In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:
```
df = pd.read_excel("Online Retail.xlsx")
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(541909, 8)
```

In [ ]:
```
df.head()
```

Out[ ]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

In [ ]:
```
df = df[~df["StockCode"].str.contains('C', na=False)]
df = df[(df.Quantity>0) & (df.UnitPrice> 0)]
df.drop_duplicates(inplace=True)
```

In [ ]:
```
print(df.isnull().sum())
df.dropna(inplace=True)
```

```
InvoiceNo           0
StockCode           0
Description         0
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     129925
Country             0
dtype: int64
```

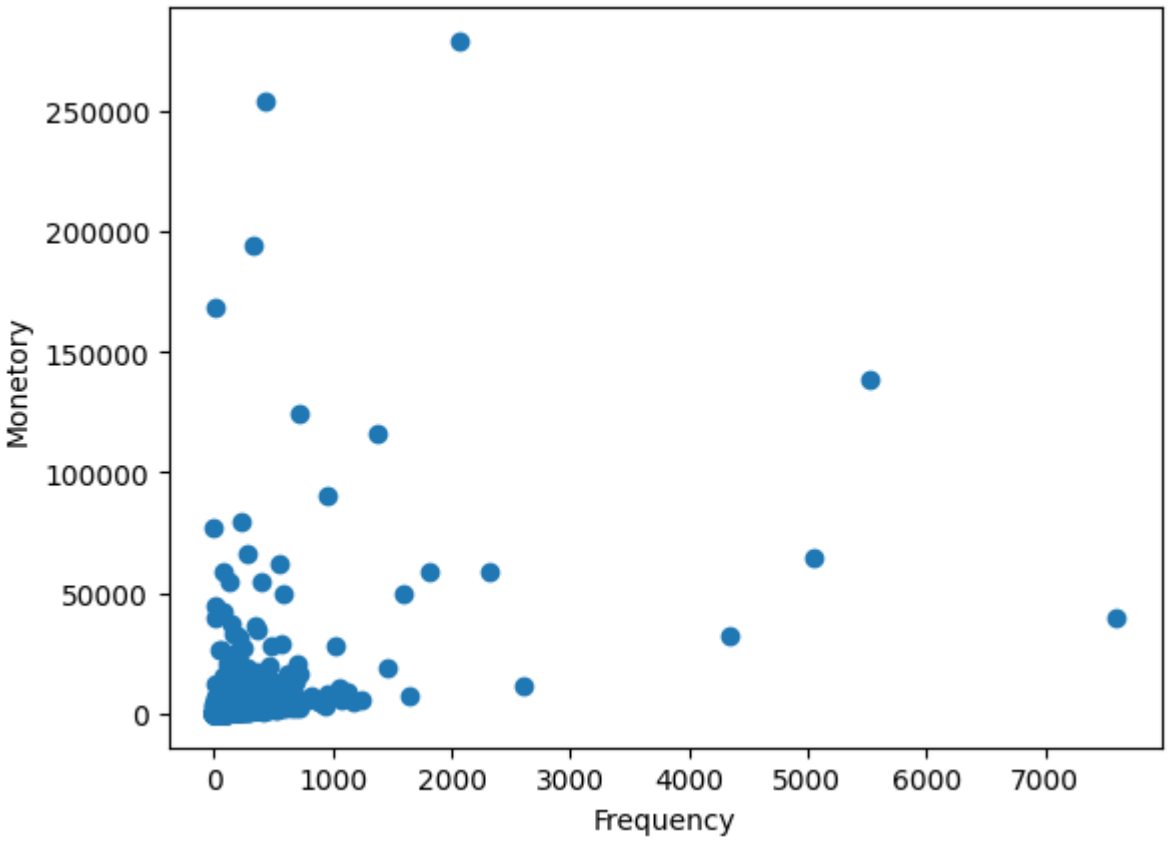
In [ ]:
```
import datetime
df["TotalAmount"] = df["UnitPrice"] * df["Quantity"]
first_date = df["InvoiceDate"].min()
last_date = df["InvoiceDate"].max()
today = last_date.to_pydatetime() + datetime.timedelta(days=1)
```

