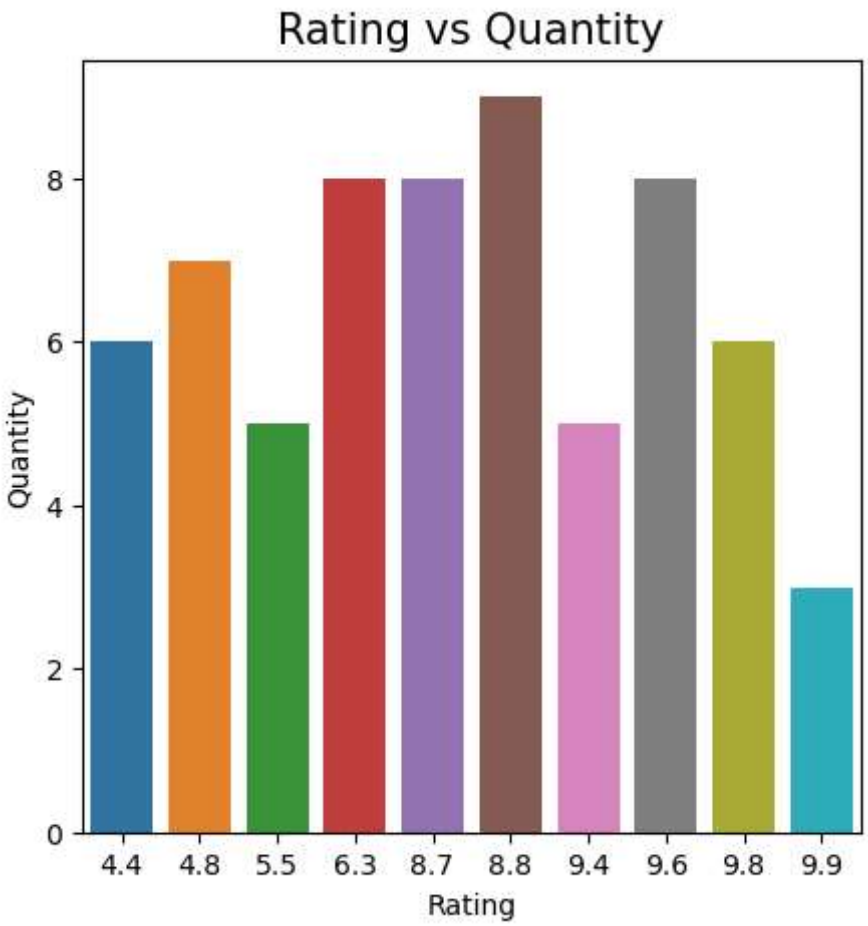
In [27]:
```
data.columns
```

Out[27]:
```
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')
```

In [28]:
```
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()
```



In [29]:
```
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()
```



In [30]:
```
#lets find the categorialfeatures
list_1=list(data.columns)
```

In [31]:
```
list_cate=[]
for i in list_1:
    if data[i].dtype=='object':
        list_cate.append(i)
```

In [32]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [33]:
```
for i in list_cate:
    data[i]=le.fit_transform(data[i])
```

In [34]:
```
data
```

Out[34]:

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Dat
0	814	0	2	0	0	3	74.69	7	26.1415	548.9715	2
1	142	2	1	1	0	0	15.28	5	3.8200	80.2200	8
2	653	0	2	1	1	4	46.33	7	16.2155	340.5255	8
3	18	0	2	0	1	3	58.22	8	23.2880	489.0480	1
4	339	0	2	1	1	5	86.31	7	30.2085	634.3785	5
...	...	...	...	...	...	...	...	...	...	...	.
995	153	2	1	1	1	3	40.35	1	2.0175	42.3675	2
996	250	1	0	1	0	4	97.38	10	48.6900	1022.4900	7
997	767	0	2	0	1	2	31.84	1	1.5920	33.4320	5
998	308	0	2	1	1	4	65.82	1	3.2910	69.1110	4
999	935	0	2	0	0	1	88.34	7	30.9190	649.2990	4

1000 rows × 17 columns

In [35]:
```
y=data['Gender']
x=data.drop('Gender',axis=1)
```

TRAINING AND TESTING DATA

In [36]:
```
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)
```

In [37]:
```
print(len(x_train))
print(len(x_test))
print(len(y_train))
print(len(y_test))
```

```
800
200
800
200
```

MODELS

1. KNeighborsClassifier

By default, the KNeighborsClassifier looks for the 5 nearest neighbors. We must explicitly tell the classifier to use Euclidean distance for determining the proximity between neighboring points.

