Final Project Submission

Name: Breden Mugambi
Student Pace: Full time
Instructor Name: Nikita Njoroge

Business Understanding

SyriaTel faces a critical issue: customer churn. Subscribers discontinuing service leads to lost revenue and hinders the company's growth. To address this challenge, a comprehensive customer churn prediction model is proposed. This model will analyze customer data to identify those at high risk of churning. By recognizing these at-risk customers through predictive analytics, SyriaTel can take proactive measures to address their concerns and needs.

This data-driven approach aims not only to reduce churn but also to enhance customer lifetime value, increase revenue, and improve overall customer satisfaction. Key stakeholders—including the marketing team, customer service, and executive management—will be integral in implementing and benefiting from these insights

Problem Statement

Syriatel wants to predict whether a customer stops doing business with them in the future according to the data presented.

Main objective

To develop a model which predicts if a customer of Syriatel will stop doing business with them

Specific objectives

- To identify which data to be used during modelling
- To undertand and prepare the data for analysis
- To perform data analysis on the prepared data
- To develop a binary classification model that predicts if a customer stops business
- To evaluate the model findings and recommend actions to be done

Data Understanding

In order to understand the data, we need to fiest see what is in it and the type of variables are there. We first need to import all relevant libraries to access the data. I will use the pandas and numpy libraries.

In [1]:
```
#This cell contains all libraries to be used in the entire project
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import OneHotEncoder

import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
#this function is used to access the data

# Define the DataFrame in the global scope
df = pd.DataFrame()

#function that loads data and gets the info about the data.
def load_and_get_info(file_path):
    # Load data
    global df
    df = pd.read_csv(file_path)
    # Display the first few rows of the DataFrame
    df_head = df.head()
    # Get information about the DataFrame
    df_info = df.info()
    return df_info, df_head
```

In [3]:

```
#using the created function, we can then reference the file and read the contentes
file_path = '/content/syriatel.csv'
df_info, df_head = load_and_get_info(file_path)
df
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account length          3333 non-null   int64
 2   area code               3333 non-null   int64
 3   phone number            3333 non-null   object
 4   international plan      3333 non-null   object
 5   voice mail plan         3333 non-null   object
 6   number vmail messages   3333 non-null   int64
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64
 15  total night charge      3333 non-null   float64
```

```
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64
 20  churn                   3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

Out[3]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	t
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	
...	...	...	...	...	...	...	...	...	...	...	...	
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...	
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...	
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...	
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...	
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...	

3333 rows × 21 columns

We can see that the Dataset contains 3333 rows and 21 columns, with the columns each having unique distinct names. After that , we can then check for the data types in the dataset, whether they are numerical or categorical. We can use a function to check for that.

In [4]:

```
def check_dtypes(df):
    """
    Check data types and identify the kind of variable for each column in the DataFrame.
    Parameters:
    - df (DataFrame): Input DataFrame.
    Returns:
    - dtypes_info (DataFrame): DataFrame containing information about data types and vari
    """
    # Get data types of each column
    dtypes = df.dtypes

    variable_types = []
#the for loop inserts the type of data into a new list
    for col in df.columns:
```

```
        if pd.api.types.is_numeric_dtype(df[col]):
            variable_types.append('Numeric')
        else:
            variable_types.append('Categorical')

    # Create DataFrame to store information
    dtypes_info = pd.DataFrame({'Column': dtypes.index, 'Data Type': dtypes.values, 'Vari

    return dtypes_info

data_types_info = check_dtypes(df)
print(data_types_info)
```

```
                   Column Data Type Variable Type
0                   state    object   Categorical
1           account length     int64       Numeric
2                area code     int64       Numeric
3            phone number    object   Categorical
4       international plan    object   Categorical
5          voice mail plan    object   Categorical
6    number vmail messages     int64       Numeric
7         total day minutes   float64       Numeric
8           total day calls     int64       Numeric
9          total day charge   float64       Numeric
10       total eve minutes   float64       Numeric
11         total eve calls     int64       Numeric
12        total eve charge   float64       Numeric
13     total night minutes   float64       Numeric
14       total night calls     int64       Numeric
15      total night charge   float64       Numeric
16      total intl minutes   float64       Numeric
17        total intl calls     int64       Numeric
18       total intl charge   float64       Numeric
19  customer service calls     int64       Numeric
20                   churn      bool       Numeric
```

As seen, there are 4 categorical values: state, phone number, international plan and voice mail plan.

The rest of the data is numerical values

Now, we need to check if the chosen data has any null or duplicate values. This can be achieved with a function that checks for null and duplicates and displays the statistics in each column.

In [5]:

```
def check_null_and_duplicates(df):
    """
    Check for both null values and duplicated rows in a DataFrame.

    Parameters:
    - df: DataFrame
        The DataFrame to check.

    Returns:
    - info_df: DataFrame
        DataFrame containing information about null values and duplicated rows.
    """
    # Check for null values
    null_values = df.isnull().sum()
    if null_values.sum() > 0:
        print("Null Values:")
        print(null_values)
    else:
        print("No Null Values Found.")
    duplicated_rows = df.duplicated().sum()
```

```
    if duplicated_rows > 0:
        print("\nDuplicated Rows Found:", duplicated_rows)
        duplicated_df = df[df.duplicated()]
        print(duplicated_df)
    else:
        print("\nNo Duplicated Rows Found.")
        duplicated_df = pd.DataFrame()  # Empty DataFrame if no duplicates

    # Create a DataFrame to store information
    info_df = pd.DataFrame({
        'Column': null_values.index,
        'Null Count': null_values.values,
        'Duplicated Rows': duplicated_rows
    })
    return info_df
info_df= check_null_and_duplicates(df)
print (info_df)
```

```
No Null Values Found.

No Duplicated Rows Found.
                    Column  Null Count  Duplicated Rows
0                    state           0                0
1           account length           0                0
2                area code           0                0
3             phone number           0                0
4       international plan           0                0
5          voice mail plan           0                0
6    number vmail messages           0                0
7        total day minutes           0                0
8          total day calls           0                0
9         total day charge           0                0
10       total eve minutes           0                0
11         total eve calls           0                0
12        total eve charge           0                0
13     total night minutes           0                0
14       total night calls           0                0
15      total night charge           0                0
16      total intl minutes           0                0
17        total intl calls           0                0
18       total intl charge           0                0
19  customer service calls           0                0
20                   churn           0                0
```

Our dataset does not have any null and duplicate values. This makes analysis easier as this shows that this is the complete data collected from the source.

Next, We need to now drop the columns that we do not need. Using domain knowledge, i can deduce that the "state" column and the "phone number" column will not be necessary as in the analysis, we do not need string data, and both state and phone number are string. I shall therefore drop those columns using a function

In [6]:
```
#this function is used to drop columns that are not needed
def drop_column(df, column_name):

  # Drop the column
  return df.drop(column_name, axis=1)
```

In [7]:
```
df = drop_column(df, 'state')
df
```

Out[7]: voice number total total total total t

	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	c
0	128	415	382-4657	no	yes	25	265.1	110	45.07	197.4	
1	107	415	371-7191	no	yes	26	161.6	123	27.47	195.5	
2	137	415	358-1921	no	no	0	243.4	114	41.38	121.2	
3	84	408	375-9999	yes	no	0	299.4	71	50.90	61.9	
4	75	415	330-6626	yes	no	0	166.7	113	28.34	148.3	
...	...	...	...	...	...	...	...	...	...	...	
3328	192	415	414-4276	no	yes	36	156.2	77	26.55	215.5	
3329	68	415	370-3271	no	no	0	231.1	57	39.29	153.4	
3330	28	510	328-8230	no	no	0	180.8	109	30.74	288.8	
3331	184	510	364-6381	yes	no	0	213.8	105	36.35	159.6	
3332	74	415	400-4344	no	yes	25	234.4	113	39.85	265.9	

3333 rows × 20 columns

In [8]:
```
df = drop_column(df, 'phone number')
df
```

Out[8]:

	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	to e char
0	128	415	no	yes	25	265.1	110	45.07	197.4	99	16.
1	107	415	no	yes	26	161.6	123	27.47	195.5	103	16.
2	137	415	no	no	0	243.4	114	41.38	121.2	110	10.
3	84	408	yes	no	0	299.4	71	50.90	61.9	88	5.
4	75	415	yes	no	0	166.7	113	28.34	148.3	122	12.
...	...	...	...	...	...	...	...	...	...	...	
3328	192	415	no	yes	36	156.2	77	26.55	215.5	126	18.
3329	68	415	no	no	0	231.1	57	39.29	153.4	55	13.
3330	28	510	no	no	0	180.8	109	30.74	288.8	58	24.
3331	184	510	yes	no	0	213.8	105	36.35	159.6	84	13.
3332	74	415	no	yes	25	234.4	113	39.85	265.9	82	22.

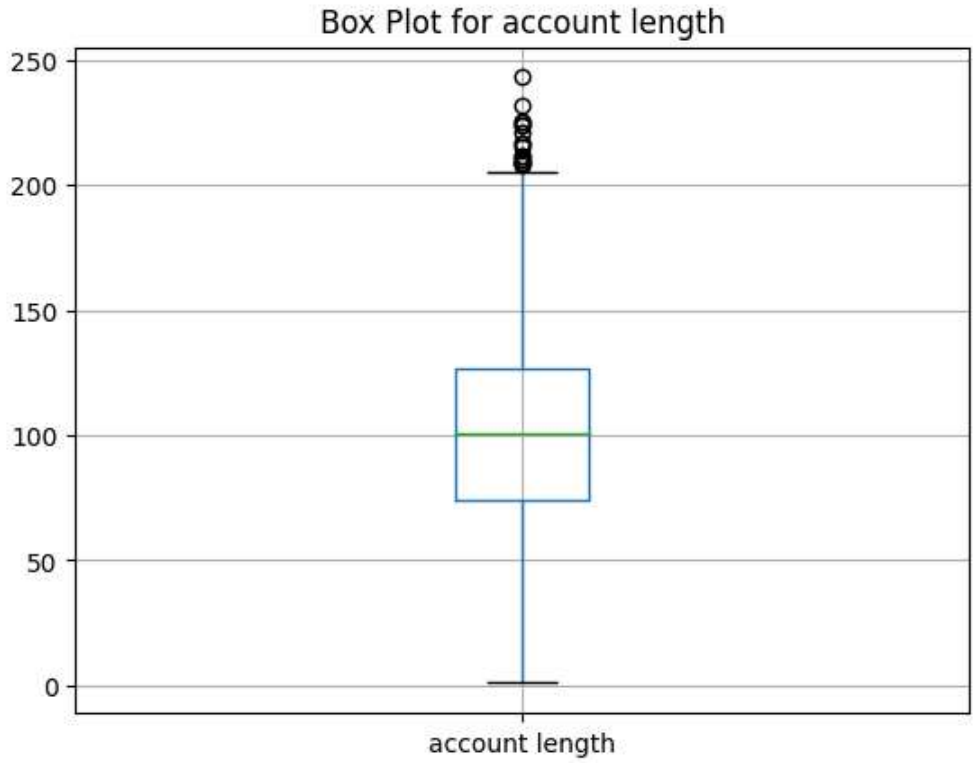
3333 rows × 19 columns

Perfect! The "state" and "phone number" columns have been dropped. Now we can check to see if we have any outliers in our data, as they may lead to erroneous results and make the model perform less optimally. We shall use a function to do so.

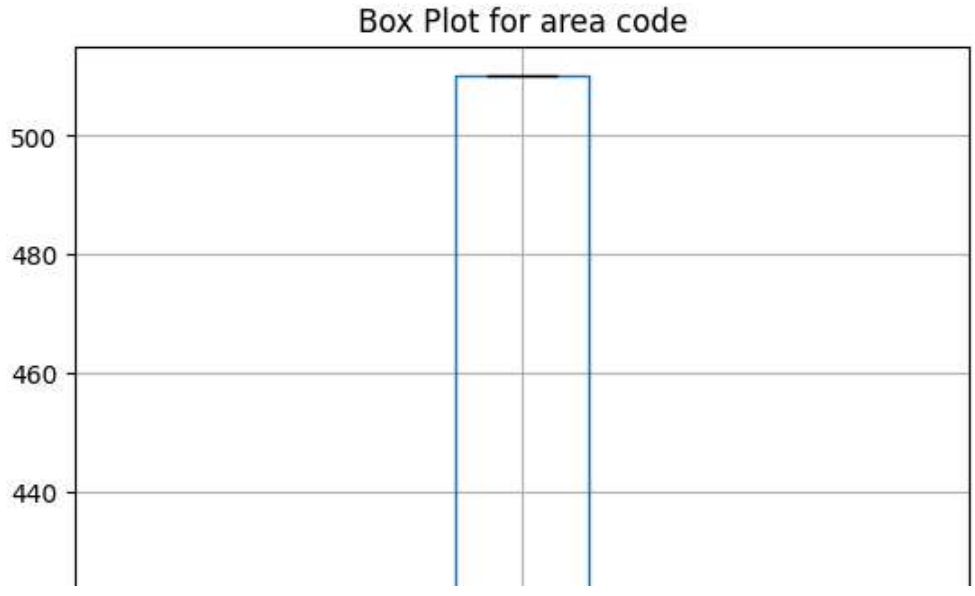
In [9]:

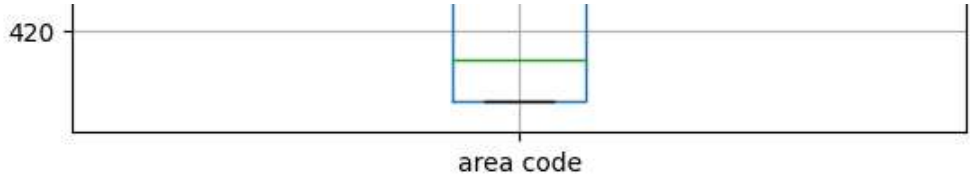
```
#checking outliers

def check_outliers_all(filename, z_thresh=3, iqr_thresh=1.5):
  """
  This function checks for outliers in all numeric columns of a pandas dataframe loaded f

  Args:
      filename (str), z_thresh, iqr_thresh
"""
  # Select numeric columns
  numeric_cols = df.select_dtypes(include=['int64', 'float64'])

  # Dictionary to store outlier information
  outlier_info = {}

  # Loop through columns
  for col in numeric_cols:
    # Descriptive statistics
    print(f"\nColumn Name: {col}")
    print(df[col].describe())

    # Box plot
    df.boxplot(column=col)
    plt.title(f"Box Plot for {col}")
    plt.show()

    # Z-score method
    df['z_score'] = (df[col] - df[col].mean()) / df[col].std()
    outliers_zscore = df[(df['z_score'] < -z_thresh) | (df['z_score'] > z_thresh)][col]

    # IQR method
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - (iqr_thresh * iqr)
    upper_bound = q3 + (iqr_thresh * iqr)
    outliers_iqr = df[(df[col] < lower_bound) | (df[col] > upper_bound)][col]

    # Combine outliers
    outliers = pd.concat([outliers_zscore, outliers_iqr]).drop_duplicates()

    # Store outlier information for this column
    outlier_info[col] = {'Z-score outliers': outliers_zscore, 'IQR outliers': outliers_iq

    # Clear temporary column
    del df['z_score']

  return outlier_info


outlier_info = check_outliers_all(file_path)
```

```
Column Name: account length
count    3333.000000
mean      101.064806
std        39.822106
min         1.000000
25%        74.000000
50%       101.000000
75%       127.000000
max       243.000000
Name: account length, dtype: float64
```

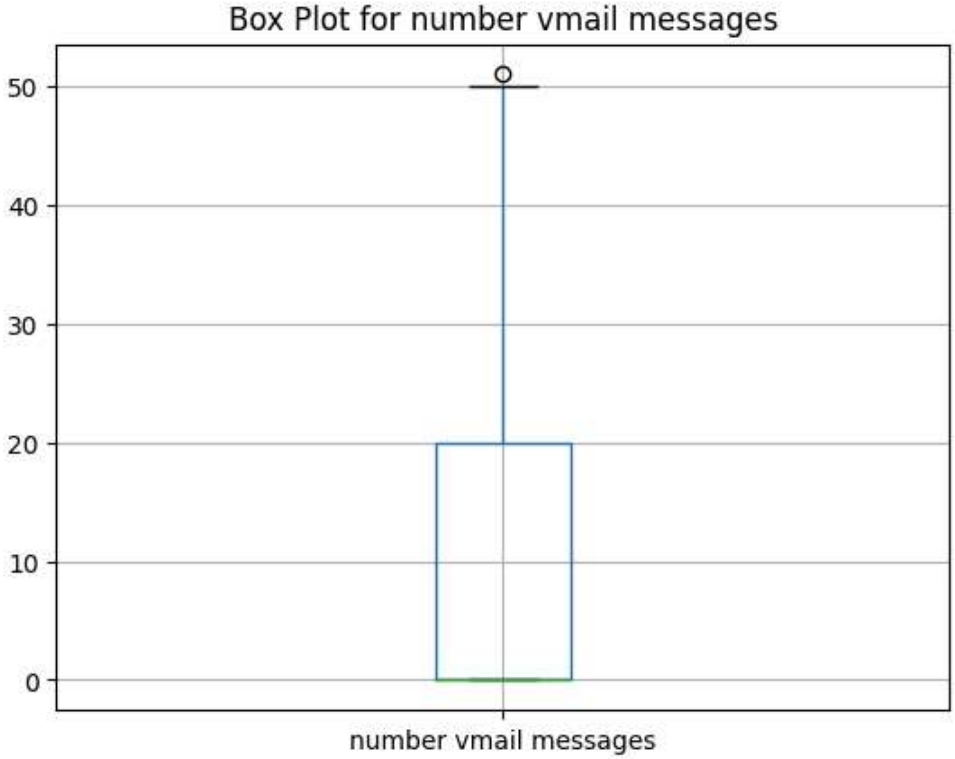



```
Column Name: area code
count    3333.000000
mean      437.182418
std        42.371290
min       408.000000
25%       408.000000
50%       415.000000
75%       510.000000
max       510.000000
Name: area code, dtype: float64
```

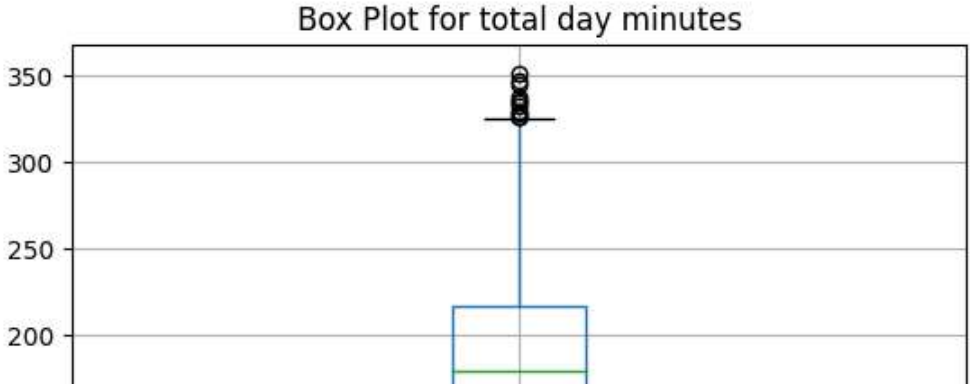


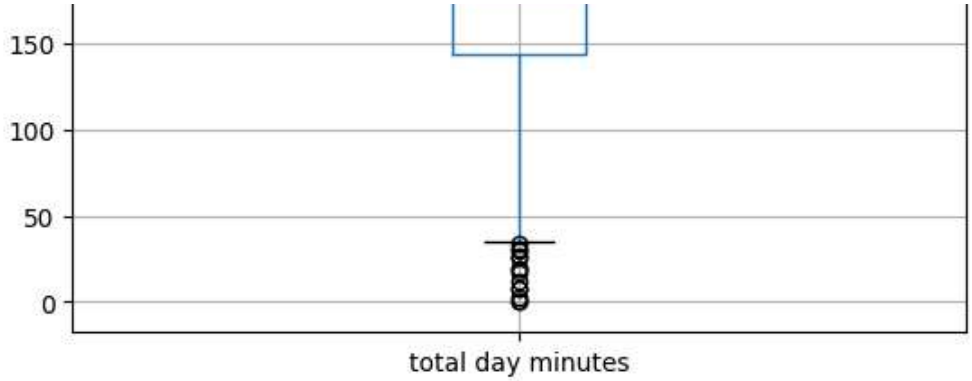



```
Column Name: number vmail messages
count    3333.000000
mean        8.099010
std        13.688365
min         0.000000
25%         0.000000
50%         0.000000
75%        20.000000
max        51.000000
Name: number vmail messages, dtype: float64
```

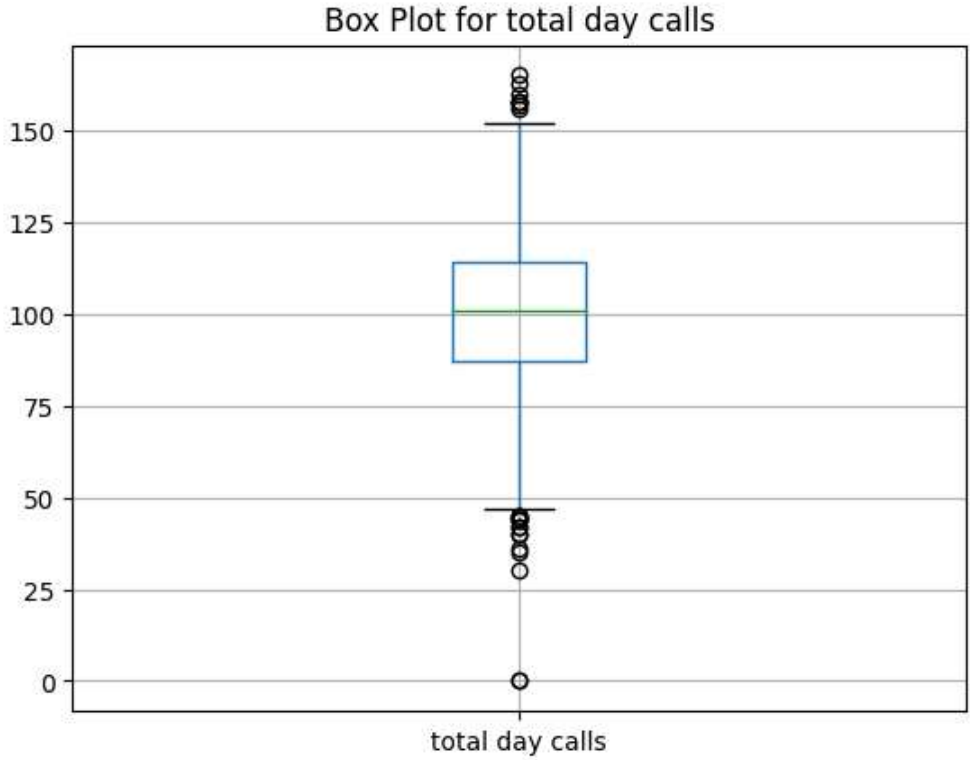



```
Column Name: total day minutes
count    3333.000000
mean      179.775098
std        54.467389
min         0.000000
25%       143.700000
50%       179.400000
75%       216.400000
max       350.800000
Name: total day minutes, dtype: float64
```

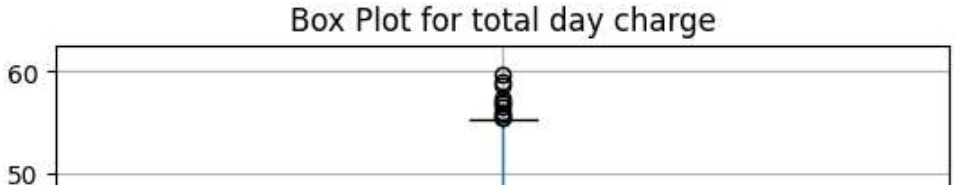


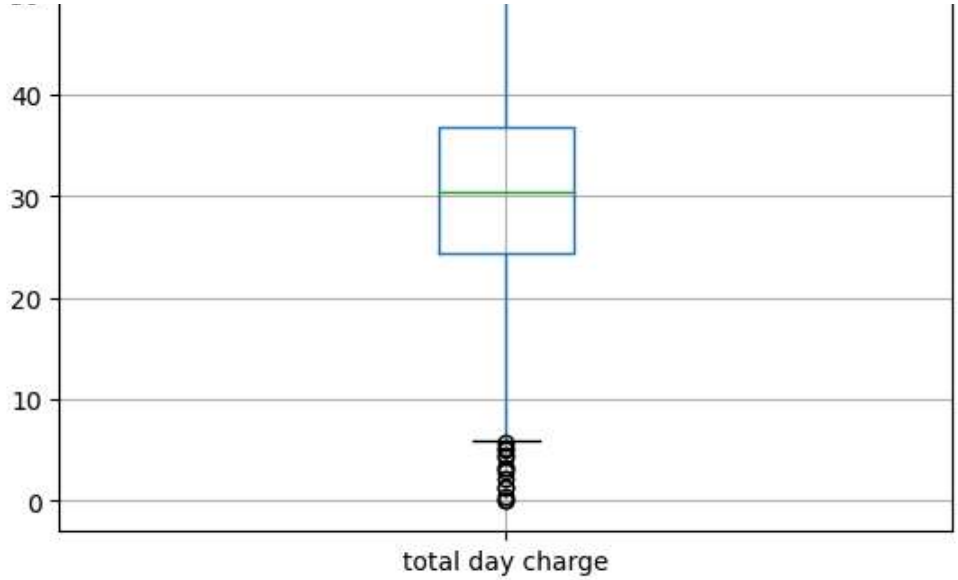



```
Column Name: total day calls
count    3333.000000
mean      100.435644
std        20.069084
min         0.000000
25%        87.000000
50%       101.000000
75%       114.000000
max       165.000000
Name: total day calls, dtype: float64
```

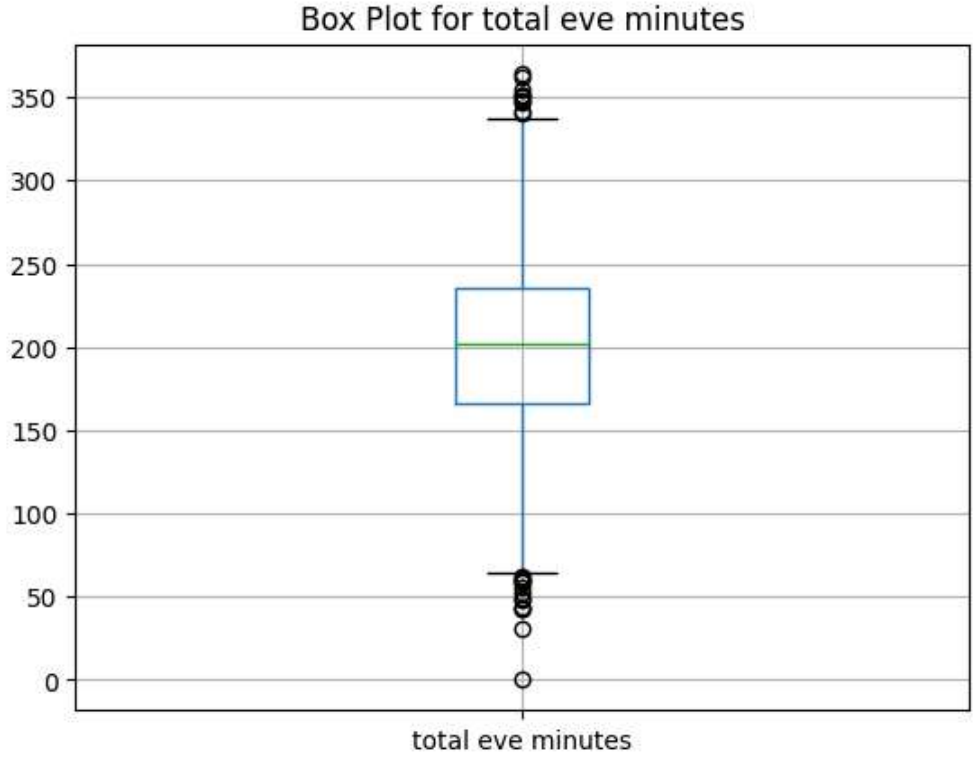



```
Column Name: total day charge
count    3333.000000
mean       30.562307
std         9.259435
min         0.000000
25%        24.430000
50%        30.500000
75%        36.790000
max        59.640000
Name: total day charge, dtype: float64
```

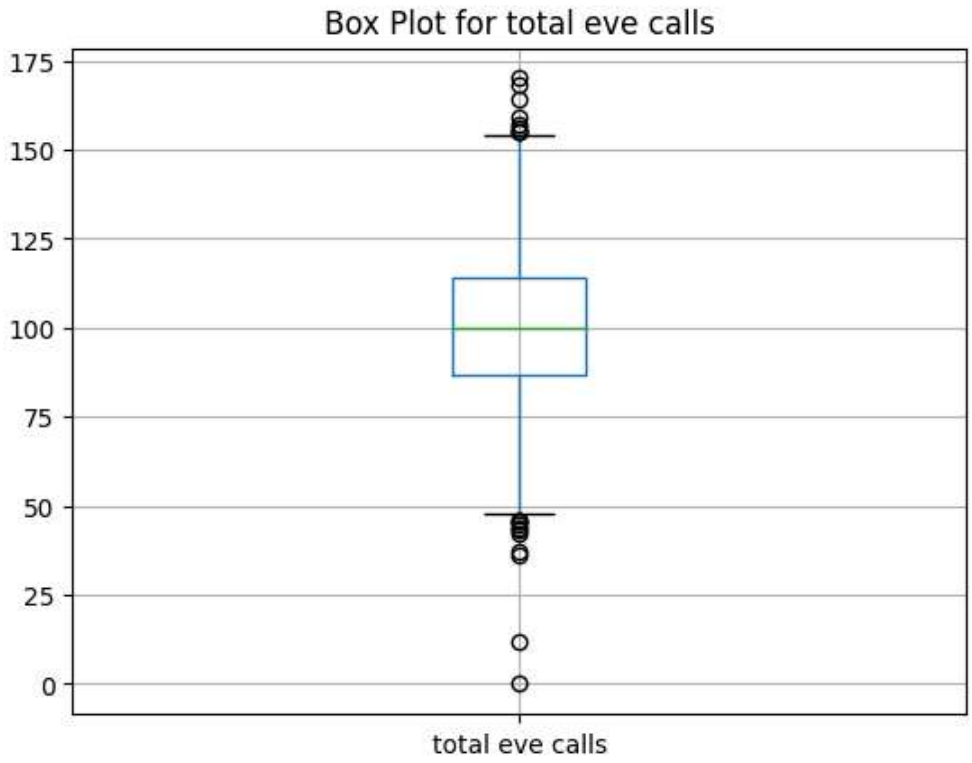




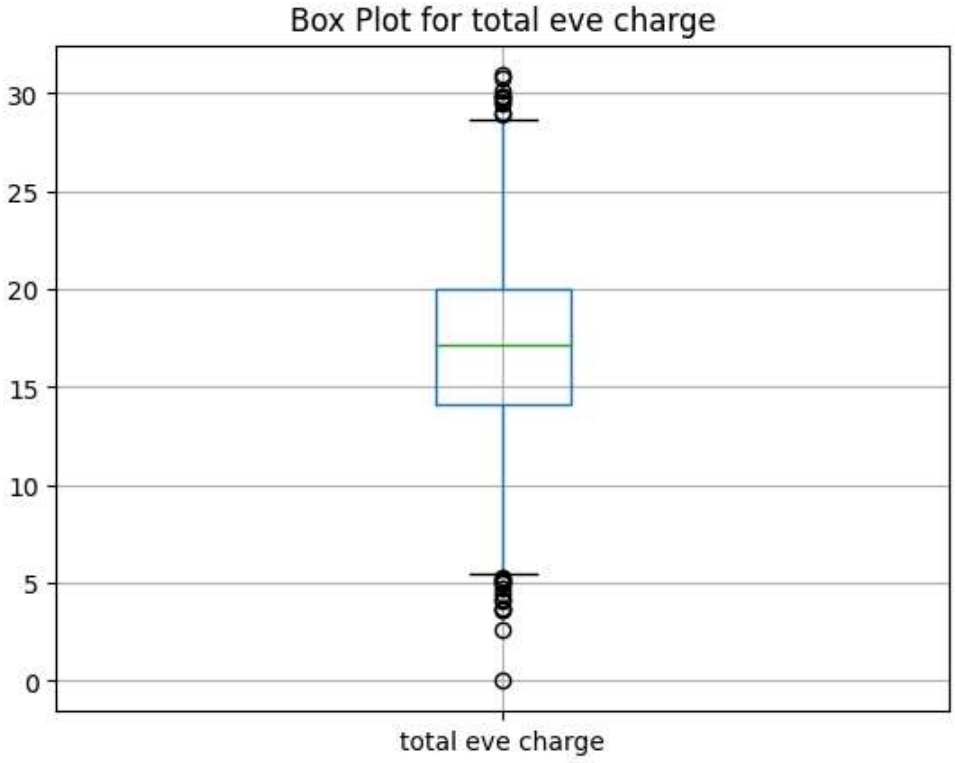

```
Column Name: total eve minutes
count    3333.000000
mean      200.980348
std        50.713844
min         0.000000
25%       166.600000
50%       201.400000
75%       235.300000
max       363.700000
Name: total eve minutes, dtype: float64
```




```
Column Name: total eve calls
count    3333.000000
mean      100.114311
std        19.922625
min         0.000000
25%        87.000000
50%       100.000000
75%       114.000000
max       170.000000
Name: total eve calls, dtype: float64
```

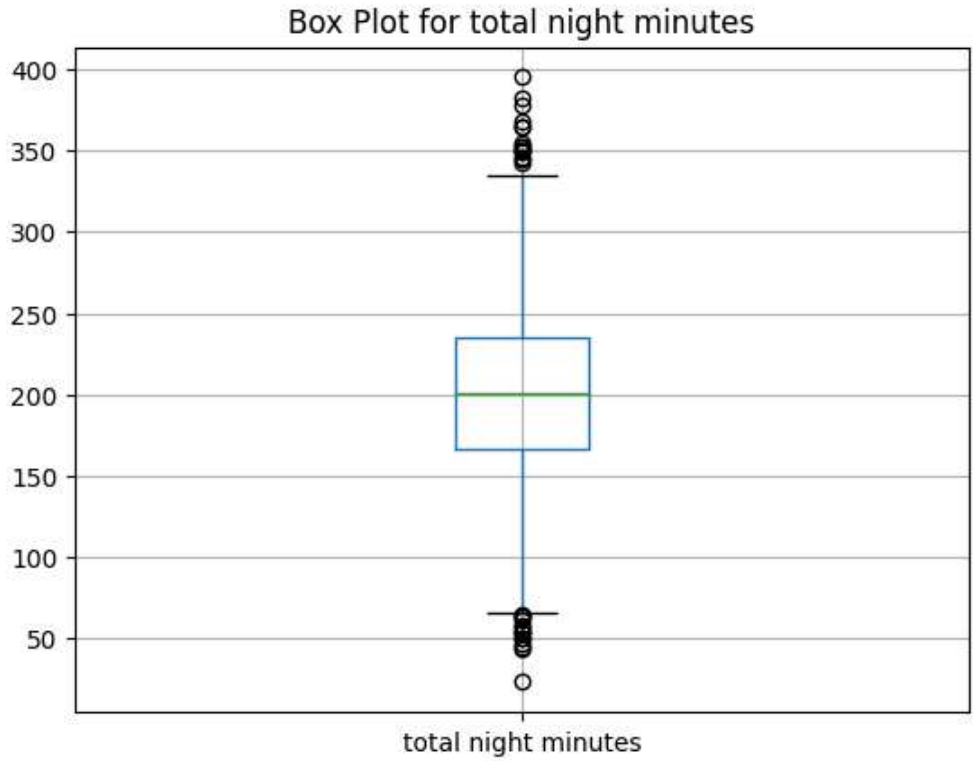



```
Column Name: total eve charge
count    3333.000000
mean       17.083540
std         4.310668
min         0.000000
25%        14.160000
50%        17.120000
75%        20.000000
max        30.910000
Name: total eve charge, dtype: float64
```

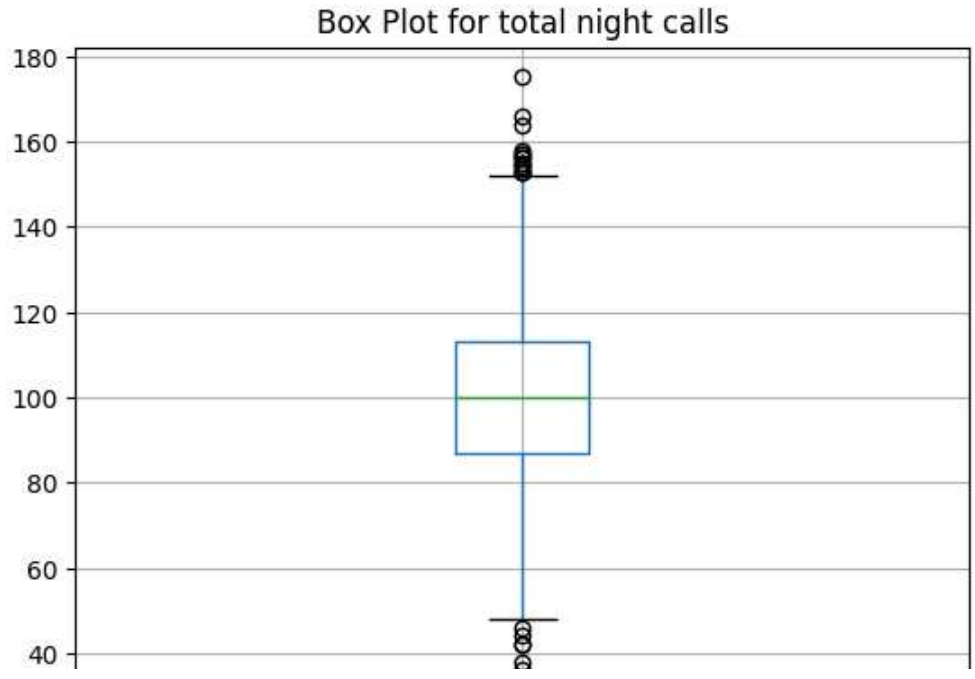


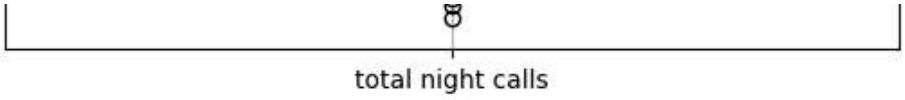

```
Column Name: total night minutes
count    3333.000000
mean      200.872037
```

```
std        50.573847
min        23.200000
25%       167.000000
50%       201.200000
75%       235.300000
max       395.000000
Name: total night minutes, dtype: float64
```

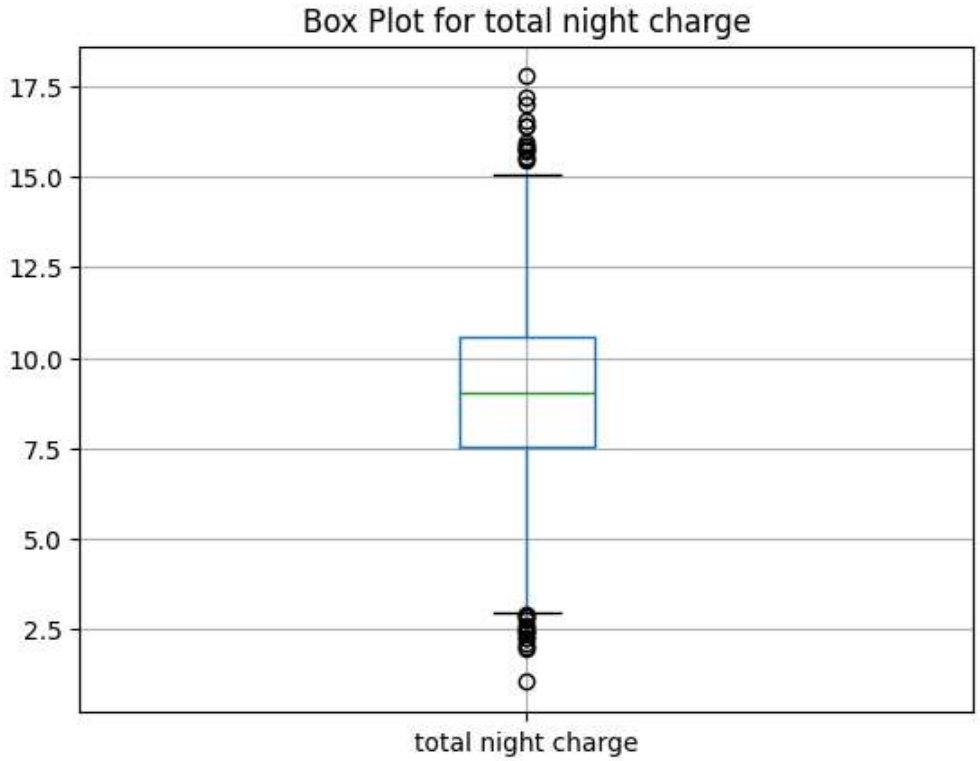



```
Column Name: total night calls
count    3333.000000
mean      100.107711
std        19.568609
min        33.000000
25%        87.000000
50%       100.000000
75%       113.000000
max       175.000000
Name: total night calls, dtype: float64
```

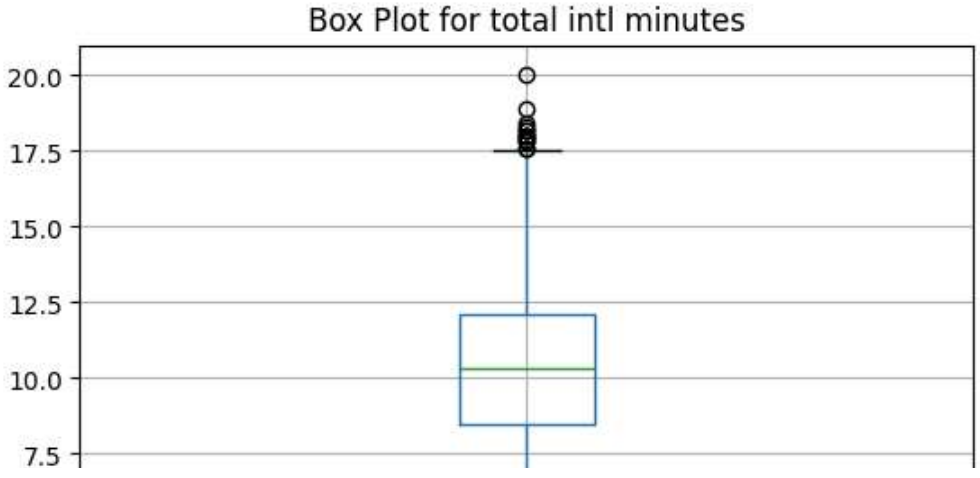


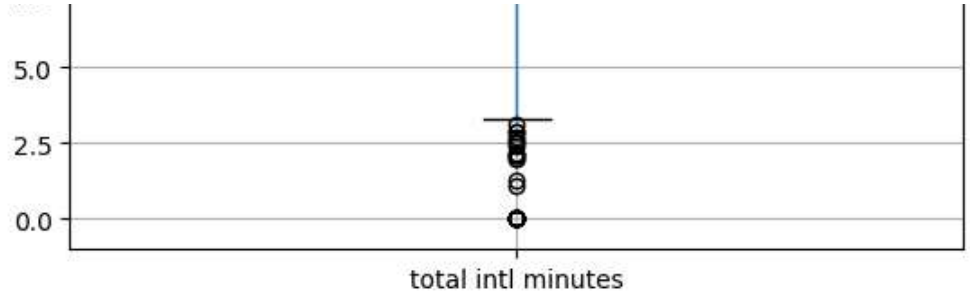



```
Column Name: total night charge
count    3333.000000
mean        9.039325
std         2.275873
min         1.040000
25%         7.520000
50%         9.050000
75%        10.590000
max        17.770000
Name: total night charge, dtype: float64
```

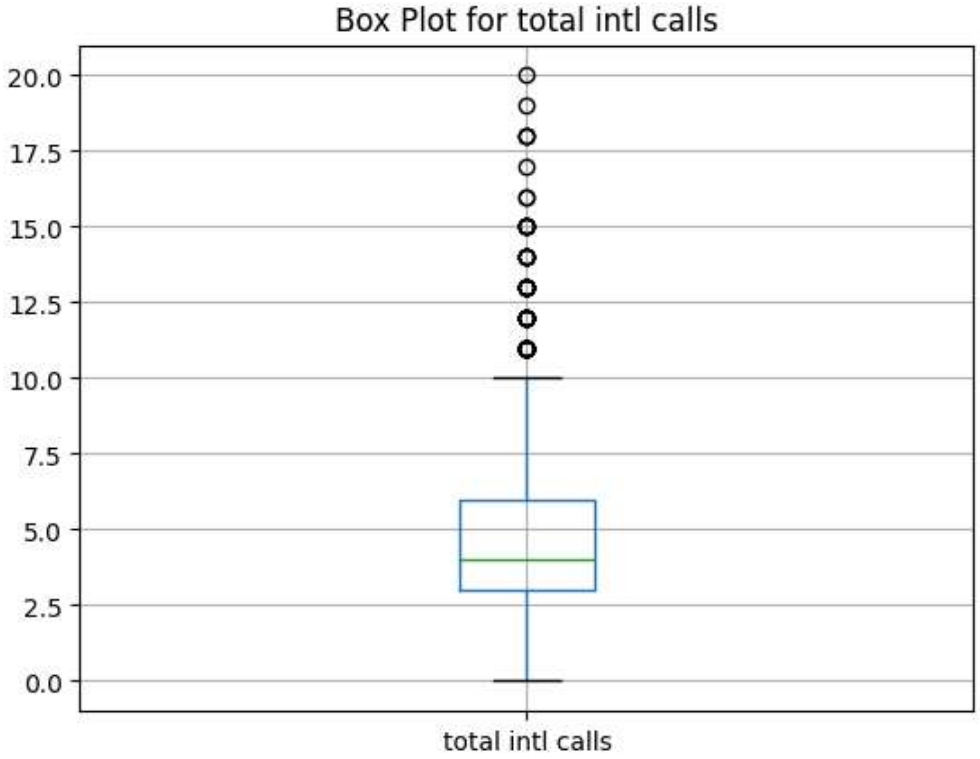



```
Column Name: total intl minutes
count    3333.000000
mean       10.237294
std         2.791840
min         0.000000
25%         8.500000
50%        10.300000
75%        12.100000
max        20.000000
Name: total intl minutes, dtype: float64
```

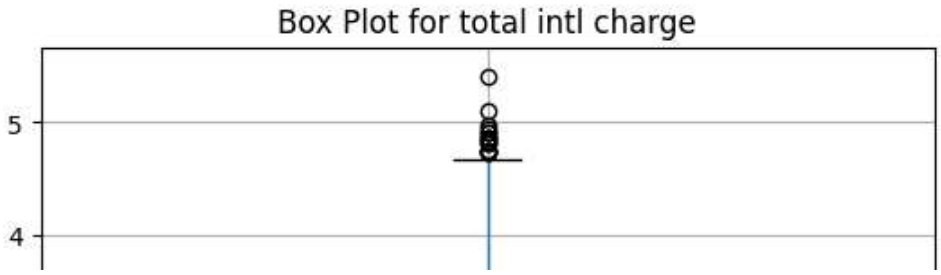


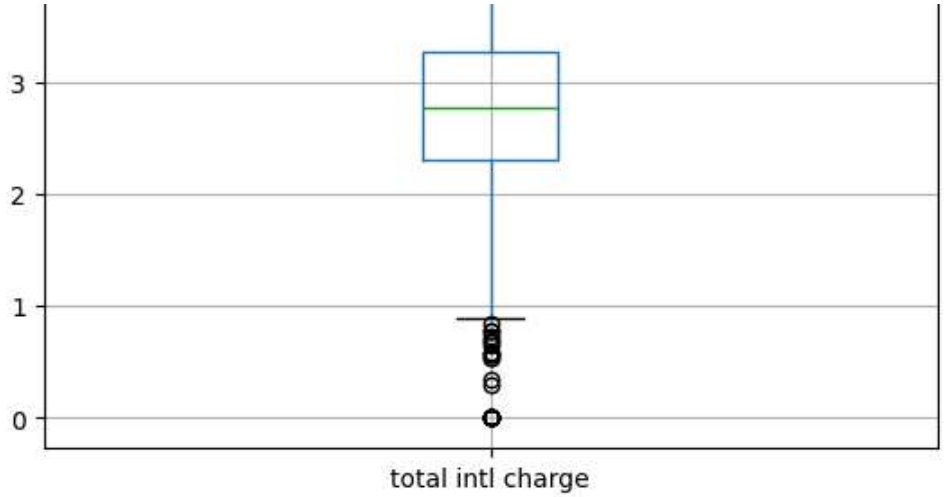



```
Column Name: total intl calls
count    3333.000000
mean        4.479448
std         2.461214
min         0.000000
25%         3.000000
50%         4.000000
75%         6.000000
max        20.000000
Name: total intl calls, dtype: float64
```

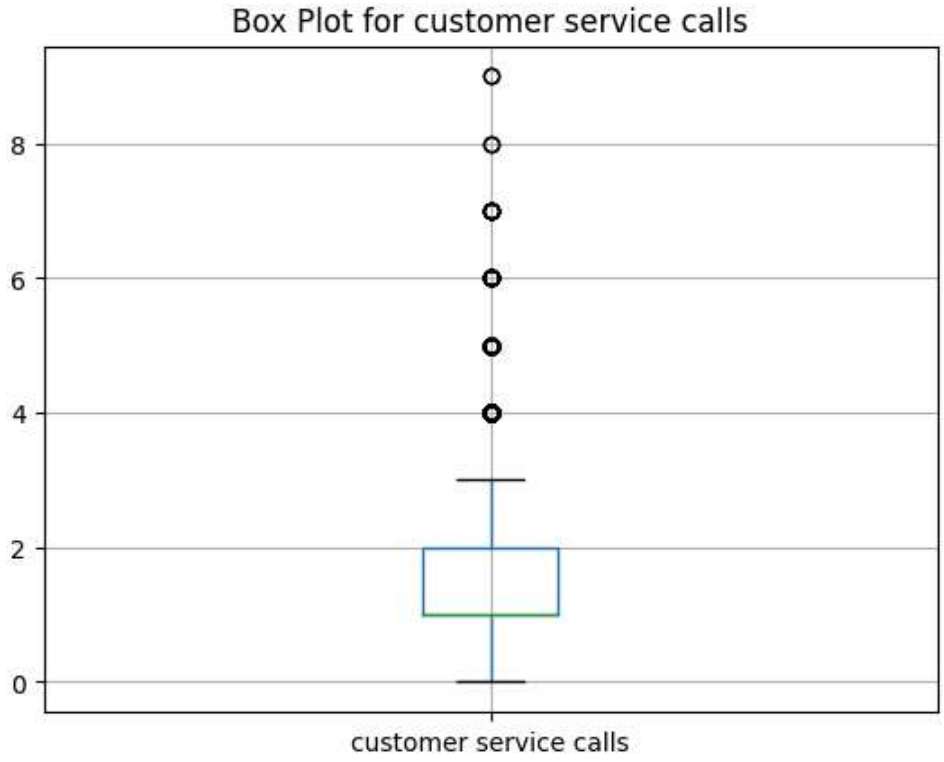



```
Column Name: total intl charge
count    3333.000000
mean        2.764581
std         0.753773
min         0.000000
25%         2.300000
50%         2.780000
75%         3.270000
max         5.400000
Name: total intl charge, dtype: float64
```






```
Column Name: customer service calls
count    3333.000000
mean        1.562856
std         1.315491
min         0.000000
25%         1.000000
50%         1.000000
75%         2.000000
max         9.000000
Name: customer service calls, dtype: float64
```



As shown, each column has outliers that have been displayed in a box plot. They can now be removed using another function to remove outliers

In [10]:

```
def remove_outliers(df, outlier_info):
  """
  This function removes outliers from a dataframe based on the information provided by th

  Args:
      df (pandas.DataFrame): The dataframe containing the data.
```

```
      outlier_info (dict): The dictionary containing outlier information for each column,
"""
  # Loop through columns
  for col, outliers in outlier_info.items():
    # Combine outliers from Z-score and IQR
    all_outliers = pd.concat([outliers['Z-score outliers'], outliers['IQR outliers']])
    # Remove outliers using try-except for potential index errors
    try:
      df.drop(all_outliers, inplace=True)
    except KeyError:
      pass  # Do nothing if index error occurs

  return df
```

In [11]:

```
df_without_outliers = remove_outliers(df.copy(), outlier_info)
print(df_without_outliers)
```

```
      account length  area code international plan voice mail plan  \
1                107        415                 no              yes
2                137        415                 no               no
3                 84        408                yes               no
10                65        415                 no               no
21                77        408                 no               no
...              ...        ...                ...              ...
3328             192        415                 no              yes
3329              68        415                 no               no
3330              28        510                 no               no
3331             184        510                yes               no
3332              74        415                 no              yes

      number vmail messages  total day minutes  total day calls  \
1                        26              161.6              123
2                         0              243.4              114
3                         0              299.4               71
10                        0              129.1              137
21                        0               62.4               89
...                     ...                ...              ...
3328                     36              156.2               77
3329                      0              231.1               57
3330                      0              180.8              109
3331                      0              213.8              105
3332                     25              234.4              113

      total day charge  total eve minutes  total eve calls  total eve charge  \
1                27.47              195.5              103             16.62
2                41.38              121.2              110             10.30
3                50.90               61.9               88              5.26
10               21.95              228.5               83             19.42
21               10.61              169.9              121             14.44
...                ...                ...              ...               ...
3328             26.55              215.5              126             18.32
3329             39.29              153.4               55             13.04
3330             30.74              288.8               58             24.55
3331             36.35              159.6               84             13.57
3332             39.85              265.9               82             22.60

      total night minutes  total night calls  total night charge  \
1                   254.4                103               11.45
2                   162.6                104                7.32
3                   196.9                 89                8.86
10                  208.8                111                9.40
```

```
21                  209.6                  64                 9.43
...                   ...                 ...                  ...
3328                279.1                  83                12.56
3329                191.3                 123                 8.61
3330                191.9                  91                 8.64
3331                139.2                 137                 6.26
3332                241.4                  77                10.86

      total intl minutes  total intl calls  total intl charge  \
1                   13.7                 3               3.70
2                   12.2                 5               3.29
3                    6.6                 7               1.78
10                  12.7                 6               3.43
21                   5.7                 6               1.54
...                  ...               ...                ...
3328                 9.9                 6               2.67
3329                 9.6                 4               2.59
3330                14.1                 6               3.81
3331                 5.0                10               1.35
3332                13.7                 4               3.70

      customer service calls  churn
1                          1  False
2                          0  False
3                          2  False
10                         4   True
21                         5   True
...                      ...    ...
3328                       2  False
3329                       3  False
3330                       2  False
3331                       2  False
3332                       0  False

[3290 rows x 19 columns]
```

In [12]:
```
df_without_outliers.isna().sum()
```

Out[12]:
```
account length            0
area code                 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
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

In [13]:
```
df = df_without_outliers
```

After all the preparation, we can refer to it as df to simplify variables
As seen, the shape of the dataset changes

In [14]:
```
df.shape
```

Out[14]: (3290, 19)

In [15]:
```
df
```

Out[15]:

	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	to e char
1	107	415	no	yes	26	161.6	123	27.47	195.5	103	16.
2	137	415	no	no	0	243.4	114	41.38	121.2	110	10.
3	84	408	yes	no	0	299.4	71	50.90	61.9	88	5.
10	65	415	no	no	0	129.1	137	21.95	228.5	83	19.
21	77	408	no	no	0	62.4	89	10.61	169.9	121	14.
...	...	...	...	...	...	...	...	...	...	...	
3328	192	415	no	yes	36	156.2	77	26.55	215.5	126	18.
3329	68	415	no	no	0	231.1	57	39.29	153.4	55	13.
3330	28	510	no	no	0	180.8	109	30.74	288.8	58	24.
3331	184	510	yes	no	0	213.8	105	36.35	159.6	84	13.
3332	74	415	no	yes	25	234.4	113	39.85	265.9	82	22.

3290 rows × 19 columns

DATA ANALYSIS

Our data is now cleaned and prepared for analysis.
I begin with univariate analysis. This involves finding measures of dispersion and central tendency.
This can be achieved by a function that checks for all measures.

In [16]:
```
#function to perform all relevant univariate analysis

def describe_column(df, column_name):
  """
  This function calculates and returns descriptive statistics for a DataFrame column.
"""
  # Get the column
  column = df[column_name]
  # Calculate statistics
  results = {
      "mean": column.mean(),
      "mode": column.mode().iloc[0] if len(column.mode()) > 0 else None,  # Handle case w
      "range": (column.min(), column.max()),
      "variance": column.var(),
      "standard_deviation": column.std()
  }
  return results
```

In [17]:
```
#It is easier to segment them into day, evening, night, and international data
#Day statistics
print("Day statistics")
statistics_day_mins = describe_column(df, 'total day minutes')
print("Day minutes:" ,statistics_day_mins)
statistics_day_calls = describe_column(df, 'total day calls')
print("Day calls:" ,statistics_day_calls)
statistics_day_charge = describe_column(df, 'total day charge')
print("Day charge:" ,statistics_day_charge)

print("*****************************************************************************")

#Evening statistics
print("Eve statistics")
statistics_eve_mins = describe_column(df, 'total eve minutes')
print("Eve minutes:" ,statistics_eve_mins)
statistics_eve_calls = describe_column(df, 'total eve calls')
print("Eve calls:" ,statistics_eve_calls)
statistics_eve_charge = describe_column(df, 'total eve charge')
print("Eve charge:" ,statistics_eve_charge)

print("*****************************************************************************")

#Night statistics
print("Night statistics")
statistics_night_mins = describe_column(df, 'total night minutes')
print("Night minutes:" ,statistics_night_mins)
statistics_night_calls = describe_column(df, 'total night calls')
print("Night calls:" ,statistics_night_calls)
statistics_night_charge = describe_column(df, 'total night charge')
print("Night charge:" ,statistics_night_charge)

print("*****************************************************************************")

#International statistics
print("International statistics")
statistics_intl_mins = describe_column(df, 'total intl minutes')
print("Intl minutes" ,statistics_intl_mins)
statistics_intl_calls = describe_column(df, 'total intl calls')
print("Intl calls:" ,statistics_night_calls)
statistics_intl_charge = describe_column(df, 'total intl charge')
print("Intl charge:" ,statistics_intl_charge)
```

```
Day statistics
Day minutes: {'mean': 179.7044984802432, 'mode': 159.5, 'range': (0.0, 350.8), 'variance':
2964.092925941774, 'standard_deviation': 54.44348377851819}
Day calls: {'mean': 100.47203647416413, 'mode': 102, 'range': (0, 165), 'variance': 402.989
9444681128, 'standard_deviation': 20.074609447461558}
Day charge: {'mean': 30.550310030395135, 'mode': 27.12, 'range': (0.0, 59.64), 'variance':
85.66191829850077, 'standard_deviation': 9.25537240193504}
*****************************************************************************
*****************************************************************
Eve statistics
Eve minutes: {'mean': 200.82483282674775, 'mode': 169.9, 'range': (0.0, 363.7), 'variance':
2560.1558471140333, 'standard_deviation': 50.59798263877754}
Eve calls: {'mean': 100.17993920972644, 'mode': 105, 'range': (0, 170), 'variance': 397.382
02371171843, 'standard_deviation': 19.93444315027933}
Eve charge: {'mean': 17.070316109422492, 'mode': 14.25, 'range': (0.0, 30.91), 'variance':
18.49707791767899, 'standard_deviation': 4.300822934936869}
*****************************************************************************
*****************************************************************
Night statistics
```

```
Night minutes: {'mean': 200.8179331306991, 'mode': 191.4, 'range': (23.2, 395.0), 'varianc
e': 2558.4746494208844, 'standard_deviation': 50.58136662270884}
Night calls: {'mean': 100.0468085106383, 'mode': 105, 'range': (33, 175), 'variance': 382.2
6415065045967, 'standard_deviation': 19.551576679400043}
Night charge: {'mean': 9.036884498480243, 'mode': 9.45, 'range': (1.04, 17.77), 'variance':
5.181178220160968, 'standard_deviation': 2.2762201607403814}
**********************************************************************************************
**************************************************************
International statistics
Intl minutes {'mean': 10.236808510638298, 'mode': 10.0, 'range': (0.0, 20.0), 'variance':
7.762709786975282, 'standard_deviation': 2.7861639914002336}
Intl calls: {'mean': 100.0468085106383, 'mode': 105, 'range': (33, 175), 'variance': 382.26
415065045967, 'standard_deviation': 19.551576679400043}
Intl charge: {'mean': 2.7644407294832822, 'mode': 2.7, 'range': (0.0, 5.4), 'variance': 0.5
658721863612798, 'standard_deviation': 0.7522447649277991}
```

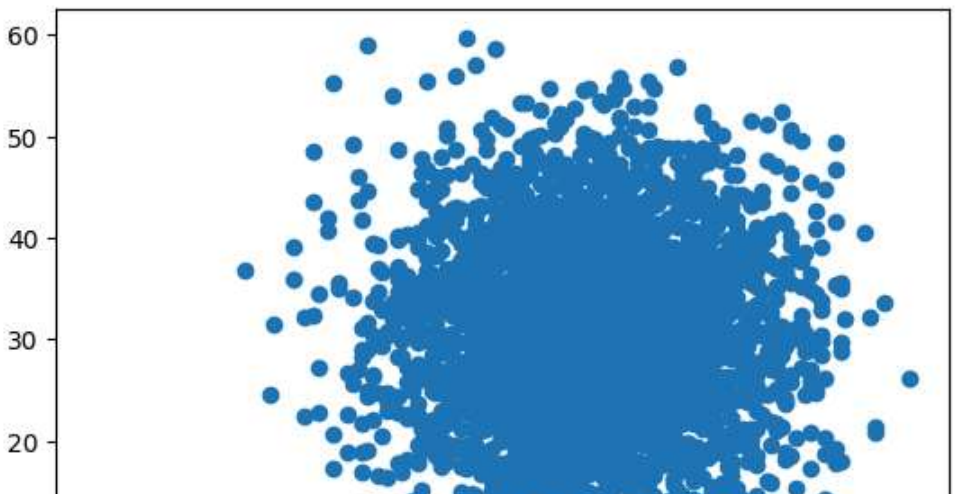
As seen, the data is moderately balanced as the mean is central and it rarely deviates from it. We now move to bivariate analysis, where we try to find relationships between variables. In order to numerically quantify a relationship, we can use correlation coefficient, which can determine if there is any relationship. The higher the value, the higher the correlation between them . We can use a function to achieve that.

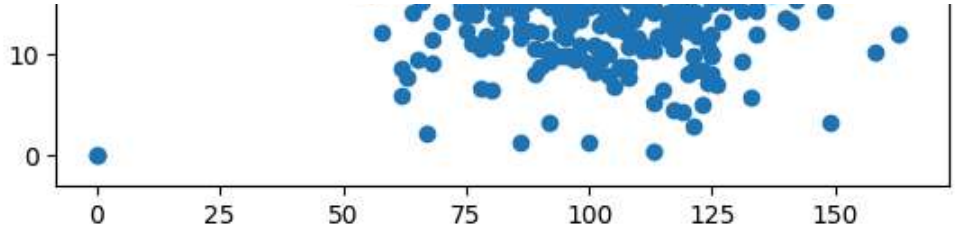
In [18]:
```
def correlation_coefficient(df, column1, column2,):
  """
  This function calculates the Pearson correlation coefficient between 2 columns
float: The Pearson correlation coefficient between the two columns.
"""
  # Calculate correlation
  correlation = df[column1].corr(df[column2])

  return correlation
```

Now we shall find the correlation between each segments. We shall start with day, then evening, then night and international statistics, and visualize them.

In [19]:
```
#correlation in days

correlation = correlation_coefficient(df, 'total day calls', 'total day charge')
print(correlation)

plt.scatter(df['total day calls'],df['total day charge'])
plt.show()
```

```
0.008359230326720412
```

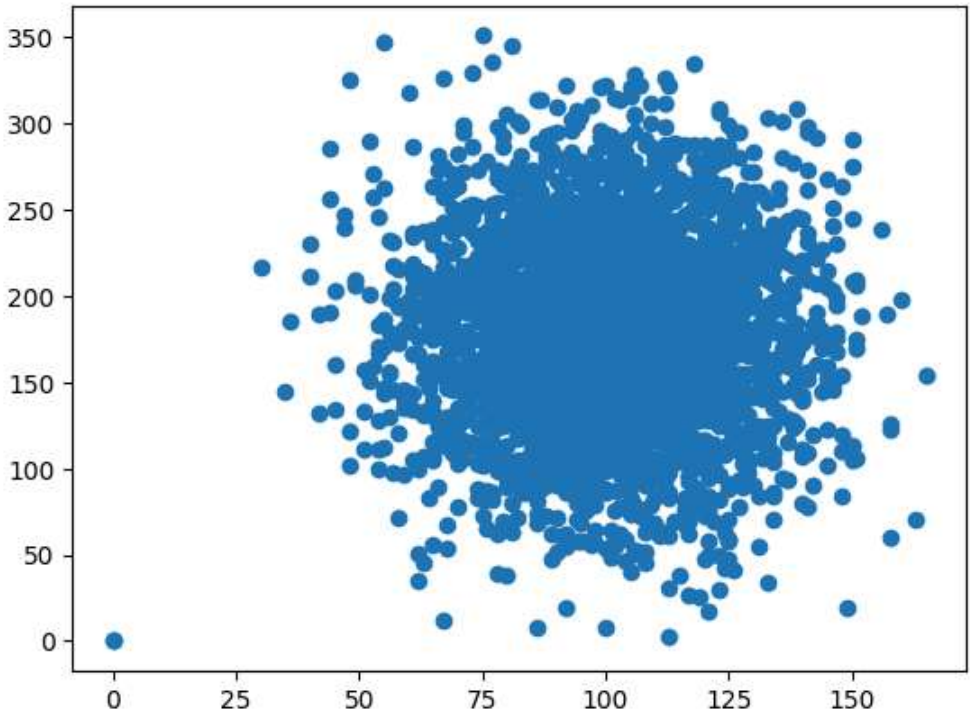




this shows that there is very little correlation between the day calls and charge

In [20]:
```
correlation = correlation_coefficient(df, 'total day calls', 'total day minutes')
print(correlation)

plt.scatter(df['total day calls'],df['total day minutes'])
plt.show()
```

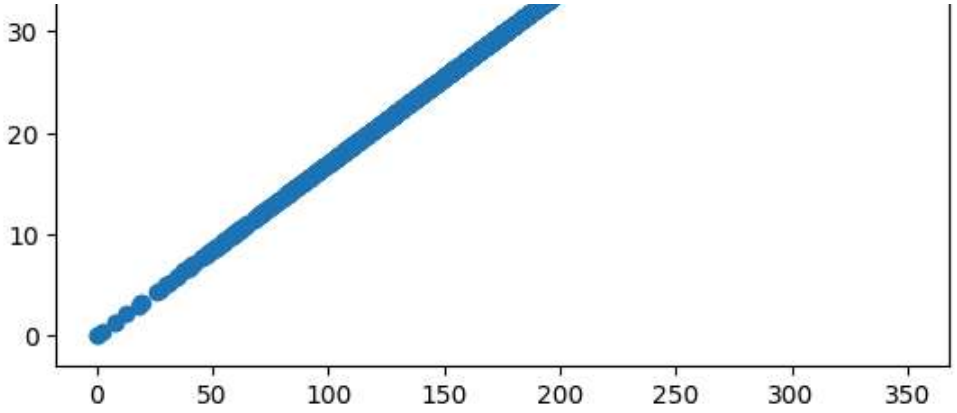
```
0.008356639686033285
```



this shows that there is very little correlation between the day calls and minutes

In [21]:
```
correlation = correlation_coefficient(df, 'total day minutes', 'total day charge')
print(correlation)

plt.scatter(df['total day minutes'],df['total day charge'])
plt.show()
```

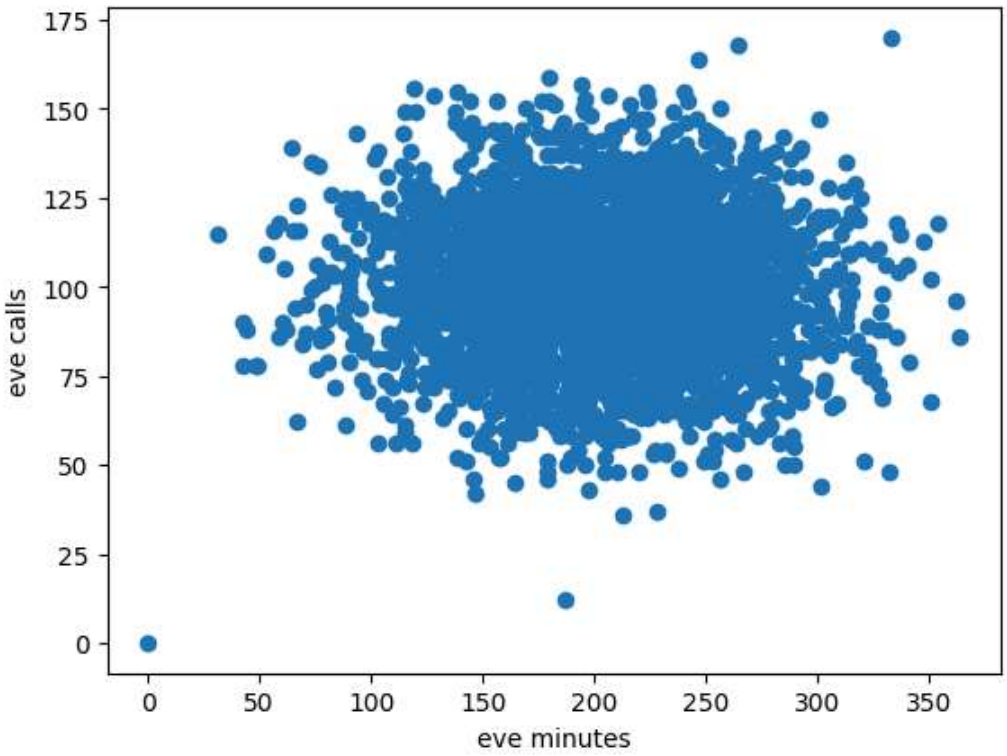
```
0.999999951997937
```



this shows that there is a very high correlation between the day minutes and charge We now move on to the evening statistics

In [22]:
```
#evening
correlation = correlation_coefficient(df, 'total eve minutes', 'total eve calls')
print(correlation)

x = df['total eve minutes']
y = df['total eve calls']
plt.scatter(x,y)

plt.xlabel("eve minutes")
plt.ylabel("eve calls")
plt.show()
```

```
-0.01078580381424453
```

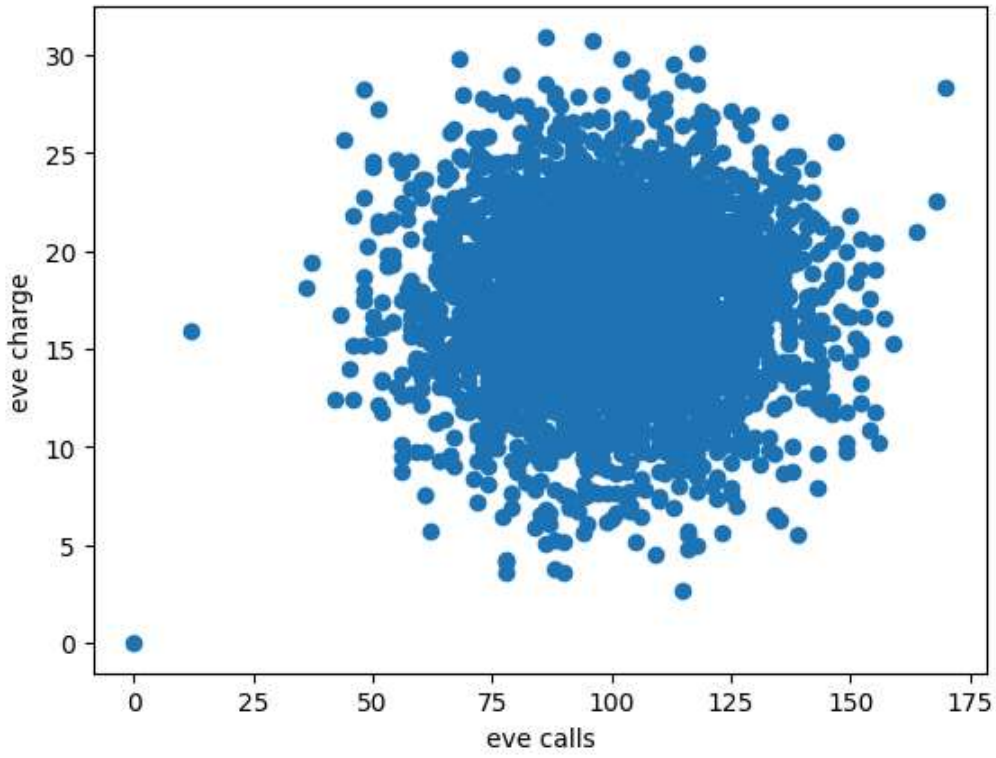


There is a negative correlation between evening calls and evening minutes

In [23]:
```
#evening
correlation = correlation_coefficient(df, 'total eve calls', 'total eve charge')
print(correlation)
```

```
x = df['total eve calls']
y = df['total eve charge']
plt.scatter(x,y)

plt.xlabel("eve calls")
plt.ylabel("eve charge")
plt.show()
```

```
-0.010777425589803671
```

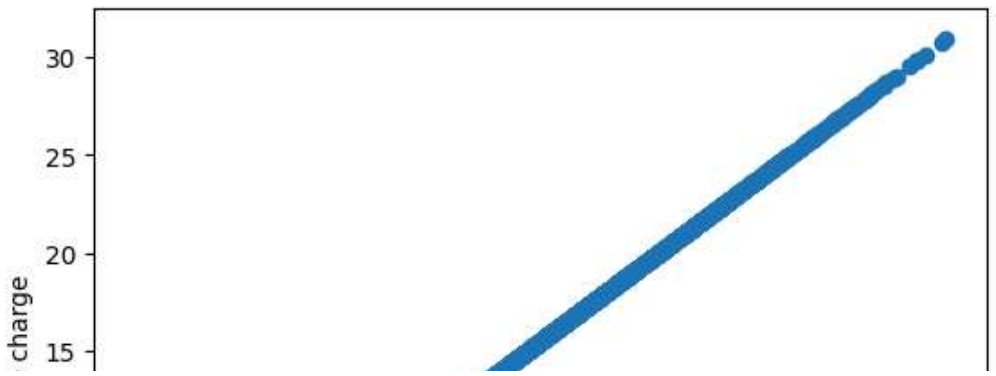


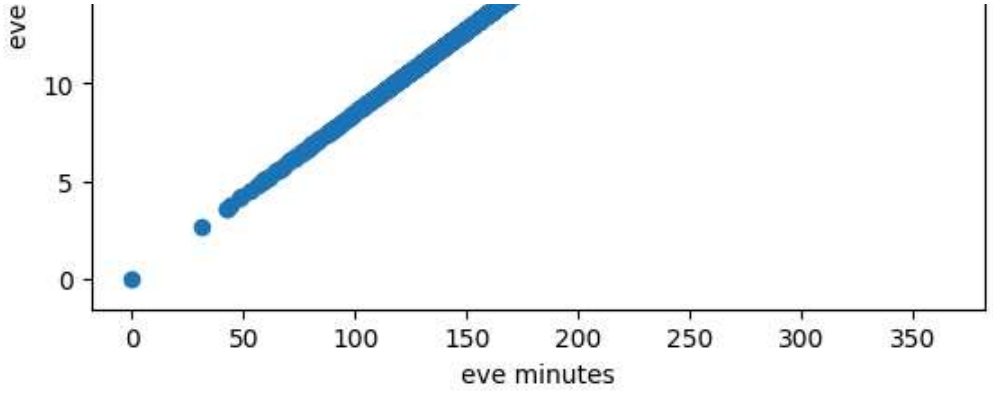
There is also negative correlation between evening calls and evening charge

In [24]:

```
correlation = correlation_coefficient(df, 'total eve minutes', 'total eve charge')
print(correlation)

x = df['total eve minutes']
y = df['total eve charge']
plt.scatter(x,y)

plt.xlabel("eve minutes")
plt.ylabel("eve charge")
plt.show()
```

```
0.9999997749686178
```

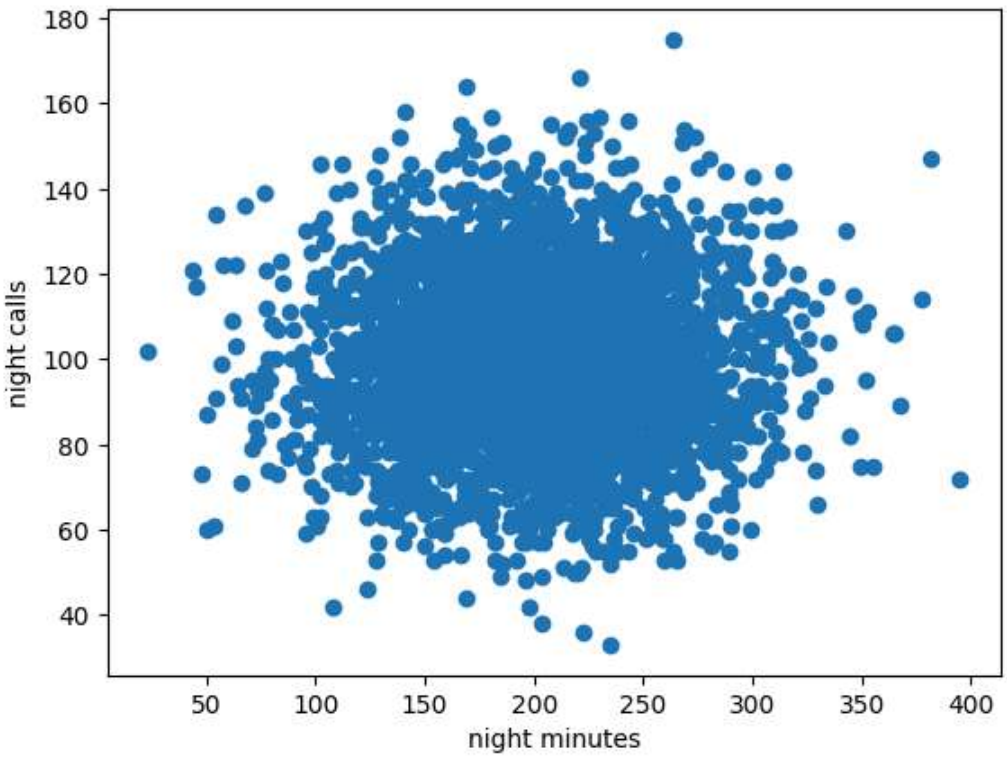




There is a very high correlation between evening minutes and evening charges
We now move on to night statistics

In [25]:
```
#night
correlation = correlation_coefficient(df, 'total night minutes', 'total eve calls')
print(correlation)

x = df['total night minutes']
y = df['total night calls']
plt.scatter(x,y)

plt.xlabel("night minutes")
plt.ylabel("night calls")
plt.show()
```

```
-0.0022194723774711283
```

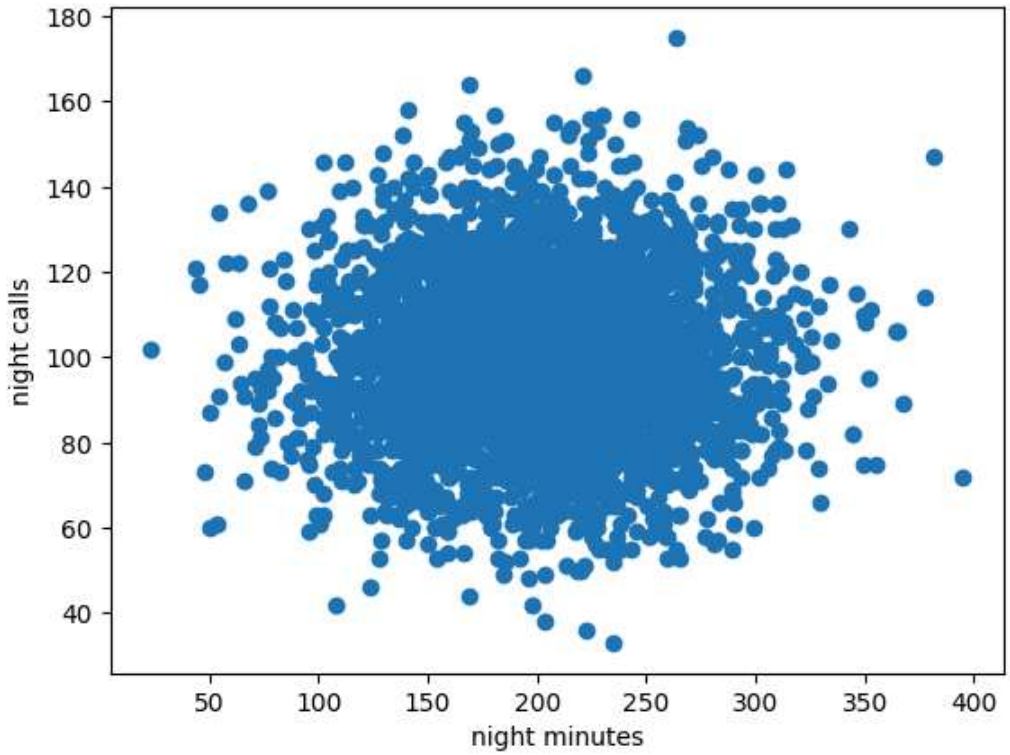


There is negative correlation

In [26]:
```
correlation = correlation_coefficient(df, 'total night minutes', 'total night calls')
print(correlation)

x = df['total night minutes']
```

```
y = df['total night calls']
plt.scatter(x,y)

plt.xlabel("night minutes")
plt.ylabel("night calls")
plt.show()
```

```
0.011190331513930318
```

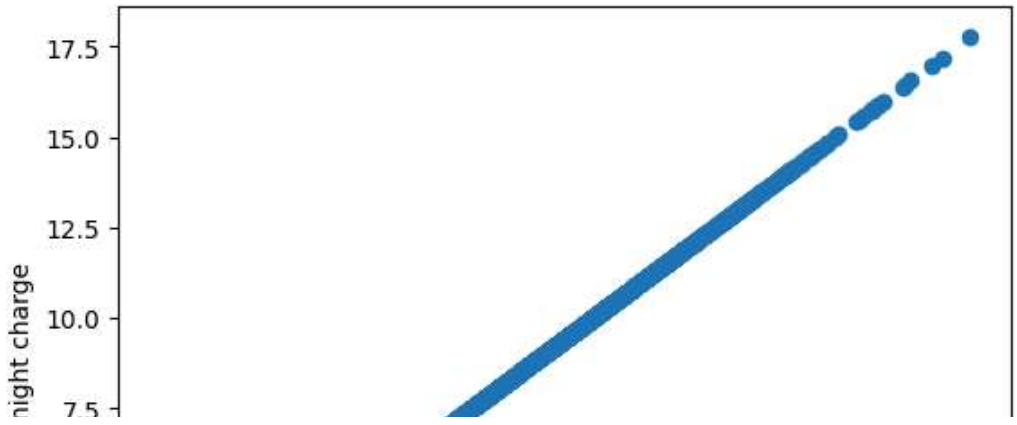


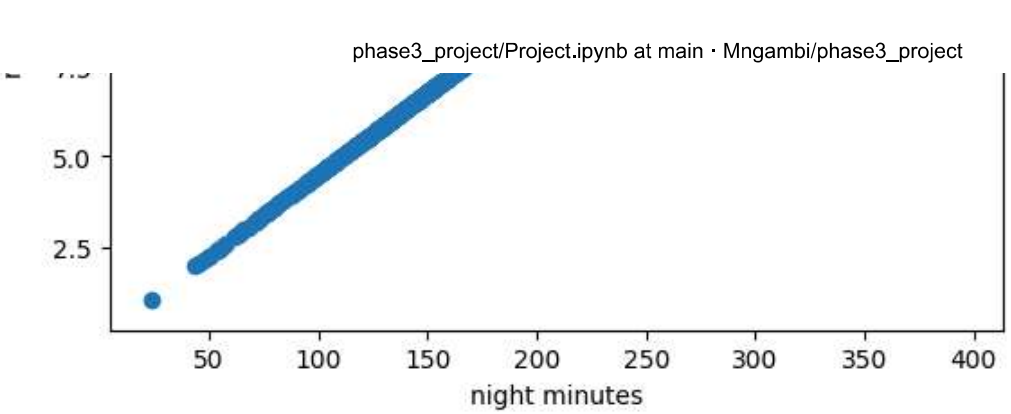
There is very low correlation

In [27]:

```
#evening
correlation = correlation_coefficient(df, 'total night minutes', 'total night charge')
print(correlation)

x = df['total night minutes']
y = df['total night charge']
plt.scatter(x,y)

plt.xlabel("night minutes")
plt.ylabel("night charge")
plt.show()
```

```
0.9999992158831577
```



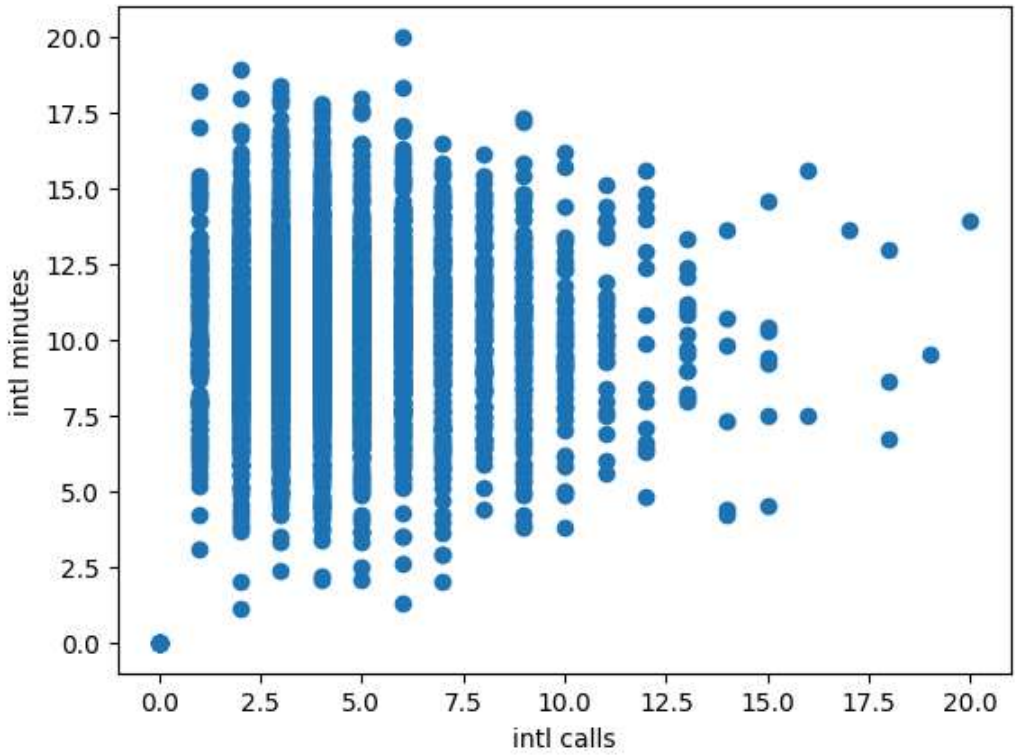


There is very high correlation

We now go to the international statistics

In [28]:
```
#international
correlation = correlation_coefficient(df, 'total intl calls', 'total intl minutes')
print(correlation)

x = df['total intl calls']
y = df['total intl minutes']
plt.scatter(x,y)

plt.xlabel("intl calls")
plt.ylabel("intl minutes")
plt.show()
```

```
0.03332698937060149
```

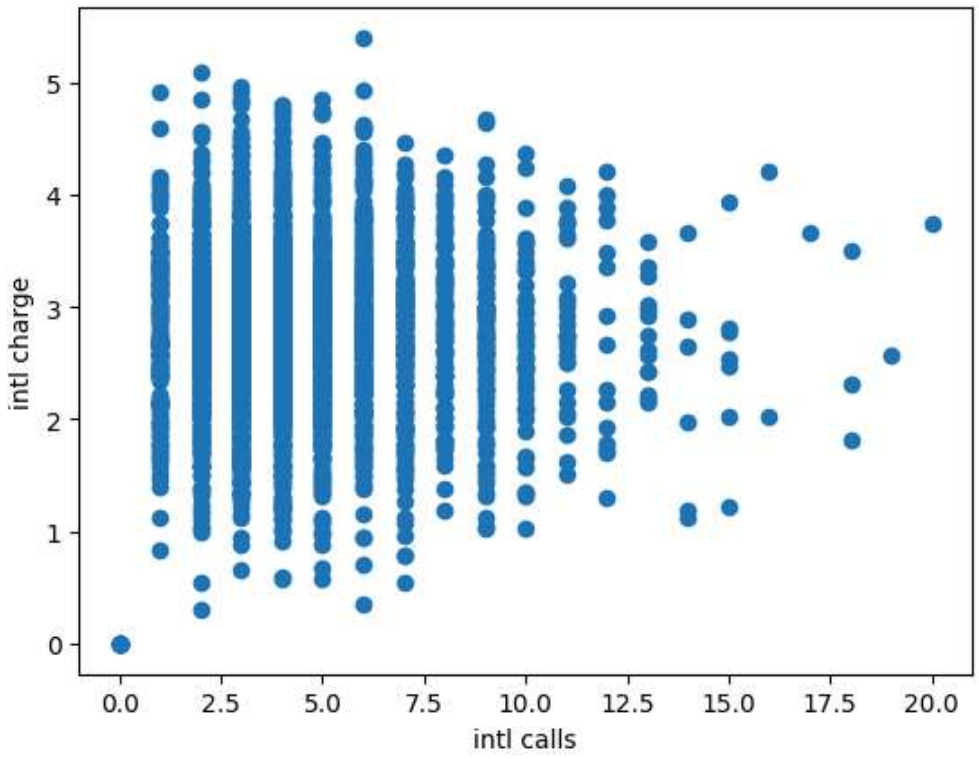


Very low correlation

In [29]:
```
correlation = correlation_coefficient(df, 'total intl calls', 'total intl charge')
print(correlation)

x = df['total intl calls']
y = df['total intl charge']
```

```
plt.scatter(x,y)

plt.xlabel("intl calls")
plt.ylabel("intl charge")
plt.show()
```

```
0.03339450917089022
```

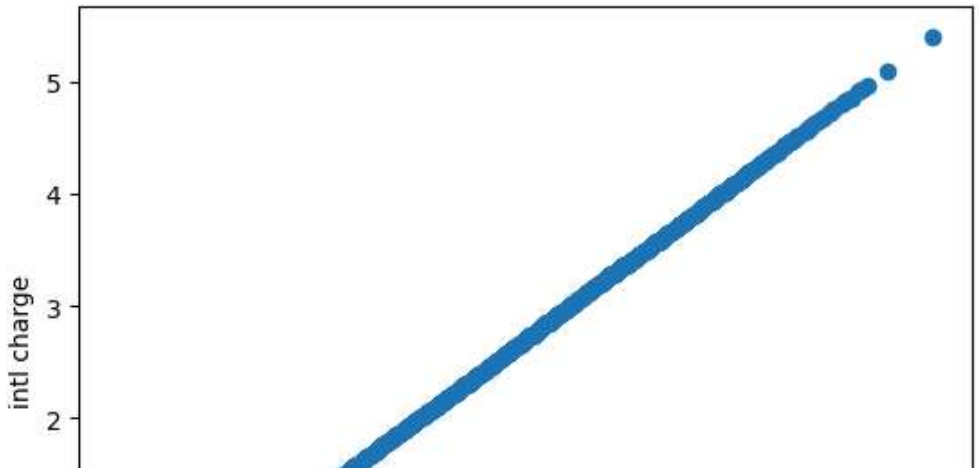


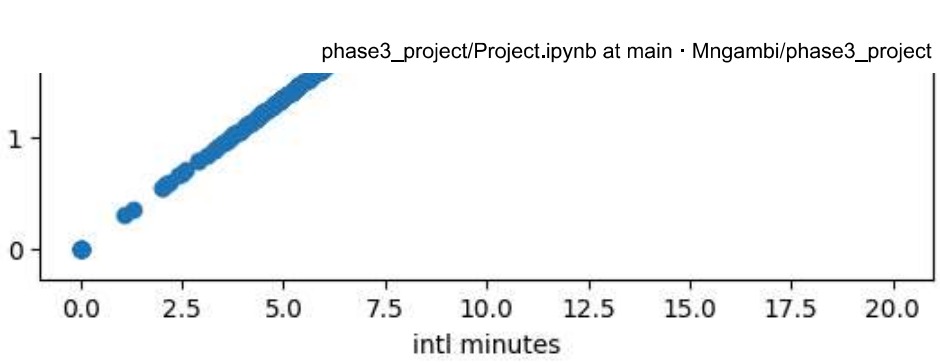
Very low correlation

In [30]:

```
#intl
correlation = correlation_coefficient(df, 'total intl minutes', 'total intl charge')
print(correlation)

x = df['total intl minutes']
y = df['total intl charge']
plt.scatter(x,y)

plt.xlabel("intl minutes")
plt.ylabel("intl charge")
plt.show()
```

```
0.999992726950177
```





Very high correlation.

We can now compare the statistics to the "churn" column, which is table containing boolean values of True and False, and seeing if there is any correlation between the variables and the "churn" column.

We shall begin with calls, in the day evening and night, and internationally

In [31]:
```
correlation = correlation_coefficient(df, 'total day calls', 'churn')
print(correlation)

correlation = correlation_coefficient(df, 'total eve calls', 'churn')
print(correlation)

correlation = correlation_coefficient(df, 'total night calls', 'churn')
print(correlation)

correlation = correlation_coefficient(df, 'total intl calls', 'churn')
print(correlation)
```

```
0.018471316256601575
0.0074002954775920626
0.0056544270748048985
-0.05577739087951315
```

All the types of calls have very low correlation, and with international calls, there is negative correlation, meaning that they do not have any correlation with the "churn database, meaning that calls don't influence the stopping of customers with the service

We now move to the minutes

In [32]:
```
correlation = correlation_coefficient(df, 'total day minutes', 'churn')
print(correlation)

correlation = correlation_coefficient(df, 'total eve minutes', 'churn')
print(correlation)

correlation = correlation_coefficient(df, 'total night minutes', 'churn')
print(correlation)

correlation = correlation_coefficient(df, 'total intl minutes', 'churn')
print(correlation)
```

```
0.20278862118110877
0.09271556076894107
0.03774672621280276
0.0688662225881076
```

There is very low correlation with the minutes, but in the case of day minutes, there is a level of

correlation, albeit low.

We now move to the charge

In [33]:
```
correlation = correlation_coefficient(df, 'total day charge', 'churn')
print(correlation)

correlation = correlation_coefficient(df, 'total eve charge', 'churn')
print(correlation)

correlation = correlation_coefficient(df, 'total night charge', 'churn')
print(correlation)

correlation = correlation_coefficient(df, 'total intl charge', 'churn')
print(correlation)
```

```
0.20278847170429118
0.09270662672421288
0.037747537175428385
0.06888554417669583
```

There is very low correlation with the charge, but in the case of day charge, there is a level of correlation, albeit low.

In [34]:
```
correlation = correlation_coefficient(df, 'number vmail messages', 'churn')
print(correlation)
```

```
-0.08954953309953936
```

there is negative correlation between the number of vmail messages with the churn

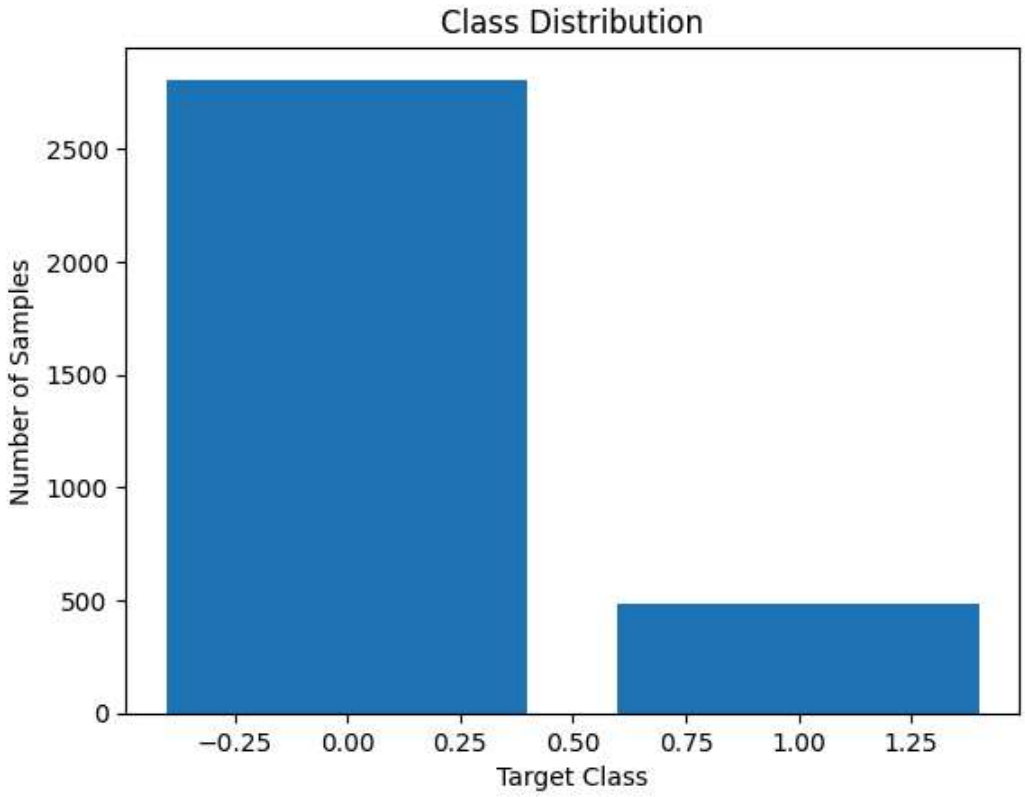
Data Modelling

After the analysis, we can now move to the modelling, in which we develop the model that predicts whether a customer will stop using the service.
First, we need to check for class imbalance .Since it is a binary classification problem, we can use class imbalance ratio to determine if there is a class imbalance.

In [35]:
```
#checking for class imbalance
# Separate features and target variable
features = df.drop('churn', axis=1)  #We will use all columns except the churn as it is t
target = df['churn']

# Count class frequencies
class_counts = target.value_counts().sort_values(ascending=False)

# Calculate class imbalance ratio
majority_class = class_counts.iloc[0]
minority_class = class_counts.iloc[-1]
imbalance_ratio = majority_class / minority_class

# Print class frequencies and imbalance ratio
print("Class frequencies:")
print(class_counts)
print(f"\nClass imbalance ratio: {imbalance_ratio:.2f}")

#Visualization

plt.bar(class_counts.index, class_counts.values)
```

```
plt.xlabel("Target Class")
plt.ylabel("Number of Samples")
plt.title("Class Distribution")
plt.show()
```

```
Class frequencies:
churn
False    2809
True      481
Name: count, dtype: int64

Class imbalance ratio: 5.84
```



With a class imbalance ration of 5.84, we can confer that the dataset is moderately imbalanced. This means that it is more prone to overfitting. This can be rectified through adjusting the test size

There are various algorithms that can be used to model , but i have decided to use decision trees as it allows for easier intepretation , can handle both categorcal and numerical data, and is efficient to train. We first begin by separating the features and target variables, which in our case, the targer variable is the "churn" column

The base model is a decision tree

In [36]:

```
# Separate features and target variable
X = df.drop('churn', axis=1)
y = df['churn'].values

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define preprocessing step (OneHotEncoder for categorical features)
encoder = OneHotEncoder(handle_unknown='ignore')  # Handles unseen categories
```

```
# Encode training and testing data
X_train_encoded = encoder.fit_transform(X_train)
X_test_encoded = encoder.transform(X_test)
```

In [37]:
```
# Model 1: Decision Tree
print("**Model 1: Decision Tree**")

# Define decision tree model
model_dt = DecisionTreeClassifier(random_state=42)

# Train the model on encoded data
model_dt.fit(X_train_encoded, y_train)
```

```
**Model 1: Decision Tree**
```

Out[37]:
```
DecisionTreeClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

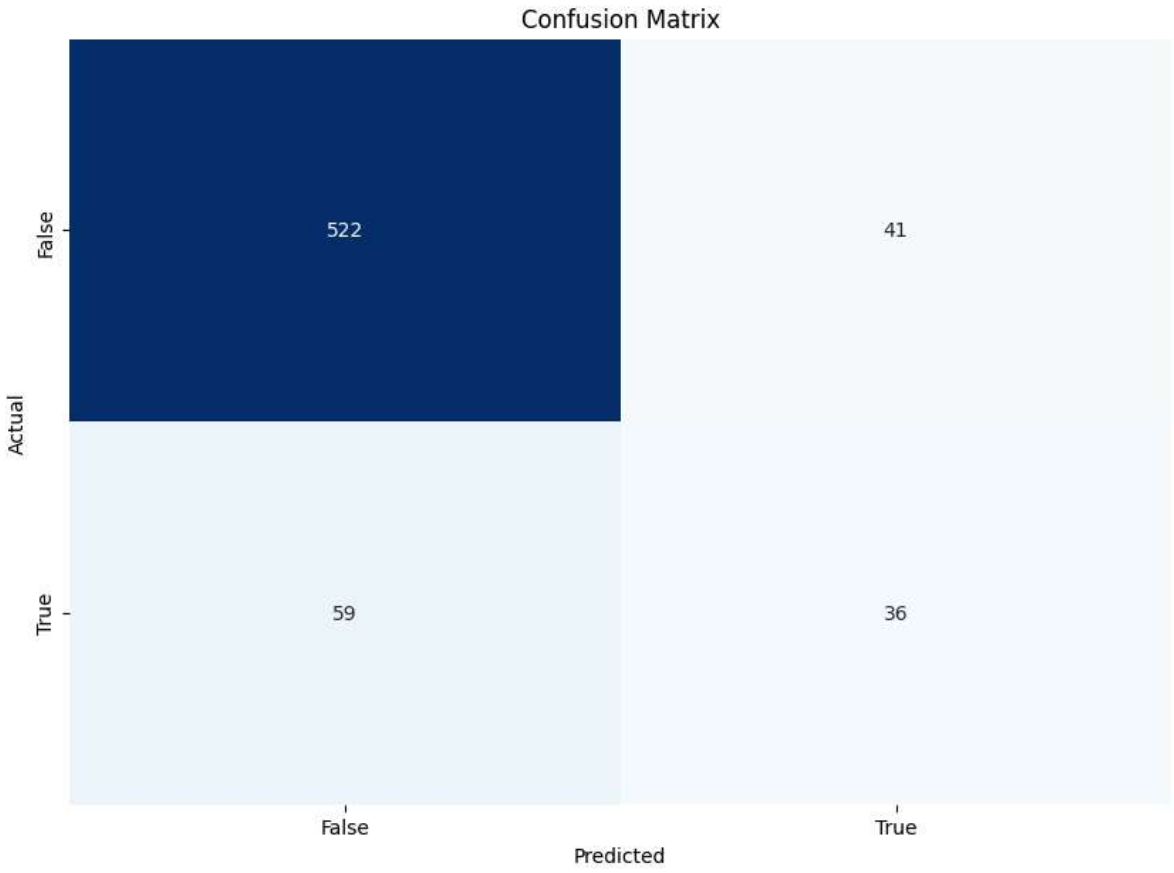
In [38]:
```
# Predict the labels for the test set
y_pred_dt = model_dt.predict(X_test_encoded)

# Evaluate accuracy
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"Model Accuracy: {accuracy_dt:.4f}")

# Evaluate precision
precision_dt = precision_score(y_test, y_pred_dt, average='weighted')
print(f"Model Precision: {precision_dt:.4f}")

# Evaluate recall
recall_dt = recall_score(y_test, y_pred_dt, average='weighted')
print(f"Model Recall: {recall_dt:.4f}")

# Evaluate F1 score
f1_dt = f1_score(y_test, y_pred_dt, average='weighted')
print(f"Model F1 Score: {f1_dt:.4f}")

# Print confusion matrix
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
print("Confusion Matrix:")
print(conf_matrix_dt)

# Print classification report
class_report_dt = classification_report(y_test, y_pred_dt)
print("Classification Report:")
print(class_report_dt)
```

```
Model Accuracy: 0.8480
Model Precision: 0.8362
Model Recall: 0.8480
Model F1 Score: 0.8413
Confusion Matrix:
[[522  41]
 [ 59  36]]
Classification Report:
              precision    recall  f1-score   support

```

```
       False       0.90      0.93      0.91       563
        True       0.47      0.38      0.42        95

    accuracy                           0.85       658
   macro avg       0.68      0.65      0.67       658
weighted avg       0.84      0.85      0.84       658
```

In [39]:

```
#plotting the confusion matrix

plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_dt, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



The second model is a random forest model which is a combination of several decision trees.

In [40]:

```
# Model 2: Random Forest
print("**Model 2: Random Forest**")

# Define random forest model
model_rf = RandomForestClassifier(random_state=42)

# Hyperparameter tuning with GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_depth': [4, 6, 8,],
    'min_samples_split': [2, 5, 10],
}
```

```
grid_search = GridSearchCV(model_rf, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_encoded, y_train)
```

Model 2: Random Forest

Out[40]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [4, 6, 8],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [100, 200, 300, 400, 500]},
             scoring='accuracy')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [41]:
```
# Best model
best_rf_model = grid_search.best_estimator_

# Make predictions on encoded test set with best model
y_pred_rf = best_rf_model.predict(X_test_encoded)

accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

print(f"Accuracy: {accuracy_rf:.4f}")
print(f"Precision: {precision_rf:.4f}")
print(f"Recall: {recall_rf:.4f}")
print(f"F1-Score: {f1_rf:.4f}")

# Print classification report
class_report_rf = classification_report(y_test, y_pred_rf)
print("Classification Report:")
print(class_report_rf)
```

```
Accuracy: 0.8556
Precision: 0.0000
Recall: 0.0000
F1-Score: 0.0000
Classification Report:
              precision    recall  f1-score   support

       False       0.86      1.00      0.92       563
        True       0.00      0.00      0.00        95

    accuracy                           0.86       658
   macro avg       0.43      0.50      0.46       658
weighted avg       0.73      0.86      0.79       658
```

The third model is tuned to optimal parameters

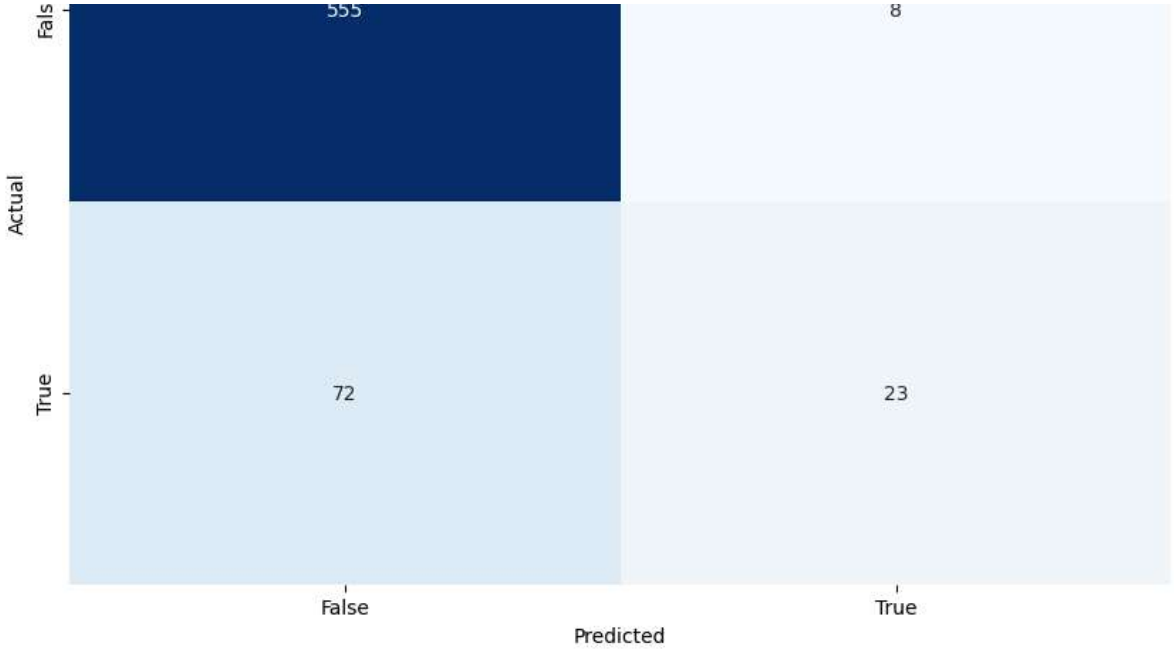
In [42]:
```
# Model 3: Decision Tree (Tuned Hyperparameters)
print("**Model 3: Decision Tree (Tuned Hyperparameters)**")

# hyperparameter tuning
tuned_decision_tree = DecisionTreeClassifier(max_depth=6, min_samples_split=5, random_sta
tuned_decision_tree.fit(X_train_encoded, y_train)

# Make predictions on encoded test set
```

```
y_pred_tuned_dt = tuned_decision_tree.predict(X_test_encoded)

# Evaluate accuracy
accuracy_tuned_dt = accuracy_score(y_test, y_pred_tuned_dt)
print(f"Accuracy (Tuned): {accuracy_tuned_dt:.4f}")

# Evaluate precision
precision_tuned_dt = precision_score(y_test, y_pred_tuned_dt)
print(f"Model Precision: {precision_tuned_dt:.4f}")

# Evaluate recall
recall_tuned_dt = recall_score(y_test, y_pred_tuned_dt)
print(f"Model Recall: {recall_tuned_dt:.4f}")

# Evaluate F1 score
f1_tuned_dt = f1_score(y_test, y_pred_tuned_dt)
print(f"Model F1 Score: {f1_tuned_dt:.4f}")

# Print confusion matrix
conf_matrix_tuned_dt = confusion_matrix(y_test, y_pred_tuned_dt)
print("Confusion Matrix:")
print(conf_matrix_tuned_dt)

# Print classification report
class_report_tuned_dt = classification_report(y_test, y_pred_tuned_dt)
print("Classification Report:")
print(class_report_tuned_dt)
```

```
**Model 3: Decision Tree (Tuned Hyperparameters)**
Accuracy (Tuned): 0.8784
Model Precision: 0.7419
Model Recall: 0.2421
Model F1 Score: 0.3651
Confusion Matrix:
[[555   8]
 [ 72  23]]
Classification Report:
              precision    recall  f1-score   support

       False       0.89      0.99      0.93       563
        True       0.74      0.24      0.37        95

    accuracy                           0.88       658
   macro avg       0.81      0.61      0.65       658
weighted avg       0.86      0.88      0.85       658
```

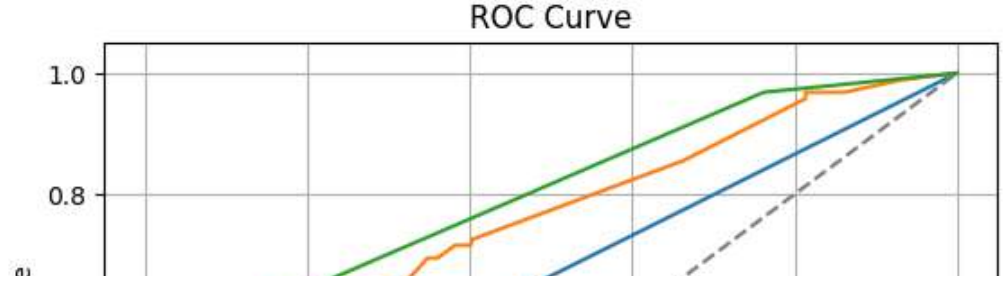
In [43]:

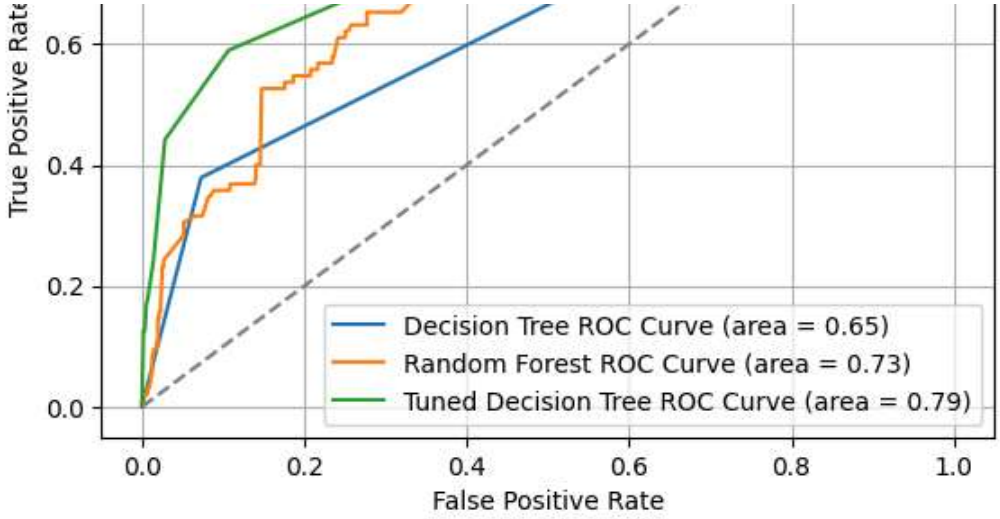
```
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_tuned_dt, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



To see a visual change, we can use the area under the ROC curve to see which is better. The closer it is to the top left, the better the model. This can be plotted visually and shown as below.

In [44]:
```
# Predict probabilities for all models
y_pred_proba_dt = model_dt.predict_proba(X_test_encoded)[:, 1]
y_pred_proba_rf = best_rf_model.predict_proba(X_test_encoded)[:, 1]
y_pred_proba_tuned_dt = tuned_decision_tree.predict_proba(X_test_encoded)[:, 1]

# Calculate ROC curves
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_pred_proba_dt)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_proba_rf)
fpr_tuned_dt, tpr_tuned_dt, _ = roc_curve(y_test, y_pred_proba_tuned_dt)


# Random Line
fpr_random = np.linspace(0, 1, 100)
tpr_random = fpr_random

# Plot ROC curves
plt.plot(fpr_dt, tpr_dt, label='Decision Tree ROC Curve (area = %0.2f)'% auc(fpr_dt, tpr_
plt.plot(fpr_rf, tpr_rf, label='Random Forest ROC Curve (area = %0.2f)'% auc(fpr_rf, tpr_
plt.plot(fpr_tuned_dt, tpr_tuned_dt, label='Tuned Decision Tree ROC Curve (area = %0.2f)'
plt.plot(fpr_random, tpr_random, linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.show()
```





As shown , the tuned decision tree is the best performing one as it has the highest area under the curve

We can compare the accuracy of the model before being tuned and after being tuned.

In [45]:
```
print("Classification metrics before tuning:")
print(class_report_dt)

print("classification metrics after tuning:")
print(class_report_tuned_dt)
```

```
Classification metrics before tuning:
              precision    recall  f1-score   support

       False       0.90      0.93      0.91       563
        True       0.47      0.38      0.42        95

    accuracy                           0.85       658
   macro avg       0.68      0.65      0.67       658
weighted avg       0.84      0.85      0.84       658

classification metrics after tuning:
              precision    recall  f1-score   support

       False       0.89      0.99      0.93       563
        True       0.74      0.24      0.37        95

    accuracy                           0.88       658
   macro avg       0.81      0.61      0.65       658
weighted avg       0.86      0.88      0.85       658
```

Recommendations

Possible recommendations include:

- Advertising for day customers - as they are the ones who are more likely to stay
- incentivize more people to use the service for long as the longer the minutes spent on the call, and therefore the service, leads to more retention
- Personalized offers, in that there will be a higher retention of customers as