In [ ]:
```
rfm_df = df.groupby(["CustomerID"]).agg({'InvoiceNo': lambda x: len(x),
                                         'TotalAmount': lambda price: price.sum(),
                                         'InvoiceDate': lambda date: (today - date.dt.to_pydatetime().max()).days})\
                                         .reset_index().sort_values('InvoiceNo')
```

In [ ]:
```
rfm_df.columns = ["CustomerID", "Frequency", "Monetary", "Recency"]
```

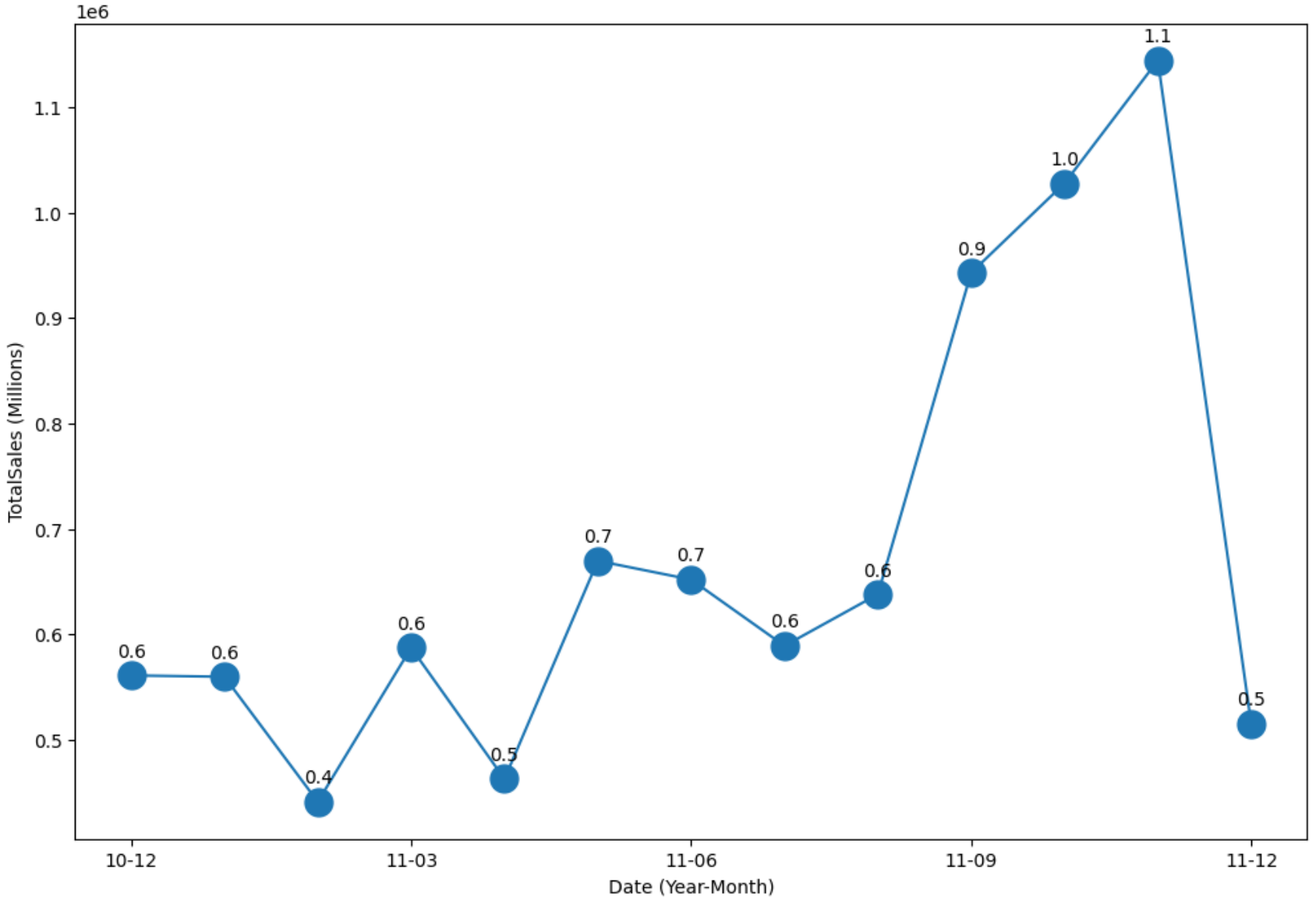
In [ ]:
```
rfm_df.reset_index(inplace=True)
```

In [ ]:
```
%matplotlib inline
plt.scatter(x=rfm_df["Frequency"], y=rfm_df["Monetary"])
plt.xlabel("Frequency")
plt.ylabel("Monetory")
plt.show()
```



In [ ]:
```
df["TrimmedDate"] = df["InvoiceDate"].dt.strftime('%y-%m')
df_trend = df.groupby('TrimmedDate').agg({'TotalAmount': lambda x: x.sum()}).reset_index().sort_values(by='TrimmedDat
```

In [ ]:
```
%matplotlib inline
fig = plt.figure(figsize=(12, 8))
plt.plot(df_trend["TrimmedDate"], df_trend["TotalAmount"], marker='o', ms=15)
plt.xlabel('Date (Year-Month)')
plt.ylabel('TotalSales (Millions)')
plt.xticks(['10-12', '11-03', '11-06', '11-09', '11-12'])
for i, (amount, date) in df_trend.iterrows():
    plt.annotate("{:.1f}".format(date/1e+6), (amount, date), textcoords="offset points", ha='center', xytext=(0,10))
plt.show()
```



In [ ]:
```
rfm_df["R"] = pd.qcut(rfm_df["Recency"], 5, labels=[5, 4, 3, 2, 1])
rfm_df["M"] = pd.qcut(rfm_df["Monetary"], 5, labels=[1, 2, 3, 4, 5])
rfm_df["F"] = pd.qcut(rfm_df["Frequency"], 5, labels=[1, 2, 3, 4, 5])
```

In [ ]:
```
rfm_df["RFM"] = rfm_df["R"].astype("str") + rfm_df["F"].astype("str") + rfm_df["M"].astype("str")
```

In [ ]:
```
from sklearn.cluster import KMeans

df_cluster = rfm_df[["CustomerID","Recency", "Frequency", "Monetary"]]

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(df_cluster)


df_cluster["Cluster"] = kmeans.fit_predict(df_cluster)
```

```
f:\Projects\.venv\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
f:\Projects\.venv\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` wil
l change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\ASUS\AppData\Local\Temp\ipykernel_17336\481568439.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
  df_cluster["Cluster"] = kmeans.fit_predict(df_cluster)
```

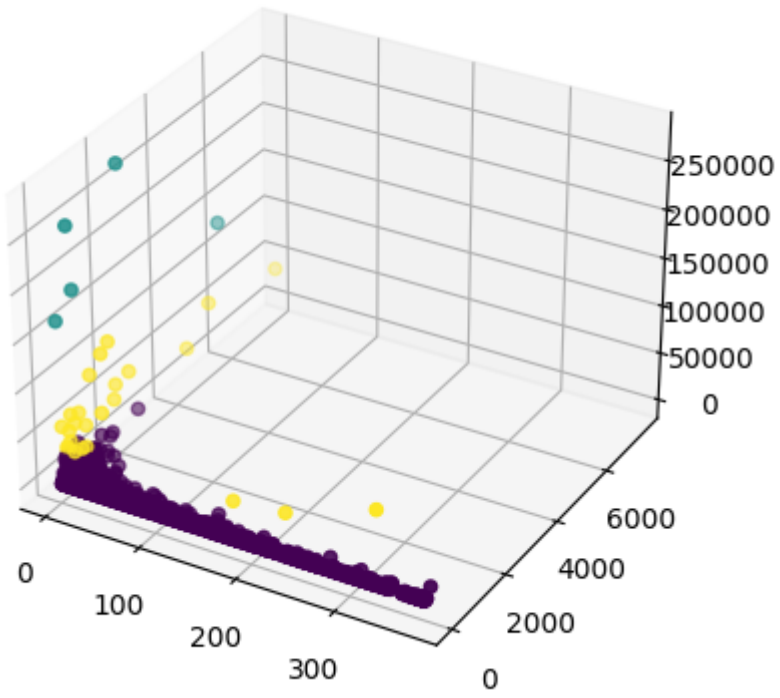
In [ ]:
```
df_cluster
```

Out[ ]:

	CustomerID	Recency	Frequency	Monetary	Cluster
0	12346.0	326	1	77183.60	2
1	18113.0	369	1	76.32	0
2	15118.0	134	1	244.80	0
3	13747.0	374	1	79.60	0
4	18133.0	212	1	931.50	0
...	...	...	...	...	...
4328	14606.0	1	2603	11718.65	0
4329	12748.0	1	4345	32509.54	2
4330	14096.0	4	5059	64824.04	2
4331	14911.0	1	5529	138460.32	1
4332	17841.0	2	7604	40177.67	2

4333 rows × 5 columns

In [ ]:
```
%matplotlib inline

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter3D(df_cluster["Recency"], df_cluster["Frequency"], df_cluster["Monetary"], c=df_cluster["Cluster"])
plt.show()
```



In [ ]:
```
df_cluster.to_csv('rfm.csv')
```

In [ ]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = Axes3D(fig)
```

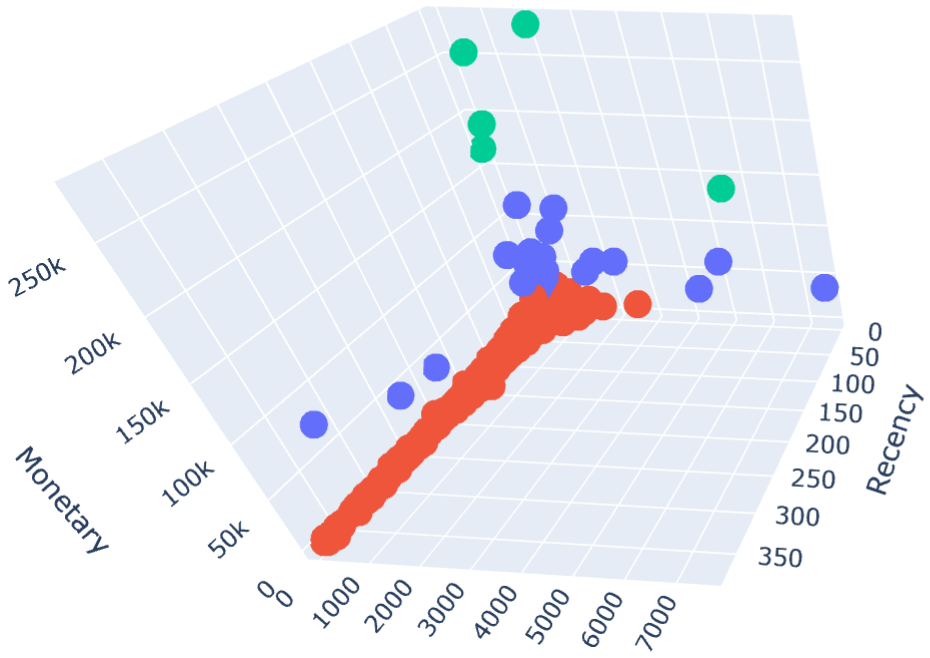
```
<Figure size 640x480 with 0 Axes>
```

In [ ]:
```
empty_arr = np.zeros(shape=(df.shape[0]))
cluster_detail = pd.Series(empty_arr, name='ClusterName')
df_cluster = pd.concat([df_cluster, cluster_detail], axis=1)
for idx, row in df_cluster.iterrows():
    if row[-2] == 1:
        df_cluster.iloc[idx, -1] = 'legendary'
    elif row[-2] == 2:
        df_cluster.iloc[idx, -1] = 'potential loyalist'
```

