Lab Assignment 06

The objective of this lab assignment is to build and evaluate classification models to predict customer churn given information from customers of a telephone company (`data_lab_06.csv`).

Instructions:

Complete each task and question by filling in the blanks (`...`) with one or more lines of code or text. Each task and question is worth **0.5 points** (out of **10 points**).

Submission:

This assignment is due **Monday, November 18, at 11:59PM (Central Time)**.

This assignment must be submitted on Gradescope as a **PDF file** containing the completed code for each task and the corresponding output. Late submissions will be accepted within **0-12** hours after the deadline with a **0.5-point (5%) penalty** and within **12-24** hours after the deadline with a **2-point (20%) penalty**. No late submissions will be accepted more than 24 hours after the deadline.

This assignment is individual. Offering or receiving any kind of unauthorized or unacknowledged assistance is a violation of the University's academic integrity policies, will result in a grade of zero for the assignment, and will be subject to disciplinary action.

Part 1: Decision Trees

In [1]:
```
# Load libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn import metrics
```

In [2]:
```
# Load dataset and display the first five rows
data = pd.read_csv('data_lab_06.csv')
data.head()
```

Out[2]:

	Account length	International plan	Voice mail plan	Number voice mail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes
0	128	0	1	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0
1	107	0	1	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7
2	137	0	0	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2
3	84	1	0	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6
4	75	1	0	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1

Task 01 (of 14): Partition the dataset into training set and test set. *Hint:* Use 75% of the data for training and 25% for testing and set parameter random_state to 0.

In [46]:
```
predicted_vals = data['Churn']
predictor_vals = data.drop(['Churn'], axis=1)
x_train, x_test, y_train, y_test = train_test_split(predictor_vals,
                                                    predicted_vals,
                                                    train_size = .75,
                                                    test_size = 0.25,
                                                    random_state = 0)
```

In [47]:
```
# Show the dimensionality of the training set and the test set
# The training set should have 2499 observations and the test set should have 834 observations
print(x_train.shape)
print(x_test.shape)
```

```
(2499, 17)
(834, 17)
```

Task 02 (of 14): Standardize the training set and test set. *Hint:* Compute the mean and standard deviation using only the training set and then apply this transformation on the training set and test set.

In [48]:
```
scaler = StandardScaler()
scaler.fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

Task 03 (of 14): Build a decision tree classifier to classify customers as churnend/non-churned. *Hint:* Use entropy as the split criterion.

In [65]:
```
classifier = DecisionTreeClassifier(criterion = "entropy")
classifier.fit(x_train_scaled, y_train)
```

Out[65]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                       splitter='best')
```

In [66]:
```
# Show the structure of the decision tree classifier
print(classifier.tree_.__getstate__()['nodes'])
len(classifier.tree_.__getstate__()['nodes'])
```

```
[(  1, 242,  4,  1.37790751e+00, 0.60293799, 2499, 2.499e+03)
 (  2, 203, 16,  1.47932553e+00, 0.4991475 , 2278, 2.278e+03)
 (  3, 190,  1,  1.39084876e+00, 0.37934172, 2105, 2.105e+03)
 (  4, 139,  6,  7.91448593e-01, 0.26832186, 1921, 1.921e+03)
 (  5,   6, 12, -1.40246248e+00, 0.1779243 , 1681, 1.681e+03)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,  124, 1.240e+02)
 (  7,   8, 10, -1.40125608e+00, 0.18885385, 1557, 1.557e+03)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
 (  9,  28,  9, -6.43147230e-01, 0.18567934, 1556, 1.556e+03)
 ( 10,  27,  9, -1.12710643e+00, 0.09482908,  411, 4.110e+02)
 ( 11,  26, 15,  5.71940318e-02, 0.16417121,  207, 2.070e+02)
 ( 12,  23, 15, -2.16688029e-03, 0.29071587,   98, 9.800e+01)
 ( 13,  18,  3,  1.93759680e+00, 0.20390588,   94, 9.400e+01)
 ( 14,  17,  0, -1.88109088e+00, 0.09140162,   86, 8.600e+01)
 ( 15,  16,  9, -1.78004909e+00, 1.        ,    2, 2.000e+00)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,   84, 8.400e+01)
 ( 19,  22,  8, -1.49175748e-02, 0.81127812,    8, 8.000e+00)
 ( 20,  21,  4,  5.10819435e-01, 0.91829583,    3, 3.000e+00)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,    5, 5.000e+00)
 ( 24,  25, 12,  7.08070576e-01, 1.        ,    4, 4.000e+00)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,  109, 1.090e+02)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,  204, 2.040e+02)
 ( 29,  94, 11,  7.33808935e-01, 0.21436617, 1145, 1.145e+03)
 ( 30,  31, 15, -7.14497805e-01, 0.17017729,  870, 8.700e+02)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,  197, 1.970e+02)
 ( 32,  47,  0, -1.02322608e-01, 0.20770499,  673, 6.730e+02)
 ( 33,  34, 12,  6.40130222e-01, 0.11670201,  318, 3.180e+02)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,  214, 2.140e+02)
 ( 35,  36, 12,  6.53279960e-01, 0.27817101,  104, 1.040e+02)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
 ( 37,  42,  8,  8.06315780e-01, 0.23692475,  103, 1.030e+02)
 ( 38,  39,  5,  2.11814451e+00, 0.09227725,   85, 8.500e+01)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,   83, 8.300e+01)
 ( 40,  41,  0, -6.40963674e-01, 1.        ,    2, 2.000e+00)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
 ( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
 ( 43,  46,  5, -5.44928372e-01, 0.65002242,   18, 1.800e+01)
```

```
( 44,  45,  6, -9.98059690e-01, 0.97095059,    5, 5.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   13, 1.300e+01)
( 48,  55,  0, -2.11031316e-03, 0.27735376,  355, 3.550e+02)
( 49,  54,  7,  1.43707240e+00, 0.73828487,   24, 2.400e+01)
( 50,  51,  8,  1.59283429e-01, 0.5746357 ,   22, 2.200e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   15, 1.500e+01)
( 52,  53, 11, -4.60047245e-01, 0.98522814,    7, 7.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( 56,  77,  8, -6.37064099e-01, 0.22484398,  331, 3.310e+02)
( 57,  66,  0,  1.98314279e-01, 0.40707681,   86, 8.600e+01)
( 58,  61, 11, -5.10849655e-01, 0.83664074,   15, 1.500e+01)
( 59,  60,  4,  8.18777606e-02, 0.81127812,    4, 4.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    3, 3.000e+00)
( 62,  65, 12, -5.32388151e-01, 0.43949699,   11, 1.100e+01)
( 63,  64,  5,  7.72849321e-02, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    9, 9.000e+00)
( 67,  72,  5,  1.02304912e+00, 0.25253077,   71, 7.100e+01)
( 68,  69,  4,  6.59936428e-01, 0.12741851,   57, 5.700e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   54, 5.400e+01)
( 70,  71, 11, -1.04427481e+00, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( 73,  76,  7,  2.71364868e-01, 0.59167278,   14, 1.400e+01)
( 74,  75,  9, -8.67666304e-02, 0.97095059,    5, 5.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    9, 9.000e+00)
( 78,  93,  0,  3.01678514e+00, 0.14372617,  245, 2.450e+02)
( 79,  80,  7, -6.26610994e-01, 0.12068101,  244, 2.440e+02)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( 81,  92, 10, -6.68515116e-02, 0.0959704 ,  243, 2.430e+02)
( 82,  91, 10, -8.36178660e-02, 0.19143325,  102, 1.020e+02)
( 83,  84,  0,  6.99375749e-01, 0.14032727,  101, 1.010e+02)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   57, 5.700e+01)
( 85,  90,  0,  7.49481916e-01, 0.26676499,   44, 4.400e+01)
( 86,  87, 14, -3.98192674e-01, 0.97095059,    5, 5.000e+00)
```

```
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( 88,  89, 12, -3.63633156e-01, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   39, 3.900e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,  141, 1.410e+02)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( 95, 120, 10,  7.37933517e-01, 0.33462053,  275, 2.750e+02)
( 96, 113,  0,  1.28812289e+00, 0.2085566 ,  213, 2.130e+02)
( 97, 102,  3,  1.42354321e+00, 0.14431028,  195, 1.950e+02)
( 98,  99,  8,  1.25426126e+00, 0.05390791,  163, 1.630e+02)
( -1,  -1, -2, -2.00000000e+00, 0.        ,  146, 1.460e+02)
(100, 101,  4, -1.03005683e+00, 0.32275696,   17, 1.700e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   16, 1.600e+01)
(103, 112, 12, -4.46914852e-01, 0.44886449,   32, 3.200e+01)
(104, 105, 15, -5.49321361e-02, 0.72192809,   15, 1.500e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    7, 7.000e+00)
(106, 107,  8, -2.63776153e-01, 0.954434  ,    8, 8.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
(108, 111, 15,  1.71270394e+00, 1.        ,    6, 6.000e+00)
(109, 110, 10, -9.37715709e-01, 0.81127812,    4, 4.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   17, 1.700e+01)
(114, 119, 12, -1.95483305e-02, 0.65002242,   18, 1.800e+01)
(115, 116,  0,  1.42591488e+00, 0.98522814,    7, 7.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
(117, 118,  5,  1.24704599e+00, 0.72192809,    5, 5.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   11, 1.100e+01)
(121, 138, 11,  1.49584484e+00, 0.6373875 ,   62, 6.200e+01)
(122, 125,  7, -4.18483824e-01, 0.81127812,   40, 4.000e+01)
(123, 124, 12,  9.18466389e-01, 0.72192809,    5, 5.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    4, 4.000e+00)
(126, 137,  9,  1.98931479e+00, 0.66096234,   35, 3.500e+01)
(127, 130, 12,  8.50526094e-01, 0.53283506,   33, 3.300e+01)
(128, 129,  8,  6.07228875e-01, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
```

```
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
(131, 136,  5, -1.71600401e-01, 0.35335934,   30, 3.000e+01)
(132, 133,  5, -4.95151341e-01, 0.65002242,   12, 1.200e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    8, 8.000e+00)
(134, 135,  4, -1.53764009e-01, 1.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   18, 1.800e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   22, 2.200e+01)
(140, 181,  9,  1.13635111e+00, 0.67825063,  240, 2.400e+02)
(141, 176,  7,  8.43958855e-01, 0.41666476,  202, 2.020e+02)
(142, 175,  5,  4.50612903e-01, 0.34918437,  183, 1.830e+02)
(143, 144,  0, -2.01888275e+00, 0.44412605,  130, 1.300e+02)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
(145, 146,  4,  7.96166897e-01, 0.42048596,  129, 1.290e+02)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
(147, 174,  0,  9.74959612e-01, 0.39553781,  128, 1.280e+02)
(148, 171,  5,  4.00835812e-01, 0.45969421,  103, 1.030e+02)
(149, 150, 15, -1.01130235e+00, 0.40502013,   99, 9.900e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   18, 1.800e+01)
(151, 152,  0, -1.90614390e+00, 0.46506984,   81, 8.100e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
(153, 170, 13,  5.00371635e-01, 0.42806963,   80, 8.000e+01)
(154, 167, 13,  3.93526316e-01, 0.59167278,   49, 4.900e+01)
(155, 156, 10, -1.39632487e+00, 0.49596907,   46, 4.600e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
(157, 158, 12, -2.38710642e-01, 0.43275016,   45, 4.500e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   17, 1.700e+01)
(159, 160, 10, -1.99009836e-01, 0.59167278,   28, 2.800e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
(161, 162,  7, -4.13598210e-01, 0.50325833,   27, 2.700e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   15, 1.500e+01)
(163, 166, 10,  1.08706820e+00, 0.81127812,   12, 1.200e+01)
(164, 165, 14, -8.07439446e-01, 0.46899559,   10, 1.000e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    9, 9.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
(168, 169,  9, -6.38549089e-01, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   31, 3.100e+01)
(172, 173,  7, -7.43865728e-01, 1.        ,    4, 4.000e+00)
```

```
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   25, 2.500e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   53, 5.300e+01)
(177, 178, 10,  6.82655796e-02, 0.83147439,   19, 1.900e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   10, 1.000e+01)
(179, 180,  3,  2.48563558e-01, 0.99107606,    9, 9.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    5, 5.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    4, 4.000e+00)
(182, 189,  2,  4.90775019e-01, 0.89974376,   38, 3.800e+01)
(183, 188,  9,  1.31568050e+00, 0.56650951,   30, 3.000e+01)
(184, 185,  7,  1.19572306e+00, 0.99107606,    9, 9.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    4, 4.000e+00)
(186, 187, 12,  4.84524995e-01, 0.72192809,    5, 5.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   21, 2.100e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    8, 8.000e+00)
(191, 192, 14, -8.07439446e-01, 0.94605843,  184, 1.840e+02)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   36, 3.600e+01)
(193, 202, 13,  1.01679075e+00, 0.74044825,  148, 1.480e+02)
(194, 201,  9,  2.40889931e+00, 0.24678396,  122, 1.220e+02)
(195, 198,  4,  1.32083797e+00, 0.16866093,  120, 1.200e+02)
(196, 197, 12, -2.31198621e+00, 0.07099895,  117, 1.170e+02)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,  116, 1.160e+02)
(199, 200,  5,  1.14749181e+00, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   26, 2.600e+01)
(204, 215,  4, -3.67314339e-01, 0.99302328,  173, 1.730e+02)
(205, 210,  7,  6.19220614e-01, 0.37123233,   70, 7.000e+01)
(206, 207,  8,  2.29946733e+00, 0.12741851,   57, 5.700e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   55, 5.500e+01)
(208, 209, 15, -2.00036585e-01, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
(211, 212,  4, -8.38597894e-01, 0.89049164,   13, 1.300e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    7, 7.000e+00)
(213, 214, 15,  2.48468071e-01, 0.91829583,    6, 6.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
```

```
(216, 221,  9, -8.99496198e-01, 0.87034605,  103, 1.030e+02)
(217, 218,  0,  8.12114596e-01, 0.35335934,   15, 1.500e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   13, 1.300e+01)
(219, 220,  2,  4.90775019e-01, 1.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
(222, 227,  6, -1.21444270e-01, 0.68403844,   88, 8.800e+01)
(223, 226,  9,  2.02919155e-01, 0.99277445,   20, 2.000e+01)
(224, 225,  6, -3.42357844e-01, 0.46899559,   10, 1.000e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    9, 9.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   10, 1.000e+01)
(228, 237,  1,  1.39084876e+00, 0.47825016,   68, 6.800e+01)
(229, 230, 11,  4.03593391e-01, 0.21357982,   59, 5.900e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   42, 4.200e+01)
(231, 236, 14, -3.98192674e-01, 0.52255937,   17, 1.700e+01)
(232, 233,  7,  1.55087262e-01, 0.97095059,    5, 5.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
(234, 235, 10, -1.33616328e+00, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   12, 1.200e+01)
(238, 241,  9,  7.24813402e-01, 0.99107606,    9, 9.000e+00)
(239, 240, 12,  1.42631784e-01, 0.72192809,    5, 5.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    4, 4.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    4, 4.000e+00)
(243, 278,  2,  4.90775019e-01, 0.99667435,  221, 2.210e+02)
(244, 263,  7, -2.56804619e-02, 0.92594006,  170, 1.700e+02)
(245, 262,  6,  2.41256452e+00, 0.97663491,   78, 7.800e+01)
(246, 251, 12, -1.29129484e-01, 0.92752659,   70, 7.000e+01)
(247, 248,  4,  1.90441966e+00, 0.36205125,   29, 2.900e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   24, 2.400e+01)
(249, 250,  5, -1.04269910e+00, 0.97095059,    5, 5.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    3, 3.000e+00)
(252, 253,  7, -1.21581602e+00, 0.99613448,   41, 4.100e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    8, 8.000e+00)
(254, 261,  6,  1.72329235e+00, 0.91829583,   33, 3.300e+01)
(255, 260, 10,  1.57132006e+00, 0.99836367,   21, 2.100e+01)
(256, 257,  9, -4.28182006e-01, 0.89603823,   16, 1.600e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    9, 9.000e+00)
(258, 259,  8, -3.63319606e-01, 0.86312057,    7, 7.000e+00)
```

```
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    5, 5.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    5, 5.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   12, 1.200e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    8, 8.000e+00)
(264, 271, 12, -8.72089744e-01, 0.55862937,   92, 9.200e+01)
(265, 270,  6,  1.78555965e+00, 0.99750255,   17, 1.700e+01)
(266, 267, 14,  1.10540837e-02, 0.68403844,   11, 1.100e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    8, 8.000e+00)
(268, 269,  4,  1.58317351e+00, 0.91829583,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    6, 6.000e+00)
(272, 277,  4,  1.48468268e+00, 0.24229219,   75, 7.500e+01)
(273, 276, 14, -8.07439446e-01, 0.81127812,   12, 1.200e+01)
(274, 275, 12,  3.46452713e-01, 0.97095059,    5, 5.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    2, 2.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    7, 7.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   63, 6.300e+01)
(279, 282,  1,  1.39084876e+00, 0.52255937,   51, 5.100e+01)
(280, 281,  6,  2.53980637e+00, 0.15935006,   43, 4.300e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,   42, 4.200e+01)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)
(283, 284,  4,  1.77003014e+00, 0.954434  ,    8, 8.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    4, 4.000e+00)
(285, 286, 15,  1.12569046e+00, 0.81127812,    4, 4.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    3, 3.000e+00)
( -1,  -1, -2, -2.00000000e+00, 0.        ,    1, 1.000e+00)]
```

Out[66]: 287

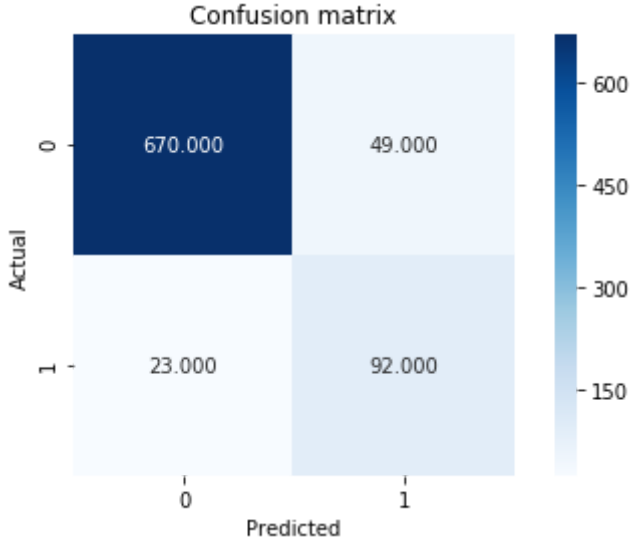
Question 01 (of 06): How many nodes are in the tree? Which variable was selected to split the root node of the tree? What can you conclude from observing the structure of the tree?

Answer: There are 287 nodes. The root node was split on Total Day Minutes. The conclusion made about this is that the initial splitting greedily splits the first root node from 2499 to 2278 and remainder, meaning that this is a greedy approach. Because of this, we can assert that Total Day Minutes is a more relevant and decisive splitting attribute than the remainder of the attributes.

Task 04 (of 14): Predict the class labels for the test set using the decision tree classifier and plot the corresponding confusion matrix.

In [67]:
```
y_pred = classifier.predict(x_test_scaled)
```

In [68]:
```
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



Task 05 (of 14): Compute evaluation metrics for the decision tree classifier.

In [69]:
```
accuracy = metrics.accuracy_score(y_test, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test, y_pred, average = None)
recall = metrics.recall_score(y_test, y_pred, average = None)
F1_score = metrics.f1_score(y_test, y_pred, average = None )
print([accuracy, error, precision, recall, F1_score])
```

```
[0.9136690647482014, 0.08633093525179858, array([0.96681097, 0.65248227]), array([0.93184979, 0.8
]), array([0.9490085, 0.71875  ])]
```

Question 02 (of 06): What can you conclude about the performance of the decision tree classifier?

Answer: The accuracy is high and the error is low. For the first metric of 0, the precision and recall are very high, as well as the f1 score. The f1 score performs very well, and so we can say the decision tree performs well. However, it is noted that this was run several times and the f1 score previously did not perform well. This is likely due to the variability of decision trees and so a random forest would be a more efficient check on the model.

Part 2: k-Nearest Neighbors

Task 06 (of 14): Build a k-nearest neighbors classifier to classify customers as churnend/non-churned. *Hint:* Use *k=3* as the number of nearest neighbors.

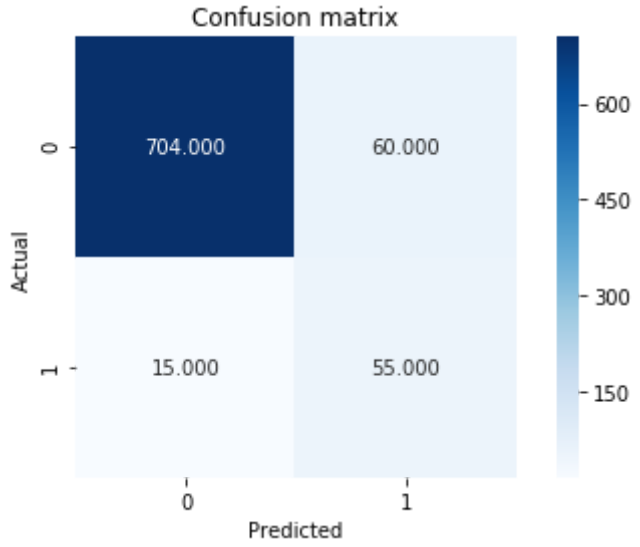
In [27]:
```
classifier = KNeighborsClassifier(n_neighbors = 3)
classifier.fit(x_train_scaled, y_train)
```

Out[27]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=1, n_neighbors=3, p=2,
                     weights='uniform')
```

Task 07 (of 14): Predict the class labels for the test set using the k-nearest neighbors classifier and plot the corresponding confusion matrix.

In [28]:
```
y_pred = classifier.predict(x_test_scaled)
```

In [29]:
```
conf_matrix = metrics.confusion_matrix(y_pred, y_test)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



Task 08 (of 14): Compute evaluation metrics for the k-nearest neighbors classifier.

In [30]:
```
accuracy = metrics.accuracy_score(y_test, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test, y_pred, average = None)
recall = metrics.recall_score(y_test, y_pred, average = None)
F1_score = metrics.f1_score(y_test, y_pred, average = None )
print([accuracy, error, precision, recall, F1_score])
```

```
[0.9100719424460432, 0.08992805755395683, array([0.92146597, 0.78571429]), array([0.97913769, 0.4782
6087]), array([0.94942684, 0.59459459])]
```

Question 03 (of 06): What can you conclude about the performance of the k-nearest neighbors classifier?

Answer: The k-nearest neighbors model performs very accurately with low error. Overall, this performs worwse than our single decision tree model. There is room for improvement on our recall, which does lower our f1 score.

Part 3: Naive Bayes

Task 09 (of 14): Build a Naive Bayes classifier to classify customers as churnend/non-churned.

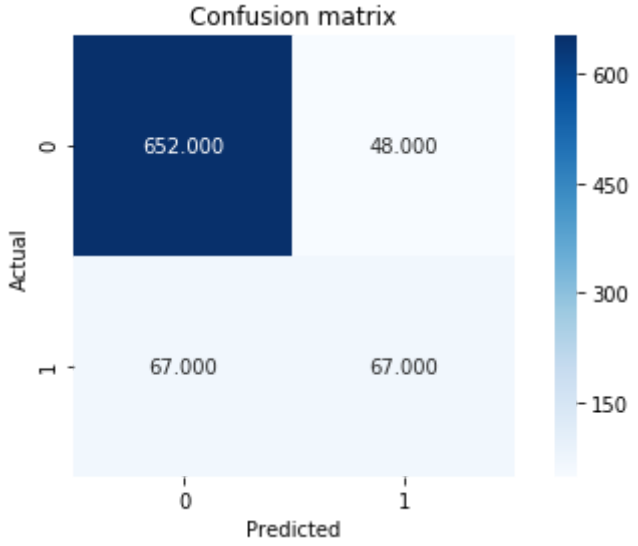
In [34]:
```
classifier = GaussianNB()
classifier.fit(x_train_scaled, y_train)
```

Out[34]:
```
GaussianNB(priors=None)
```

Task 10 (of 14): Predict the class labels for the test set using the Naive Bayes classifier and plot the corresponding confusion matrix.

In [35]:
```
y_pred = classifier.predict(x_test_scaled)
```

In [36]:
```
conf_matrix = metrics.confusion_matrix(y_pred, y_test)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



Task 11 (of 14): Compute evaluation metrics for the Naive Bayes classifier.

In [37]:
```
accuracy = metrics.accuracy_score(y_test, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test, y_pred, average = None)
recall = metrics.recall_score(y_test, y_pred, average = None)
F1_score = metrics.f1_score(y_test, y_pred, average = None )
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8621103117505995, 0.1378896882494005, array([0.93142857, 0.5       ]), array([0.90681502, 0.58260
87 ]), array([0.91895701, 0.53815261])]
```

Question 04 (of 06): What can you conclude about the performance of the Naive Bayes classifier?

Answer: The Naive Bayes classifier is less accurate than other models, but still performs very well. It still boasts a good precision, recall, and f1 score. The f1 score performs worse than our decision tree and k nearest neighbors models.

Part 4: Support Vector Machines

Task 12 (of 14): Build an SVM classifier to classify customers as churnend/non-churned. *Hint:* Use *rbf* (radial basis function) as the kernel function.

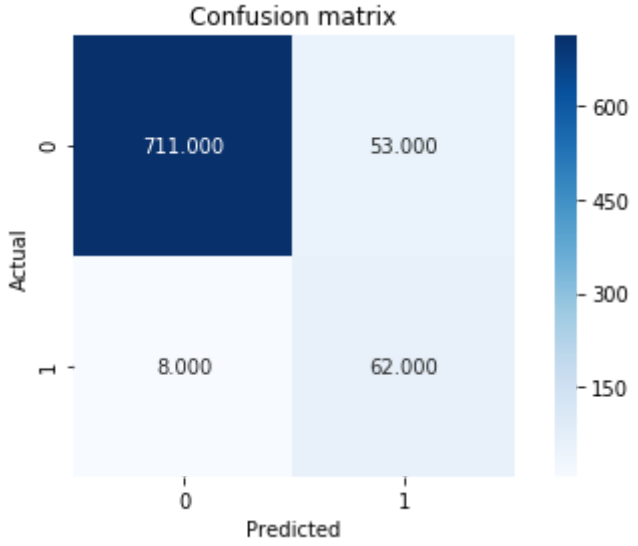
In [38]:
```
classifier = SVC(kernel = 'rbf')
classifier.fit(x_train_scaled, y_train)
```

Out[38]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Task 13 (of 14): Predict the class labels for the test set using the SVM classifier and plot the corresponding confusion matrix.

In [39]:
```
y_pred = classifier.predict(x_test_scaled)
```

In [40]:
```
conf_matrix = metrics.confusion_matrix(y_pred, y_test)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



Task 14 (of 14): Compute evaluation metrics for the SVM classifier.

In [41]:
```
accuracy = metrics.accuracy_score(y_test, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test, y_pred, average = None)
recall = metrics.recall_score(y_test, y_pred, average = None)
F1_score = metrics.f1_score(y_test, y_pred, average = None )
print([accuracy, error, precision, recall, F1_score])
```

```
[0.9268585131894485, 0.07314148681055155, array([0.93062827, 0.88571429]), array([0.98887344, 0.5391
3043]), array([0.95886716, 0.67027027])]
```

Question 05 (of 06): What can you conclude about the performance of the SVM classifier?

Answer: The SVM classifier has very high accuracy, with also very high precision, recall and f1 score in comparison to the other classifiers. This is showing that of the classifiers chosen to test, the SVM has very promising results.

Question 06 (of 06): Which of the classifiers had the best performance?

Answer: The Decision Tree has the best classification performance score. The second is the rbh SVM. However, it should be noted that the decision tree has high variability. Previously the decision tree had significantly worse results. I would have liked to test a random forest against the SVM to confirm if it was better or not.