Dissertation

1. Data Preparation

In [18]:

```
import pandas as pd
import os
import glob

import warnings
warnings.filterwarnings('ignore')

path = r'C:\Users\Umar Fadhil Ramadhan\OneDrive - University of Leeds\99. Dissertation\99. Final Dissertation\0
3. Disseration\04. Code\dataset'
csv_files = glob.glob(os.path.join(path, "*.csv"))

df = []

for f in csv_files:
    data = pd.read_csv(f, index_col = None, header = 0)
    df.append(data)

df = pd.concat(df, axis = 0, ignore index = True)
```

In [19]:

```
pd.set_option('display.max_columns', None)
df.sample(5)
```

Out[19]:

	ID	Inquiry ID	Title	Code	Hash	Token	Type	Amount	Meter Account ID	Paymer Gatewa
311802	1325191	5059272	Pembelian Token 500.000	NaN	3A678943940A27F3492F4B135B8EE631D411E336-TRANSACTION	02698607903556738372	prepaid	500000.0	35228.0	bc
792366	3268315	8383097	Pembayaran Tagihan DES21	NaN	47CC94F30D9228FB84E3E8C25B24A16332481615-TRANSACTION	NaN	postpaid	2988244.0	10827637.0	b
81798	515422	2242974	Pembelian Token 100.000	NaN	70FFA979EAAE4C7CF7315FF45C34562A9013BAB1-TRANSACTION	56386465315399031705	prepaid	100000.0	NaN	mandi
2972813	2345014	6300884	Pembelian Token 50.000	NaN	8282E854CAA651CDE983CA9A575BB9762834041B-TRANSACTION	NaN	prepaid	50000.0	5479950.0	mandi
874429	3352098	8498696	Pembayaran Tagihan DES21	NaN	80D7D39BF6C0848EC4BEA868CC2900D0BE86046D-TRANSACTION	NaN	postpaid	3930.0	NaN	mandi

In [20]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3394781 entries, 0 to 3394780
Data columns (total 27 columns):
 #   Column              Dtype
---  ------              -----
 0   ID                  int64
 1   Inquiry ID          int64
 2   Title               object
 3   Code                object
 4   Hash                object
 5   Token               object
 6   Type                object
 7   Amount              float64
 8   Meter Account ID    float64
 9   Payment Gateway     object
 10  Status ID           float64
 11  Description         float64
 12  Data                object
 13  Deleted At          object
```

```
 14  Created At           object
 15  Updated At           object
 16  Linkaja Trx ID       float64
 17  Linkaja Is Reversal  bool
 18  Virtual Account      object
 19  User ID              float64
 20  Meter Number         float64
 21  Status Code          object
 22  Tax                  float64
 23  Total                float64
 24  Trx ID               object
 25  Response Code        object
 26  Message              object
dtypes: bool(1), float64(9), int64(2), object(15)
memory usage: 676.6+ MB
```

1.1 Data Cleaning

In [21]:

```
#Column Data extraction
df_transform = df[['ID', 'Data']]
df_transform['Data'] = df_transform[['Data']].replace('[][]', '', regex = True)
df_transform['Data'] = df_transform['Data'].replace('},{', ',', regex = True)
df_transform['Data'].replace('', np.nan, inplace = True)

df_transform = df_transform.dropna()


null = None

df_json = df_transform['Data'].reset_index().drop(columns='index')
df_json = df_json['Data']
df_eval = df_json.apply(lambda x: eval(x))
df_normalize = pd.json_normalize(df_eval)

#Concating with original dataset
df_data = pd.concat([df_transform['ID'].reset_index().drop(columns='index'), df_normalize], axis = 1)
df_data = df_data[['ID', 'data.tarif', 'data.idpel', 'data.daya']]
df_data.rename(columns = {'data.tarif':'Tariff', 'data.idpel':'Meter No', 'data.daya':'Power'}, inplace = True)
```

In [23]:

```
from datetime import datetime
import datetime as dt

datecorrected = []
for i in df['Updated At']:
    datecorrected.append(i[0:10])

df['Updated At'] = datecorrected
df['Updated At'] = pd.to_datetime(df['Updated At'])
```

1.2 Expolatory Data Analysis

In [24]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import sys

pd.set_option('display.max_colwidth', None)
pd.set_option('display.max_columns', None)

#All transactions
df_eda = df[['ID', 'User ID', 'Type', 'Payment Gateway', 'Amount', 'Tax', 'Total', 'Status Code', 'Updated At']
]
df_eda.rename(columns={'Updated At' : 'Payment Date'}, inplace = True)
print('All transactions:', len(df_eda.index))

#Paid & transactions of 2021
df_eda = df_eda[(df_eda['Status Code'] == 'paid') & (df_eda['Payment Date'] < '2022-01-01')].reset_index().drop
(columns = 'index')

#Merge with Data table
df_eda = pd.merge(df_eda, df_data, on = 'ID', how = 'left')
print('Paid transactions:', len(df_eda.index))
```

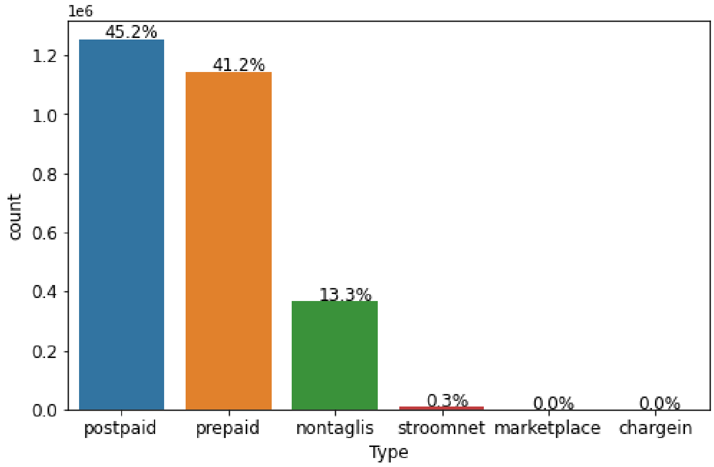
```
All transactions: 3394781
Paid transactions: 2771848
```

- **Type of Transactions**

In [26]:

```
def plot_count(ax, feature):
    total = len(feature)
    for i in ax.patches:
```

```
        percentage = '{:.1f}%'.format(100 * i.get_height()/total)
        x = i.get_x() + i.get_width() / 3.3
        y = i.get_y() + i.get_height() * 1.005
        ax.annotate(percentage, (x, y), size = 12)

plt.figure(figsize = (8, 5))
ax = sns.countplot(x = 'Type', data = df_eda)

plt.xticks(size = 12)
plt.xlabel('Type', size = 12)
plt.yticks(size = 12)
plt.ylabel('count', size = 12)

plot_count(ax, df_eda.Type)
```



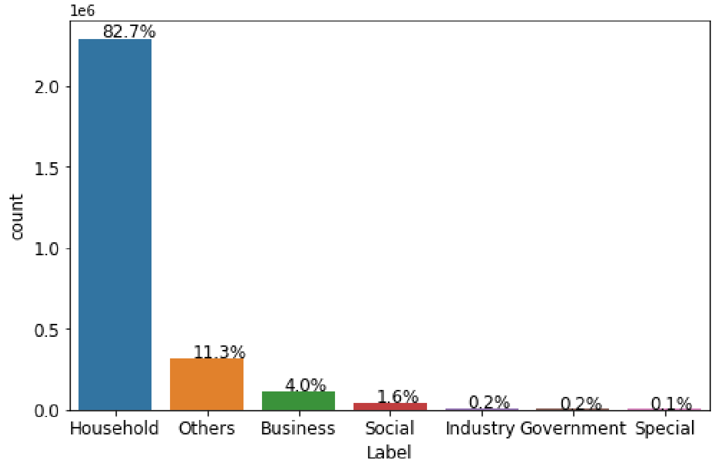
- **Type of Customers**

In [27]:

```
import re

df_toc = df_eda[['Tariff']]
cust_label = []

def cust_transform(data_frame):
    for i in data_frame:
        if i == '':
            lab = 'Others'
            cust_label.append(lab)
        elif pd.isna(i) == True :
            lab = 'Others'
            cust_label.append(lab)
        elif i[0] == 'R':
            lab = 'Household'
            cust_label.append(lab)
        elif i[0] == 'B':
            lab = 'Business'
            cust_label.append(lab)
        elif i[0] == 'P':
            lab = 'Government'
            cust_label.append(lab)
        elif i[0] == 'I':
            lab = 'Industry'
            cust_label.append(lab)
        elif i[0] == 'S':
            lab = 'Social'
            cust_label.append(lab)
        else:
            lab = 'Special'
            cust_label.append(lab)

cust_transform(df_toc['Tariff'])

df_toc = pd.concat([df_toc, pd.DataFrame(cust_label, columns=['Label'])], axis = 1)
```

In [28]:

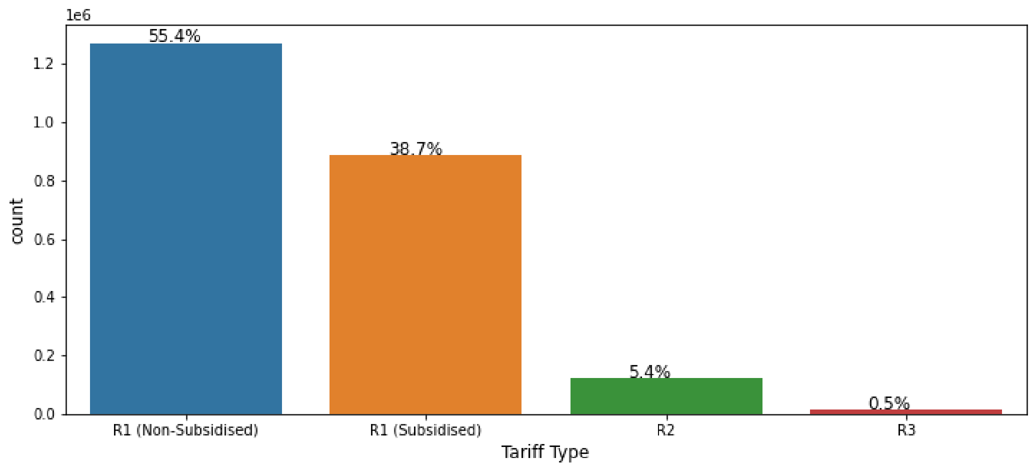
```
plt.figure(figsize = (8,5))
ax = sns.countplot(x = 'Label', data = df_toc, order = df_toc['Label'].value_counts().index)

plt.xticks(size = 12)
plt.xlabel('Label', size = 12)
plt.yticks(size = 12)
plt.ylabel('count', size = 12)

plot_count(ax, df_toc.Label)
```



- **Household Customers**

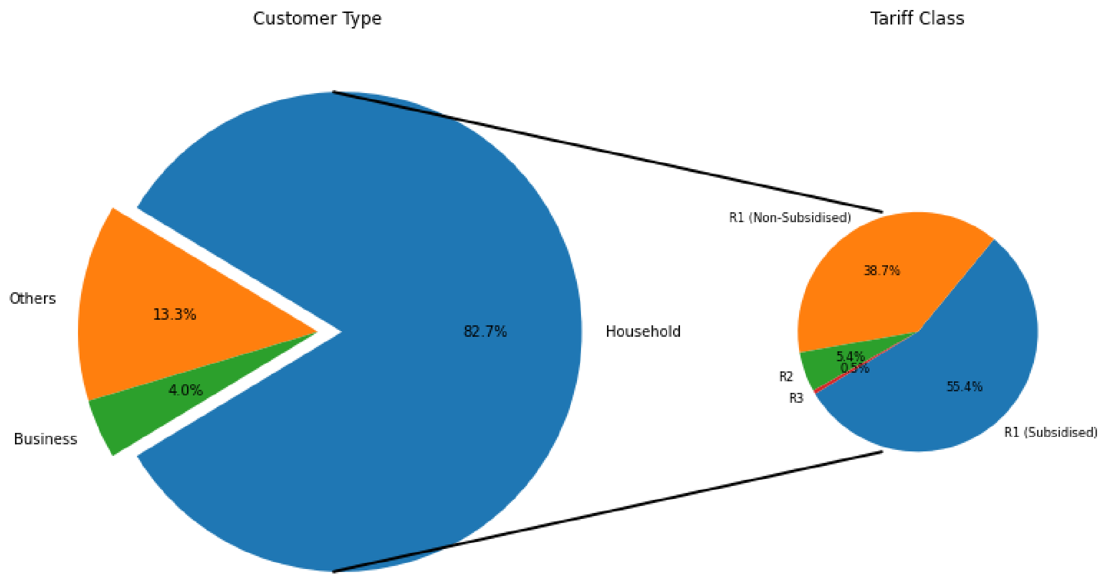
In [31]:

```
df_household = pd.concat([df_toc[['Tariff', 'Label']], df_eda[['Power']]], axis = 1)
df_household = df_household[df_household['Label'] == 'Household']
df_household['Power'] = df_household[['Power']].astype('int64')

tariff = []

for i, j in zip(df_household['Tariff'], df_household['Power']):
    if (i == 'R1') & (j <= 900):
        tariff.append('R1 (Subsidised)')
    elif (i == 'R1T') & (j <= 900):
        tariff.append('R1 (Subsidised)')
    elif (i == 'R1M') & (j <= 900):
        tariff.append('R1 (Non-Subsidised)')
    elif (i == 'R1MT') & (j <= 900):
        tariff.append('R1 (Non-Subsidised)')
    elif (i == 'R1') & (j <= 2200):
        tariff.append('R1 (Non-Subsidised)')
    elif (i == 'R1T') & (j <= 2200):
        tariff.append('R1 (Non-Subsidised)')
    elif (i == 'R2') & (j <= 5500):
        tariff.append('R2')
    elif (i == 'R2T') & (j <= 5500):
        tariff.append('R2')
    elif (i == 'R3') & (j >= 6600):
        tariff.append('R3')
    elif (i == 'R3T') & (j >= 6600):
        tariff.append('R3')
    else:
        tariff.append('Not Classified')

df_household['Tariff Type'] = tariff

plt.figure(figsize = (12,5))
ax = sns.countplot(x = 'Tariff Type', data = df_household, order = df_household['Tariff Type'].value_counts().i
ndex)

plt.xticks(size = 10)
plt.xlabel('Tariff Type', size = 12)
plt.yticks(size = 10)
plt.ylabel('count', size = 12)

plot_count(ax, df_household.Label)
```



In [32]:

```
import matplotlib.pyplot as plt
from matplotlib.patches import ConnectionPatch
import numpy as np

# make figure and assign axis objects
fig = plt.figure(figsize=(15, 10))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
fig.subplots_adjust(wspace = 0.0)

# large pie chart parameters
ratios = [.827, .133, .040]
labels = ['Household', 'Others', 'Business']
explode = [0.1, 0, 0]

# rotate so that first wedge is split by the x-axis
angle = -180 * ratios[0]
ax1.pie(ratios, autopct = '%1.1f%%', startangle = angle,
        labels = labels, explode = explode)

# small pie chart parameters
ratios = [.554, .387, .054, .005]
labels = ['R1 (Subsidised)', 'R1 (Non-Subsidised)', 'R2', 'R3']
width = .3

ax2.pie(ratios, autopct = '%1.1f%%', startangle = angle,
        labels = labels, radius = 0.5, textprops = {'size': 'smaller'})

ax1.set_title('Customer Type')
ax2.set_title('Tariff Class')

# use ConnectionPatch to draw lines between the two plots
theta1, theta2 = ax1.patches[0].theta1, ax1.patches[0].theta2
center, r = ax1.patches[0].center, ax1.patches[0].r

# draw top connecting line
x = r * np.cos(np.pi / 100 * theta2) + center[0]
y = np.sin(np.pi / -100 * theta2) + center[1]
con = ConnectionPatch(xyA = (- width / 2, .5), xyB = (x, y),
                      coordsA = "data", coordsB = "data", axesA = ax2, axesB = ax1)
con.set_color([0, 0, 0])
con.set_linewidth(2)
ax2.add_artist(con)

# draw bottom connecting line
x = r * np.cos(np.pi / 100 * theta1) + center[0]
y = np.sin(np.pi / -100 * theta1) + center[1]
con = ConnectionPatch(xyA = (- width / 2, -.5), xyB = (x, y), coordsA="data",
                      coordsB = "data", axesA = ax2, axesB = ax1)
con.set_color([0, 0, 0])
ax2.add_artist(con)
con.set_linewidth(2)

plt.show()
```



1.3 Handling Missing Values

In [33]:

```
#replace blank with null value
df_eda = df_eda.replace('', np.nan, regex=True)

#count missing values
df_eda.isna().sum()
```

Out[33]:

```
ID                      0
User ID                 3
Type                    0
Payment Gateway         0
Amount                  0
Tax                     0
Total                   0
Status Code             0
Payment Date            0
Tariff             313760
Meter No            85977
Power              313760
dtype: int64
```

- **Missing Data in User ID**

In [34]:

```
df_eda[df_eda['User ID'].isnull()]
```

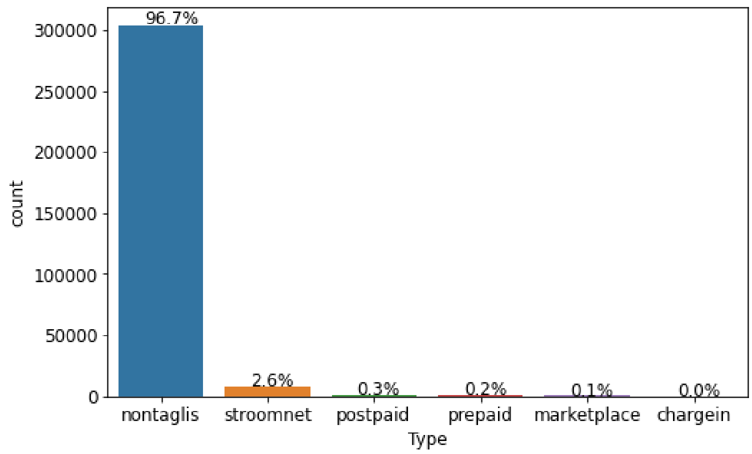
Out[34]:

	ID	User ID	Type	Payment Gateway	Amount	Tax	Total	Status Code	Payment Date	Tariff	Meter No	Power
999821	146341	NaN	prepaid	mandiri	50000.0	0.0	50000.0	paid	2021-01-05	R1M	535212059977	900
1145687	1034158	NaN	prepaid	mandiri	200000.0	0.0	200000.0	paid	2021-07-20	R1	437000344090	1300
1530421	614660	NaN	postpaid	mandiri	266736.0	0.0	266736.0	paid	2021-05-14	R1M	524030748083	900

- **Missing Data in Tariff, Payment Date, Power**

In [35]:

```
misdat = df_eda[df_eda['Tariff'].isnull()]

plt.figure(figsize = (8,5))
ax = sns.countplot(x = 'Type', data = misdat, order = misdat['Type'].value_counts().index)

plt.xticks(size = 12)
plt.xlabel('Type', size = 12)
plt.yticks(size = 12)
plt.ylabel('count', size = 12)

plot_count(ax, misdat.Type)
```



In [36]:

```
misdat['Type'].value_counts()
```

Out[36]:

```
nontaglis      303454
stroomnet        8252
postpaid          976
prepaid           629
marketplace       408
chargein           41
```

```
Name: Type, dtype: int64
```

- **Drop NaN Rows**

In [37]:

```
df_eda = df_eda.dropna(how='any', axis=0)
print('Number of rows:', len(df_eda.index))
```

```
Number of rows: 2394126
```

1.4 Remove Duplicates

In [38]:

```
df_eda = df_eda.drop_duplicates()
print('Number of rows:', len(df_eda.index))
```

```
Number of rows: 2394126
```

1.5 Feature Selection

In [63]:

```
df_fs = df_eda[['Meter No', 'Type', 'Tariff', 'Power',  'Total', 'Payment Date']]
df_fs = df_fs[(df_fs['Type'] == 'prepaid') | (df_fs['Type'] == 'postpaid')]
df_fs = df_fs[['Meter No', 'Type', 'Tariff', 'Power',  'Total', 'Payment Date']]
```

In [64]:

```
df_fs
```

Out[64]:

	Meter No	Type	Tariff	Power	Total	Payment Date
0	521511582351	postpaid	R1M	900	90982.0	2021-04-12
1	520561242795	prepaid	S2	900	21750.0	2021-04-12
2	538310351546	prepaid	R1M	900	201750.0	2021-04-12
3	546201277487	prepaid	R2	3500	501750.0	2021-04-12
4	513080145836	prepaid	S2	900	20000.0	2021-04-12
...	...	...	...	...	...	...
2771843	231571176952	postpaid	R1	450	51338.0	2021-09-01
2771844	120030888140	prepaid	R1	1300	100000.0	2021-09-01
2771845	515050048052	postpaid	R1	450	27786.0	2021-09-01
2771846	514032032886	prepaid	R1M	900	50000.0	2021-09-01
2771847	532411813437	postpaid	R1M	900	231066.0	2021-09-01

2394104 rows × 6 columns

1.6 Data Transformation

- **Tariff Type**

In [65]:

```
df_dt = df_fs
cust_label = []

cust_transform(df_dt['Tariff'])
df_dt['Customer Type'] = cust_label

#Only select household customers
df_dt = df_dt[df_dt['Customer Type'] == 'Household']
df_dt.drop(columns=['Customer Type'], inplace = True)

#Convert from object to int format
df_dt['Power'] = df_dt['Power'].astype(int)
```

In [66]:

```
df_dt.sample(5)
```

Out[66]:

	Meter No	Type	Tariff	Power	Total	Payment Date
195696	221310316648	postpaid	R1	450	23436.0	2021-08-10
1426643	513530013104	postpaid	R1	1300	83635.0	2021-03-09
2654688	516720739819	prepaid	R1M	900	21750.0	2021-09-21
2100451	537316113704	postpaid	R1	1300	504145.0	2021-10-06
1106229	547201150490	prepaid	R1	1300	101750.0	2021-07-15

1.7 RFM Model

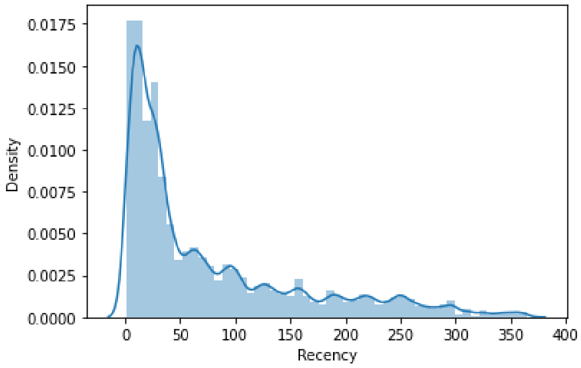
- **Recency**

In [67]:

```
from datetime import datetime
import datetime as dt

df_r = df_dt[['Meter No', 'Payment Date']].sort_values(by = ['Meter No', 'Payment Date'], ascending = True)
df_r = df_r.drop_duplicates(subset=['Meter No'], keep='last')

df_r['Recency'] = datetime.strptime("2022-01-01", "%Y-%m-%d") - df_r['Payment Date']

df_r['Recency'] = (df_r['Recency']).dt.days.astype('int16')


#Plotting recency
sns.distplot(df_r['Recency'])
```

Out[67]:

```
<AxesSubplot:xlabel='Recency', ylabel='Density'>
```



In [68]:

```
df_r.describe(percentiles=[0.01, 0.02, 0.05, 0.10, 0.25, 0.50, 0.75, 0.90, 0.95, 0.98, 0.99])
```

Out[68]:

	Recency
count	812923.000000
mean	75.047827
std	82.511256
min	1.000000
1%	1.000000
2%	2.000000
5%	4.000000
10%	7.000000
25%	15.000000
50%	37.000000
75%	107.000000
90%	214.000000
95%	257.000000
98%	298.000000
99%	328.000000
max	365.000000

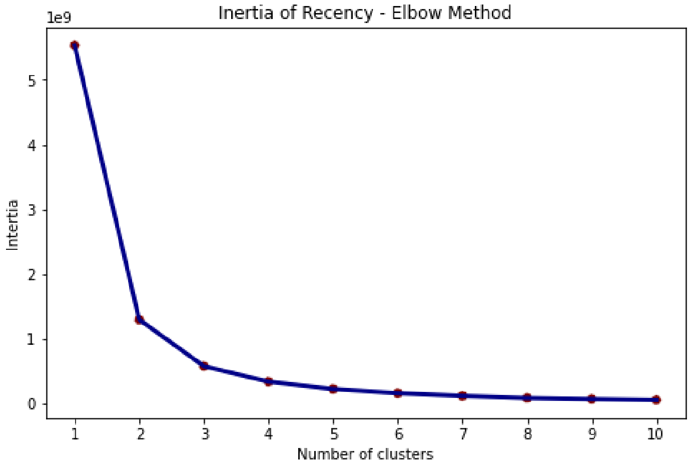
In [69]:

```
from sklearn import cluster
from sklearn.cluster import KMeans
```

```python
from itertools import count

#Define number of R cluster
sse = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 100, random_state = 201578925)
    kmeans.fit(df_r[['Recency']].values)
    sse.append(kmeans.inertia_)



plt.figure(figsize=(8, 5))

sns.lineplot(x=range(1, 11), y = sse, color = '#000087', linewidth = 3)
sns.scatterplot(x=range(1, 11), y = sse, s = 50, color = '#800000',  linestyle = '--')
plt.xticks(range(1, 11), range(1, 11))
plt.xlabel('Number of clusters')
plt.ylabel('Intertia')
plt.title('Inertia of Recency - Elbow Method')
```

Out[69]:

```
Text(0.5, 1.0, 'Inertia of Recency - Elbow Method')
```



In [70]:

```python
#Scoring with k-means clustering
kmeans = KMeans(n_clusters = 4, max_iter = 300, random_state = 201578925)
kmeans.fit(df_r[['Recency']])

df_r = df_r.assign(cluster = kmeans.labels_)

#Sorting cluster
df_r.replace({
    'cluster' : {
        0 : 4,
        1 : 1,
        2 : 3,
        3 : 2,
    }
}, inplace = True)

#Rename column
df_r = df_r[['Meter No', 'Recency', 'cluster']]
df_r.rename(columns = {'cluster' : 'R'}, inplace = True)

#Descriptive stats of clusters
df_r.groupby(['R']).agg(
    count = ('Meter No', 'count'),
    min_recency = ('Recency', min),
    max_recency = ('Recency', max),
    std_recency = ('Recency', 'std'),
    avg_recency = ('Recency', 'mean')
).sort_values(by = 'avg_recency')
```

Out[70]:

R	count	min_recency	max_recency	std_recency	avg_recency
4	467210	1	52	13.008064	20.018322
3	171532	53	127	21.127403	84.556718
2	101872	128	221	27.810120	171.000049
1	72309	222	365	37.567359	272.871219

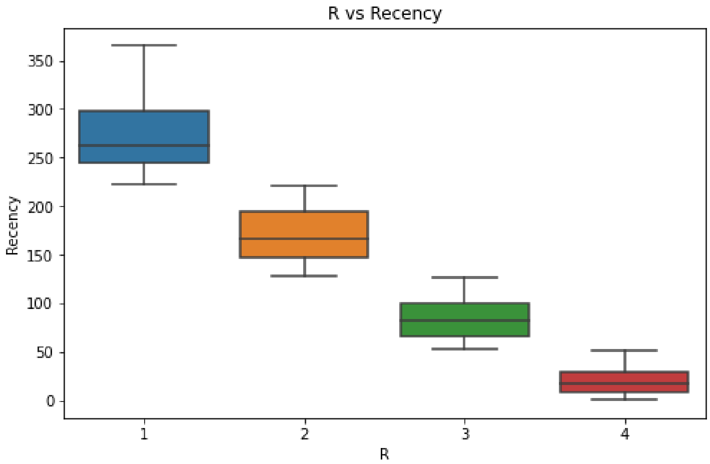
In [71]:

```python
plt.figure(figsize = (8, 5))
plt.title('R vs Recency')
```

```
sns.boxplot(data = df_r, x = 'R', y = 'Recency')
```

Out[71]:

```
<AxesSubplot:title={'center':'R vs Recency'}, xlabel='R', ylabel='Recency'>
```



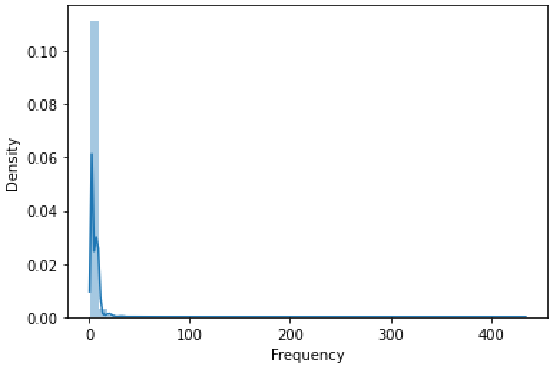
- **Frequency**

In [72]:

```
df_f = pd.DataFrame(df_dt['Meter No'].value_counts()).reset_index()
df_f = df_f.rename(columns={'Meter No':'Frequency', 'index':'Meter No'})

#Plotting recency
sns.distplot(df_f['Frequency'])
```

Out[72]:

```
<AxesSubplot:xlabel='Frequency', ylabel='Density'>
```



In [73]:

```
df_f.describe(percentiles=[0.01, 0.02, 0.05, 0.10, 0.25, 0.50, 0.75, 0.90, 0.95, 0.98, 0.99])
```

Out[73]:

	Frequency
count	812923.000000
mean	2.751726
std	3.724116
min	1.000000
1%	1.000000
2%	1.000000
5%	1.000000
10%	1.000000
25%	1.000000
50%	1.000000
75%	3.000000
90%	6.000000
95%	8.000000
98%	12.000000
99%	16.000000
max	434.000000

In [74]:

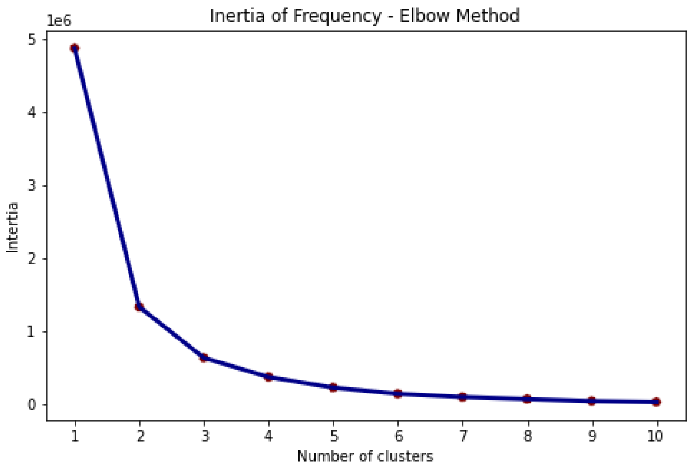
```
#Removing outliers
df_f = df_f[df_f['Frequency'] <= 16]
```

In [75]:

```
#Define number of F cluster
sse = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 100, random_state = 201578925)
    kmeans.fit(df_f[['Frequency']].values)
    sse.append(kmeans.inertia_)



plt.figure(figsize=(8, 5))

sns.lineplot(x=range(1, 11), y = sse, color = '#000087', linewidth = 3)
sns.scatterplot(x=range(1, 11), y = sse, s = 50, color = '#800000',  linestyle = '--')
plt.xticks(range(1, 11), range(1, 11))
plt.xlabel('Number of clusters')
plt.ylabel('Intertia')
plt.title('Inertia of Frequency - Elbow Method')
```

Out[75]:

```
Text(0.5, 1.0, 'Inertia of Frequency - Elbow Method')
```



In [76]:

```
#Scoring with k-means clustering
kmeans = KMeans(n_clusters = 4, max_iter = 300, random_state = 201578925)
kmeans.fit(df_f[['Frequency']])

df_f = df_f.assign(cluster = kmeans.labels_)

#Sorting cluster
df_f.replace({
    'cluster' : {
        0 : 3,
        1 : 1,
        2 : 2,
        3 : 4
    }
}, inplace = True)

#Rename column
df_f = df_f[['Meter No', 'Frequency', 'cluster']]
df_f.rename(columns = {'cluster' : 'F'}, inplace = True)

#Descriptive stats of clusters
df_f.groupby(['F']).agg(
    count = ('Meter No', 'count'),
    min_frequency = ('Frequency', min),
    max_frequency = ('Frequency', max),
    std_frequency = ('Frequency', 'std'),
    avg_frequency = ('Frequency', 'mean')
).sort_values(by = 'avg_frequency')
```

Out[76]:

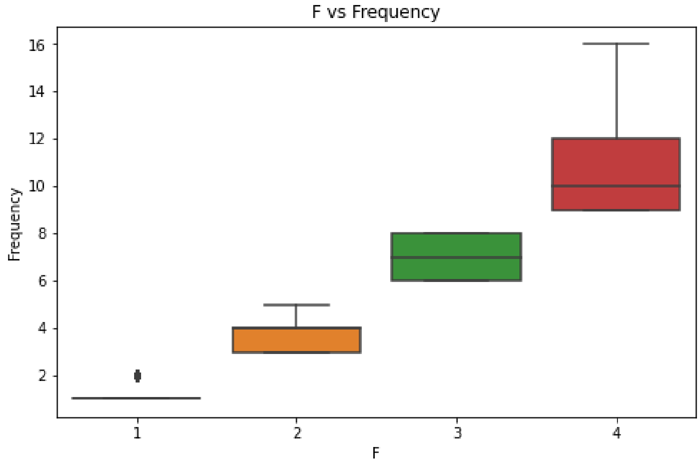
F	count	min_frequency	max_frequency	std_frequency	avg_frequency
1	560207	1	2	0.425555	1.237505
2	151200	3	5	0.791644	3.745456
3	60873	6	8	0.808992	6.859232
4	32861	9	16	1.980709	10.736800

In [78]:

```
plt.figure(figsize = (8, 5))
sns.boxplot(data = df_f, x = 'F', y = 'Frequency')
plt.title('F vs Frequency')
```

Out[78]:

```
Text(0.5, 1.0, 'F vs Frequency')
```



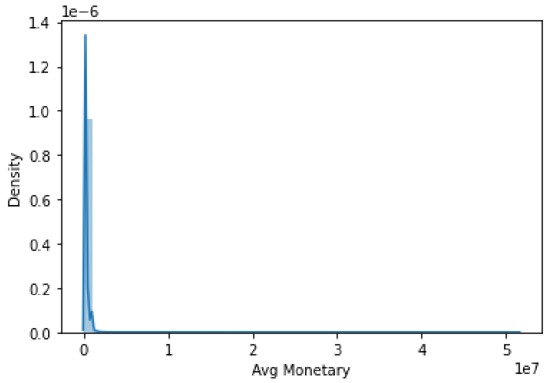
- **Monetary**

In [79]:

```
df_m = pd.pivot_table(df_dt, values = 'Total', index = ['Meter No'],
                aggfunc = np.sum).reset_index()
df_m = df_m.rename(columns = {'Total' : 'Monetary'})

#Merging with frequency to find average transaction amount
df_m = df_m.merge(df_f[['Meter No', 'Frequency']], how = 'left', on = 'Meter No')
df_m['Avg Monetary'] = df_m['Monetary']/df_m['Frequency']

#Plotting recency
sns.distplot(df_m['Avg Monetary'])
```

Out[79]:

```
<AxesSubplot:xlabel='Avg Monetary', ylabel='Density'>
```



In [80]:

```
df_m.describe(percentiles=[0.01, 0.02, 0.05, 0.10, 0.25, 0.50, 0.75, 0.90, 0.95, 0.98, 0.99])
```

Out[80]:

	Monetary	Frequency	Avg Monetary
count	8.129230e+05	805141.000000	8.051410e+05
mean	3.625992e+05	2.521218	1.252398e+05
std	1.072771e+06	2.461483	2.595797e+05
min	1.000000e+00	1.000000	1.000000e+00
1%	5.000000e+03	1.000000	4.306283e+03
2%	5.550000e+03	1.000000	5.000000e+03
5%	9.834000e+03	1.000000	6.750000e+03
10%	1.636900e+04	1.000000	1.138200e+04
25%	3.435800e+04	1.000000	2.150000e+04
50%	1.000000e+05	1.000000	5.055000e+04
75%	2.943320e+05	3.000000	1.297033e+05

	Monetary	Frequency	Avg Monetary
90%	[illegible]	[illegible]	[illegible]
95%	1.526050e+06	8.000000	5.000000e+05
98%	2.910500e+06	10.000000	7.641638e+05
99%	4.226987e+06	12.000000	1.001750e+06
max	2.654823e+08	16.000000	5.156278e+07

In [82]:

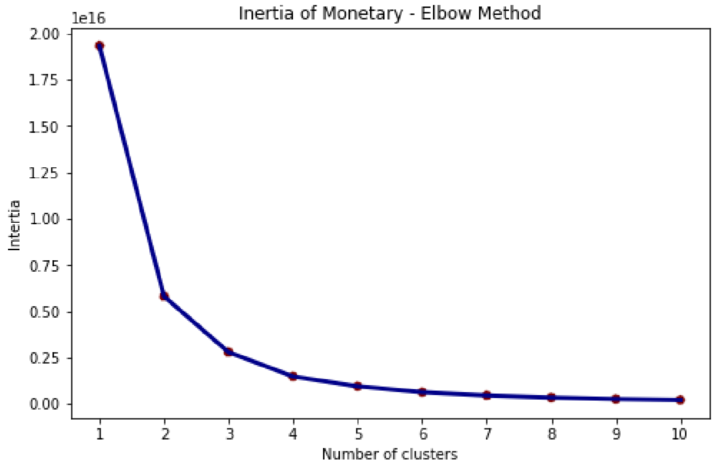
```
df_m = df_m[(df_m['Avg Monetary'] >= 4306.283) & (df_m['Avg Monetary'] <= 1001750)]
```

In [83]:

```
#Define number of M cluster
sse = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 100, random_state = 201578925)
    kmeans.fit(df_m[['Avg Monetary']].values)
    sse.append(kmeans.inertia_)



plt.figure(figsize=(8, 5))

sns.lineplot(x=range(1, 11), y = sse, color = '#000087', linewidth = 3)
sns.scatterplot(x=range(1, 11), y = sse, s = 50, color = '#800000',  linestyle = '--')
plt.xticks(range(1, 11), range(1, 11))
plt.xlabel('Number of clusters')
plt.ylabel('Intertia')
plt.title('Inertia of Monetary - Elbow Method')
```

Out[83]:

```
Text(0.5, 1.0, 'Inertia of Monetary - Elbow Method')
```



In [84]:

```
#Scoring with k-means clustering
kmeans = KMeans(n_clusters = 4, max_iter = 300, random_state = 201578925)
kmeans.fit(df_m[['Avg Monetary']])

df_m = df_m.assign(cluster = kmeans.labels_)

#Sorting cluster
df_m.replace({
    'cluster' : {
        0 : 2,
        1 : 1,
        2 : 4,
        3 : 3
    }
}, inplace = True)

#Rename column
df_m = df_m[['Meter No', 'Avg Monetary', 'cluster']]
df_m.rename(columns = {'Avg Monetary' : 'Monetary', 'cluster' : 'M'}, inplace = True)

#Descriptive stats of clusters
df_m.groupby(['M']).agg(
    count = ('Meter No', 'count'),
    min_monetary = ('Monetary', min),
    max_monetary = ('Monetary', max),
    std_monetary = ('Monetary', 'std'),
    avg_monetary = ('Monetary', 'mean')
).sort_values(by = 'avg_monetary')
```

Out[84]:

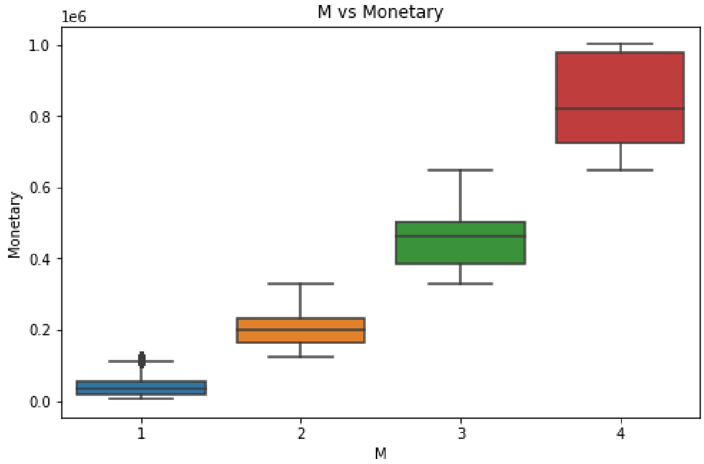
M	count	min_monetary	max_monetary	std_monetary	avg_monetary
1	590447	4306.333333	123375.0	30925.232292	43114.494018
2	132945	123378.000000	330037.0	51653.731813	203715.162777
3	52040	330047.000000	646186.0	80173.693633	456711.311021
4	14792	646316.333333	1001750.0	123562.261958	836393.539848

In [85]:

```
plt.figure(figsize = (8, 5))
sns.boxplot(data = df_m, x = 'M', y = 'Monetary')
plt.title('M vs Monetary')
```

Out[85]:

```
Text(0.5, 1.0, 'M vs Monetary')
```



- **Financial Ability**

In [86]:

```
#Tariff classification
df_fa = df_dt[['Meter No', 'Tariff', 'Power']]
tariff = []

for i, j in zip(df_fa['Tariff'], df_fa['Power']):
    if (i == 'R1') & (j <= 900):
        tariff.append(1)
    elif (i == 'R1M') & (j <= 900):
        tariff.append(2)
    elif (i == 'R1') & (j <= 2200):
        tariff.append(2)
    elif (i == 'R2') & (j <= 5500):
        tariff.append(3)
    elif (i == 'R3') & (j >= 6600):
        tariff.append(4)
    else:
        tariff.append('Not Classified')

df_fa['Tariff Type'] = tariff
```