```
    else:
        df_cluster.iloc[idx, -1] ='hibernate'
```

In [ ]:

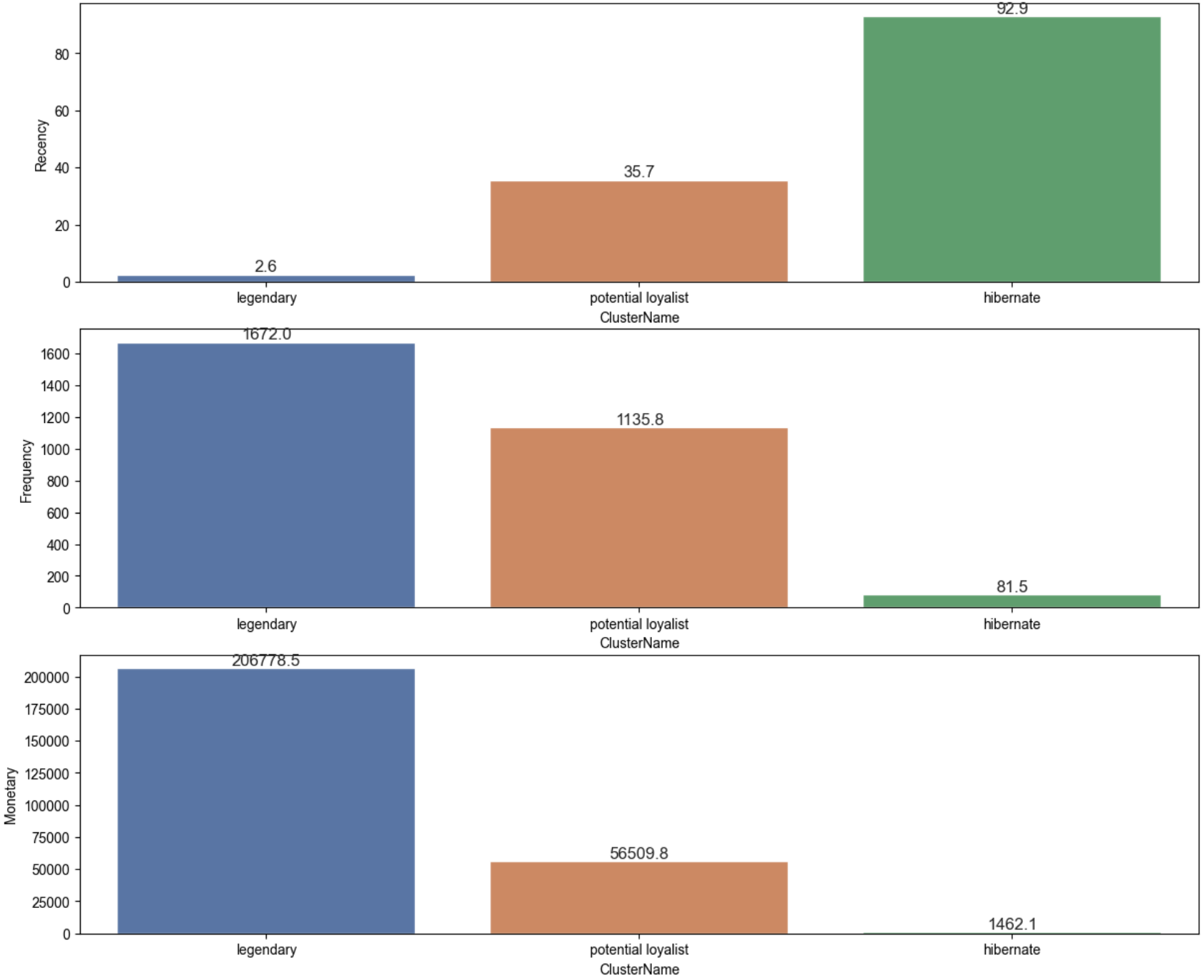
```
import plotly.express as px
from plotly.offline import init_notebook_mode
%matplotlib inline

