In [1]:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.metrics import silhouette_score
import numpy as np
```

In [3]:

```python
# Load the dataset
file_path = r'C:\Users\zahus\Desktop\DATA science\Module 11-Dissertaion\Dataset\2-Customer Segmentation & Behaviour Data/Online
df = pd.read_csv(file_path, encoding='ISO-8859-1')
```

In [4]:

```python
# Display the first few rows of the dataset
df.head()
```

Out[4]:

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

In [5]:

```python
# Step 1: Data Cleaning
# Drop rows with missing CustomerID
df_cleaned = df.dropna(subset=['CustomerID'])
```

In [6]:

```python
# Convert InvoiceDate to datetime
df_cleaned['InvoiceDate'] = pd.to_datetime(df_cleaned['InvoiceDate'])
```

```
c:\users\zahus\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:2: 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-vers
us-a-copy
```

In [7]:

```
# Remove negative or zero quantities and prices (returns or errors)
df_cleaned = df_cleaned[(df_cleaned['Quantity'] > 0) & (df_cleaned['UnitPrice'] > 0)]
```

In [8]:

```
# Create TotalPrice = Quantity * UnitPrice
df_cleaned['TotalPrice'] = df_cleaned['Quantity'] * df_cleaned['UnitPrice']
```

In [9]:

```
# Step 2: Basic EDA
summary_stats = df_cleaned[['Quantity', 'UnitPrice', 'TotalPrice']].describe()
```

In [10]:

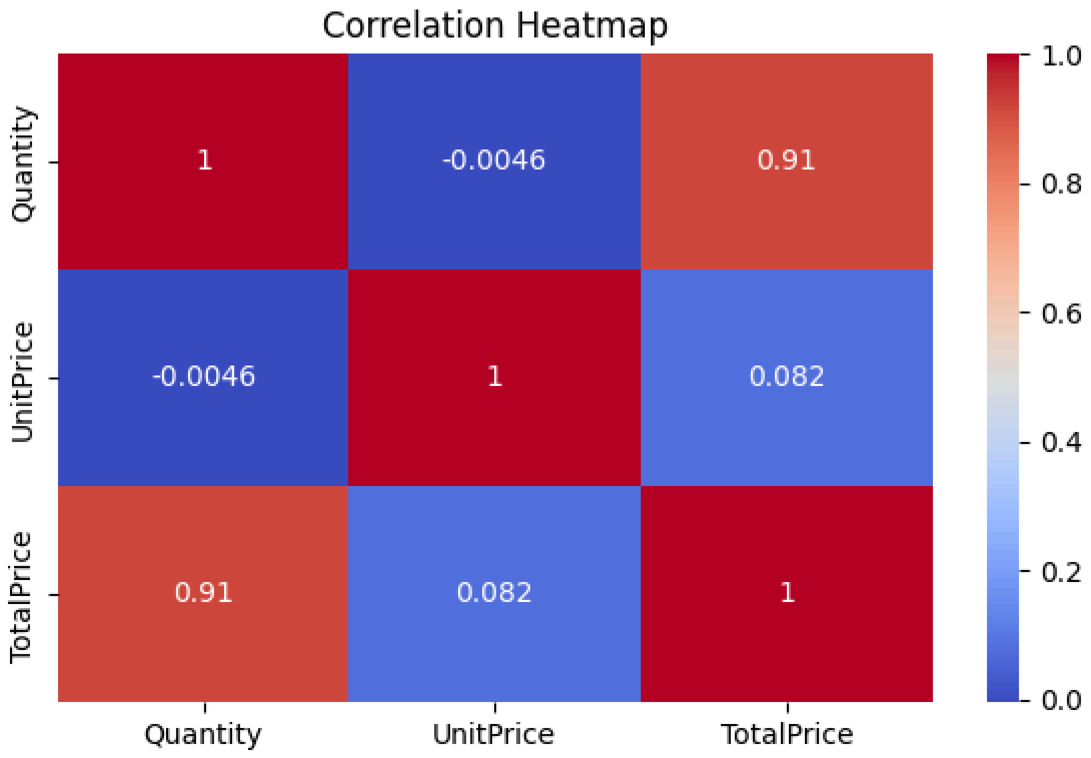
```
# Step 3: Correlation Heatmap
correlation_matrix = df_cleaned[['Quantity', 'UnitPrice', 'TotalPrice']].corr()
```

In [11]:

```
# Plotting correlation heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.tight_layout()

summary_stats
```

Out[11]:

	Quantity	UnitPrice	TotalPrice
count	397884.000000	397884.000000	397884.000000
mean	12.988238	3.116488	22.397000
std	179.331775	22.097877	309.071041
min	1.000000	0.001000	0.001000
25%	2.000000	1.250000	4.680000
50%	6.000000	1.950000	11.800000
75%	12.000000	3.750000	19.800000
max	80995.000000	8142.750000	168469.600000



Next steps:

Create RFM features (Recency, Frequency, Monetary)

Preprocess for clustering (e.g. scaling, encoding)

Apply clustering models (K-Means, DBSCAN, Hierarchical)

Visualise and compare clustering performance

In [12]:

```
# Reference date for recency calculation (assume max date in dataset)
latest_date = df_cleaned['InvoiceDate'].max()
```

In [13]:

```
# Step 1: Calculate RFM features for each customer
rfm = df_cleaned.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (latest_date - x.max()).days,  # Recency
    'InvoiceNo': 'nunique',  # Frequency
    'TotalPrice': 'sum'  # Monetary
}).reset_index()
```

In [14]:

```
# Rename columns
rfm.columns = ['CustomerID', 'Recency', 'Frequency', 'Monetary']
```

In [15]:

```
# Step 2: Check RFM stats
rfm_stats = rfm.describe()

rfm_stats
```

Out[15]:

	CustomerID	Recency	Frequency	Monetary
count	4338.000000	4338.000000	4338.000000	4338.000000
mean	15300.408022	105.470954	4.272015	2054.266460
std	1721.808492	115.082161	7.697998	8989.230441
min	12346.000000	0.000000	1.000000	3.750000
25%	13813.250000	22.000000	1.000000	307.415000
50%	15299.500000	61.000000	2.000000	674.485000
75%	16778.750000	161.750000	5.000000	1661.740000
max	18287.000000	697.000000	209.000000	280206.020000

I'll proceed to:

Scale the RFM data

Apply clustering algorithms: K-Means, Hierarchical Clustering, and DBSCAN

Evaluate & visualise the results

In [16]:

```
# Step 1: Scale the RFM features
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm[['Recency', 'Frequency', 'Monetary']])
```

In [17]:

```
# Step 2: K-Means Clustering (try 2 to 6 clusters and select the best with Silhouette Score)
kmeans_scores = {}
for k in range(2, 7):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(rfm_scaled)
    score = silhouette_score(rfm_scaled, labels)
    kmeans_scores[k] = score
```

In [18]:

```
# Step 3: Best K-Means model
best_k = max(kmeans_scores, key=kmeans_scores.get)
```

```
kmeans_final = KMeans(n_clusters=best_k, random_state=42, n_init=10)
rfm['KMeans_Cluster'] = kmeans_final.fit_predict(rfm_scaled)
```

In [19]:

```
# Step 4: Hierarchical Clustering
hierarchical = AgglomerativeClustering(n_clusters=best_k)
rfm['Hierarchical_Cluster'] = hierarchical.fit_predict(rfm_scaled)
```

In [20]:

```
# Step 5: DBSCAN (density-based, may not need pre-defined clusters)
dbscan = DBSCAN(eps=1.5, min_samples=5)
rfm['DBSCAN_Cluster'] = dbscan.fit_predict(rfm_scaled)
```

Next up:

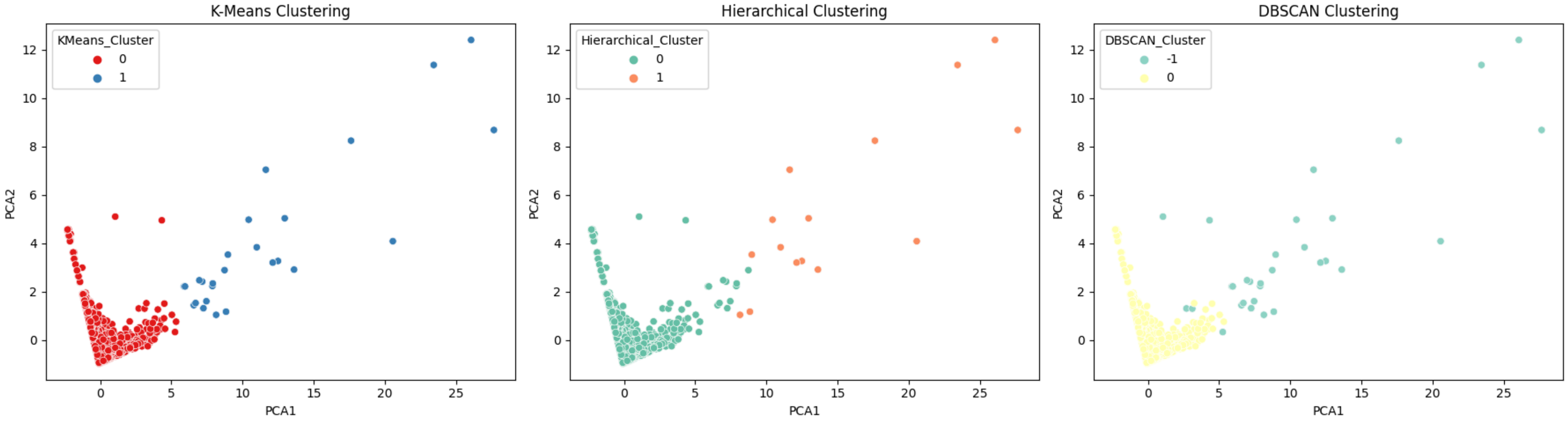
Visualise clustering results (2D projection using PCA or t-SNE)

Show cluster statistics for each method

In [21]:

```
from sklearn.decomposition import PCA

# Reduce to 2D using PCA for visualisation
pca = PCA(n_components=2)
rfm_pca = pca.fit_transform(rfm_scaled)

# Add PCA components to the dataframe
rfm['PCA1'] = rfm_pca[:, 0]
rfm['PCA2'] = rfm_pca[:, 1]

# Plotting
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# K-Means
sns.scatterplot(ax=axes[0], data=rfm, x='PCA1', y='PCA2', hue='KMeans_Cluster', palette='Set1')
axes[0].set_title('K-Means Clustering')

# Hierarchical
sns.scatterplot(ax=axes[1], data=rfm, x='PCA1', y='PCA2', hue='Hierarchical_Cluster', palette='Set2')
axes[1].set_title('Hierarchical Clustering')

# DBSCAN
```

```
sns.scatterplot(ax=axes[2], data=rfm, x='PCA1', y='PCA2', hue='DBSCAN_Cluster', palette='Set3')
axes[2].set_title('DBSCAN Clustering')

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



cluster comparison table with average Recency, Frequency, and Monetary values for each clustering model:

In [22]:

```
# Clean and preprocess
df_cleaned = df.dropna(subset=['CustomerID'])
df_cleaned['InvoiceDate'] = pd.to_datetime(df_cleaned['InvoiceDate'])
df_cleaned = df_cleaned[(df_cleaned['Quantity'] > 0) & (df_cleaned['UnitPrice'] > 0)]
df_cleaned['TotalPrice'] = df_cleaned['Quantity'] * df_cleaned['UnitPrice']

# Create RFM table
latest_date = df_cleaned['InvoiceDate'].max()
rfm = df_cleaned.groupby('CustomerID').agg({
    'InvoiceDate': lambda x: (latest_date - x.max()).days,
    'InvoiceNo': 'nunique',
    'TotalPrice': 'sum'
}).reset_index()
rfm.columns = ['CustomerID', 'Recency', 'Frequency', 'Monetary']

# Scale the RFM features
```

```
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm[['Recency', 'Frequency', 'Monetary']])

# Apply clustering models
rfm['KMeans_Cluster'] = KMeans(n_clusters=3, random_state=42, n_init=10).fit_predict(rfm_scaled)
rfm['Hierarchical_Cluster'] = AgglomerativeClustering(n_clusters=3).fit_predict(rfm_scaled)
rfm['DBSCAN_Cluster'] = DBSCAN(eps=1.5, min_samples=5).fit_predict(rfm_scaled)

# Combine RFM summaries without GMM
rfm_comparison_table= pd.concat({
    'KMeans': rfm.groupby('KMeans_Cluster')[['Recency', 'Frequency', 'Monetary']].mean().round(2),
    'Hierarchical': rfm.groupby('Hierarchical_Cluster')[['Recency', 'Frequency', 'Monetary']].mean().round(2),
    'DBSCAN': rfm[rfm['DBSCAN_Cluster'] != -1].groupby('DBSCAN_Cluster')[['Recency', 'Frequency', 'Monetary']].mean().round(2)
}, axis=0)

print(rfm_comparison_table)
```

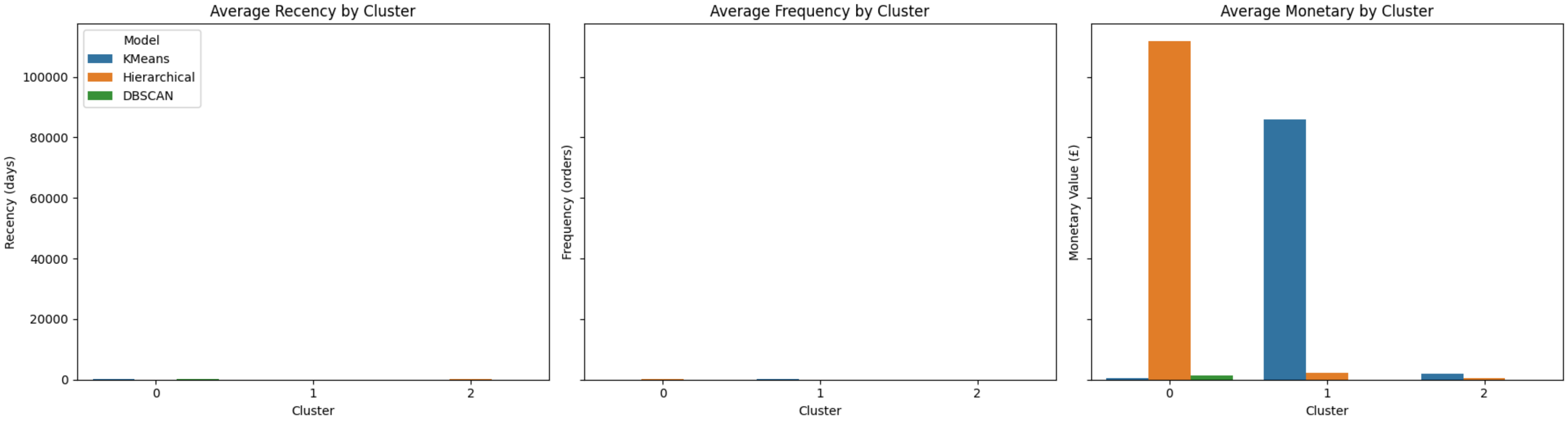
```
c:\users\zahus\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers
us-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
                Recency  Frequency   Monetary
KMeans       0   260.92       1.55     569.48
             1    14.12      66.42   85904.35
             2    47.80       4.78    1916.73
Hierarchical 0    12.73      83.47  111916.31
             1    36.80       5.27    2298.14
             2   225.93       1.79     584.64
DBSCAN       0   105.89       3.88    1509.72
```

In [24]:

```
# Prepare data for visualisation
rfm_plot = rfm_comparison_table.reset_index().rename(columns={'level_0': 'Model', 'level_1': 'Cluster'})

# Set up the figure
fig, axes = plt.subplots(1, 3, figsize=(18, 5), sharey=True)

# Plot Recency
sns.barplot(data=rfm_plot, x='Cluster', y='Recency', hue='Model', ax=axes[0])
```

```
axes[0].set_title('Average Recency by Cluster')
axes[0].set_ylabel('Recency (days)')
axes[0].legend(title='Model')

# Plot Frequency
sns.barplot(data=rfm_plot, x='Cluster', y='Frequency', hue='Model', ax=axes[1])
axes[1].set_title('Average Frequency by Cluster')
axes[1].set_ylabel('Frequency (orders)')
axes[1].legend().remove()

# Plot Monetary
sns.barplot(data=rfm_plot, x='Cluster', y='Monetary', hue='Model', ax=axes[2])
axes[2].set_title('Average Monetary by Cluster')
axes[2].set_ylabel('Monetary Value (£)')
axes[2].legend().remove()

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



```
In [ ]:
```