Churn Prediction-Iranian

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df=pd.read_csv('Iranian Customer Churn.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Call Failure	Complains	Subscription Length	Charge Amount	Seconds of Use	Frequency of use	Frequency of SMS	Distinct Called Numbers	Age Group
0	8	0	38	0	4370	71	5	17	3
1	0	0	39	0	318	5	7	4	2
2	10	0	37	0	2453	60	359	24	3
3	10	0	38	0	4198	66	1	35	1
4	3	0	38	0	2393	58	2	33	1

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
Call  Failure              0
Complains                  0
Subscription  Length       0
Charge  Amount             0
Seconds of Use             0
Frequency of use           0
Frequency of SMS           0
Distinct Called Numbers    0
Age Group                  0
Tariff Plan                0
Status                     0
Age                        0
Customer Value             0
Churn                      0
dtype: int64
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3150 entries, 0 to 3149
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Call  Failure            3150 non-null   int64
 1   Complains                3150 non-null   int64
 2   Subscription  Length     3150 non-null   int64
 3   Charge  Amount           3150 non-null   int64
 4   Seconds of Use           3150 non-null   int64
 5   Frequency of use         3150 non-null   int64
 6   Frequency of SMS         3150 non-null   int64
 7   Distinct Called Numbers  3150 non-null   int64
 8   Age Group                3150 non-null   int64
 9   Tariff Plan              3150 non-null   int64
 10  Status                   3150 non-null   int64
 11  Age                      3150 non-null   int64
 12  Customer Value           3150 non-null   float64
 13  Churn                    3150 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 344.7 KB
```

In [6]:
```
#Renaming the columns to lower case
df=df.rename(columns={"Call  Failure": "call_failure", "Complains": "complains", "S
                      "Seconds of Use": "total_sec_calls", "Frequency of use": "total_
                      "Age Group": "age_group", "Tariff Plan": "tariff_plan", "Status'
```

In [7]:
```
df['customer_id'] = range(1, len(df) + 1)

# Move the 'customer_id' column to the first position
column_order = ['customer_id'] + [col for col in df.columns if col != 'customer_id'
df = df[column_order]
```

In [8]:
```
df.nunique()
```

Out[8]:
```
customer_id           3150
call_failure            37
complains                2
subs_len                45
charge_amount           11
total_sec_calls       1756
total_num_calls        242
total_num_sms          405
distinct_call_nums      92
age_group                5
tariff_plan              2
status                   2
age                      5
customer_value        2654
Churn                    2
dtype: int64
```

In [9]:
```
df=df[df['customer_value'] != 0]
```

In [10]:
```
duplicate_rows = df[df.duplicated()]

# Display duplicate rows (if any)
if not duplicate_rows.empty:
    print("Duplicate Rows:")
    print(duplicate_rows)
else:
    print("No duplicate rows found.")
```

```
# Drop duplicate rows and keep the first occurrence
df_no_duplicates = df.drop_duplicates()

# Display the resulting DataFrame without duplicates
print("\nDataFrame after removing duplicates:")
print(df_no_duplicates)
```

```
No duplicate rows found.

DataFrame after removing duplicates:
      customer_id  call_failure  complains  subs_len  charge_amount  \
0               1             8          0        38              0
1               2             0          0        39              0
2               3            10          0        37              0
3               4            10          0        38              0
4               5             3          0        38              0
...           ...           ...        ...       ...            ...
3145         3146            21          0        19              2
3146         3147            17          0        17              1
3147         3148            13          0        18              4
3148         3149             7          0        11              2
3149         3150             8          1        11              2

      total_sec_calls  total_num_calls  total_num_sms  distinct_call_nums  \
0                4370               71              5                  17
1                 318                5              7                   4
2                2453               60            359                  24
3                4198               66              1                  35
4                2393               58              2                  33
...               ...              ...            ...                 ...
3145             6697              147             92                  44
3146             9237              177             80                  42
3147             3157               51             38                  21
3148             4695               46            222                  12
3149             1792               25              7                   9

      age_group  tariff_plan  status  age  customer_value  Churn
0             3            1       1   30         197.640      0
1             2            1       2   25          46.035      0
2             3            1       1   30        1536.520      0
3             1            1       1   15         240.020      0
4             1            1       1   15         145.805      0
...         ...          ...     ...  ...             ...    ...
3145          2            2       1   25         721.980      0
3146          5            1       1   55         261.210      0
3147          3            1       1   30         280.320      0
3148          3            1       1   30        1077.640      0
3149          3            1       1   30         100.680      1

[3018 rows x 15 columns]
```

In [11]:
```
df.describe()
```

Out[11]:

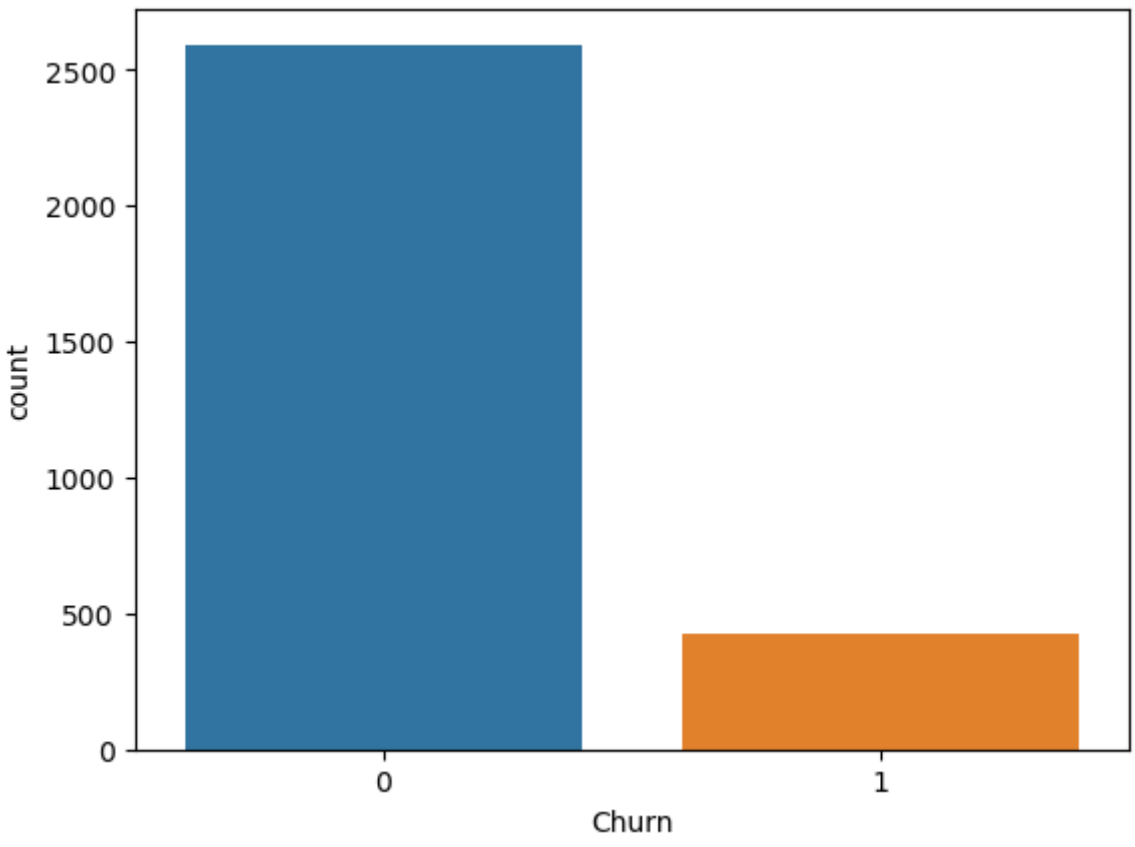
	customer_id	call_failure	complains	subs_len	charge_amount	total_sec_calls	total_n
count	3018.000000	3018.000000	3018.000000	3018.000000	3018.000000	3018.000000	301
mean	1587.632207	7.961564	0.074553	32.554341	0.983433	4668.074221	7
std	916.463049	7.239854	0.262712	8.686214	1.541082	4180.910144	5
min	1.000000	0.000000	0.000000	3.000000	0.000000	0.000000	
25%	790.250000	2.000000	0.000000	29.000000	0.000000	1610.750000	3
50%	1601.500000	7.000000	0.000000	35.000000	0.000000	3157.000000	5
75%	2389.750000	12.000000	0.000000	38.000000	2.000000	6581.500000	9
max	3150.000000	36.000000	1.000000	47.000000	10.000000	17090.000000	25

EDA-Exploratory Data Analysis

In [12]:
```
import seaborn as sns
```

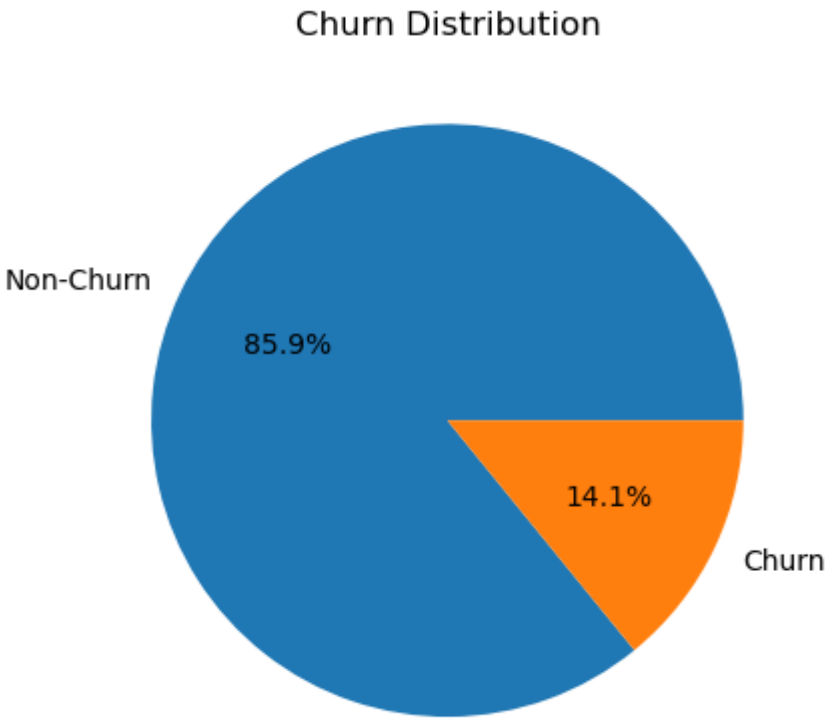
In [13]:
```
#see target class is imbalanced
sns.countplot(x="Churn", data=df)
```

Out[13]:
```
<Axes: xlabel='Churn', ylabel='count'>
```



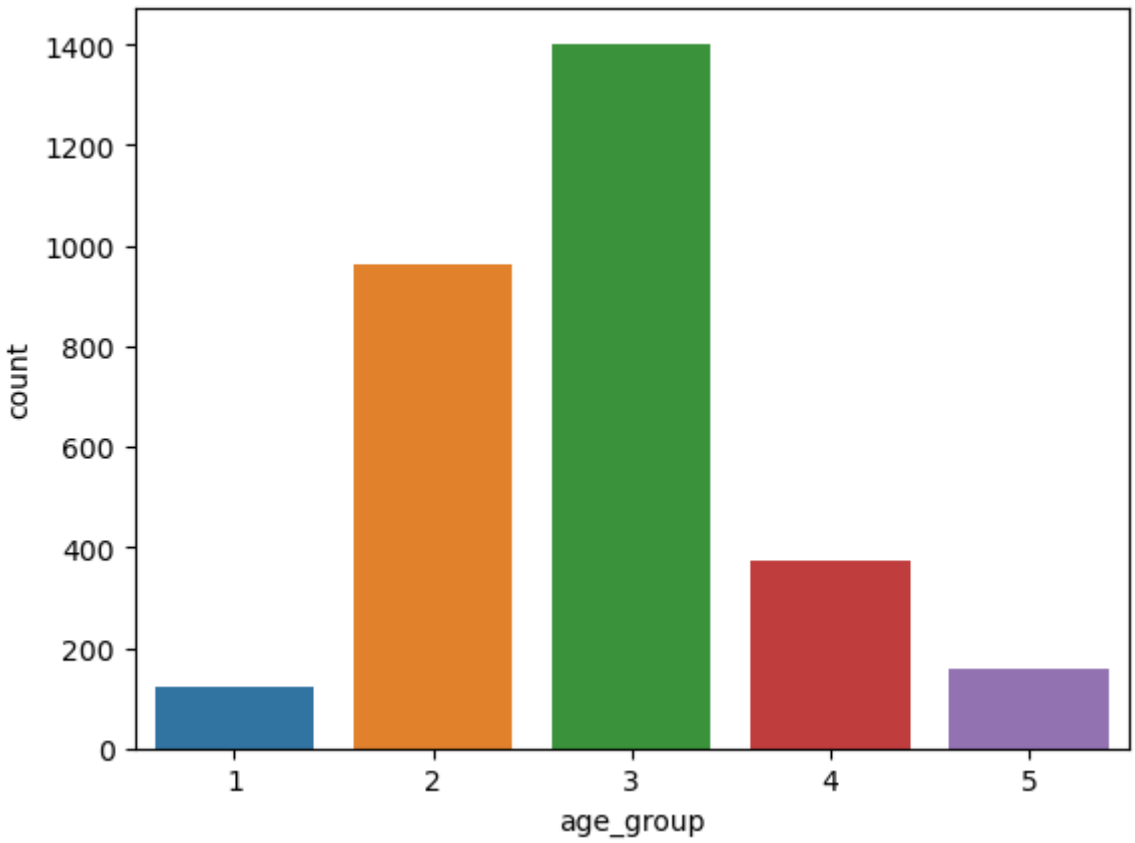
In [14]:
```
import matplotlib.pyplot as plt
```

In [15]:
```
churned = df['Churn'].value_counts()
plt.pie(churned, labels=['Non-Churn', 'Churn'], autopct='%1.1f%%')
plt.title('Churn Distribution')
plt.show()
```



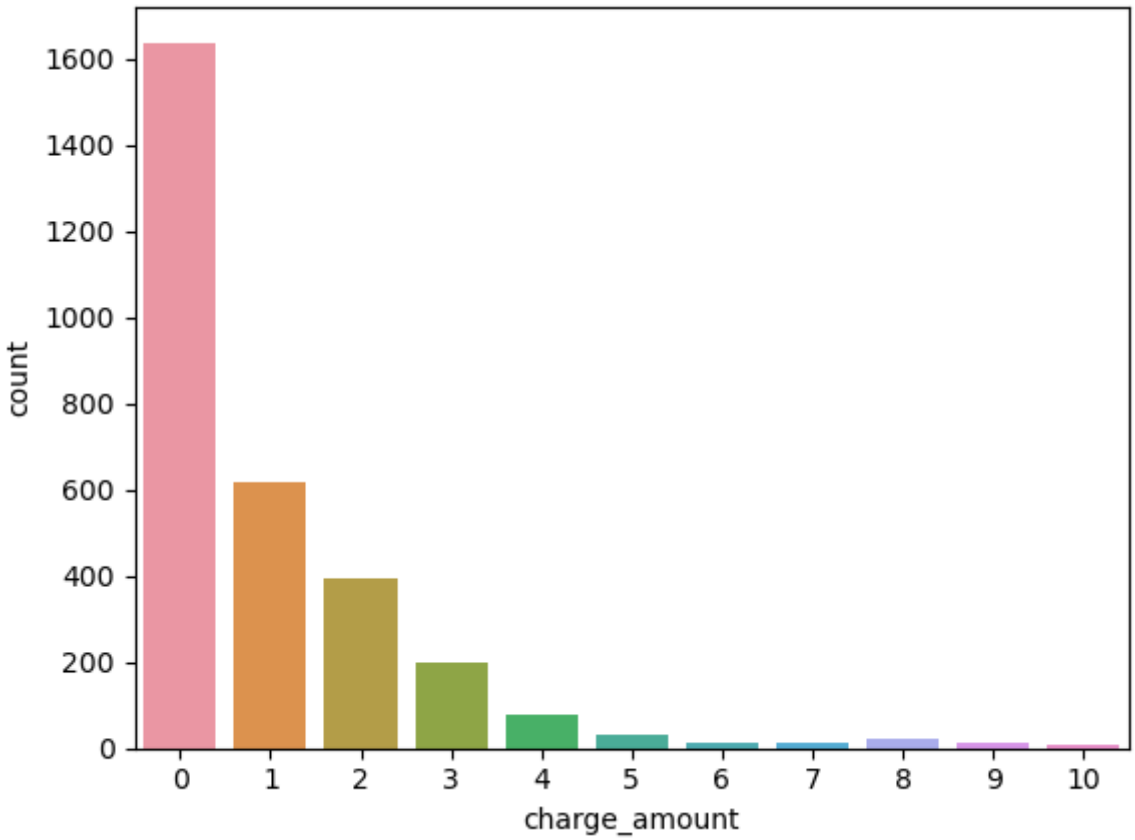
In [16]:
```
sns.countplot(x="age_group",data=df)
```

Out[16]:
```
<Axes: xlabel='age_group', ylabel='count'>
```



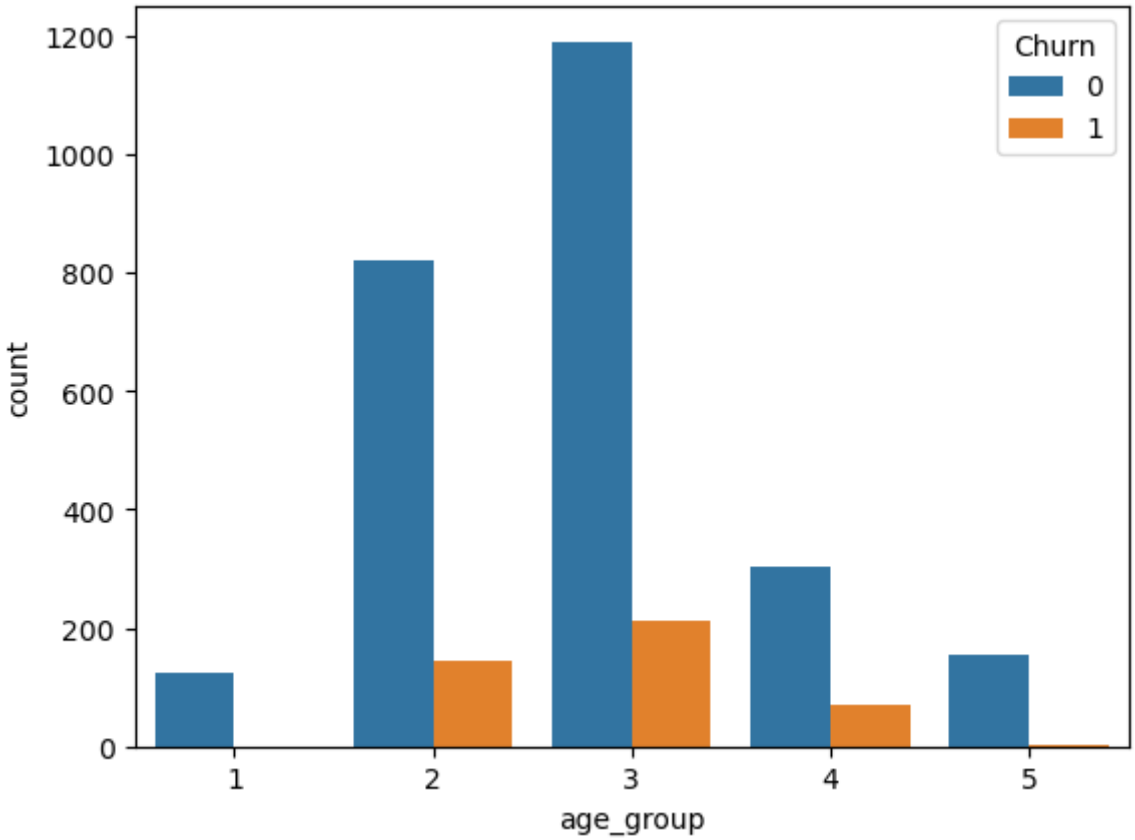
In [17]:
```
sns.countplot(x="charge_amount", data=df)
```

Out[17]:
```
<Axes: xlabel='charge_amount', ylabel='count'>
```



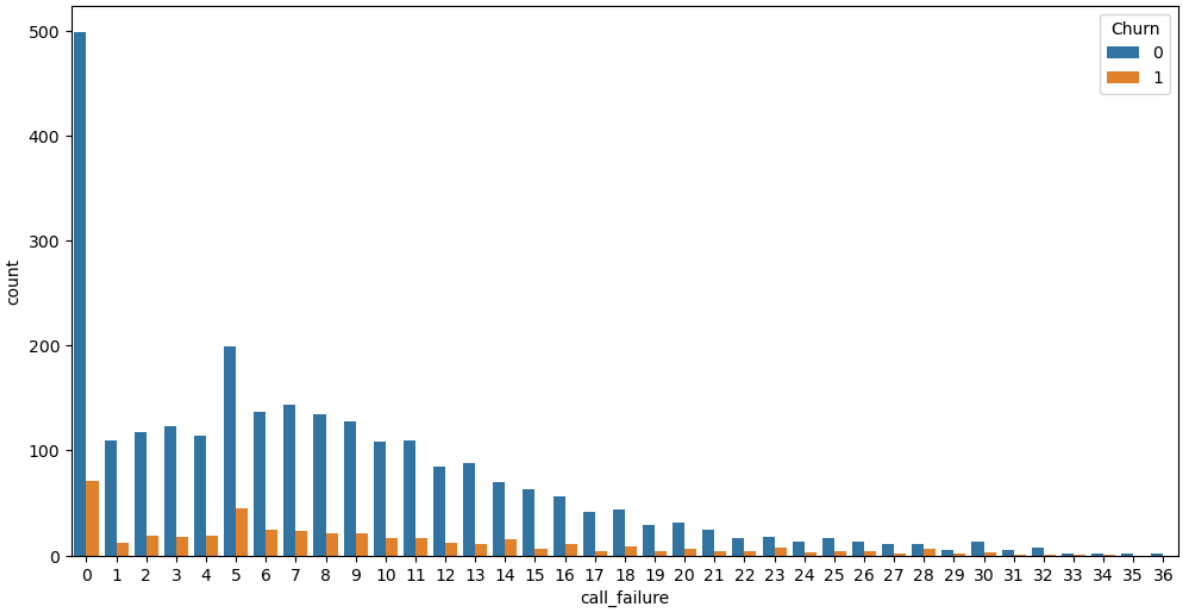
In [18]:
```
sns.countplot(x="age_group", data=df, hue="Churn")
```

Out[18]:
```
<Axes: xlabel='age_group', ylabel='count'>
```



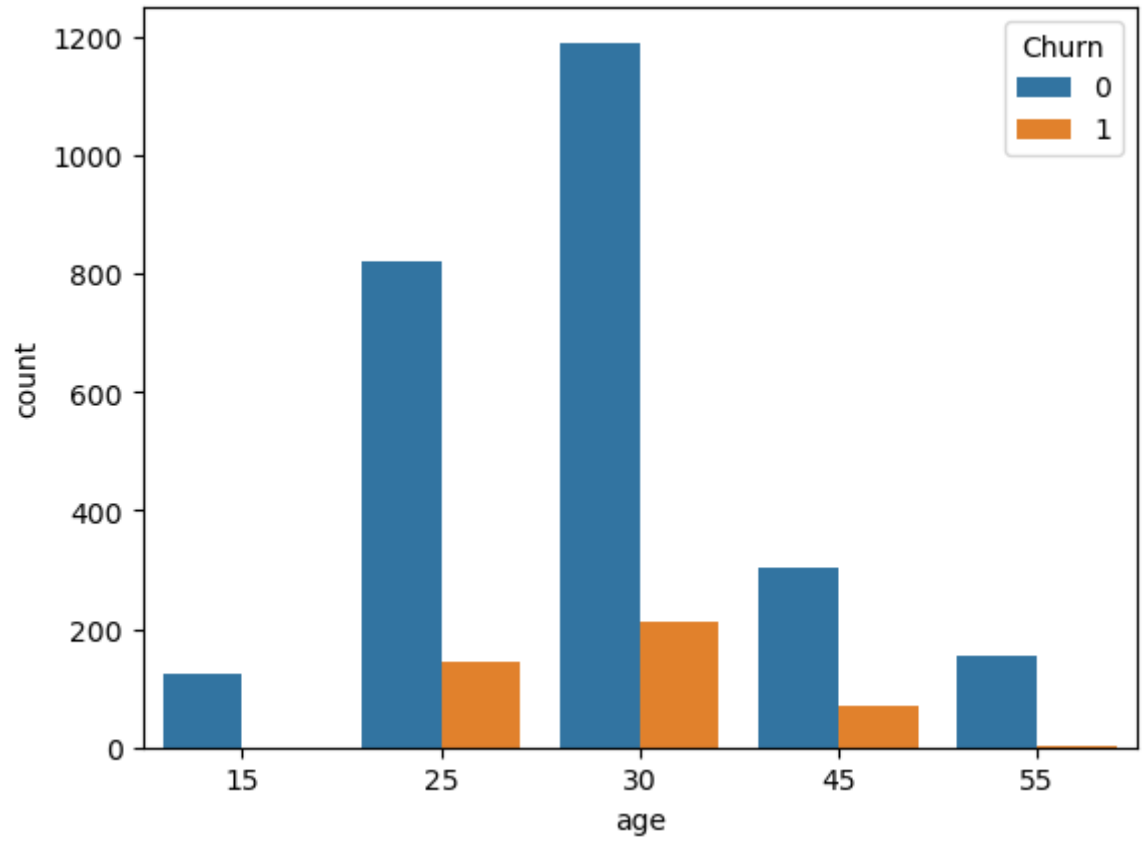
In [19]:
```
plt.figure(figsize=(12, 6))
sns.countplot(x='call_failure', data=df, hue="Churn")
```

Out[19]:
```
<Axes: xlabel='call_failure', ylabel='count'>
```



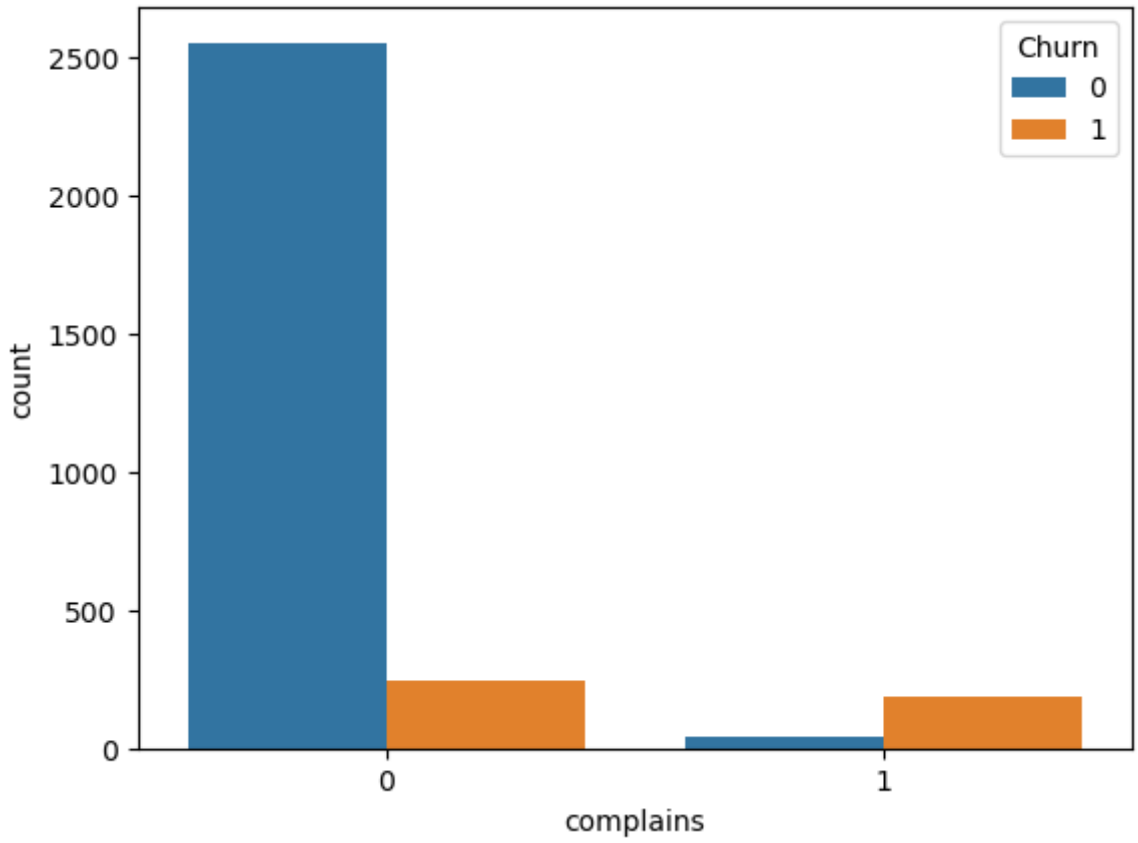
In [20]:
```
sns.countplot(x='age', data=df, hue="Churn")
```

Out[20]:
```
<Axes: xlabel='age', ylabel='count'>
```



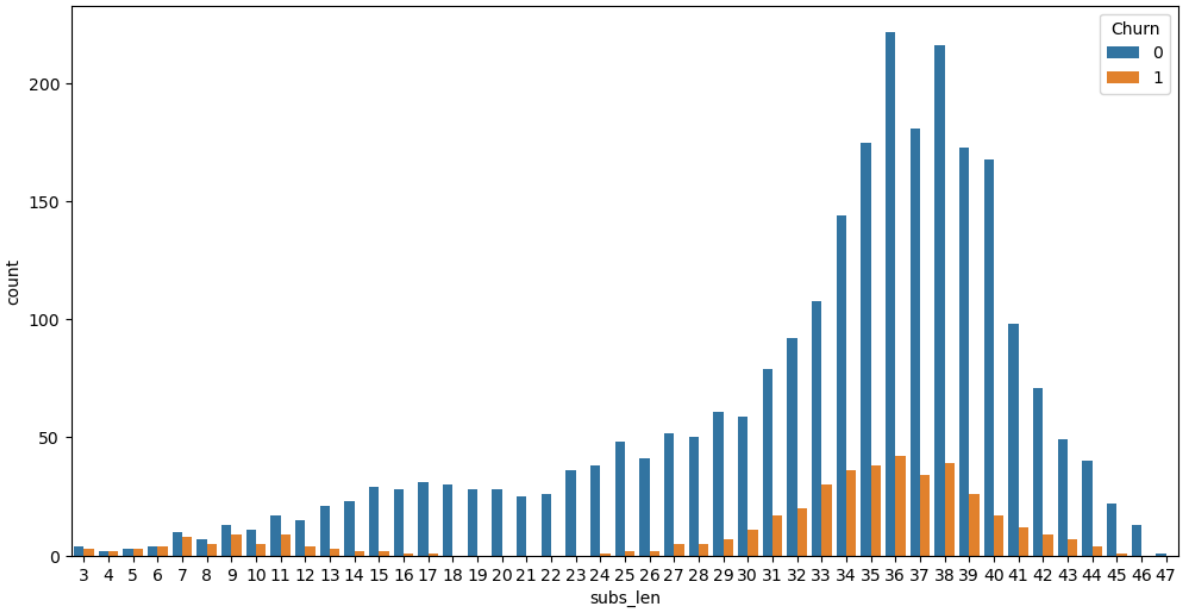
In [21]:
```
sns.countplot(x='complains', data=df, hue="Churn")
```

Out[21]:
```
<Axes: xlabel='complains', ylabel='count'>
```



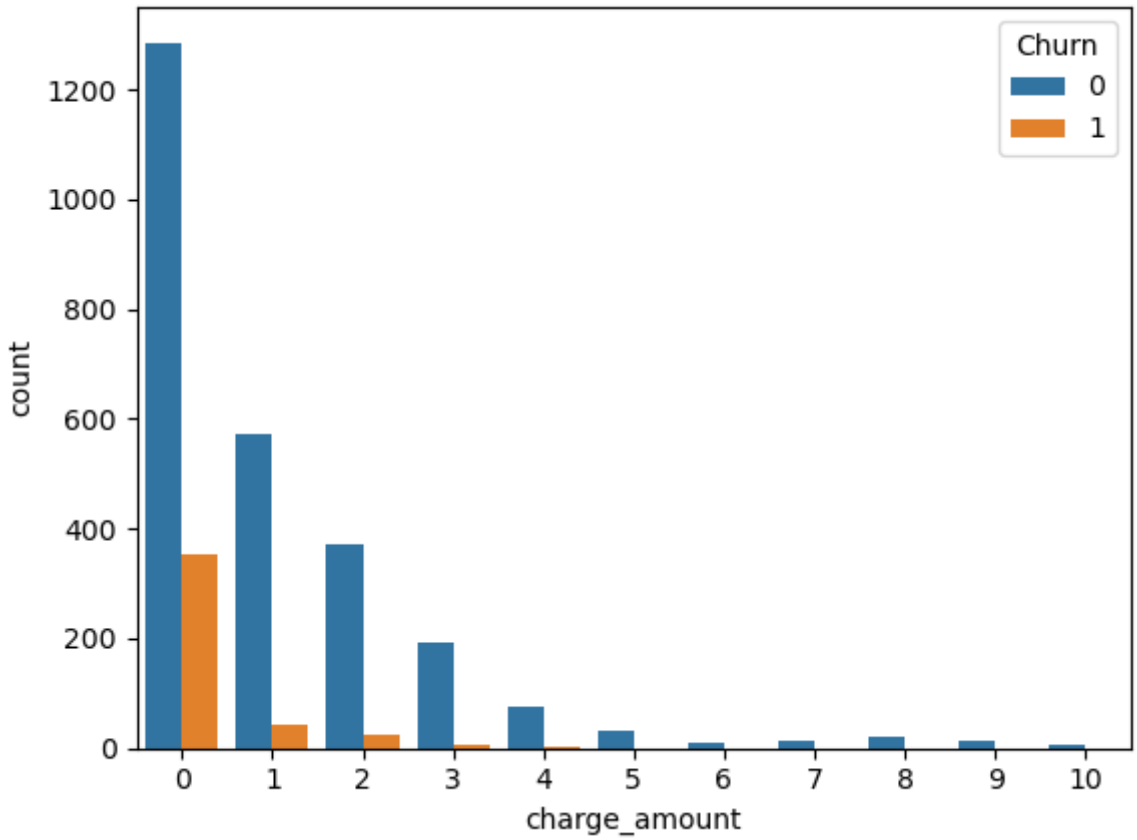
In [22]:
```
plt.figure(figsize=(12, 6))
sns.countplot(x='subs_len', data=df, hue="Churn")
```

Out[22]:
```
<Axes: xlabel='subs_len', ylabel='count'>
```



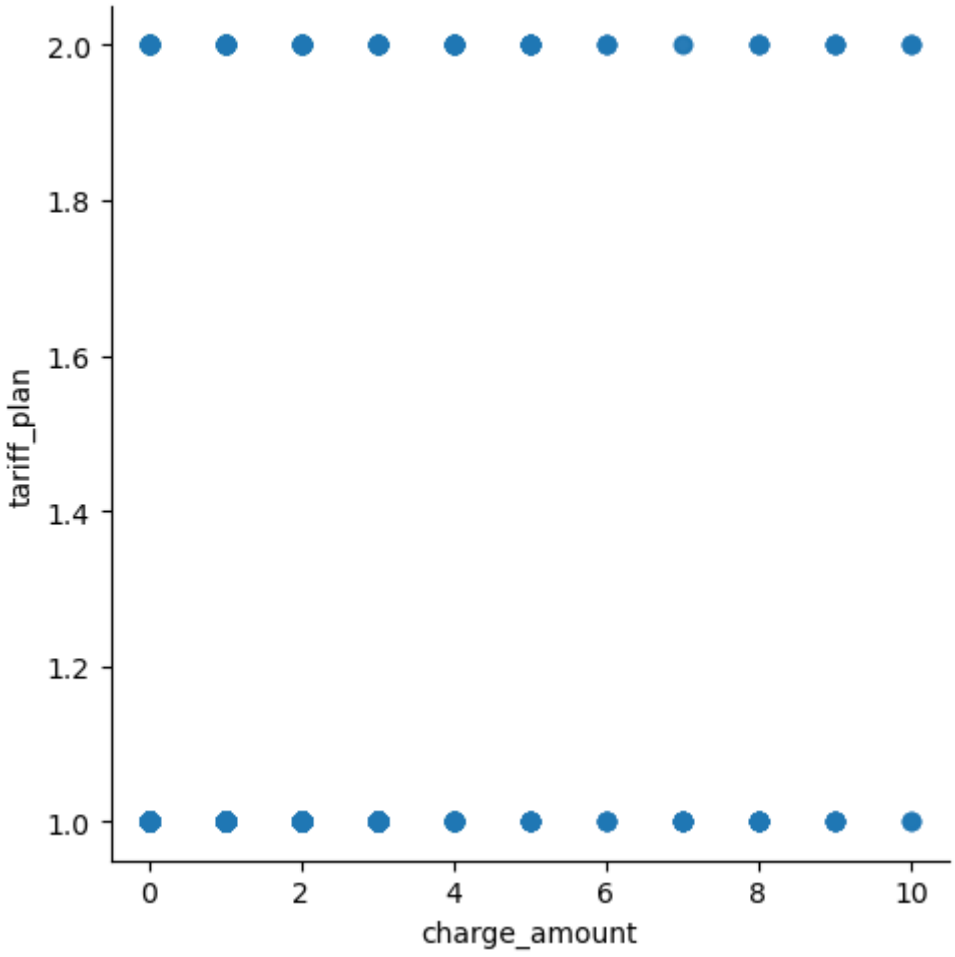
In [23]:
```
sns.countplot(x='charge_amount', data=df, hue="Churn")
```

Out[23]:
```
<Axes: xlabel='charge_amount', ylabel='count'>
```

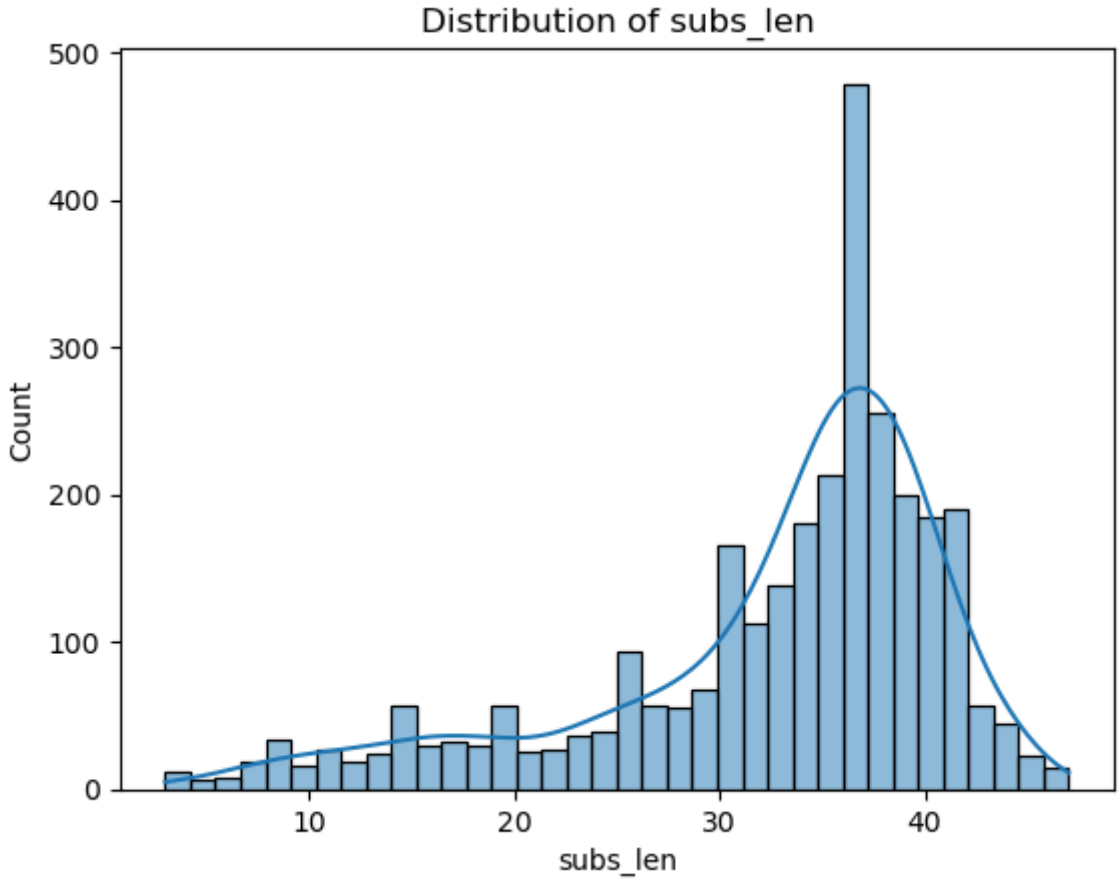


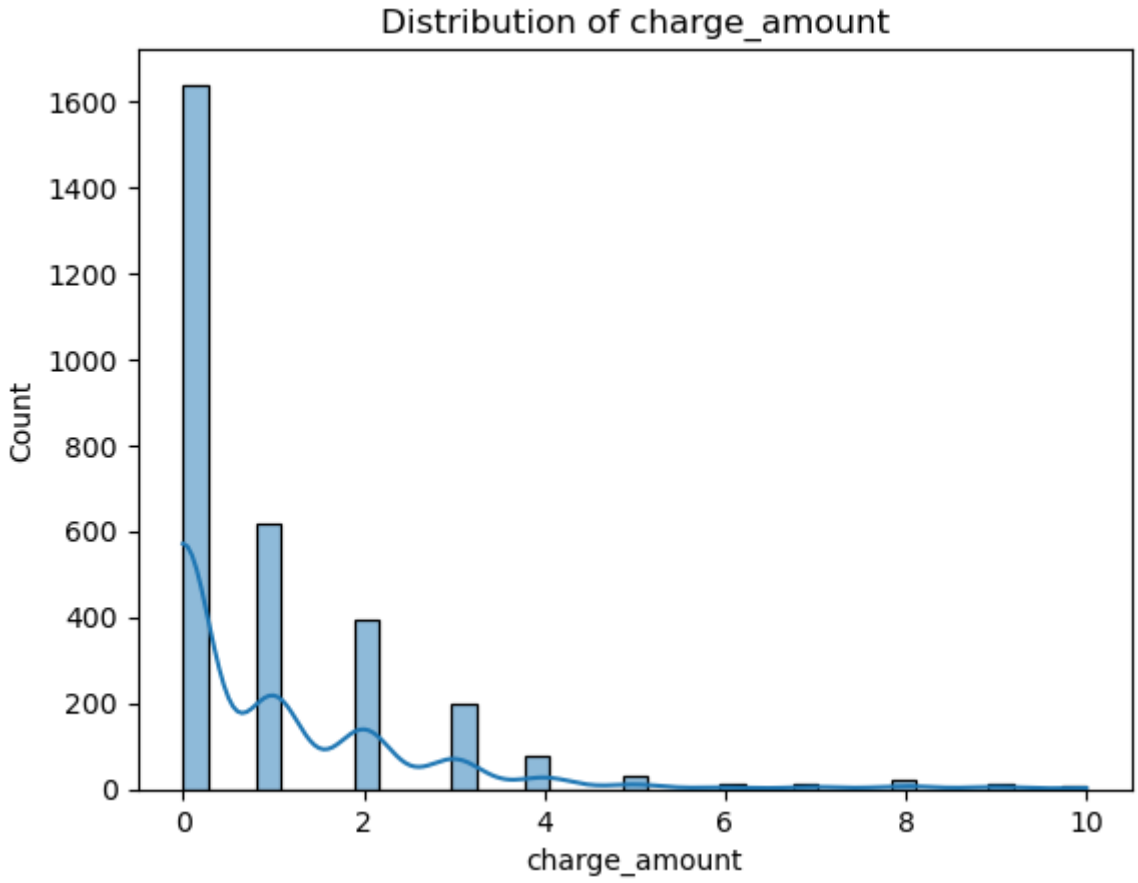
In [24]:
```
sns.lmplot(data=df, x='charge_amount', y='tariff_plan', fit_reg=False)
```

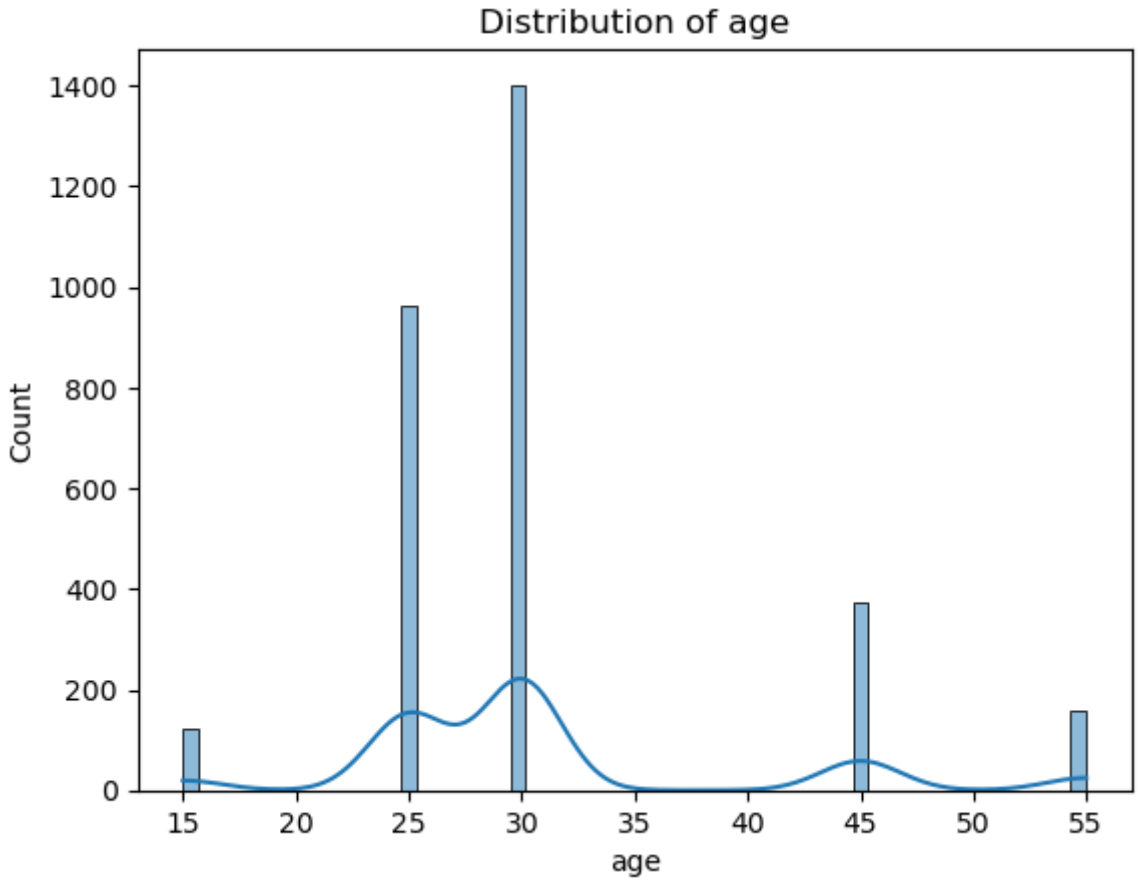
Out[24]:
```
<seaborn.axisgrid.FacetGrid at 0x1f941df1f30>
```

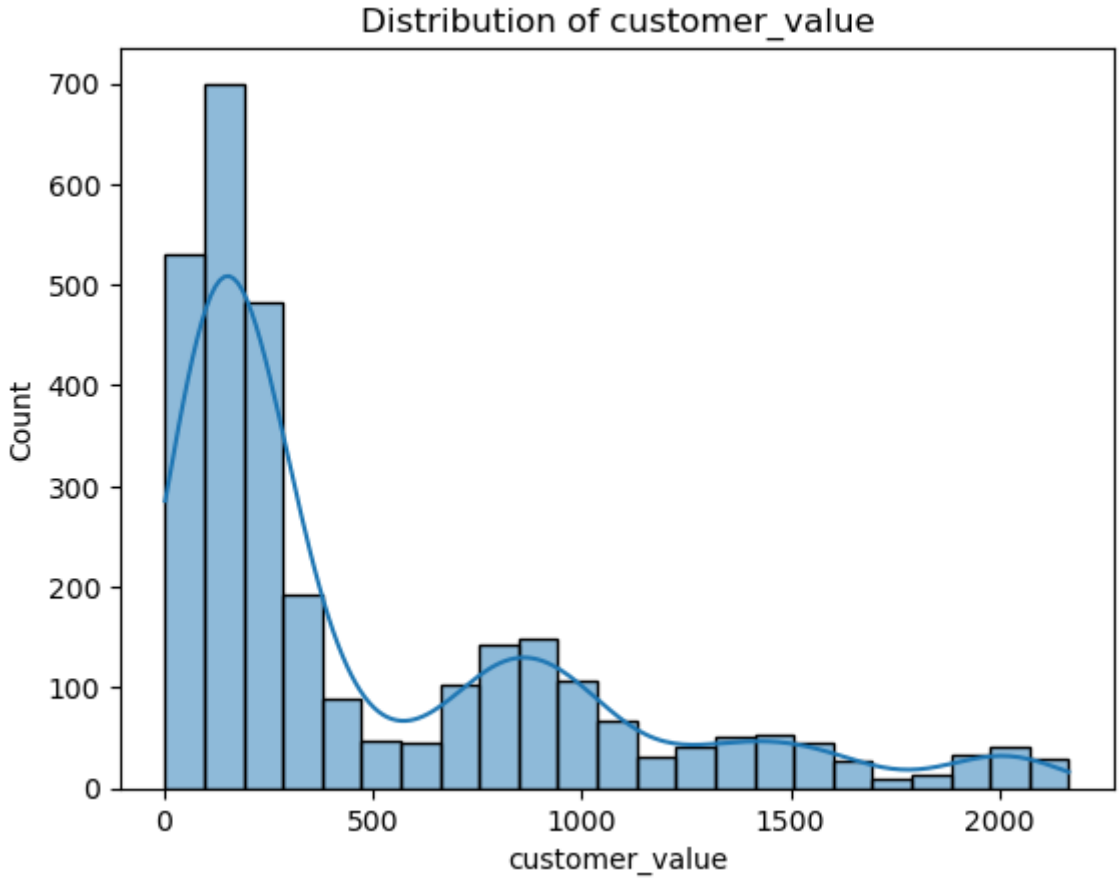


In [25]:
```
numeric_features = ['subs_len', 'charge_amount', 'age', 'customer_value']
for feature in numeric_features:
    sns.histplot(df[feature], kde=True)
    plt.title(f'Distribution of {feature}')
    plt.show()
```

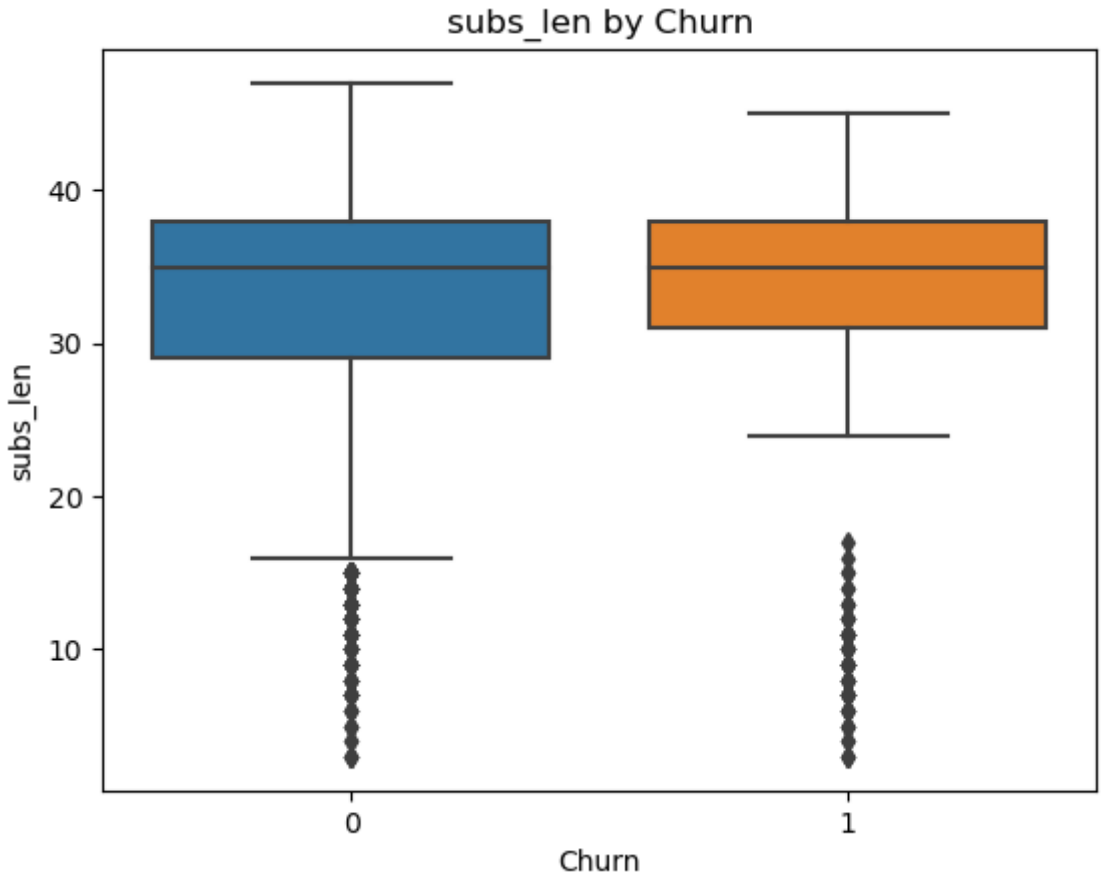


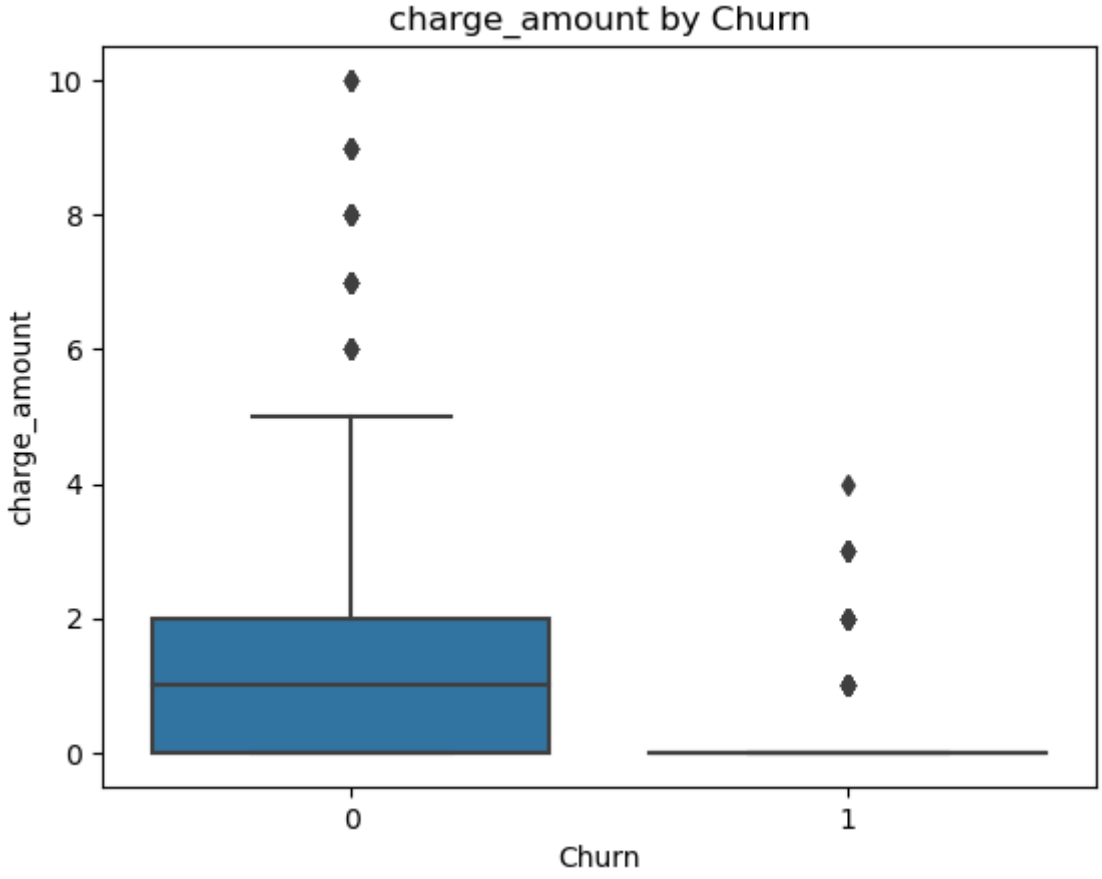


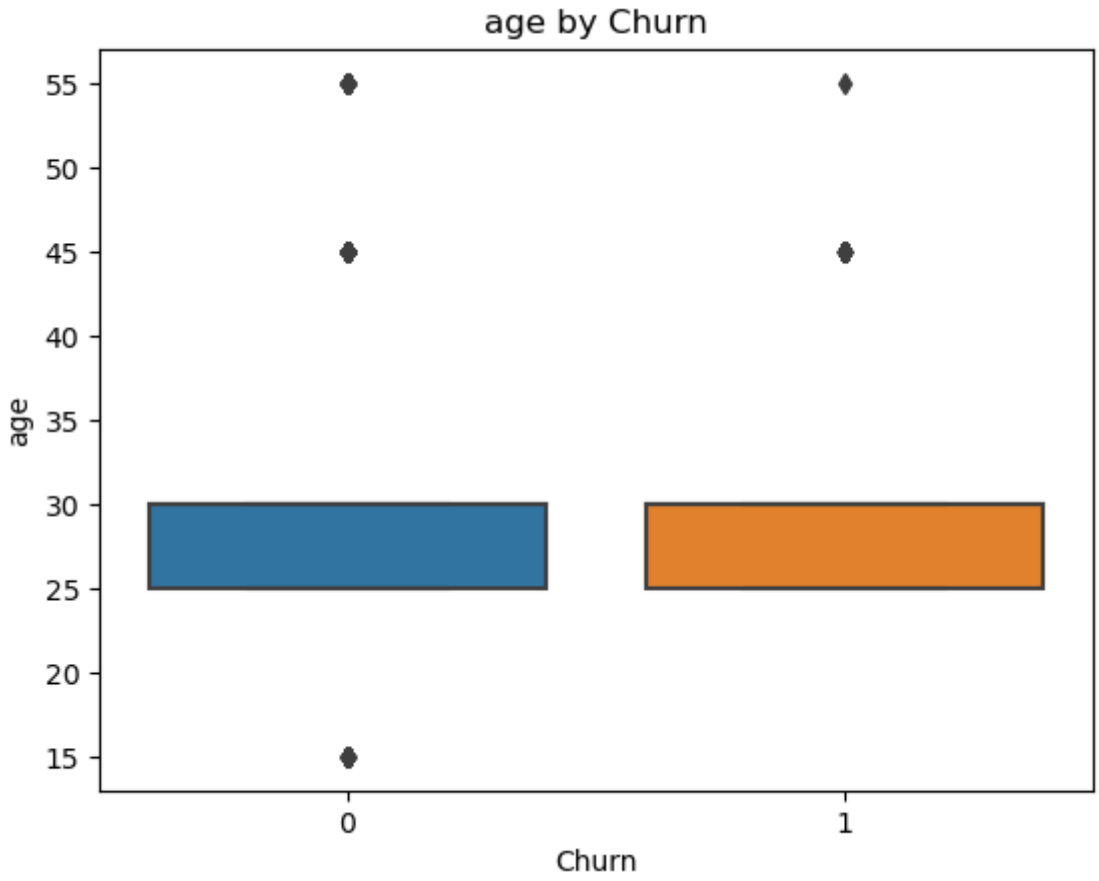


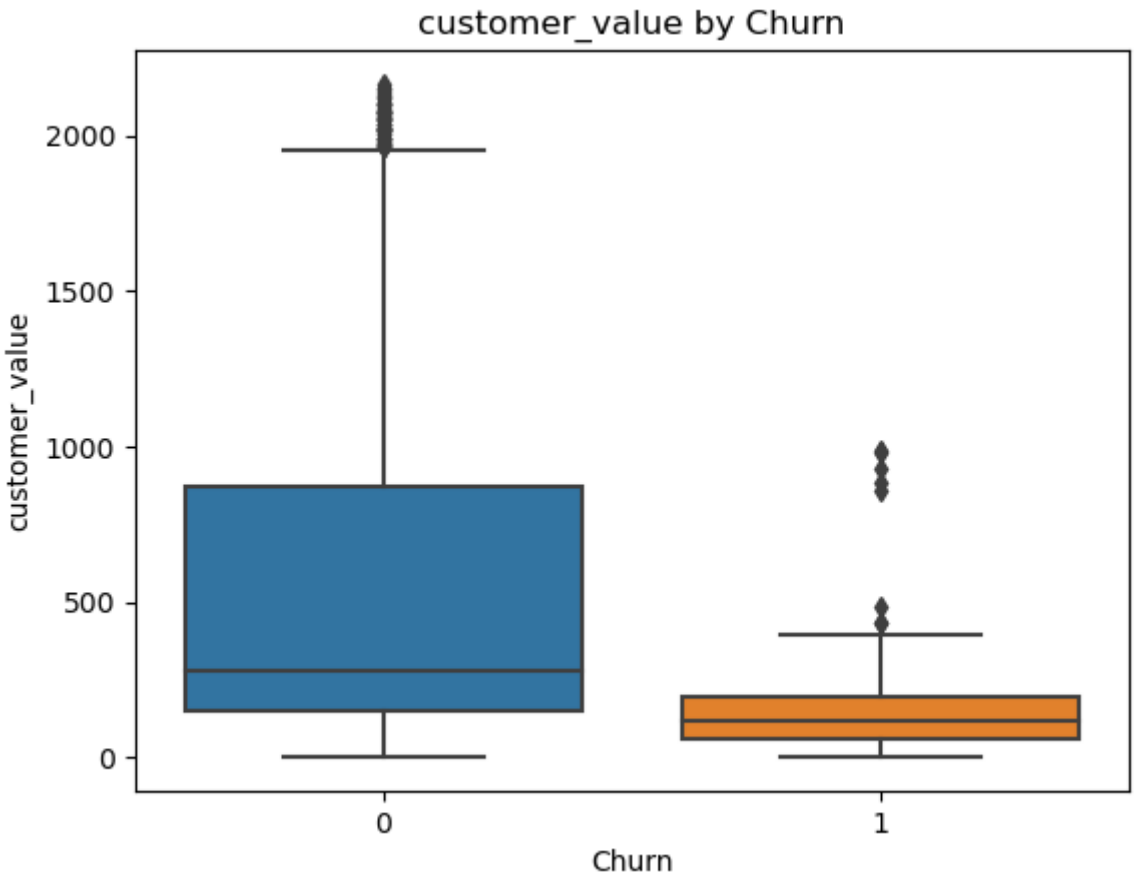


In [26]:
```
for feature in numeric_features:
    sns.boxplot(data=df, y=feature, x='Churn')
    plt.title(f'{feature} by Churn')
    plt.show()
```

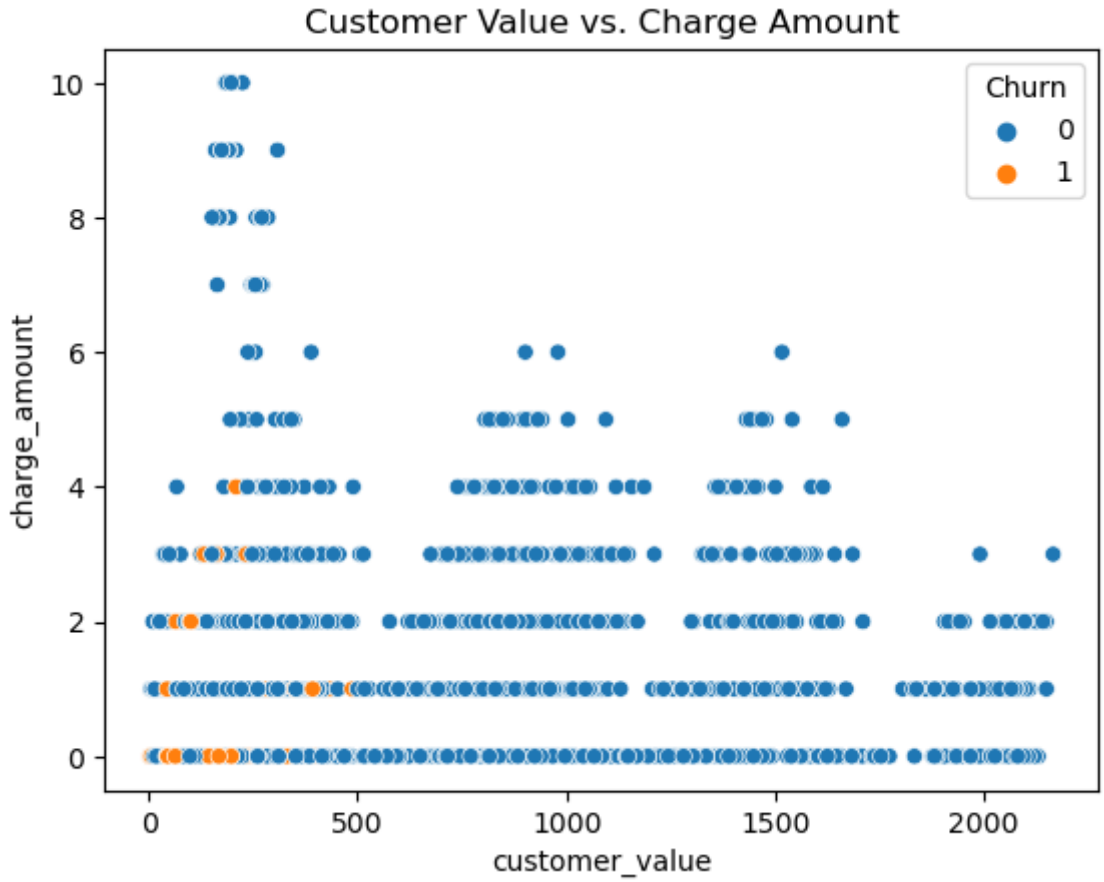

subs_len by Churn
subs_len
40
30
20
10
0
1
Churn


charge_amount by Churn
charge_amount
10
8
6
4
2
0
0
1
Churn





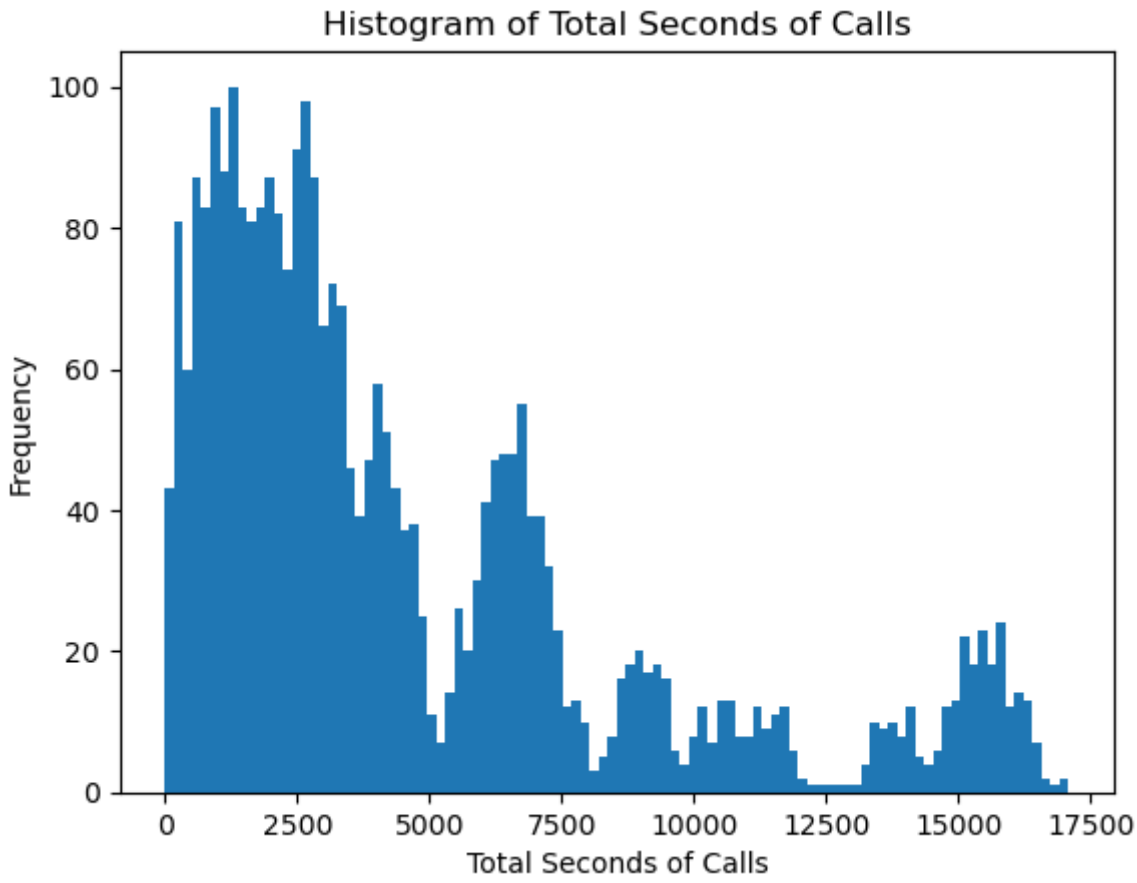
In [27]:
```
sns.scatterplot(data=df, x='customer_value', y='charge_amount', hue='Churn')
plt.title('Customer Value vs. Charge Amount')
plt.show()
```



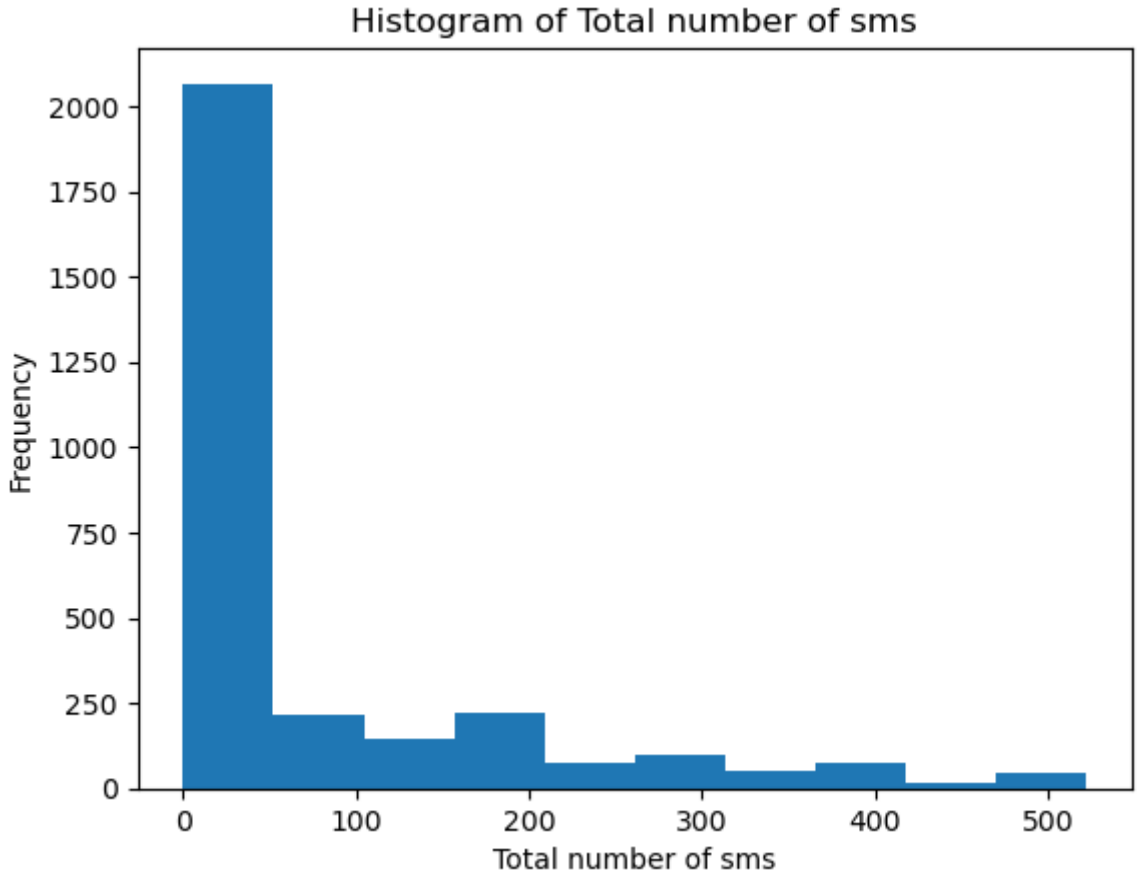
In [28]:
```
df.columns
```

Out[28]:
```
Index(['customer_id', 'call_failure', 'complains', 'subs_len', 'charge_amount',
       'total_sec_calls', 'total_num_calls', 'total_num_sms',
       'distinct_call_nums', 'age_group', 'tariff_plan', 'status', 'age',
       'customer_value', 'Churn'],
      dtype='object')
```

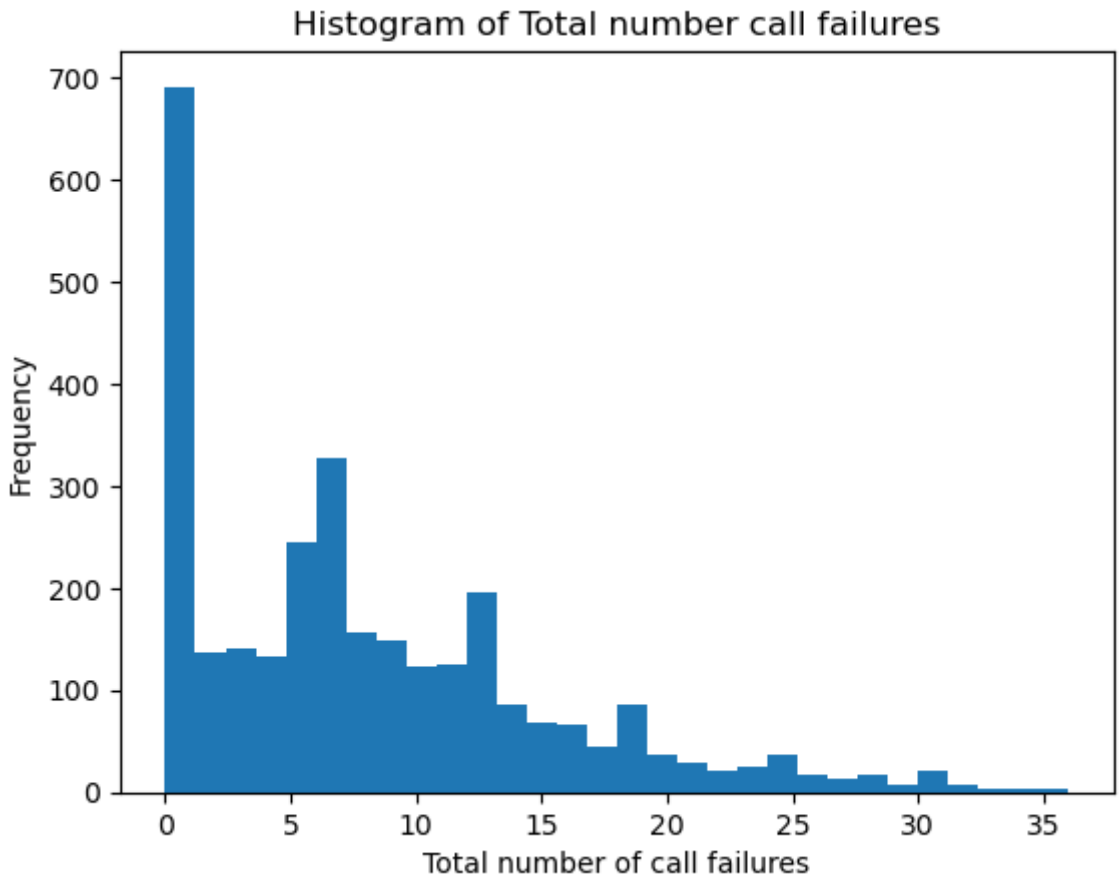
In [29]:
```
plt.hist(df['total_sec_calls'], bins=100)
plt.xlabel('Total Seconds of Calls')
plt.ylabel('Frequency')
plt.title('Histogram of Total Seconds of Calls')
plt.show()
```



In [30]:
```
plt.hist(df['total_num_sms'], bins=10)
plt.xlabel('Total number of sms')
plt.ylabel('Frequency')
plt.title('Histogram of Total number of sms')
plt.show()
```



In [31]:
```
plt.hist(df['call_failure'], bins=30)
plt.xlabel('Total number of call failures')
plt.ylabel('Frequency')
plt.title('Histogram of Total number call failures')
plt.show()
```

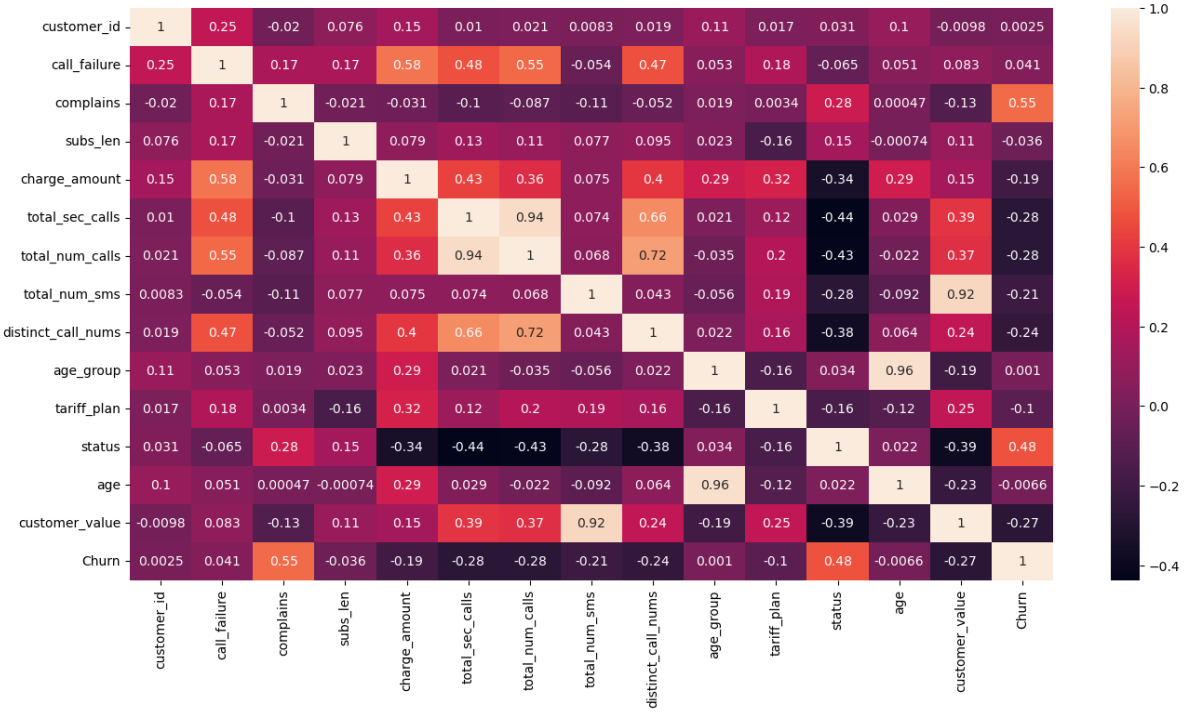


In [32]:
```
df.corr()
```

Out[32]:

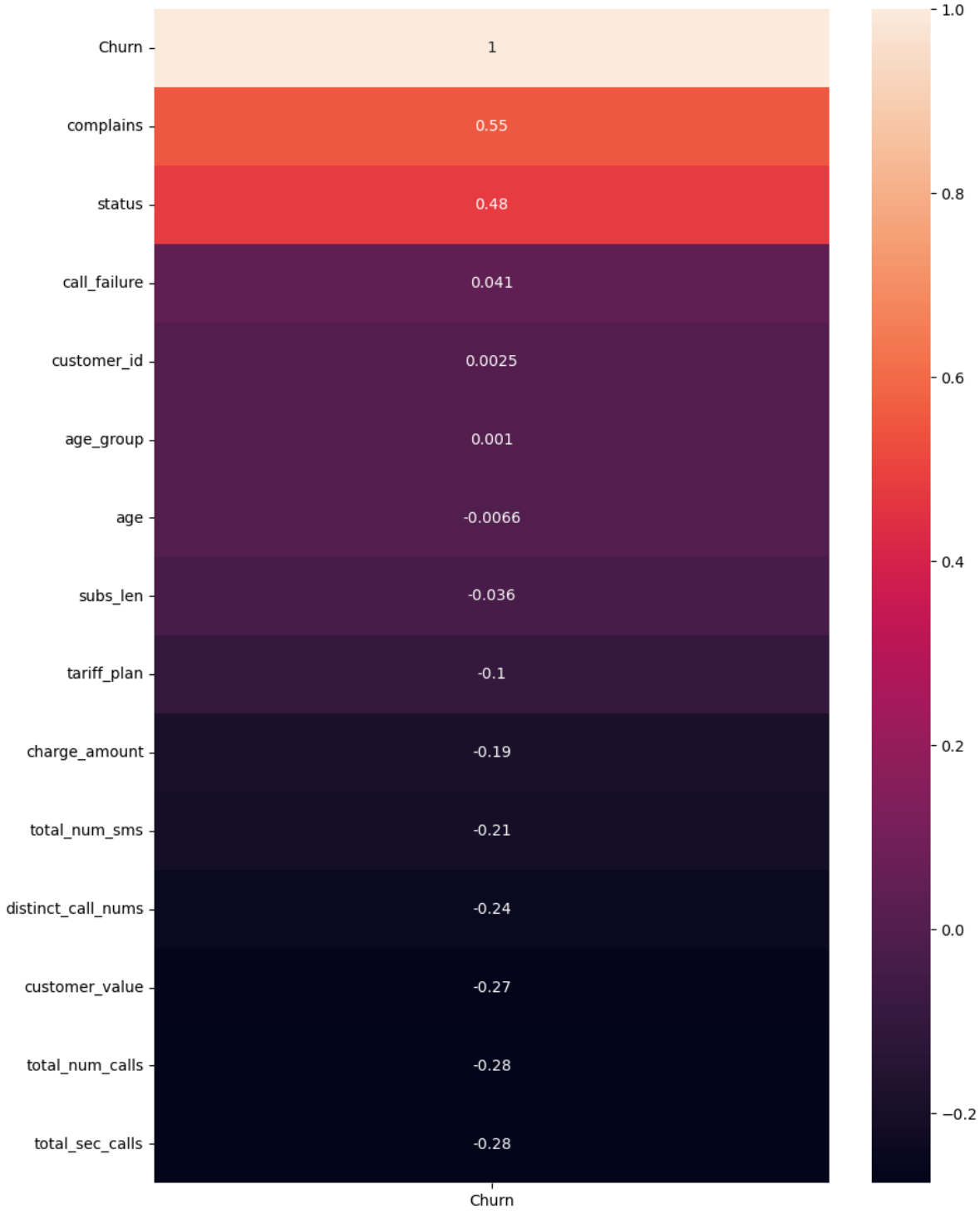
	customer_id	call_failure	complains	subs_len	charge_amount	total_sec_calls	t
customer_id	1.000000	0.246487	-0.019955	0.076303	0.149866	0.010287	
call_failure	0.246487	1.000000	0.170198	0.173870	0.579867	0.475958	
complains	-0.019955	0.170198	1.000000	-0.021021	-0.030515	-0.102991	
subs_len	0.076303	0.173870	-0.021021	1.000000	0.079005	0.127305	
charge_amount	0.149866	0.579867	-0.030515	0.079005	1.000000	0.432694	
total_sec_calls	0.010287	0.475958	-0.102991	0.127305	0.432694	1.000000	
total_num_calls	0.020738	0.548582	-0.087493	0.109212	0.361484	0.943775	
total_num_sms	0.008271	-0.054049	-0.111494	0.076720	0.075462	0.074278	
distinct_call_nums	0.019451	0.472032	-0.051947	0.094728	0.398535	0.656024	
age_group	0.114653	0.052787	0.018619	0.023380	0.289156	0.020760	
tariff_plan	0.017295	0.183764	0.003393	-0.161742	0.319726	0.123383	
status	0.030745	-0.064590	0.281348	0.148362	-0.344536	-0.438485	
age	0.103030	0.051357	0.000472	-0.000739	0.293089	0.029375	
customer_value	-0.009794	0.082878	-0.132900	0.111187	0.149067	0.389371	
Churn	0.002511	0.040673	0.551570	-0.035629	-0.188340	-0.276173	

In [33]:
```
plt.figure(figsize=(16,8))
sns.heatmap(df.corr(), annot = True);
```



In [34]:
```
# Creating a correlation matrix plot in order
fig, ax = plt.subplots(figsize=(10,14))
churn_data_corr = df.corr()[['Churn']].sort_values(
 by='Churn', ascending=False)
sns.heatmap(churn_data_corr, annot=True, ax=ax)
```

Out[34]:
```
<Axes: >
```



Modelling

In [35]:
```
current_date = pd.to_datetime('today')
recency = df.groupby('customer_value')['subs_len'].max()  # Use max to get the toto
frequency = df.groupby('customer_value')['subs_len'].count()
monetary = df.groupby('customer_value')['charge_amount'].sum()
```

In [36]:
```
rfm_df = pd.DataFrame({
    'Recency': recency,
    'Frequency': frequency,
    'Monetary': monetary
})
```

In [37]:
```
rfm_df.head()
```

Out[37]:

customer_value	Recency	Frequency	Monetary
2.34	31	1	0
4.00	39	1	0
4.41	36	1	1
4.50	40	2	0
5.13	38	1	0

In [38]:

```
rfm_df['R_rank'] = rfm_df['Recency'].rank(ascending=False)
rfm_df['F_rank'] = rfm_df['Frequency'].rank(ascending=True)
rfm_df['M_rank'] = rfm_df['Monetary'].rank(ascending=True)

# normalizing the rank of the customers
rfm_df['R_rank_norm'] = (rfm_df['R_rank']/rfm_df['R_rank'].max())*100
rfm_df['F_rank_norm'] = (rfm_df['F_rank']/rfm_df['F_rank'].max())*100
rfm_df['M_rank_norm'] = (rfm_df['F_rank']/rfm_df['M_rank'].max())*100

rfm_df.drop(columns=['R_rank', 'F_rank', 'M_rank'], inplace=True)

rfm_df.head()
```

Out[38]:

customer_value	Recency	Frequency	Monetary	R_rank_norm	F_rank_norm	M_rank_norm
2.34	31	1	0	70.584906	46.155296	46.155296
4.00	39	1	0	21.660377	46.155296	46.155296
4.41	36	1	1	43.962264	46.155296	46.155296
4.50	40	2	0	15.283019	93.592160	93.592160
5.13	38	1	0	28.943396	46.155296	46.155296

In [39]:

```
rfm_df['RFM_Score'] = 0.15*rfm_df['R_rank_norm']+0.28 * \
    rfm_df['F_rank_norm']+0.57*rfm_df['M_rank_norm']
rfm_df['RFM_Score'] *= 0.05
rfm_df = rfm_df.round(2)
rfm_df.head(7)
```

Out[39]:

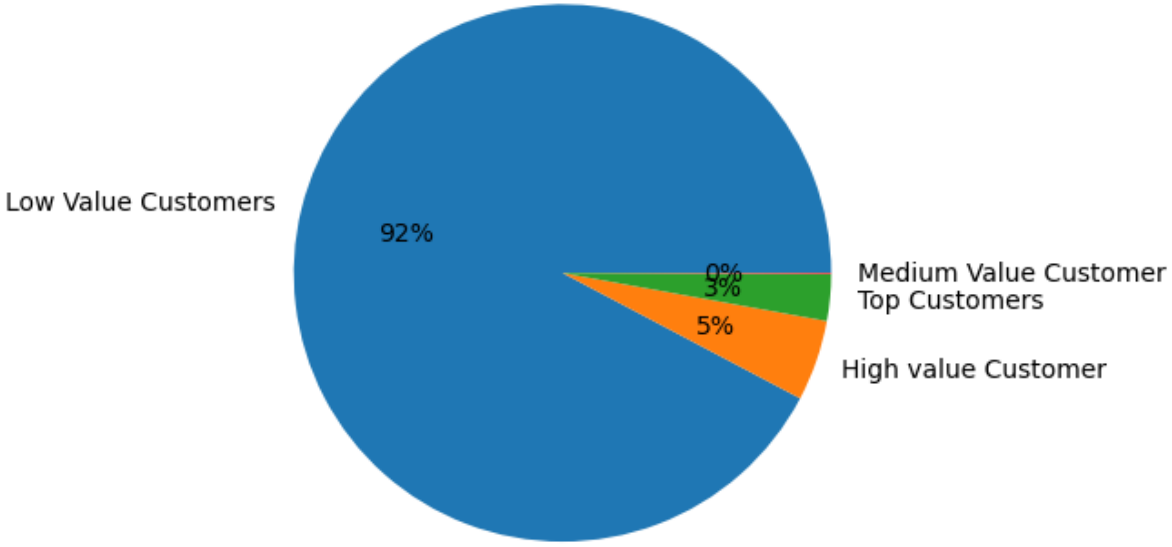
customer_value	Recency	Frequency	Monetary	R_rank_norm	F_rank_norm	M_rank_norm	RFM_S
2.340	31	1	0	70.58	46.16	46.16	
4.000	39	1	0	21.66	46.16	46.16	
4.410	36	1	1	43.96	46.16	46.16	
4.500	40	2	0	15.28	93.59	93.59	
5.130	38	1	0	28.94	46.16	46.16	
5.175	27	1	0	79.87	46.16	46.16	
5.400	36	3	0	43.96	97.30	97.30	

In [40]:
```
rfm_df["Customer_segment"] = np.where(rfm_df['RFM_Score'] >
                                      4.5, "Top Customers",
                                      (np.where(
                                        rfm_df['RFM_Score'] > 4,
                                        "High value Customer",
                                        (np.where(
    rfm_df['RFM_Score'] > 3,
                               "Medium Value Customer",
                               np.where(rfm_df['RFM_Score'] > 1.6,
                              'Low Value Customers', 'Lost Customers'))))))
rfm_df[[ 'RFM_Score', 'Customer_segment']].head(20)
```

Out[40]:

customer_value	RFM_Score	Customer_segment
2.340	2.49	Low Value Customers
4.000	2.12	Low Value Customers
4.410	2.29	Low Value Customers
4.500	4.09	High value Customer
5.130	2.18	Low Value Customers
5.175	2.56	Low Value Customers
5.400	4.47	High value Customer
5.625	2.23	Low Value Customers
5.940	2.12	Low Value Customers
7.500	2.62	Low Value Customers
7.560	2.29	Low Value Customers
8.000	2.39	Low Value Customers
8.055	2.12	Low Value Customers
8.575	2.49	Low Value Customers
9.000	4.36	High value Customer
9.930	2.18	Low Value Customers
9.990	2.29	Low Value Customers
10.000	2.62	Low Value Customers
10.035	2.58	Low Value Customers
10.125	2.61	Low Value Customers

In [41]:
```
import matplotlib.pyplot as plt
plt.pie(rfm_df.Customer_segment.value_counts(),
        labels=rfm_df.Customer_segment.value_counts().index,
        autopct='%.0f%%')
plt.show()
```



In [42]:
```
#USING VARIUOS ALGORITHMS
```

In [43]:
```
#import sklearn methods
from sklearn.metrics import accuracy_score, roc_curve, confusion_matrix, classifica
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB, ComplementNB, Categorica
from sklearn.svm import LinearSVC,SVC
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import learning_curve
import sys
import os
```

In [44]:
```
# split df to X and Y
X = df[['call_failure', 'complains', 'age_group', 'status','subs_len','tariff_plan'
y = df['Churn']

# split data into 80-20 for training set / test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, stratify

# cross-validation with 5 splits
cv = StratifiedShuffleSplit(n_splits=5, random_state = 88)

#hold-out
hold_out=StratifiedShuffleSplit(n_splits=1, test_size=0.25, random_state = 88)
```

In [45]:
```
#normalization(make all values bet. 0-1)

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_normalized_arr=scaler.transform(X_train)
X_train_normalized_df=pd.DataFrame(X_train_normalized_arr, columns=list(X.columns))

X_test_normalized_arr=scaler.transform(X_test)
X_test_normalized_df=pd.DataFrame(X_test_normalized_arr, columns=list(X.columns))
```

In [46]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
```

In [47]:
```
g=GaussianNB()
b=BernoulliNB()
m=MultinomialNB()
```

In [48]:
```
g.fit(X_train_normalized_df,y_train)
b.fit(X_train_normalized_df,y_train)
m.fit(X_train_normalized_df,y_train)
```

Out[48]:
```
▾ MultinomialNB
MultinomialNB()
```

In [49]:
```
predg=g.predict(X_test_normalized_df)
predb=b.predict(X_test_normalized_df)
predm=m.predict(X_test_normalized_df)
```

In [50]:
```
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score, c
```

In [51]:
```
accuracy_score(predg,y_test)
```

Out[51]:
```
0.7483443708609272
```

In [52]:
```
accuracy_score(predb,y_test)
```

Out[52]:
```
0.9056291390728477
```

In [53]:
```
accuracy_score(predm,y_test)
```

Out[53]:
```
0.9072847682119205
```

In [54]:
```
# Assuming 'Churn' is the actual churn column in your DataFrame
actual_churn = y_test

# Assuming 'predg' is the predicted churn values using Gaussian Naive Bayes
predicted_churn_gaussian = predg

# Assuming 'predb' is the predicted churn values using Bernoulli Naive Bayes
predicted_churn_bernoulli = predb
predicted_churn_multinomial = predm
# Filter the DataFrame based on your conditions
mg = X_test[(predicted_churn_gaussian == 1) & (actual_churn == 0)]
mb = X_test[(predicted_churn_bernoulli == 1) & (actual_churn == 0)]
mt=X_test[(predicted_churn_multinomial == 1) & (actual_churn == 0)]
# Print or analyze the misclassified customers
print("Misclassified customers (Gaussian Naive Bayes):")
print(mg)

print("\nMisclassified customers (Bernoulli Naive Bayes):")
print(mb)
print("\nMisclassified customers (Multinomial Naive Bayes):")
print(mt)
```

```
Misclassified customers (Gaussian Naive Bayes):
      call_failure  complains  age_group  status  subs_len  tariff_plan  \
591              0          0          4       2        27            1
61              12          0          3       1        40            1
2951            11          0          3       2        42            1
916              0          0          2       2        36            1
2443             5          0          3       1        21            1
...            ...        ...        ...     ...       ...          ...
1851             0          0          3       2        42            1
991              0          0          4       2        26            1
417              0          0          3       2        35            1
1744             0          0          2       1        16            1
592             10          0          2       1        26            1

      charge_amount  total_num_sms  distinct_call_nums  total_num_calls  \
591               0             53                   7               16
61                1             13                  19               57
2951              0             22                   6               22
916               0              0                  11                3
2443              1              7                  19               57
...             ...            ...                 ...              ...
1851              0              8                   3                5
991               0             21                   3               10
417               0              2                   3                6
1744              0             30                  15               21
592               1             51                  17               38

      total_sec_calls
591               940
61               2298
2951             1333
916               165
2443             3270
...               ...
1851               13
991               825
417               530
1744             1580
592              1340

[143 rows x 11 columns]

Misclassified customers (Bernoulli Naive Bayes):
      call_failure  complains  age_group  status  subs_len  tariff_plan  \
2261            15          1          3       2        44            1
811             11          1          3       2        43            1
2111            14          1          3       2        40            1
261             14          1          3       2        44            1
2661            14          1          4       2        41            1

      charge_amount  total_num_sms  distinct_call_nums  total_num_calls  \
2261              0             25                  23               62
811               0             16                  18               60
2111              0             15                  19               60
261               0             23                  23               64
2661              0             47                  23               66

      total_sec_calls
2261             2623
811              2238
2111             2543
261              2498
2661             2658
```

```
Misclassified customers (Multinomial Naive Bayes):
      call_failure  complains  age_group  status  subs_len  tariff_plan  \
2261            15          1          3       2        44            1
811             11          1          3       2        43            1
2111            14          1          3       2        40            1
261             14          1          3       2        44            1
2661            14          1          4       2        41            1

      charge_amount  total_num_sms  distinct_call_nums  total_num_calls  \
2261              0             25                  23               62
811               0             16                  18               60
2111              0             15                  19               60
261               0             23                  23               64
2661              0             47                  23               66

      total_sec_calls
2261             2623
811              2238
2111             2543
261              2498
2661             2658
```

In [55]:
```
mt=mt.reset_index(drop=True)
```

In [56]:
```
mt.describe()
```

Out[56]:

	call_failure	complains	age_group	status	subs_len	tariff_plan	charge_amount	tot
count	143.000000	143.000000	143.000000	143.000000	143.000000	143.0	143.000000	
mean	4.951049	0.034965	2.790210	1.566434	31.475524	1.0	0.160839	
std	5.247842	0.184337	0.648434	0.497309	7.842492	0.0	0.368674	
min	0.000000	0.000000	1.000000	1.000000	9.000000	1.0	0.000000	
25%	0.000000	0.000000	2.000000	1.000000	25.000000	1.0	0.000000	
50%	4.000000	0.000000	3.000000	2.000000	33.000000	1.0	0.000000	
75%	8.000000	0.000000	3.000000	2.000000	38.000000	1.0	0.000000	
max	20.000000	1.000000	4.000000	2.000000	45.000000	1.0	1.000000	

In [57]:
```
mt
```

Out[57]:

	call_failure	complains	age_group	status	subs_len	tariff_plan	charge_amount	total_num_sr
0	0	0	4	2	27	1	0	
1	12	0	3	1	40	1	1	
2	11	0	3	2	42	1	0	
3	0	0	2	2	36	1	0	
4	5	0	3	1	21	1	1	
...	...	...	...	...	...	...	...	
138	0	0	3	2	42	1	0	
139	0	0	4	2	26	1	0	
140	0	0	3	2	35	1	0	
141	0	0	2	1	16	1	0	
142	10	0	2	1	26	1	1	

143 rows × 11 columns

In [58]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the style for Seaborn plots
sns.set(style="whitegrid")

# Create subplots
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))

# Plot 1 - Call Failure Distribution
sns.histplot(mt['call_failure'], bins=15, kde=True, ax=axes[0, 0])
axes[0, 0].set_title('Call Failure Distribution')

# Plot 2 - Complaints Distribution
sns.countplot(x='complains', data=mt, ax=axes[0, 1])
axes[0, 1].set_title('Complaints Distribution')

# Plot 3 - Age Group Distribution
sns.countplot(x='age_group', data=mt, ax=axes[0, 2])
axes[0, 2].set_title('Age Group Distribution')

# Plot 4 - Status Distribution
sns.countplot(x='status', data=mt, ax=axes[1, 0])
axes[1, 0].set_title('Status Distribution')

# Plot 5 - Subscription Length Distribution
sns.histplot(mt['subs_len'], bins=15, kde=True, ax=axes[1, 1])
axes[1, 1].set_title('Subscription Length Distribution')

# Plot 6 - Charge Amount Distribution
sns.countplot(x='charge_amount', data=mt, ax=axes[1, 2])
axes[1, 2].set_title('Charge Amount Distribution')

# Plot 7 - Total Number of SMS Distribution
sns.histplot(mt['total_num_sms'], bins=15, kde=True, ax=axes[2, 0])
axes[2, 0].set_title('Total Number of SMS Distribution')

# Plot 8 - Distinct Call Numbers Distribution
```

```
sns.histplot(mt['distinct_call_nums'], bins=15, kde=True, ax=axes[2, 1])
axes[2, 1].set_title('Distinct Call Numbers Distribution')

# Plot 9 - Total Number of Calls Distribution
sns.histplot(mt['total_num_calls'], bins=15, kde=True, ax=axes[2, 2])
axes[2, 2].set_title('Total Number of Calls Distribution')

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()
```