In [87]:

```
df_fa = pd.pivot_table(df_fa, values = 'Tariff', index = ['Meter No', 'Power', 'Tariff Type'],
                aggfunc = 'count').sort_values(by = ['Meter No', 'Tariff'], ascending = False).reset_index()
df_fa = df_fa[['Meter No', 'Power', 'Tariff Type']].drop_duplicates(subset=['Meter No'], keep='first')
df_fa.rename(columns = {'Tariff Type':'FA'}, inplace = True)

#Descriptive stats of clusters
df_fa.groupby(['FA']).agg(
    count = ('Meter No', 'count'),
    min_power = ('Power', min),
    max_power = ('Power', max),
    std_power = ('Power', 'std'),
    avg_power = ('Power', 'mean')
).sort_values(by = 'avg_power')
```

Out[87]:

FA	count	min_power	max_power	std_power	avg_power
1	369884	450	900	176.964058	536.043868
2	407829	900	2200	442.590376	1227.366617
3	31833	3500	5500	855.346169	4275.182986
4	3377	6600	131000	6825.963616	10833.283980

In [88]:

```
df_fa[['Power']].describe(percentiles=[0.01, 0.02, 0.05, 0.10, 0.25, 0.50, 0.75, 0.90, 0.95, 0.98, 0.99, 0.999]
)
```

Out[88]:

	Power
count	812923.000000
mean	1072.064205
std	1129.195314
min	450.000000
1%	450.000000
2%	450.000000
5%	450.000000
10%	450.000000
25%	450.000000
50%	900.000000
75%	1300.000000
90%	2200.000000
95%	2200.000000
98%	4400.000000
99%	5500.000000
99.9%	11000.000000
max	131000.000000

In [89]:

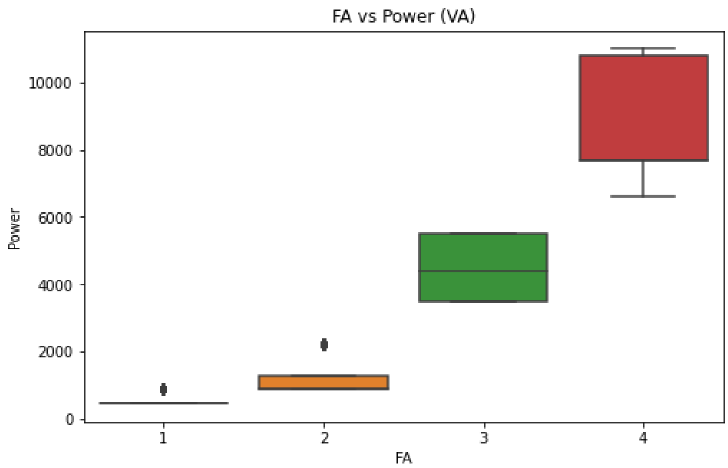
```
df_fa = df_fa[df_fa['Power'] <= 11000]
```

In [92]:

```
plt.figure(figsize = (8, 5))
sns.boxplot(data = df_fa, x = 'FA', y = 'Power')
plt.title('FA vs Power (VA)')
```

Out[92]:

Text(0.5, 1.0, 'FA vs Power (VA)')



- **RFM-FA Model**

In [93]:

```
from functools import reduce

df_rfmfa = [df_r[['Meter No', 'R', 'Recency']], df_f[['Meter No', 'F', 'Frequency']], df_m[['Meter No', 'M', 'M
onetary']], df_fa[['Meter No', 'FA', 'Power']]]
```

```
df_rfmfa = reduce(lambda left, right: pd.merge(left, right, on = 'Meter No', how = 'inner'), df_rfmfa)
```

In [94]:

```
df_rfmfa
```

Out[94]:

	Meter No	R	Recency	F	Frequency	M	Monetary	FA	Power
0	110000000010	4	23	1	1	1	50550.0	2	900
1	110000004308	2	152	1	2	2	152000.0	2	1300
2	110000004638	3	101	1	2	1	24529.5	1	450
3	110000005298	1	228	1	1	1	101750.0	2	1300
4	110000008591	3	61	1	2	2	199088.5	3	3500
...	...	...	...	...	...	...	...	...	...
790032	566602046701	4	8	1	1	1	101750.0	2	1300
790033	566602050451	4	28	1	1	1	51500.0	2	900
790034	566602055765	4	14	1	2	1	75000.0	2	900
790035	566602060157	4	11	1	1	1	11750.0	1	450
790036	566602065843	4	7	1	1	1	21750.0	2	900

790037 rows × 9 columns

2. Modelling

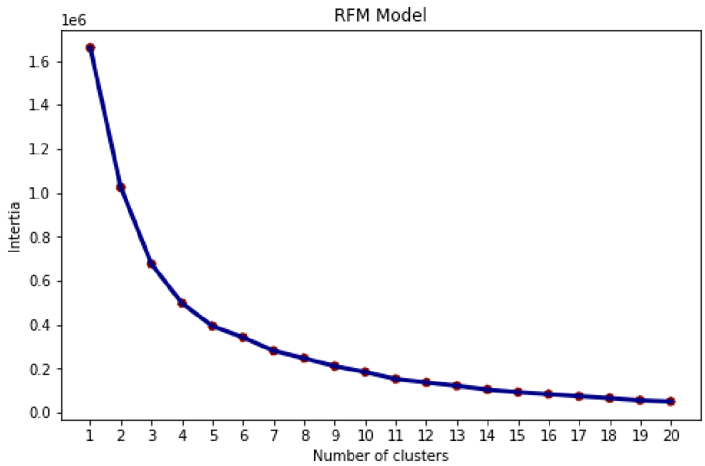
2.1 RFM

In [95]:

```
df_rfm = df_rfmfa[['Meter No', 'R', 'F', 'M']]

#Define number of cluster - RFM Model
sse = []

for i in range(1, 21):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, random_state = 201578925)
    kmeans.fit(df_rfm[['R', 'F', 'M']].values)
    sse.append(kmeans.inertia_)



plt.figure(figsize=(8, 5))

sns.lineplot(x=range(1, 21), y = sse, color = '#000087', linewidth = 3)
sns.scatterplot(x=range(1, 21), y = sse, s = 50, color = '#800000',  linestyle = '--')
plt.xticks(range(1, 21), range(1, 21))
plt.xlabel('Number of clusters')
plt.ylabel('Intertia')
plt.title('RFM Model')
```

Out[95]:

```
Text(0.5, 1.0, 'RFM Model')
```



In [96]:

```
sse
```

Out[96]:

```
[1659577.738690699,
 1024325.0315057974,
 676323.5346462115,
 497563.9326118915,
 393389.6488204917,
 342023.3212651442,
 280814.1588181788,
 247100.14874940182,
 211328.4742568324,
 184669.70118191215,
 152543.91916370078,
 136960.8823996782,
 121846.57499773597,
 104586.2898958969,
 92739.96236532267,
 84020.42416137207,
 74693.2099739924,
 65930.82972490226,
 55826.423126506386,
 50092.41510135109]
```

In [97]:

```
#Clustering
kmeans = KMeans(n_clusters = 6, init = 'k-means++', max_iter = 300, random_state = 201578925)
kmeans.fit(df_rfm[['R', 'F', 'M']])

label_rfm = kmeans.predict(df_rfm[['R', 'F', 'M']])

df_rfm['Cluster'] = label_rfm
df_rfm.sample(5)
```

Out[97]:

	Meter No	R	F	M	Cluster
545323	523080159519	4	3	1	0
786814	566201028006	4	1	2	3
376921	514041389786	2	1	2	1
132363	171510341959	4	1	1	2
406964	516020805340	4	3	3	4

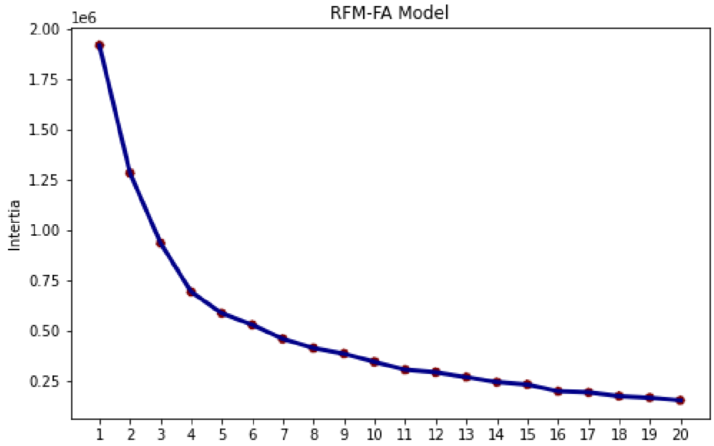
2.2 RFM-FA Model

In [98]:

```
#Define number of clusters
sse = []

for i in range(1, 21):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, random_state = 201578925)
    kmeans.fit(df_rfmfa[['R', 'F', 'M', 'FA']].values)
    sse.append(kmeans.inertia_)



plt.figure(figsize=(8, 5))

sns.lineplot(x=range(1, 21), y = sse, color = '#000087', linewidth = 3)
sns.scatterplot(x=range(1, 21), y = sse, s = 50, color = '#800000',  linestyle = '--')
plt.xticks(range(1, 21), range(1, 21))
plt.xlabel('Number of clusters')
plt.ylabel('Intertia')
plt.title('RFM-FA Model')
```

Out[98]:

```
Text(0.5, 1.0, 'RFM-FA Model')
```



Number of clusters

In [99]:

```
sse
```

Out[99]:

```
[1919781.0945424442,
 1284528.2208637516,
 936416.0698499976,
 694295.7608094394,
 587966.0478406792,
 530607.7086839657,
 460524.34125937306,
 414895.9382956324,
 386849.00082366954,
 346216.1854661005,
 307468.6283791872,
 294623.1478104383,
 270369.3662454835,
 245950.68167769111,
 233464.31577694137,
 200215.40628656116,
 194715.0705813596,
 176029.48629015786,
 168181.9839963822,
 155019.00120590354]
```

In [100]:

```
kmeans = KMeans(n_clusters = 10, init = 'k-means++', max_iter = 300, random_state = 201578925)
kmeans.fit(df_rfmfa[['R', 'F', 'M', 'FA']])

label_rfmfa = kmeans.predict(df_rfmfa[['R', 'F', 'M', 'FA']])

df_rfmfa['Cluster'] = label_rfmfa
df_rfmfa[['Meter No', 'R', 'F', 'M', 'FA', 'Cluster']].sample(5)
```

Out[100]:

	Meter No	R	F	M	FA	Cluster
712706	539611683159	2	2	1	1	0
108922	144000127192	4	2	2	2	1
714948	539910833178	4	1	1	1	2
206460	233501113583	4	1	1	2	8
253141	325511016653	4	2	4	2	3

3. Validation

3.1 RFM

In [101]:

```
from sklearn.metrics import silhouette_score, silhouette_samples

x = df_rfm[['R', 'F', 'M']]
y = label_rfm

ss = silhouette_score(x, y, metric = 'euclidean', sample_size = 100000, random_state = 201578925)

print(f'Silhouette score for k = 6: ' + str(np.round(ss, 3)))
```

```
Silhouette score for k = 6: 0.585
```

3.2 RFM-FA

In [102]:

```
x = df_rfmfa[['R', 'F', 'M', 'FA']]
y = label_rfmfa

print(f'Silhouette score for k = 10: ' + str(np.round(silhouette_score(x, y, metric = 'euclidean', sample_size
= 100000, random_state = 201578925), 3)))
```

```
Silhouette score for k = 10: 0.539
```

4. Interpretation

In [103]:

```
df_int = [df_r[['Meter No', 'Recency', 'R']], df_f[['Meter No', 'Frequency', 'F']], df_m[['Meter No', 'Monetary
', 'M']], df_fa[['Meter No', 'Power', 'FA']], df_rfm[['Meter No', 'Cluster']], df_rfmfa[['Meter No', 'Cluster']
]]
```

```
df_int = reduce(lambda left, right: pd.merge(left, right, on = 'Meter No', how = 'inner'), df_int)
df_int.rename(columns = {'Avg Monetary' : 'Monetary',
                         'Cluster_x' : 'Cluster RFM',
                         'Cluster_y' : 'Cluster RFM-FA'}, inplace = True)
df_int = df_int[['Meter No', 'Recency', 'Frequency', 'Monetary', 'Power', 'R', 'F', 'M', 'FA', 'Cluster RFM', '
Cluster RFM-FA']]
```

In [ ]:

```
# df int.to excel('result v4.xlsx')
```

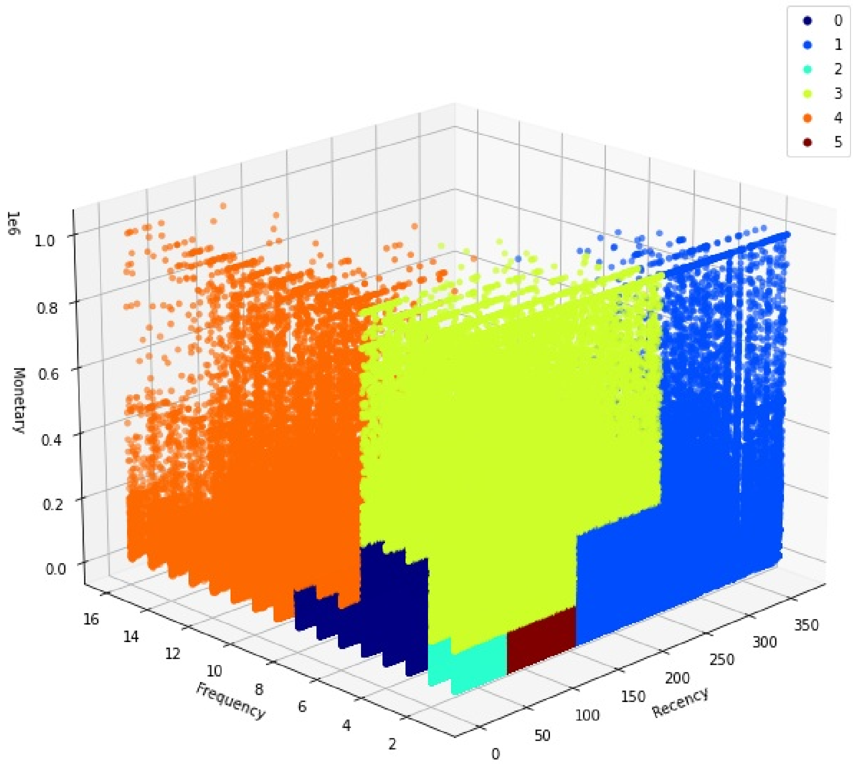
4.1 RFM Analysis

4.1 RFM Analysis

- **RFM Model**

In [104]:

```
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline

fig = plt.figure(figsize = (10, 8))
ax = Axes3D(fig)

data_np = df_int[['Recency', 'Frequency', 'Monetary', 'Cluster RFM']].to_numpy()

x = data_np[:,0]
y = data_np[:,1]
z = data_np[:,2]
c = data_np[:,3]

sc = ax.scatter3D(x, y, z, c = c, cmap = 'jet', linewidth=0.1)
plt.legend(*sc.legend_elements())


ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary')

ax.view init(20, -135)
```

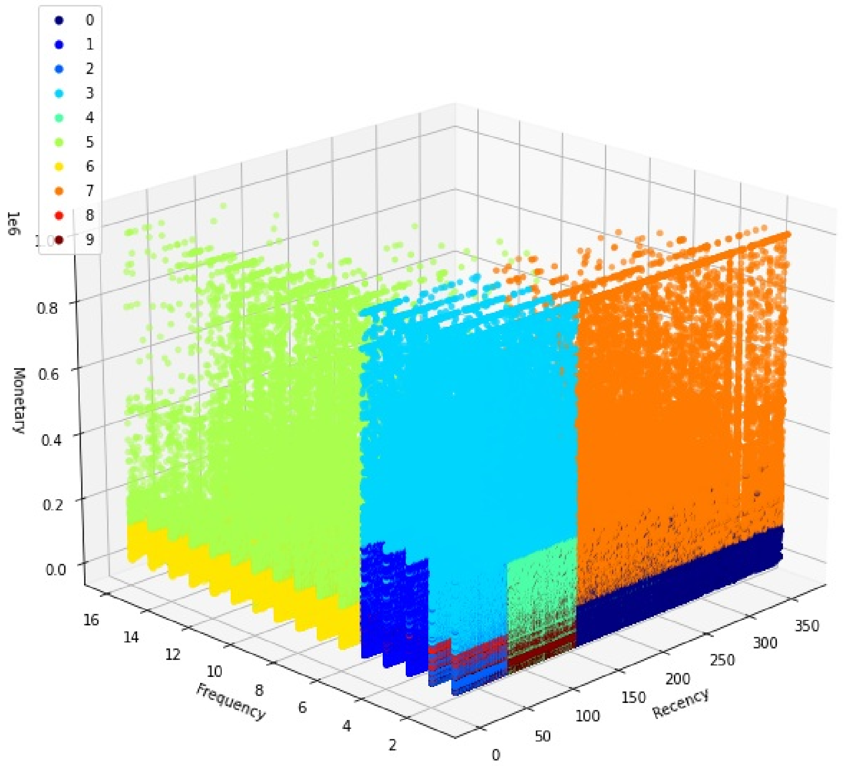


- **RFM-FA Model**

In [105]:

```
fig = plt.figure(figsize = (10, 8))
ax = Axes3D(fig)

data_np = df_int[['Recency', 'Frequency', 'Monetary', 'Cluster RFM-FA']].to_numpy()
```

```
x = data_np[:,0]
y = data_np[:,1]
z = data_np[:,2]
c = data_np[:,3]

sc = ax.scatter3D(x, y, z, c = c, cmap = 'jet', linewidth=0.1)
plt.legend(*sc.legend_elements())


ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary')

ax.view_init(20, -135)
```



4.2 Cluster Characteristics

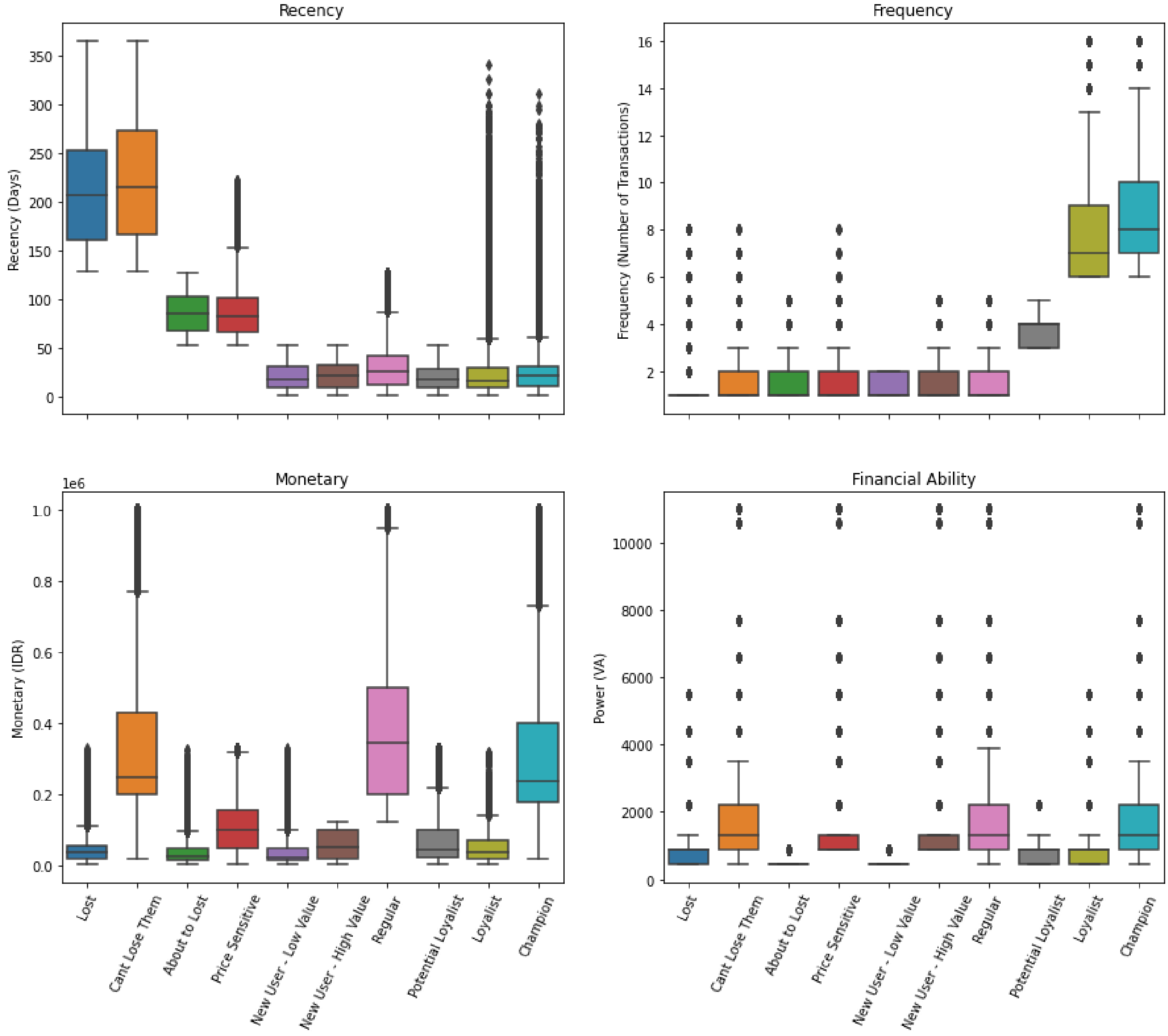
In [106]:

```
df_cc = df_int
cluster_name = []

for i in df_int['Cluster RFM-FA']:
    if i == 0:
        cluster_name.append('Lost')
    elif i == 1:
        cluster_name.append('Potential Loyalist')
    elif i == 2:
        cluster_name.append('New User - Low Value')
    elif i == 3:
        cluster_name.append('Regular')
    elif i == 4:
        cluster_name.append('Price Sensitive')
    elif i == 5:
        cluster_name.append('Champion')
    elif i == 6:
        cluster_name.append('Loyalist')
    elif i == 7:
        cluster_name.append('Cant Lose Them')
    elif i == 8:
        cluster_name.append('New User - High Value')
    else:
        cluster_name.append('About to Lost')

df_cc['Cluster Name'] = cluster_name
```

In [107]:

```
from turtle import title


fig, axes = plt.subplots(2, 2, figsize = (15,12), sharex = True)
```

```
#Recency
sns.boxplot(x = 'Cluster Name', y = 'Recency', data = df_cc,
            order = ['Lost', 'Cant Lose Them', 'About to Lost', 'Price Sensitive',
            'New User - Low Value', 'New User - High Value', 'Regular',
            'Potential Loyalist', 'Loyalist', 'Champion'], ax = axes[0,0])

plt.setp(axes[0,0], ylabel = 'Recency (Days)')
plt.setp(axes[0,0], xlabel = '')
plt.setp(axes[0,0], title = 'Recency')

#Frequency
sns.boxplot(x = 'Cluster Name', y = 'Frequency', data = df_cc,
            order = ['Lost', 'Cant Lose Them', 'About to Lost', 'Price Sensitive',
            'New User - Low Value', 'New User - High Value', 'Regular',
            'Potential Loyalist', 'Loyalist', 'Champion'], ax = axes[0,1])
plt.setp(axes[0,1], ylabel = 'Frequency (Number of Transactions)')
plt.setp(axes[0,1], xlabel = '')
plt.setp(axes[0,1], title = 'Frequency')

#Monetary
sns.boxplot(x = 'Cluster Name', y = 'Monetary', data = df_cc,
            order = ['Lost', 'Cant Lose Them', 'About to Lost', 'Price Sensitive',
            'New User - Low Value', 'New User - High Value', 'Regular',
            'Potential Loyalist', 'Loyalist', 'Champion'], ax = axes[1,0])
plt.setp(axes[1,0], ylabel = 'Monetary (IDR)')
plt.setp(axes[1,0], xlabel = '')
plt.setp(axes[1,0], title = 'Monetary')
axes[1,0].tick_params(axis = 'x', rotation = 65)


#Financial Ability
sns.boxplot(x = 'Cluster Name', y = 'Power', data = df_cc,
            order = ['Lost', 'Cant Lose Them', 'About to Lost', 'Price Sensitive',
            'New User - Low Value', 'New User - High Value', 'Regular',
            'Potential Loyalist', 'Loyalist', 'Champion'], ax = axes[1,1])
plt.setp(axes[1,1], ylabel = 'Power (VA)')
plt.setp(axes[1,1], xlabel = '')
plt.setp(axes[1,1], title = 'Financial Ability')
axes[1,1].tick_params(axis = 'x', rotation = 65)
```



In [111]:

```
np.corrcoef(df_cc['Power'], df_cc['Monetary'])
```

Out[111]:

```
array([[1.        , 0.47313318],
       [0.47313318, 1.        ]])
```