In [38]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=7)

knn.fit(x_train,y_train)
```

Out[38]:
```
KNeighborsClassifier(n_neighbors=7)
```

In [39]:
```
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
```

2. SVC

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis.

In [40]:
```
from sklearn.svm import SVC

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

Out[40]:
```
SVC()
```

In [41]:
```
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)
```

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

           0       0.45      0.49      0.47       100
           1       0.44      0.40      0.42       100

    accuracy                           0.45       200
   macro avg       0.44      0.45      0.44       200
weighted avg       0.44      0.45      0.44       200

Confusion Matrix:
 [[49 51]
 [60 40]]
Training Score:
 55.50000000000001
```

3. Naive Bayes

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

In [42]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train,y_train)
```

Out[42]:
```
▾ GaussianNB
GaussianNB()
```

In [43]:
```
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)
```

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

           0       0.51      0.35      0.41       100
           1       0.50      0.66      0.57       100

    accuracy                           0.51       200
   macro avg       0.51      0.51      0.49       200
weighted avg       0.51      0.51      0.49       200

Confusion Matrix:
 [[35 65]
 [34 66]]
Training Score:
 55.125
```

4. DECISION TREE CLASSIFIER

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. ... The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

In [44]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(max_depth=6, random_state=123,criterion='entropy')

dtree.fit(x_train,y_train)
```

Out[44]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=6, random_state=12
3)
```

In [45]:
```
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)
```

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

           0       0.52      0.79      0.63       100
           1       0.56      0.27      0.36       100

    accuracy                           0.53       200
   macro avg       0.54      0.53      0.50       200
weighted avg       0.54      0.53      0.50       200

Confusion Matrix:
 [[79 21]
 [73 27]]
Training Score:
 63.87500000000001
```

5.Random Forest Classifier

A random forest classifier. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

In [46]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[46]:
```
RandomForestClassifier()
```

In [47]:
```
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)
```

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

           0       0.50      0.54      0.52       100
           1       0.51      0.47      0.49       100

    accuracy                           0.51       200
   macro avg       0.51      0.51      0.50       200
weighted avg       0.51      0.51      0.50       200

Confusion Matrix:
 [[54 46]
 [53 47]]
Training Score:
 100.0
```

In [48]:
```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
data
```

Out[48]:

	Actual	Predicted
993	1	1
859	0	0
298	1	1
553	1	0
672	0	0
...	...	...
679	1	0
722	1	1
215	1	0
653	1	0
150	0	0

200 rows × 2 columns

6. XGBClassifier

XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.

In [49]:
```
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)
```

```
[23:45:08] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-gr
oup-i-0fdc6d574b9c0d168-1\xgboost\xgboost-ci-windows\src\objective\regression_obj.
cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[49]:

```
XGBClassifier(alpha=10, base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.3, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_type
s=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_ty
pe=None,
              interaction_constraints=None, learning_rate=0.1, max_bin
=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
```

In [50]:

```
y_pred=xgb.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",xgb.score(x_train,y_train)*100)
```

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

           0       0.49      0.56      0.52       100
           1       0.48      0.41      0.44       100

    accuracy                           0.48       200
   macro avg       0.48      0.48      0.48       200
weighted avg       0.48      0.48      0.48       200

Confusion Matrix:
 [[56 44]
 [59 41]]
Training Score:
 62.625
```

7. ExtraTreesClassifier

Extremely Randomized Trees Classifier(Extra Trees Classifier) is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a “forest” to output it's classification result.

In [51]:

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

Out[51]:

```
ExtraTreesClassifier(random_state=0)
```

In [52]:

```
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)
```

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

           0       0.50      0.50      0.50       100
           1       0.50      0.50      0.50       100

    accuracy                           0.50       200
   macro avg       0.50      0.50      0.50       200
weighted avg       0.50      0.50      0.50       200

Confusion Matrix:
 [[50 50]
 [50 50]]
Training Score:
 100.0
```

8.Bagging Classifier

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. ... The base estimator to fit on random subsets of the dataset.

In [53]:
```
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)
```

Out[53]:
```
0.515
```

In [54]:
```
data = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
data
```

Out[54]:

	Actual	Predicted
993	1	1
859	0	1
298	1	1
553	1	0
672	0	1
...	...	...
679	1	1
722	1	1
215	1	0
653	1	0
150	0	0

200 rows × 2 columns

CONCLUSION :

ACCURACIES OF DIFFERENT MODELS ARE:

KNeighbors Classifier= 64.75 %

SVC= 55.50 %

Naiye Bayes= 55.10 %

Decision Tree Classifier= 64 %

Random Forest Classifier= 100 %

XGB Classifier= 64 %

Extra Trees Classifier= 100 %

Bagging Classifier = 51 %

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

The accuracy of other models can be increased further by HyperTuning.

THANK YOU!

In [ ]: