欄位介紹

InvoiceNo: 發票號碼. 唯一值, "C"開頭代表該交易取消

ItemCode: 產品代碼, 每個產品的唯一值

DescriptionCode: 商品敘述代碼

Quantity: 每個商品數量

SellDate: 發票產生時間, 代表每筆交易發生的日期跟時間

NewTaiwanDollors: 物品單價

CustomerID: 用戶代碼, 唯一值

District: 銷售縣市

EDA

In [169]:

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import sqlite3
import pandas as pd
import numpy as np
import time
import math
import datetime as dt
import sklearn.cluster as cluster
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_samples, silhouette_score
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
import seaborn as sns

conn = sqlite3.connect('ecommerce.db')
c = conn.cursor()

retail = pd.read_sql('''SELECT * FROM ecommerce''', conn)

# 顯示基本訊值
retail.head()
```

Out[169]:

	InvoiceNo	ItemCode	DescriptionCode	Quantity	SellDate	NewTaiwanDollors	CustomerID
0	536365	85123A	1546686	6	12/1/2010 8:26	255.0	17850.0
1	536365	71053	1466048	6	12/1/2010 8:26	339.0	17850.0
2	536365	84406B	4510747	8	12/1/2010 8:26	275.0	17850.0
3	536365	84029G	6497318	6	12/1/2010 8:26	339.0	17850.0
4	536365	84029E	3876120	6	12/1/2010 8:26	339.0	17850.0

In [172]:

```
# 做 EDA, 先看缺值值
retail.isna().sum().sort_values(ascending=False)

# 發現 CustomerID 占比最高
pd.DataFrame(data = (retail.isna().sum() / retail.shape[0]) * 100, index = retai
l.columns, columns = ['% Null Values'])

# 踢掉空值:
retail.dropna(subset=['CustomerID'],how='any',inplace=True)

# 再查看一次缺值
retail.isna().sum()
```

Out[172]:

```
CustomerID          135080
DescriptionCode       1454
District                 0
NewTaiwanDollors         0
SellDate                 0
Quantity                 0
ItemCode                 0
InvoiceNo                0
dtype: int64
```

Out[172]:

	% Null Values
InvoiceNo	0.000000
ItemCode	0.000000
DescriptionCode	0.268311
Quantity	0.000000
SellDate	0.000000
NewTaiwanDollors	0.000000
CustomerID	24.926694
District	0.000000

Out[172]:

```
InvoiceNo           0
ItemCode            0
DescriptionCode     0
Quantity            0
SellDate            0
NewTaiwanDollors    0
CustomerID          0
District            0
dtype: int64
```

In [173]:

```
# 查看重複
retail.duplicated().sum()
# 去重複
retail.drop_duplicates(inplace=True)
```

Out[173]:

```
5225
```

In [174]:

```
# Removing the cancelled orders
retail = retail[retail['Quantity'] > 0]
```

In [175]:

```
# 確認交易次數, 購買數量, 客戶數
pd.DataFrame(data=[retail['InvoiceNo'].nunique(),retail['ItemCode'].nunique(),retail['CustomerID'].nunique()],columns=['Count'],
                   index=['Number of Transactions','Number of Unique Products Bought','Number of Unique Customers'])
```

Out[175]:

	Count
Number of Transactions	18536
Number of Unique Products Bought	3665
Number of Unique Customers	4339

RFM Analysis (Recency、Frequency、Monetary)

Recency 最近一次消費

In [210]:

```
# 最近消費時間：
retail['InvoiceDate'] = retail['SellDate'].astype('datetime64')
retail['InvoiceDate'].max()
tsp = str(retail['InvoiceDate'].max()).split(' ')[0].split('-')
now = dt.date(int(tsp[0]),int(tsp[1]),int(tsp[2]))
print('Recent Consumption: ',now)

# 每個客戶近一次消費時間：
retail['Date'] = retail['InvoiceDate'].apply(lambda x: x.date())
recency_df = retail.groupby(by='CustomerID', as_index=False)['Date'].max()
recency_df.columns = ['CustomerID','LastPurshaceDate']
recency_df.head()
recency_df['LastPurshaceDate'].max()

# 每個客戶距離最近一次消費時間
recency_df['Recency'] = recency_df['LastPurshaceDate'].apply(lambda x: (now - x)
.days)
# recency_df.head()
recency_df.drop('LastPurshaceDate',axis=1,inplace=True)
recency_df.head()
```

Out[210]:

```
Timestamp('2011-12-09 12:50:00')
```

```
Recent Consumption:  2011-12-09
```

Out[210]:

	CustomerID	LastPurshaceDate
0	12346.0	2011-01-18
1	12347.0	2011-12-07
2	12348.0	2011-09-25
3	12349.0	2011-11-21
4	12350.0	2011-02-02

Out[210]:

```
datetime.date(2011, 12, 9)
```

Out[210]:

	CustomerID	Recency
0	12346.0	325
1	12347.0	2
2	12348.0	75
3	12349.0	18
4	12350.0	310

Frequency 消費頻率

In [209]:

```
temp = retail.copy()
temp.drop_duplicates(['InvoiceNo','CustomerID'],keep='first',inplace=True)
frequency_df = temp.groupby(by=['CustomerID'], as_index=False)['InvoiceNo'].coun
t()
frequency_df.columns = ['CustomerID','Frequency']
frequency_df.head()
```

Out[209]:

	CustomerID	Frequency
0	12346.0	1
1	12347.0	7
2	12348.0	4
3	12349.0	1
4	12350.0	1

Monetary 消費金額

In [212]:

```
retail['TotalCost'] = retail['Quantity'] * retail['NewTaiwanDollors']

# 每個顧客消費總金額
monetary_df = retail.groupby(by='CustomerID',as_index=False).agg({'TotalCost':
'sum'})
monetary_df.columns = ['CustomerID','Monetary']
monetary_df.head()
```

Out[212]:

	CustomerID	Monetary
0	12346.0	7718360.0
1	12347.0	431000.0
2	12348.0	179724.0
3	12349.0	175755.0
4	12350.0	33440.0

Create RFM Table

In [213]:

```
rfm_df = recency_df.merge(frequency_df,on='CustomerID').merge(monetary_df,on='Cu
stomerID')
rfm_df.set_index('CustomerID',inplace=True)
rfm_df.head()
```

Out[213]:

	Recency	Frequency	Monetary
CustomerID			
12346.0	325	1	7718360.0
12347.0	2	7	431000.0
12348.0	75	4	179724.0
12349.0	18	1	175755.0
12350.0	310	1	33440.0

Customer segments with RFM Model

In [225]:

```
### 假設 80-20 法則： 20% 顧客消費金額貢獻了80%的總金額

pareto_cutoff = rfm_df['Monetary'].sum() * 0.8
print('總體80%的金額：', pareto_cutoff)

# 先建立 Rank 欄位：
customers_ranked = rfm_df
customers_ranked['Rank'] = customers_ranked['Monetary'].rank(ascending=False)

# Rank 排序：
customers_ranked.sort_values(by='Rank',ascending=True,inplace=True)
customers_ranked.shape
# 取前20%
top_20_cutoff = customers_ranked.shape[0] * 20 /100
top_20_cutoff = math.ceil(top_20_cutoff)
revenueByTop20 = customers_ranked[customers_ranked['Rank'] <= 868]['Monetary'].s
um()
print('前20%的人花費金額：', revenueByTop20)
print()
print('80-20 Rule Ratio:', revenueByTop20/pareto_cutoff)
```

```
總體80%的金額： 712912632.3200002
```

Out[225]:

```
(4339, 4)
```

```
前20%的人花費金額： 664943746.1

80-20 Rule Ratio: 0.9327142148345764
```

Applying RFM Score Formula

In [228]:

```
# 用 4 分位來做 RFM Quartiles
quantiles = rfm_df.quantile(q=[0.25,0.5,0.75])
quantiles
quantiles.to_dict()
```

Out[228]:

	Recency	Frequency	Monetary	Rank
0.25	17.0	1.0	30724.5	1085.5
0.50	50.0	2.0	67445.0	2170.0
0.75	141.5	5.0	166164.0	3254.5

Out[228]:

```
{'Recency': {0.25: 17.0, 0.5: 50.0, 0.75: 141.5},
 'Frequency': {0.25: 1.0, 0.5: 2.0, 0.75: 5.0},
 'Monetary': {0.25: 30724.5, 0.5: 67445.0, 0.75: 166164.0},
 'Rank': {0.25: 1085.5, 0.5: 2170.0, 0.75: 3254.5}}
```

In [229]:

```
# 依據 RFM 評分：
#  Arguments (x = value, p = recency, monetary_value, frequency, d = quartiles d
ict)
def RScore(x,p,d):
    if x <= d[p][0.25]:
        return 4
    elif x <= d[p][0.50]:
        return 3
    elif x <= d[p][0.75]:
        return 2
    else:
        return 1

# Arguments (x = value, p = recency, monetary_value, frequency, k = quartiles di
ct)
def FMScore(x,p,d):
    if x <= d[p][0.25]:
        return 1
    elif x <= d[p][0.50]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4
```

In [230]:

```
rfm_segmentation = rfm_df
rfm_segmentation['R_Quartile'] = rfm_segmentation['Recency'].apply(RScore, args=('Recency',quantiles,))
rfm_segmentation['F_Quartile'] = rfm_segmentation['Frequency'].apply(FMScore, args=('Frequency',quantiles,))
rfm_segmentation['M_Quartile'] = rfm_segmentation['Monetary'].apply(FMScore, args=('Monetary',quantiles,))
```

In [232]:

```
rfm_segmentation.head()
```

Out[232]:

	Recency	Frequency	Monetary	Rank	R_Quartile	F_Quartile	M_Quartile
CustomerID							
14646.0	1	74	28020602.0	1.0	4	4	4
18102.0	0	60	25965730.0	2.0	4	4	4
17450.0	8	46	19455079.0	3.0	4	4	4
16446.0	0	2	16847250.0	4.0	4	2	4
14911.0	1	201	14382506.0	5.0	4	4	4

In [233]:

```
# Best Recency score = 4: most recently purchased.
# Best Frequency score = 4: most quantity purchase.
# Best Monetary score = 4: spent the most.
rfm_segmentation['RFMScore'] = rfm_segmentation.R_Quartile.map(str) \
                            + rfm_segmentation.F_Quartile.map(str) \
                            + rfm_segmentation.M_Quartile.map(str)
rfm_segmentation.head()
```

Out[233]:

	Recency	Frequency	Monetary	Rank	R_Quartile	F_Quartile	M_Quartile	RFM
CustomerID								
14646.0	1	74	28020602.0	1.0	4	4	4	
18102.0	0	60	25965730.0	2.0	4	4	4	
17450.0	8	46	19455079.0	3.0	4	4	4	
16446.0	0	2	16847250.0	4.0	4	2	4	
14911.0	1	201	14382506.0	5.0	4	4	4	

In [235]:

```
# 以消費總金額來做排序，看哪些用戶拿到 444滿分：
rfm_segmentation[rfm_segmentation['RFMScore']=='444'].sort_values('Monetary', as
cending=False).head(10)
```

Out[235]:

CustomerID	Recency	Frequency	Monetary	Rank	R_Quartile	F_Quartile	M_Quartile	RFM
14646.0	1	74	28020602.0	1.0	4	4	4	
18102.0	0	60	25965730.0	2.0	4	4	4	
17450.0	8	46	19455079.0	3.0	4	4	4	
14911.0	1	201	14382506.0	5.0	4	4	4	
14156.0	9	55	11737963.0	7.0	4	4	4	
17511.0	2	31	9106238.0	8.0	4	4	4	
16684.0	4	28	6665356.0	11.0	4	4	4	
14096.0	4	17	6516479.0	12.0	4	4	4	
13694.0	3	50	6503962.0	13.0	4	4	4	
15311.0	0	91	6076790.0	14.0	4	4	4	

In [237]:

```
# How many customers do we have in each segment?

print("Best Customers: ",len(rfm_segmentation[rfm_segmentation['RFMScore']=='44
4'])) # 滿分
print('Loyal Customers: ',len(rfm_segmentation[rfm_segmentation['F_Quartile']==4
])) # 頻率最高
print("Big Spenders: ",len(rfm_segmentation[rfm_segmentation['M_Quartile']==4]))
# 大戶
print('Customers at risk of churning: ', len(rfm_segmentation[rfm_segmentation[
'RFMScore']=='244'])) # 淺在流失，因為很久沒回來了
print('Almost Churned Customers: ',len(rfm_segmentation[rfm_segmentation['RFMSco
re']=='144'])) # 幾乎流失客戶，太久沒回來消費
print('Churned Customers: ',len(rfm_segmentation[rfm_segmentation['RFMScore']==
'111'])) # 886
```

```
Best Customers:  456
Loyal Customers:  872
Big Spenders:  1085
Customers at risk of churning:  70
Almost Churned Customers:  10
Churned Customers:  443
```

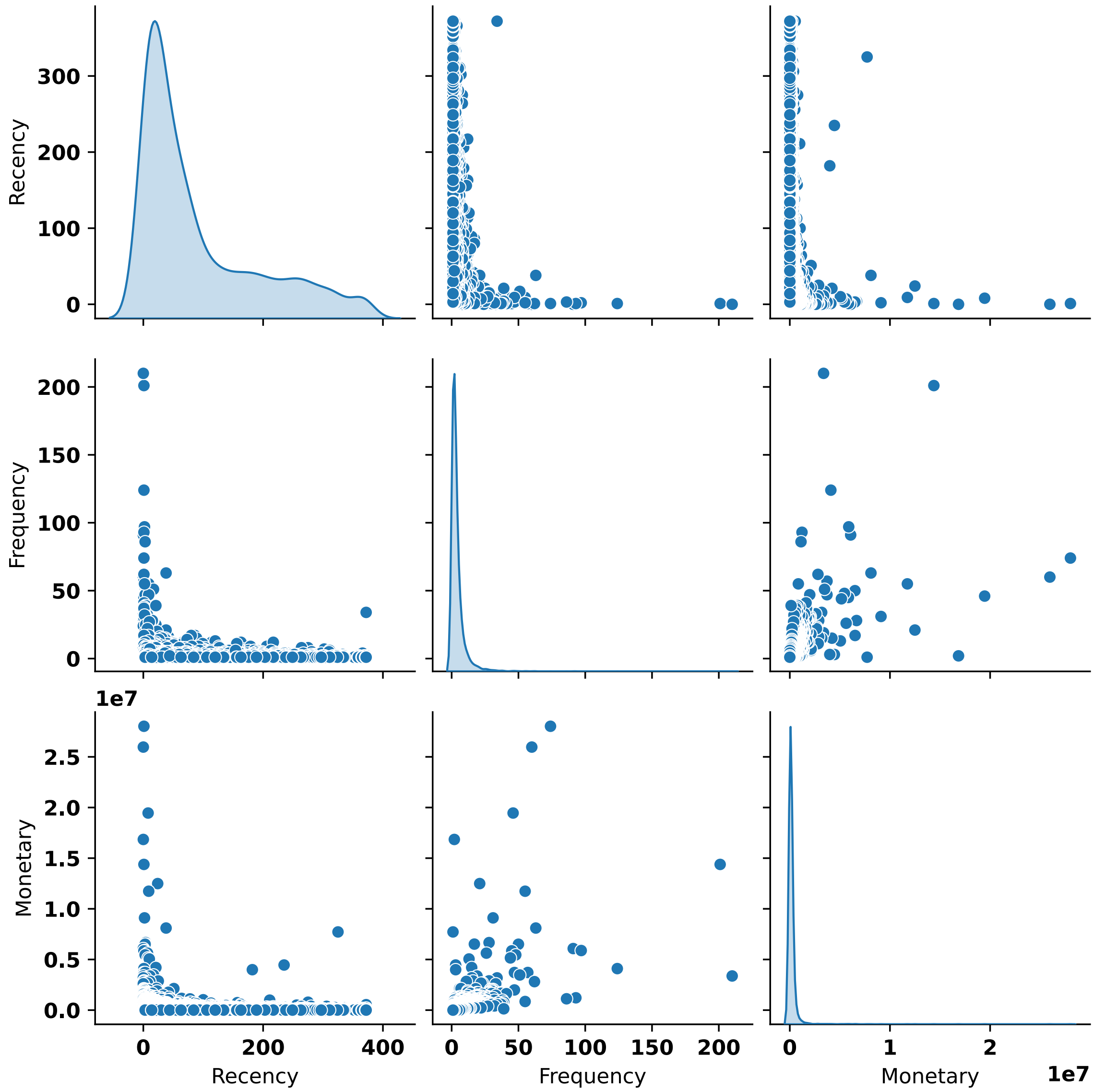
In [242]:

```
# 繪製RFM關聯圖:
rfm_data = rfm_df.drop(['R_Quartile','F_Quartile','M_Quartile','RFMScore','Rank'
],axis=1)
rfm_data.head()

_ = sns.pairplot(rfm_data,diag_kind='kde')
print('All the features are highly right skewed.')
```

Out[242]:

	Recency	Frequency	Monetary
CustomerID			
14646.0	1	74	28020602.0
18102.0	0	60	25965730.0
17450.0	8	46	19455079.0
16446.0	0	2	16847250.0
14911.0	1	201	14382506.0

```
All the features are highly right skewed.
```



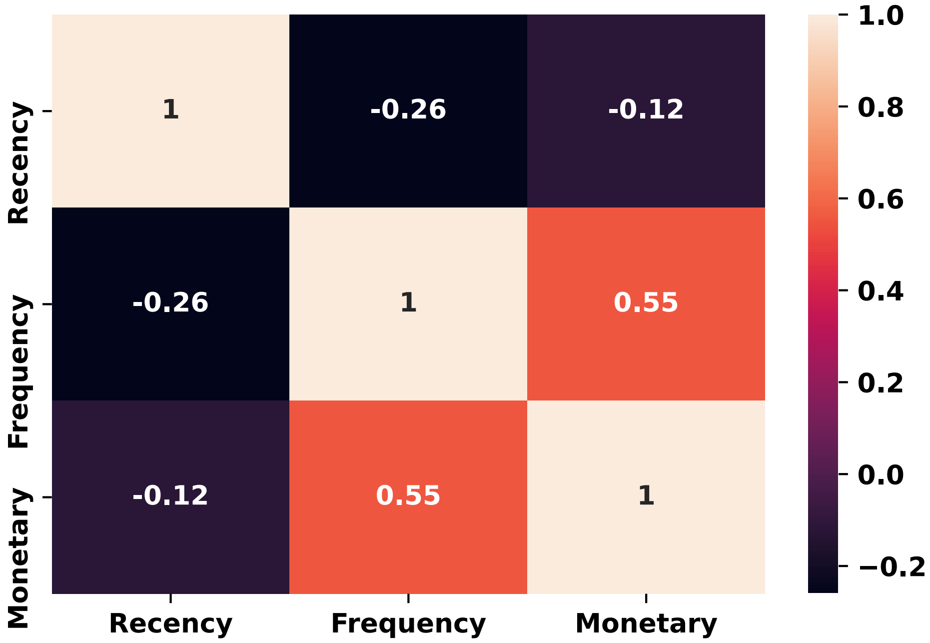
In [244]:

```
sns.heatmap(rfm_data.corr(),annot=True)
print('There is some positive correlation between Monetary and Frequency feature
s.')
```

Out[244]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x17fc857b8>
```

```
There is some positive correlation between Monetary and Frequency fe
atures.
```



In [246]:

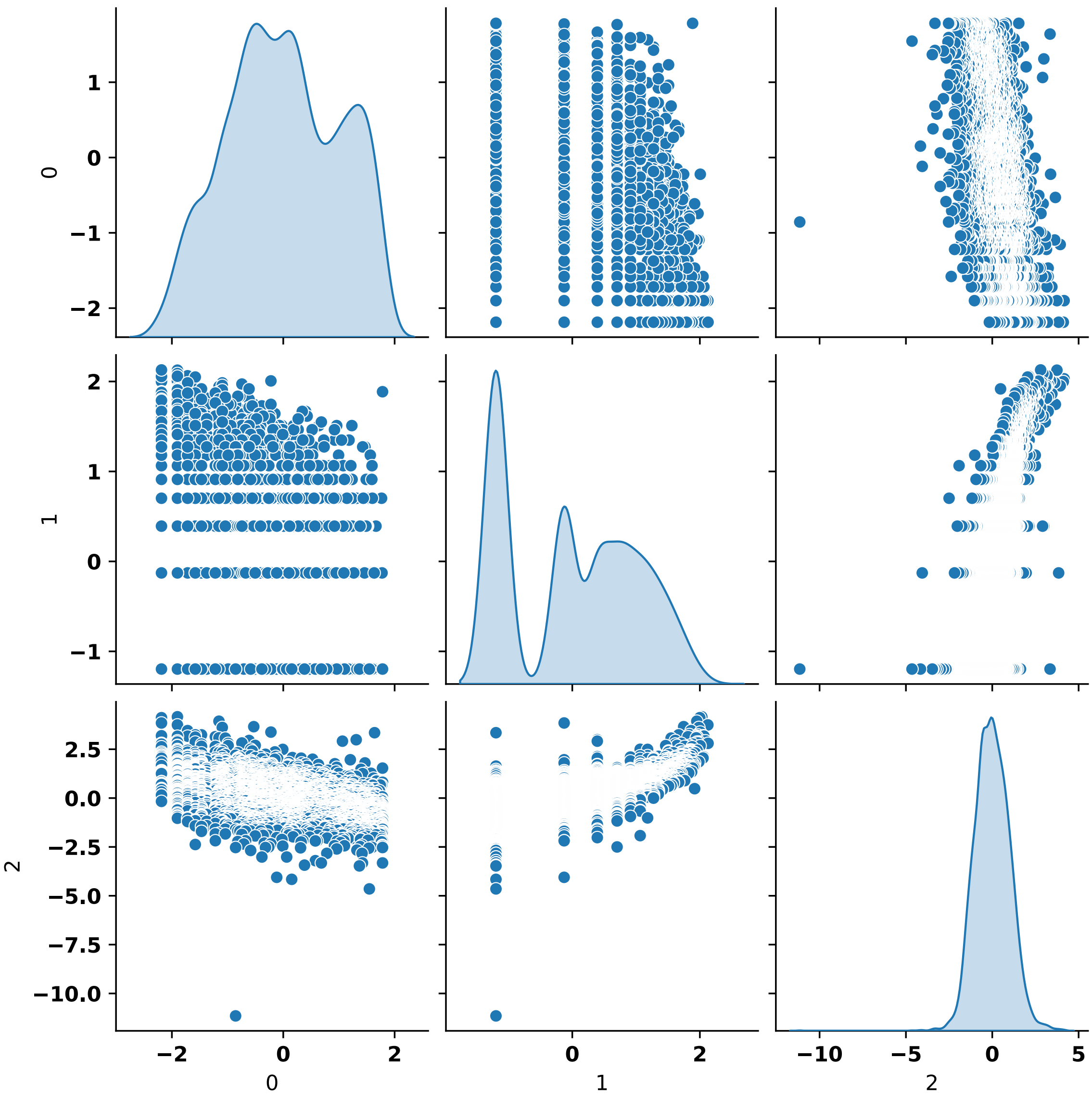
```
# 映射到高斯分布
from sklearn.preprocessing import PowerTransformer

features = rfm_data.columns
pt = PowerTransformer()
rfm_data = pd.DataFrame(pt.fit_transform(rfm_data))
rfm_data.columns = features
rfm_data.head()

sns.pairplot(rfm_data,diag_kind='kde')
```

Out[246]:

	0	1	2
0	-1.900648	2.029872	4.161541
1	-2.187012	1.998894	4.114387
2	-1.154873	1.951991	3.934513
3	-2.187012	-0.127947	3.844145
4	-1.900648	2.125579	3.744293

Out[246]:

<seaborn.axisgrid.PairGrid at 0x18276b780>



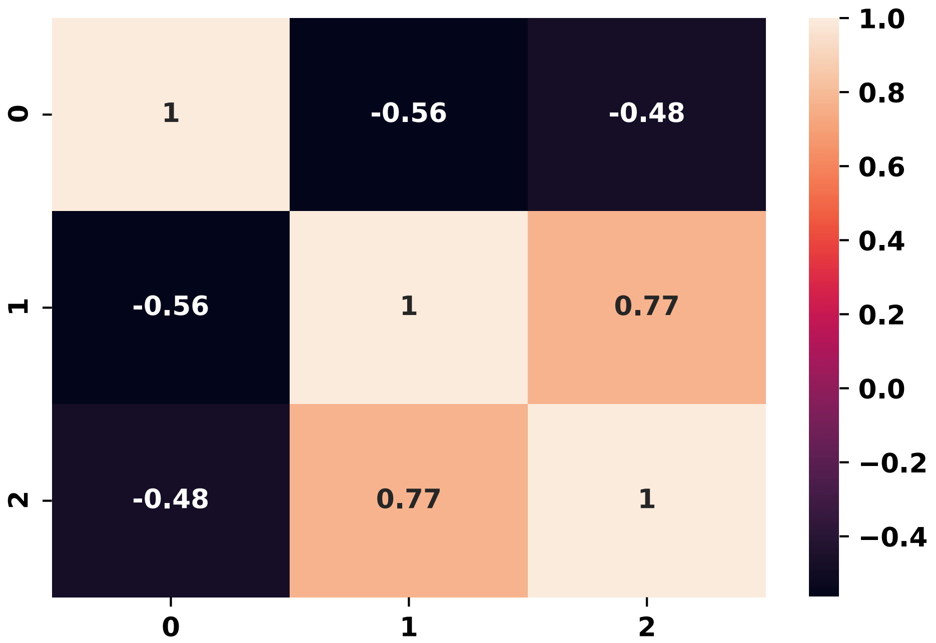
In [248]:

```
sns.heatmap(rfm_data.corr(),annot=True)
print('There is high positive correlation between Frequency and Monetary feature
s after applying Power transformation.')
```

Out[248]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x185e56fd0>
```

```
There is high positive correlation between Frequency and Monetary fe
atures after applying Power transformation.
```



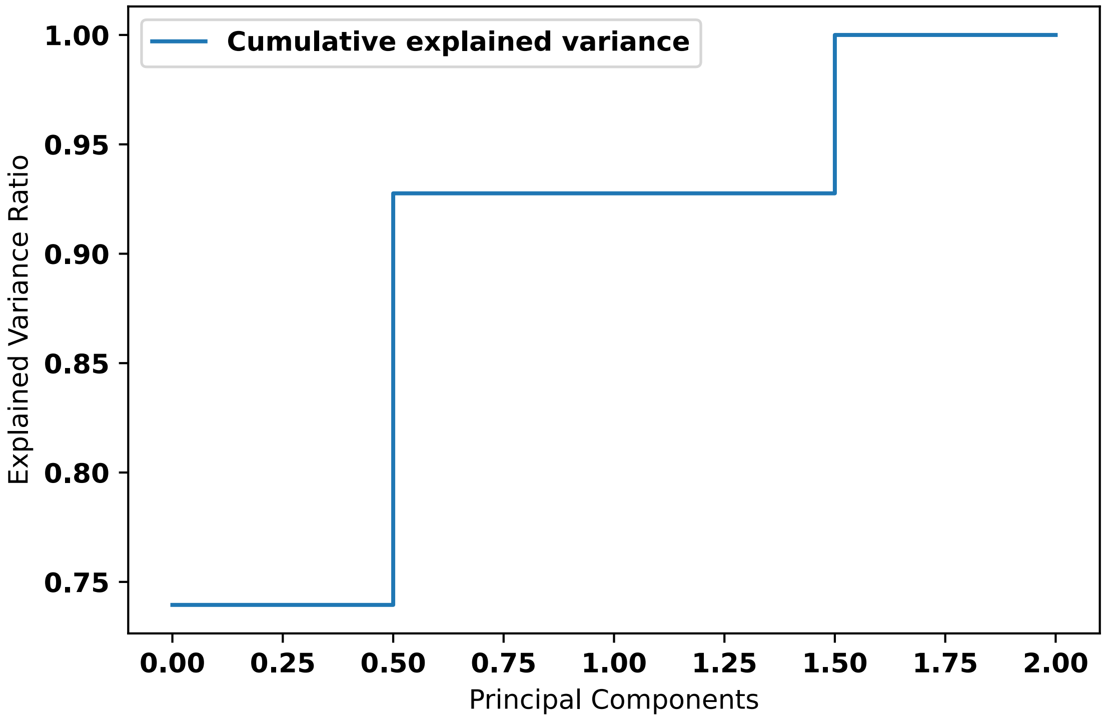
位# PCA

In [249]:

```
# 大多 Machine Learning 算法中，使用 StandardScaler 做特徵縮放，因為 MinMaxScaler 對異
常值較敏感。
# 在PCA，cluster，Logistic Regression，SVM，NN 等算法裡, StandardScaler 往往是最好的選
擇。
# MinMaxScaler 在不涉及距離度量、梯度、協方差以及數據需要被壓縮到特定區間時使用較多，比如數字
圖像使用MinMaxScaler将數據壓縮在[0,1]之間。
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
rfm_scaled = sc.fit_transform(rfm_data)


from sklearn.decomposition impor看暗降為後
pca = PCA()
看按將為後pca_transformed_data = pca.fit_transform(rfm_scaled)
```

In [251]:

```
# 看降維度後, 可解釋性
var_exp = pca.explained_variance_ratio_
_ = plt.figure(figsize=(6,4))
# plt.bar(range(3), var_exp, alpha=0.5, align='center', label='Individual explai
ned variance')
_ = plt.step(range(3), np.cumsum(var_exp), where='mid', label='Cumulative explai
ned variance')
_ = plt.ylabel('Explained Variance Ratio')
_ = plt.xlabel('Principal Components')
_ = plt.legend(loc='best')
_ = plt.tight_layout()
_ = plt.show()
```



In [252]:

```
# 選兩個特徵：
X = rfm_scaled.copy()
pca = PCA(n_components=2)
df_pca = pca.fit_transform(X)

df_pca = pd.DataFrame(df_pca)
df_pca.head()
```

Out[252]:

	0	1
0	-4.704521	-0.880061
1	-4.805591	-0.608291
2	-4.135941	-1.382083
3	-3.333856	0.055160
4	-4.515919	-0.706353

K- means

In [263]:

```
X = df_pca.copy()


from sklearn.cluster import KMeans

cluster_range = range(1, 15)
cluster_intertia = []
cluster_sil_scores = []

for num_clusters in cluster_range:
  clusters = KMeans( num_clusters, n_init = 100,init='k-means++',random_state=0)
  clusters.fit(X)
  labels = clusters.labels_                         # capture the cluster lables
  centroids = clusters.cluster_centers_             # capture the centroids
  cluster_intertia.append( clusters.inertia_ )      # capture the intertia

# combine the cluster_range and cluster_errors into a dataframe by combining the
m
clusters_df = pd.DataFrame({ "num_clusters":cluster_range, "cluster_intertia": c
luster_intertia} )
clusters_df[0:10]
```

Out[263]:

```
KMeans(n_clusters=1, n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_clusters=2, n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_clusters=3, n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_clusters=4, n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_clusters=5, n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_clusters=6, n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_clusters=7, n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_clusters=9, n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_clusters=10, n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_clusters=11, n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_clusters=12, n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_clusters=13, n_init=100, random_state=0)
```

Out[263]:

```
KMeans(n_clusters=14, n_init=100, random_state=0)
```

Out[263]:

	num_clusters	cluster_intertia
0	1	12074.654981
1	2	5276.164648
2	3	3869.345601
3	4	2892.728505
4	5	2307.360119
5	6	1919.882942
6	7	1697.645678
7	8	1539.444050
8	9	1393.334694
9	10	1268.514234

In [268]:

```
# Elbow plot
plt.figure(figsize=(12,6))
plt.plot(clusters_df['num_clusters'], clusters_df['cluster_intertia'], marker =
"o" )
plt.xlabel('Number of Clusters')
plt.ylabel('Cluster Errors')
```

Out[268]:

```
<Figure size 864x432 with 0 Axes>
```

Out[268]:

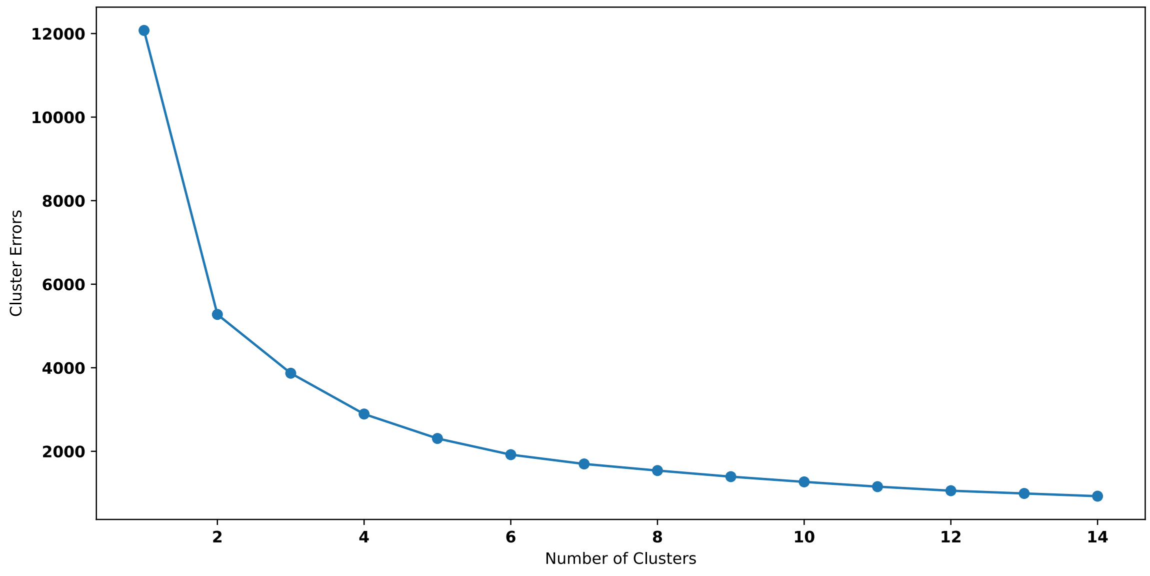
```
[<matplotlib.lines.Line2D at 0x1871c1ef0>]
```

Out[268]:

```
Text(0.5, 0, 'Number of Clusters')
```

Out[268]:

```
Text(0, 0.5, 'Cluster Errors')
```



In [269]:

```
for k in range(2,16):
    cluster = KMeans(n_clusters=k, random_state=0)
    labels = cluster.fit_predict(df_pca)

    sil_avg = silhouette_score(df_pca, labels)
    print('For',k,'clusters, average silhoutte score =',sil_avg)
```

```
For 2 clusters, average silhoutte score = 0.46740081776787407
For 3 clusters, average silhoutte score = 0.36728641819227253
For 4 clusters, average silhoutte score = 0.3848771782833163
For 5 clusters, average silhoutte score = 0.3816977939584511
For 6 clusters, average silhoutte score = 0.38197778145769046
For 7 clusters, average silhoutte score = 0.37118533218312866
For 8 clusters, average silhoutte score = 0.36187586738900696
For 9 clusters, average silhoutte score = 0.3566763805706895
For 10 clusters, average silhoutte score = 0.35570466164431186
For 11 clusters, average silhoutte score = 0.3583662484365603
For 12 clusters, average silhoutte score = 0.343229457767183
For 13 clusters, average silhoutte score = 0.3405816713290833
For 14 clusters, average silhoutte score = 0.34234023773479394
For 15 clusters, average silhoutte score = 0.3382226170245794
```

In [272]:

```
kmeans = KMeans(n_clusters=4)
kmeans = kmeans.fit(df_pca)
labels = kmeans.predict(df_pca)
centroids = kmeans.cluster_centers_


df_pca['Cluster'] = labels
df_pca.head()

df_pca['Cluster'].value_counts()

sns.pairplot(df_pca,diag_kind='kde',hue='Cluster')
```

Out[272]:

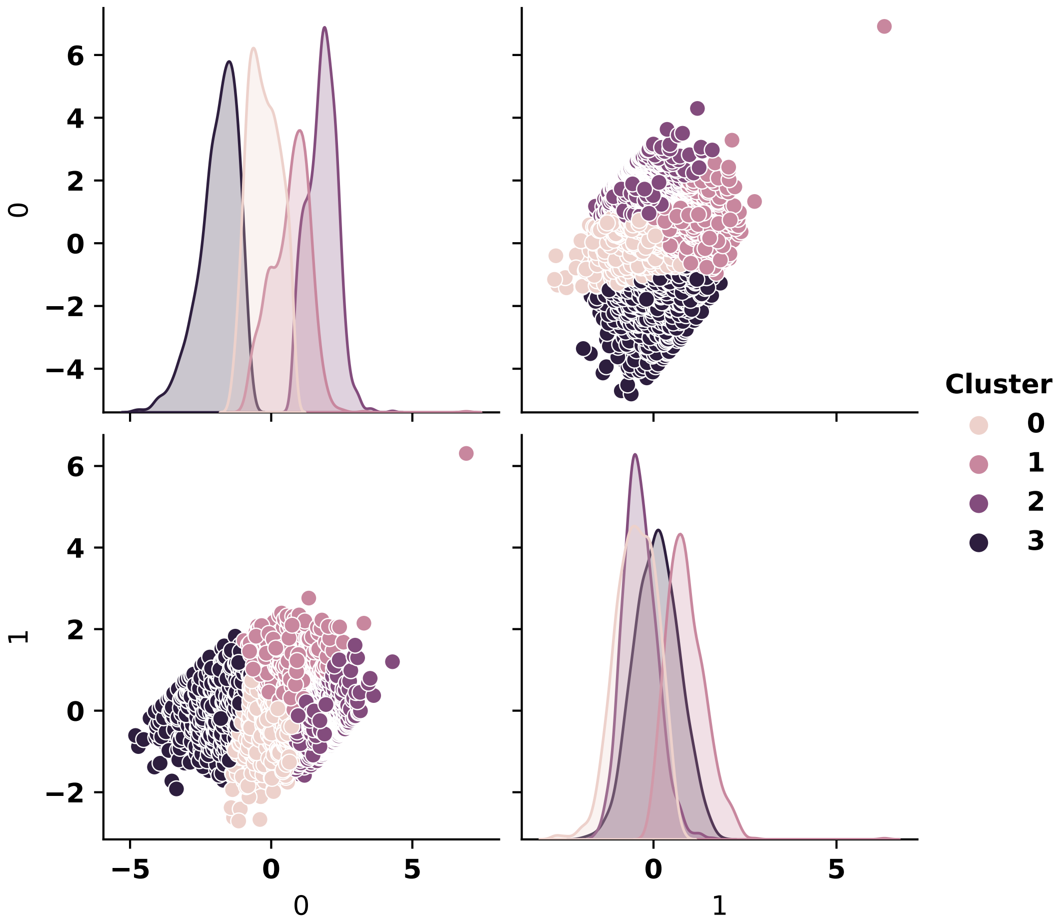
	0	1	Cluster
0	-4.704521	-0.880061	3
1	-4.805591	-0.608291	3
2	-4.135941	-1.382083	3
3	-3.333856	0.055160	3
4	-4.515919	-0.706353	3

Out[272]:

```
0    1197
3    1178
2    1032
1     932
Name: Cluster, dtype: int64
```

Out[272]:

```
<seaborn.axisgrid.PairGrid at 0x187114da0>
```



In [273]:

```
# 客戶分群：
customers_grouped = pd.DataFrame(pt.inverse_transform(rfm_data),columns=rfm_data
.columns,index=rfm_df.index)
customers_grouped['Cluster'] = df_pca['Cluster'].values
customers_grouped['RFMScore'] = rfm_segmentation['RFMScore'].values
customers_grouped.head()
```

Out[273]:

CustomerID	0	1	2	Cluster	RFMScore
14646.0	-1.928590	2.228021	4.307308	3	444
18102.0	-2.225967	2.189298	4.257275	3	444
17450.0	-1.159197	2.130843	4.066576	3	444
16446.0	-2.225967	-0.195479	3.970866	3	424
14911.0	-1.928590	2.348232	3.865187	3	444

In [274]:

```
# 客戶分群： Top_spenders and loyal_customers
top_spenders_and_loyal_customers = customers_grouped[(customers_grouped['RFMScor
e'] == '444') | (customers_grouped['RFMScore'] == '443') | (customers_grouped['R
FMScore'] == '434')]
top_spenders_and_loyal_customers
```

Out[274]:

CustomerID	0	1	2	Cluster	RFMScore
14646.0	-1.928590	2.228021	4.307308	3	444
18102.0	-2.225967	2.189298	4.257275	3	444
17450.0	-1.159197	2.130843	4.066576	3	444
14911.0	-1.928590	2.348232	3.865187	3	444
14156.0	-1.098117	2.171392	3.728681	3	444
...	...	...	...	...	...
14289.0	-1.159197	1.067664	0.197468	3	443
13991.0	-0.941513	1.067664	0.148446	3	443
16877.0	-0.941513	1.067664	0.146383	3	443
16600.0	-1.481860	1.067664	0.108540	3	443
17114.0	-1.384970	1.309999	0.103797	3	443

607 rows × 5 columns

In [276]:

```
# 客戶分群： customers_churned
customers_churned = customers_grouped[(customers_grouped['RFMScore'] == '111') |
(customers_grouped['RFMScore'] == '112') | (customers_grouped['RFMScore'] == '12
1')]
customers_churned
```

Out[276]:

CustomerID	0	1	2	Cluster	RFMScore
14603.0	1.449926	-1.158092	-0.056190	2	112
17046.0	0.925617	-1.158092	-0.078025	2	112
14000.0	1.198027	-1.158092	-0.085900	2	112
16022.0	1.398503	-1.158092	-0.095141	2	112
14036.0	1.485325	-1.158092	-0.097197	2	112
...	...	...	...	...	...
17102.0	1.402252	-1.158092	-2.759849	2	111
15823.0	1.758542	-1.158092	-3.226808	2	111
17763.0	1.409714	-1.158092	-3.226808	2	111
17956.0	1.356444	-1.158092	-3.371326	2	111
16738.0	1.529812	-1.158092	-4.482103	2	111

729 rows × 5 columns

In [277]:

```
# 客戶分群： customers_at_risk_of_churning
customers_at_risk_of_churning = customers_grouped[(customers_grouped['RFMScore']
== '144') | (customers_grouped['RFMScore'] == '143') | (customers_grouped['RFMSc
ore'] == '134') | (customers_grouped['RFMScore'] == '133') | (customers_grouped[
'RFMScore'] == '142') | (customers_grouped['RFMScore'] == '124')]
customers_at_risk_of_churning
```

Out[277]:

	0	1	2	Cluster	RFMScore
CustomerID					
15749.0	1.300601	0.327537	3.065714	0	134
15098.0	1.060140	0.327537	2.989581	0	134
12590.0	1.198027	-0.195479	1.994545	0	124
13093.0	1.453514	1.309999	1.826224	0	144
12980.0	0.925617	1.398537	1.782096	0	144
...	...	...	...	...	...
15574.0	1.034530	0.661012	-0.007203	0	133
16997.0	1.499192	0.327537	-0.007484	0	133
17874.0	1.075224	1.067664	-0.028603	0	143
16306.0	1.206939	1.067664	-0.090131	0	142
15107.0	1.575930	1.067664	-0.635915	0	142

94 rows × 5 columns

In [279]:

```
# 客戶分群： new_customers or avg_spenders

new_customers_or_avg_spenders = customers_grouped[(customers_grouped['RFMScore'] == '422') | (customers_grouped['RFMScore'] == '411') | (customers_grouped['RFMScore'] == '412') | (customers_grouped['RFMScore'] == '421') | (customers_grouped['RFMScore'] == '413') | (customers_grouped['RFMScore'] == '431')]

new_customers_or_avg_spenders
```

Out[279]:

	0	1	2	Cluster	RFMScore
CustomerID					
16800.0	-0.989982	-1.158092	0.412886	1	413
17727.0	-0.812876	-1.158092	0.315678	1	413
12713.0	-2.225967	-1.158092	0.141620	1	413
14756.0	-0.812876	-1.158092	0.032954	1	413
12478.0	-1.596697	-1.158092	-0.031461	1	413
...	...	...	...	...	...
14865.0	-1.226283	-0.195479	-2.141213	1	421
18184.0	-0.812876	-1.158092	-2.181444	1	411
15992.0	-1.596697	-1.158092	-2.327735	1	411
16856.0	-0.853313	-1.158092	-2.474908	1	411
13256.0	-0.853313	-1.158092	-10.506459	1	411

211 rows × 5 columns

Modelling

Logistic Regression

In [280]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, plot_roc_curve
```

In [281]:

```
y = df_pca['Cluster']
```

In [282]:

```
X_train, X_test, y_train, y_test = train_test_split(df_pca, y, test_size=0.3, ra
ndom_state=42, stratify=y)

lr = LogisticRegression(max_iter=1000,random_state=0)
lr.fit(X_train, y_train)
```

Out[282]:

```
LogisticRegression(max_iter=1000, random_state=0)
```

In [283]:

```
y_test_predicted = lr.predict(X_test)
y_train_predicted = lr.predict(X_train)
```

In [289]:

```
accuracy_train = accuracy_score(y_train, y_train_predicted)
accuracy_test = accuracy_score(y_test, y_test_predicted)
print('Train Set Accuracy for Power Transformed Data:',round(accuracy_train*100,
2),'%')
print('Test Set Accuracy for Power Transformed Data:',round(accuracy_test*100,2
),'%')

kf= KFold(shuffle=True, n_splits=5, random_state=0)
score = cross_val_score(lr, df_pca, y, cv=kf, scoring='f1_weighted')
print('Bias Error:',1-np.mean(score))
print('Variance Error:',np.std(score,ddof=1))

cm = confusion_matrix(y_test, y_test_predicted)
print()
print('confusion_matrix:\n' ,cm)
print()
print(classification_report(y_test,y_test_predicted))
```

```
Train Set Accuracy for Power Transformed Data: 100.0 %
Test Set Accuracy for Power Transformed Data: 100.0 %
Bias Error: 0.0
Variance Error: 0.0

confusion_matrix:
 [[359   0   0   0]
 [  0 280   0   0]
 [  0   0 310   0]
 [  0   0   0 353]]

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       359
           1       1.00      1.00      1.00       280
           2       1.00      1.00      1.00       310
           3       1.00      1.00      1.00       353

    accuracy                           1.00      1302
   macro avg       1.00      1.00      1.00      1302
weighted avg       1.00      1.00      1.00      1302
```

Naive Bayes

In [290]:

```
from sklearn.naive_bayes import GaussianNB
```

In [291]:

```
nb = GaussianNB()
score = cross_val_score(nb, df_pca, y, cv=kf, scoring='f1_weighted')
print('Bias Error:',1-np.mean(score))
print('Variance Error:',np.std(score,ddof=1))

nb.fit(X_train,y_train)

y_train_predicted = nb.predict(X_train)
y_test_predicted = nb.predict(X_test)

accuracy_train = accuracy_score(y_train, y_train_predicted)
accuracy_test = accuracy_score(y_test, y_test_predicted)

print('Train Set Accuracy for Power Transformed Data:',round(accuracy_train*100,
2),'%')
print('Test Set Accuracy for Power Transformed Data:',round(accuracy_test*100,2
),'%')

print(confusion_matrix(y_test, y_test_predicted))

print(classification_report(y_test, y_test_predicted))
```

```
Bias Error: 0.0
Variance Error: 0.0
```

Out[291]:

```
GaussianNB()
```

```
Train Set Accuracy for Power Transformed Data: 100.0 %
Test Set Accuracy for Power Transformed Data: 100.0 %
[[359   0   0   0]
 [  0 280   0   0]
 [  0   0 310   0]
 [  0   0   0 353]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       359
           1       1.00      1.00      1.00       280
           2       1.00      1.00      1.00       310
           3       1.00      1.00      1.00       353

    accuracy                           1.00      1302
   macro avg       1.00      1.00      1.00      1302
weighted avg       1.00      1.00      1.00      1302
```

In [ ]:

```

```