
Name: Nandhana Rajeev

Class: 3 MSc DS B

Reg No: 23122125

RANDOM FOREST REGRESSION ANALYSIS

Data Exploration

In [ ]:
```
# importing the libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
# reading dataset

df=pd.read_csv('supermarket_sales.csv')
df
```

Out[ ]:

	invoice_id	branch	city	customer_type	gender_customer	product_line	unit
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	
...	...	...	...	...	...	...	
995	233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty	
996	303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle	
997	727-02-1313	A	Yangon	Member	Male	Food and beverages	
998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle	
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories	

1000 rows × 17 columns

In [ ]:

```
# printing the first 5 rows of the dataset

df.head()
```

Out[ ]:

	invoice_id	branch	city	customer_type	gender_customer	product_line	unit_c
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86

In [ ]:
```
# printing sample rows of the dataset

df.sample(5)
```

Out[ ]:

	invoice_id	branch	city	customer_type	gender_customer	product_line	unit
268	144-51-6085	A	Yangon	Member	Male	Home and lifestyle	
329	340-66-0321	A	Yangon	Member	Male	Electronic accessories	
737	137-63-5492	C	Naypyitaw	Normal	Male	Electronic accessories	
148	262-47-2794	B	Mandalay	Member	Male	Home and lifestyle	
993	690-01-6631	B	Mandalay	Normal	Male	Fashion accessories	

In [ ]:
```
# dimensions of the dataset

df.shape
```

Out[ ]: (1000, 17)

In [ ]:
```
# returns the concise summary of the dataset

df.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_customer  1000 non-null   object 
 5   product_line     1000 non-null   object 
 6   unit_cost        1000 non-null   float64
 7   quantity         1000 non-null   int64  
 8   5pct_markup      1000 non-null   float64
 9   revenue          1000 non-null   float64
 10  date             1000 non-null   object 
 11  time             1000 non-null   object 
 12  payment_method   1000 non-null   object 
 13  cogs             1000 non-null   float64
 14  gm_pct           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 [ ]:
```
# returns statistical summary
```

```
df.describe()
```

Out[ ]:

	unit_cost	quantity	5pct_markup	revenue	cogs	gm_pct	g
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000	
mean	55.672130	5.510000	15.379369	322.966749	307.58738	4.761905	
std	26.494628	2.923431	11.708825	245.885335	234.17651	0.000000	
min	10.080000	1.000000	0.508500	10.678500	10.17000	4.761905	
25%	32.875000	3.000000	5.924875	124.422375	118.49750	4.761905	
50%	55.230000	5.000000	12.088000	253.848000	241.76000	4.761905	
75%	77.935000	8.000000	22.445250	471.350250	448.90500	4.761905	
max	99.960000	10.000000	49.650000	1042.650000	993.00000	4.761905	

In [ ]:

```
# displaying the number of unique data in each column

df.nunique()
```

Out[ ]:

```
invoice_id         1000
branch                3
city                  3
customer_type         2
gender_customer       2
product_line          6
unit_cost           943
quantity             10
5pct_markup         990
revenue             990
date                 89
time                506
payment_method        3
cogs                990
gm_pct                1
gross_income        990
rating               61
dtype: int64
```

In [ ]:

```
# checking for duplicates in the dataset

df.duplicated().sum()
```

Out[ ]: 0

In [ ]:

```
# checking for duplicates in the dataset

df.duplicated().sum()
```

Out[ ]: 0

In [ ]:

```
# finding the columns
df.columns
```

Out[ ]:
```
Index(['invoice_id', 'branch', 'city', 'customer_type', 'gender_customer',
       'product_line', 'unit_cost', 'quantity', '5pct_markup', 'revenue',
       'date', 'time', 'payment_method', 'cogs', 'gm_pct', 'gross_income',
       'rating'],
      dtype='object')
```

Data Preprocessing

In [ ]:
```
df = df.drop(columns=['invoice_id'])  # Drop unnecessary columns
df = df.drop(columns=['date','time'])
df
```

Out[ ]:

	branch	city	customer_type	gender_customer	product_line	unit_cost	quar
0	A	Yangon	Member	Female	Health and beauty	74.69	
1	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	
2	A	Yangon	Normal	Male	Home and lifestyle	46.33	
3	A	Yangon	Member	Male	Health and beauty	58.22	
4	A	Yangon	Normal	Male	Sports and travel	86.31	
...	...	...	...	...	...	...	
995	C	Naypyitaw	Normal	Male	Health and beauty	40.35	
996	B	Mandalay	Normal	Female	Home and lifestyle	97.38	
997	A	Yangon	Member	Male	Food and beverages	31.84	
998	A	Yangon	Normal	Male	Home and lifestyle	65.82	
999	A	Yangon	Member	Female	Fashion accessories	88.34	

1000 rows × 14 columns

In [ ]:
```
#for the data type of the column
str(df["gender_customer"].dtype)
```

Out[ ]: 'object'

In [ ]:
```
#converting categorical values to continuous
for column in df.columns:
    if str(df[column].dtype) == "object":
        for i in range(len(df[column].unique())):
            df[column] = df[column].replace(df[column].unique()[i],i)
```

```
df
```

Out[ ]:

	branch	city	customer_type	gender_customer	product_line	unit_cost	quantity
0	0	0	0	0	0	74.69	7
1	1	1	1	0	1	15.28	5
2	0	0	1	1	2	46.33	7
3	0	0	0	1	0	58.22	8
4	0	0	1	1	3	86.31	7
...	...	...	...	...	...	...	...
995	1	1	1	1	0	40.35	1
996	2	2	1	0	2	97.38	10
997	0	0	0	1	4	31.84	1
998	0	0	1	1	2	65.82	1
999	0	0	0	0	5	88.34	7

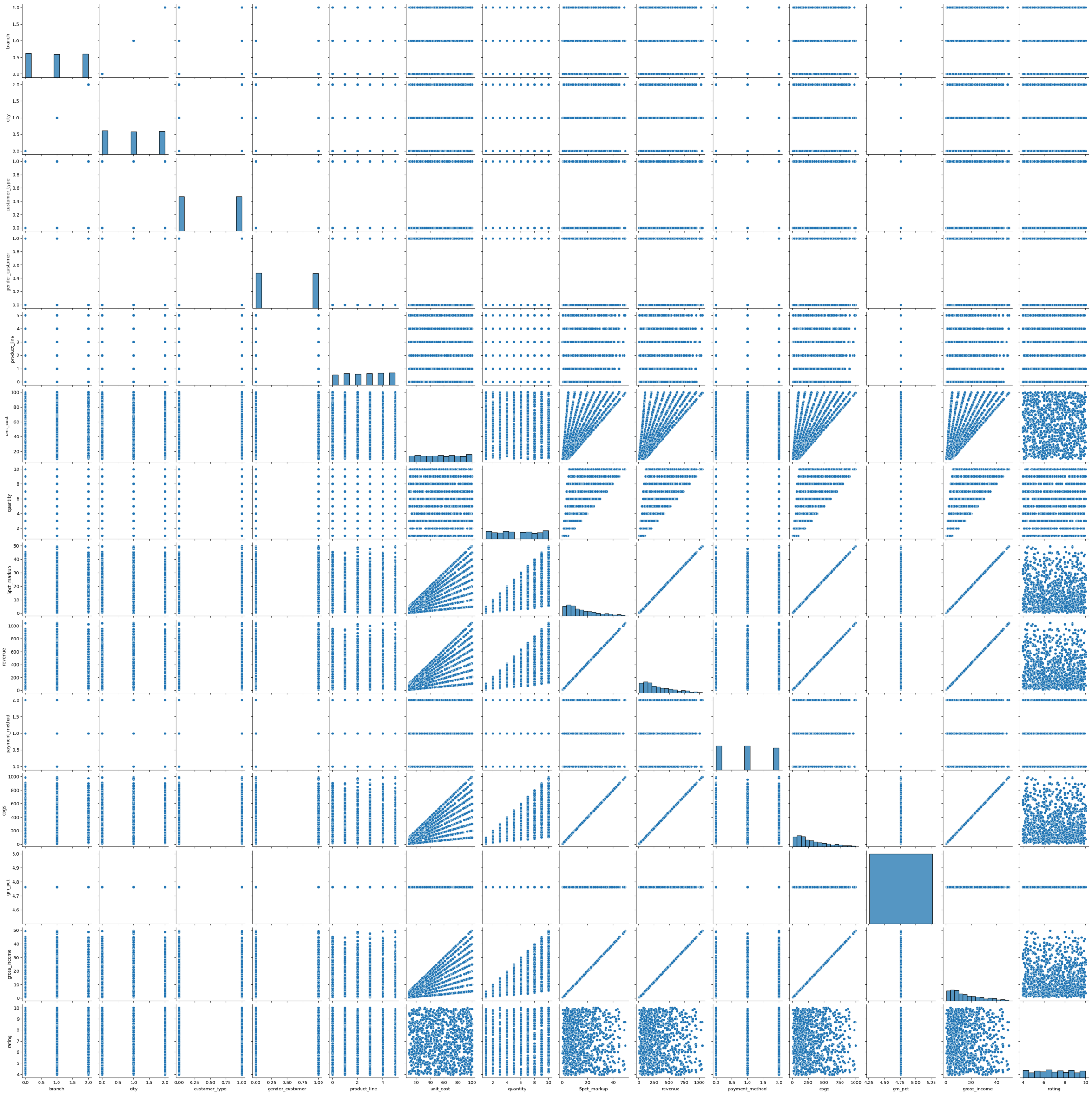
1000 rows × 14 columns

Data Visualization

In [ ]:

