Project : Churn rate prediction

Company target

- Churn rate is 1.7%
- retention rate is 80%

1. Importing frameworks and Data preparation

According to current situation, how to do proactive action to reduce churn rate, especially, in postpaid loyolty churner.

- Filter postpaid customers.

Loyalty customer is customers those who stayed with True more than 3 years.

- Filter loyalty customers.

In [1]:

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

df = pd.read_csv('customer_dataset.csv')
df_postpaid_loyalty = df[(df['ACCT_TYPE']=='Postpaid') & (df['AOU_DAY']>1095)]
df_postpaid_loyalty.head()
```

Out[1]:

	MNTH_ID	SUBR_ID	ACCT_TYPE	CURR_MAIN_PKG_FEE	AOU_DAY	AOU_DVC	DVC_BRAND	DVC_MODEL	DVC_GRP	DVC_CLASS	...	VC_DO
0	202009	1879	Postpaid	1099.0	10519	467	SAMSUNG	GALAXY A10 [SM-A105GDS]	Smartphone	Phone	...	
1	202010	1879	Postpaid	1099.0	10550	498	SAMSUNG	GALAXY A10 [SM-A105GDS]	Smartphone	Phone	...	
2	202011	1879	Postpaid	1099.0	10580	528	SAMSUNG	GALAXY A10 [SM-A105GDS]	Smartphone	Phone	...	

	MNTH_ID	SUBR_ID	ACCT_TYPE	CURR_MAIN_PKG_FEE	AOU_DAY	AOU_DVC	DVC_BRAND	DVC_MODEL	DVC_GRP	DVC_CLASS	...	VC_DO
3	202012	1879	Postpaid	1099.0	10611	559	SAMSUNG	GALAXY A10 [SM-A105GDS]	Smartphone	Phone	...	
4	202009	7739	Postpaid	999.0	10403	110	SAMSUNG	GALAXY A6 [SM-A600GDS]	Smartphone	Phone	...	

5 rows × 30 columns

In [2]:

```
df_postpaid_loyalty.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 180133 entries, 0 to 211339
Data columns (total 30 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   MNTH_ID               180133 non-null  int64  
 1   SUBR_ID               180133 non-null  int64  
 2   ACCT_TYPE             180133 non-null  object 
 3   CURR_MAIN_PKG_FEE     180133 non-null  float64
 4   AOU_DAY               180133 non-null  int64  
 5   AOU_DVC               180133 non-null  object 
 6   DVC_BRAND             179994 non-null  object 
 7   DVC_MODEL             179994 non-null  object 
 8   DVC_GRP               180133 non-null  object 
 9   DVC_CLASS             180133 non-null  object 
 10  DVC_SUPPORT           178599 non-null  object 
 11  MOST_USED_4W_REGION   180133 non-null  object 
 12  MOST_USED_4W_PRVNC    180133 non-null  object 
 13  MOST_USED_4W_AMPHUR   180133 non-null  object 
 14  MOST_USED_4W_TUMBON   180133 non-null  object 
 15  DTAC_RWRD_SEGMENT     180133 non-null  object 
 16  Churn_Flag            180133 non-null  object 
 17  REVENUE               180133 non-null  float64
 18  AVG3M_REVENUE         180133 non-null  float64
 19  VC_DOM_MOU            180133 non-null  int64  
 20  VC_DOM_CNT            180133 non-null  int64  
 21  AVG3M_VC_DOM_MOU      180133 non-null  float64
 22  AVG3M_VC_DOM_CNT      180133 non-null  float64
 23  DATA_MB               180133 non-null  float64
 24  AVG3M_DATA_MB         180133 non-null  float64
 25  CIN_CALLCNT           180133 non-null  int64  
```

```
 26  VOC_ACTIVEDAY        180133 non-null  int64
 27  DATA_ACTIVEDAY       180133 non-null  int64
 28  PM_PMMTHDCOMMON      180133 non-null  object
 29  PCT_CALL_DROP        180133 non-null  float64
dtypes: float64(8), int64(8), object(14)
memory usage: 42.6+ MB
```

In [3]:

```
df_postpaid_loyalty.describe()
```

Out[3]:

	MNTH_ID	SUBR_ID	CURR_MAIN_PKG_FEE	AOU_DAY	REVENUE	AVG3M_REVENUE	VC_DOM_MOU	VC_DOM_CNT	AVG:
count	180133.000000	1.801330e+05	180133.000000	180133.000000	180133.000000	180133.000000	180133.000000	180133.000000	
mean	202010.500230	1.239937e+08	522.164028	3672.153181	532.613637	541.229503	188.340448	78.756186	
std	1.116564	8.887744e+07	312.603697	1920.463407	345.475348	326.534668	314.279055	115.979303	
min	202009.000000	1.879000e+03	0.000000	1096.000000	-1520.064500	-406.688200	0.000000	0.000000	
25%	202010.000000	3.039002e+07	299.000000	2021.000000	299.000000	309.333300	41.000000	20.000000	
50%	202010.000000	1.195342e+08	449.000000	3253.000000	463.500000	482.333300	107.000000	48.000000	
75%	202011.000000	2.146236e+08	699.000000	5118.000000	699.000000	699.000000	225.000000	96.000000	
max	202012.000000	2.694149e+08	5607.480000	10700.000000	17093.450000	7339.618700	19116.000000	6185.000000	

- Drop features, which are not useful.

In [4]:

```
df_fs = df_postpaid_loyalty.drop([
    'MNTH_ID', 'SUBR_ID', 'ACCT_TYPE', 'DVC_BRAND', 'DVC_MODEL', 'MOST_USED_4W_PRVNC', 'MOST_USED_4W_AMPHUR', 'MOST_USED_
], axis=1)
```

In [5]:

```
df_fs.head(5)
```

Out[5]:

	CURR_MAIN_PKG_FEE	AOU_DAY	AOU_DVC	DVC_GRP	DVC_CLASS	DVC_SUPPORT	MOST_USED_4W_REGION	DTAC_RWRD_SEGMENT	Churn
0	1099.0	10519	467	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue	01) N Custo
1	1099.0	10550	498	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue	01) N Custo

	CURR_MAIN_PKG_FEE	AOU_DAY	AOU_DVC	DVC_GRP	DVC_CLASS	DVC_SUPPORT	MOST_USED_4W_REGION	DTAC_RWRD_SEGMENT	Churn
2	1099.0	10580	528	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue	01) N Custo
3	1099.0	10611	559	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue	01) N Custo
4	999.0	10403	110	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue	01) N Custo

5 rows × 22 columns

In [6]:

```
df_fs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 180133 entries, 0 to 211339
Data columns (total 22 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   CURR_MAIN_PKG_FEE    180133 non-null  float64
 1   AOU_DAY              180133 non-null  int64
 2   AOU_DVC              180133 non-null  object
 3   DVC_GRP              180133 non-null  object
 4   DVC_CLASS            180133 non-null  object
 5   DVC_SUPPORT          178599 non-null  object
 6   MOST_USED_4W_REGION  180133 non-null  object
 7   DTAC_RWRD_SEGMENT    180133 non-null  object
 8   Churn_Flag           180133 non-null  object
 9   REVENUE              180133 non-null  float64
 10  AVG3M_REVENUE        180133 non-null  float64
 11  VC_DOM_MOU           180133 non-null  int64
 12  VC_DOM_CNT           180133 non-null  int64
 13  AVG3M_VC_DOM_MOU     180133 non-null  float64
 14  AVG3M_VC_DOM_CNT     180133 non-null  float64
 15  DATA_MB              180133 non-null  float64
 16  AVG3M_DATA_MB        180133 non-null  float64
 17  CIN_CALLCNT          180133 non-null  int64
 18  VOC_ACTIVEDAY        180133 non-null  int64
 19  DATA_ACTIVEDAY       180133 non-null  int64
 20  PM_PMMTHDCOMMON      180133 non-null  object
 21  PCT_CALL_DROP        180133 non-null  float64
dtypes: float64(8), int64(6), object(8)
memory usage: 31.6+ MB
```

In [7]:
```
df_fs.columns
```

Out[7]:
```
Index(['CURR_MAIN_PKG_FEE', 'AOU_DAY', 'AOU_DVC', 'DVC_GRP', 'DVC_CLASS',
       'DVC_SUPPORT', 'MOST_USED_4W_REGION', 'DTAC_RWRD_SEGMENT', 'Churn_Flag',
       'REVENUE', 'AVG3M_REVENUE', 'VC_DOM_MOU', 'VC_DOM_CNT',
       'AVG3M_VC_DOM_MOU', 'AVG3M_VC_DOM_CNT', 'DATA_MB', 'AVG3M_DATA_MB',
       'CIN_CALLCNT', 'VOC_ACTIVEDAY', 'DATA_ACTIVEDAY', 'PM_PMMTHDCOMMON',
       'PCT_CALL_DROP'],
      dtype='object')
```

In [8]:
```
df_fs['AOU_DVC'] = pd.to_numeric(df_fs['AOU_DVC'], errors='coerce')
```

- Data cleaning

In [9]:
```
df_fs.drop_duplicates(inplace=True)
df_fs.dropna(inplace=True)
df_fs = df_fs[~df_fs.isin(['No Information']).any(axis=1)]
df_fs
```

Out[9]:

	CURR_MAIN_PKG_FEE	AOU_DAY	AOU_DVC	DVC_GRP	DVC_CLASS	DVC_SUPPORT	MOST_USED_4W_REGION	DTAC_RWRD_SEGMENT
0	1099.0	10519	467.0	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue
1	1099.0	10550	498.0	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue
2	1099.0	10580	528.0	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue
3	1099.0	10611	559.0	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue
4	999.0	10403	110.0	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue
...	...	...	...	...	...	...	...	...
211323	499.0	1096	751.0	Smartphone	Phone	4G 2300MHz	BMA	Silver
211327	299.0	1096	22.0	Smartphone	Phone	4G 2300MHz	Northeast	Gold

	CURR_MAIN_PKG_FEE	AOU_DAY	AOU_DVC	DVC_GRP	DVC_CLASS	DVC_SUPPORT	MOST_USED_4W_REGION	DTAC_RWRD_SEGMENT
211331	399.0	1096	762.0	Smartphone	Phone	4G 2300MHz	North	Silver
211335	399.0	1096	333.0	Smartphone	Phone	4G 2300MHz	BMA	Silver
211339	699.0	1096	1095.0	Tablet	Tablet	4G 2300MHz	Central&West	Silver

178503 rows × 22 columns

In [10]:

```
numerical_features_list = ['CURR_MAIN_PKG_FEE', 'AOU_DAY', 'AOU_DVC', 'REVENUE', 'AVG3M_REVENUE', 'VC_DOM_MOU', 'VC_DOM_C
                           'AVG3M_VC_DOM_MOU', 'AVG3M_VC_DOM_CNT', 'DATA_MB', 'AVG3M_DATA_MB', 'CIN_CALLCNT', 'VOC_ACTIVE
                           'DATA_ACTIVEDAY', 'PCT_CALL_DROP']

for col in df_fs.columns:
    if col not in numerical_features_list:
        print(col, df_fs[col].unique())
    print("-"*50)
```

```
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
DVC_GRP ['Smartphone' 'Feature Phone' 'Tablet' 'Standalone connectivity device']
--------------------------------------------------
DVC_CLASS ['Phone' 'Tablet' 'Connectivity Devices']
--------------------------------------------------
DVC_SUPPORT ['4G 2300MHz' '3G 2100MHz' '4G 2100MHz' '2G 1800MHz' '4G 1800MHz'
 '3G 850MHz']
--------------------------------------------------
MOST_USED_4W_REGION ['BMA' 'East' 'South' 'Central&West' 'unknown' 'Northeast' 'North']
--------------------------------------------------
DTAC_RWRD_SEGMENT ['Platinum Blue' 'Silver' 'Gold' 'Welcome']
--------------------------------------------------
Churn_Flag ['01) Normal Customers' '02) Churn Customers']
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
```

```
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
PM_PMMTHDCOMMON ['VISA' 'Bank transfer' 'Cash' 'Recurring RC' 'Cheque Counter'
 'Recurring DD' 'MASTER' 'Cheque Mail' 'AMEX' 'Debit Card'
 'Payment After Adjust' 'Pre to Post' 'JCB']
--------------------------------------------------
--------------------------------------------------
```

In [11]:

```
print(df_fs.isnull().sum())
```

```
CURR_MAIN_PKG_FEE      0
AOU_DAY                0
AOU_DVC                0
DVC_GRP                0
DVC_CLASS              0
DVC_SUPPORT            0
MOST_USED_4W_REGION    0
DTAC_RWRD_SEGMENT      0
Churn_Flag             0
REVENUE                0
AVG3M_REVENUE          0
VC_DOM_MOU             0
VC_DOM_CNT             0
AVG3M_VC_DOM_MOU       0
AVG3M_VC_DOM_CNT       0
DATA_MB                0
AVG3M_DATA_MB          0
CIN_CALLCNT            0
VOC_ACTIVEDAY          0
DATA_ACTIVEDAY         0
PM_PMMTHDCOMMON        0
PCT_CALL_DROP          0
dtype: int64
```

In [12]:

```
df_fs['Churn_Flag'].value_counts()
```

Out[12]:

```
01) Normal Customers    175494
02) Churn Customers       3009
Name: Churn_Flag, dtype: int64
```

2. Exploratory data analysis (EDA)

In [13]:

```
df_fs.shape
```

Out[13]:
```
(178503, 22)
```

In [14]:
```
df_fs.columns
```

Out[14]:
```
Index(['CURR_MAIN_PKG_FEE', 'AOU_DAY', 'AOU_DVC', 'DVC_GRP', 'DVC_CLASS',
       'DVC_SUPPORT', 'MOST_USED_4W_REGION', 'DTAC_RWRD_SEGMENT', 'Churn_Flag',
       'REVENUE', 'AVG3M_REVENUE', 'VC_DOM_MOU', 'VC_DOM_CNT',
       'AVG3M_VC_DOM_MOU', 'AVG3M_VC_DOM_CNT', 'DATA_MB', 'AVG3M_DATA_MB',
       'CIN_CALLCNT', 'VOC_ACTIVEDAY', 'DATA_ACTIVEDAY', 'PM_PMMTHDCOMMON',
       'PCT_CALL_DROP'],
      dtype='object')
```

In [15]:
```
df_fs.head()
```

Out[15]:

	CURR_MAIN_PKG_FEE	AOU_DAY	AOU_DVC	DVC_GRP	DVC_CLASS	DVC_SUPPORT	MOST_USED_4W_REGION	DTAC_RWRD_SEGMENT	Churn
0	1099.0	10519	467.0	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue	01) N Custo
1	1099.0	10550	498.0	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue	01) N Custo
2	1099.0	10580	528.0	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue	01) N Custo
3	1099.0	10611	559.0	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue	01) N Custo
4	999.0	10403	110.0	Smartphone	Phone	4G 2300MHz	BMA	Platinum Blue	01) N Custo

5 rows × 22 columns

In [16]:
```
df_fs.describe()
```

Out[16]:

	CURR_MAIN_PKG_FEE	AOU_DAY	AOU_DVC	REVENUE	AVG3M_REVENUE	VC_DOM_MOU	VC_DOM_CNT	AVG3M_VC_DOM_M
count	178503.000000	178503.000000	178503.000000	178503.000000	178503.000000	178503.000000	178503.000000	178503.000
mean	523.807431	3672.628309	627.353025	534.316257	542.878558	188.492485	78.693691	190.697

	CURR_MAIN_PKG_FEE	AOU_DAY	AOU_DVC	REVENUE	AVG3M_REVENUE	VC_DOM_MOU	VC_DOM_CNT	AVG3M_VC_DOM_M
std	312.610852	1920.474449	502.803347	344.979079	326.018580	314.517588	115.152689	303.103
min	0.000000	1096.000000	1.000000	-1520.064500	-406.688200	0.000000	0.000000	0.000
25%	299.000000	2021.000000	222.000000	299.000000	313.808600	41.000000	20.000000	48.666
50%	449.000000	3253.000000	521.000000	468.000000	487.661900	108.000000	48.000000	114.000
75%	699.000000	5118.000000	929.000000	699.000000	699.000000	225.000000	96.000000	225.666
max	5607.480000	10700.000000	3684.000000	17093.450000	7339.618700	19116.000000	6185.000000	15133.333

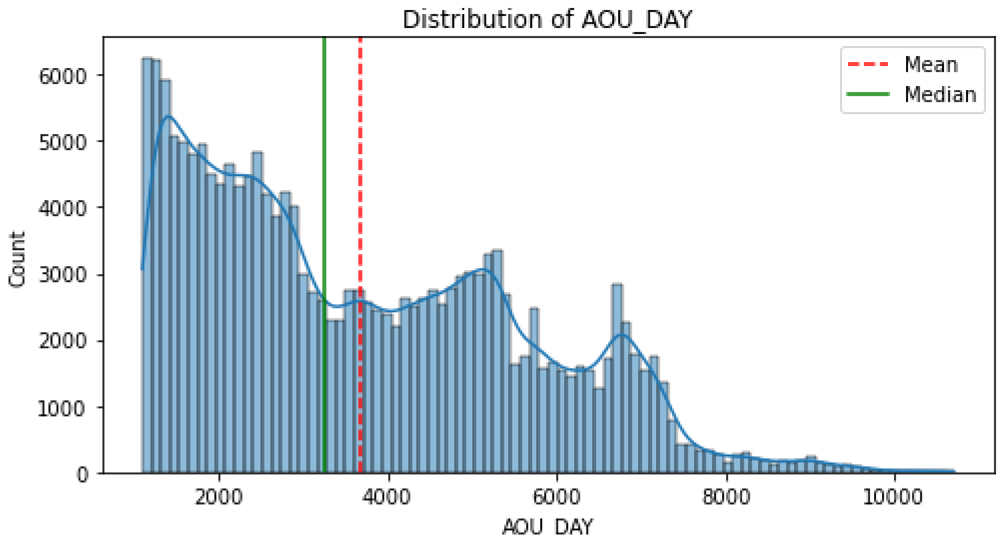
- Understand the distribution of features

In [17]:

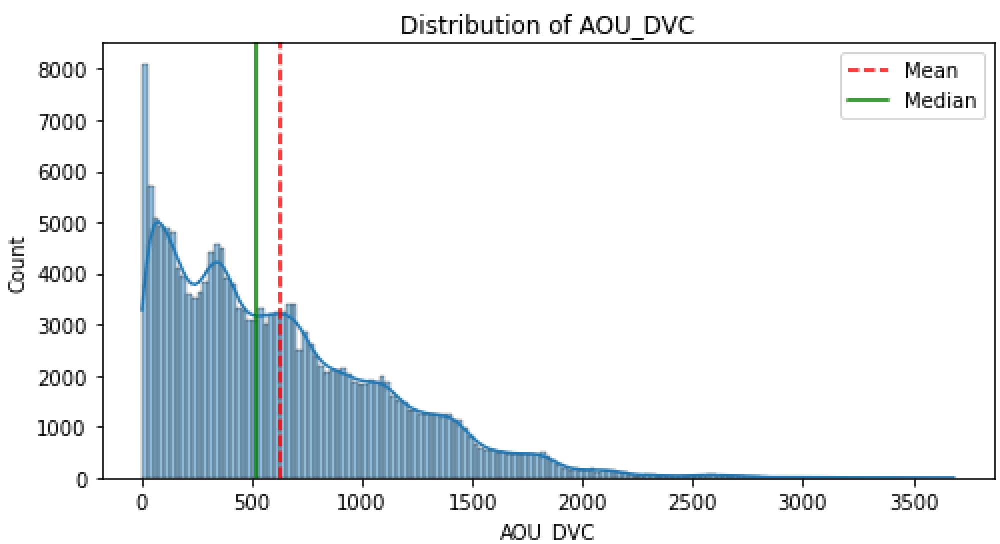
```
def plot_histogram(df_fs, column_name):

  plt.figure(figsize=(8, 4))
  sns.histplot(df_fs[column_name], kde=True)
  plt.title(f"Distribution of {column_name}")

  # calculate the mean and median values for the columns
  col_mean = df_fs[column_name].mean()
  col_median = df_fs[column_name].median()

  # add vertical lines for mean and median
  plt.axvline(col_mean, color="red", linestyle="--", label="Mean")
  plt.axvline(col_median, color="green", linestyle="-", label="Median")

  plt.legend()

  plt.show()
```

In [18]:

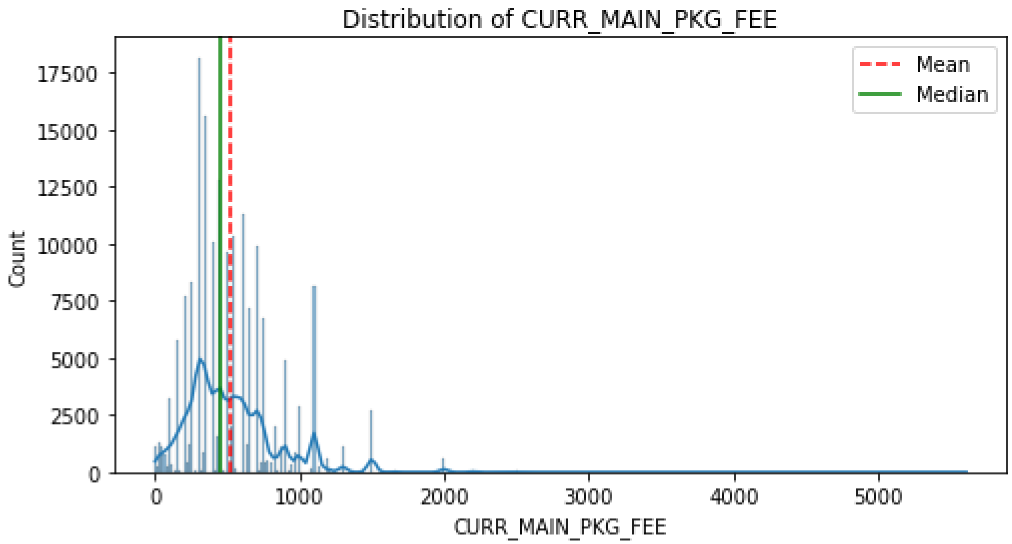
```
plot_histogram(df_fs, 'AOU_DAY')
```



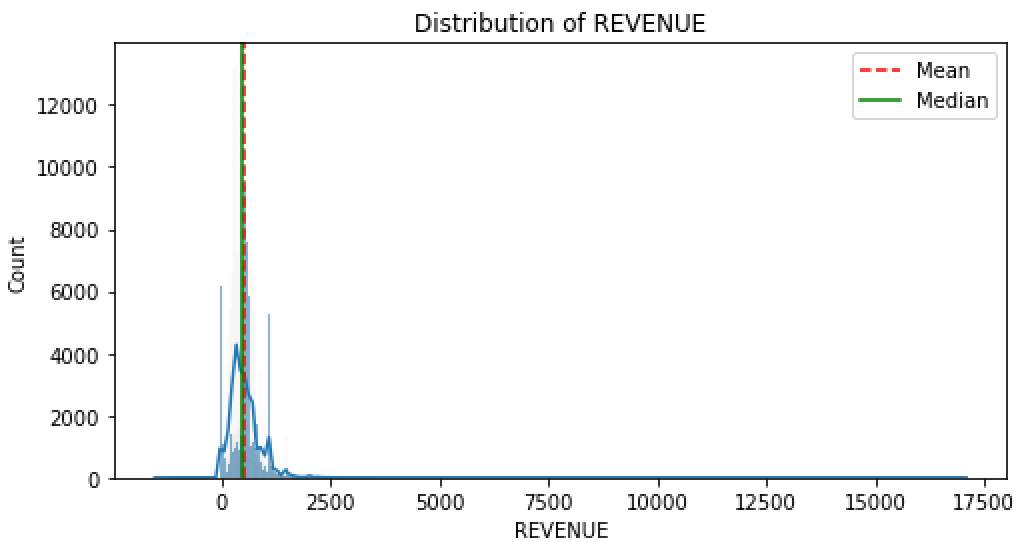
In [19]:
```
plot_histogram(df_fs, 'AOU_DVC')
```



In [20]:
```
plot_histogram(df_fs, 'CURR_MAIN_PKG_FEE')
```

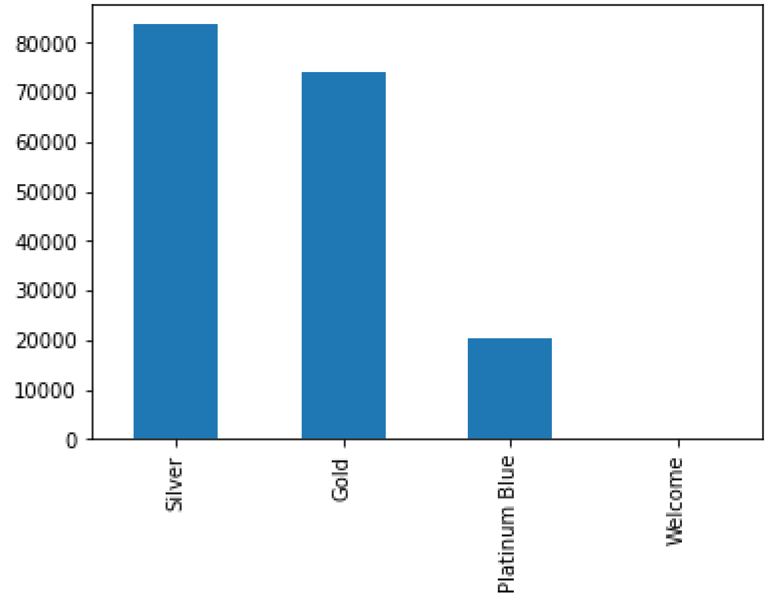


In [21]:
```
plot_histogram(df_fs, 'REVENUE')
```



In [22]:
```
df_fs['DTAC_RWRD_SEGMENT'].value_counts().plot(kind='bar')
```

Out[22]: <AxesSubplot:>

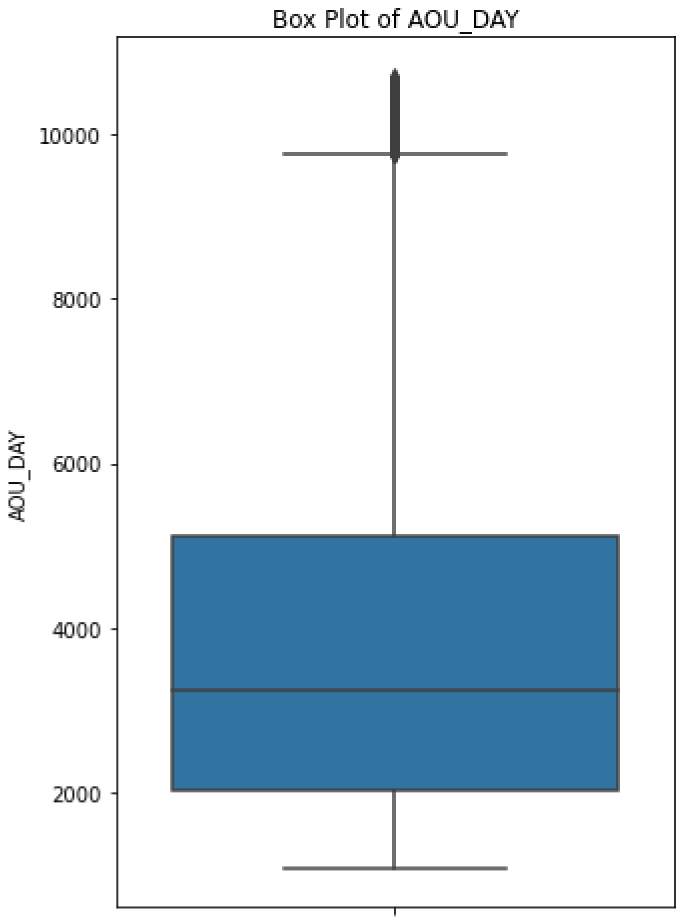


In [73]:

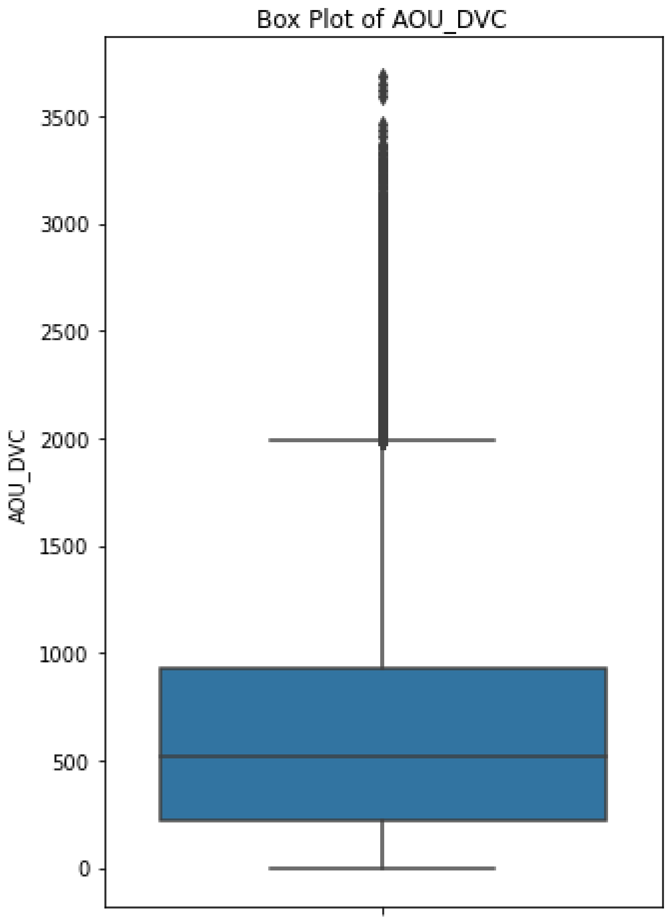
```
def plot_boxplot(df_fs, column_name):

  plt.figure(figsize=(5, 8))
  sns.boxplot(y=df_fs[column_name])
  plt.title(f"Box Plot of {column_name}")
  plt.ylabel(column_name)
  plt.show
```

In [74]:

```
plot_boxplot(df_fs, "AOU_DAY")
```

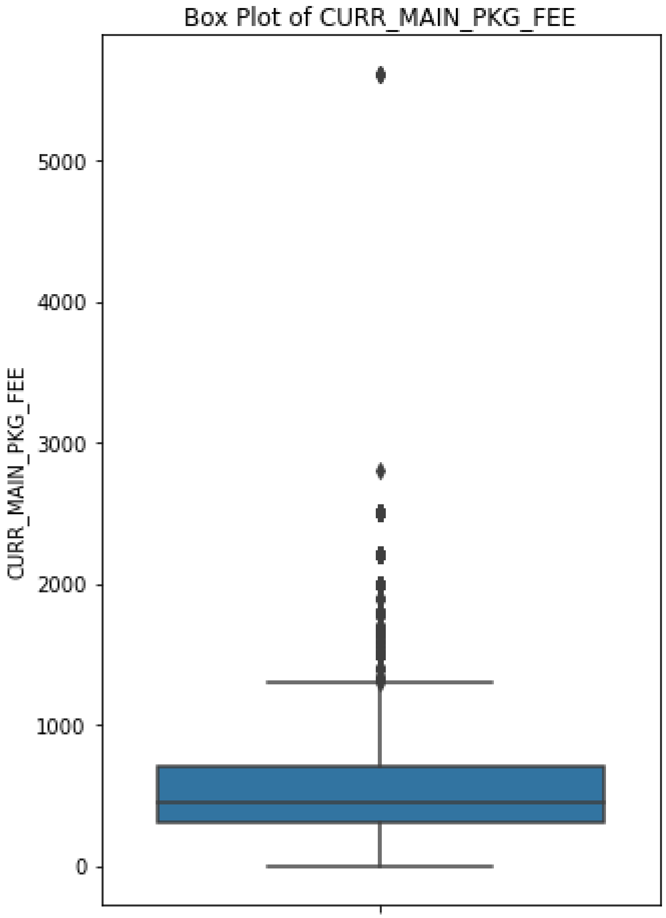


In [75]:
```
plot_boxplot(df_fs, "AOU_DVC")
```



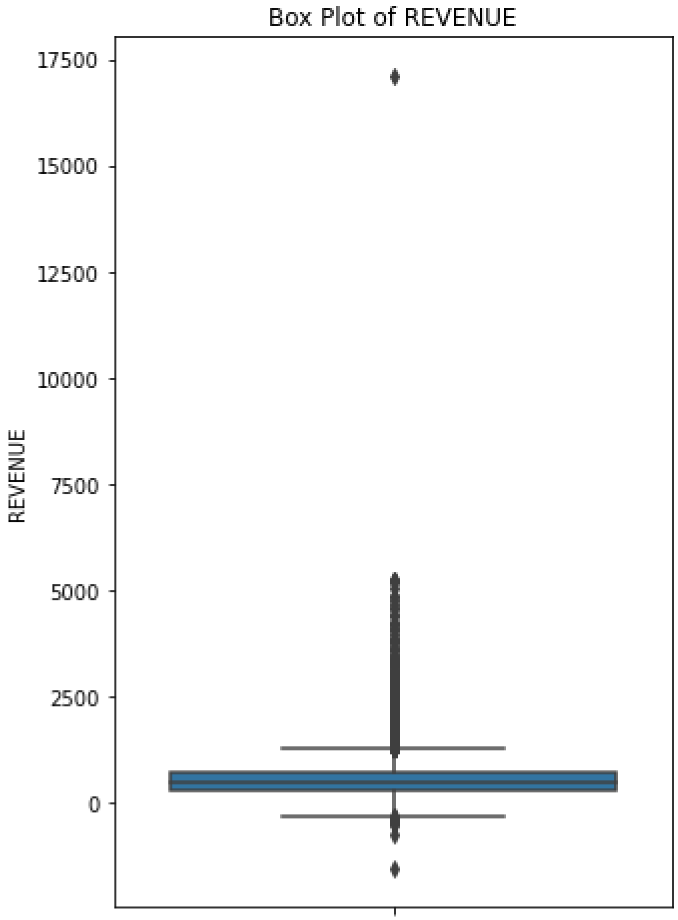
In [76]:

```
plot_boxplot(df_fs, "CURR_MAIN_PKG_FEE")
```



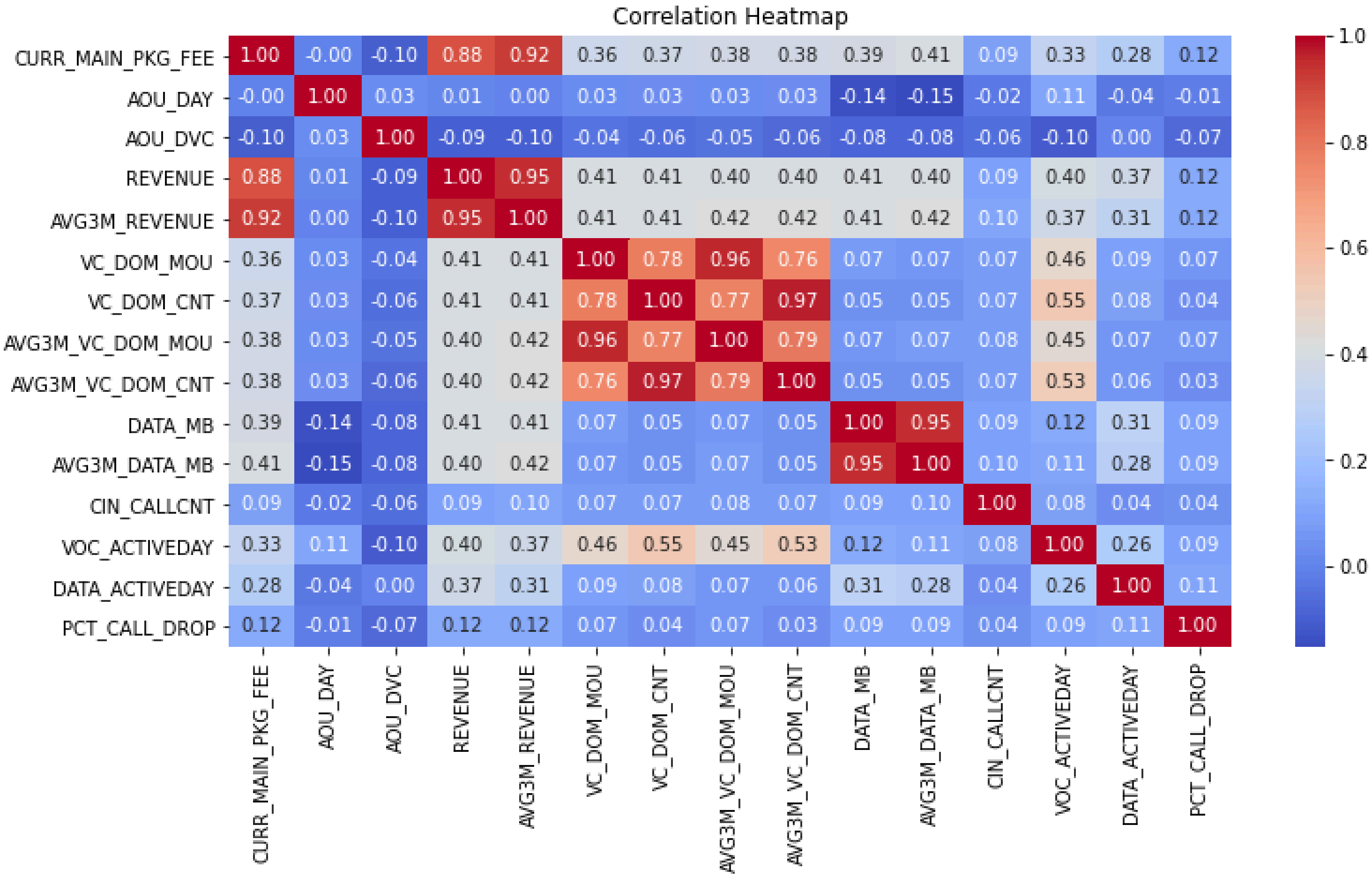
In [77]:

```
plot_boxplot(df_fs, "REVENUE")
```



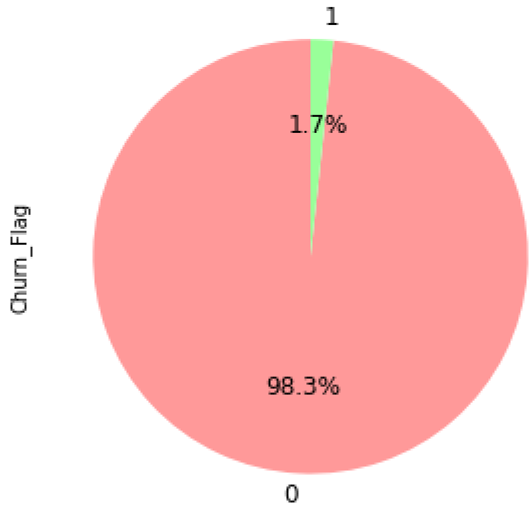
In [28]:

```
plt.figure(figsize=(12, 6))
sns.heatmap(df_fs[numerical_features_list].corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



In [49]:

```
df_pie = df_fs['Churn_Flag'].value_counts()

plt.figure(figsize=(5, 5))
df_pie.plot(kind='pie', autopct='%1.1f%%', startangle=90, colors=['#ff9999','#99ff99'], fontsize = 12)
plt.title('', size = 20)
plt.show()
```



3. Model conducting

In [29]:
```
df_fs['Churn_Flag'] = df_fs['Churn_Flag'].replace({'02) Churn Customers': 1, '01) Normal Customers': 0})
df_fs['Churn_Flag'].value_counts()
```

Out[29]:
```
0    175494
1      3009
Name: Churn_Flag, dtype: int64
```

- Label encoding for categorical features

In [30]:
```
object_columns = df_fs.select_dtypes(include="object").columns
object_columns
```

Out[30]:
```
Index(['DVC_GRP', 'DVC_CLASS', 'DVC_SUPPORT', 'MOST_USED_4W_REGION',
       'DTAC_RWRD_SEGMENT', 'PM_PMMTHDCOMMON'],
      dtype='object')
```

In [31]:
```
from sklearn.preprocessing import LabelEncoder
import pickle

encoders = {}
```

```
for column in object_columns:
  label_encoder = LabelEncoder()
  df_fs[column] = label_encoder.fit_transform(df_fs[column])
  encoders[column] = label_encoder


# save the encoders to a pickle file
with open("encoders.pkl", "wb") as f:
  pickle.dump(encoders, f)
```

In [32]:

```
encoders
```

Out[32]:

```
{'DVC_GRP': LabelEncoder(),
 'DVC_CLASS': LabelEncoder(),
 'DVC_SUPPORT': LabelEncoder(),
 'MOST_USED_4W_REGION': LabelEncoder(),
 'DTAC_RWRD_SEGMENT': LabelEncoder(),
 'PM_PMMTHDCOMMON': LabelEncoder()}
```

In [33]:

```
df_fs.head()
```

Out[33]:

	CURR_MAIN_PKG_FEE	AOU_DAY	AOU_DVC	DVC_GRP	DVC_CLASS	DVC_SUPPORT	MOST_USED_4W_REGION	DTAC_RWRD_SEGMENT	Churn_F
0	1099.0	10519	467.0	1	1	5	0	1	
1	1099.0	10550	498.0	1	1	5	0	1	
2	1099.0	10580	528.0	1	1	5	0	1	
3	1099.0	10611	559.0	1	1	5	0	1	
4	999.0	10403	110.0	1	1	5	0	1	

5 rows × 22 columns

- Train test split

In [34]:

```
X = df_fs.drop(columns='Churn_Flag')
y = df_fs['Churn_Flag']
```

```
print(X.shape)
print(y.shape)
```

```
(178503, 21)
(178503,)
```

In [35]:

```
from sklearn.model_selection import train_test_split, cross_val_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(X_train.shape)
print(y_train.shape)
```

```
(142802, 21)
(142802,)
```

- Synthetic Minority Oversampling Technique (SMOTE)

In [36]:

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

print(y_train_smote.shape)
print(y_train_smote.value_counts())
```

```
(280790,)
0    140395
1    140395
Name: Churn_Flag, dtype: int64
```

- Model training

In [41]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42)
}
```

In [42]:
```
cv_scores = {}

for model_name, model in models.items():
  print(f"Training {model_name}")
  scores = cross_val_score(model, X_train_smote, y_train_smote, cv=5, scoring="accuracy")
  cv_scores[model_name] = scores
  print(f"{model_name} cross-validation accuracy: {np.mean(scores):.2f}")
  print("-"*70)
```

```
Training Decision Tree
Decision Tree cross-validation accuracy: 1.00
----------------------------------------------------------------------
Training Random Forest
Random Forest cross-validation accuracy: 1.00
----------------------------------------------------------------------
Training XGBoost
XGBoost cross-validation accuracy: 1.00
----------------------------------------------------------------------
```

In [43]:
```
cv_scores
```

Out[43]:
```
{'Decision Tree': array([0.99864668, 0.99960825, 0.99976851, 0.99985754, 0.99966167]),
 'Random Forest': array([0.99964386, 0.9997507 , 0.99982193, 0.99991097, 0.9997329 ]),
 'XGBoost': array([0.99966167, 0.99971509, 0.9997507 , 0.99982193, 0.99960825])}
```

In [44]:
```
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train_smote, y_train_smote)
```

Out[44]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

4. Model evaluation

In [45]:
```
y_test_pred = rf.predict(X_test)

print("Accuracy Score:\n", accuracy_score(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Classification Report:\n", classification_report(y_test, y_test_pred))
```

```
Accuracy Score:
 0.9994397916024761
Confsuion Matrix:
 [[35079    20]
 [    0   602]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     35099
           1       0.97      1.00      0.98       602

    accuracy                           1.00     35701
   macro avg       0.98      1.00      0.99     35701
weighted avg       1.00      1.00      1.00     35701
```

To do:

1. Apply hyperparameter tuning to optimize model performance.
2. Experiment with different model architectures to identify the most effective one.
3. Use downsampling techniques to address class imbalance.
4. Implement strategies to reduce overfitting and improve generalization.
5. Utilize stratified K-Fold cross-validation for more reliable model evaluation.

5. Customer segmentation

- Categorized customers, who stay

In [51]:

```
df_st = df_fs[df_fs['Churn_Flag'] == 0]
df_st.head()
```

Out[51]:

	CURR_MAIN_PKG_FEE	AOU_DAY	AOU_DVC	DVC_GRP	DVC_CLASS	DVC_SUPPORT	MOST_USED_4W_REGION	DTAC_RWRD_SEGMENT	Churn_F
0	1099.0	10519	467.0	1	1	5	0	1	
1	1099.0	10550	498.0	1	1	5	0	1	
2	1099.0	10580	528.0	1	1	5	0	1	
3	1099.0	10611	559.0	1	1	5	0	1	
4	999.0	10403	110.0	1	1	5	0	1	

5 rows × 22 columns

In [52]:
```
df_st.shape
```

Out[52]:
```
(175494, 22)
```

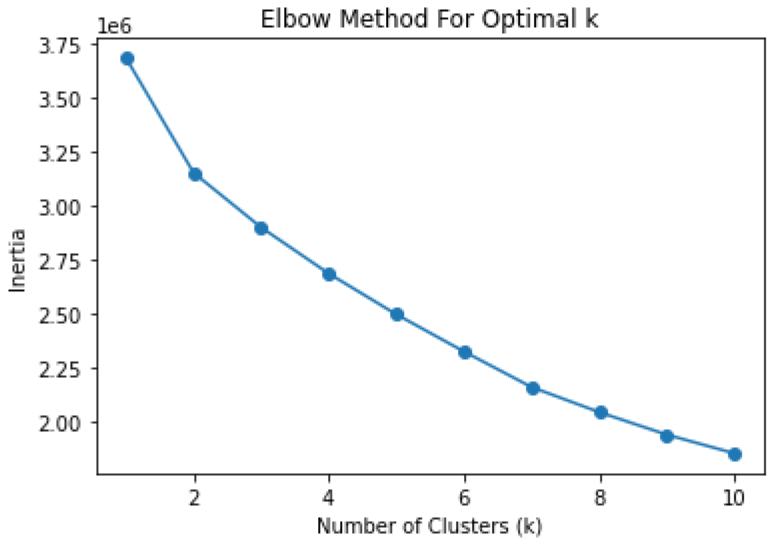
In [56]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

X = df_st.drop(columns='Churn_Flag')

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

- Figure out an optimal point

In [58]:
```
inertia = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Method
plt.plot(k_range, inertia, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method For Optimal k')
plt.show()
```

```
C:\Users\ADMIN\AppData\Roaming\Python\Python38\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will 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\ADMIN\AppData\Roaming\Python\Python38\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will 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\ADMIN\AppData\Roaming\Python\Python38\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default value of `n_init` will 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\ADMIN\AppData\Roaming\Python\Python38\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default
```

```
value of `n_init` will 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\ADMIN\AppData\Roaming\Python\Python38\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default
value of `n_init` will 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\ADMIN\AppData\Roaming\Python\Python38\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default
value of `n_init` will 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\ADMIN\AppData\Roaming\Python\Python38\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default
value of `n_init` will 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\ADMIN\AppData\Roaming\Python\Python38\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default
value of `n_init` will 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\ADMIN\AppData\Roaming\Python\Python38\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default
value of `n_init` will 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\ADMIN\AppData\Roaming\Python\Python38\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default
value of `n_init` will 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)
```



In [61]:

```
kmeans = KMeans(n_clusters=5, random_state=42)
df_st['Cluster'] = kmeans.fit_predict(X_scaled)
```

```
C:\Users\ADMIN\AppData\Roaming\Python\Python38\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The default
value of `n_init` will 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)
<ipython-input-61-081a115a96a4>: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-v
iew-versus-a-copy
  df_st['Cluster'] = kmeans.fit_predict(X_scaled)
```