init_notebook_mode(connected=True)
fig = px.scatter_3d(df_cluster, x='Recency', y='Frequency', z='Monetary',
              color='ClusterName')
fig.update_layout(showlegend=True)
fig.show()
```



In [ ]:
```
df_analysis = df_cluster.groupby('ClusterName').agg({'Recency': lambda r: r.mean(),
                                                     'Frequency': lambda f: f.mean(),
                                                     'Monetary': lambda m: m.mean()}).sort_values(by='Recency').reset_index()
```

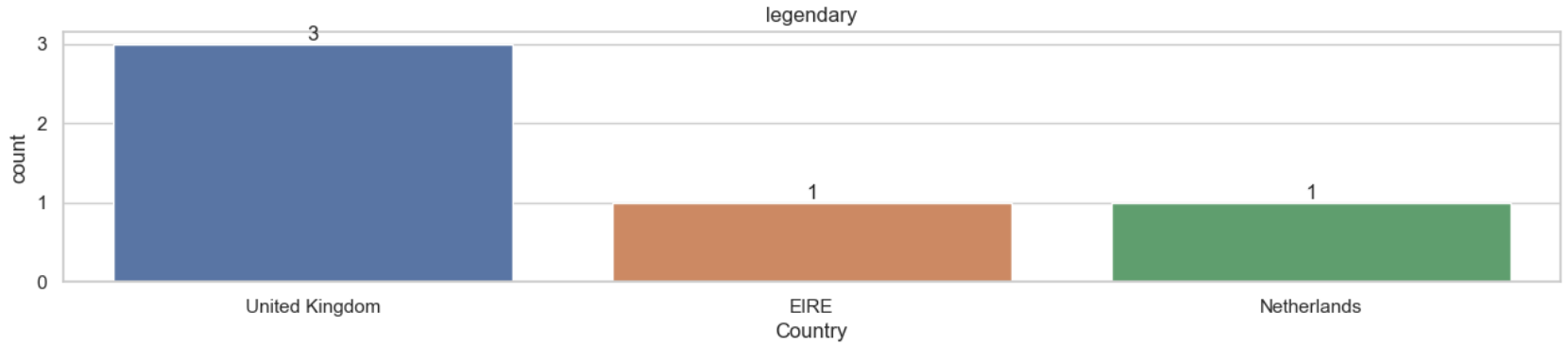
In [ ]:
```
import seaborn as sns

figure, axes = plt.subplots(3, 1, figsize=(12,10), )
sns.set_theme(style='whitegrid')
```

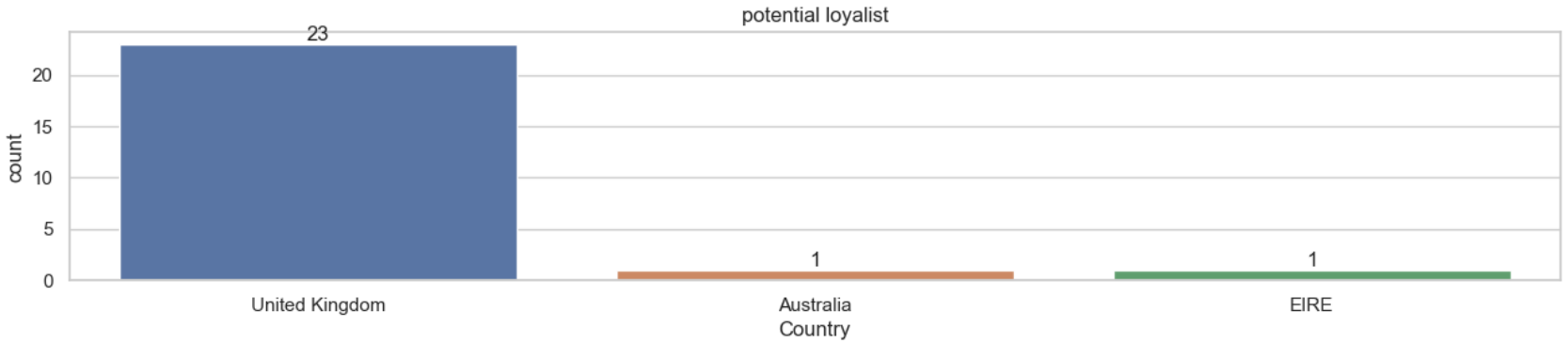
```
col_list = ['Recency', 'Frequency', 'Monetary']
plt.tight_layout()
for i, ax in enumerate(axes):
    axis = sns.barplot(data=df_analysis, x='ClusterName', y=col_list[i], ax=axes[i])
    axis.bar_label(axis.containers[0], fmt='%.1f')

