Project: Analyzing Customer Churn

Module 2 | Chapter 8 | Notebook 2

In this final project you will analyze the customer churn of a telecommunications company. In this project you will do the following by yourself:

- Clean the data
- Identify targets for marketing campaigns
- Create visualizations to back up your findings

In this notebook you will be provided with very little instruction and structure. Try to get as far as you can. In the next chapter you will have access to the notebook *Project Assistance: Analyzing Customer Churn (Chapter 9)*. That notebook contains the same project, but with additional instruction. If you have any problems, you can access the forum as usual and you can contact StackFuel Support.

Scenario: You work for a telecommunications company called Teleconfia. They are in the process of establishing themselves in the USA. They started by running a trial in Florida. New customers were able to use cellphones on the Teleconfia network at very low rates for one year. Not all customers stayed with Teleconfia for the whole year. Many of them left. This is known as customer churn. The company wants to tackle the problem of customer churn with two marketing campaigns.

The first marketing campaign is aimed at cities and will involve large posters. One city counts as a small town or a district in a larger city. The four cities with the highest rate of customer churn will be selected for the marketing campaign.The second marketing campaign will focus on individual customers. Customers who are likely to leave Teleconfia should receive a phone call to offer them a special deal.

The aim is to identify cities and customers who should be targeted with marketing campaigns. You will need to find out which data series you should use to make these decisions.You also have to find out if there are certain points from which customers should be contacted. Finally, use logistic regression to determine the critical value at which a customer is more likely to churn than remain.

You will have to justify your decisions and recommendations and visualize them.

You should split the project up into the following steps:

- **1) Import** the data

- **2) Check** and **clean** the data

- **3)** What are the names of the **four cities** with the highest rates of customer churn?

- **4a)** Which **categorical** data series should be used to identify customers who will possibly leave soon? Which customers should be contacted based on this data series?

- **4b)** Which **integer** (`int`) data series should be used to identify customers who might leave soon? How would you set the **threshold**? Which customers should be contacted based on this data series?

- **4c)** Which **floating point** (`float`) data series could you use to help with this selection? Determine the **threshold** for this using **logistic regression**. Which customers should be contacted based on this data series?

- **5)** Create **visualization** of the cities and the three other selected data series

- **6)** Formulate a recommendation.

We recommend that you stick to the order of the tasks above.

You have a lot of space to experiment in this notebook. The steps listed above are only given below as subheadings. If you need additional code cells, you can add new ones by clicking on the + button in the top bar, next to the save button (see *A First Look at Python* *(Module 1, Chapter 1)*.

You can access the data in a database named *telco_churn.db*. The following tables contain explanations about the database structure.

`churn_data` table:

Column number	Column name	Type	Description
0	`account_length`	numerical (`int`)	Unknown units of time, how long the customer has been a customer.
1	`international_plan`	categorical (nominal)	Contract with special conditions for cheaper calls to other countries.
2	`voice_mail_plan`	categorical (nominal)	Contract with special conditions for more voicemail storage.
3	`number_vmail_messages`	numerical (`int`)	Number of voicemail messages.
4	`total_day_minutes`	numerical (`float`)	Duration in minutes of all calls from 8am to 4pm.
5	`total_day_calls`	numerical (`int`)	Number of all calls from 8am to 4pm.
6	`total_day_charge`	numerical (`float`)	Calculated costs for all calls from 8am to 4pm.
7	`total_eve_minutes`	numerical (`float`)	Duration in minutes of all calls from 4pm to 10pm.
8	`total_eve_calls`	numerical (`int`)	Number of all calls from 4pm to 10pm.
9	`total_eve_charge`	numerical (`float`)	Calculated costs for all calls from 4pm to 10pm.
10	`total_night_minutes`	numerical (`float`)	Duration in minutes of all calls from 10pm to 8am.
11	`total_night_calls`	numerical	Number of all calls from 10pm to 8am.

Column number	Column name	Type	Description
		(int)	
12	total_night_charge	numerical (float)	Calculated costs for all calls from 10pm to 8am.
13	customer_service_calls	numerical (int)	Number of calls to customer service, e.g. due to technical problems.
14	churn	categorical (nominal)	Did the customer leave? (1=yes 0=no)
15	local_area_code	categorical (nominal)	local area code for telephone.
16	phone_num	Categorical (nominal)	Customers telephone number not including the local area code.

`cities` table:

Column number	Column name	Type	Description
0	city	categorical (nominal)	Cities.
1	area_code	categorical (nominal)	local area code for telephone.

Congratulations: You have gone through the task description. Now you can work your way through the data pipeline as you have learned and practiced throughout this module: importing, cleaning, exploring, analyzing and visualizing the data. Now you will need to use the SQL skills you learned in Chapter 4.

1) Importing the data

For this step you will need to use things you learned in the following lessons:

- *Preparing Data with pandas (Chapter 1)*
- *Querying a Database (Chapter 4)*
- *Combining Two Tables (Chapter 4)*

In [1]:

```
# import libraries
import pandas as pd
import sqlalchemy as sa

#create an engine
engine = sa.create_engine('sqlite:///telco_churn.db')
connection = engine.connect()

#create an inspector
inspector = sa.inspect(engine)

# Get the list of table names in the database
table_names = inspector.get_table_names()
table_names
```

Out[1]:

```
['churn_data', 'cities']
```

SQL query to fetch the first 10 rows from the 'churn_data' table

In [2]:

```
query_churn_data = '''SELECT *
FROM churn_data
LIMIT 10
'''
df_churn_data = pd.read_sql(query_churn_data, connection)
df_churn_data
```

Out[2]:

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	tc
0	131.0	no	no	0.0	187.9	
1	63.0	yes	yes	21.0	151.5	
2	13.0	no	no	0.0	303.2	
3	129.0	no	no	0.0	159.1	
4	120.0	no	yes	28.0	215.8	
5	108.0	no	no	0.0	210.7	
6	109.0	no	no	0.0	180.0	
7	107.0	no	no	0.0	189.7	
8	74.0	no	no	0.0	282.5	
9	28.0	no	no	0.0	168.2	

SQL query to fetch the first 10 rows from the 'cities' table

In [3]:

```
query_cities = '''SELECT *
FROM cities
LIMIT 10
'''
df_cities = pd.read_sql(query_cities, connection)
df_cities
```

Out[3]:

	city	area_code
0	Orlando1	321
1	Orlando2	407
2	Miami1	305
3	Miami2	786
4	Jacksonville	904
5	Tampa	813
6	West Palm Beach	561
7	Daytona Beach	386

	city	area_code
8	Clearwater	727
9	Sarasota	941

Joining both tables

In [4]:

```
# This is a SQL query string that selects all columns from the 'churn_data' table
# and joins it with the 'cities' table based on the 'local_area_code' and 'area_code
query_string = '''SELECT *
FROM churn_data
JOIN cities
ON churn_data.local_area_code = cities.area_code'''
df = pd.read_sql(query_string, connection)


# Read complete data using the sql query.
df
```

Out[4]:

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
0	131.0	no	no	0.0	187.9
1	63.0	yes	yes	21.0	151.5
2	13.0	no	no	0.0	303.2
3	129.0	no	no	0.0	159.1
4	120.0	no	yes	28.0	215.8
...	...	...	...	...	...
3328	94.0	no	no	0.0	190.4
3329	158.0	no	no	0.0	158.0
3330	67.0	no	no	0.0	171.7
3331	105.0	no	yes	27.0	141.2
3332	124.0	no	no	0.0	178.4

3333 rows × 19 columns

Close connection

In [5]:

```
connection.close()
```

Congratulations: You imported the data. Now you can begin cleaning it. This is such a varied task that you will need to use skills you have learned from almost every chapter in the second module. Don't let the task overwhelm you. Proceed step by step, checking different aspects of the dataset.

2) Check and clean the data

For this step you will need to use things you learned in the following lessons:

- *Preparing Data with pandas (Chapter 1)*
- *Analyzing Data with pandas (1) (Chapter 1)*
- *Exploring Data (Chapter 2)*
- *Project: Beer Ratings (Data Cleaning) (Chapter 3))*
- *Boolean Masking (Chapter 4)*
- *Project: Share Prices (Preparing Data) (Chapter 6)*

Check the dtypes

In [6]:
```
df.dtypes
```

Out[6]:
```
account_length              float64
international_plan           object
voice_mail_plan              object
number_vmail_messages       float64
total_day_minutes           float64
total_day_calls             float64
total_day_charge            float64
total_eve_minutes           float64
total_eve_calls             float64
total_eve_charge            float64
total_night_minutes         float64
total_night_calls           float64
total_night_charge          float64
customer_service_calls      float64
churn                         int64
local_area_code             float64
phone_num                   float64
city                         object
area_code                     int64
dtype: object
```

The columns 'city', 'international_plan', 'churn' and 'voice_mail_plan' have the object data type.

converted these into category data type

In [7]:
```
df['city']  = df['city'].astype('category')
df['international_plan']  = df['international_plan'].astype('category')
df['voice_mail_plan']  = df['voice_mail_plan'].astype('category')
df['churn']  = df['churn'].astype('category')
df.dtypes
```

Out[7]:
```
account_length              float64
international_plan         category
voice_mail_plan            category
number_vmail_messages       float64
total_day_minutes           float64
total_day_calls             float64
total_day_charge            float64
total_eve_minutes           float64
total_eve_calls             float64
total_eve_charge            float64
total_night_minutes         float64
total_night_calls           float64
total_night_charge          float64
customer_service_calls      float64
```

```
churn                       category
local_area_code              float64
phone_num                    float64
city                        category
area_code                      int64
dtype: object
```

Check for the duplicate column

In [8]:
```
duplicate_columns = df.columns[df.columns.duplicated()]
duplicate_columns
```

Out[8]:
```
Index([], dtype='object')
```

In [9]:
```
#here there are no duplicated columns as dataframe is empty.
df.shape
```

Out[9]:
```
(3333, 19)
```

Check the missing values

In [10]:
```
print(df.isnull().sum(), '\n')
print('Total missing values = ',df.isnull().sum().sum())
```

```
account_length            0
international_plan        0
voice_mail_plan           0
number_vmail_messages    22
total_day_minutes         0
total_day_calls          11
total_day_charge          0
total_eve_minutes         0
total_eve_calls           8
total_eve_charge          0
total_night_minutes       0
total_night_calls         5
total_night_charge        0
customer_service_calls    0
churn                     0
local_area_code           0
phone_num                 0
city                      0
area_code                 0
dtype: int64

Total missing values =  46
```

Removed all those rows having missing values

In [11]:
```
#df.dropna will remove all the rows having missing values
df_cleaned = df.dropna()
df_cleaned.head(5)
```

Out[11]:

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	tc
0	131.0	no	no	0.0	187.9	

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	tc
1	63.0	yes	yes	21.0	151.5	
2	13.0	no	no	0.0	303.2	
3	129.0	no	no	0.0	159.1	
4	120.0	no	yes	28.0	215.8	

In [12]:
```
df_cleaned.shape
```

Out[12]:
```
(3287, 19)
```

In [13]:
```
#verified those missing values

df_cleaned.isnull().sum()
```

Out[13]:
```
account_length           0
international_plan       0
voice_mail_plan          0
number_vmail_messages    0
total_day_minutes        0
total_day_calls          0
total_day_charge         0
total_eve_minutes        0
total_eve_calls          0
total_eve_charge         0
total_night_minutes      0
total_night_calls        0
total_night_charge       0
customer_service_calls   0
churn                    0
local_area_code          0
phone_num                0
city                     0
area_code                0
dtype: int64
```

To Check for any wrong or impossible values

In [14]:
```
df_cleaned.describe()
```

Out[14]:

	account_length	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge
count	3287.000000	3287.000000	3287.000000	3287.000000	3287.000000
mean	101.071494	8.167630	180.082081	100.463645	30.595875
std	39.741711	13.726955	54.912752	19.968793	9.267864
min	1.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	0.000000	143.700000	87.000000	24.430000
50%	101.000000	0.000000	179.700000	101.000000	30.550000
75%	127.000000	20.000000	216.750000	114.000000	36.850000
max	232.000000	51.000000	448.600000	165.000000	59.640000

*From the above result,I found, there is minimum value '-2' in the 'customer_service_calls'...which seems impossible.

So I tried to replace all the negative values from 'customer_service_calls' column with the median value.

In [15]:

```
#We have the DataFrame df_cleaned with the column 'customer_service_calls'

# Calculated the mean of 'customer_service_calls'
column_mean = df_cleaned['customer_service_calls'].median()

# Define a custom function to replace negative values with the mean
def replace_negative_with_mean(x):
    return column_mean if x < 0 else x

# Use the custom function with apply to replace negative values in the column
df_cleaned['customer_service_calls'] = df_cleaned['customer_service_calls'].apply(re

#here we can see minmum values of all the columns are non negative.
df_cleaned.describe()
```

```
/tmp/ipykernel_3371/3979867799.py:11: 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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['customer_service_calls'] = df_cleaned['customer_service_calls'].apply
(replace_negative_with_mean)
```

Out[15]:

	account_length	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge
count	3287.000000	3287.000000	3287.000000	3287.000000	3287.000000
mean	101.071494	8.167630	180.082081	100.463645	30.595875
std	39.741711	13.726955	54.912752	19.968793	9.267864
min	1.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	0.000000	143.700000	87.000000	24.430000
50%	101.000000	0.000000	179.700000	101.000000	30.550000
75%	127.000000	20.000000	216.750000	114.000000	36.850000
max	232.000000	51.000000	448.600000	165.000000	59.640000

In [16]:

```
df_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3287 entries, 0 to 3332
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   account_length          3287 non-null   float64
 1   international_plan      3287 non-null   category
 2   voice_mail_plan         3287 non-null   category
 3   number_vmail_messages   3287 non-null   float64
 4   total_day_minutes       3287 non-null   float64
 5   total_day_calls         3287 non-null   float64
```

```
 6   total_day_charge        3287 non-null   float64
 7   total_eve_minutes       3287 non-null   float64
 8   total_eve_calls         3287 non-null   float64
 9   total_eve_charge        3287 non-null   float64
 10  total_night_minutes     3287 non-null   float64
 11  total_night_calls       3287 non-null   float64
 12  total_night_charge      3287 non-null   float64
 13  customer_service_calls  3287 non-null   float64
 14  churn                   3287 non-null   category
 15  local_area_code         3287 non-null   float64
 16  phone_num               3287 non-null   float64
 17  city                    3287 non-null   category
 18  area_code               3287 non-null   int64
dtypes: category(4), float64(14), int64(1)
memory usage: 424.4 KB
```

Congratulations: You have cleaned the data. For a lot of data projects, this can take up just as much time as analyzing the data. You can move onto that now.

3) What are the names of the **four cities** with the highest rates of customer churn?

For this step you will need to use things you learned in the following lessons:

- *Exploring Categories* (Chapter 2)
- *Boolean Masking* (Chapter 4)

Total number of customer who have churned

***First, I tried to find all the customers who left the network**

In [17]:

```
# What is the overall churn ratio in the dataset?

customers_left= df_cleaned[df_cleaned['churn'] == 1]
print('Total customers who left the network: ',customers_left['churn'].count(), '\n'
customers_left
```

```
Total customers who left the network:  478
```

Out[17]:

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
2	13.0	no	no	0.0	303.2
8	74.0	no	no	0.0	282.5
10	34.0	no	no	0.0	293.7
20	110.0	yes	no	0.0	293.3
25	108.0	yes	no	0.0	115.1
...	...	...	...	...	...
3289	46.0	no	no	0.0	250.3
3299	50.0	yes	no	0.0	99.6
3302	80.0	no	no	0.0	268.7
3305	100.0	no	no	0.0	70.8
3324	114.0	no	no	0.0	147.1

478 rows × 19 columns

*sorted total number of churn for each city in descending ordr

In [18]:
```
# Applied groupby on city column to extact all the information about each city
# then used count
df_city_churned=customers_left.groupby('city').count()
customers_left_city = customers_left.groupby('city')['churn'].count()
customers_left_city_sort = customers_left_city.sort_values(ascending=False)
pd.DataFrame(customers_left_city_sort)
```

Out[18]:

city	churn
Jacksonville	99
Orlando1	70
Cape Coral	64
Orlando2	58
Daytona Beach	46
Miami1	35
Miami2	31
Sarasota	23
Tallahassee	17
Clearwater	17
West Palm Beach	9
Tampa	9

Total number of customer who joined the network from each city

In [19]:
```
collected_samples_entries_city = df_cleaned.groupby('city').count()
collected_samples_entries_city.sort_values(by='account_length', ascending = False)
```

Out[19]:

city	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_n
Jacksonville	326	326	326	326	
Orlando2	300	300	300	300	
Orlando1	298	298	298	298	
Cape Coral	295	295	295	295	
Miami2	288	288	288	288	
Tallahassee	277	277	277	277	
Daytona	275	275	275	275	

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_n
city					
Beach					
Miami1	258	258	258	258	
Tampa	251	251	251	251	
Sarasota	246	246	246	246	
Clearwater	240	240	240	240	
West Palm Beach	233	233	233	233	

Churn ration of each city

In [20]:

```
# Calculate churn ratio for cities in df_city_churned and sort in descending order
#Customers who left the network from each city in PERCENTAGE:

df_city_churned['churn_ratio']=df_city_churned['account_length']/collected_samples_e
df_city_churned_ratio = df_city_churned[['churn_ratio','account_length']].sort_value
df2=df_city_churned_ratio.rename(columns={'account_length': 'churn_nos.'})
df2
```

Out[20]:

	churn_ratio	churn_nos.
city		
Jacksonville	30.368098	99
Orlando1	23.489933	70
Cape Coral	21.694915	64
Orlando2	19.333333	58
Daytona Beach	16.727273	46
Miami1	13.565891	35
Miami2	10.763889	31
Sarasota	9.349593	23
Clearwater	7.083333	17
Tallahassee	6.137184	17
West Palm Beach	3.862661	9
Tampa	3.585657	9

AVG churn ratio

In [21]:

```
print('Average churn ratio from the cities: ',df2['churn_ratio'].mean())
```

```
Average churn ratio from the cities:  13.830146831794282
```

Top 4 cities with the highest churn ratio

In [22]:

```
collected_samples_entries_city.sort_values('account_length', ascending= False).head(
```

Out[22]:

city	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_n
Jacksonville	326	326	326	326	
Orlando2	300	300	300	300	
Orlando1	298	298	298	298	
Cape Coral	295	295	295	295	

In [23]:

```
# Here we have these 4 cities having highest churn ratio.
df_city_churned_ratio.rename(columns={'account_length': 'churn_nos.'}).head(4)
```

Out[23]:

city	churn_ratio	churn_nos.
Jacksonville	30.368098	99
Orlando1	23.489933	70
Cape Coral	21.694915	64
Orlando2	19.333333	58

Congratulations: You have identified four cities where the marketing campaign should be launched. Next you will identify individual customers who should be contacted.

4a) Which **categorical** data series should be used to identify customers who will possibly leave soon? Which customers should be contacted based on this data series?

For this step, you will find things you learned in the following lessons helpful:

- *Exploring Data* (Chapter 2)
- *Exploring Categories* (Chapter 2)
- *Boolean Masking* (Chapter 4)

***Here I tried to found which columns are categorical columns**

I found that *voice_mail_plan,* international_plan, *area_code,* city, *churn have categorical data type

It is appearntly logical that we can use only two columns (voice_mail_plan,international_plan) analyze to identify customers who will possibly leave soon.

In [24]:

```
df_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3287 entries, 0 to 3332
Data columns (total 19 columns):
```

```
 #   Column                 Non-Null Count  Dtype   
---  ------                 --------------  -----   
 0   account_length         3287 non-null   float64 
 1   international_plan     3287 non-null   category
 2   voice_mail_plan        3287 non-null   category
 3   number_vmail_messages  3287 non-null   float64 
 4   total_day_minutes      3287 non-null   float64 
 5   total_day_calls        3287 non-null   float64 
 6   total_day_charge       3287 non-null   float64 
 7   total_eve_minutes      3287 non-null   float64 
 8   total_eve_calls        3287 non-null   float64 
 9   total_eve_charge       3287 non-null   float64 
 10  total_night_minutes    3287 non-null   float64 
 11  total_night_calls      3287 non-null   float64 
 12  total_night_charge     3287 non-null   float64 
 13  customer_service_calls 3287 non-null   float64 
 14  churn                  3287 non-null   category
 15  local_area_code        3287 non-null   float64 
 16  phone_num              3287 non-null   float64 
 17  city                   3287 non-null   category
 18  area_code              3287 non-null   int64   
dtypes: category(4), float64(14), int64(1)
memory usage: 424.4 KB
```

Total number customers having international_plan

In [25]:

```
#df_cleaned[df_cleaned['international_plan'] == 'yes']
```

In [26]:

```
pd.crosstab(index = df_cleaned.loc[:, 'international_plan'], columns = df_cleaned.lc
```

Out[26]:

churn	0	1
international_plan		
no	2624	341
yes	185	137

In [27]:

```
pd.crosstab(index = df_cleaned.loc[:, 'international_plan'], columns = df_cleaned.lc
```

Out[27]:

churn	0	1
international_plan		
no	0.884992	0.115008
yes	0.574534	0.425466
All	0.854579	0.145421

Total number and index of customers who had international_plan and left the network

In [28]:

```
customers_left_international_plan=df_cleaned[(df_cleaned['international_plan'] == 'y
customers_left_international_plan[['international_plan', 'churn','voice_mail_plan']]
```

Out[28]:

	international_plan	churn	voice_mail_plan
20	yes	1	no
25	yes	1	no
42	yes	1	yes
107	yes	1	no
114	yes	1	no
...	...	...	...
3267	yes	1	yes
3269	yes	1	no
3283	yes	1	no
3286	yes	1	no
3299	yes	1	no

137 rows × 3 columns

Here it can be observed that over 40% customers left the network who had international_plan.

Here I found the percenage.

In [29]:

```
137/332*100
```

Out[29]:

```
41.265060240963855
```

Total number of customer having 'voice_mail_plan'

In [30]:

```
#df_cleaned[df_cleaned['voice_mail_plan'] == 'yes']
```

In [31]:

```
pd.crosstab(index = df_cleaned.loc[:, 'voice_mail_plan'], columns = df_cleaned.loc[:
```

Out[31]:

churn	0	1
voice_mail_plan		
no	1972	398
yes	837	80

In [32]:

```
pd.crosstab(index = df_cleaned.loc[:, 'voice_mail_plan'], columns = df_cleaned.loc[:
```

Out[32]:

churn	0	1
voice_mail_plan		
no	0.832068	0.167932
yes	0.912759	0.087241

churn	0	1
voice_mail_plan		
All	0.854579	0.145421

Total number and index of customers who had voice_mail_plan and left the network

In [33]:
```
customers_left_international_plan = df_cleaned[(df_cleaned['voice_mail_plan'] == 'ye
customers_left_international_plan[['voice_mail_plan', 'churn']]
```

Out[33]:

	voice_mail_plan	churn
42	yes	1
47	yes	1
119	yes	1
130	yes	1
142	yes	1
...	...	...
3196	yes	1
3202	yes	1
3204	yes	1
3256	yes	1
3267	yes	1

80 rows × 2 columns

Here it can be observed that around 9% customers left the network who had voice_mail_plan.

In [34]:
```
(80/917)*100
```

Out[34]:
```
8.724100327153762
```

here we found that in international_plan column CHURN RATIO is 40% whie in the voice_mail_plan column churn ratio is around 9 percentage

which is toooo low in comparision of international_plan.

Hence international_plan is our desired column which will be used to identify the customers who should be contacted.

Customers who should be contacted.

In [35]:
```
customers_left_international_plan=df_cleaned[(df_cleaned['international_plan'] == 'y
customers_left_international_plan[['international_plan', 'churn']]
```

Out[35]:

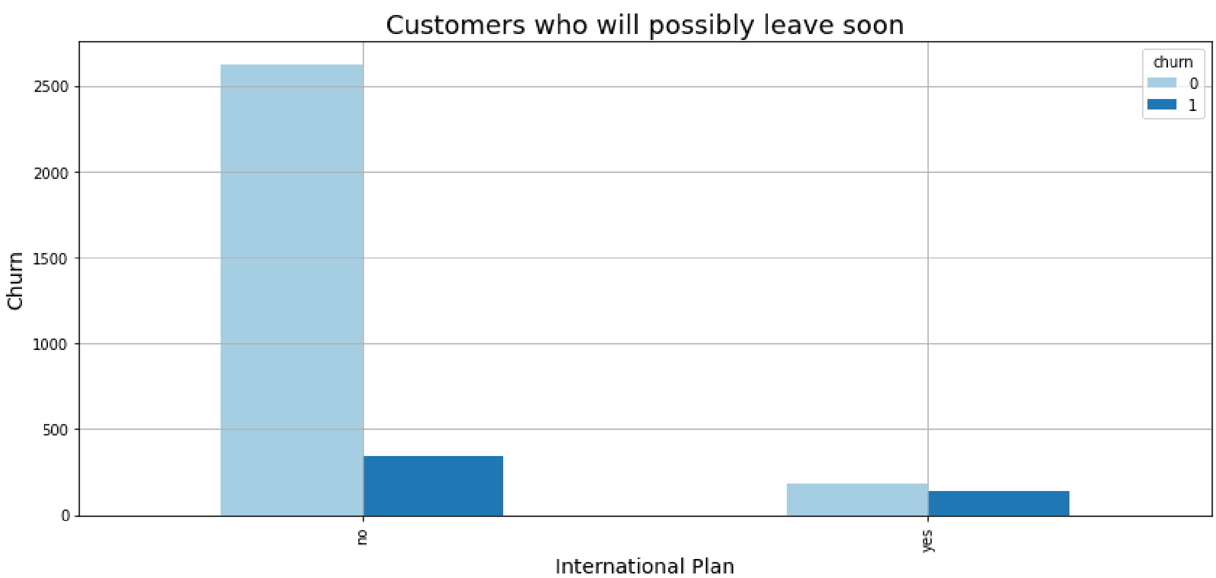
	international_plan	churn
1	yes	0
24	yes	0
33	yes	0
57	yes	0
59	yes	0
...	...	...
3258	yes	0
3259	yes	0
3278	yes	0
3295	yes	0
3301	yes	0

185 rows × 2 columns

Bar plot for the category which has high risk of churning

In [36]:

```
%matplotlib inline
import matplotlib.pyplot as plt
colors = plt.cm.Paired.colors

var_cross_tab = pd.crosstab(index=df_cleaned.loc[:, 'international_plan'],
                            columns=df_cleaned.loc[:, 'churn'])

ax = var_cross_tab.plot(kind='bar', legend=True, title='Customers who will possibly
                        figsize=(14, 6))
ax.grid(True)
plt.title('Customers who will possibly leave soon', fontsize=18)
plt.xlabel('International Plan', fontsize=14)
plt.ylabel('Churn', fontsize=14)
plt.show()
```



In [37]:

```
df_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3287 entries, 0 to 3332
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype   
---  ------                  --------------  -----   
 0   account_length          3287 non-null   float64 
 1   international_plan      3287 non-null   category
 2   voice_mail_plan         3287 non-null   category
 3   number_vmail_messages   3287 non-null   float64 
 4   total_day_minutes       3287 non-null   float64 
 5   total_day_calls         3287 non-null   float64 
 6   total_day_charge        3287 non-null   float64 
 7   total_eve_minutes       3287 non-null   float64 
 8   total_eve_calls         3287 non-null   float64 
 9   total_eve_charge        3287 non-null   float64 
 10  total_night_minutes     3287 non-null   float64 
 11  total_night_calls       3287 non-null   float64 
 12  total_night_charge      3287 non-null   float64 
 13  customer_service_calls  3287 non-null   float64 
 14  churn                   3287 non-null   category
 15  local_area_code         3287 non-null   float64 
 16  phone_num               3287 non-null   float64 
 17  city                    3287 non-null   category
 18  area_code               3287 non-null   int64   
dtypes: category(4), float64(14), int64(1)
memory usage: 424.4 KB
```

Congratulations: You have found a way to categorize the customers in relation to customer churn. Next you will focus on a numerical data series that also correlates with customer churn.

4b) Which **integer** data series would you also use for this and how would you set the **threshold**? Which customers should be contacted based on this data series?

You might find the following lessons helpful for this step:

- *Exploring Data (Chapter 2)*
- *Visualizing Data Distributions using Histograms (Chapter 2)*
- *Exploring Categories (Chapter 2)*
- *Importing and Cleaning a Business Data Set (Chapter 2)*
- *Box Plots (Chapter 3)*
- *Boolean Masking (Chapter 4)*

Extracting a subset of integer columns from the cleaned DataFrame for further analysis.

In [38]:

```
df_cleaned['account_length'] = df_cleaned['account_length'].astype('int')
df_cleaned['number_vmail_messages'] = df_cleaned['number_vmail_messages'].astype('in
df_cleaned['total_day_calls'] = df_cleaned['total_day_calls'].astype('int')
df_cleaned['total_eve_calls'] = df_cleaned['total_eve_calls'].astype('int')
df_cleaned['total_night_calls'] = df_cleaned['total_night_calls'].astype('int')
df_cleaned['customer_service_calls'] = df_cleaned['customer_service_calls'].astype('
```

```
/tmp/ipykernel_3371/1038106163.py:1: 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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['account_length'] = df_cleaned['account_length'].astype('int')
/tmp/ipykernel_3371/1038106163.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['number_vmail_messages'] = df_cleaned['number_vmail_messages'].astype
('int')
/tmp/ipykernel_3371/1038106163.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['total_day_calls'] = df_cleaned['total_day_calls'].astype('int')
/tmp/ipykernel_3371/1038106163.py:4: 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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['total_eve_calls'] = df_cleaned['total_eve_calls'].astype('int')
/tmp/ipykernel_3371/1038106163.py:5: 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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['total_night_calls'] = df_cleaned['total_night_calls'].astype('int')
/tmp/ipykernel_3371/1038106163.py:6: 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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['customer_service_calls'] = df_cleaned['customer_service_calls'].astype
('int')
```

In [39]:

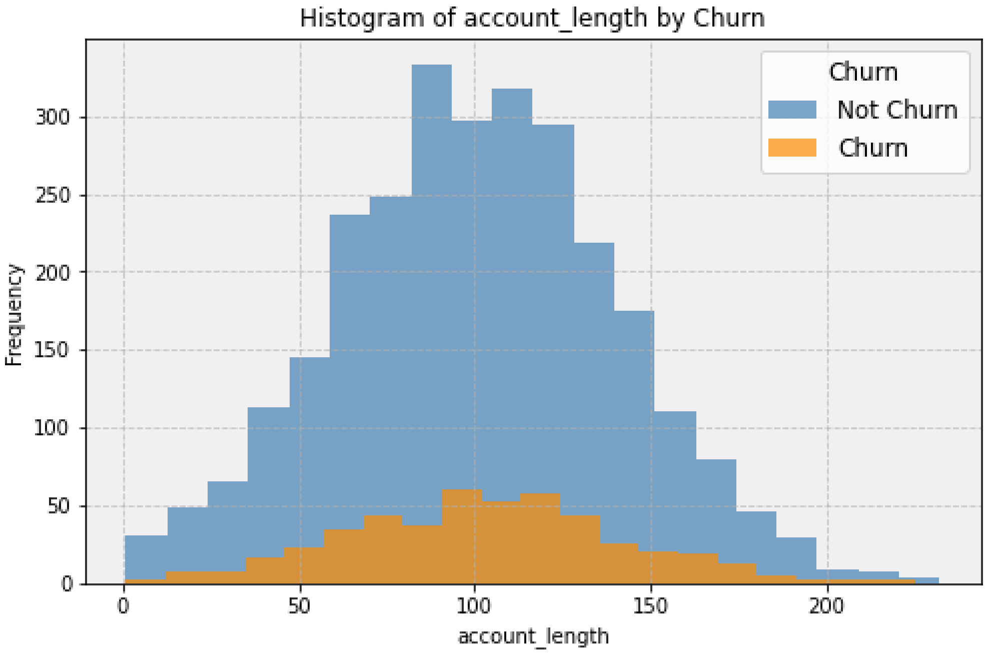
```
df_int_columns = df_cleaned[['account_length', 'number_vmail_messages', 'total_day_c
                  'total_night_calls', 'customer_service_calls']]
```

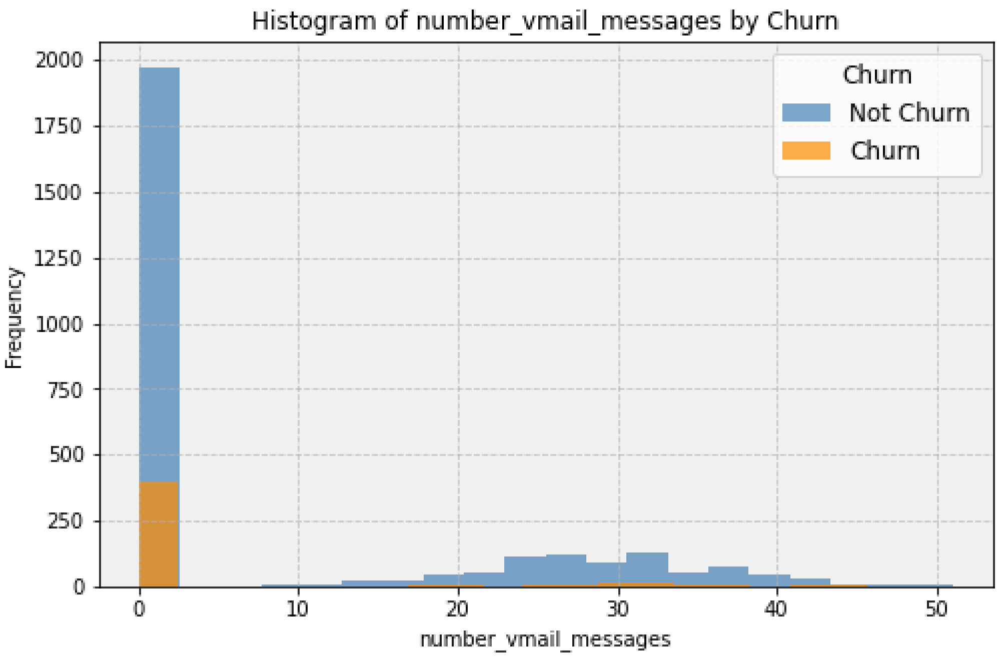
Plotted histograms for each df_int_columns

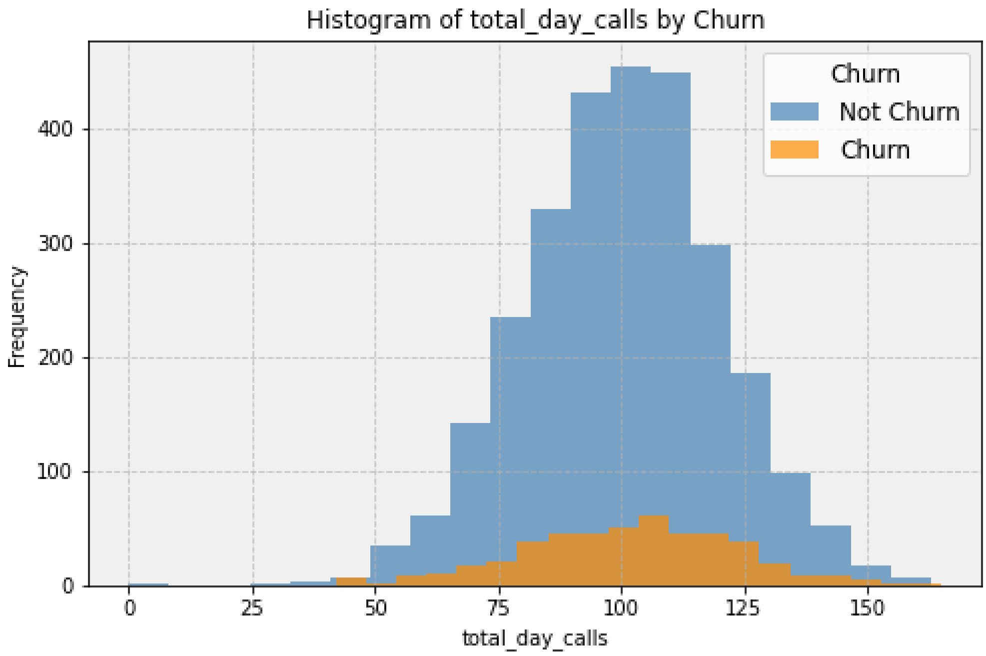
In [40]:

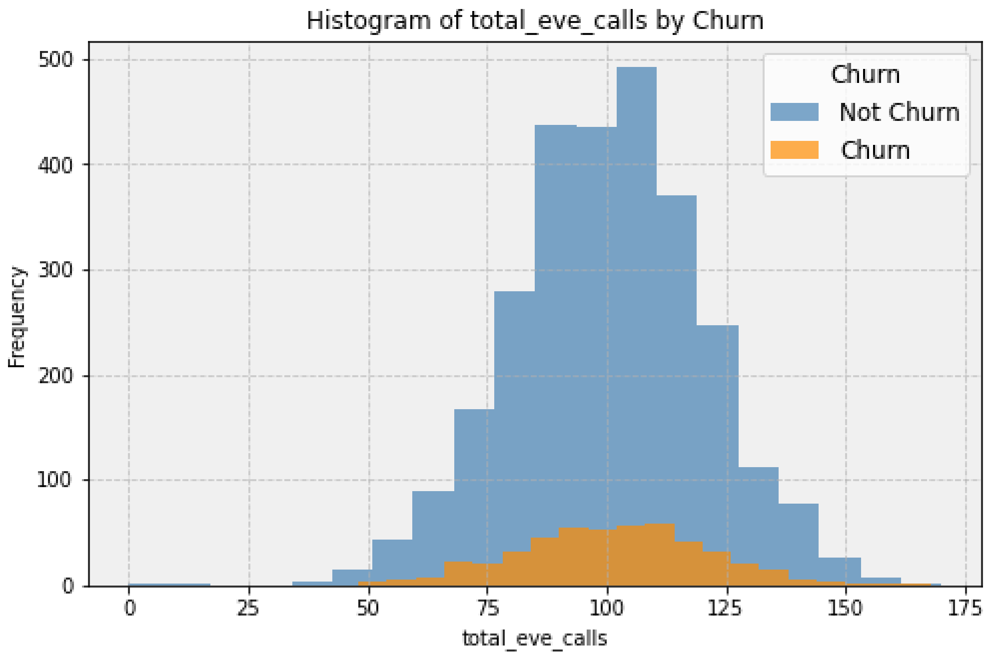
```
import matplotlib.pyplot as plt

colors = ['steelblue', 'darkorange']

for i in df_int_columns:
    plt.figure(figsize=(8, 5))
    for churn_val, color in zip(df_cleaned['churn'].unique(), colors):
        data = df_cleaned[df_cleaned['churn'] == churn_val][i]
        plt.hist(data, bins=20, alpha=0.7, color=color, label='Churn' if churn_val =

    plt.title('Histogram of {} by Churn'.format(i))
```

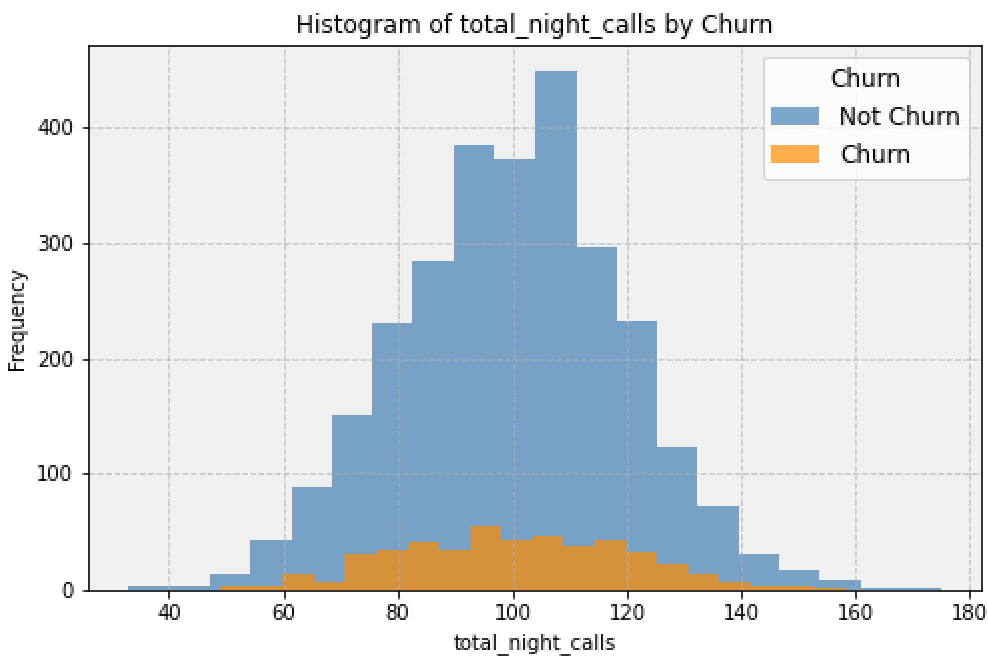
```
    plt.xlabel(i)
    plt.ylabel('Frequency')
    plt.legend(title='Churn', labels=['Not Churn', 'Churn'], fontsize=12, title_font

    plt.grid(True, linestyle='--', alpha=0.7)
    plt.gca().set_facecolor('#f0f0f0')

    plt.show()
```

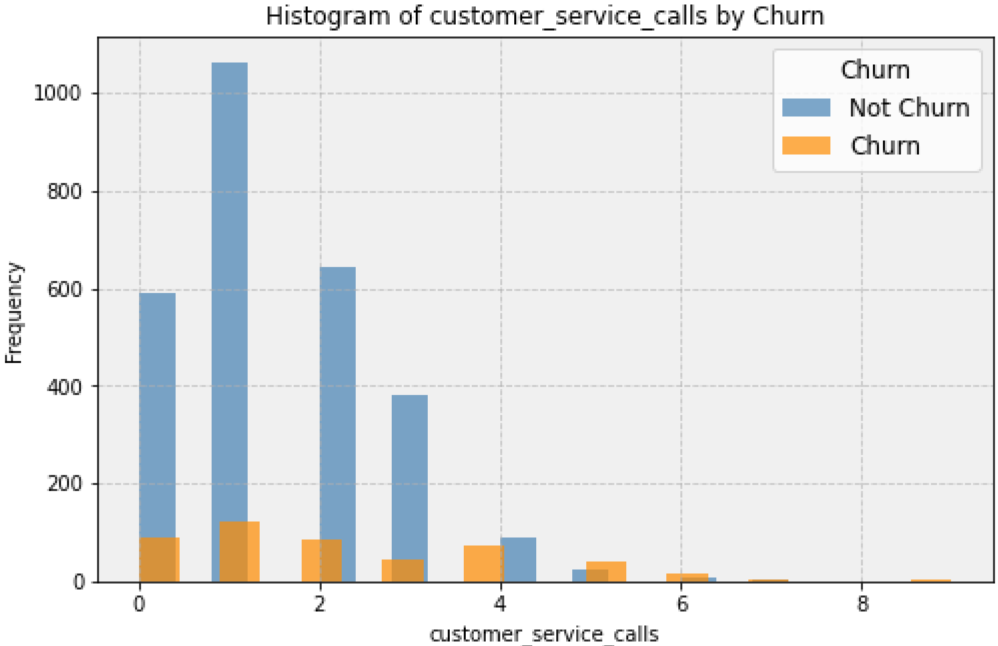





Histogram of total_day_calls by Churn
Churn
Not Churn
Churn
Frequency
total_day_calls
0
100
200
300
400
25
50
75
125
150


Histogram of total_eve_calls by Churn
Churn
Not Churn
Churn
Frequency
total_eve_calls
0
100
200
300
400
500
25
50
75
125
150
175


Histogram of total_night_calls by Churn
Churn
Not Churn
Churn
Frequency
total_night_calls
0
100
200
300
400
40
60
80
120
140
160
180

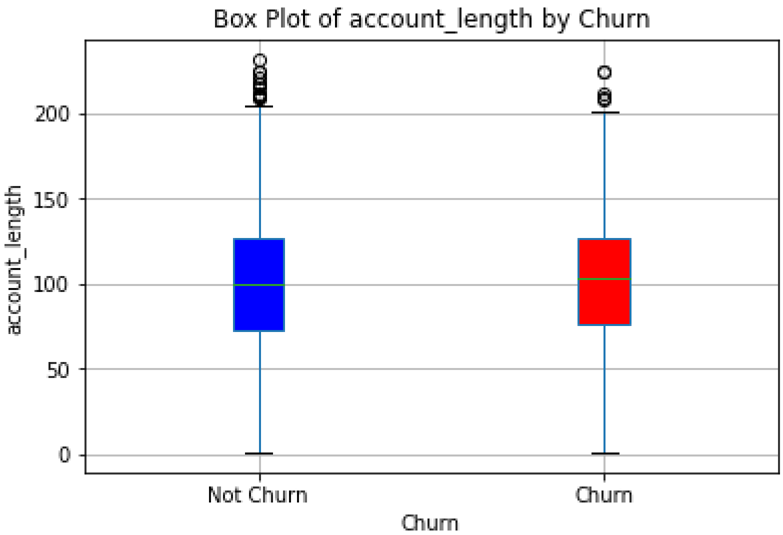


Box plot for each df_int_column

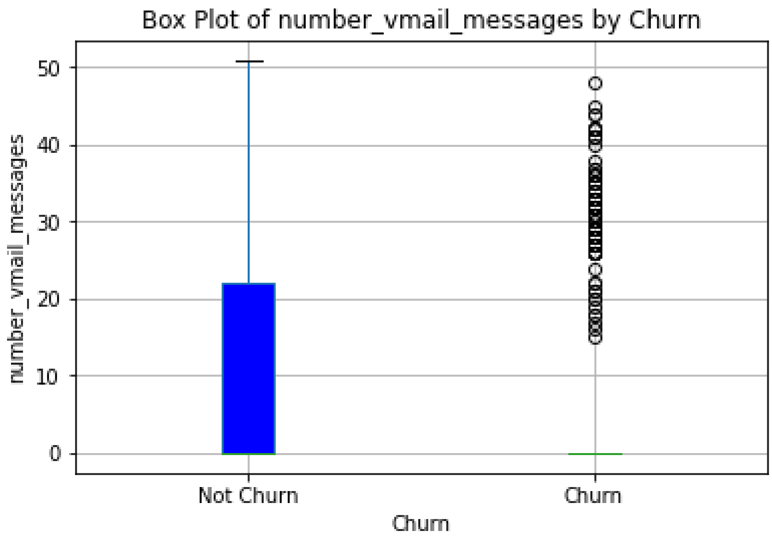
In [41]:

```
import matplotlib.pyplot as plt

for i in df_int_columns:
    plt.figure()  # Create a new figure for each box plot
    ax = df_cleaned.boxplot(column=i, by='churn', grid=True, patch_artist=True)  # S
    plt.title('Box Plot of {} by Churn'.format(i))  # Add a title to the plot using
    plt.xlabel('Churn')  # Add an x-axis label
    plt.ylabel(i)  # Add a y-axis label (column name)
    plt.suptitle('')  # Remove the default title generated by pandas boxplot

    colors = ['blue', 'red']  # Set custom colors for the boxes (in the order of 'No
    for box, color in zip(ax.artists, colors):
        box.set_facecolor(color)  # Set the box color

    plt.xticks([1, 2], ['Not Churn', 'Churn'])  # Set custom x-axis labels
    plt.show()  # Show the plot
```

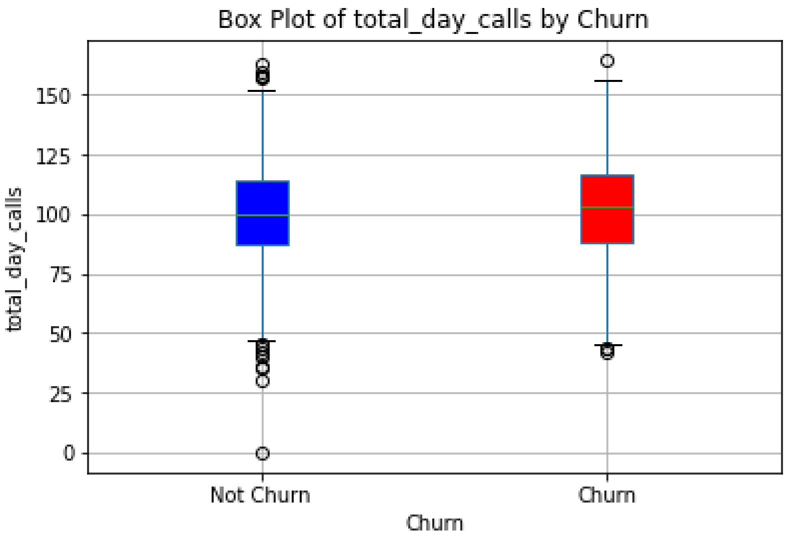
```
<Figure size 432x288 with 0 Axes>
```



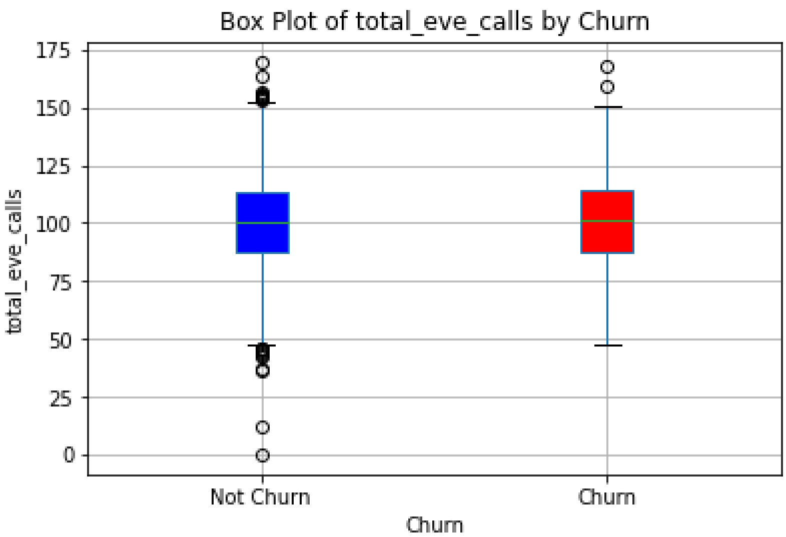

```
<Figure size 432x288 with 0 Axes>
```



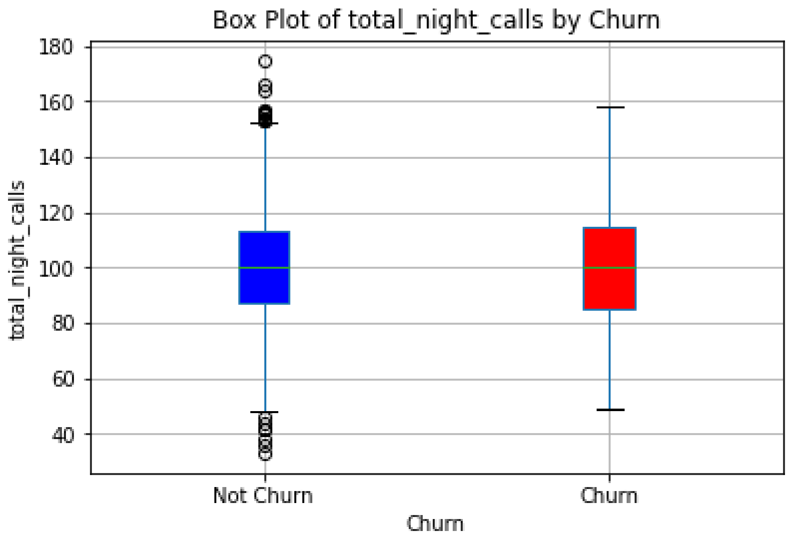
<Figure size 432x288 with 0 Axes>



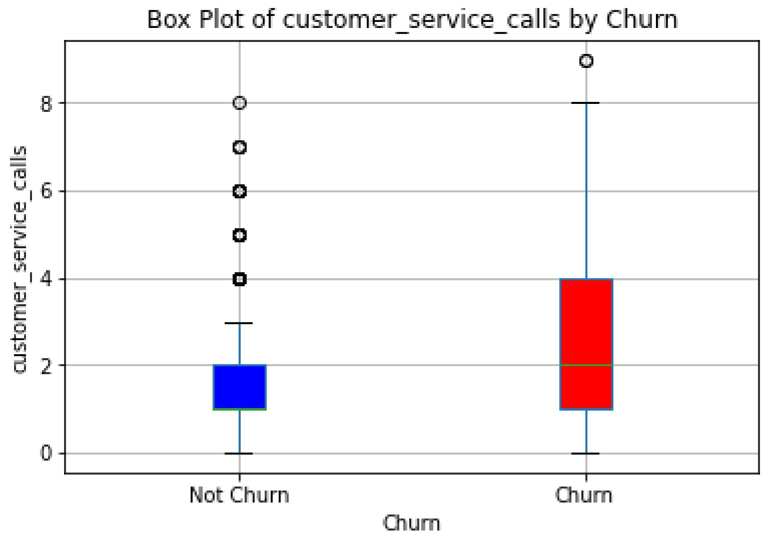
<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



From these histograms and from the bar plot it has been observed that at some point the churn is HIGHER than non churn of custmoers,

This insight is NOT there for any other column,

Hence, It is decided that 'customer_service_calls' is the desried column to determin who should be contacted.

Number of customers who called to customer service more than 3 times are on high risk to churn

In [42]:

```
pd.crosstab(index = df_cleaned.loc[:, 'customer_service_calls'], columns = df_cleane
```

Out[42]:

churn	0	1
customer_service_calls		
0	592	91
1	1063	124
2	644	84
3	381	44
4	90	74

churn	0	1
customer_service_calls		
5	26	39
6	8	14
7	4	5
8	1	1
9	0	2

In [43]:

```
compare=pd.crosstab(index = df_cleaned.loc[:, 'customer_service_calls'], columns = d
compare
```

Out[43]:

churn	0	1
customer_service_calls		
0	0.866764	0.133236
1	0.895535	0.104465
2	0.884615	0.115385
3	0.896471	0.103529
4	0.548780	0.451220
5	0.400000	0.600000
6	0.363636	0.636364
7	0.444444	0.555556
8	0.500000	0.500000
9	0.000000	1.000000
All	0.854579	0.145421

In [44]:

```
compare['Difference'] = compare[0] - compare[1]
compare

# The diffrence between customers who,
# left and didn't left the network is very less for those who called at customers se
# based on this insight I would prefer to set my cuttoff for this column is 'Custome
```

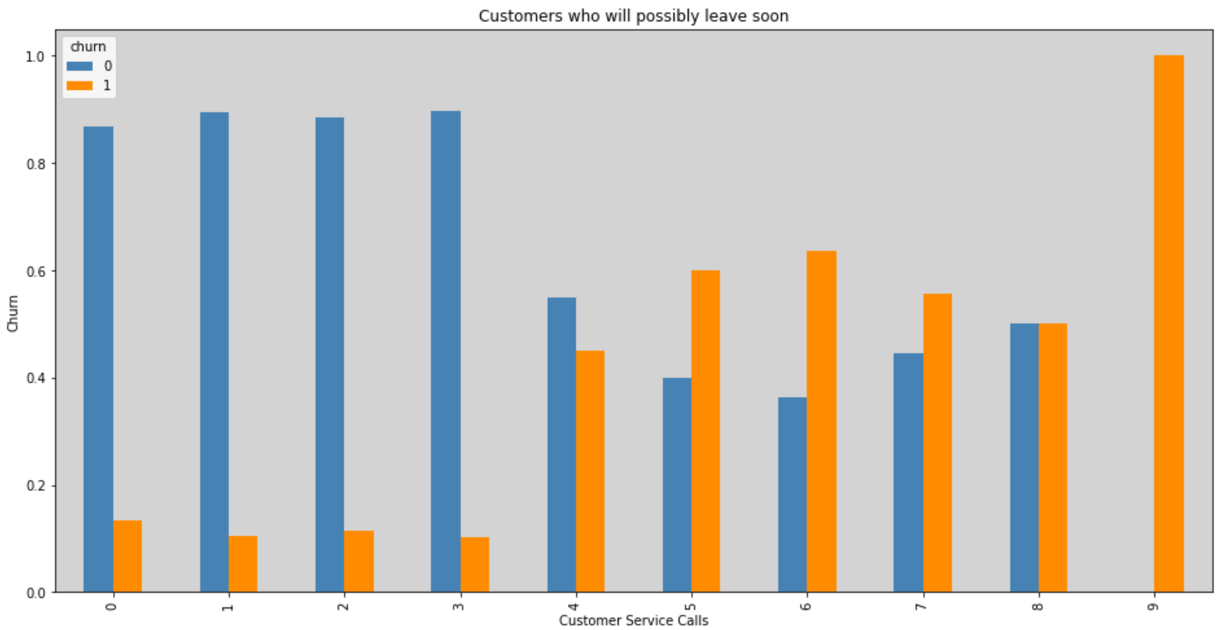
Out[44]:

churn	0	1	Difference
customer_service_calls			
0	0.866764	0.133236	0.733529
1	0.895535	0.104465	0.791070
2	0.884615	0.115385	0.769231
3	0.896471	0.103529	0.792941
4	0.548780	0.451220	0.097561
5	0.400000	0.600000	-0.200000

churn	0	1	Difference
customer_service_calls			
6	0.363636	0.636364	-0.272727
7	0.444444	0.555556	-0.111111
8	0.500000	0.500000	0.000000
9	0.000000	1.000000	-1.000000
All	0.854579	0.145421	0.709157

In [45]:

```
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline

var_cross_tab = pd.crosstab(index=df_cleaned.loc[:, 'customer_service_calls'],
                            columns=df_cleaned.loc[:, 'churn'], normalize='index')

bar_colors = ['steelblue', 'darkorange']

ax = var_cross_tab.plot(kind='bar', legend=True, figsize=(16, 8), title='Customers w

ax.set_facecolor('lightgray')
plt.xlabel('Customer Service Calls')
plt.ylabel('Churn')
plt.show()
```



Customers who should be contacted.

In [46]:

```
print('Total no. of people who contacted more than 4 times:',df_cleaned[df_cleaned['
df_cleaned[(df_cleaned['customer_service_calls']>=4) & (df_cleaned['churn'] == 0)]
```

```
Total no. of people who contacted more than 4 times: 264
```

Out[46]:

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
7	107	no	no	0	189.7

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
58	104	no	no	0	280.4
86	101	no	no	0	153.8
156	64	no	yes	40	210.0
163	67	no	no	0	260.4
...	...	...	...	...	...
3253	105	no	no	0	162.3
3268	87	no	no	0	256.2
3272	106	no	yes	32	165.9
3290	217	no	no	0	176.4
3316	96	no	no	0	260.4

129 rows × 19 columns

Congratulations: You have identified an integer data series that you can use to identify customers who are more likely to leave. Next you need to identify a floating point data series that you can use to predict which customers are likely to leave.

4c) Which **floating point** data series could you use to help with this selection? Determine the threshold for this by using **logistic regression**. Which customers should be contacted based on this data series?

For this step you will need to use skills you learned in the following lessons:

- *Visualizing Data Distributions using Histograms* *(Chapter 2)*
- *Visualizing Correlations with Scatter Plots* *(Chapter 2)*
- *Box Plots* *(Chapter 3)*
- *Project: Share Prices Linear Regression Modeling* *(Chapter 6)*
- *Making Predictions Using Logistic Regression* *(Chapter 3)*
- *Boolean Masking* *(Chapter 4)*
- *Measures of Central Tendency* *(Chapter 3)*

Select columns of float64 data type

In [47]:

```
float64_columns = df_cleaned.select_dtypes(include=['float64']).columns.tolist()
print(float64_columns)
```

```
['total_day_minutes', 'total_day_charge', 'total_eve_minutes', 'total_eve_charge',
'total_night_minutes', 'total_night_charge', 'local_area_code', 'phone_num']
```

Here local_area_code and phone_num are more like categorical columns,

Hence these two are not suitable for further analysis as floating data deries.

In [48]:

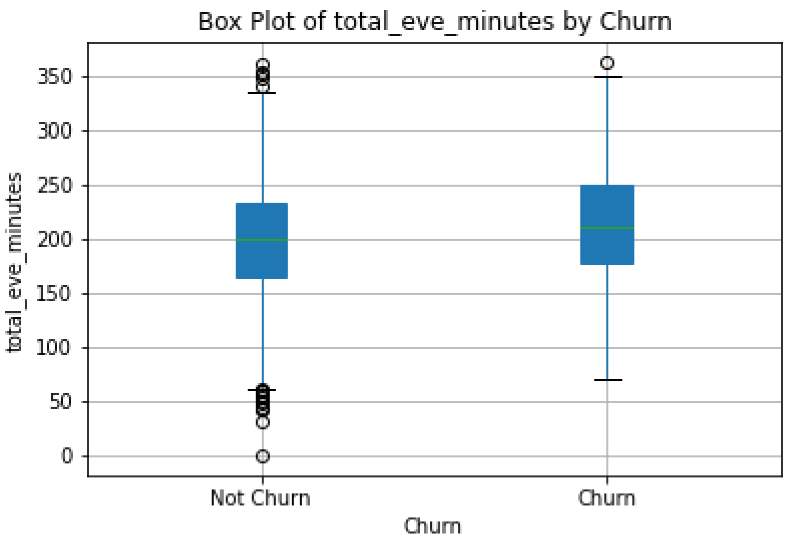
```
flot_columns=['total_eve_minutes', 'total_eve_charge', 'total_night_minutes',
              'total_night_charge', 'total_day_minutes', 'total_day_charge']
```

Box plot for each float column

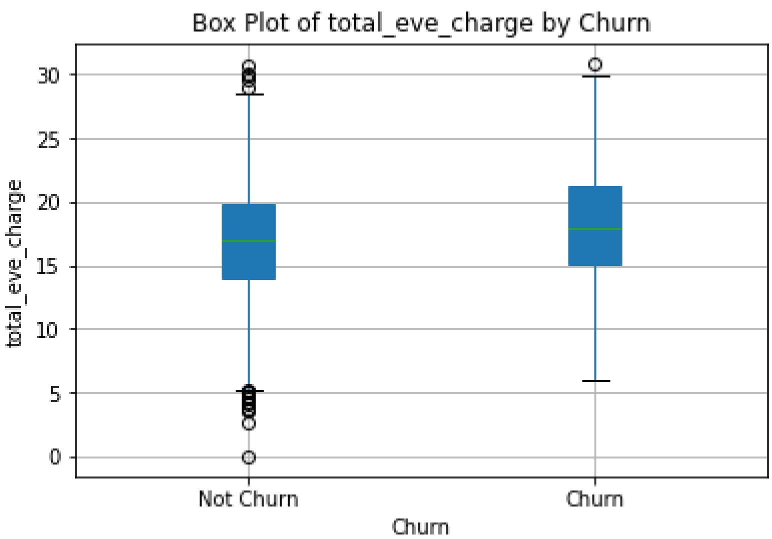
In [49]:

```
# Check ..._charge and ..._minutes columns;  Is there any kind of linear depency?
#import matplotlib.pyplot as plt
for i in flot_columns:
    plt.figure()  # Create a new figure for each box plot
    df_cleaned.boxplot(column=i, by='churn', grid=True, patch_artist=True)
    plt.title('Box Plot of {} by Churn'.format(i))  # Add a title to the plot using
    plt.xlabel('Churn')  # Add an x-axis label
    plt.ylabel(i)  # Add a y-axis label (column name)
    plt.suptitle('')  # Remove the default title generated by pandas boxplot
    plt.xticks([1, 2], ['Not Churn', 'Churn'])  # Set custom x-axis labels
    colors = ['blue', '']  # Set custom colors for the boxes (in the order of 'Not C
    for box, color in zip(ax.artists, colors):
        box.set_facecolor(color)  # Set the box color

    plt.xticks([1, 2], ['Not Churn', 'Churn'])  # Set custom x-axis labels
    plt.show()  # Show the plot
    plt.show()
```

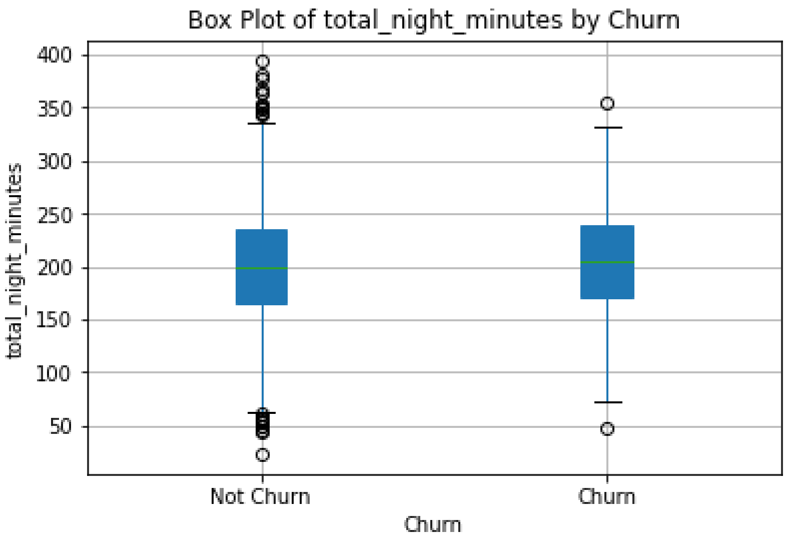
<Figure size 432x288 with 0 Axes>



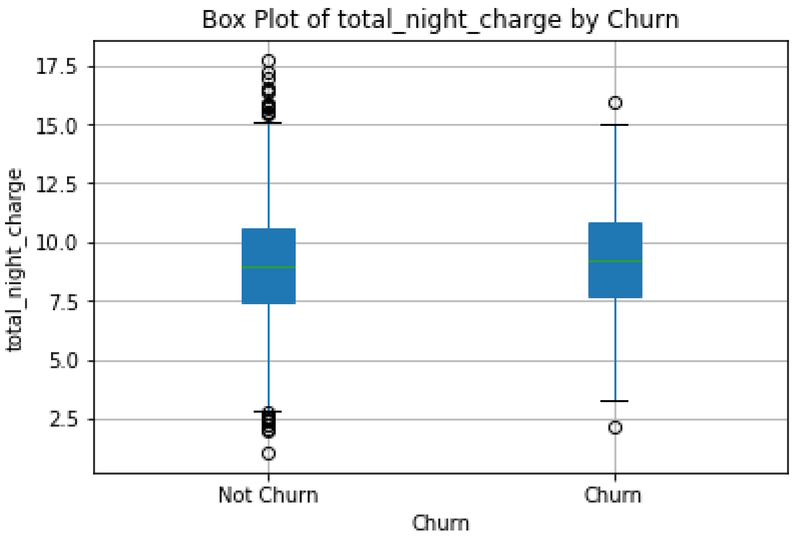
<Figure size 432x288 with 0 Axes>



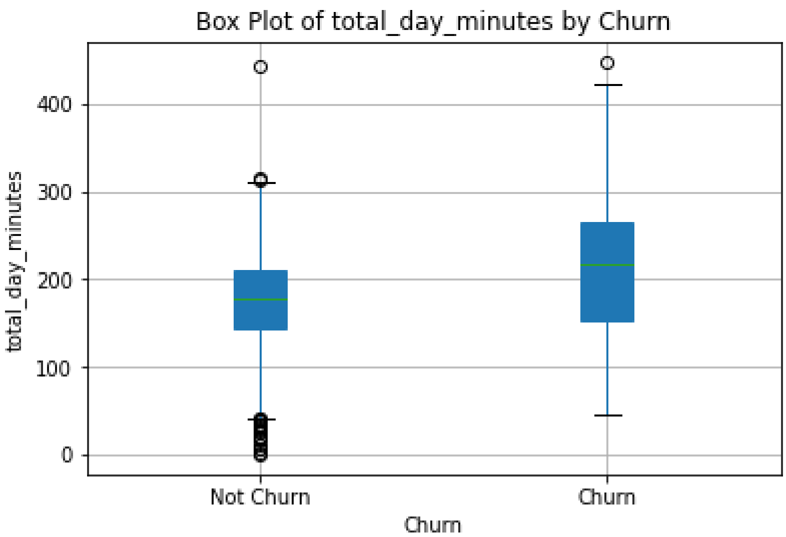
<Figure size 432x288 with 0 Axes>



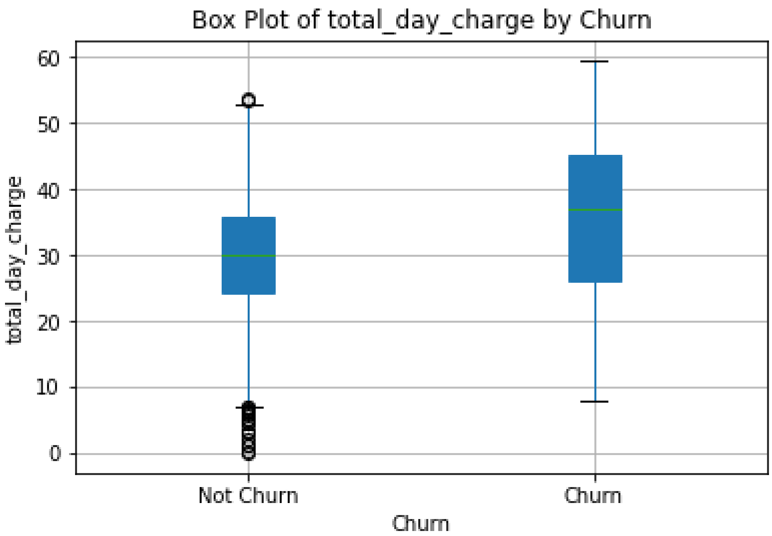
<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



*From box plots it is clear that people who pay more during day have high churn rate

In [50]:

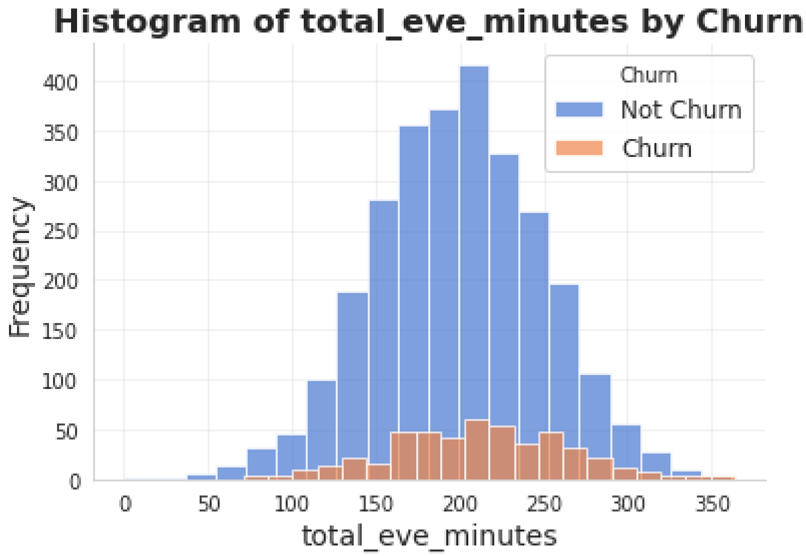
```
columns_to_check = [
    'total_eve_minutes',
    'total_eve_charge',
    'total_night_minutes',
    'total_night_charge',
    'total_day_minutes',
    'total_day_charge'
]
```

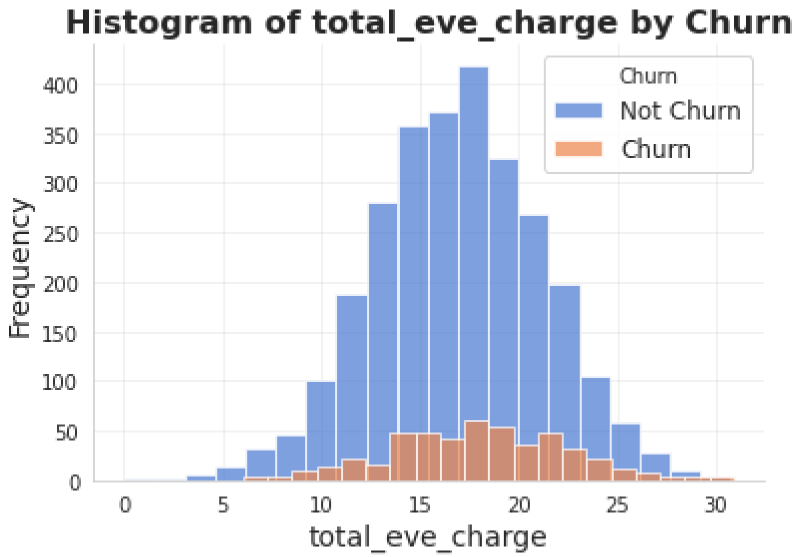
Created separated histograms for each relevant column

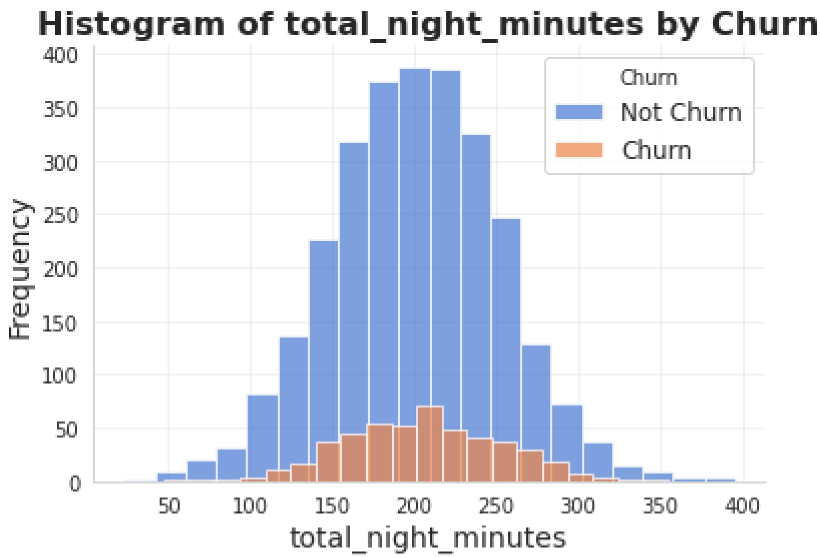
In [51]:

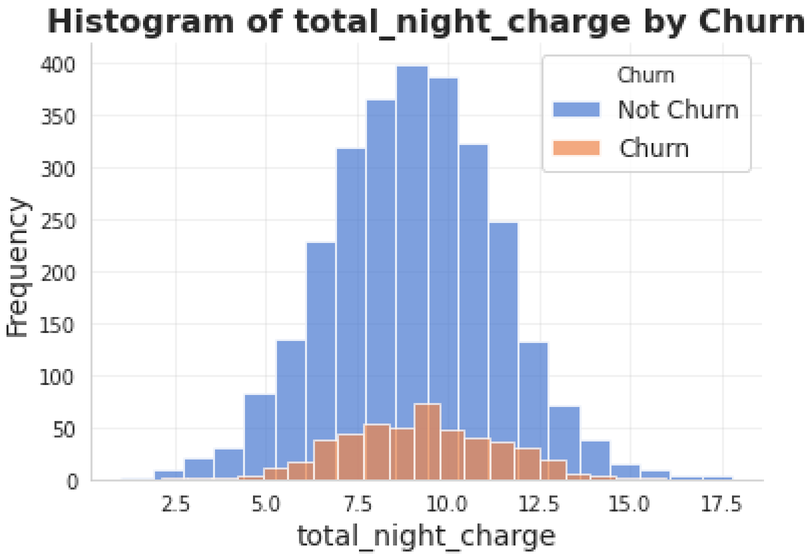
```
import matplotlib.pyplot as plt
import seaborn as sns

custom_palette = sns.color_palette("muted")
sns.set_palette(custom_palette)
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))

for i in columns_to_check:
    plt.figure()
    df_cleaned.groupby('churn')[i].plot(kind='hist', alpha=0.7, legend=True, bins=20
    plt.title('Histogram of {} by Churn'.format(i), fontsize=16, fontweight='bold')
    plt.xlabel(i, fontsize=14)
    plt.ylabel('Frequency', fontsize=14)
    plt.legend(title='Churn', labels=['Not Churn', 'Churn'], fontsize=12)
    plt.grid(True, alpha=0.3)
    sns.despine()
    plt.show()
```

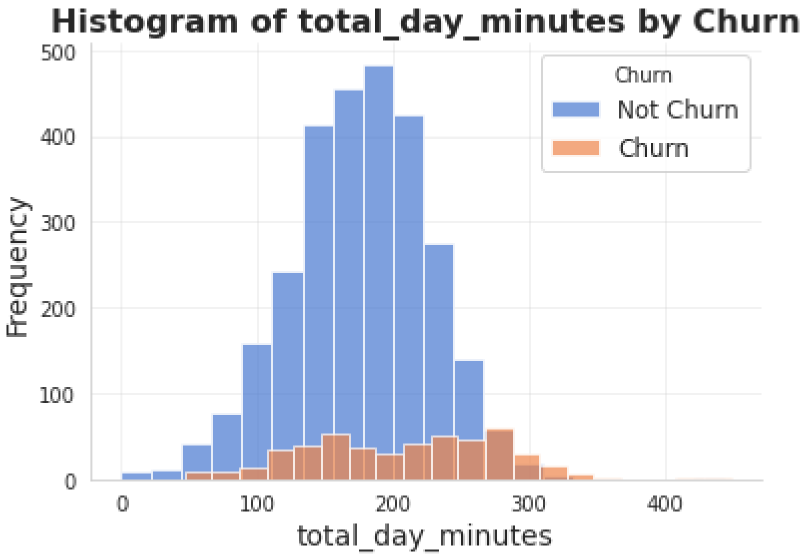
```
<Figure size 720x432 with 0 Axes>
```

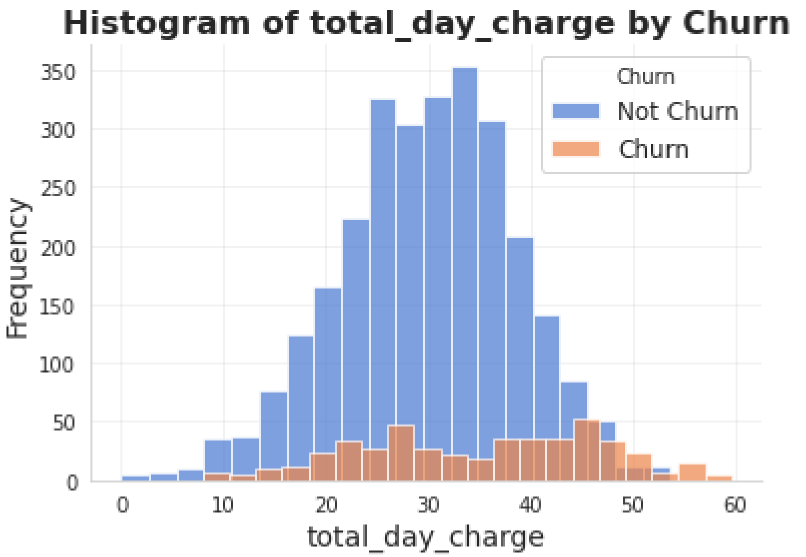

Histogram of total_eve_minutes by Churn
Frequency
total_eve_minutes
0
50
100
150
200
250
300
350
400
Churn
Not Churn
Churn


Histogram of total_eve_charge by Churn
Frequency
total_eve_charge
0
5
10
15
20
25
30
50
100
150
200
250
300
350
400
Churn
Not Churn
Churn


Histogram of total_night_minutes by Churn
Frequency
total_night_minutes
0
50
100
150
200
250
300
350
400
Churn
Not Churn
Churn







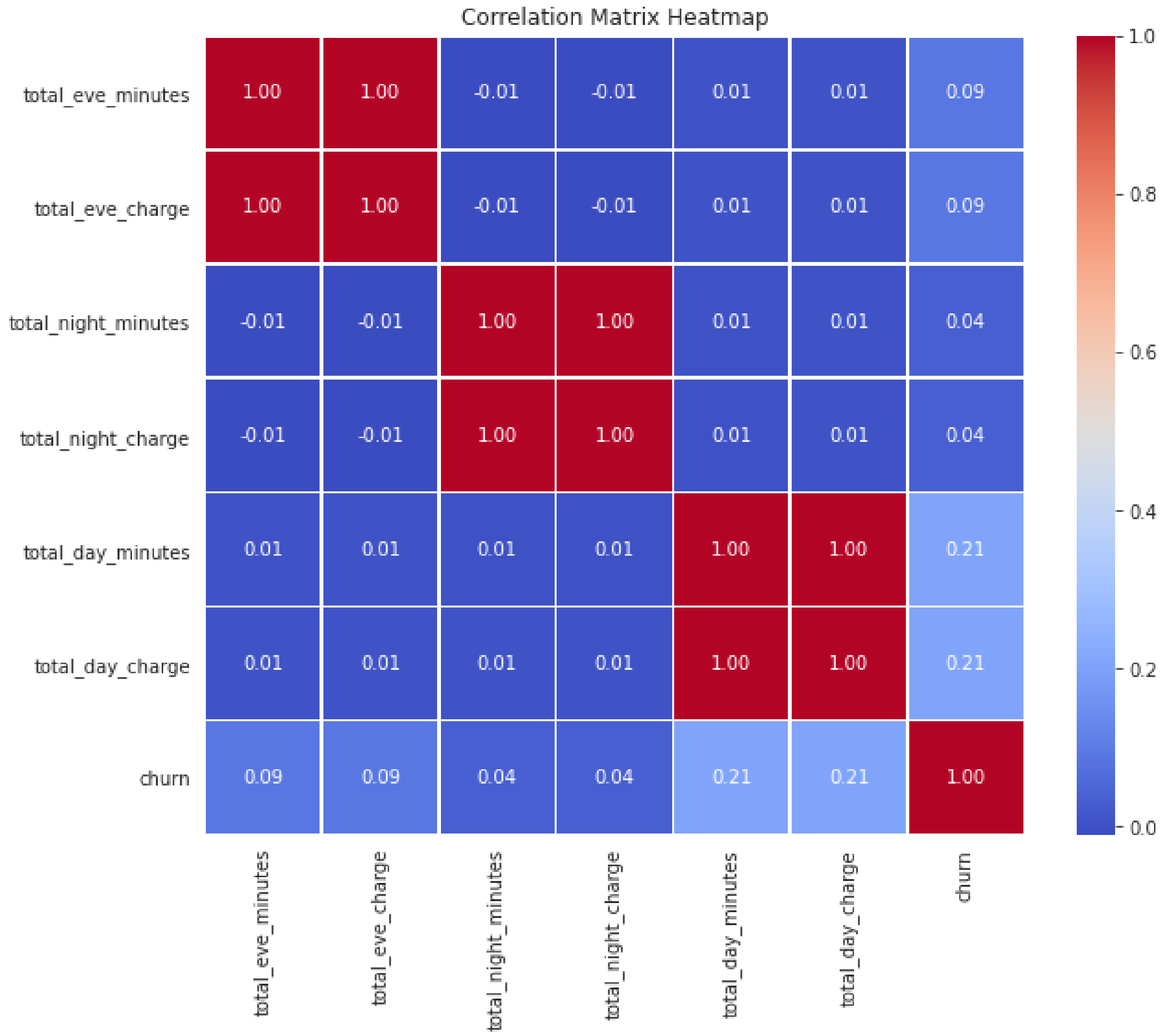
*From Histogram I have have observed that the columns 'total_day_minutes' and 'total_day_charge' are on high risk of churn

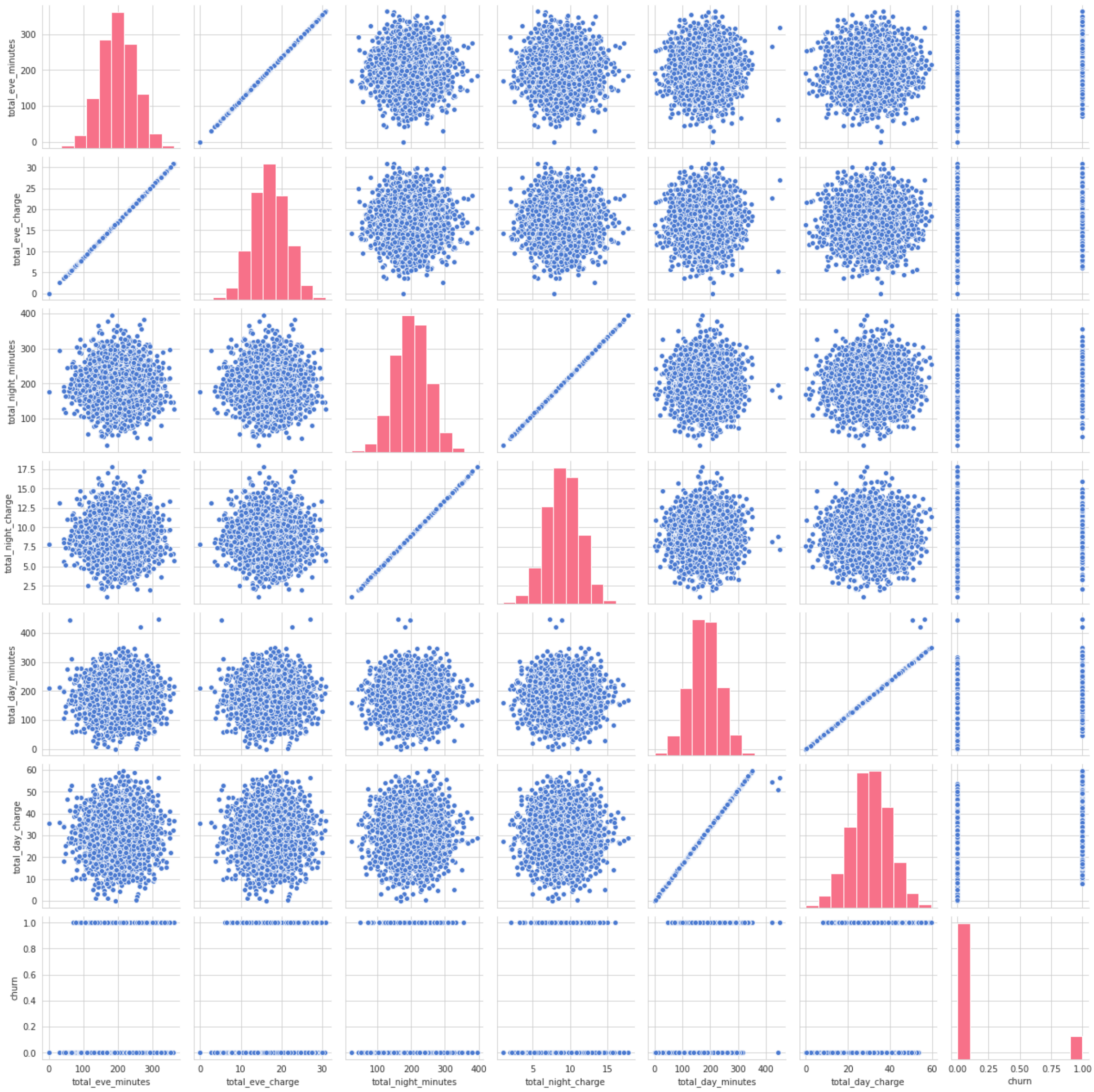
HeatMap & Pairplot

In [67]:

```
import seaborn as sns
import matplotlib.pyplot as plt
columns_to_check = [
    'total_eve_minutes',
```

```
    'total_eve_charge',
    'total_night_minutes',
    'total_night_charge',
    'total_day_minutes',
    'total_day_charge', 'churn'
]


correlation_matrix = df_cleaned[columns_to_check].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0
plt.title('Correlation Matrix Heatmap')
plt.show()

hist_palette = 'husl'
sns.pairplot(df_cleaned[columns_to_check], diag_kws={'color': sns.color_palette(hist
plt.show()
```





From the overall visualisation I determined that 'total_day_minutes' and 'total_day_charge' are for the further analysis.

For logistic regression , need to change data type of 'churn'

In [53]:
```python
df_cleaned['churn'] = df_cleaned['churn'].astype(int)
```

```
/tmp/ipykernel_3371/885040238.py:1: 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/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['churn'] = df_cleaned['churn'].astype(int)
```

In [54]:
```python
df_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3287 entries, 0 to 3332
Data columns (total 19 columns):
 #   Column                    Non-Null Count  Dtype
```

```
---  ------                 --------------  -----
 0   account_length         3287 non-null   int64
 1   international_plan     3287 non-null   category
 2   voice_mail_plan        3287 non-null   category
 3   number_vmail_messages  3287 non-null   int64
 4   total_day_minutes      3287 non-null   float64
 5   total_day_calls        3287 non-null   int64
 6   total_day_charge       3287 non-null   float64
 7   total_eve_minutes      3287 non-null   float64
 8   total_eve_calls        3287 non-null   int64
 9   total_eve_charge       3287 non-null   float64
 10  total_night_minutes    3287 non-null   float64
 11  total_night_calls      3287 non-null   int64
 12  total_night_charge     3287 non-null   float64
 13  customer_service_calls 3287 non-null   int64
 14  churn                  3287 non-null   int64
 15  local_area_code        3287 non-null   float64
 16  phone_num              3287 non-null   float64
 17  city                   3287 non-null   category
 18  area_code              3287 non-null   int64
dtypes: category(3), float64(8), int64(8)
memory usage: 606.8 KB
```

To determine the threshold, used logistic regression.

In [55]:

```python
# Logistic regression
import statsmodels.formula.api as smf
model = smf.logit(formula='churn ~ total_day_minutes',
                  data=df_cleaned)


# Define a Logit model and fit it
result = model.fit()



# Check model summary
result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.392458
         Iterations 6
```

Out[55]:

Logit Regression Results

Dep. Variable:	churn	**No. Observations:**	3287
Model:	Logit	**Df Residuals:**	3285
Method:	MLE	**Df Model:**	1
Date:	Wed, 09 Aug 2023	**Pseudo R-squ.:**	0.05360
Time:	13:00:57	**Log-Likelihood:**	-1290.0
converged:	True	**LL-Null:**	-1363.1
Covariance Type:	nonrobust	**LLR p-value:**	1.225e-33

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	-3.9262	0.202	-19.430	0.000	-4.322	-3.530
total_day_minutes	0.0113	0.001	11.613	0.000	0.009	0.013

In [56]:

```python
# Logistic regression


import statsmodels.formula.api as smf
model = smf.logit(formula='churn ~ total_day_charge',
                  data=df_cleaned)


# Define a logit model and fit it
results = model.fit()



# Check model summary
results.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.392483
         Iterations 6
```

Out[56]:

Logit Regression Results

Dep. Variable:	churn	**No. Observations:**	3287
Model:	Logit	**Df Residuals:**	3285
Method:	MLE	**Df Model:**	1
Date:	Wed, 09 Aug 2023	**Pseudo R-squ.:**	0.05354
Time:	13:00:57	**Log-Likelihood:**	-1290.1
converged:	True	**LL-Null:**	-1363.1
Covariance Type:	nonrobust	**LLR p-value:**	1.333e-33

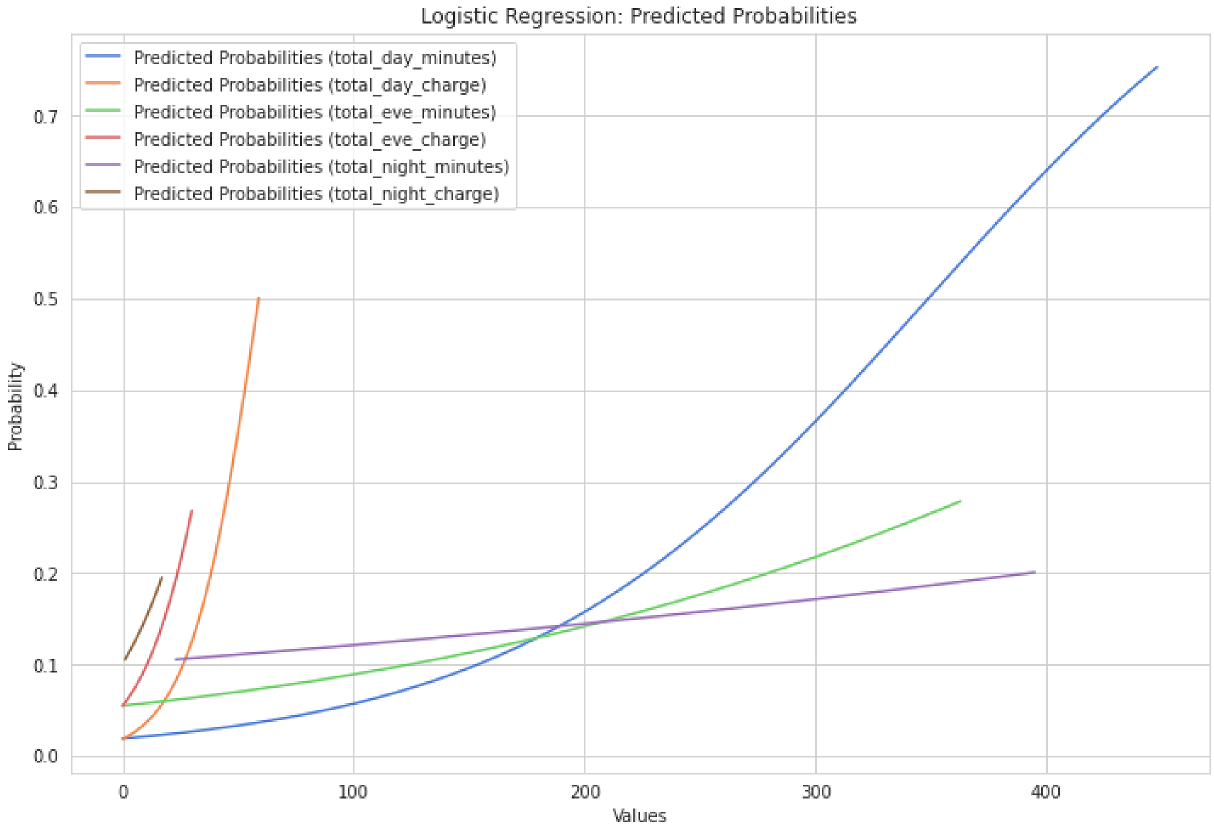
	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-3.9541	0.204	-19.342	0.000	-4.355	-3.553
total_day_charge	0.0671	0.006	11.627	0.000	0.056	0.078

In [57]:

```python
#List of columns to apply logistic regression on
import math
columns_to_analyze = ['total_day_minutes', 'total_day_charge', 'total_eve_minutes',
                      'total_eve_charge', 'total_night_minutes', 'total_night_charge
                      ]

# Define a function to perform logistic regression and plot the predicted probabilit
def logistic_regression_and_plot(column_name):
    # Create the formula for the logistic regression
    formula = 'churn ~ {}'.format(column_name)

    # Create the logistic regression model
    model = smf.logit(formula=formula, data=df_cleaned)

    # Fit the model
    result = model.fit()

    # Get the range of values for the column to predict probabilities
    min_value = df_cleaned[column_name].min()
    max_value = df_cleaned[column_name].max()
    X = pd.Series(range(int(min_value), int(max_value) + 1))

    # Get the intercept and slope from the model results
```

```python
    intercept = result.params['Intercept']
    slope = result.params[column_name]

    # Calculate the predicted probabilities
    p_y = 1 / (1 + math.e**-(intercept + (slope * X)))

    # Plot the predicted probabilities
    plt.plot(X, p_y, label='Predicted Probabilities ({})'.format(column_name))

# Create a plot for each column
plt.figure(figsize=(12, 8))
for column in columns_to_analyze:
    logistic_regression_and_plot(column)

# Add labels and legend to the plot
plt.xlabel('Values')
plt.ylabel('Probability')
plt.title('Logistic Regression: Predicted Probabilities')
plt.legend()

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

```
Optimization terminated successfully.
         Current function value: 0.392458
         Iterations 6
Optimization terminated successfully.
         Current function value: 0.392483
         Iterations 6
Optimization terminated successfully.
         Current function value: 0.410424
         Iterations 6
Optimization terminated successfully.
         Current function value: 0.410425
         Iterations 6
Optimization terminated successfully.
         Current function value: 0.414029
         Iterations 6
Optimization terminated successfully.
         Current function value: 0.414028
         Iterations 6
```

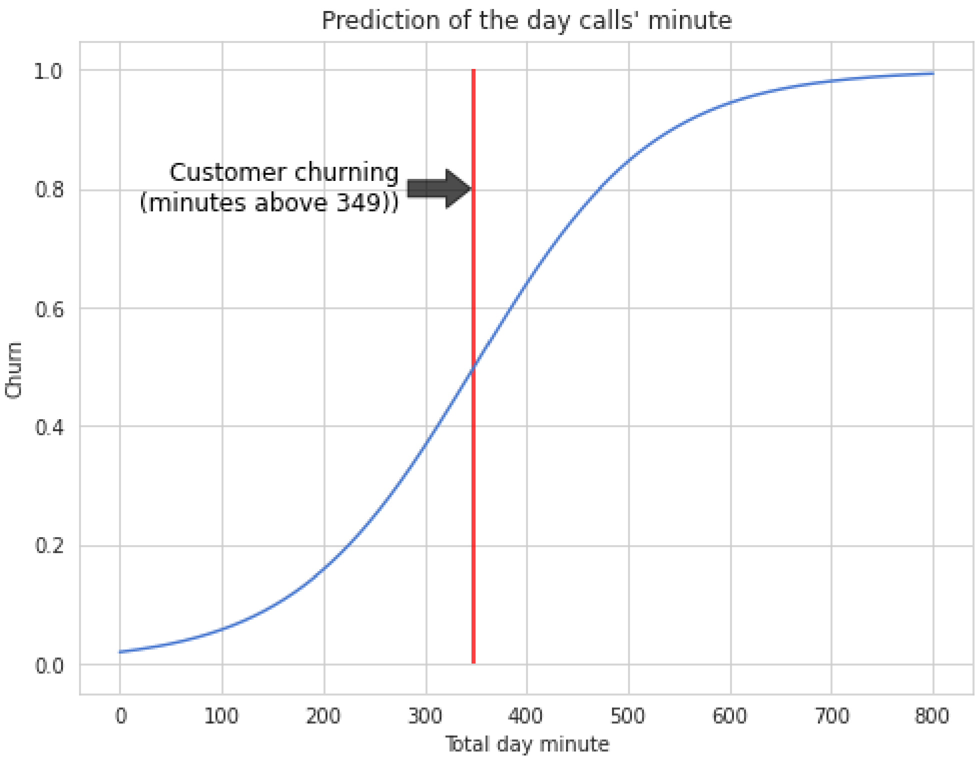


In [58]:

```
import math
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have already defined the 'result' variable from your analysis

b = 'total_day_minutes'
X = pd.Series(range(801))
intercept = result.params['Intercept']
slope = result.params[b]
p_y = 1 / (1 + math.e**-(intercept + (slope * X)))
X_df = pd.DataFrame(X)
X_df.columns = [b]
p_y = result.predict(X_df)

# Set the figsize to (8, 6)
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(p_y)
ax.vlines(x=p_y[p_y >= 0.5].index[0], ymin=0, ymax=1, colors='r')

# You can find the x-value at the point where the y value is 0.5 like this:
print('x-value at the point where the y value is 0.5 like this:', p_y[p_y >= 0.5].in


# Annotation
ax.annotate('Customer churning\n(minutes above 349))',
            xy=(349, 0.8),
            xytext=(275, 0.8),
            ha='right', va='center',
            arrowprops=dict(facecolor='black', shrink=0.05, width=8, headwidth=19, a
            color='black', fontsize=12)

ax.set(title="Prediction of the day calls' minute",
       xlabel="Total day minute",
ylabel="Churn")
```

```
plt.show()
```

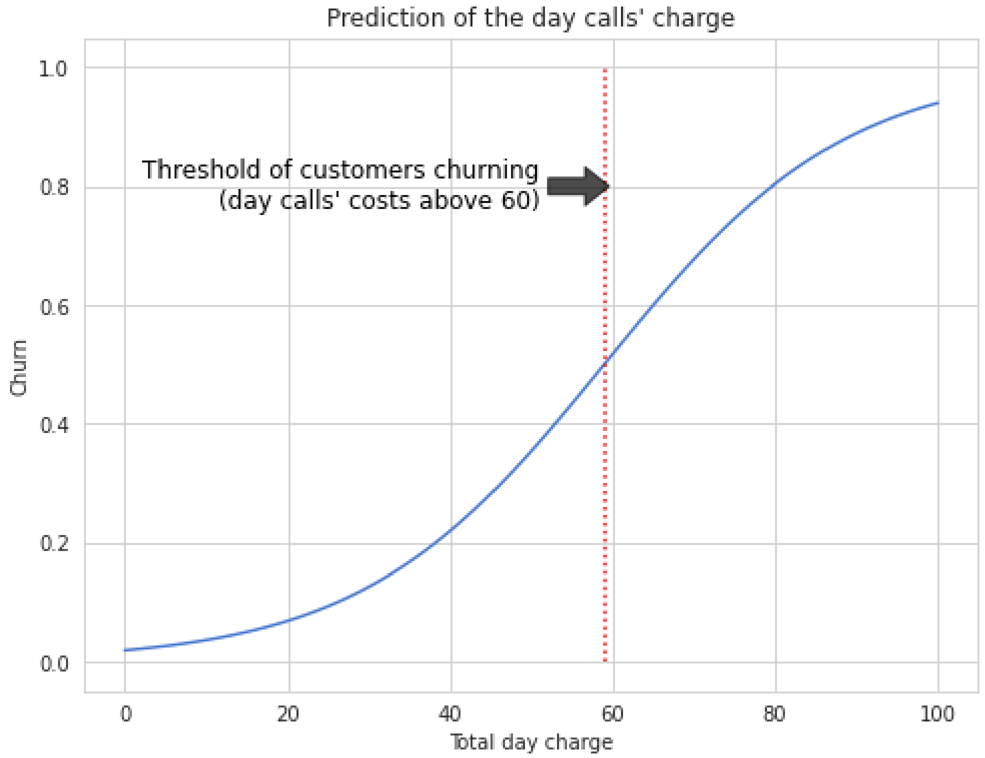
```
x-value at the point where the y value is 0.5 like this: 349
```



In [59]:

```
import math
import pandas as pd
import matplotlib.pyplot as plt

X = pd.Series(range(101))
max_total_day_charge = 60

# Assuming that you have a result object from a logistic regression model
intercept = results.params['Intercept']
slope = results.params['total_day_charge']

p_y = 1 / (1 + math.e ** -(intercept + (slope * X)))
# Alternatively, you can use result.predict(X_df)

# Finding the x-value for y being 0.5 (50%)
threshold_point = p_y[p_y >= 0.5].index[0]

fig_logistic_regression, ax = plt.subplots(figsize=[8, 6])

# Plotting the logistic regression curve
ax.plot(X, p_y)

# Plotting the line of abandonment
ax.plot([threshold_point, threshold_point], [0, 1], linestyle=':', color='r')

# Annotation
ax.annotate('Threshold of customers churning\n(day calls\' costs above 60)',
            xy=(60, 0.8),
            xytext=(51, 0.8),
            ha='right', va='center',
            arrowprops=dict(facecolor='black', shrink=0.05, width=8, headwidth=19, a
            color='black', fontsize=12)

ax.set(title="Prediction of the day calls' charge",
```

```
        xlabel="Total day charge",
ylabel="Churn")

plt.show()
```



Average charge of customers who haven't churned and churned based on 'total_day_charge'

In [60]:

```
threshold_of_customer_reach_out = df_cleaned.groupby('churn')['total_day_charge'].me
threshold_of_customer_reach_out
```

Out[60]:

```
churn
0    29.800968
1    35.267197
Name: total_day_charge, dtype: float64
```

*The output shows the result of calculation, indicating that the average day charge for customers who have not churned (churn=0) is approx USD29.80, while for customers who have churned (churn=1), the average day charge is about USD35.27

Number of customers should be contacted based on 'total_day_charge'

In [61]:

```
mask_logistic_customers = (df_cleaned.loc[:, 'total_day_charge'] >= threshold_of_cus

df_customers_to_be_contacted = df_cleaned.loc[mask_logistic_customers, :]
print('Customers likely to leave soon: {}'.format(df_customers_to_be_contacted.shape
df_customers_to_be_contacted
```

```
Customers likely to leave soon: 764
```

Out[61]:

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
4	120	no	yes	28	215.8

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
5	108	no	no	0	210.7
18	41	no	no	0	209.9
19	85	no	no	0	235.8
21	117	no	yes	25	216.0
...	...	...	...	...	...
3310	131	no	no	0	263.4
3312	118	no	yes	36	294.9
3316	96	no	no	0	260.4
3322	73	no	no	0	240.3
3323	111	no	no	0	224.9

764 rows × 19 columns

Average minutes of customers who haven't churned and churned based on 'total_day_minutes'

In [62]:

```
threshold_of_customer_reach_out1 = df_cleaned.groupby('churn')['total_day_minutes'].
threshold_of_customer_reach_out1
```

Out[62]:

```
churn
0    175.347775
1    207.903556
Name: total_day_minutes, dtype: float64
```

*The output shows the result of calculation, For customers who did not churn, the average number of minutes they spend on calls during the day is approximately is 175.35 minutes and The output shows the result of calculation, For customers who did churn, the average number of minutes they spend on calls during the day is approximately is 207.90 minutes

Number of customers should be contacted based on 'total_day_minutes'

In [63]:

```
mask_logistic_customers1 = (df_cleaned.loc[:, 'total_day_minutes'] >= threshold_of_c

df_customers_to_be_contacted1 = df_cleaned.loc[mask_logistic_customers1, :]
print('Customers likely to leave soon: {}'.format(df_customers_to_be_contacted1.shap
df_customers_to_be_contacted1
```

```
Customers likely to leave soon: 748
```

Out[63]:

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
4	120	no	yes	28	215.8
5	108	no	no	0	210.7
18	41	no	no	0	209.9

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
19	85	no	no	0	235.8
21	117	no	yes	25	216.0
...	...	...	...	...	...
3310	131	no	no	0	263.4
3312	118	no	yes	36	294.9
3316	96	no	no	0	260.4
3322	73	no	no	0	240.3
3323	111	no	no	0	224.9

748 rows × 19 columns

*** From the overall analysis I came up with the conclusion that total_day_charge column seems more targeted to determine the customers who should be contacted**

Congratulations: You have used logistic regression to predict which kinds of customers are likely to leave. Now you have finished analyzing the data and can focus on visualizing the results.

5) Visualizing the cities and other selected data series.

You will find examples from the following lessons helpful for this step:

- *Exploring Categories (Chapter 2)*
- *Visualizing Categories (Chapter 2)*
- *Combining Visualizations with matplotlib (Chapter 2)*
- *Customizing Visualizations with matplotlib (Chapter 2)*
- *Box Plots (Chapter 3)*

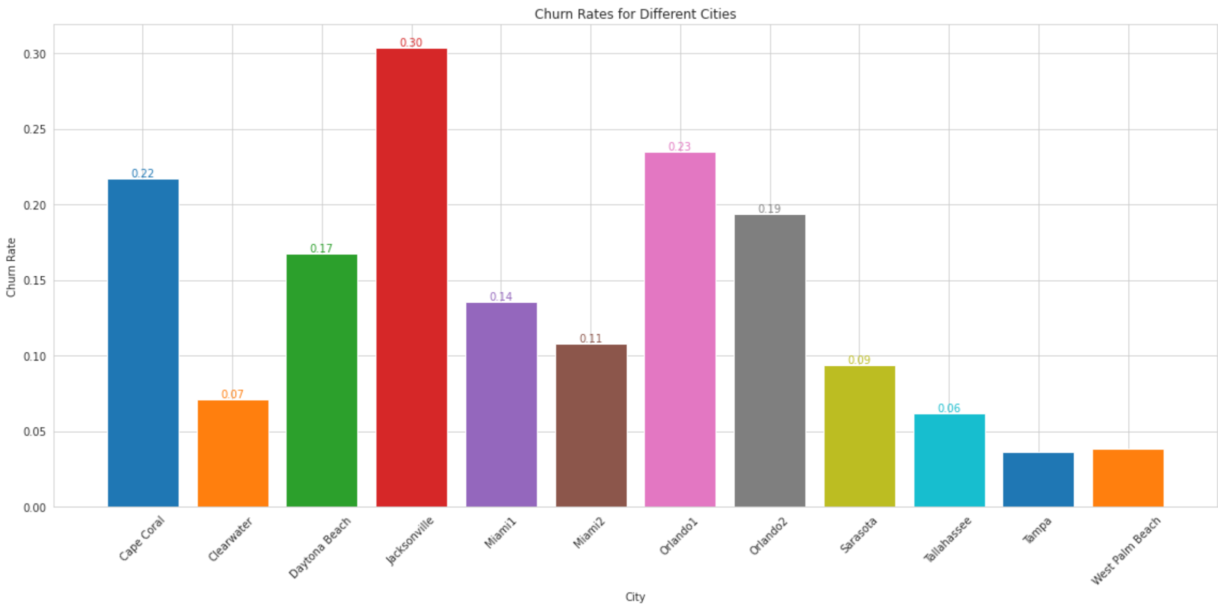
*** Here I plotted all the important and necessary visulizations which helped me to determine , the customers who should be contacted and likely to churn**

Bar plot of Cities VS Churn

In [64]:

```
# Column chart of urban areas
%matplotlib inline
# Group the data by 'City' and calculate the churn rates
churn_rates = df_cleaned.groupby('city')['churn'].mean()

# Define predefined colors for the bar chart
colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown

# Create a bar chart
plt.figure(figsize=(16, 8))
plt.bar(churn_rates.index, churn_rates.values, color=colors)

# Annotate the bars with text (churn rates) in the same colors as the bars
for city_name, rate, color in zip(churn_rates.index, churn_rates.values, colors):
    plt.text(city_name, rate, '{:.2f}'.format(rate), ha='center', va='bottom', color
```

```
# Add labels and title
plt.xlabel('City')
plt.ylabel('Churn Rate')
plt.title('Churn Rates for Different Cities')

# Rotate the x-axis labels for better visibility
plt.xticks(rotation=45)

# Show the plot
plt.tight_layout()
plt.show()
```

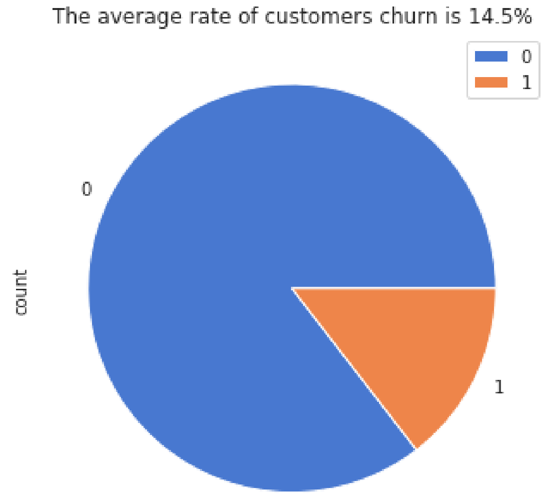


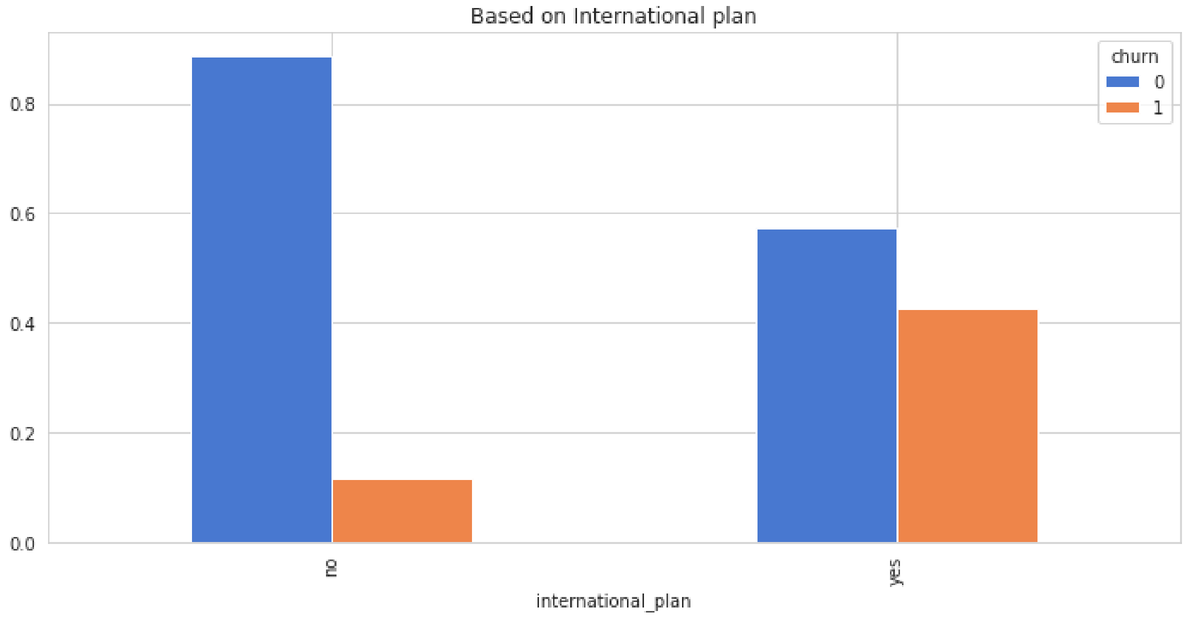
Visulization of other data series

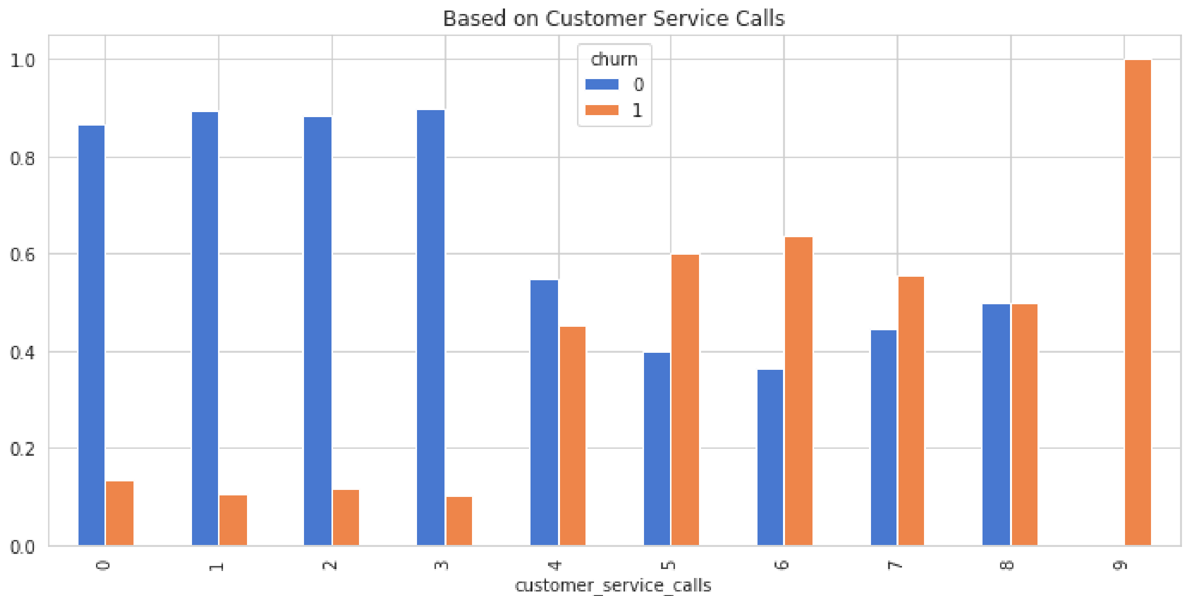
In [65]:

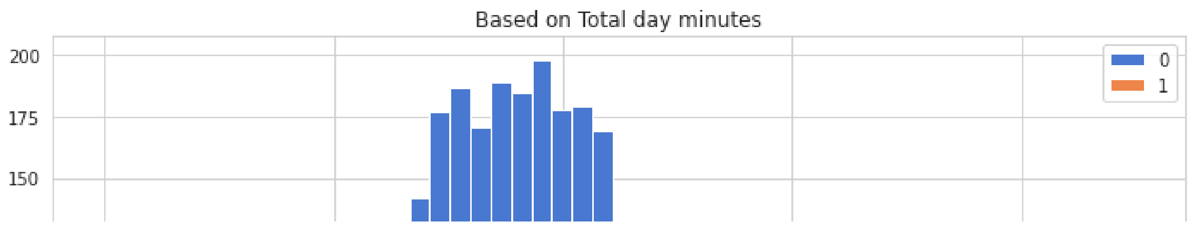
```
# Solution:
%matplotlib inline


fig_grouped, ax = plt.subplots(nrows=4,
                       ncols=1,
                       figsize=[10, 20])# Din A4
####################################################################################

var_cross_tab = pd.crosstab(index=df_cleaned.loc[:, 'churn'], columns='count')
var_cross_tab.plot(kind='pie',
                   y='count',
                   ax=ax[0],
                  legend=True,
                   title='The average rate of customers churn is 14.5%')

####################################################################################

var_cross_tab = pd.crosstab(index=df_cleaned.loc[:, 'international_plan'],
                            columns=df_cleaned.loc[:, 'churn'],
                           normalize='index')
var_cross_tab.plot(kind='bar',title='Based on International plan',
                   legend=True,
                  ax=ax[1])


####################################################################################
```

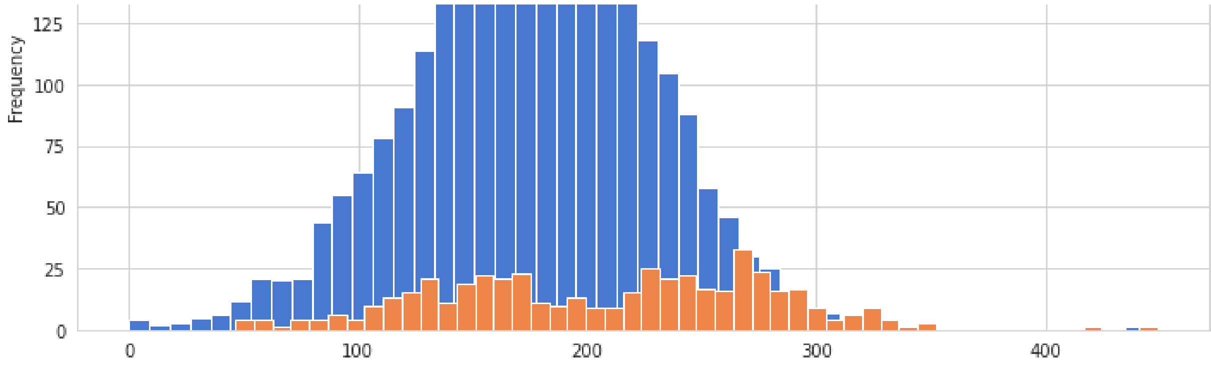
```
df_cleaned.groupby('churn')['total_day_minutes'].plot(kind='hist',title='Based on To
                                                                      bins = 50,
                                                                      legend=True,
                                                                     ax=ax[3])

####################################################################################

var_cross_tab = pd.crosstab(index=df_cleaned.loc[:, 'customer_service_calls'],
                            columns=df_cleaned.loc[:, 'churn'],
                           normalize='index')

var_cross_tab.plot(kind='bar',title='Based on Customer Service Calls',
                   legend=True,
                  ax=ax[2])

plt.tight_layout()
```


The average rate of customers churn is 14.5%
0
1
count
0
1


Based on International plan
churn
0
1
0.0
0.2
0.4
0.6
0.8
no
yes
international_plan


Based on Customer Service Calls
churn
0
1
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
8
9
customer_service_calls

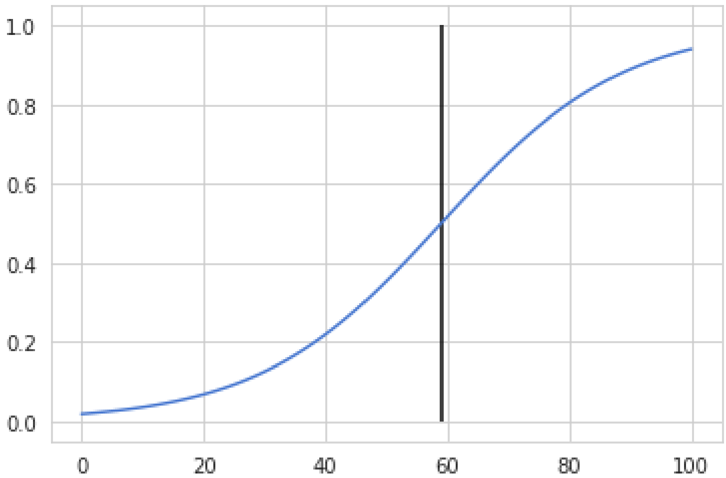

Based on Total day minutes
0
1
150
175
200



In [66]:
```
# Line plot for findings in logistic regression

print('x-value at the point where the y value is 0.5 like this:', p_y[p_y >= 0.5].in
%matplotlib inline
fig, ax= plt.subplots()
ax.plot(p_y)
ax.vlines(x=p_y[p_y>=0.5].index[0], ymin=0, ymax=1)
```

```
x-value at the point where the y value is 0.5 like this: 59
```

Out[66]:
```
<matplotlib.collections.LineCollection at 0x7fe65aafef40>
```



In [ ]:

6) Formulating a recommendation

We have two marketing campaigns: In the first, four cities with the highest customer churn are to be targeted with poster campaigns. In the second, customers who are ready to churn are to be approached individually in order to retain them as customers using special offers.

What recommendations would you give the telecommunications provider?

Write your observations, notes and recommendations from the data set in bullet points as comments with `#` in the following code cell.

Observations:

1. The data analysis reveals that four cities in USA have the highest churn rates - Jacksonville, Orlando1,Cape Coral and Orlando2.
2. By analyzing individual customer data, we identified two significant factors contributing to churn - those who have taken an international plan and those who have made more than three calls to customer service.
3. The data indicates that customers with high total day minutes are more likely to churn.
4. Logistic regression has been used to estimate the probability of churn for individual customers.

Recommendation:

1. For the first marketing campaign, we suggest focusing on these four cities. We will design and display big posters in key locations to promote Teleconfia's services and encourage customers to stay. The posters should highlight the unique benefits of our phone services and emphasize the value they receive by remaining with Teleconfia.
2. For the second marketing campaign, we propose a personalized approach to retain at-risk customers. We will prioritize contacting customers who meet either of the following criteria: a. Customers who have an international plan: We will reach out to these customers with offers, discounts, and features that cater to their specific international calling needs. b. We'll help customers with frequent calls, resolving issues promptly for a better experience.
3. We will reach out to customers with exceptionally high total day minutes, as they might be experiencing service-related issues due to heavy usage. Personalized communication will help us understand their needs better and offer suitable plans or features to cater to their usage patterns effectively.
4. We will leverage the logistic regression model to predict customers who are more likely to churn in the future. This proactive approach will enable us to reach out to potential churners early and offer them exclusive deals and incentives to persuade them to continue using Teleconfia's services.

Congratulations: If you have made it this far then you have completed the last exercise in this training course! It wasn't always easy, but you have learned a lot. You can be proud of yourself for sticking with it to the end. Now you can consider yourself a data analyst.

Remember:

- Python can do all kinds of things. You will never run out of things to learn, so you should continue learning more skills as you progress. You will learn something new with each new data set.
- You have what it takes to be a data analyst! Python is now one of the tools you have at your disposal.

Do you have any questions about this exercise? Look in the forum to see if they have already been discussed.

Found a mistake? Contact Support at support@stackfuel.com.