In [62]:

```
cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=X.columns)
print("Cluster Centers (Centroids):")
print(cluster_centers)
```

```
Cluster Centers (Centroids):
   CURR_MAIN_PKG_FEE   AOU_DAY   AOU_DVC   DVC_GRP  DVC_CLASS  DVC_SUPPORT  \
0          -0.510445 -0.496990 -0.012716 -0.045965   0.163243    -0.061491
1           1.168251 -0.285142 -0.165067 -0.055352   0.065311     0.263982
2           1.668617  0.097881 -0.132899 -0.116135   0.054526     0.140319
3          -0.130720  0.836246  0.051033 -0.087462   0.071069     0.110117
4          -1.427695 -1.004254  1.289711  3.274363  -5.519996    -3.470822

   MOST_USED_4W_REGION  DTAC_RWRD_SEGMENT   REVENUE  AVG3M_REVENUE  ...  \
0             0.076695           0.962281 -0.522215      -0.531044  ...
1            -0.150665          -0.427131  1.167230       1.183017  ...
2            -0.007134          -0.211030  1.789458       1.820152  ...
3            -0.016509          -0.995528 -0.130737      -0.128832  ...
4             0.145196           1.013655 -1.382877      -1.438633  ...

   VC_DOM_CNT  AVG3M_VC_DOM_MOU  AVG3M_VC_DOM_CNT   DATA_MB  AVG3M_DATA_MB  \
0   -0.270602         -0.267479         -0.271259 -0.286204      -0.286306
1    0.206188          0.219908          0.209136  1.163129       1.177253
2    3.491926          3.470099          3.492057  0.023961       0.021879
3   -0.110546         -0.123284         -0.110368 -0.273698      -0.279691
4   -0.669763         -0.614825         -0.688362 -0.760467      -0.792348

   CIN_CALLCNT  VOC_ACTIVEDAY  DATA_ACTIVEDAY  PM_PMMTHDCOMMON  PCT_CALL_DROP
0    -0.069949      -0.281305       -0.253073        -0.055759      -0.089757
1     0.238471       0.438188        0.394719        -0.150501       0.280229
2     0.180587       1.180238        0.115629         0.244005       0.101554
3    -0.052597       0.088184        0.110715         0.081573      -0.040614
4    -0.293429      -1.972550       -0.593347         0.860256      -0.345078

[5 rows x 21 columns]
```

In [64]:

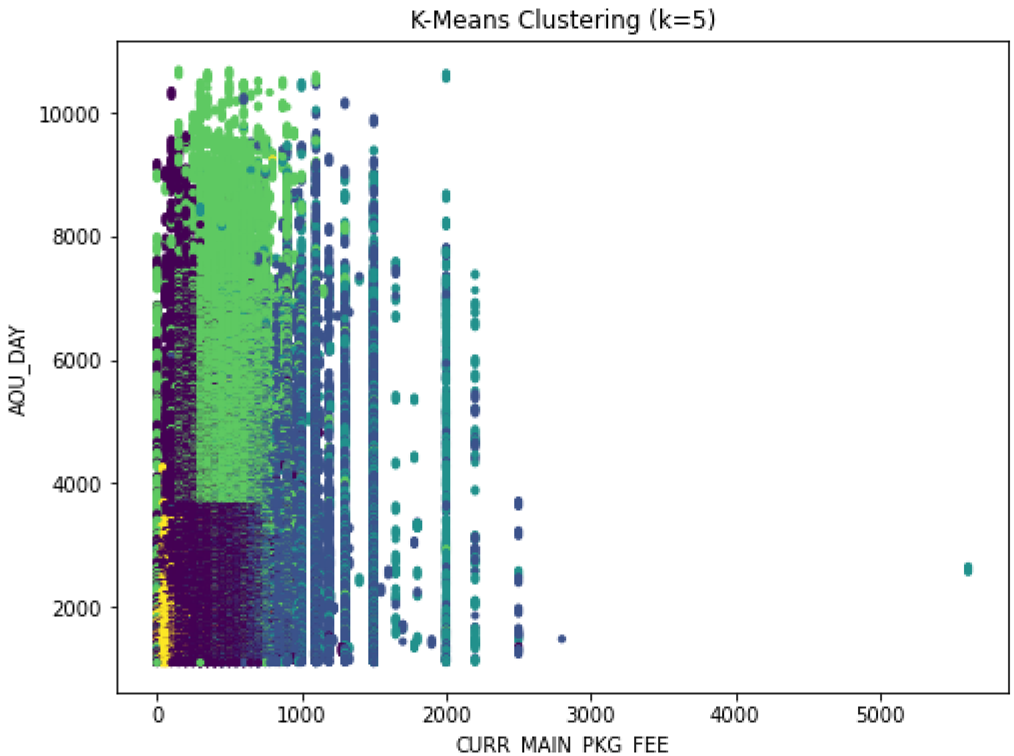
```
cluster_summary = df_st.groupby('Cluster').mean()  # Average for each cluster
print("\nCluster Summary (Average values per cluster):")
print(cluster_summary)
```

```
Cluster Summary (Average values per cluster):
         CURR_MAIN_PKG_FEE      AOU_DAY      AOU_DVC    DVC_GRP  DVC_CLASS  \
```

```
Cluster
0            365.786732  2728.004712   620.082630  1.014341   1.022949
1            890.345556  3136.309154   543.762973  1.011574   1.005331
2           1046.892168  3871.231666   560.168852  0.993686   1.003400
3            484.368021  5290.843931   652.005568  1.002067   1.006388
4             79.205726  1753.207202  1272.420897  2.000000   0.000000

         DVC_SUPPORT  MOST_USED_4W_REGION  DTAC_RWRD_SEGMENT  Churn_Flag  \
Cluster
0           4.650291             1.919359           1.957371         0.0
1           4.967662             1.496908           0.650144         0.0
2           4.848308             1.763639           0.853327         0.0
3           4.817612             1.745965           0.115472         0.0
4           1.324085             2.046635           2.005608         0.0

             REVENUE  ...  VC_DOM_CNT  AVG3M_VC_DOM_MOU  AVG3M_VC_DOM_CNT  \
Cluster               ...
0         365.562444  ...   48.240425        110.423989         49.272431
1         941.231515  ...  103.491400        258.974407        103.826579
2        1153.402936  ...  484.199126       1249.517997        476.497949
3         498.843061  ...   66.749398        154.305238         67.504372
4          72.341502  ...    2.030401          4.657713          1.960449

              DATA_MB  AVG3M_DATA_MB  CIN_CALLCNT  VOC_ACTIVEDAY  \
Cluster
0        12011.480431   12335.799479     0.905576      16.164919
1        47843.794290   46911.936373     2.094599      22.850592
2        19693.985545   19626.491304     1.874373      29.738546
3        12318.506255   12489.781942     0.972725      19.594524
4          275.739687     371.205955     0.043388       0.457202

         DATA_ACTIVEDAY  PM_PMMTHDCOMMON  PCT_CALL_DROP
Cluster
0             24.455587         1.878214       0.416897
1             30.130500         1.612829       1.019842
2             27.694026         2.718472       0.729498
3             27.640706         2.263070       0.496850
4             21.473731         4.441558       0.000000

[5 rows x 22 columns]
```

In [68]:

```python
plt.figure(figsize=(8,6))
plt.scatter(df_st.iloc[:, 0], df_st.iloc[:, 1], c=df_st['Cluster'], cmap='viridis', s=10)
plt.title('K-Means Clustering (k=5)')
plt.xlabel(X.columns[0])
plt.ylabel(X.columns[1])
plt.show()
```



To do:

1. Tailor benefits and offers for each customer segment based on their specific needs and behavior.
2. Monitor and evaluate the effectiveness of each offer or strategy using measurable KPIs.
3. Identify which offers drive positive results, and proactively present those to customers who are at high risk of churning.