plt.show()
```

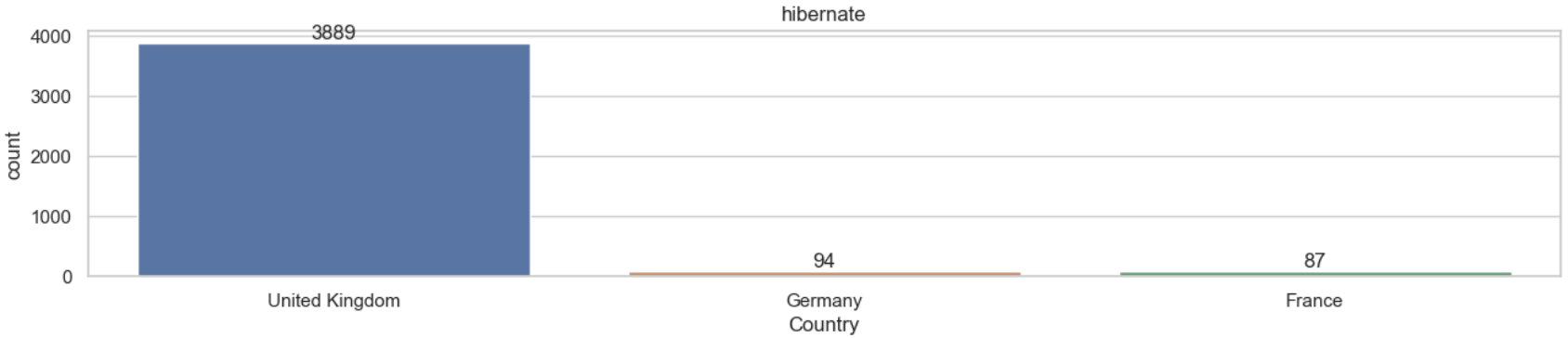

Recency
0
20
40
60
80
2.6
35.7
92.9
legendary
potential loyalist
hibernate
ClusterName
Frequency
0
200
400
600
800
1000
1200
1400
1600
1672.0
1135.8
81.5
legendary
potential loyalist
hibernate
ClusterName
Monetary
0
25000
50000
75000
100000
125000
150000
175000
200000
206778.5
56509.8
1462.1
legendary
potential loyalist
hibernate
ClusterName

In [ ]:

```
df_total = pd.merge(right=df, left=df_cluster, on='CustomerID')
df_total.head()
df_total.drop_duplicates('CustomerID', inplace=True)
fig, axes = plt.subplots(3, 1, figsize=(12, 8))
cluster_names = ['legendary', 'potential loyalist', 'hibernate']
plt.tight_layout()
plt.subplots_adjust(left=1,
                    bottom=1,
                    right=2,
                    top=2,
                    wspace=0.1,
                    hspace=0.5)
for i, cluster in enumerate(cluster_names):
    temp = df_total[df_total['ClusterName'] == cluster].value_counts('Country').reset_index().sort_values(by='count',
    axis = sns.barplot(data=temp, x='Country', y='count', ax=axes[i])
    axis.bar_label(axis.containers[0])
    axes[i].set_title(cluster)
plt.show()
```


legendary
3
1
1
count
United Kingdom
EIRE
Netherlands
Country


potential loyalist
23
1
1
count
United Kingdom
Australia
EIRE
Country


hibernate
3889
94
87
count
United Kingdom
Germany
France
Country

In [ ]: