```
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
```

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

```
data
```

	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	1/5
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	3/8
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	3/3
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1/27
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2/8
...	...	...	...	...	...	...	...	...	...	...	
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	40.35	1	2.0175	42.3675	1/29
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	97.38	10	48.6900	1022.4900	3/2
997	727-02-1313	A	Yangon	Member	Male	Food and beverages	31.84	1	1.5920	33.4320	2/9
998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	65.82	1	3.2910	69.1110	2/22
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7	30.9190	649.2990	2/18

1000 rows × 17 columns

```
data.head()
```

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	5
	226-31-					Electronic				

```
data.describe()
```

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000	100
mean	55.672130	5.510000	15.379369	322.966749	307.58738	4.761905	15
std	26.494628	2.923431	11.708825	245.885335	234.17651	0.000000	11
min	10.080000	1.000000	0.508500	10.678500	10.17000	4.761905	0
25%	32.875000	3.000000	5.924875	124.422375	118.49750	4.761905	5
50%	55.230000	5.000000	12.088000	253.848000	241.76000	4.761905	12
75%	77.935000	8.000000	22.445250	471.350250	448.90500	4.761905	22
max	99.960000	10.000000	49.650000	1042.650000	993.00000	4.761905	49

```
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
```

```
data.value_counts()
```

```
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
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
```

```
data.shape
```

```
(1000, 17)
```

```
data.dtypes
```

```
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
```

```
data.columns
```

```
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')
```

```
data.isnull().sum()
```

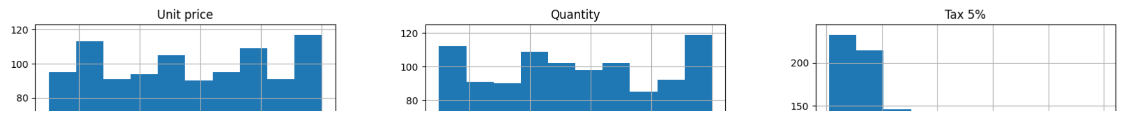
```
Invoice ID                 0
Branch                     0
City                       0
Customer type              0
Gender                     0
```

```
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
```

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

```
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
```

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



```
data.corr()
```

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income
Unit price	1.000000	0.010778	0.633962	0.633962	0.633962	NaN	0.633962
Quantity	0.010778	1.000000	0.705510	0.705510	0.705510	NaN	0.705510
Tax 5%	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000
Total	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000
cogs	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000
gross margin percentage	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gross	0.633962	0.705510	1.000000	1.000000	1.000000	NaN	1.000000

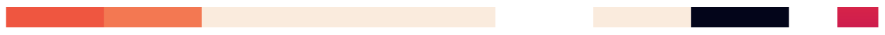
```
plt.figure(figsize = (12,10))

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

```
<Axes: >
```

Unit price
1
0.011
0.63
0.63
0.63
0.63
-0.0088
1.0

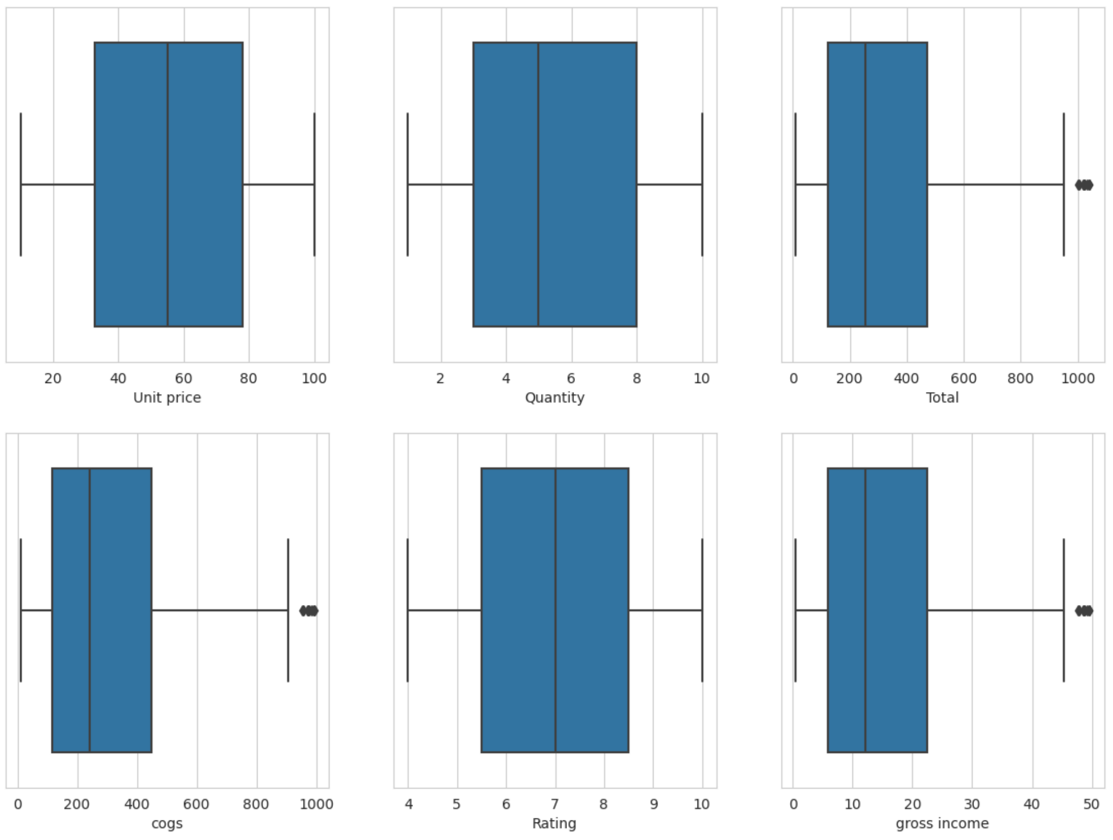
```
data.columns
```

```
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')
```



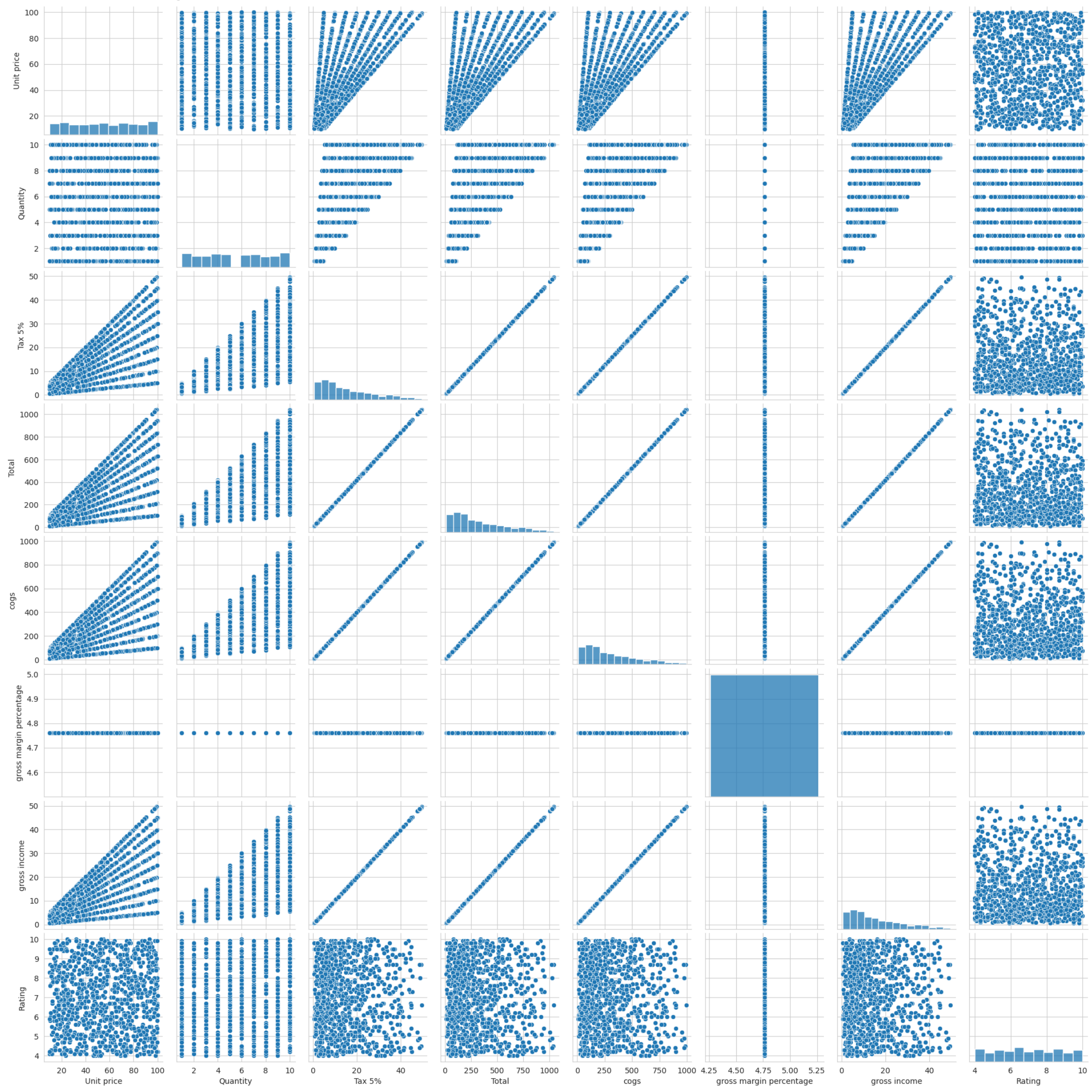
```
plt.figure(figsize=(14,10))
sns.set_style(style='whitegrid')
plt.subplot(2,3,1)
sns.boxplot(x='Unit price',data=data)
plt.subplot(2,3,2)
sns.boxplot(x='Quantity',data=data)
plt.subplot(2,3,3)
sns.boxplot(x='Total',data=data)
plt.subplot(2,3,4)
sns.boxplot(x='cogs',data=data)
plt.subplot(2,3,5)
sns.boxplot(x='Rating',data=data)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

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



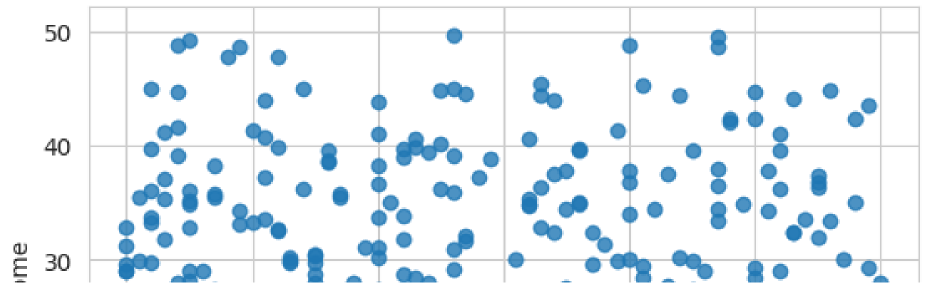
```
sns.pairplot(data=data)
```

```
<seaborn.axisgrid.PairGrid at 0x78e975897f10>
```



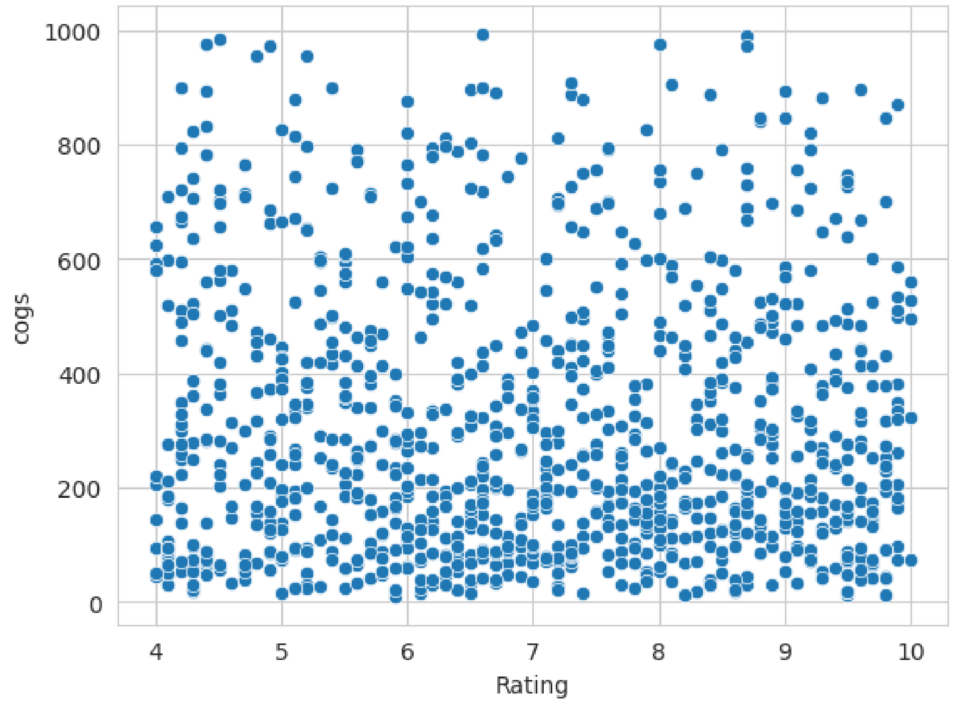
```
sns.regplot(x='Rating', y= 'gross income', data=data)
```

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



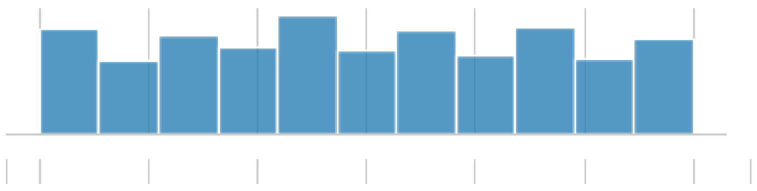
```
sns.scatterplot(x='Rating', y= 'cogs', data=data)
```

```
<Axes: xlabel='Rating', ylabel='cogs'>
```



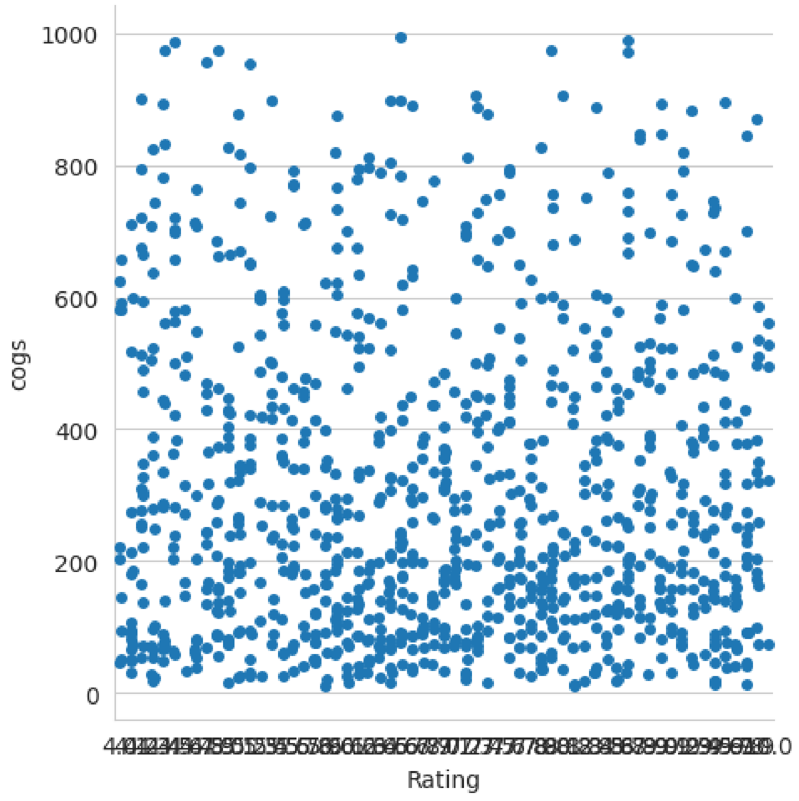
```
sns.jointplot(x='Rating', y= 'Total', data=data)
```

```
<seaborn.axisgrid.JointGrid at 0x78e978475300>
```



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

```
<seaborn.axisgrid.FacetGrid at 0x78e9af093760>
```



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

```
<seaborn.axisgrid.FacetGrid at 0x78e97201bd30>
```

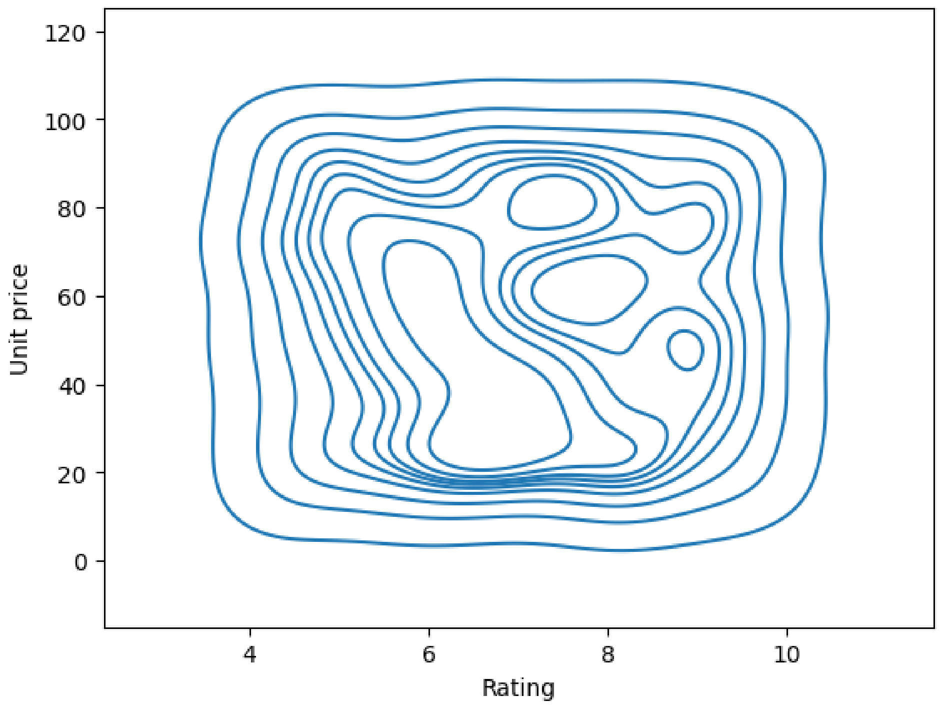
```
data.columns
```

```
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')
```

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

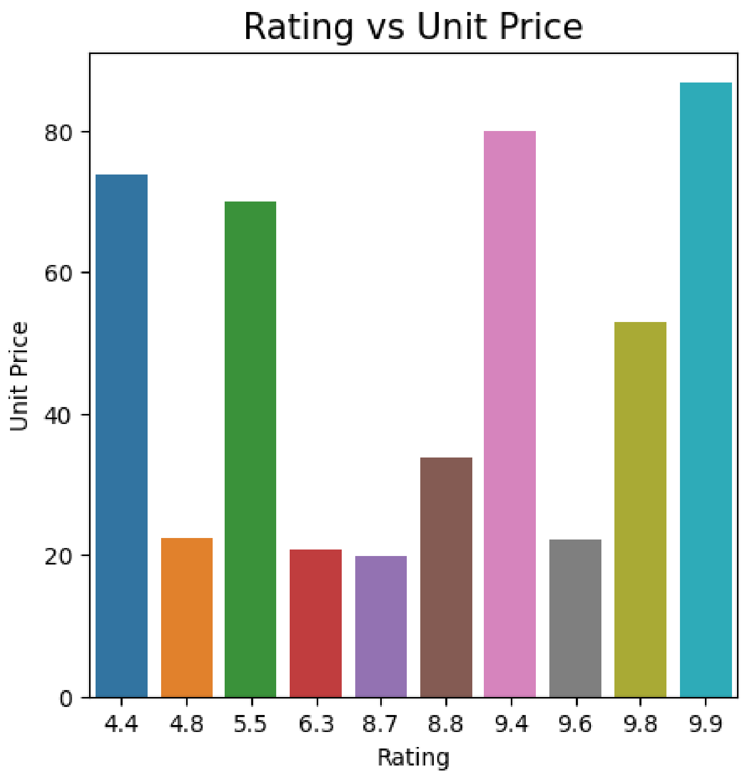
```
sns.kdeplot(x='Rating', y= 'Unit price', data=data)
```

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



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

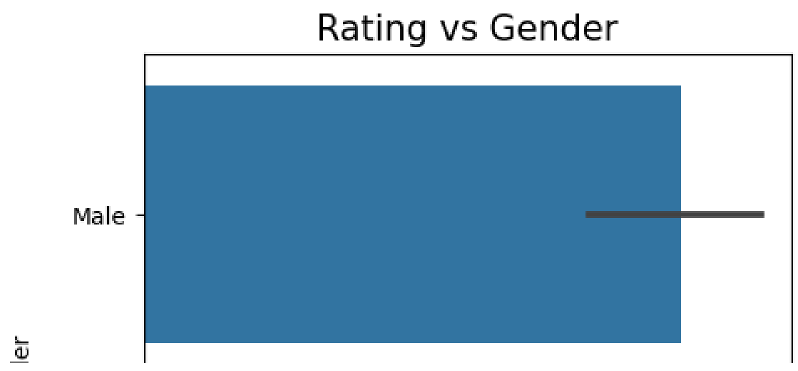
```
plt.style.use("default")
plt.figure(figsize=(5,5))
sns.barplot(x="Rating", y="Unit price", data=data[170:180])
plt.title("Rating vs Unit Price",fontsize=15)
plt.xlabel("Rating")
plt.ylabel("Unit Price")
plt.show()
```



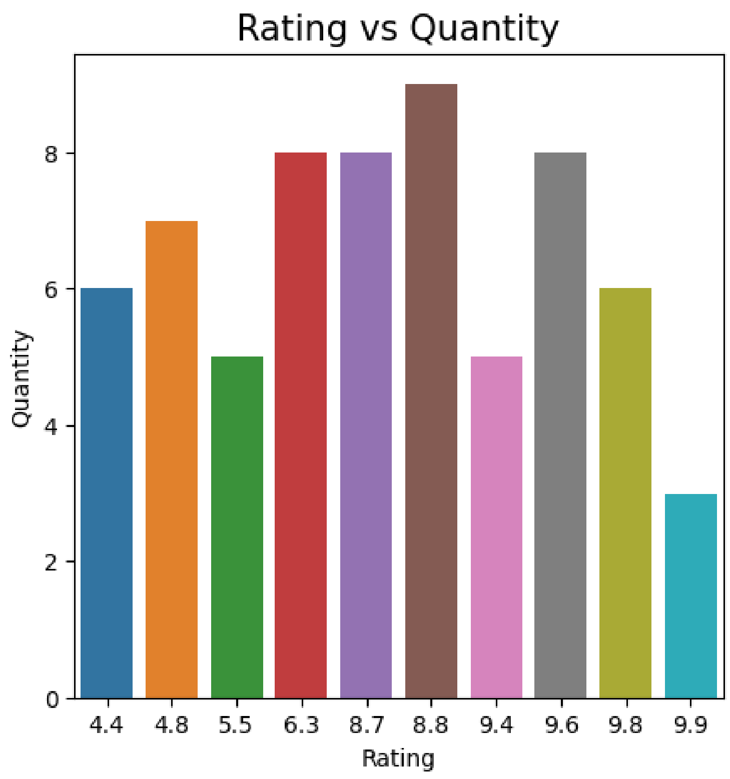
```
data.columns
```

```
     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')
```

```
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()
```



```
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()
```



```
#lets find the categorialfeatures
list_1=list(data.columns)
```

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

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

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

```
data
```

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	T
0	814	0	2	0	0	3	74.69	7	26.1415	548.
1	142	2	1	1	0	0	15.28	5	3.8200	80.
2	653	0	2	1	1	4	46.33	7	16.2155	340.
3	18	0	2	0	1	3	58.22	8	23.2880	489.
4	339	0	2	1	1	5	86.31	7	30.2085	634.
...	...	...	...	...	...	...	...	...	...	
995	153	2	1	1	1	3	40.35	1	2.0175	42.
996	250	1	0	1	0	4	97.38	10	48.6900	1022.
997	767	0	2	0	1	2	31.84	1	1.5920	33.
998	308	0	2	1	1	4	65.82	1	3.2910	69.
999	935	0	2	0	0	1	88.34	7	30.9190	649.

1000 rows × 17 columns

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

```
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)
```

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

```
800
200
800
200
```

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

knn.fit(x_train,y_train)
```

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

```
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
from sklearn.svm import SVC

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

```
▾ SVC
SVC()
```

```
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
```

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

```
▾ GaussianNB
GaussianNB()
```

```
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
```

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

dtree.fit(x_train,y_train)
```

```
▾                          DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=6, random_state=123)
```

```
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
```

```
#Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
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       0.51      0.56      0.53       100
           1       0.51      0.46      0.48       100

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

Confusion Matrix:
 [[56 44]
 [54 46]]
Training Score:
 100.0
```

```
#Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier
gbc=GradientBoostingClassifier()
gbc.fit(x_train,y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

```
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)
```

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

           0       0.47      0.47      0.47       100
           1       0.46      0.46      0.46       100

    accuracy                           0.47       200
   macro avg       0.46      0.46      0.46       200
weighted avg       0.46      0.47      0.46       200

Confusion Matrix:
 [[47 53]
 [54 46]]
Training Score:
 88.75
```

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

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

```
#XGB Classifier
from xgboost import XGBClassifier

xgb =XGBClassifier(objective ='reg:linear', colsample_bytree = 0.3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 10)

xgb.fit(x_train, y_train)
```

```
▾                                    XGBClassifier
XGBClassifier(alpha=10, base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.3, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=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,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=10, n_jobs=None,
              num_parallel_tree=None, ...)
```

```
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       0.46      0.48      0.47       100
           1       0.46      0.44      0.45       100

    accuracy                           0.46       200
   macro avg       0.46      0.46      0.46       200
weighted avg       0.46      0.46      0.46       200

Confusion Matrix:
 [[48 52]
 [56 44]]
Training Score:
 62.0
```

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

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

```
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
```

```
#Bagging Classifier
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)
```

```
0.54
```

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

	Actual	Predicted
859	0	1
[illegible]72	0	1
...	...	...
679	1	1
722	1	1
215	1	0