```
sns.pairplot(df)
```

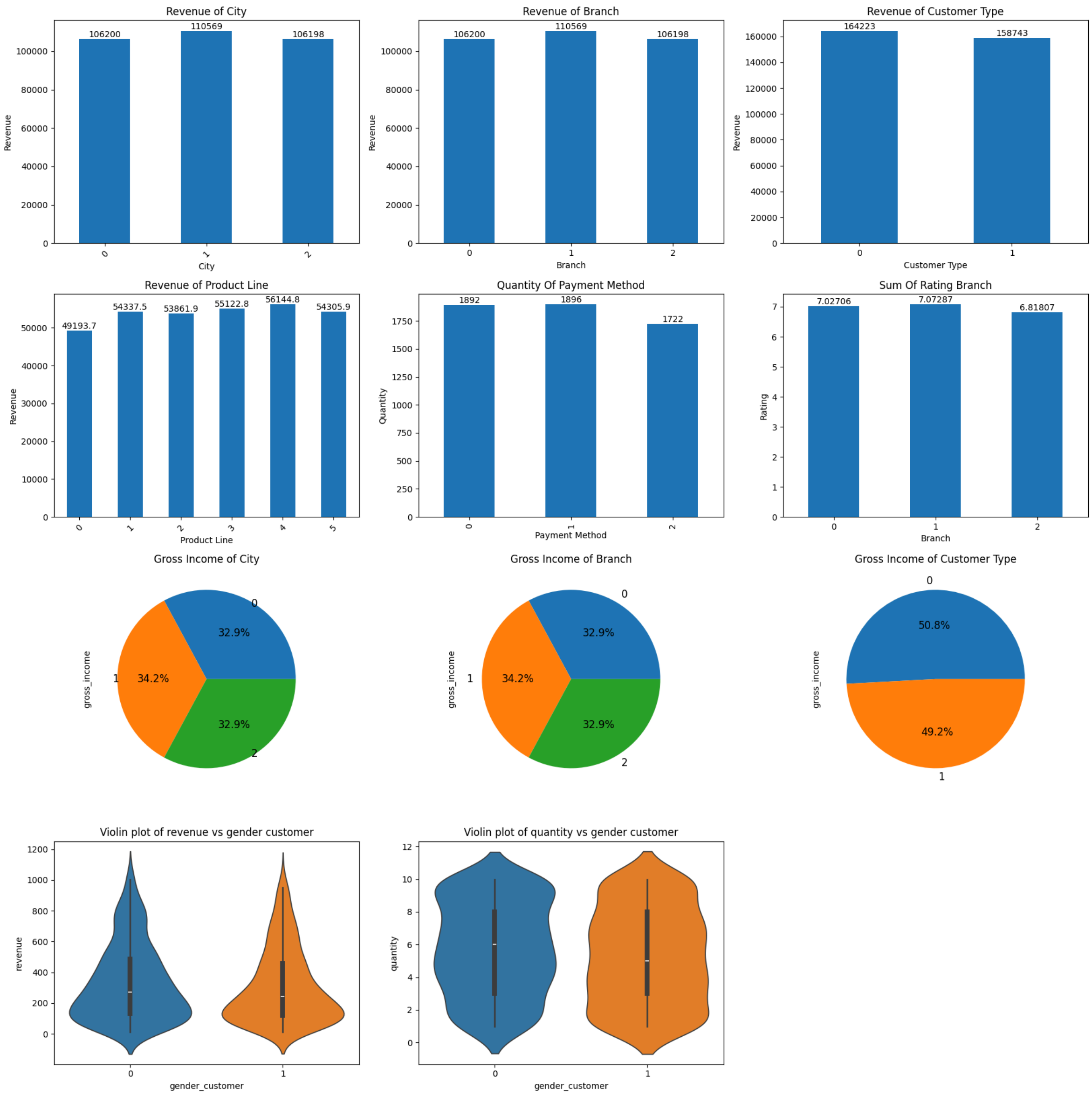
Out[ ]: <seaborn.axisgrid.PairGrid at 0x2092de8eb90>



In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create subplots with 4 rows and 3 columns
fig, axes = plt.subplots(4, 3, figsize=(18, 18))

# Flatten the axes array
axes = axes.flatten()

# Plot 1: Revenue of City
city_revenue = df.groupby('city')['revenue'].sum()
city_revenue.plot(kind='bar', ax=axes[0])
axes[0].set_title('Revenue of City')
axes[0].set_xlabel('City')
axes[0].set_ylabel('Revenue')
axes[0].tick_params(axis='x', rotation=45)
axes[0].bar_label(axes[0].containers[0], label_type='edge')

# Plot 2: Revenue of Branch
branch_revenue = df.groupby('branch')['revenue'].sum()
branch_revenue.plot(kind='bar', ax=axes[1])
axes[1].set_title('Revenue of Branch')
axes[1].set_xlabel('Branch')
axes[1].set_ylabel('Revenue')
```

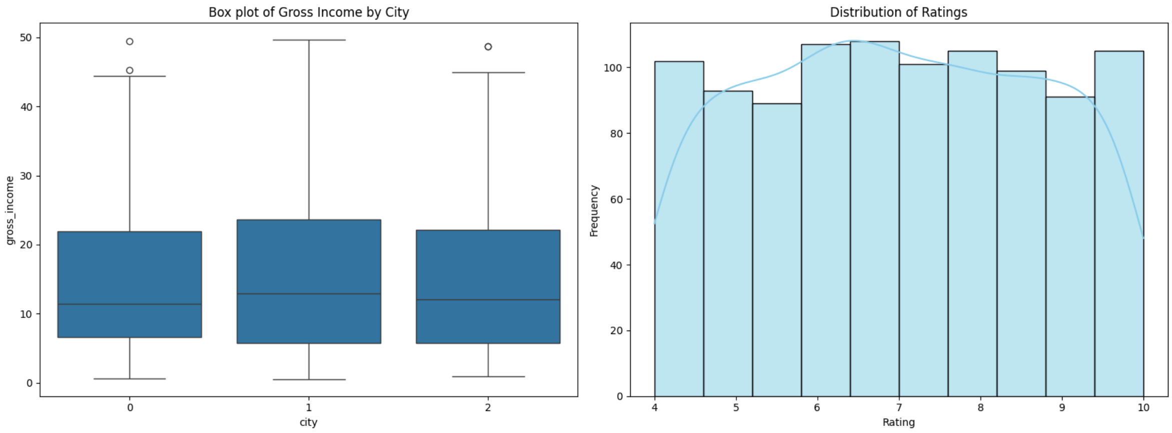
```
axes[1].tick_params(axis='x', rotation=0)
axes[1].bar_label(axes[1].containers[0], label_type='edge')

# Plot 3: Revenue of Customer Type
customer_type_revenue = df.groupby('customer_type')['revenue'].sum()
customer_type_revenue.plot(kind='bar', ax=axes[2])
axes[2].set_title('Revenue of Customer Type')
axes[2].set_xlabel('Customer Type')
axes[2].set_ylabel('Revenue')
axes[2].tick_params(axis='x', rotation=0)
axes[2].bar_label(axes[2].containers[0], label_type='edge')

# Plot 4: Revenue of Product Line
product_line_revenue = df.groupby('product_line')['revenue'].sum()
product_line_revenue.plot(kind='bar', ax=axes[3])
axes[3].set_title('Revenue of Product Line')
axes[3].set_xlabel('Product Line')
axes[3].set_ylabel('Revenue')
axes[3].tick_params(axis='x', rotation=45)
axes[3].bar_label(axes[3].containers[0], label_type='edge')

# Plot 5: Quantity Of Payment Method
payment_method_quantity = df.groupby('payment_method')['quantity'].sum()
payment_method_quantity.plot(kind='bar', ax=axes[4])
axes[4].set_title('Quantity Of Payment Method')
axes[4].set_xlabel('Payment Method')
axes[4].set_ylabel('Quantity')
axes[4].tick_params(axis='x', rotation=90)
axes[4].bar_label(axes[4].containers[0], label_type='edge')

# Plot 6: Sum Of Rating Branch
Sum_of_rating_branch = df.groupby('branch')['rating'].mean()
Sum_of_rating_branch.plot(kind='bar', ax=axes[5])
axes[5].set_title('Sum Of Rating Branch')
axes[5].set_xlabel('Branch')
axes[5].set_ylabel('Rating')
axes[5].tick_params(axis='x', rotation=0)
axes[5].bar_label(axes[5].containers[0], label_type='edge')

# Plot 7: Gross Income of City
city_gross_income = df.groupby('city')['gross_income'].sum()
city_gross_income.plot(kind='pie', labeldistance=0.98, autopct='%1.1f%%', ax=axe
axes[6].set_title('Gross Income of City')

# Plot 8: Gross Income of Branch
branch_gross_income = df.groupby('branch')['gross_income'].sum()
branch_gross_income.plot(kind='pie', autopct='%1.1f%%', ax=axes[7], textprops={"
axes[7].set_title('Gross Income of Branch')

# Plot 9: Gross Income of Customer Type
customer_type_gross_income = df.groupby('customer_type')['gross_income'].sum()
customer_type_gross_income.plot(kind='pie', autopct='%1.1f%%', ax=axes[8], textp
axes[8].set_title('Gross Income of Customer Type')

# Plot 10: Violin plot of revenue vs gender customer
sns.violinplot(x='gender_customer', y='revenue', data=df, palette="tab10", ax=ax
axes[9].set_title('Violin plot of revenue vs gender customer')

# Plot 11: Violin plot of quantity vs gender customer
sns.violinplot(x='gender_customer', y='quantity', data=df, palette="tab10", ax=a
```

```
axes[10].set_title('Violin plot of quantity vs gender customer')

# Hide the empty subplot
axes[11].axis('off')

plt.tight_layout()
plt.show()
```

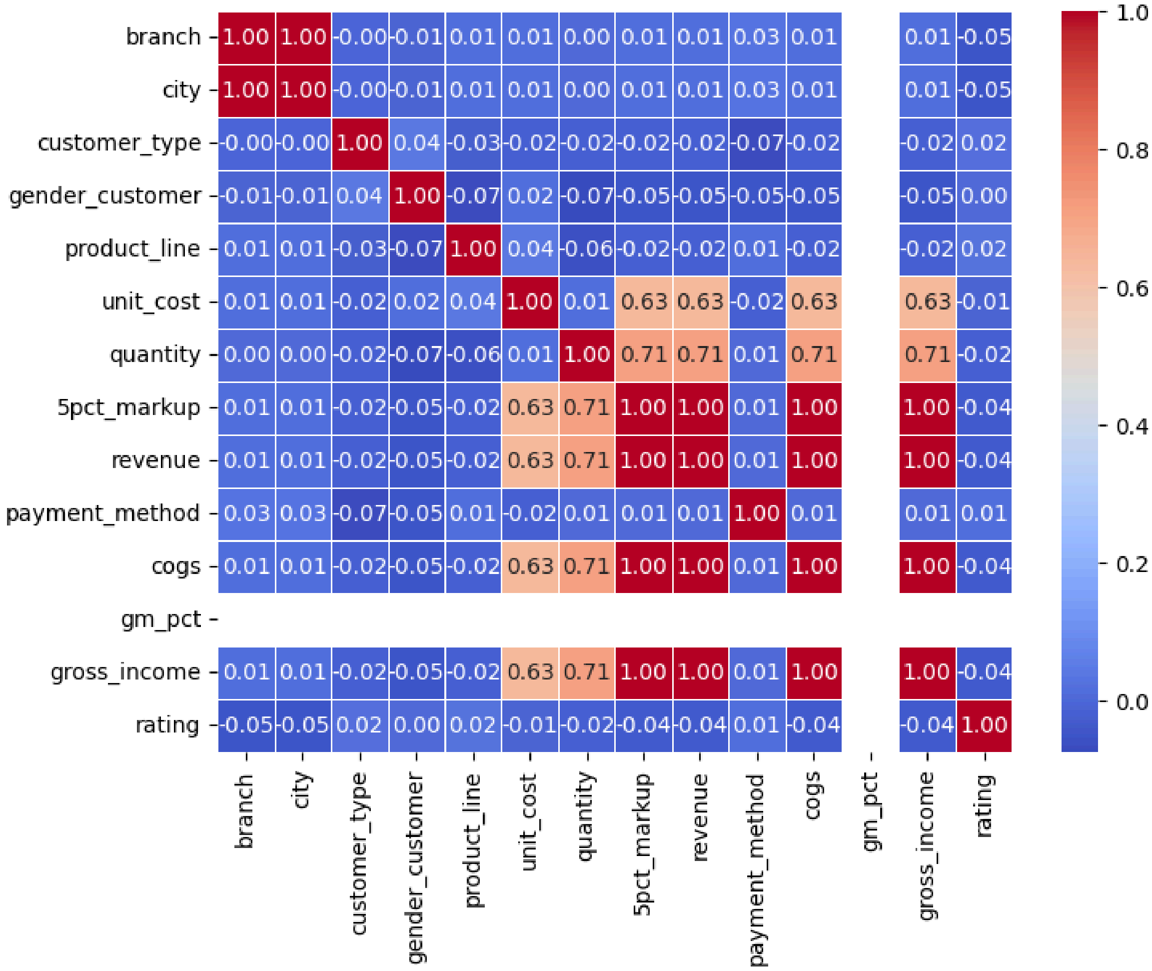


In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create subplots with 1 row and 2 columns
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Plot 1: Box plot of Gross Income by City
sns.boxplot(x='city', y='gross_income', data=df, ax=axes[0])
axes[0].set_title('Box plot of Gross Income by City')

# Plot 2: Distribution of Ratings
sns.histplot(df['rating'], kde=True, bins=10, color='skyblue', ax=axes[1])
axes[1].set_title('Distribution of Ratings')
axes[1].set_xlabel('Rating')
axes[1].set_ylabel('Frequency')

# Adjust layout
```

```
plt.tight_layout()

# Show plots
plt.show()
```



In [ ]:
```
#plotting heatmap
numeric_columns = df.select_dtypes(include=[np.number]).columns
numeric_df = df[numeric_columns]
correlation_matrix = numeric_df.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidt
plt.show()
```



In [ ]:
```
# Splitting the Data
from sklearn.model_selection import train_test_split
```

In [ ]:
```
# Selected features for X
selected_features = ['customer_type','gender_customer','product_line','unit_cost
X = df[selected_features]

# Target variable for y
y = df['gross_income']
```

In [ ]:
```
# Splitting Data into Training and Testing Sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

Training the Random Forest Regressor

In [ ]:
```
# Training the Random Forest Classifier
from sklearn.ensemble import RandomForestRegressor
rf_regressor = RandomForestRegressor()

# Fitting the model on the training data
rf_regressor.fit(X_train, y_train)
```

Out[ ]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

Making Predictions and Evaluating the Model of the testing set

In [ ]:
```
# Making Predictions
y_pred = rf_regressor.predict(X_test)

# Evaluating the Model
from sklearn.metrics import r2_score,mean_squared_error

mse = mean_squared_error(y_test,y_pred)
r2 = r2_score(y_test,y_pred)
print("Mean Squared Error for the testing  is " ,mse)
print("R squared value  for the testing is ",r2)
```

```
Mean Squared Error for the testing  is  0.004048309357999716
R squared value  for the testing is  0.9999698979573257
```

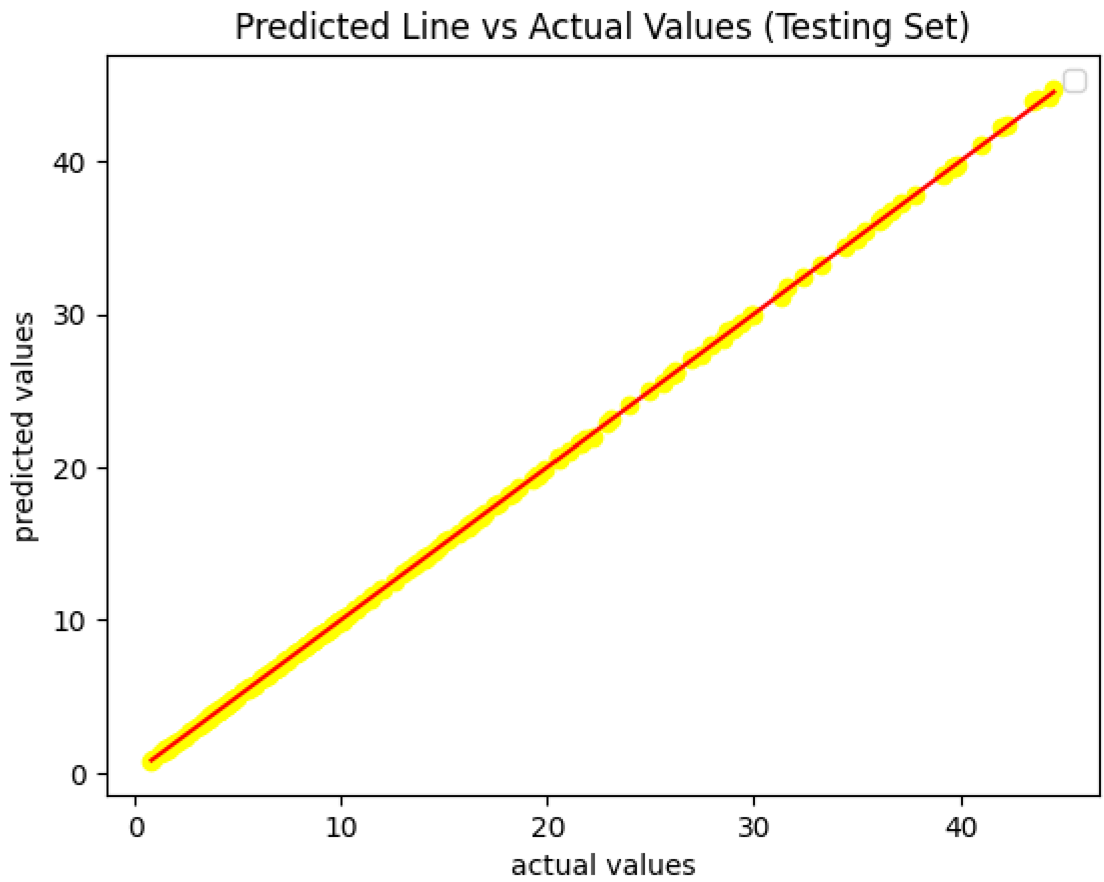
The Random Forest model demonstrates outstanding performance on the testing data. The Mean Squared Error and R-squared value indicate that the model predicts gross_income with very high accuracy and explains nearly all the variability in the target variable. However, it is important to verify this performance with additional validation methods to ensure the model is not overfitting.

Visualizing the Test Set Predictions

In [ ]:
```
# Plotting the actual vs predicted values for the test set
import matplotlib.pyplot as plt
plt.scatter(y_test,y_pred,  color='yellow')
plt.plot([min(y_test),max(y_test)],[min(y_test),max(y_test)],color="red")

plt.ylabel('predicted values')
plt.xlabel('actual values')
```

```
plt.title('Predicted Line vs Actual Values (Testing Set)')
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label sta
rt with an underscore are ignored when legend() is called with no argument.
```



Making Predictions and Evaluating the Model of the training set

In [ ]:

```
# Making predictions on the training set
y_pred_train = rf_regressor.predict(X_train)

# Evaluating the model performance on the training set
mse_train = mean_squared_error(y_train, y_pred_train)
r2_train = r2_score(y_train, y_pred_train)

# Printing the evaluation metrics for the training set
print("Mean Squared Error on Training Set is ", mse_train)
print("R squared value on Training Set is ", r2_train)
```

```
Mean Squared Error on Training Set is  0.00077098326965621
R squared value on Training Set is  0.9999943865125777
```

The Random Forest model shows exceptional performance on the training data.The Mean Squared Error and R-squared value indicate that the model predicts gross_income with extremely high accuracy and explains virtually all the variability in the target variable on the training set.

Visualizing the Train Set Predictions

In [ ]:

```
# Plotting the actual vs predicted values for the training set
plt.scatter(y_train, y_pred_train, color='yellow')
```

```
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)], color="red"

plt.ylabel('Predicted Values')
plt.xlabel('Actual Values')
plt.title('Predicted vs Actual Values (Training Set)')
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label sta
rt with an underscore are ignored when legend() is called with no argument.
```