Table of Contents

Contents

10.2 Results
11 XGBoost
11.1 Results
12 Gaussian N
12.1 Results
13 AdaBoostC
13.1 Results
14 Gradient Bo
14.1 Results
15 SVM Classi
15.1 Tuning
15.1.1 Hyp
15.1.1.1
15.2 Results
15.2.1 SVI
15.2.2 SV
15.3 Results
16 Stacking
16.1 Results
17 SGDClassif
17.1 Results
18 Logistic Re
18.1 Results
19 SGDClassif
19.1 Tuning
19.2 Results
20 Data Interp
20.1 Most In
20.1.1 Top
20.2 Top Po
20.2.1 To
20.2.2 Top
21 Conclusion
21.1 Increas
21.1.1 Tota
21.1.2 Cus
21.1.3 Tota
21.1.4 Tota
21.2 Decrea
21.2.1 Voi
21.2.2 Inte
22 Recomend
23 Future Wor

1 Abstract

Customer churn in the telecomunication industry In this project, machine learning models are made to predict possible customer churn in a telecomunication industry. Our models are powerful which can determine churn factors Therefore, we can give a company the advantage of taking action before losing customers.

2 Business Problem For a Telecommunication Company

1) Determine important factors affection churn.

2) Determine factors increasing churn rate.

3) Determine factors decresing churn rate.

3 Import Libaries

Contents

In [1]:

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

from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split


from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier


from sklearn import tree
from sklearn import svm
from sklearn import ensemble
from sklearn import neighbors
from sklearn import linear_model
from sklearn import metrics
from sklearn import preprocessing

from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score, confusion_matrix, classific
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.model_selection import cross_val_score


from sklearn.model_selection import cross_validate
%matplotlib inline

from IPython.display import Image
import matplotlib as mlp
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import sklearn


from sklearn.model_selection import ShuffleSplit

from sklearn.model_selection import StratifiedKFold

from sklearn.metrics import classification_report

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

Contents

```
import seaborn as sns
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, Standard
from sklearn.model_selection import train_test_split, cross_val_score,
from sklearn.metrics import make_scorer
from sklearn.metrics import classification_report, confusion_matrix, ac
from sklearn.metrics import  f1_score, fbeta_score, r2_score, roc_auc_s

from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve, average_precis

from sklearn.metrics import plot_confusion_matrix

```

Contents

4 Load Data

In [2]:

```
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')

print (df.shape)

```

```
(3333, 21)
```

In [231]:

```
# Load data

df.head(10)
```

Out[231]:

ce ail an	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	c
yes	25	265.1	110	45.07	...	99	16.78	244.7	91	
yes	26	161.6	123	27.47	...	103	16.62	254.4	103	
no	0	243.4	114	41.38	...	110	10.30	162.6	104	
no	0	299.4	71	50.90	...	88	5.26	196.9	89	
no	0	166.7	113	28.34	...	122	12.61	186.9	121	
no	0	223.4	98	37.98	...	101	18.75	203.9	118	
yes	24	218.2	88	37.09	...	108	29.62	212.6	118	
no	0	157.0	79	26.69	...	94	8.76	211.8	96	
no	0	184.5	97	31.37	...	80	29.89	215.8	90	
yes	37	258.6	84	43.96	...	111	18.87	326.4	97	

Contents

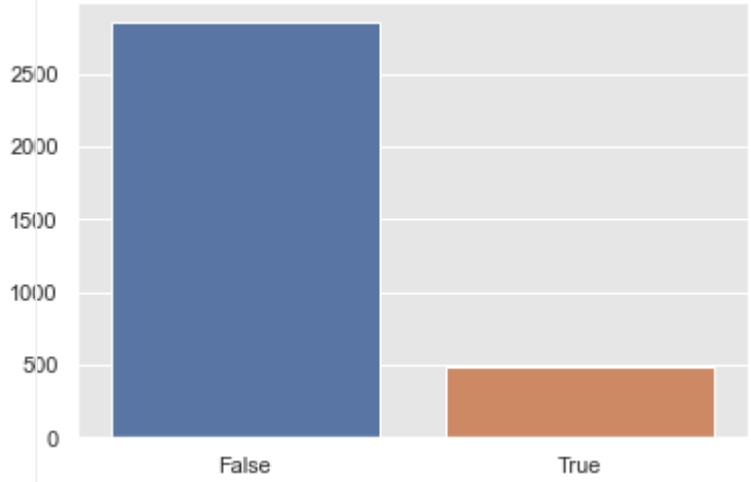
In [232]:
```
1 df.dtypes
```

Out[232]:
```
state                     object
account length             int64
area code                  int64
phone number              object
international plan        object
voice mail plan           object
number vmail messages      int64
total day minutes        float64
total day calls            int64
total day charge         float64
total eve minutes        float64
total eve calls            int64
total eve charge         float64
total night minutes      float64
total night calls          int64
total night charge       float64
total intl minutes       float64
total intl calls           int64
total intl charge        float64
customer service calls     int64
churn                       bool
dtype: object
```


5 Data Exploration & Visualization

In [233]:
```
#Data Visualization

y = df["churn"].value_counts()
#print (y)
sns.barplot(y.index, y.values)

#sns.countplot(x ='churn', data = churn_data)
#plt.show()
```

Out[233]: <AxesSubplot:>

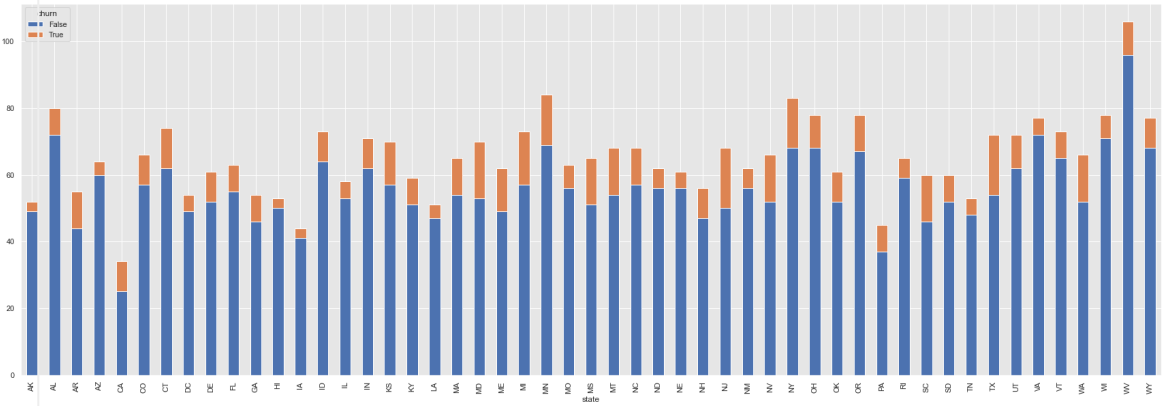


In [234]:
```
y_True = df["churn"][df["churn"] == True]
print ("Churn Percentage = "+str( (y_True.shape[0] / df["churn"].shape[
```

Churn Percentage = 14.491449144914492

In [235]:
```
#Churn by state
df.groupby(["state", "churn"]).size().unstack().plot(kind='bar', stacke
```

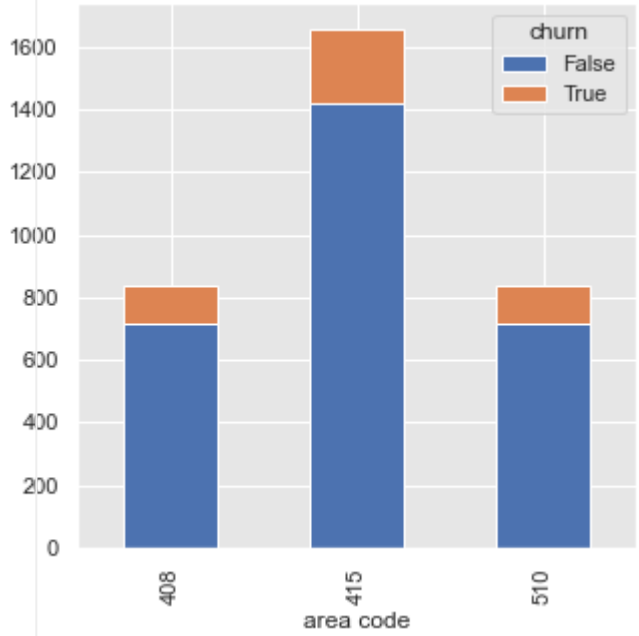
Out[235]: <AxesSubplot:xlabel='state'>



Contents

In [236]:
```
1 #Churn by area code
2 df.groupby(["area code", "churn"]).size().unstack().plot(kind='bar', st
```

Out[236]: <AxesSubplot:xlabel='area code'>



Contents

In [237]:
```
1 df.info()
```

```
<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
```

Contents

In [238]:
```
1 df.dtypes
```

Out[238]:
```
state                    object
account length            int64
area code                 int64
phone number             object
international plan       object
voice mail plan          object
number vmail messages     int64
total day minutes       float64
total day calls           int64
total day charge        float64
total eve minutes       float64
total eve calls           int64
total eve charge        float64
total night minutes     float64
total night calls         int64
total night charge      float64
total intl minutes      float64
total intl calls          int64
total intl charge       float64
customer service calls    int64
churn                      bool
dtype: object
```


In [239]:
```
1 #check for missing values
2 df.isnull().sum()
```

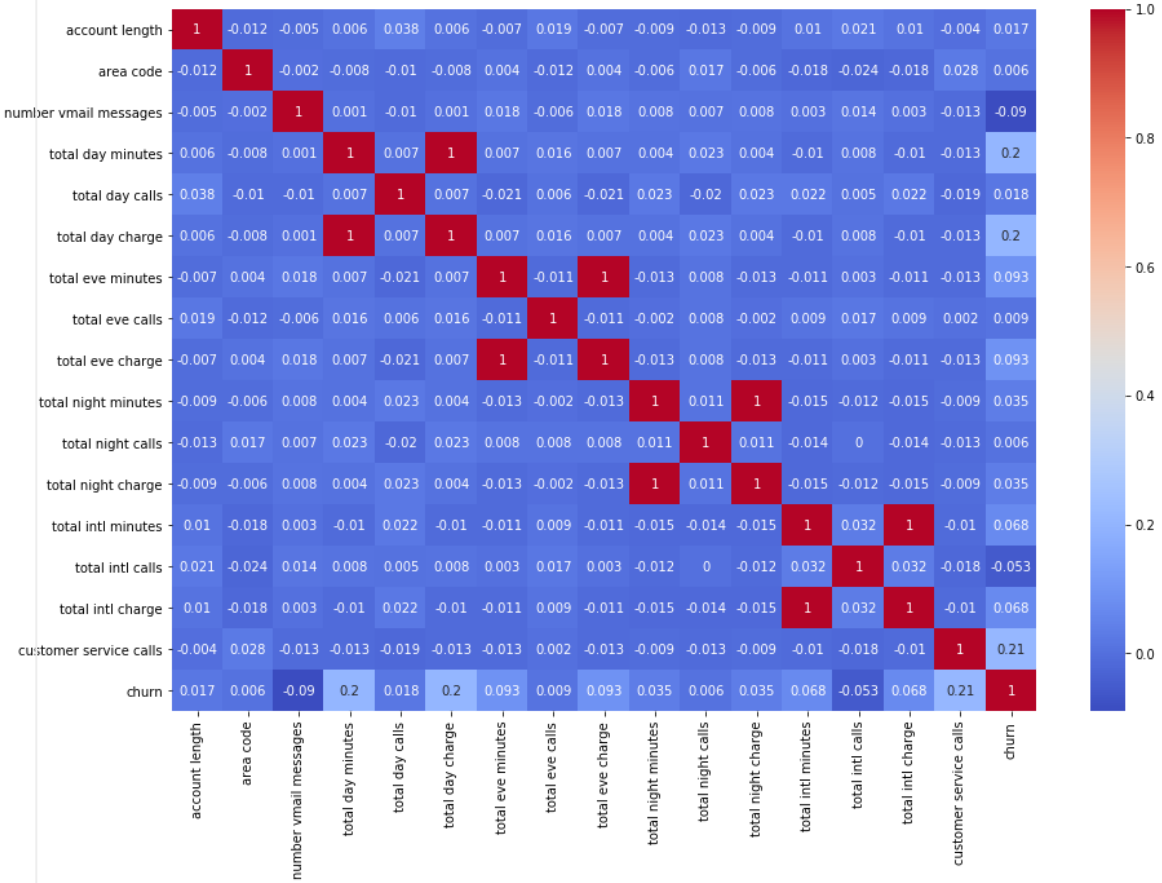
Out[239]:
```
state                   0
account length          0
area code               0
phone number            0
international plan      0
voice mail plan         0
number vmail messages   0
total day minutes       0
total day calls         0
total day charge        0
total eve minutes       0
total 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 [240]:
```
# check for duplicated rows
df.duplicated().sum()
```

Out[240]: 0

Contents

In [15]:
```
#Correlation Analysis
cor = df.corr()
plt.figure(figsize=(15,10))
sns.heatmap(cor.round(3),annot=True,cmap='coolwarm')
plt.show()
```



In [267]:
```
1 df.dtypes
```

Out[267]:
```
state                     object
account length             int64
area code                  int64
phone number              object
international plan        object
voice mail plan           object
number vmail messages      int64
total day minutes        float64
total day calls            int64
total day charge         float64
total eve minutes        float64
total eve calls            int64
total eve charge         float64
total night minutes      float64
total night calls          int64
total night charge       float64
total intl minutes       float64
total intl calls           int64
total intl charge        float64
customer service calls     int64
churn                       bool
dtype: object
```

Contents

In [260]:
```
1 df.corr()
```

Out[260]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	to ca
account length	1.000000	-0.012463	-0.004628	0.006216	0.038470	0.006214	-0.006757	0.0
area code	-0.012463	1.000000	-0.001994	-0.008264	-0.009646	-0.008264	0.003580	-0.0
number vmail messages	-0.004628	-0.001994	1.000000	0.000778	-0.009548	0.000776	0.017562	-0.0
total day minutes	0.006216	-0.008264	0.000778	1.000000	0.006750	1.000000	0.007043	0.0
total day calls	0.038470	-0.009646	-0.009548	0.006750	1.000000	0.006753	-0.021451	0.0
total day charge	0.006214	-0.008264	0.000776	1.000000	0.006753	1.000000	0.007050	0.0
total eve minutes	-0.006757	0.003580	0.017562	0.007043	-0.021451	0.007050	1.000000	-0.0
total eve calls	0.019260	-0.011886	-0.005864	0.015769	0.006462	0.015769	-0.011430	1.0
total eve charge	-0.006745	0.003607	0.017578	0.007029	-0.021449	0.007036	1.000000	-0.0
total night minutes	-0.008955	-0.005825	0.007681	0.004323	0.022938	0.004324	-0.012584	-0.0
total night calls	-0.013176	0.016522	0.007123	0.022972	-0.019557	0.022972	0.007586	0.0
total night charge	-0.008960	-0.005845	0.007663	0.004300	0.022927	0.004301	-0.012593	-0.0
total intl minutes	0.009514	-0.018288	0.002856	-0.010155	0.021565	-0.010157	-0.011035	0.0
total intl calls	0.020661	-0.024179	0.013957	0.008033	0.004574	0.008032	0.002541	0.0
total intl charge	0.009546	-0.018395	0.002884	-0.010092	0.021666	-0.010094	-0.011067	0.0
customer service calls	-0.003796	0.027572	-0.013263	-0.013423	-0.018942	-0.013427	-0.012985	0.0
churn	0.016541	0.006174	-0.089728	0.205151	0.018459	0.205151	0.092796	0.0

Contents

6 Data Engineering

Contents

In [268]:
```
df.groupby(['state', 'churn']).sum().head()
```

Out[268]:

state	churn	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes
AK	False	4640	21567	471	8796.3	4684	1495.39	9088.6
	True	414	1245	0	479.7	270	81.55	494.1
AL	False	7065	31063	545	13235.4	7156	2250.02	13770.8
	True	777	3387	29	1645.4	772	279.70	1866.2
AR	False	4173	19506	367	7721.0	4484	1312.59	8541.4

6.1 Group state column into south and north

In [269]:
```
df['churn'] = df['churn'].astype('str')
```

In [179]:
```
south=["MD", "DE", "VA", "WV", "KY", "TN", "NC", "SC", "FL", "GA", "AL"
df['state_south']=df['state'].isin(south)
```

In [180]:
```
df['state_south'].value_counts()
```

Out[180]:
```
False    2226
True     1107
Name: state_south, dtype: int64
```

In [181]:
```
north=[ "AZ", "CA", "CO", "CT", "DC", "HI", "ID", "IL", "IN", "IA", "KS
df['state_north']=df['state'].isin(north)
```

In [182]:
```
df['state_north'] = df['state_north'].astype('str')
df['state_south'] = df['state_south'].astype('str')
```

In [183]:
```
df.head()
```

Out[183]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes
0	KS	128	415	382-4657	no	yes	25	265.1
1	OH	107	415	371-7191	no	yes	26	161.6
2	NJ	137	415	358-1921	no	no	0	243.4
3	OH	84	408	375-9999	yes	no	0	299.4
4	OK	75	415	330-6626	yes	no	0	166.7

5 rows × 23 columns

6.1.1 Comments

I tried to include state_north and state_southin the models but both features are insignificant. I decided to remove state feature.

Contents

In [332]:
```
1 df.dtypes
```

Out[332]:
```
state                    object
account length            int64
area code                 int64
phone number             object
international plan       object
voice mail plan          object
number vmail messages     int64
total day minutes       float64
total day calls           int64
total day charge        float64
total eve minutes       float64
total eve calls           int64
total eve charge        float64
total night minutes     float64
total night calls         int64
total night charge      float64
total intl minutes      float64
total intl calls          int64
total intl charge       float64
customer service calls    int64
churn                      bool
state_south                bool
state_north                bool
dtype: object
```


6.2 Drop unnecessary columns

In [241]:
```
1 #Remove phonenumber (is not important) ; redundant column
2
3 df=df.drop(['phone number'], axis=1)
```

In [242]:
```
1 df=df.drop(['state'], axis=1)
```

In [243]:
```
1 df=df.drop(['account length'], axis=1 )
```

6.3 Drop higly correlated features

In [244]:
```
1 # We kept total intl charge
2 df=df.drop(['total intl minutes'], axis=1)
```

In [245]:
```
# We kept total night charge
df=df.drop(['total night minutes'], axis=1)
```

In [246]:
```
#We kept total eve charge
df=df.drop(['total eve minutes'], axis=1)

```

In [247]:
```
# We kept total day charge
df=df.drop(['total day minutes'], axis=1)
```

In [46]:
```
df.dtypes
```

Out[46]:
```
state                     object
account length             int64
area code                  int64
phone number              object
international plan        object
voice mail plan           object
number vmail messages      int64
total day minutes        float64
total day calls            int64
total day charge         float64
total eve minutes        float64
total eve calls            int64
total eve charge         float64
total night minutes      float64
total night calls          int64
total night charge       float64
total intl minutes       float64
total intl calls           int64
total intl charge        float64
customer service calls     int64
churn                       bool
dtype: object
```

Contents

In [41]:
```
1 df.head()
```

Out[41]:

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	t cha
0	KS	128	415	no	yes	25	265.1	110	
1	OH	107	415	no	yes	26	161.6	123	
2	NJ	137	415	no	no	0	243.4	114	
3	OH	84	408	yes	no	0	299.4	71	
4	OK	75	415	yes	no	0	166.7	113	

Contents

In [145]:

```
for c in cat_col_names:
    print(df[c].value_counts())
```

```
0    3010
1     323
Name: international plan, dtype: int64
49    106
23     84
34     83
1      80
35     78
48     78
37     78
50     77
45     77
6      74
22     73
13     73
46     73
43     72
44     72
15     71
16     70
20     70
27     68
26     68
31     68
47     66
33     66
5      66
19     65
39     65
25     65
3      64
9      63
24     63
32     62
28     62
21     62
36     61
8      61
29     61
40     60
41     60
17     59
14     58
30     56
2      55
7      54
10     54
42     53
11     53
0      52
18     51
38     45
```

Contents

```
12     44
4      34
Name: state, dtype: int64
0    2411
1     922
Name: voice mail plan, dtype: int64
```


6.4 One Hot Encoder Categorial Columns

In [138]:

```
total_col_names=df.columns

#find numeric columns

num_cols=df._get_numeric_data().columns

# Remove "object"-type features from X
cont_features = [col for col in df.columns if df[col].dtype in [np.floa

# Remove "object"-type features from X_train and X_test
df_cont =df.loc[:, cont_features]

#Category column

cat_col_names=list(set(total_col_names)-set(num_cols))

# Create X_cat which contains only the categorical variables
features_cat = [col for col in df.columns if df[col].dtype in [np.objec
df_cat = df.loc[:, features_cat]

# OneHotEncode categorical variables
ohe = OneHotEncoder(handle_unknown='ignore')

df_cat_ohe = ohe.fit_transform(df_cat)

# Convert these columns into a DataFrame
columns = ohe.get_feature_names(input_features=df_cat.columns)
df_cat = pd.DataFrame(df_cat_ohe.todense(), columns=columns)

```

In [140]:
```
1 df_cont.head()
```

Out[140]:

	area code	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night calls	total night charge	total intl calls	cha
0	415	25	110	45.07	99	16.78	91	11.01	3	
1	415	26	123	27.47	103	16.62	103	11.45	3	
2	415	0	114	41.38	110	10.30	104	7.32	5	
3	408	0	71	50.90	88	5.26	89	8.86	7	
4	415	0	113	28.34	122	12.61	121	8.41	3	

In [49]:
```
1 df_cont.columns
```

Out[49]:
```
Index(['area code', 'number vmail messages', 'total day calls',
       'total day charge', 'total eve calls', 'total eve charge',
       'total night calls', 'total night charge', 'total intl calls',
       'total intl charge', 'customer service calls', 'churn'],
      dtype='object')
```

In [141]:
```
1 df = pd.concat([pd.DataFrame(df_cont), df_cat], axis=1)
```

In [196]:
```
1 df.head()
```

Out[196]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	tota eve charge
0	128	415	25	265.1	110	45.07	197.4	99	16
1	107	415	26	161.6	123	27.47	195.5	103	16
2	137	415	0	243.4	114	41.38	121.2	110	10
3	84	408	0	299.4	71	50.90	61.9	88	5
4	75	415	0	166.7	113	28.34	148.3	122	12

5 rows × 72 columns

In [349]:
```
1 df.columns
```

Out[349]:
```
Index(['area code', 'number vmail messages', 'total day calls',
       'total day charge', 'total eve calls', 'total eve charge',
       'total night calls', 'total night charge', 'total intl calls',
       'total intl charge', 'customer service calls', 'churn',
       'international plan_no', 'international plan_yes', 'voice mail plan_
no',
       'voice mail plan_yes', 'state_south_False', 'state_south_True',
       'state_north_False', 'state_north_True'],
      dtype='object')
```

In [198]:
```
1 df.dtypes
```

Out[198]:
```
account length              int64
area code                   int64
number vmail messages       int64
total day minutes         float64
total day calls             int64
                            ...
state_WY                  float64
international plan_no     float64
international plan_yes    float64
voice mail plan_no        float64
voice mail plan_yes       float64
Length: 72, dtype: object
```

In [50]:
```
df.shape
```

Out[50]: (3333, 16)

In [51]:
```
df.shape
```

Out[51]: (3333, 16)

6.5 Train test split

In [142]:
```
## separate dependent and independent variables
X = df.drop(['churn'], axis=1)
y = df['churn']
```

In [143]:
```
## splitting whole dataset into train and test dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
print(f"Size Of The Train Dataset :- {len(X_train)}")
print(f"Size Of The Test Dataset :- {len(X_test)}")
```

```
Size Of The Train Dataset :- 2666
Size Of The Test Dataset :- 667
```

6.6 Standarize Features

In [144]:
```
scaler = StandardScaler()
scaled_x_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)
scaled_X_train = pd.DataFrame(scaled_x_train, columns=X_train.columns)
```

6.7 Building analysis functions

In [145]:

```
#Function to draw the ROC curve and to calculate the area under the cur
from sklearn.metrics import roc_curve, auc
from sklearn.svm import SVC

def roc_curve_and_auc(clf, X_train, X_test, y_train, y_test):

    # Calculate the probability scores of each point in the training se
    y_train_score = clf.fit(X_train, y_train).decision_function(X_trair

    # Calculate the fpr, tpr, and thresholds for the training set
    train_fpr, train_tpr, thresholds = roc_curve(y_train, y_train_score

    # Calculate the probability scores of each point in the test set
    y_test_score = clf.decision_function(X_test)

    # Calculate the fpr, tpr, and thresholds for the test set
    test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_scor

    # ROC curve for training set
    plt.figure(figsize=(10, 8))
    lw = 2
    plt.plot(train_fpr, train_tpr, color='darkorange',
             lw=lw, label='Train ROC curve')
    plt.plot(test_fpr, test_tpr, color='blue',
             lw=lw, label='Test ROC curve')
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.yticks([i/20.0 for i in range(21)])
    plt.xticks([i/20.0 for i in range(21)])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(
        'Receiver operating characteristic (ROC) Curve for Training and
    plt.legend(loc='lower right')
    plt.show()
    # Print the area under the roc curve
    print('Training AUC: {}'.format(round(auc(train_fpr, train_tpr), 5)
    print('Testing AUC: {}'.format(round(auc(test_fpr, test_tpr), 5)))
```

In [146]:

```
#Plot coefficients
def plot_coefficients(clf):
    weights_clf = pd.Series(clf.coef_[0], index=X.columns.values)
    weights_clf.sort_values(inplace=True)
    plt.figure(figsize=(15, 6))
    plt.xticks(rotation=90)
    features = plt.bar(weights_clf.index, weights_clf.values)
```

Contents

In [147]:

```
#Function to draw precision recall curve
def plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train):
    fig = plt.figure(figsize = (8, 8))
    lw =3
    no_skill = len(y[y==1]) / len(y)
    # plot the no skill precision-recall curve
    plt.plot([0, 1], [no_skill, no_skill], linestyle='--', lw = lw, lab
    # calculate model precision-recall curve
    precision, recall, _ = precision_recall_curve(y_test, y_pred_test)
    train_precision, train_recall, _ = precision_recall_curve(y_train,
    # plot the model precision-recall curve
    plt.plot(recall, precision, marker='.', lw = lw, label='Test Set')
    plt.plot(train_recall,train_precision, marker='.',lw=lw, label='Tra
    # axis labels
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    # show the legend
    plt.legend()

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

Contents

In [148]:

```
#To plot coefficients
def plot_feature_importances(model):
    n_features = X_train.shape[1]
    plt.figure(figsize=(8, 8))
    plt.barh(range(n_features), model.feature_importances_, align='cent
    plt.yticks(np.arange(n_features), X_train.columns.values)
    plt.xlabel('Feature importance')
    plt.ylabel('Feature')
```

In [149]:

```

#Plot confusion matrix
def confusion_matrix_df(y_test, y_pred_test):
    confusion_mat = confusion_matrix(y_test, y_pred_test, labels=[0, 1]
    _row = confusion_mat.sum(axis=0)
    _col = [np.nan] + list(confusion_mat.sum(axis=1)) + [sum(_row)]
    con_df = pd.DataFrame({})
    con_df["Predicted"] = ["Actual"] + ["No Churn", "Churn"] + ["All"]
    for label, idx in {"No Churn": 0, "Churn": 1}.items():
        temp = [np.nan] + list(confusion_mat[:, idx]) + [_row[idx]]
        con_df[label] = temp

    con_df["All"] = _col
    return con_df
```

In [150]:

```
#Function to print the metrics: Confusion Matrix, Classification Report
def model_evaluation(X_train, X_test, y_train, y_test, y_pred_train, y_

    print('MODEL EVALUATION METRICS:\n',
          '-----------------------------------------------------')

    print('Confusion Matrix for train & test set: \n')
    print(confusion_matrix_df(y_train, y_pred_train), '\n')
    print(confusion_matrix_df(y_test, y_pred_test), '\n')

    print('-----------------------------------------------------')
    print('\nClassification Report for train & test set\n',
          '\nTrain set\n',
          classification_report(y_train, y_pred_train),
          '\n\nTest set\n',
          classification_report(y_test, y_pred_test))

    print('-----------------------------------------------------\n')
    print("Cohen's Kappa for train and test set:\n ",
          round(cohen_kappa_score(y_train, y_pred_train), 4),
          round(cohen_kappa_score(y_test, y_pred_test), 4))

    print("f2 score for train and test set: \n ",
          round(fbeta_score(y_train, y_pred_train, 2.0), 4),
          round(fbeta_score(y_test, y_pred_test, 2.0), 4))

    print('roc auc score for train and test set:\n ',
          round(roc_auc_score(y_train, y_pred_train), 4),
          round(roc_auc_score(y_test, y_pred_test), 4))

    print('Mean Cross Validation Score:\n', round(cross_val_score(clf,
```

Contents

7 Logistic Regression

7.1 Optimization and Tuning

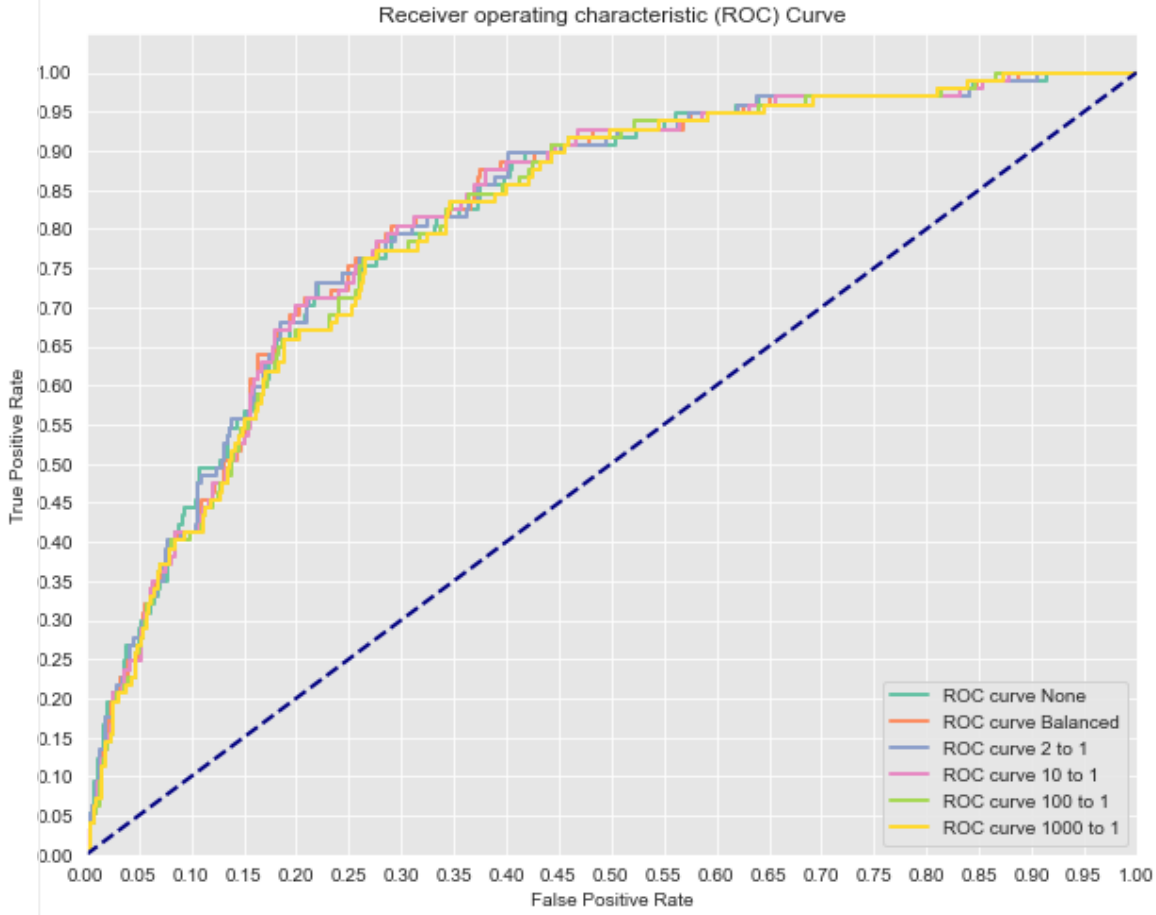
In [61]:

```
# Now let's compare a few different regularization performances on the
weights = [None, 'balanced', {1:2, 0:1}, {1:10, 0:1}, {1:100, 0:1}, {1:
names = ['None', 'Balanced', '2 to 1', '10 to 1', '100 to 1', '1000 to
colors = sns.color_palette('Set2')

plt.figure(figsize=(10,8))

for n, weight in enumerate(weights):
    # Fit a model
    logreg = LogisticRegression(fit_intercept=False, C=1e20, class_weig
    model_log = logreg.fit(scaled_X_train, y_train)
    print(model_log)

    # Predict
    y_hat_test = logreg.predict(scaled_X_test)

    y_score = logreg.fit(scaled_X_train, y_train).decision_function(sca

    fpr, tpr, thresholds = roc_curve(y_test, y_score)

    print('AUC for {}: {}'.format(names[n], auc(fpr, tpr)))
    print('----------------------------------------------------------------
    lw = 2
    plt.plot(fpr, tpr, color=colors[n],
             lw=lw, label='ROC curve {}'.format(names[n]))

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
LogisticRegression(C=1e+20, fit_intercept=False)
AUC for None: 0.81423403870501
------------------------------------------------------------------------
---------------
LogisticRegression(C=1e+20, class_weight='balanced', fit_intercept=Fal
se)
AUC for Balanced: 0.814794718755652
------------------------------------------------------------------------
---------------
LogisticRegression(C=1e+20, class_weight={0: 1, 1: 2}, fit_intercept=F
alse)
AUC for 2 to 1: 0.8150479291011034
------------------------------------------------------------------------
---------------
LogisticRegression(C=1e+20, class_weight={0: 1, 1: 10}, fit_intercept=
False)
AUC for 10 to 1: 0.8137457044673538
```

Contents

```
--------------------------------------------------------------------------
---------------
LogisticRegression(C=1e+20, class_weight={0: 1, 1: 100}, fit_intercept
=False)
AUC for 100 to 1: 0.8077590884427563
--------------------------------------------------------------------------
---------------
LogisticRegression(C=1e+20, class_weight={0: 1, 1: 1000}, fit_intercep
t=False)
AUC for 1000 to 1: 0.8059504431181045
--------------------------------------------------------------------------
---------------
```



In [362]:
```
1 ! pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\mirnamamaranda
\anaconda3\anaconda\lib\site-packages (0.8.0)
Requirement already satisfied: numpy>=1.13.3 in c:\users\mirnamamaranda\ana
conda3\anaconda\lib\site-packages (from imbalanced-learn) (1.19.2)
Requirement already satisfied: scipy>=0.19.1 in c:\users\mirnamamaranda\ana
conda3\anaconda\lib\site-packages (from imbalanced-learn) (1.5.2)
Requirement already satisfied: joblib>=0.11 in c:\users\mirnamamaranda\anac
onda3\anaconda\lib\site-packages (from imbalanced-learn) (0.17.0)
Requirement already satisfied: scikit-learn>=0.24 in c:\users\mirnamamarand
a\anaconda3\anaconda\lib\site-packages (from imbalanced-learn) (0.24.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\mirnamamara
nda\anaconda3\anaconda\lib\site-packages (from scikit-learn>=0.24->imbalanc
ed-learn) (2.1.0)
```

Contents

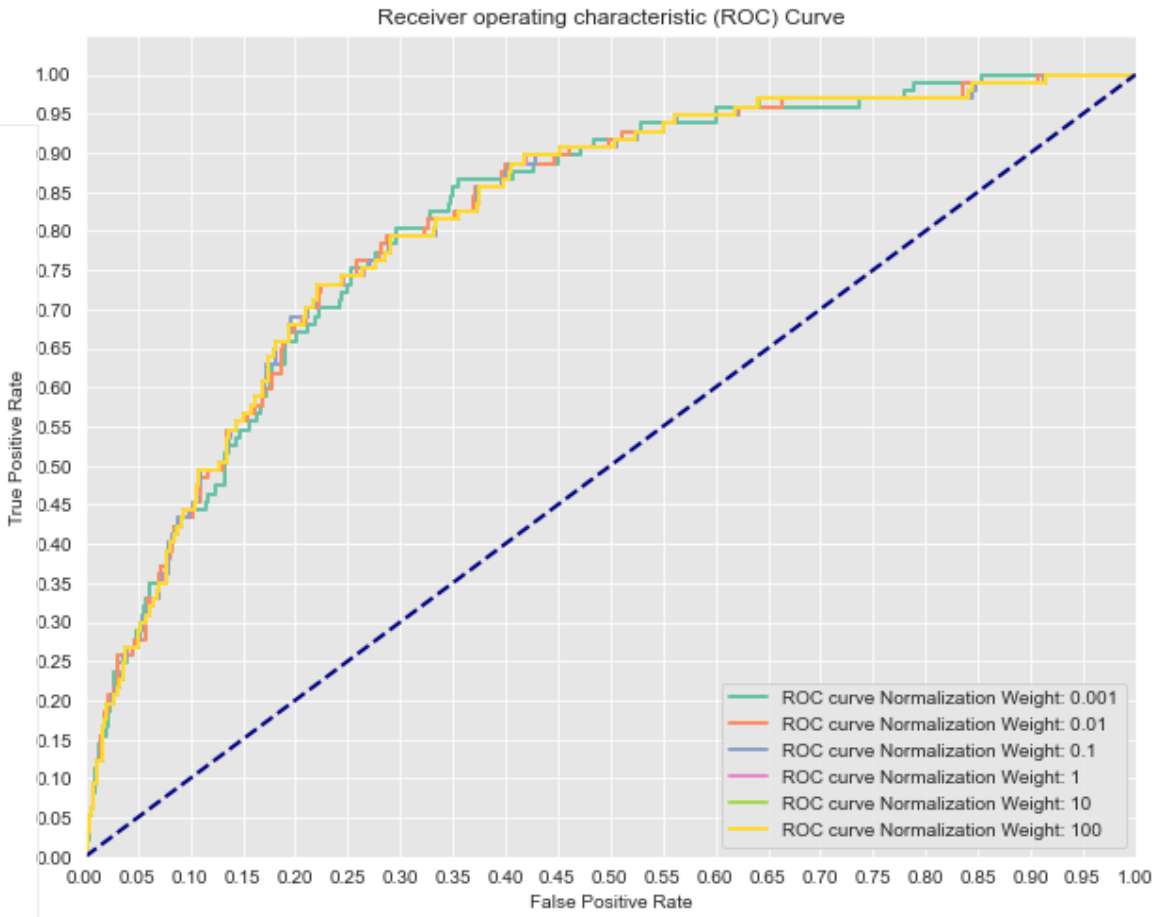
In [63]:

```
# Now let's compare a few different regularization performances on the
C_param_range = [0.001, 0.01, 0.1, 1, 10, 100]
names = [0.001, 0.01, 0.1, 1, 10, 100]
colors = sns.color_palette('Set2')

plt.figure(figsize=(10, 8))

for n, c in enumerate(C_param_range):
    # Fit a model
    logreg = LogisticRegression(fit_intercept=False, C=c, solver='libli
    model_log = logreg.fit(scaled_X_train, y_train)
    print(model_log)
# Preview model params

# Predict
    y_hat_test = logreg.predict(scaled_X_test)

    y_score = logreg.fit(scaled_X_train, y_train).decision_function(sca

    fpr, tpr, thresholds = roc_curve(y_test, y_score)

    print('AUC for {}: {}'.format(names[n], auc(fpr, tpr)))
    print('----------------------------------------------------')
    lw = 2
    plt.plot(fpr, tpr, color=colors[n],
             lw=lw, label='ROC curve Normalization Weight: {}'.format(r

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
LogisticRegression(C=0.001, fit_intercept=False, solver='liblinear')
AUC for 0.001: 0.8137999638270935
----------------------------------------------------
LogisticRegression(C=0.01, fit_intercept=False, solver='liblinear')
AUC for 0.01: 0.8139989148128052
----------------------------------------------------
LogisticRegression(C=0.1, fit_intercept=False, solver='liblinear')
AUC for 0.1: 0.8140893470790378
----------------------------------------------------
LogisticRegression(C=1, fit_intercept=False, solver='liblinear')
AUC for 1: 0.8142159522517635
----------------------------------------------------
LogisticRegression(C=10, fit_intercept=False, solver='liblinear')
AUC for 10: 0.8142340387050099
----------------------------------------------------
LogisticRegression(C=100, fit_intercept=False, solver='liblinear')
```

Contents

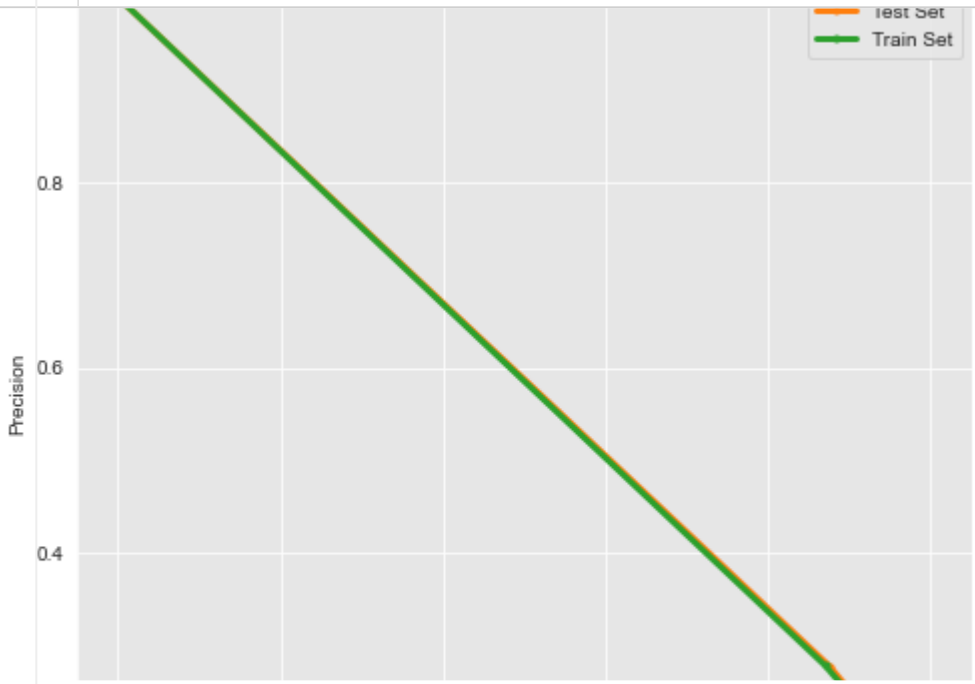
```
AUC for 100: 0.81423403870501
--------------------------------------------------------
```



7.2 Best performing parameters for Logistic Regression Model

In [64]:

```
clf = LogisticRegression(fit_intercept=False, C=100,
                         solver='liblinear', class_weight='balanced')
clf.fit(scaled_X_train, y_train)
y_pred_train = clf.predict(scaled_X_train)
y_pred_test = clf.predict(scaled_X_test)


model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_prec

roc_curve_and_auc(clf, scaled_X_train, scaled_X_test, y_train, y_test)

plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)
```



In [65]:

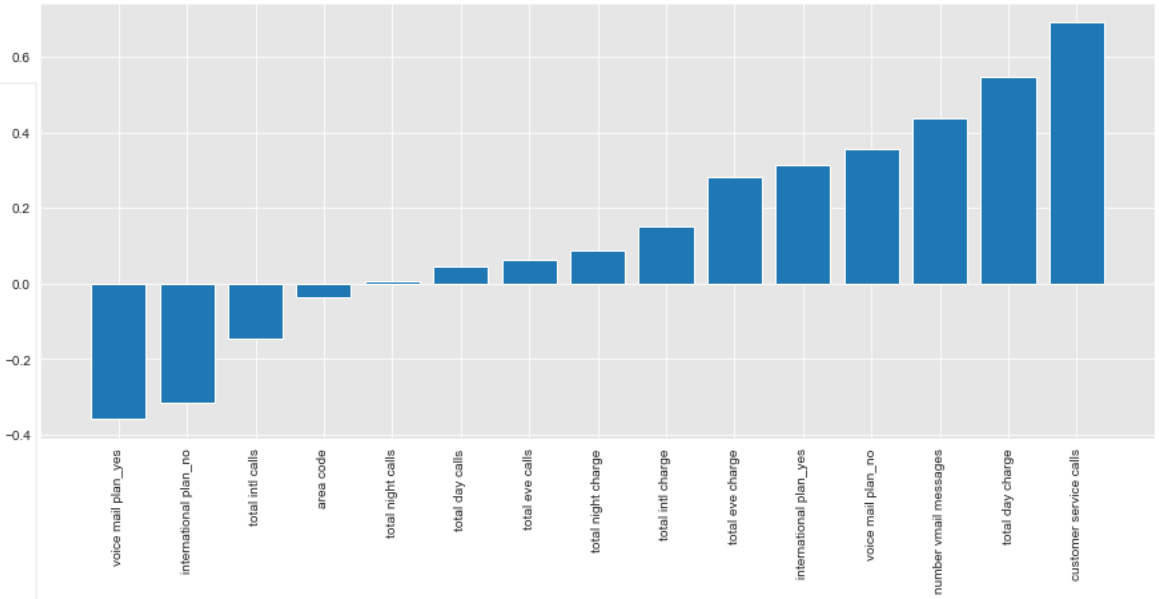
```
clf.coef_.round(2)
```

Out[65]:

```
array([[-0.04,  0.44,  0.05,  0.55,  0.06,  0.28,  0.01,  0.09, -0.15,
         0.15,  0.69, -0.32,  0.32,  0.36, -0.36]])
```

Contents

In [66]:
```
1 plot_coefficients(clf)
```



7.3 Results Comments

The mean cross validation is 76.87%. We have a precision of 0.28 and recall of 0.88 for the test set. We don't have overfitting. It means we are able to detect the churn customers 88% but 28% of the customers we say 'churn' was true.

Most important positive churn factors:

1)Customer service calls

2)Total day charge

3)Number voicemail messages

4)Voice mail plan _no

5)International plan_yes

6)Total evening charge

Most important negative churn features:

1)Voice mail plan _yes

2)International plan_no

3)Total International call

4)Area code

8 Descion Tree Classifier

In [67]:

```
from sklearn.tree import DecisionTreeClassifier


tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=5)
tree_clf.fit(scaled_X_train, y_train)

# Model Performance
# Test set predictions
pred = tree_clf.predict(scaled_X_test)



tree_clf_score = cross_val_score(tree_clf, scaled_X_train, y_train, cv=
mean_tree_clf_score = np.mean(tree_clf_score)

print(f"Mean Cross Validation Score: {mean_tree_clf_score :.2%}")
```

```
Mean Cross Validation Score: 93.96%
```

In [68]:

```
print('Training r^2:', tree_clf.score(scaled_X_train, y_train))
print('Test r^2:', tree_clf.score(scaled_X_test, y_test))

print(classification_report(y_test,pred))
```

```
Training r^2: 0.9538634658664666
Test r^2: 0.9175412293853074
              precision    recall  f1-score   support

       False       0.94      0.96      0.95       570
        True       0.76      0.64      0.69        97

    accuracy                           0.92       667
   macro avg       0.85      0.80      0.82       667
weighted avg       0.91      0.92      0.91       667
```

8.1 Hyperparameter Tuning for decision tree using Gridsearch

In [69]:
```
dt_param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 2, 3, 4, 5, 6],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 3, 4, 5, 6]
}

num_decision_trees = 3 * 2 * 6 * 3 * 3 * 6
print(f"Grid Search will have to search through {num_decision_trees} di
```

```
Grid Search will have to search through 648 different permutations.
```

In [70]:
```
from sklearn.model_selection import train_test_split, GridSearchCV, cro
# Instantiate GridSearchCV
dt_grid_search = GridSearchCV(tree_clf, dt_param_grid, cv=10, return_tr

# Fit to the data
dt_grid_search.fit(scaled_X_train, y_train)

#Examine best parameters

# Mean training score
dt_gs_training_score = np.mean(dt_grid_search.cv_results_['mean_train_s

# Mean test score
dt_gs_testing_score = dt_grid_search.score(scaled_X_test, y_test)

print(f"Mean Training Score: {dt_gs_training_score :.2%}")
print(f"Mean Test Score: {dt_gs_testing_score :.2%}")
print("Best Parameter Combination Found During Grid Search:")
dt_grid_search.best_params_
```

```
Mean Training Score: 93.67%
Mean Test Score: 92.80%
Best Parameter Combination Found During Grid Search:
```

Out[70]:
```
{'criterion': 'gini',
 'max_depth': 6,
 'min_samples_leaf': 4,
 'min_samples_split': 10}
```

8.1.1 Select Best Parameters for Descion tree classifier

Contents

In [71]:

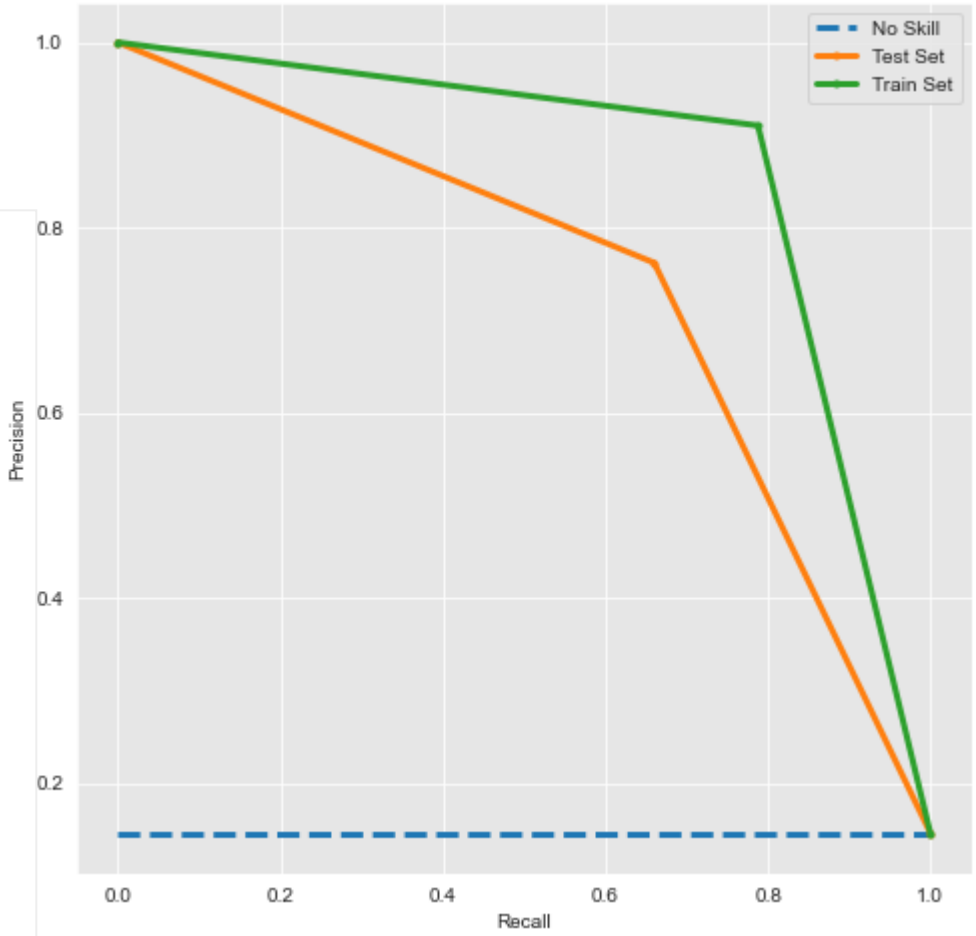
```
dt_param_grid2 = {
    'criterion': ['gini'],
    'max_depth': [ 6],
    'min_samples_split': [4],
    'min_samples_leaf': [10]
}

dt_grid_search2 = GridSearchCV(tree_clf, dt_param_grid2, cv=10, return_

# Fit to the data

clf=dt_grid_search2 .fit(scaled_X_train, y_train)



#Examine best parameters

# Mean training score
dt_gs_training_score2 = np.mean(clf.cv_results_['mean_train_score'])

# Mean test score
dt_gs_testing_score2 = clf.score(scaled_X_test, y_test)


print(f"Mean Training Score: {dt_gs_training_score2 :.2%}")
print(f"Mean Test Score: {dt_gs_testing_score2 :.2%}")

# Model Performance
# Test set predictions
y_pred_test =clf.predict(scaled_X_test)
y_pred_train =clf.predict(scaled_X_train)

print(classification_report(y_test,y_pred_test))

model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_pred



plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)

plot_feature_importances(tree_clf)
```

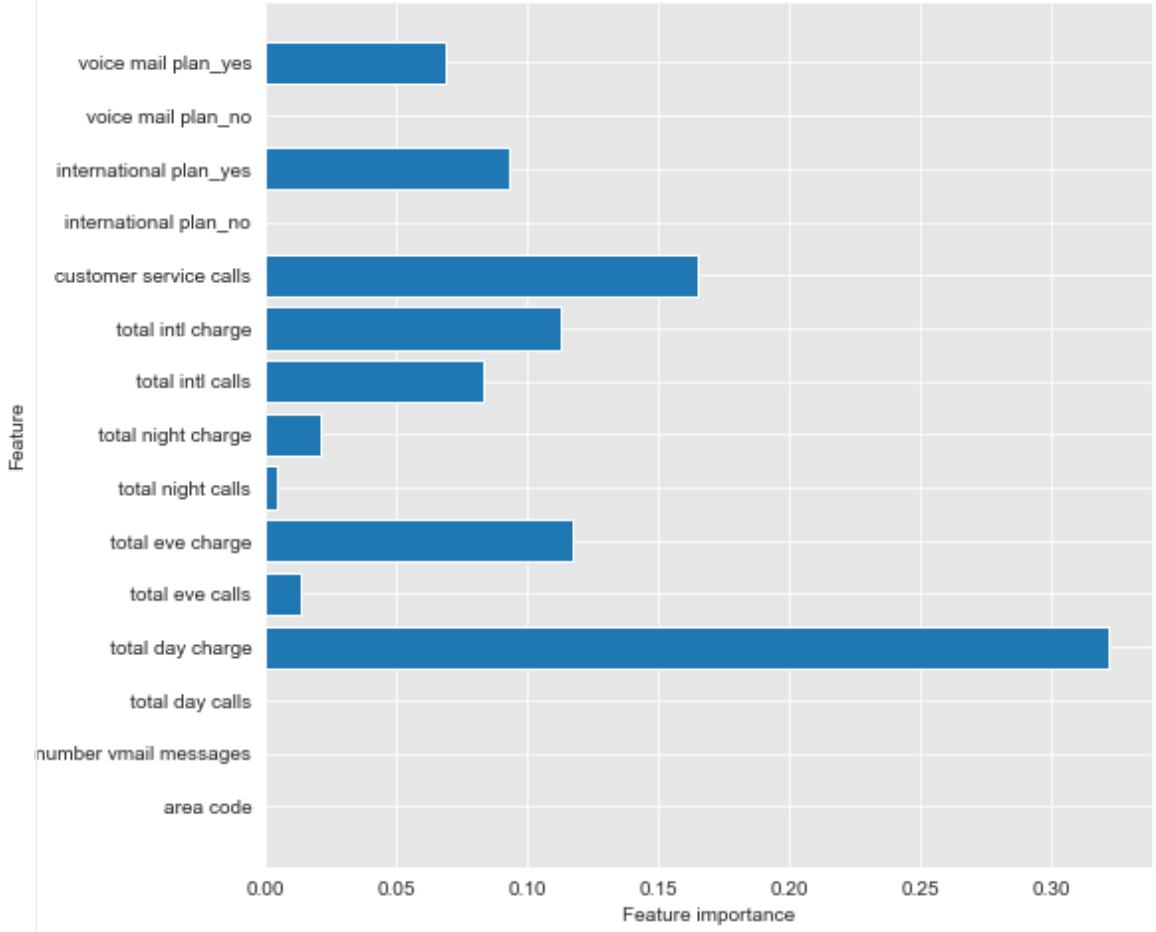
Contents


```
Mean Training Score: 95.79%
Mean Test Score: 92.05%
              precision    recall  f1-score   support

       False       0.94      0.96      0.95       570
        True       0.76      0.66      0.71        97

    accuracy                           0.92       667
   macro avg       0.85      0.81      0.83       667
weighted avg       0.92      0.92      0.92       667

MODEL EVALUATION METRICS:
 -----------------------------------------------------
```

```
Confusion Matrix for train & test set:

  Predicted  No Churn  Churn     All
0    Actual       NaN    NaN     NaN
1  No Churn    2250.0   30.0  2280.0
2     Churn      82.0  304.0   386.0
3       All    2332.0  334.0  2666.0

  Predicted  No Churn  Churn    All
0    Actual       NaN    NaN    NaN
1  No Churn     550.0   20.0  570.0
2     Churn      33.0   64.0   97.0
3       All     583.0   84.0  667.0

----------------------------------------------------

Classification Report for train & test set

Train set
              precision    recall  f1-score   support

       False       0.96      0.99      0.98      2280
        True       0.91      0.79      0.84       386

    accuracy                           0.96      2666
   macro avg       0.94      0.89      0.91      2666
weighted avg       0.96      0.96      0.96      2666


Test set
              precision    recall  f1-score   support

       False       0.94      0.96      0.95       570
        True       0.76      0.66      0.71        97

    accuracy                           0.92       667
   macro avg       0.85      0.81      0.83       667
weighted avg       0.92      0.92      0.92       667

----------------------------------------------------

Cohen's Kappa for train and test set:
  0.8203 0.6615
f2 score for train and test set:
  0.8094 0.678
roc auc score for train and test set:
  0.8872 0.8124
Mean Cross Validation Score:
 0.9373
```

Contents





Contents

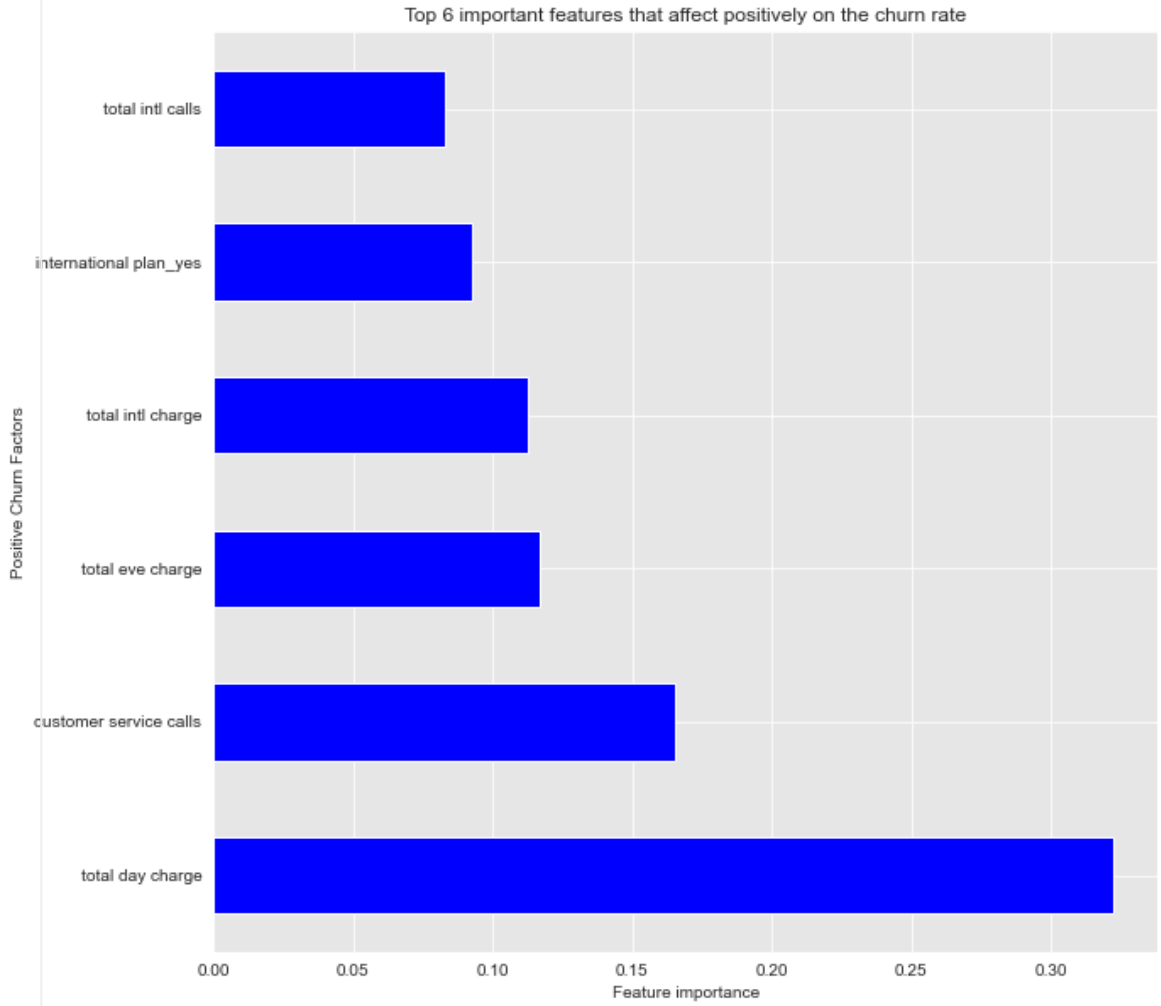
In [72]:

```
# Get Feature Importance from the classifier

feature_importance = tree_clf.feature_importances_
print (tree_clf.feature_importances_)
feat_importances = pd.Series(tree_clf.feature_importances_, index=X.col
feat_importances = feat_importances.nlargest(6)
feat_importances.plot(kind='barh' , figsize=(10,10), color="b")

plt.xlabel('Feature importance')
plt.ylabel('Positive Churn Factors')
plt.title('Top 6 important features that affect positively on the churr
```

```
[0.         0.         0.         0.32193796 0.0134261  0.11687289
 0.00422496 0.02131784 0.08281552 0.11242512 0.1652558  0.
 0.0928464  0.         0.0688774 ]
```

Out[72]:

```
Text(0.5, 1.0, 'Top 6 important features that affect positively on the chur
n rate')
```



8.2 Results Comments

We have a precision of 0.87 and recall of 0.65 for the test set. We don't have overfitting. It means we are able to detect the churn customers 65% but 87% of the customers we say 'churn' was true.

The top 5 factors affecting te churn rate:

1) Total day charge 2) Customer service calls 3) Total evening charge 4) Total international charge
3) Total International plan_yes 4) Total international calls

Contents

9 Random Forest Classifier

In [76]:

```
rf_clf = RandomForestClassifier()
rf_clf .fit(scaled_X_train, y_train)

# Model Performance
# Test set predictions
pred = rf_clf .predict(scaled_X_test)



rf_clf_score = cross_val_score(rf_clf, scaled_X_train, y_train, cv=10)
mean_rf_clf_score = np.mean(tree_clf_score)

print(f"Mean Cross Validation Score: {mean_rf_clf_score :.2%}")
```

```
Mean Cross Validation Score: 93.96%
```

In [77]:

```
print('Training r^2:', tree_clf.score(scaled_X_train, y_train))
print('Test r^2:', tree_clf.score(scaled_X_test, y_test))

print(classification_report(y_test,pred))
```

```
Training r^2: 0.9538634658664666
Test r^2: 0.9175412293853074
              precision    recall  f1-score   support

       False       0.94      0.98      0.96       570
        True       0.87      0.60      0.71        97

    accuracy                           0.93       667
   macro avg       0.90      0.79      0.83       667
weighted avg       0.92      0.93      0.92       667
```

9.1 Hyperparameter Tuning for Random Forest using Gridsearch

In [78]:

```
rf_param_grid = {
    'n_estimators': [10, 30, 100],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 2, 6, 10],
    'min_samples_split': [5, 10],
    'min_samples_leaf': [3, 6]
}

rf_grid_search = GridSearchCV(rf_clf, dt_param_grid, cv=10, return_trai

# Fit to the data
rf_grid_search.fit(scaled_X_train, y_train)

#Examine best parameters

# Mean training score
rf_gs_training_score = np.mean(rf_grid_search.cv_results_['mean_train_s

# Mean test score
rf_gs_testing_score = rf_grid_search.score(scaled_X_test, y_test)

print(f"Mean Training Score: {rf_gs_training_score :.2%}")
print(f"Mean Test Score: {rf_gs_testing_score :.2%}")
print("Best Parameter Combination Found During Grid Search:")
rf_grid_search.best_params_
```



```
Mean Training Score: 91.03%
Mean Test Score: 93.85%
Best Parameter Combination Found During Grid Search:
```

Out[78]:

```
{'criterion': 'entropy',
 'max_depth': None,
 'min_samples_leaf': 1,
 'min_samples_split': 5}
```

9.2 Best Paramerts for Random Forest

In [79]:

```
dt_param_grid2 = {
    'criterion': ['entropy'],
    'max_depth': [ None],
    'min_samples_split': [5],
    'min_samples_leaf': [1]
}

dt_grid_search2 = GridSearchCV(rf_clf, dt_param_grid2, cv=10, return_tr

# Fit to the data

clf=dt_grid_search2.fit(scaled_X_train, y_train)



#Examine best parameters

# Mean training score
dt_gs_training_score2 = np.mean(clf.cv_results_['mean_train_score'])

# Mean test score
dt_gs_testing_score2 = clf.score(scaled_X_test, y_test)


print(f"Mean Training Score: {dt_gs_training_score2 :.2%}")
print(f"Mean Test Score: {dt_gs_testing_score2 :.2%}")

# Model Performance
# Test set predictions
y_pred_test =clf.predict(scaled_X_test)
y_pred_train =clf.predict(scaled_X_train)
```

```
Mean Training Score: 98.82%
Mean Test Score: 93.25%
```

Contents

In [80]:

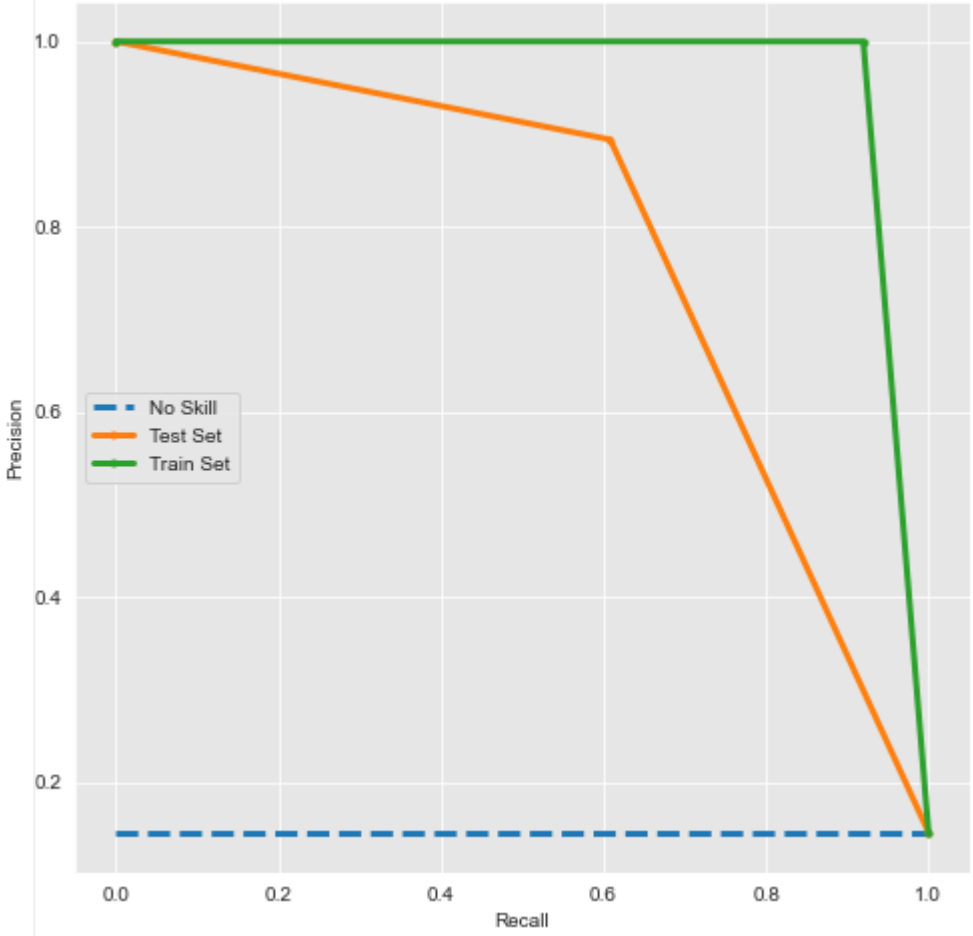
```
print(classification_report(y_test,y_pred_test))

model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_pred

plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)

plot_feature_importances(rf_clf)
```



```
              precision    recall  f1-score   support

       False       0.94      0.99      0.96       570
        True       0.89      0.61      0.72        97

    accuracy                           0.93       667
   macro avg       0.92      0.80      0.84       667
weighted avg       0.93      0.93      0.93       667

MODEL EVALUATION METRICS:
 -----------------------------------------------------
Confusion Matrix for train & test set:

  Predicted  No Churn  Churn     All
0    Actual       NaN    NaN     NaN
1  No Churn    2280.0    0.0  2280.0
2     Churn      31.0  355.0   386.0
3       All    2311.0  355.0  2666.0

  Predicted  No Churn  Churn    All
0    Actual       NaN    NaN    NaN
1  No Churn     563.0    7.0  570.0
2     Churn      38.0   59.0   97.0
3       All     601.0   66.0  667.0

-----------------------------------------------------

Classification Report for train & test set

Train set
              precision    recall  f1-score   support

       False       0.99      1.00      0.99      2280
        True       1.00      0.92      0.96       386

    accuracy                           0.99      2666
   macro avg       0.99      0.96      0.98      2666
weighted avg       0.99      0.99      0.99      2666


Test set
              precision    recall  f1-score   support

       False       0.94      0.99      0.96       570
        True       0.89      0.61      0.72        97
```

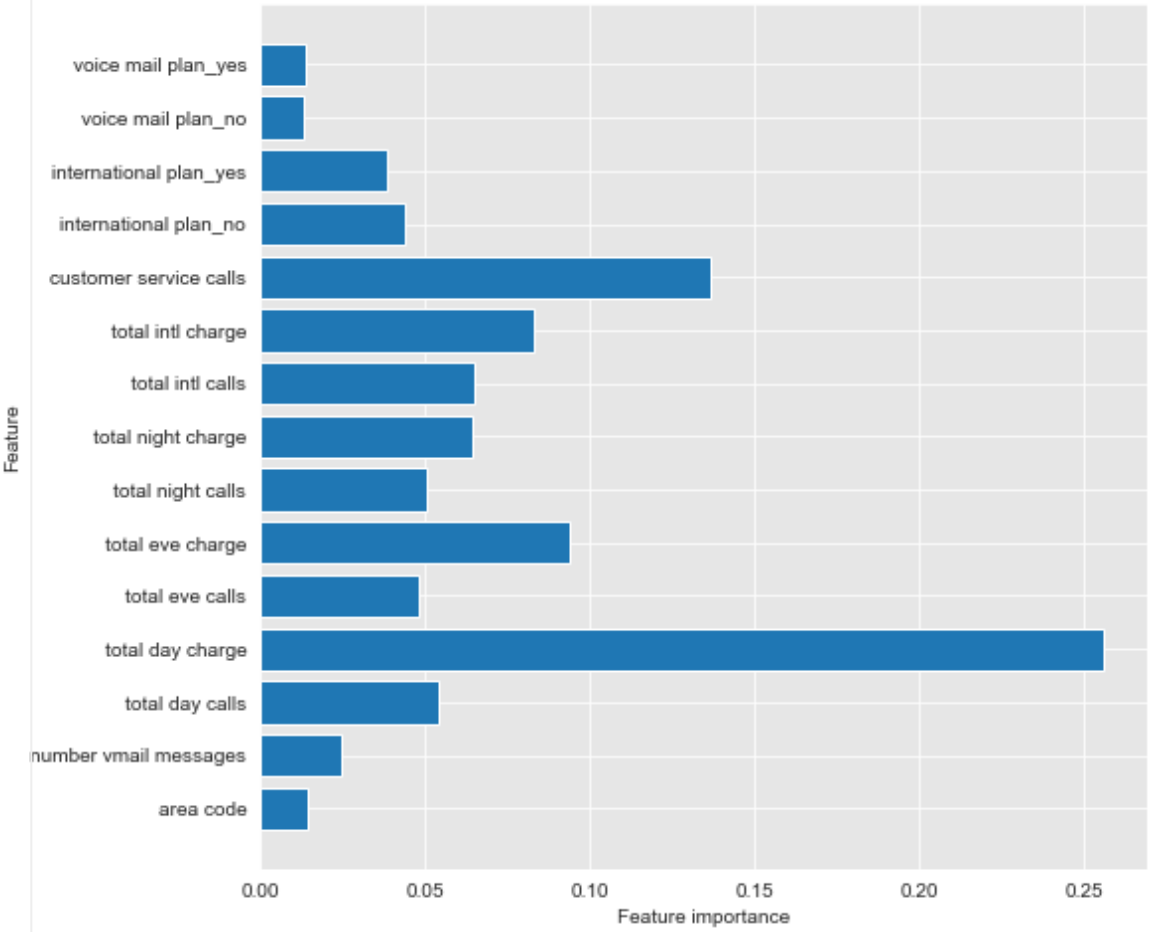
```
    accuracy                           0.93       667
   macro avg       0.92      0.80      0.84       667
weighted avg       0.93      0.93      0.93       667

--------------------------------------------------
```

```
Cohen's Kappa for train and test set:
  0.9514 0.6871
f2 score for train and test set:
  0.9347 0.6498
roc auc score for train and test set:
  0.9598 0.798
Mean Cross Validation Score:
 0.9466
```



Contents

Contents



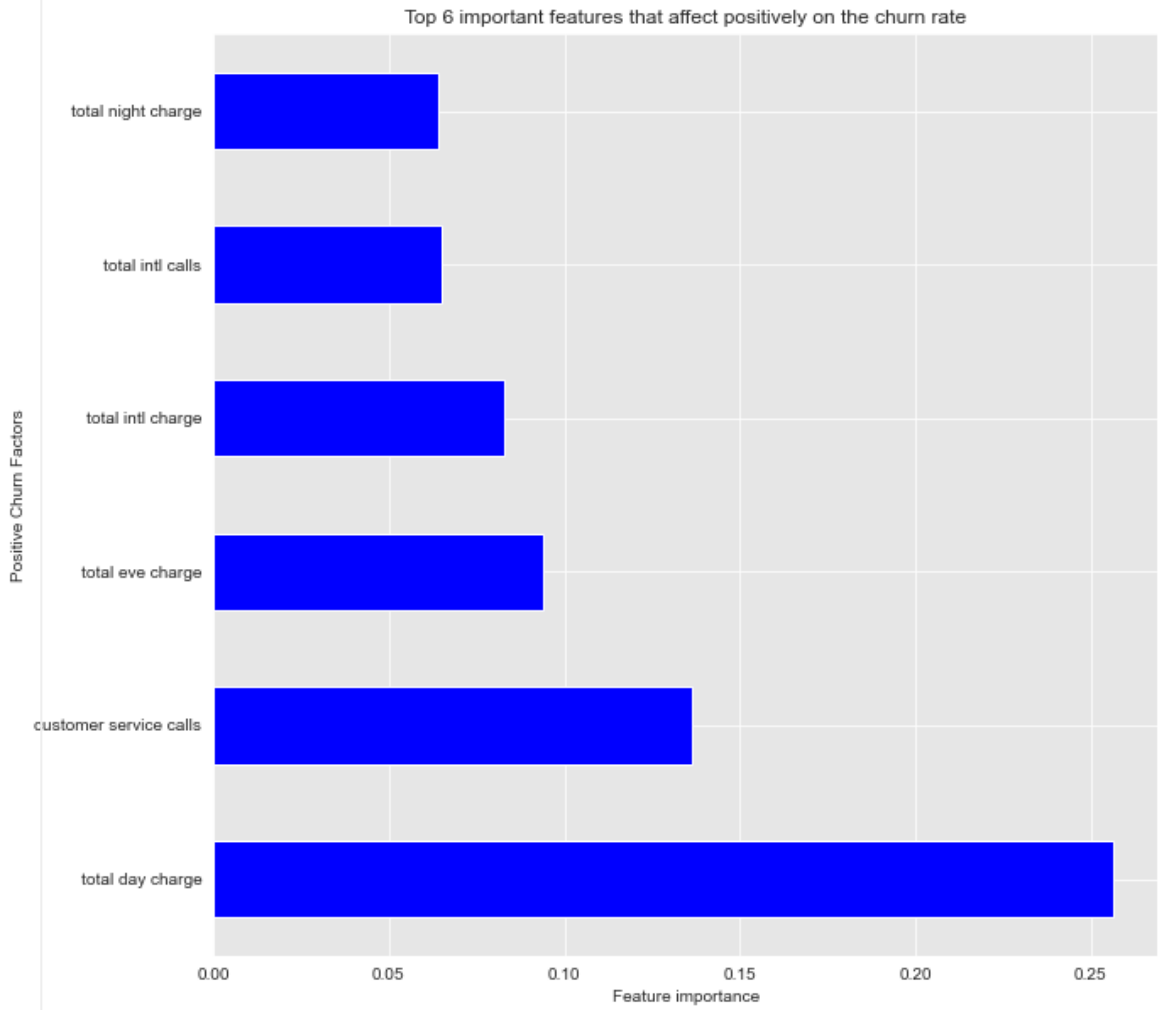
In [81]:

```
feature_importance = rf_clf.feature_importances_
print (rf_clf.feature_importances_)
feat_importances = pd.Series(rf_clf.feature_importances_, index=X.colum
feat_importances = feat_importances.nlargest(6)
feat_importances.plot(kind='barh' , figsize=(10,10), color="b")

plt.xlabel('Feature importance')
plt.ylabel('Positive Churn Factors')
plt.title('Top 6 important features that affect positively on the churr
```

```
[0.01450957 0.02484146 0.05424423 0.25617375 0.04835243 0.09392077
 0.05054068 0.06424145 0.06487218 0.08277946 0.13643413 0.04379005
 0.03817828 0.01335585 0.01376572]
```

Out[81]:

```
Text(0.5, 1.0, 'Top 6 important features that affect positively on the chur
n rate')
```



Contents

9.3 Results & Comments

We have a precision of 0.61 and recall of 0.89 for the test set. We have some overfitting. It means we are able to detect the churn customers 61% but 89% of the customers we say 'churn' was true.

The top 5 factors affecting te churn rate:

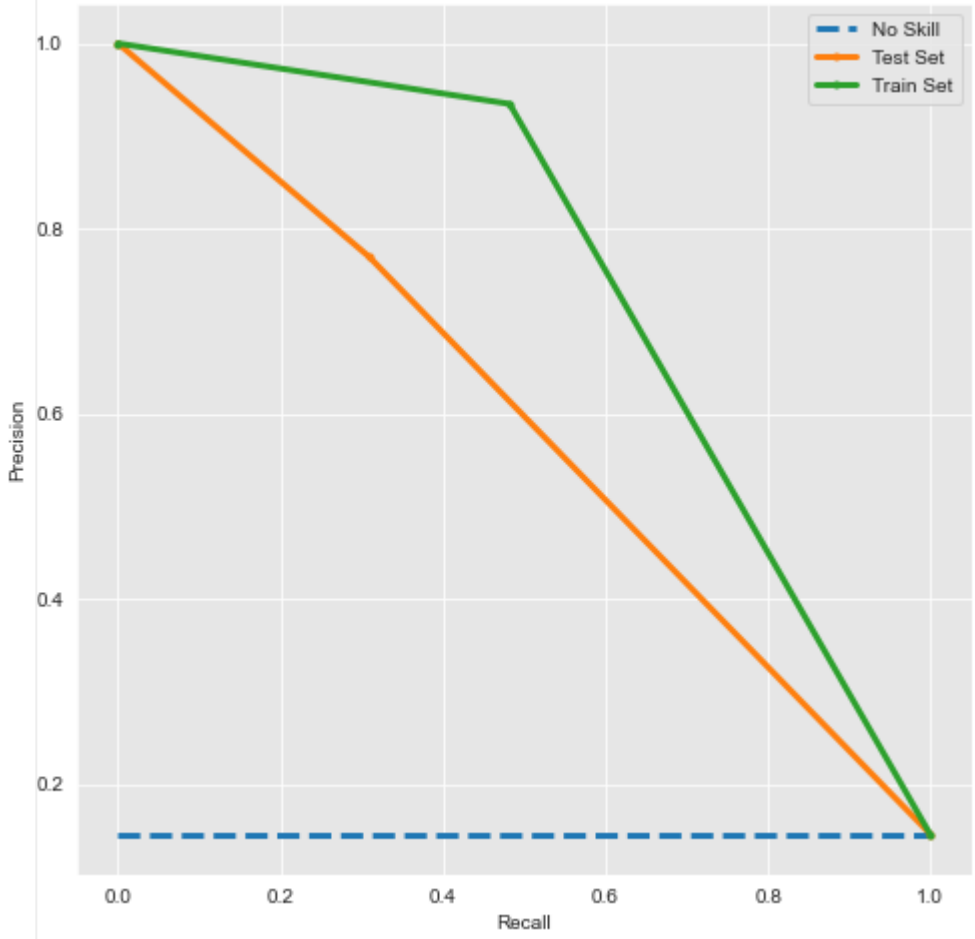
1) Total day charge 2) Customer service calls 3) Total evening charge 4) Total international charge 5) Total international calls 6) Total night charge

10 KNN Classifier

Contents

In [82]:

```
#Instantiate KNeighborsClassifier
clf= KNeighborsClassifier()

# Fit the classifier
clf.fit(scaled_X_train, y_train)

# Predict on the test set
test_preds = clf.predict(scaled_X_test)
train_preds=clf.predict(scaled_X_train)




plot_pr_rc_curve(y_test, test_preds, y_train, train_preds)
```



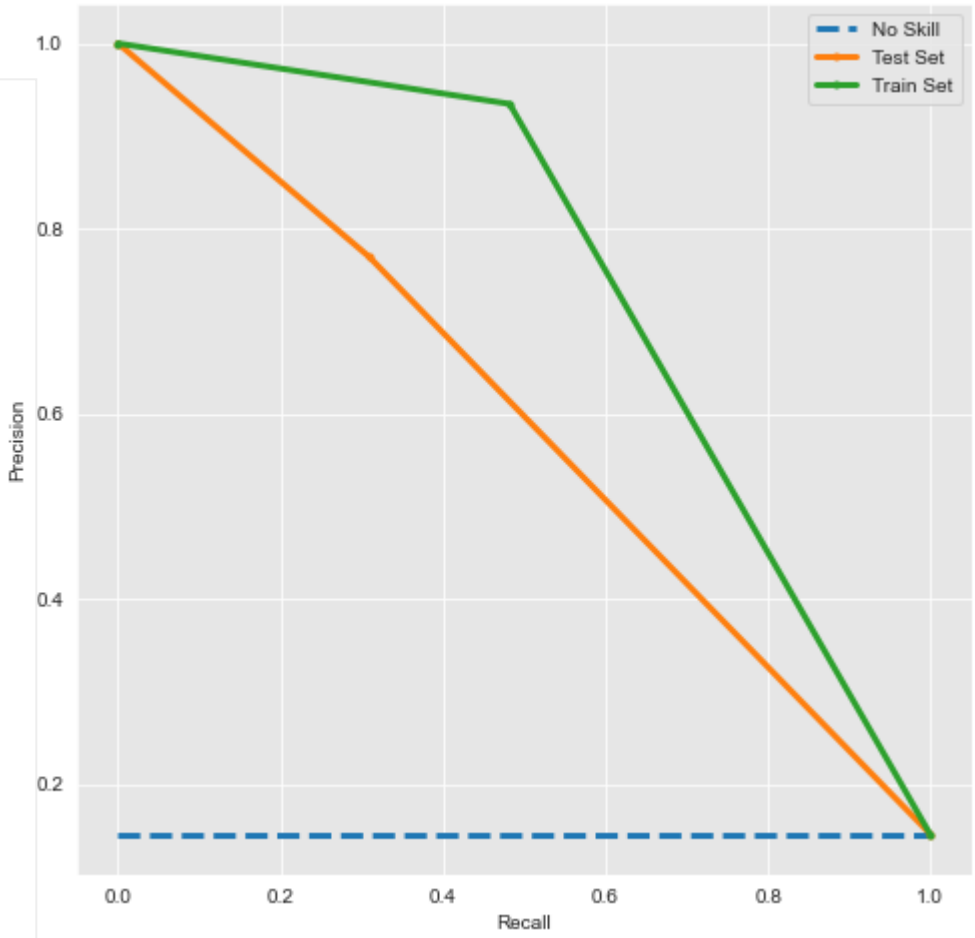
Contents

In [83]:
```
1 model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, train_
```

```
MODEL EVALUATION METRICS:
 -----------------------------------------------------
Confusion Matrix for train & test set:

  Predicted  No Churn  Churn     All
0    Actual       NaN    NaN     NaN
1  No Churn    2267.0   13.0  2280.0
2     Churn     200.0  186.0   386.0
3       All    2467.0  199.0  2666.0

  Predicted  No Churn  Churn    All
0    Actual       NaN    NaN    NaN
1  No Churn     561.0    9.0  570.0
2     Churn      67.0   30.0   97.0
3       All     628.0   39.0  667.0

-----------------------------------------------------

Classification Report for train & test set

Train set
              precision    recall  f1-score   support

       False       0.92      0.99      0.96      2280
        True       0.93      0.48      0.64       386

    accuracy                           0.92      2666
   macro avg       0.93      0.74      0.80      2666
weighted avg       0.92      0.92      0.91      2666


Test set
              precision    recall  f1-score   support

       False       0.89      0.98      0.94       570
        True       0.77      0.31      0.44        97

    accuracy                           0.89       667
   macro avg       0.83      0.65      0.69       667
weighted avg       0.88      0.89      0.86       667

-----------------------------------------------------

Cohen's Kappa for train and test set:
  0.5961 0.3903
f2 score for train and test set:
  0.5336 0.3513
roc auc score for train and test set:
  0.7381 0.6467
Mean Cross Validation Score:
 0.8614
```

Contents

In [84]:
```
plot_pr_rc_curve(y_test, test_preds, y_train, train_preds)
```



Contents

10.1 Tuning & Optimization (Iterate over n values)

In [85]:
```
def find_best_k(scaled_X_train, y_train, scaled_X_test, y_test, min_k=1
    best_k = 0
    best_score = 0.0
    for k in range(min_k, max_k+1, 2):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(scaled_X_train, y_train)
        preds = knn.predict(scaled_X_test)
        f1 = f1_score(y_test, preds)
        if f1 > best_score:
            best_k = k
            best_score = f1

    print("Best Value for k: {}".format(best_k))
    print("F1-Score: {}".format(best_score))
```

In [86]:
```
1 find_best_k(scaled_X_train, y_train, scaled_X_test, y_test)
```

```
Best Value for k: 3
F1-Score: 0.4583333333333334
```

Contents

10.1.1 Select Best K for KNN Machine Learning Model

10.1.1.1 K=3

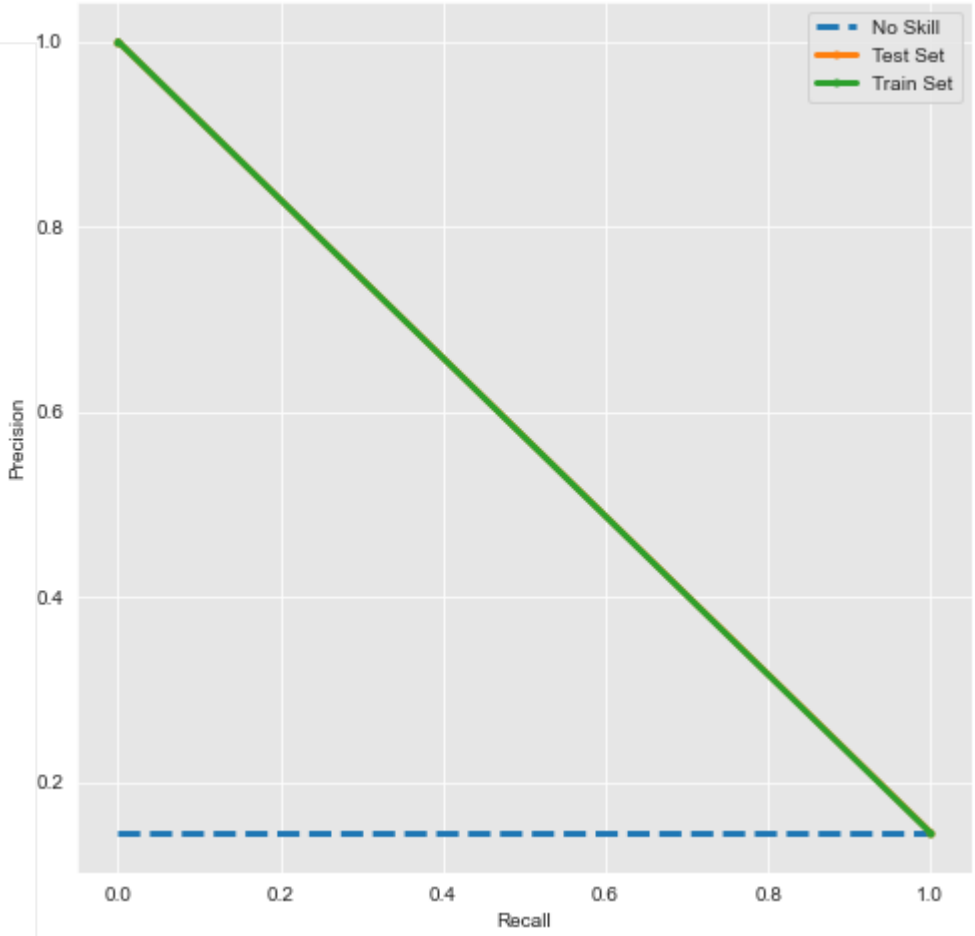
In [87]:

```
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)
train_preds = clf.predict(scaled_X_train)
test_preds = clf.predict(scaled_X_test)
model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, train_
plot_pr_rc_curve(y_test, test_preds, y_train, train_preds)

```

Contents


```
MODEL EVALUATION METRICS:
 -----------------------------------------------------
Confusion Matrix for train & test set:

  Predicted  No Churn  Churn     All
0    Actual       NaN    NaN     NaN
1  No Churn    2280.0    0.0  2280.0
2     Churn     386.0    0.0   386.0
3       All    2666.0    0.0  2666.0

  Predicted  No Churn  Churn    All
0    Actual       NaN    NaN    NaN
1  No Churn     570.0    0.0  570.0
2     Churn      97.0    0.0   97.0
3       All     667.0    0.0  667.0

-----------------------------------------------------

Classification Report for train & test set

Train set
              precision    recall  f1-score   support

       False       0.86      1.00      0.92      2280
        True       0.00      0.00      0.00       386

    accuracy                           0.86      2666
   macro avg       0.43      0.50      0.46      2666
weighted avg       0.73      0.86      0.79      2666


Test set
              precision    recall  f1-score   support

       False       0.85      1.00      0.92       570
        True       0.00      0.00      0.00        97

    accuracy                           0.85       667
   macro avg       0.43      0.50      0.46       667
weighted avg       0.73      0.85      0.79       667

-----------------------------------------------------

Cohen's Kappa for train and test set:
  0.0 0.0
f2 score for train and test set:
  0.0 0.0
```

```
roc auc score for train and test set:
  0.5 0.5
Mean Cross Validation Score:
 0.8437
```



Contents

In [88]:

```
KNN_clf_score = cross_val_score(clf, scaled_X_train, y_train, cv=10)
mean_KNN_clf_score = np.mean(KNN_clf_score)


print(f"Mean Cross Validation Score: {mean_KNN_clf_score :.2%}")


```

```
Mean Cross Validation Score: 89.05%
```

10.1.2 Iterate over 1 < n < 26

In [89]:

```
k_range = range(1, 26)

for k in k_range:
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(scaled_X_train, y_train)
    y_pred_train = clf.predict(scaled_X_train)
    y_pred_test = clf.predict(scaled_X_test)

    print('------------')
    print(k)
    print('\nClassification Report for train & test set\n',
          '\nTrain set\n',
          classification_report(y_train, y_pred_train),
          '\n\nTest set\n',
          classification_report(y_test, y_pred_test))
    print('-----------------------------------------------\n')

    print("f2 score for train and test set: \n ",
          round(fbeta_score(y_train, y_pred_train, 2.0), 4),
          round(fbeta_score(y_test, y_pred_test, 2.0), 4))
```

```
------------
1

Classification Report for train & test set

Train set
              precision    recall  f1-score   support

       False       1.00      1.00      1.00      2280
        True       1.00      1.00      1.00       386

    accuracy                           1.00      2666
   macro avg       1.00      1.00      1.00      2666
weighted avg       1.00      1.00      1.00      2666


Test set
              precision    recall  f1-score   support
```


10.1.2.1 K=1

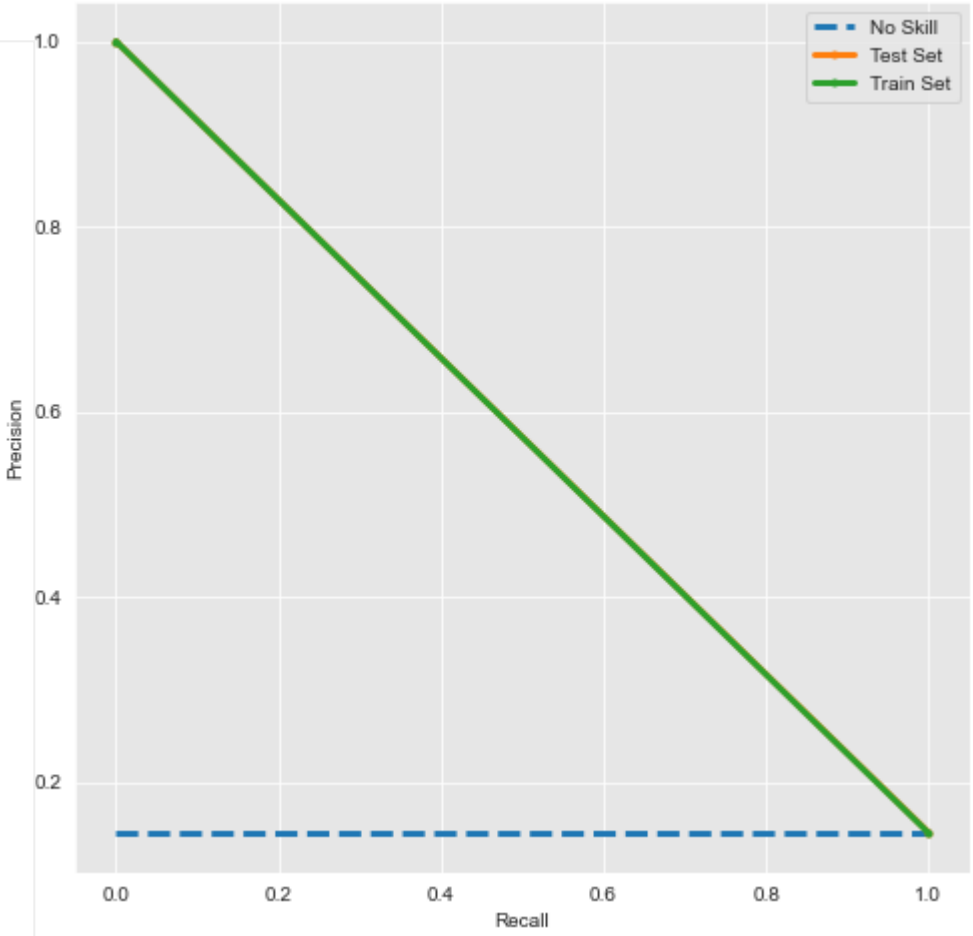
In [90]:

```
clf = KNeighborsClassifier(n_neighbors=1)
clf.fit(X_train, y_train)
train_preds = clf.predict(scaled_X_train)
test_preds = clf.predict(scaled_X_test)
model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, train_
plot_pr_rc_curve(y_test, test_preds, y_train, train_preds)

```

Contents


```
MODEL EVALUATION METRICS:
 -----------------------------------------------------
Confusion Matrix for train & test set:

  Predicted  No Churn  Churn     All
0    Actual       NaN    NaN     NaN
1  No Churn    2280.0    0.0  2280.0
2     Churn     386.0    0.0   386.0
3       All    2666.0    0.0  2666.0

  Predicted  No Churn  Churn    All
0    Actual       NaN    NaN    NaN
1  No Churn     570.0    0.0  570.0
2     Churn      97.0    0.0   97.0
3       All     667.0    0.0  667.0

-----------------------------------------------------

Classification Report for train & test set

Train set
              precision    recall  f1-score   support

       False       0.86      1.00      0.92      2280
        True       0.00      0.00      0.00       386

    accuracy                           0.86      2666
   macro avg       0.43      0.50      0.46      2666
weighted avg       0.73      0.86      0.79      2666


Test set
              precision    recall  f1-score   support

       False       0.85      1.00      0.92       570
        True       0.00      0.00      0.00        97

    accuracy                           0.85       667
   macro avg       0.43      0.50      0.46       667
weighted avg       0.73      0.85      0.79       667

-----------------------------------------------------

Cohen's Kappa for train and test set:
  0.0 0.0
f2 score for train and test set:
  0.0 0.0
```

```
roc auc score for train and test set:
  0.5 0.5
Mean Cross Validation Score:
 0.7942
```



10.2 Results & Comments

The mean cross validation is around 85%. This model performed poorly even for the optimal selected K (K=1 or 3) with poor recall snd precision for True Churn.

In [93]:

```
1 ! pip install  xgboost
```

```
Requirement already satisfied: xgboost in c:\users\mirnamamaranda\anaconda3
\lib\site-packages (1.3.3)
Requirement already satisfied: scipy in c:\users\mirnamamaranda\anaconda3\l
ib\site-packages (from xgboost) (1.4.1)
Requirement already satisfied: numpy in c:\users\mirnamamaranda\anaconda3\l
ib\site-packages (from xgboost) (1.19.2)
```

11 XGBoost

In [ ]:

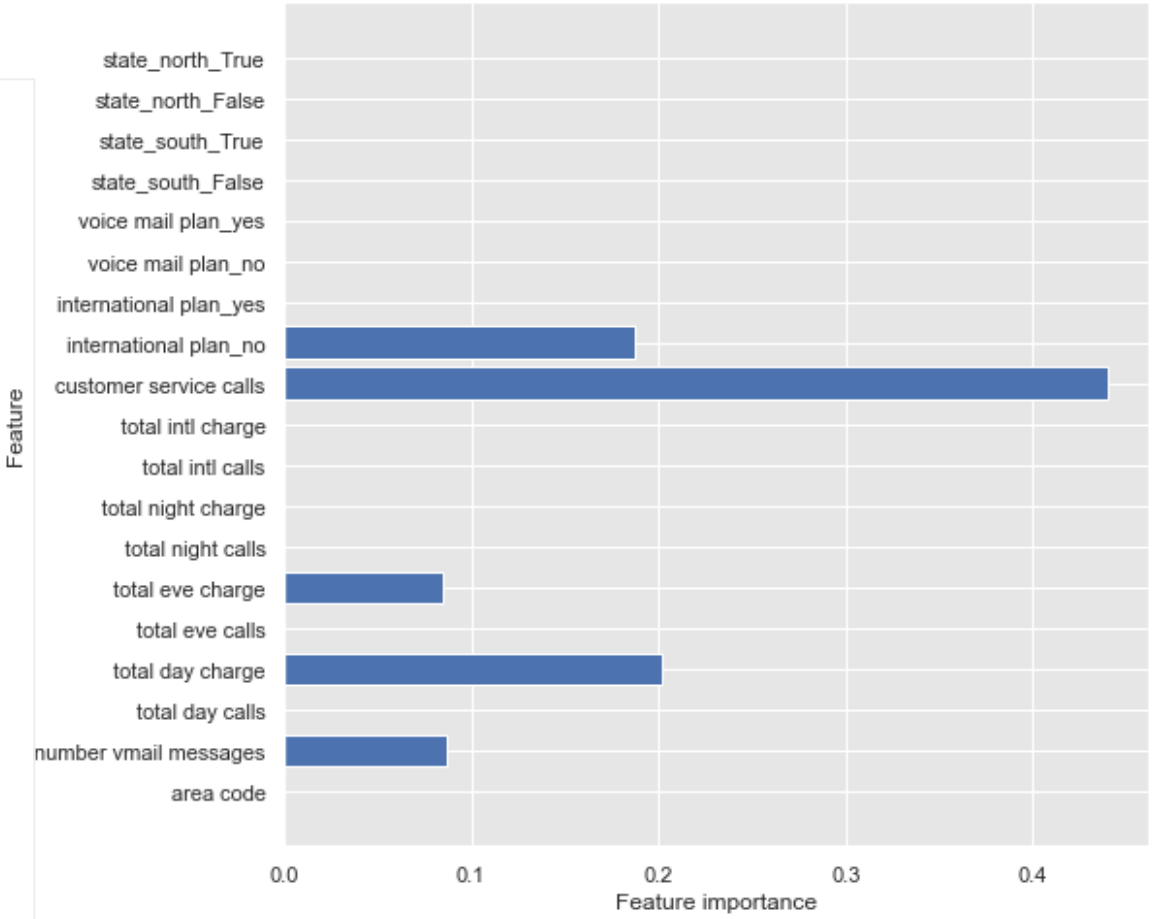
```
from xgboost import XGBClassifier



clf = XGBClassifier(objective='binary:logistic',
                    nthread=4,
                    scale_pos_weight=3,
                    seed=42,
                    max_depth=3,
                    n_estimators=140,
                    learning_rate=0.005)

clf.fit(scaled_X_train, y_train)
y_pred_train = clf.predict(scaled_X_train)
y_pred_test = clf.predict(scaled_X_test)
```

In [ ]:

```
model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_prec
plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)
```

Contents

In [399]:
```
1 plot_feature_importances(clf)
```



Contents

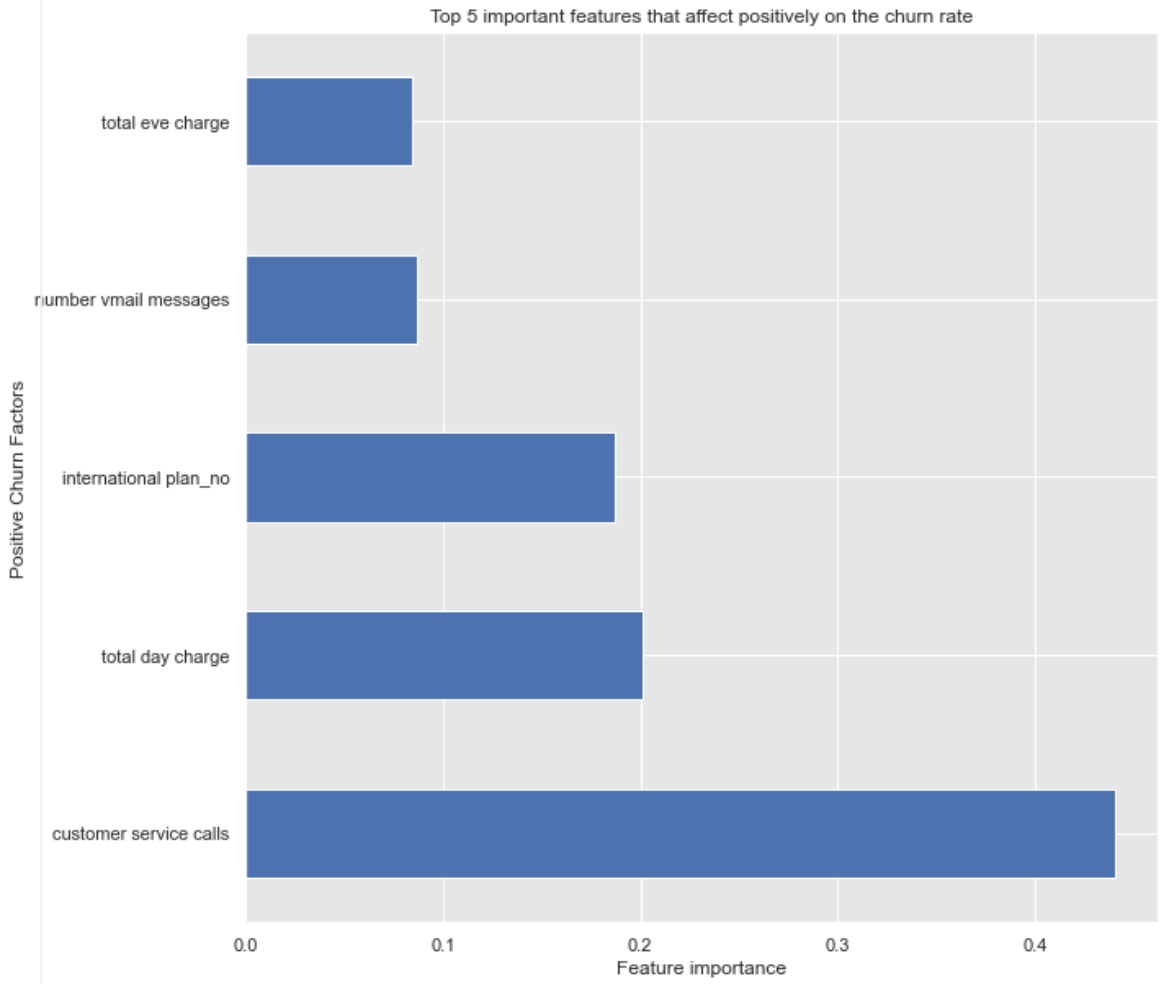
In [401]:

```
feature_importance = clf.feature_importances_
print (clf.feature_importances_)
feat_importances = pd.Series(clf.feature_importances_, index=X.columns)
feat_importances = feat_importances.nlargest(5)
feat_importances.plot(kind='barh' , figsize=(10,10), color="b")

plt.xlabel('Feature importance')
plt.ylabel('Positive Churn Factors')
plt.title('Top 5 important features that affect positively on the churr
```



```
[0.         0.08669046 0.         0.20160858 0.         0.08463372
 0.         0.         0.         0.         0.4402261  0.18684117
 0.         0.         0.         0.         0.         0.
 0.        ]
```

Out[401]: Text(0.5, 1.0, 'Top 5 important features that affect positively on the churn rate')



Contents

11.1 Results & comments

The mean cross validation is 88.69%. We have a precision of 0.62 and recall of 0.74 for the test set. We don't have overfitting. It means we are able to detect the 74% of churn customers but 62% of the customers we say 'churn' was true.

The top 5 factors affecting te churn rate:

1) Customer service calls

2) Total day charge

3) Internaltional plan_no

4) Number voice messages

5) Total evening charge

12 Gaussian NB

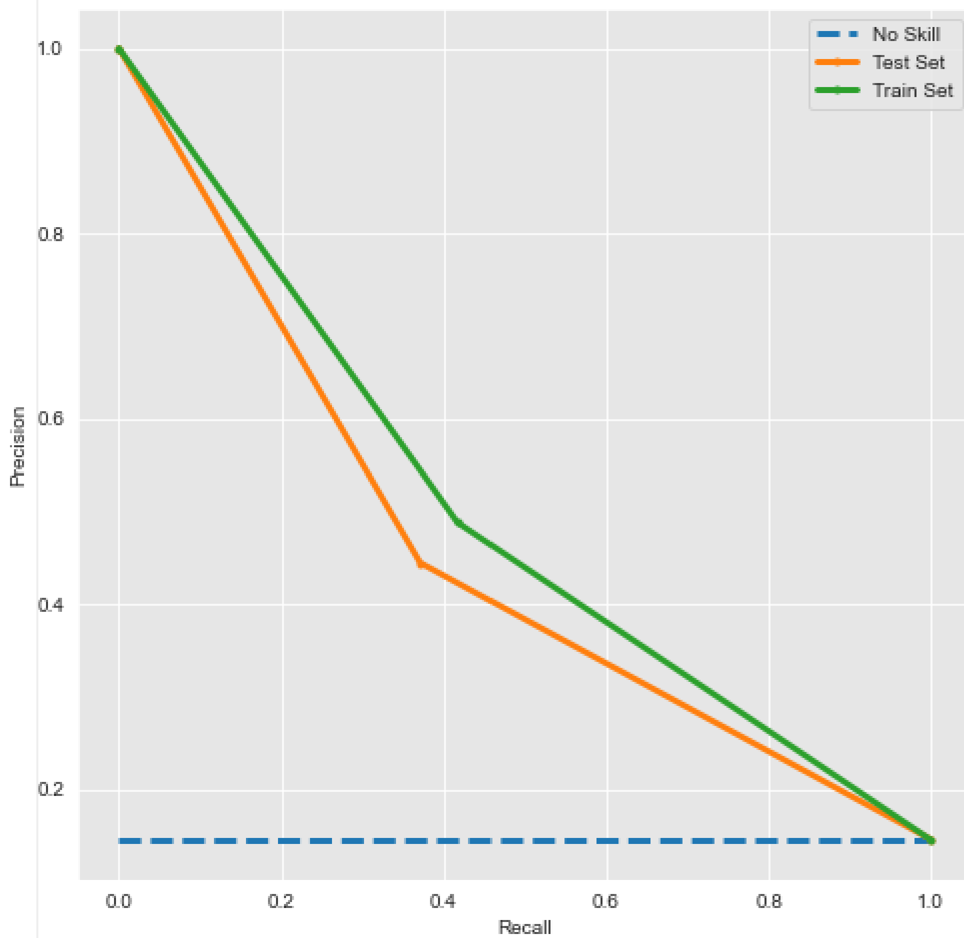
In [95]:

```
nb = GaussianNB()
nb_model=nb.fit(scaled_X_train, y_train)
y_pred_train = nb_model.predict(scaled_X_train)
y_pred_test = nb_model.predict(scaled_X_test)

```

In [96]:
```
model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_trai

plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)
```

```
MODEL EVALUATION METRICS:
 -----------------------------------------------------
Confusion Matrix for train & test set:

  Predicted  No Churn  Churn     All
0    Actual       NaN    NaN     NaN
1  No Churn    2280.0    0.0  2280.0
2     Churn       0.0  386.0   386.0
3       All    2280.0  386.0  2666.0

  Predicted  No Churn  Churn    All
0    Actual       NaN    NaN    NaN
1  No Churn     570.0    0.0  570.0
2     Churn       0.0   97.0   97.0
3       All     570.0   97.0  667.0

-----------------------------------------------------

Classification Report for train & test set

Train set
              precision    recall  f1-score   support

       False       1.00      1.00      1.00      2280
        True       1.00      1.00      1.00       386

    accuracy                           1.00      2666
   macro avg       1.00      1.00      1.00      2666
weighted avg       1.00      1.00      1.00      2666


Test set
              precision    recall  f1-score   support

       False       1.00      1.00      1.00       570
        True       1.00      1.00      1.00        97

    accuracy                           1.00       667
   macro avg       1.00      1.00      1.00       667
weighted avg       1.00      1.00      1.00       667

-----------------------------------------------------

Cohen's Kappa for train and test set:
  1.0 1.0
f2 score for train and test set:
  1.0 1.0
roc auc score for train and test set:
  1.0 1.0
[14:57:27] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
```

```
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
[14:57:27] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
[14:57:27] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
[14:57:27] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
[14:57:27] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
Mean Cross Validation Score:
 0.8869
```



Contents

In [82]:
```
1 nb_model.get_params
```

Out[82]: <bound method BaseEstimator.get_params of GaussianNB()>

Contents

12.1 Results & Comments

The mean cross validation is 88.7%. We have a precision of 1 and recall of 1 for the test set. We don't have overfitting. It means we are able to detect the 100% of churn customers and 100% of the customers we say 'churn' was true.

13 AdaBoostClassifier

In [97]:

```
from sklearn.ensemble import AdaBoostClassifier


ada = AdaBoostClassifier(n_estimators=120,random_state=5,learning_rate=
ada_model=ada.fit(scaled_X_train,y_train)

y_pred_train = ada_model.predict(scaled_X_train)
y_pred_test = ada_model.predict(scaled_X_test)

model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_trai

plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)
```



```
MODEL EVALUATION METRICS:
 -----------------------------------------------------
Confusion Matrix for train & test set:

  Predicted  No Churn  Churn     All
0    Actual       NaN    NaN     NaN
1  No Churn    2280.0    0.0  2280.0
2     Churn       0.0  386.0   386.0
3       All    2280.0  386.0  2666.0

  Predicted  No Churn  Churn    All
0    Actual       NaN    NaN    NaN
1  No Churn     570.0    0.0  570.0
2     Churn       0.0   97.0   97.0
3       All     570.0   97.0  667.0

-----------------------------------------------------

Classification Report for train & test set

Train set
              precision    recall  f1-score   support

       False       1.00      1.00      1.00      2280
        True       1.00      1.00      1.00       386

    accuracy                           1.00      2666
   macro avg       1.00      1.00      1.00      2666
weighted avg       1.00      1.00      1.00      2666


Test set
              precision    recall  f1-score   support

       False       1.00      1.00      1.00       570
        True       1.00      1.00      1.00        97

    accuracy                           1.00       667
   macro avg       1.00      1.00      1.00       667
weighted avg       1.00      1.00      1.00       667

-----------------------------------------------------
```

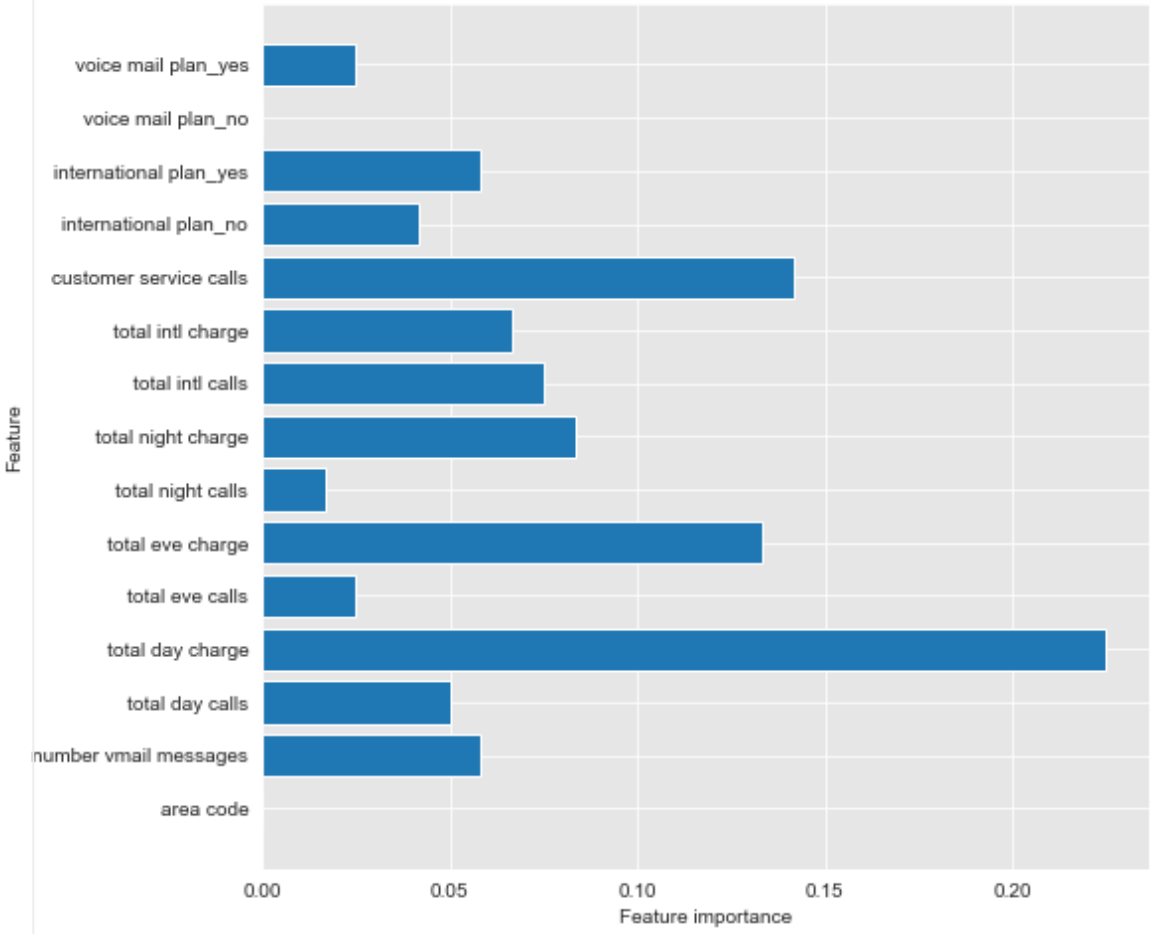
```
Cohen's Kappa for train and test set:
  1.0 1.0
f2 score for train and test set:
  1.0 1.0
roc auc score for train and test set:
  1.0 1.0
[14:58:28] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logisti
c' was changed from 'error' to 'logloss'. Explicitly set eval_metric i
f you'd like to restore the old behavior.
[14:58:28] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logisti
c' was changed from 'error' to 'logloss'. Explicitly set eval_metric i
f you'd like to restore the old behavior.
[14:58:28] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logisti
c' was changed from 'error' to 'logloss'. Explicitly set eval_metric i
f you'd like to restore the old behavior.
[14:58:29] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logisti
c' was changed from 'error' to 'logloss'. Explicitly set eval_metric i
f you'd like to restore the old behavior.
[14:58:29] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'binary:logisti
c' was changed from 'error' to 'logloss'. Explicitly set eval_metric i
f you'd like to restore the old behavior.
Mean Cross Validation Score:
 0.8869
```

Contents

Contents

In [98]:

```
1 plot_feature_importances(ada_model)
```

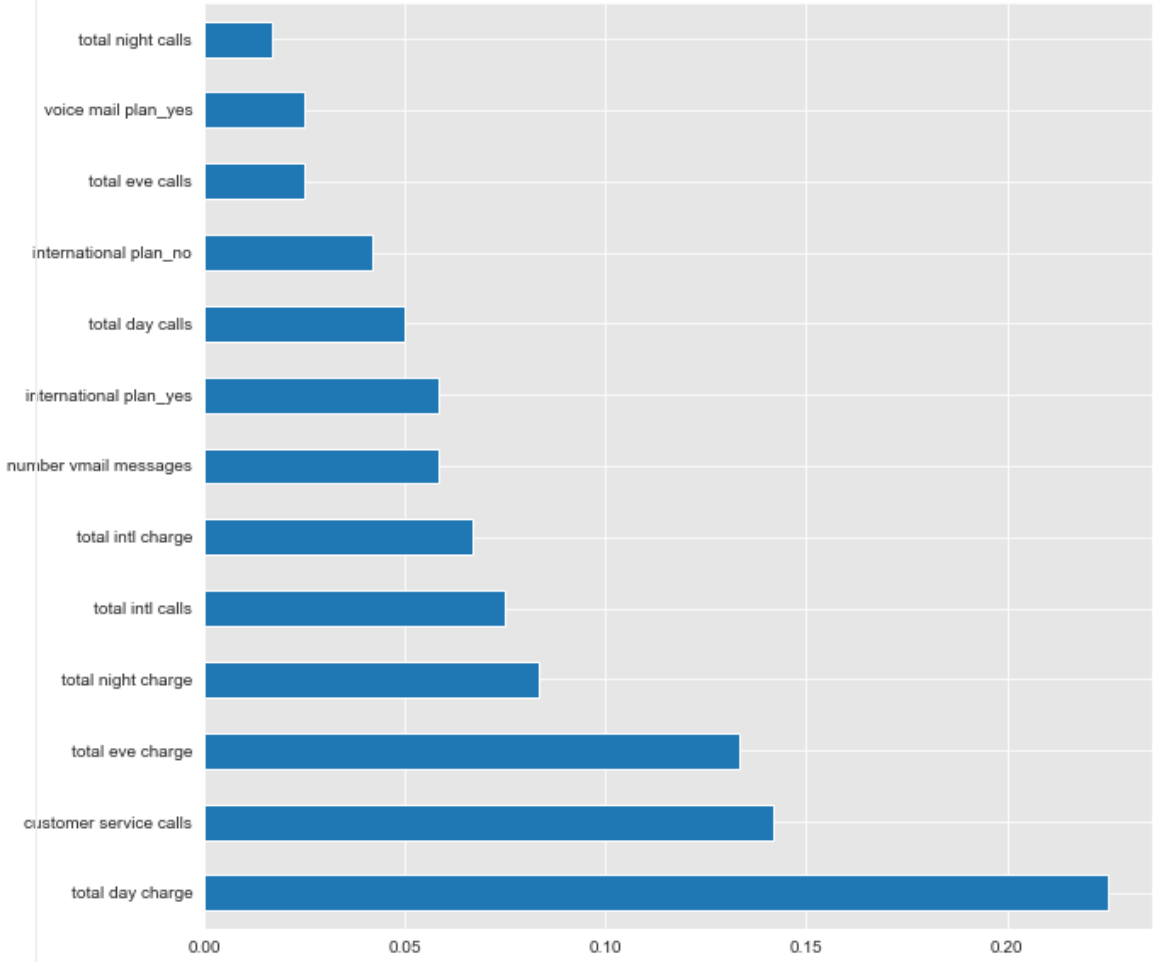


In [99]:

```
# Get Feature Importance from the classifier
feature_importance = ada_model.feature_importances_
print (ada_model.feature_importances_)
feat_importances = pd.Series(ada_model.feature_importances_, index=X.cc
feat_importances = feat_importances.nlargest(13)
feat_importances.plot(kind='barh' , figsize=(10,10))
```

```
[0.         0.05833333 0.05       0.225      0.025      0.13333333
 0.01666667 0.08333333 0.075      0.06666667 0.14166667 0.04166667
 0.05833333 0.         0.025     ]
```

Out[99]: <AxesSubplot:>



13.1 Results & Comments

The mean cross validation is 88.7%. We have a precision of 1 and recall of 1 for the test set. We don't have overfitting. It means we are able to detect the 100% of churn customers and 100% of the customers we say 'churn' was true.

The most important churn factors:

1) Total day charge

2) Customer service calls

3) Total evening charge

4) Total International calls

5) Total night charge

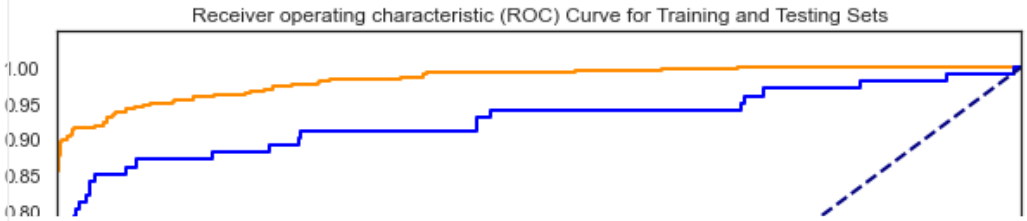
14 Gradient Boosting Classifier

In [187]:

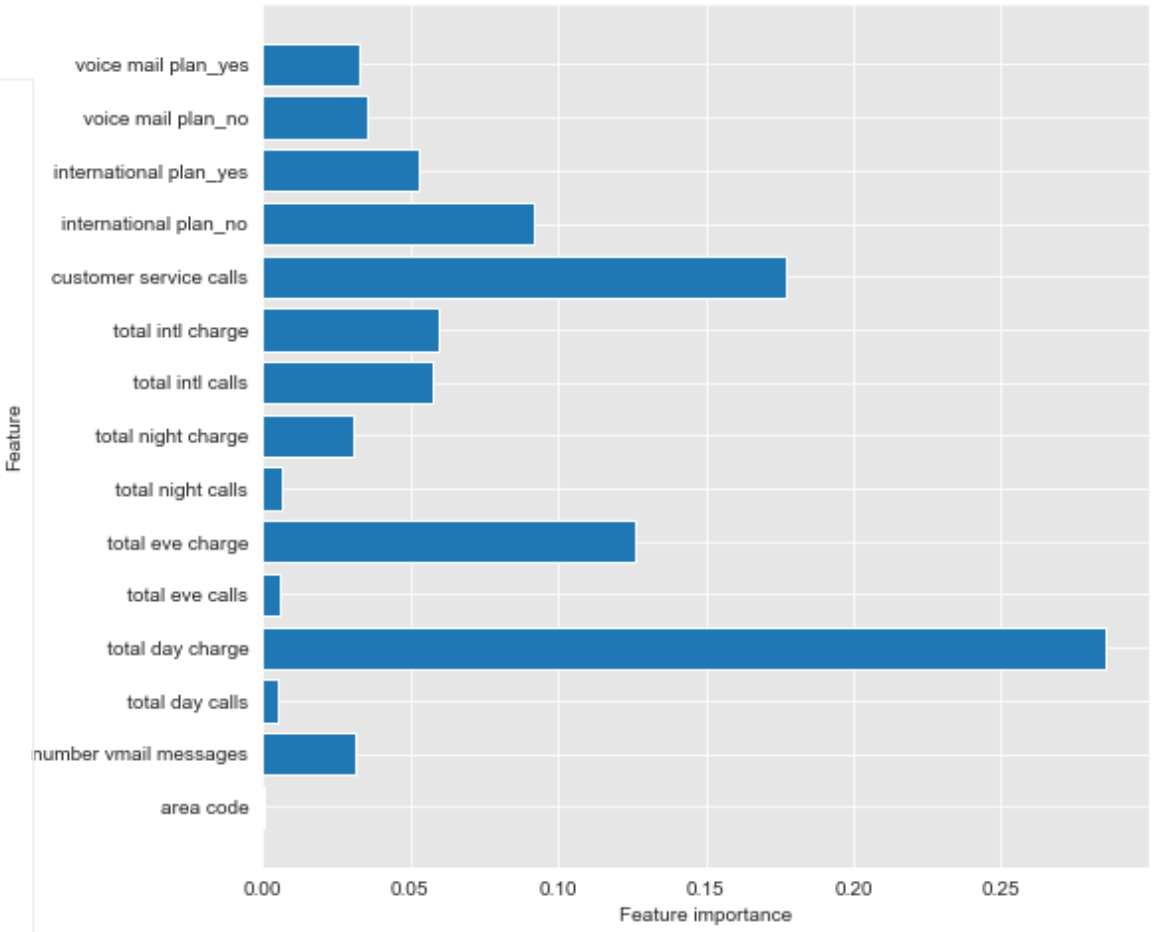
```
# Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(learning_rate=0.1,n_estimators=150,rand
gb_model=gb.fit(scaled_X_train,y_train)



y_pred_train = gb_model.predict(scaled_X_train)
y_pred_test = gb_model.predict(scaled_X_test)

model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_trai
roc_curve_and_auc(gb_model, scaled_X_train, scaled_X_test, y_train, y_t

plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)
```



```
weighted avg       1.00      1.00      1.00       667

----------------------------------------------------

Cohen's Kappa for train and test set:
  1.0 1.0
f2 score for train and test set:
  1.0 1.0
roc auc score for train and test set:
  1.0 1.0
Mean Cross Validation Score:
 0.7687
```



In [101]:
```
1 plot_feature_importances(gb_model)
```



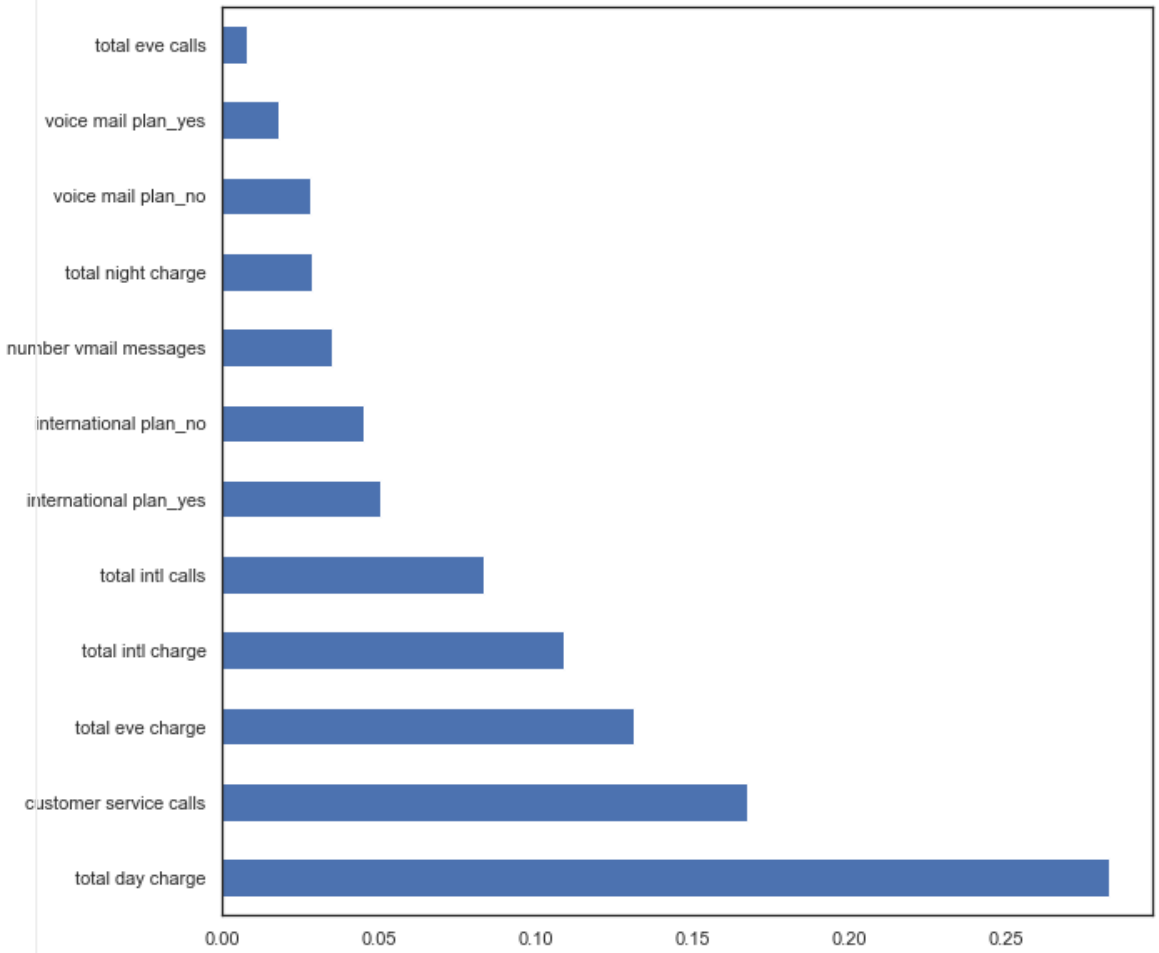
Contents

In [188]:
```
# Get Feature Importance from the classifier
feature_importance = gb_model.feature_importances_
print (gb_model.feature_importances_)
feat_importances = pd.Series(gb_model.feature_importances_, index=X.col
feat_importances = feat_importances.nlargest(12)
feat_importances.plot(kind='barh' , figsize=(10,10))
```



```
[0.00174112 0.03503597 0.00639164 0.28248003 0.00768167 0.13115795
 0.00619365 0.02847671 0.08348926 0.108813   0.16733934 0.04515022
 0.05019546 0.02789222 0.01796176]
```

Out[188]: <AxesSubplot:>



14.1 Results & comments

The mean cross validation is 88.7%. We have a precision of 1 and recall of 1 for the test set. We don't have overfitting. It means we are able to detect the 100% of churn customers and 100% of the customers we say 'churn' was true.

The most important churn factors:

1) Total day charge

2) Customer service calls

3) Total evening charge

4) Total International charge

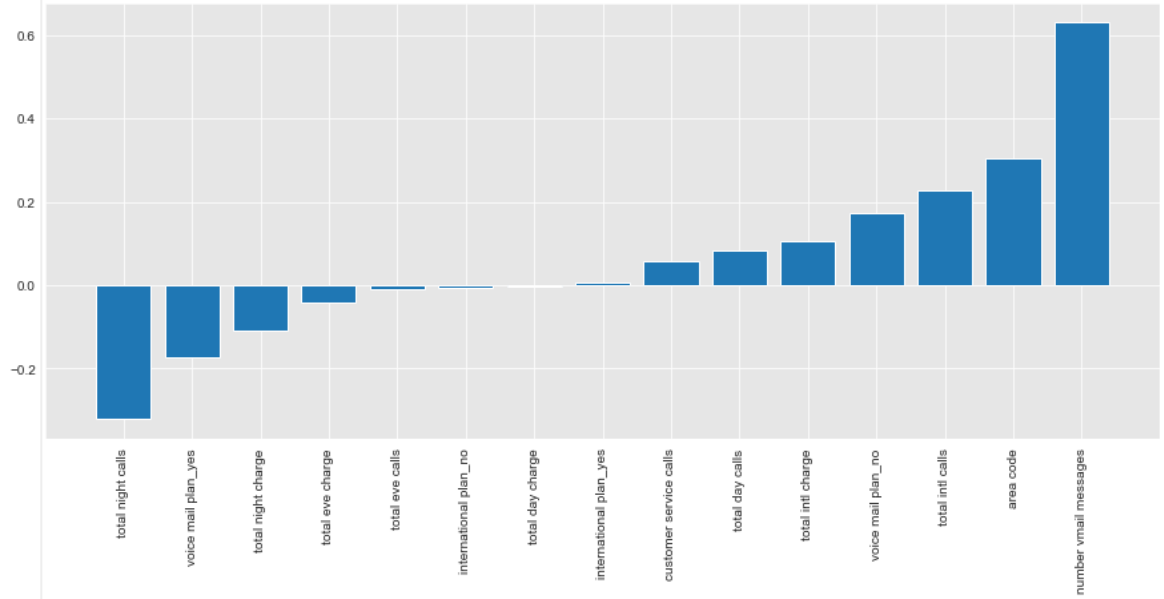
5) Total international calls

15 SVM Classifier

In [103]:
```
#SV M Classifier
#Make a sample to run SVM faster

sample_X_train = scaled_X_train.sample(n=1000)
sample_y_train = y_train.sample(n=1000)
```

In [104]:
```
#SVM Classifier (Balanced Class Weight)
svc = SVC(kernel='linear', class_weight='balanced')
svc.fit(sample_X_train, sample_y_train)
y_pred_train = svc.predict(sample_X_train)
y_pred_test = svc.predict(scaled_X_test)
```

In [105]:
```
plot_coefficients(svc)
```



In [106]:
```
1 model_evaluation(sample_X_train, scaled_X_test, sample_y_train, y_test,
```

```
MODEL EVALUATION METRICS:
 -----------------------------------------------------
Confusion Matrix for train & test set:

  Predicted  No Churn  Churn     All
0    Actual       NaN    NaN     NaN
1  No Churn     510.0  328.0   838.0
2     Churn      85.0   77.0   162.0
3       All     595.0  405.0  1000.0

  Predicted  No Churn  Churn    All
0    Actual       NaN    NaN    NaN
1  No Churn     347.0  223.0  570.0
2     Churn      65.0   32.0   97.0
3       All     412.0  255.0  667.0

-----------------------------------------------------

Classification Report for train & test set

Train set
              precision    recall  f1-score   support

       False       0.86      0.61      0.71       838
        True       0.19      0.48      0.27       162

    accuracy                           0.59      1000
   macro avg       0.52      0.54      0.49      1000
weighted avg       0.75      0.59      0.64      1000


Test set
              precision    recall  f1-score   support

       False       0.84      0.61      0.71       570
        True       0.13      0.33      0.18        97

    accuracy                           0.57       667
   macro avg       0.48      0.47      0.44       667
weighted avg       0.74      0.57      0.63       667

-----------------------------------------------------

Cohen's Kappa for train and test set:
  0.0523 -0.0366
f2 score for train and test set:
  0.3656 0.2488
roc auc score for train and test set:
  0.542 0.4693
[15:00:57] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
```

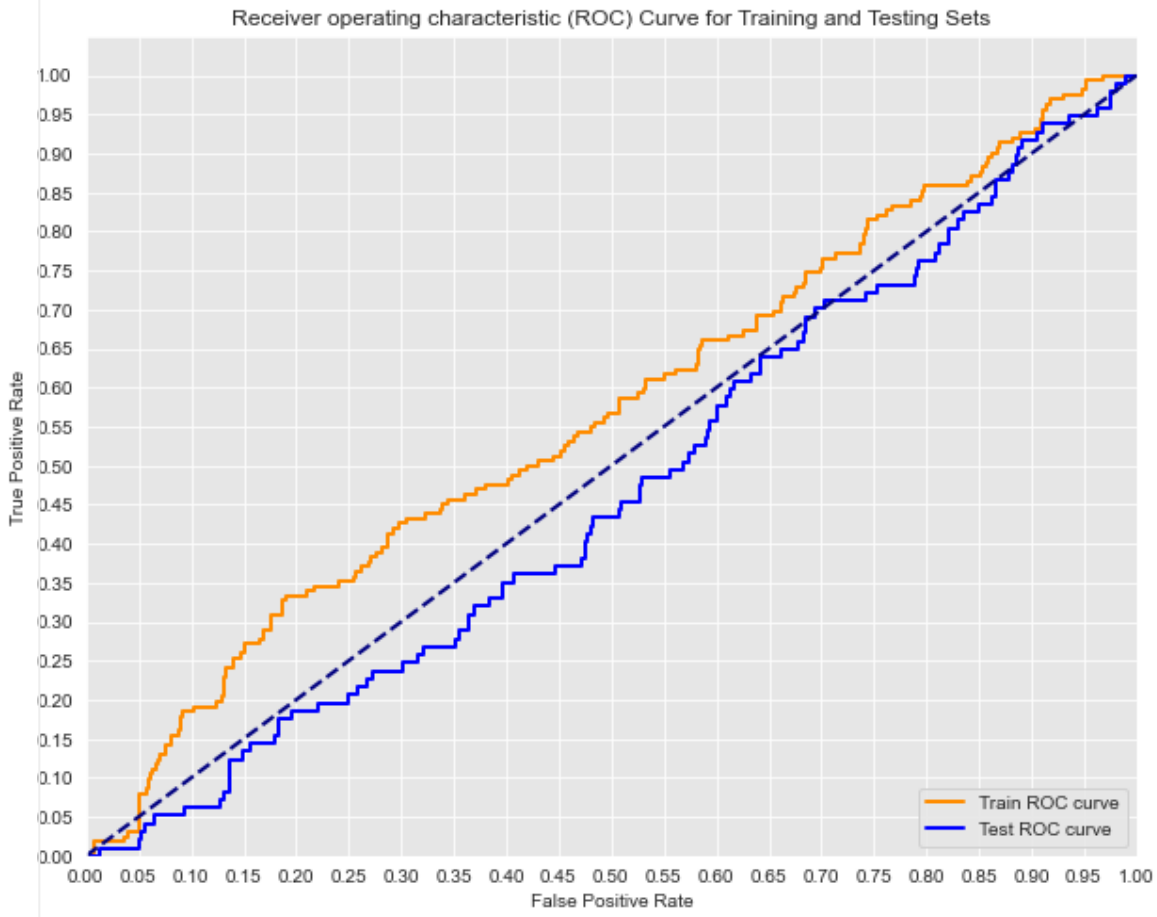
Contents

```
[15:00:57] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
[15:00:57] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
[15:00:58] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
[15:00:58] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was cha
nged from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
Mean Cross Validation Score:
 0.8869
```

Contents

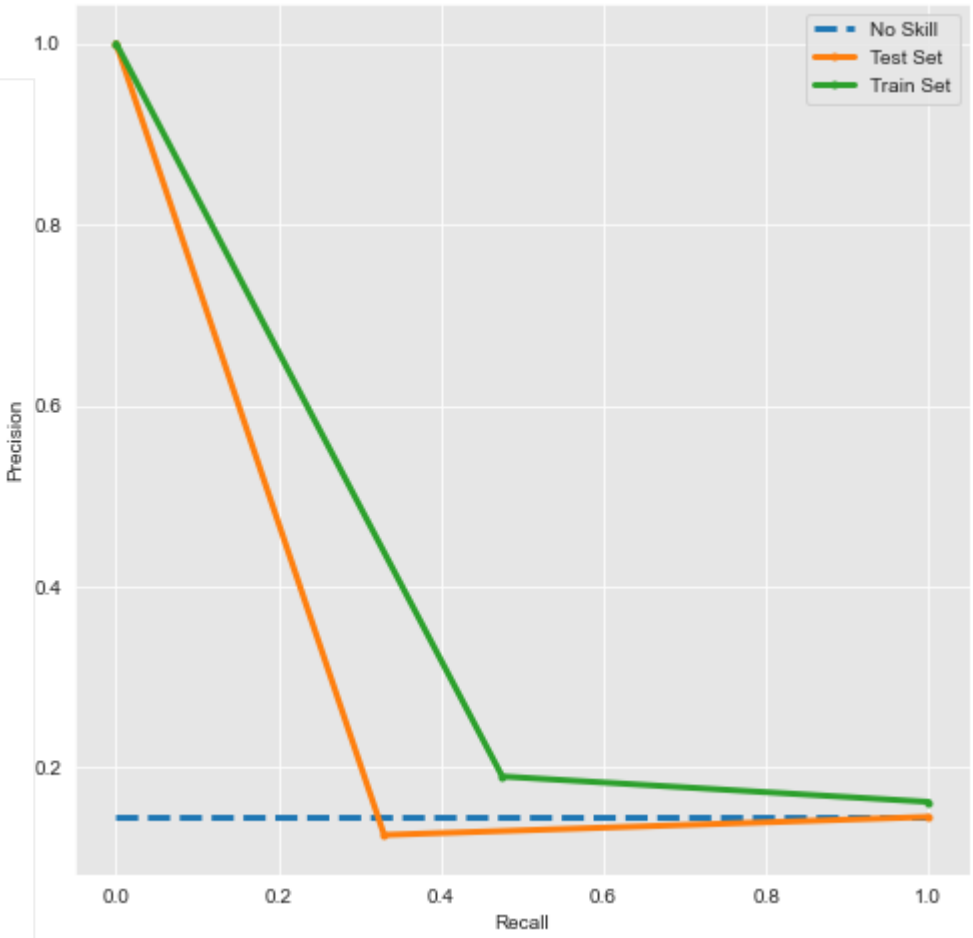
In [107]:

```
1 roc_curve_and_auc(svc, sample_X_train, scaled_X_test, sample_y_train, y
```




```
Training AUC: 0.56507
Testing AUC: 0.46609
```

In [108]:
```
1 plot_pr_rc_curve(y_test, y_pred_test, sample_y_train, y_pred_train)
```



15.1 Tuning & Optimization

15.1.1 Hypertuning of SVM Classifier throgh GridSearhCV

In [109]:

```
#SVM Classifier (balanced class weight and gridsearch)

param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf', "linear"]}

svc2 = GridSearchCV(SVC(class_weight = 'balanced'), param_grid, refit=T


svc2.fit(scaled_X_train, y_train)
y_pred_train = svc2.predict(scaled_X_train)
y_pred_test = svc2.predict(X_test)

model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_pred

roc_curve_and_auc(svc2, scaled_X_train, scaled_X_test, y_train, y_test)

plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)

svc2.best_params_
```


Confusion Matrix for train & test set:

	Predicted	No Churn	Churn	All
0	Actual	NaN	NaN	NaN
1	No Churn	2154.0	126.0	2280.0
2	Churn	37.0	349.0	386.0
3	All	2191.0	475.0	2666.0

	Predicted	No Churn	Churn	All
0	Actual	NaN	NaN	NaN
1	No Churn	570.0	0.0	570.0
2	Churn	97.0	0.0	97.0
3	All	667.0	0.0	667.0

Classification Report for train & test set

Train set

	precision	recall	f1-score	support

In [432]:

```
svc2.best_params_
```

Out[432]:

```
{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
```

15.1.1.1 Best paramets for SVM Classifier

In [115]:

```
param_grid2 = {'C': [1],
               'gamma': [ 0.01],
               'kernel': ['rbf']}

svc3 = GridSearchCV(SVC(class_weight = 'balanced'), param_grid2, refit=

svc3.fit(scaled_X_train, y_train)
y_pred_train = clf.predict(scaled_X_train)
y_pred_test = clf.predict(scaled_X_test)

model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_prec

roc_curve_and_auc(svc3, scaled_X_train, scaled_X_test, y_train, y_test)

plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)
```



```
           True       0.13      0.11      0.12       382

       accuracy                           0.76      2666
      macro avg       0.49      0.49      0.49      2666
   weighted avg       0.75      0.76      0.76      2666


Test set
                 precision    recall  f1-score   support

          False       0.84      0.89      0.86       566
           True       0.10      0.07      0.08       101

       accuracy                           0.76       667
      macro avg       0.47      0.48      0.47       667
   weighted avg       0.73      0.76      0.75       667

--------------------------------------------------------

Cohen's Kappa for train and test set:
```

15.2 Results & Comments

The mean cross validation is 95.02%. We have a precision of 0.10 and recall of 0.07 for the test set. This model performed poorly even after parameter optimization

In [439]:
```
1 ! pip install delayed
```

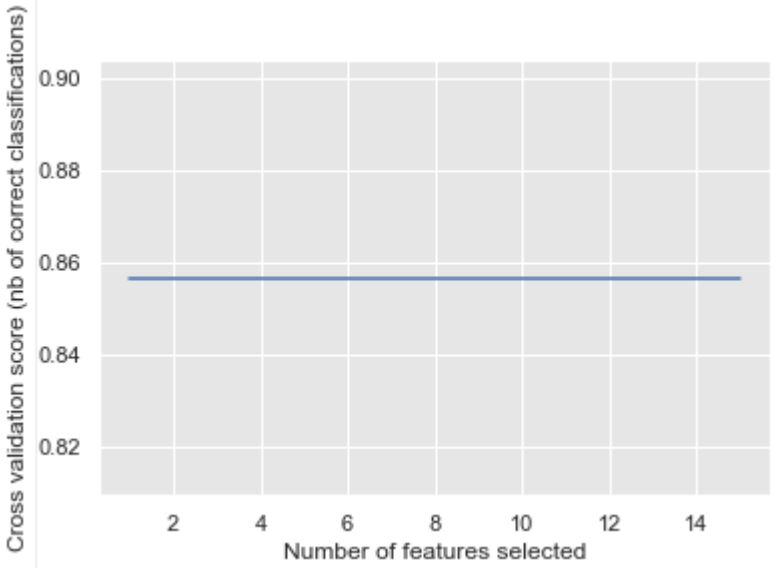
```
Collecting delayed
  Downloading delayed-0.11.0b1-py2.py3-none-any.whl (19 kB)
Collecting hiredis
  Downloading hiredis-2.0.0-cp38-cp38-win_amd64.whl (18 kB)
Collecting redis
  Downloading redis-3.5.3-py2.py3-none-any.whl (72 kB)
Installing collected packages: hiredis, redis, delayed
Successfully installed delayed-0.11.0b1 hiredis-2.0.0 redis-3.5.3
```


15.2.1 SVM with RFE

In [153]:

```
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import RFECV
from sklearn.datasets import make_classification


# Build a classification task using 3 informative features


# Create the RFE object and compute a cross-validated score.
svc = SVC(kernel="linear")
# The "accuracy" scoring is proportional to the number of correct
# classifications

min_features_to_select = 1  # Minimum number of features to consider
rfecv = RFECV(estimator=svc, step=1, cv=StratifiedKFold(2),
              scoring='accuracy',
              min_features_to_select=min_features_to_select)
rfecv.fit(scaled_X_train, y_train)

print("Optimal number of features : %d" % rfecv.n_features_)

# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(min_features_to_select,
               len(rfecv.grid_scores_) + min_features_to_select),
         rfecv.grid_scores_)
plt.show()
```

```
Optimal number of features : 1
```



In [154]:
```
1 rfecv.support_
```

Out[154]:
```
array([False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False])
```


In [155]:
```
1 pd.DataFrame(data=rfecv.support_.reshape(1,-1), columns= X.columns)
```

Out[155]:

	area code	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night calls	total night charge	total intl calls	cha
0	False	True	False	False	False	False	False	False	False	

In [156]:
```
1 rfecv.ranking_
```

Out[156]:
```
array([10,  1,  8,  7,  9,  4, 13, 15, 11, 14,  5, 12,  6,  3,  2])
```

In [157]:
```
1 pd.DataFrame(data=rfecv.ranking_.reshape(1,-1), columns= X.columns)
```

Out[157]:

	area code	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night calls	total night charge	total intl calls	cha
0	10	1	8	7	9	4	13	15	11	

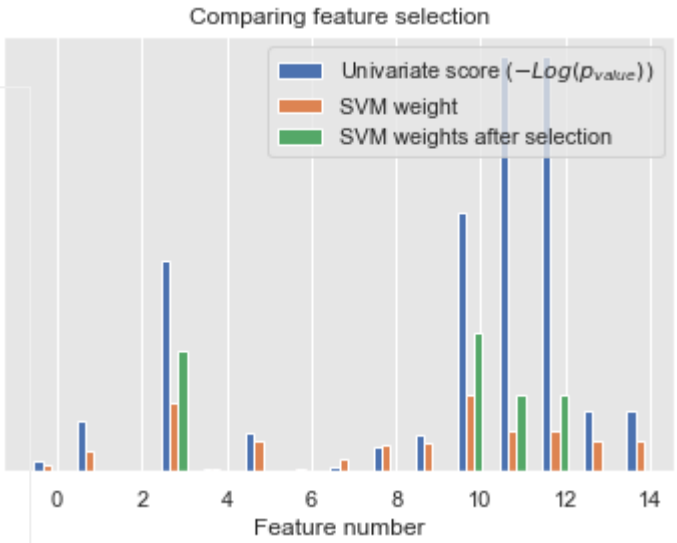
15.2.2 SVM with SelectKBest

In [158]:

```
from sklearn.feature_selection import f_regression, mutual_info_regress

from sklearn.svm import LinearSVC
from sklearn.pipeline import make_pipeline
from sklearn.feature_selection import SelectKBest, f_classif


plt.figure(1)
plt.clf()

X_indices = np.arange(X.shape[-1])

# #############################################################################
# Univariate feature selection with F-test for feature scoring
# We use the default selection function to select the four
# most significant features
selector = SelectKBest(f_classif, k=4)
selector.fit(scaled_X_train, y_train)
scores = -np.log10(selector.pvalues_)
scores /= scores.max()
plt.bar(X_indices - .45, scores, width=.2,
        label=r'Univariate score ($-Log(p_{value})$)')

# #############################################################################
# Compare to the weights of an SVM
clf = make_pipeline( LinearSVC())
clf.fit(scaled_X_train, y_train)
print('Classification accuracy without selecting features: {:.3f}'
      .format(clf.score(scaled_X_test, y_test)))

svm_weights = np.abs(clf[-1].coef_).sum(axis=0)
svm_weights /= svm_weights.sum()

plt.bar(X_indices - .25, svm_weights, width=.2, label='SVM weight')

clf_selected = make_pipeline(
        SelectKBest(f_classif, k=4), LinearSVC())

clf_selected.fit(scaled_X_train, y_train)
print('Classification accuracy after univariate feature selection: {:.3
      .format(clf_selected.score(scaled_X_test, y_test)))

svm_weights_selected = np.abs(clf_selected[-1].coef_).sum(axis=0)
svm_weights_selected /= svm_weights_selected.sum()

plt.bar(X_indices[selector.get_support()] - .05, svm_weights_selected,
        width=.2, label='SVM weights after selection')


plt.title("Comparing feature selection")
plt.xlabel('Feature number')
plt.yticks(())
plt.axis('tight')
plt.legend(loc='upper right')
plt.show()
```

Contents

```
Classification accuracy without selecting features: 0.853
Classification accuracy after univariate feature selection: 0.844
```



In [159]:

```
y_pred_test= clf_selected.predict(scaled_X_test)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

       False       0.86      0.98      0.91       566
        True       0.41      0.07      0.12       101

    accuracy                           0.84       667
   macro avg       0.63      0.53      0.52       667
weighted avg       0.79      0.84      0.79       667
```

15.3 Results & Comments

The SVC classifiers performed poorly even with RFE and SELECTK.

In [153]:
```
1 ! pip install sklearn.ensemble
```

```
ERROR: Could not find a version that satisfies the requirement sklearn.ense
mble (from versions: none)
ERROR: No matching distribution found for sklearn.ensemble
```

Contents

16 Stacking

In [160]:

```
from  sklearn.ensemble  import StackingClassifier
from sklearn.ensemble import GradientBoostingClassifier
#Stacking

#Using Decisoin Tree, Random Forest, Gradient Boosting as base learners

log_model = LogisticRegression()
dt_model = DecisionTreeClassifier(random_state=5,max_depth=10)
rf_model = RandomForestClassifier(n_estimators=100,random_state=5,max_d
gb_model = GradientBoostingClassifier(learning_rate=0.01,n_estimators=1


clf= StackingClassifier(estimators=[('tree', dt_model),('forest', rf_mo


clf.fit(scaled_X_train,y_train)

y_pred_train = clf.predict(scaled_X_train)
y_pred_test = clf.predict(scaled_X_test)

model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_pred

roc_curve_and_auc(clf, scaled_X_train, scaled_X_test, y_train, y_test)

plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)

```

```
              precision    recall  f1-score   support

       False       0.98      1.00      0.99      2284
        True       1.00      0.86      0.92       382

    accuracy                           0.98      2666
   macro avg       0.99      0.93      0.96      2666
weighted avg       0.98      0.98      0.98      2666


Test set
              precision    recall  f1-score   support

       False       0.96      0.99      0.97       566
        True       0.93      0.74      0.82       101

    accuracy                           0.95       667
   macro avg       0.94      0.87      0.90       667
weighted avg       0.95      0.95      0.95       667
```

Contents

In [50]:
```
1 clf.get_params
```

Out[50]:
```
<bound method _BaseHeterogeneousEnsemble.get_params of StackingClassifier(e
stimators=[('tree',
                                DecisionTreeClassifier(max_depth=10,
                                                       random_state=5)),
                               ('forest',
                                RandomForestClassifier(max_depth=10,
                                                       random_state=5)),
                               ('gb',
                                GradientBoostingClassifier(learning_rate=0.
01,
                                                           n_estimators=12
0,
                                                           random_state=
5))],
                   final_estimator=LogisticRegression())>
```


In [51]:
```
1 clf.final_estimator
```

Out[51]:
```
LogisticRegression()
```

16.1 Results & Comments

The mean cross validation is 95%. We have a precision of 0.93 and recall of 0.74 for the test set. We don't have overfitting. It means we are able to detect the 74% of churn customers and 93% of the customers we say 'churn' was true.

17 SGDClassifier (Lasso & Ridge regularizations)

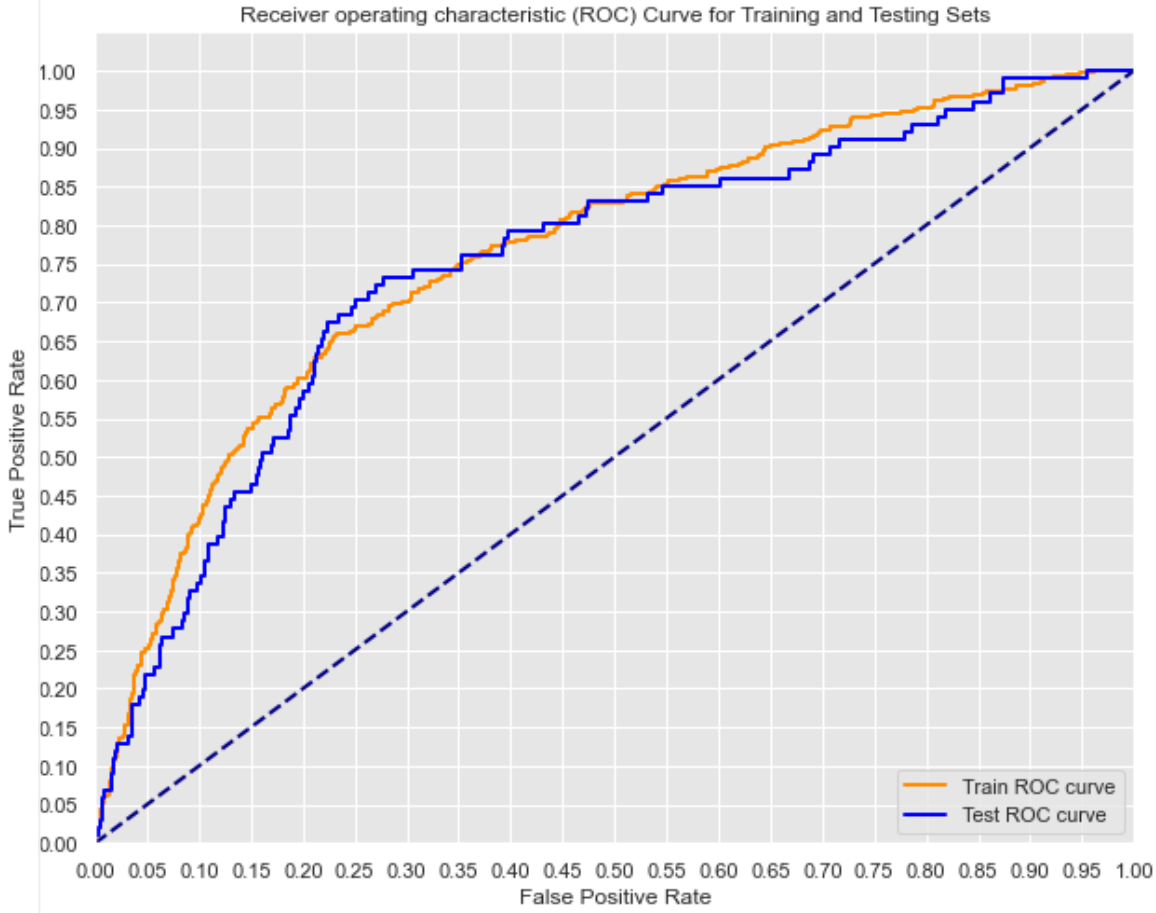
In [161]:

```
#SCDClassifier (balanced class weight)

scd = SGDClassifier(penalty='elasticnet', class_weight='balanced')
scd.fit(scaled_X_train, y_train)
y_pred_train = scd.predict(scaled_X_train)
y_pred_test = scd.predict(scaled_X_test)


model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_pred
roc_curve_and_auc(scd, scaled_X_train, scaled_X_test, y_train, y_test)

plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)
```



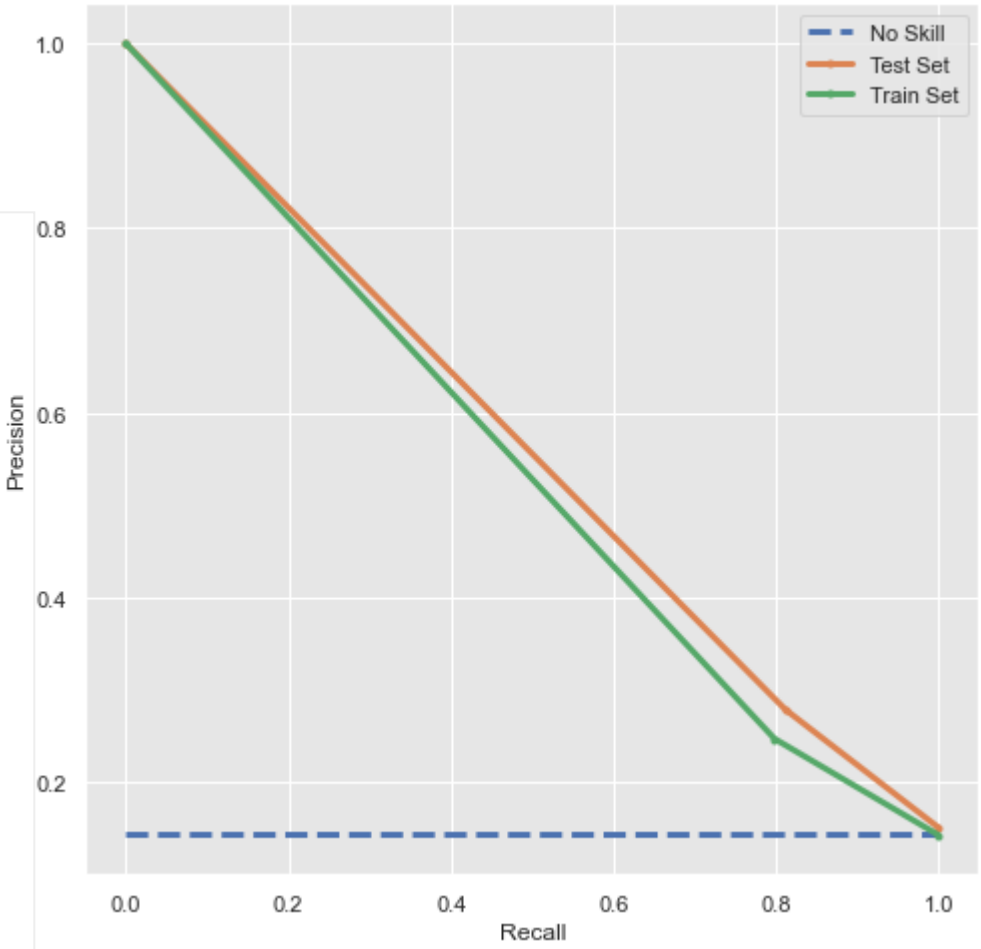
```
MODEL EVALUATION METRICS:
 -----------------------------------------------------
Confusion Matrix for train & test set:

  Predicted  No Churn   Churn     All
0    Actual       NaN     NaN     NaN
1  No Churn    1356.0   928.0  2284.0
2     Churn      77.0   305.0   382.0
3       All    1433.0  1233.0  2666.0

  Predicted  No Churn  Churn    All
0    Actual       NaN    NaN    NaN
1  No Churn     354.0  212.0  566.0
2     Churn      19.0   82.0  101.0
3       All     373.0  294.0  667.0

-----------------------------------------------------

Classification Report for train & test set

Train set
              precision    recall  f1-score   support

       False       0.95      0.59      0.73      2284
        True       0.25      0.80      0.38       382

    accuracy                           0.62      2666
   macro avg       0.60      0.70      0.55      2666
weighted avg       0.85      0.62      0.68      2666


Test set
              precision    recall  f1-score   support

       False       0.95      0.63      0.75       566
        True       0.28      0.81      0.42       101

    accuracy                           0.65       667
   macro avg       0.61      0.72      0.58       667
weighted avg       0.85      0.65      0.70       667
```

```
------------------------------------------------------

Cohen's Kappa for train and test set:
  0.2034 0.245
f2 score for train and test set:
  0.5523 0.5874
roc auc score for train and test set:
  0.6961 0.7187
Mean Cross Validation Score:
 0.9502
```



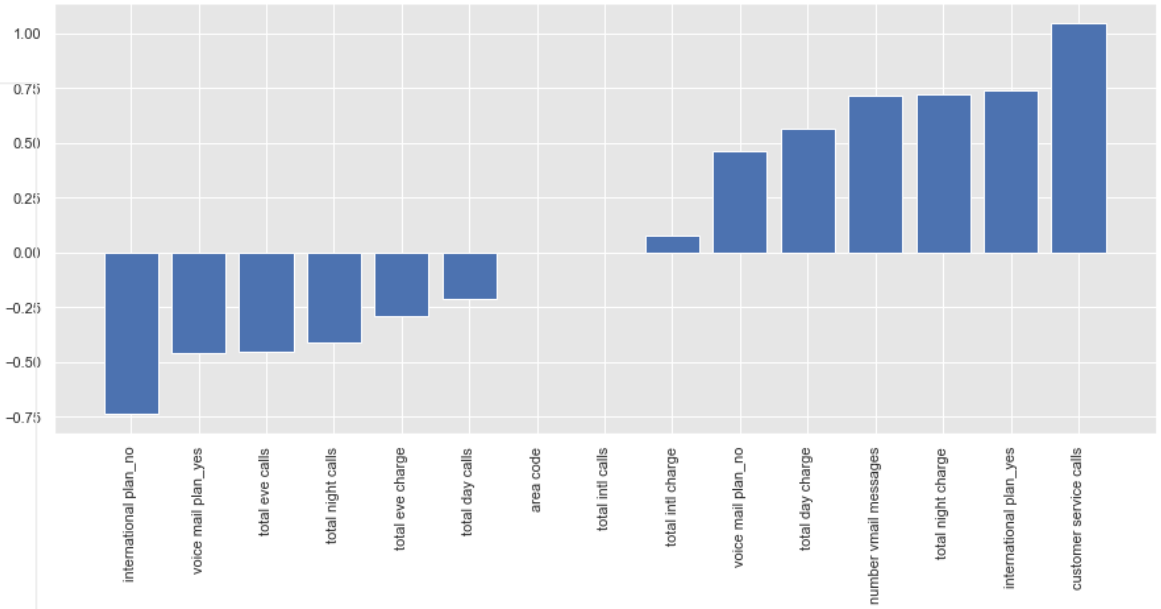

```
Training AUC: 0.76274
Testing AUC: 0.74873
```

Contents



Contents

In [162]:
```
1 plot_coefficients(scd)
```



17.1 Results & Comments

The mean cross validation is 95%. We have a precision of 0.28 and recall of 0.81 for the test set. We don't have overfitting. It means we are able to detect the 81% of churn customers and 28% of the customers we say 'churn' was true.

18 Logistic Regession - RandomizedSearchCV

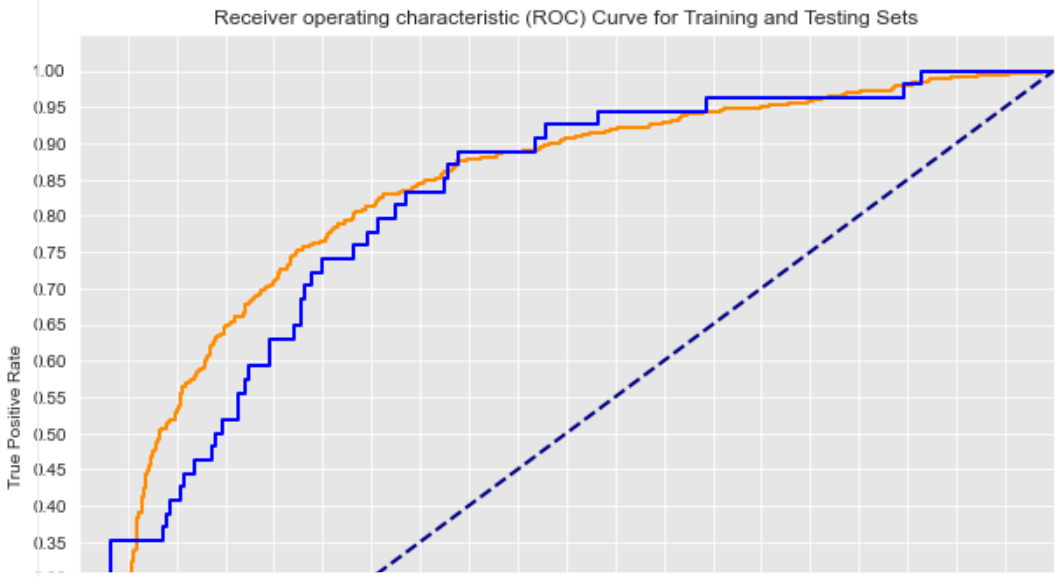
In [164]:
```
1 from sklearn.model_selection import RandomizedSearchCV
2 from scipy.stats import uniform
3
4 logistic = LogisticRegression(solver='saga', tol=1e-2, max_iter=200,
5                                random_state=0)
6 distributions = dict(C=uniform(loc=0, scale=4),
7                       penalty=['l2', 'l1'])
8 clf = RandomizedSearchCV(logistic, distributions, random_state=0)
9 search = clf.fit(scaled_X_train, y_train)
10
11 search.best_params_
```

Out[164]: {'C': 1.75034884505077, 'penalty': 'l2'}

In [165]:
```
logistic = LogisticRegression(solver='saga', tol=1e-2, max_iter=200,
                              random_state=0)
distributions = dict(C=[1.5], penalty=['l2'])

clf = RandomizedSearchCV(logistic, distributions, random_state=0)
search = clf.fit(scaled_X_train, y_train)
```

In [44]:
```
y_pred_train = clf.predict(scaled_X_train)
y_pred_test = clf.predict(scaled_X_test)


model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_pred
roc_curve_and_auc(clf, scaled_X_train, scaled_X_test, y_train, y_test)

plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)
```

```
Mean Cross Validation Score:
 0.8566
```



18.1 Results & Comments

This model had a mean cross validation of 85.66%. Only 20% of the time the churn was true with 61 % precision.

19 SGDClassifier - RandomizedSearchCV

In [167]:

```
from time import time
import scipy.stats as stats
from sklearn.utils.fixes import loguniform

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.datasets import load_digits
from sklearn.linear_model import SGDClassifier

# build a classifier
clf = SGDClassifier(loss='hinge', penalty='elasticnet',
                    fit_intercept=True)


# Utility function to report best scores
def report(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})"
                  .format(results['mean_test_score'][candidate],
                          results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]
            print("")


# specify parameters and distributions to sample from
param_dist = {'average': [True, False],
              'l1_ratio': stats.uniform(0, 1),
              'alpha': loguniform(1e-4, 1e0)}

# run randomized search
n_iter_search = 20
random_search = RandomizedSearchCV(clf, param_distributions=param_dist,
                                   n_iter=n_iter_search)

start = time()
random_search.fit(scaled_X_train, y_train)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time() - start), n_iter_search))
report(random_search.cv_results_)

```

```
RandomizedSearchCV took 0.88 seconds for 20 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.863 (std: 0.006)
Parameters: {'alpha': 0.0006626483822186211, 'average': False, 'l1_ratio':
0.503491101662517}

Model with rank: 2
Mean validation score: 0.860 (std: 0.008)
Parameters: {'alpha': 0.0005634122292363634, 'average': False, 'l1_ratio':
0.7403534721013764}

Model with rank: 3
```

Contents

```
Mean validation score: 0.857 (std: 0.001)
Parameters: {'alpha': 0.04172285339373269, 'average': False, 'l1_ratio': 0.
6586706021204276}

Model with rank: 3
Mean validation score: 0.857 (std: 0.001)
Parameters: {'alpha': 0.062422867830419115, 'average': False, 'l1_ratio':
0.9302673127586086}

Model with rank: 3
Mean validation score: 0.857 (std: 0.001)
Parameters: {'alpha': 0.07482750672804837, 'average': False, 'l1_ratio': 0.
8430808042120077}

Model with rank: 3
Mean validation score: 0.857 (std: 0.001)
Parameters: {'alpha': 0.18281454383299717, 'average': False, 'l1_ratio': 0.
3788213628609819}

Model with rank: 3
Mean validation score: 0.857 (std: 0.001)
Parameters: {'alpha': 0.03808645305940271, 'average': False, 'l1_ratio': 0.
369735002520865}

Model with rank: 3
Mean validation score: 0.857 (std: 0.001)
Parameters: {'alpha': 0.020721941661317956, 'average': False, 'l1_ratio':
0.2931709072344363}

Model with rank: 3
Mean validation score: 0.857 (std: 0.001)
Parameters: {'alpha': 0.0034212589075411893, 'average': False, 'l1_ratio':
0.6651574325714098}
```


19.1 Tuning & Optimization

In [168]:
```
1 random_search.best_params_
```

Out[168]:
```
{'alpha': 0.0006626483822186211,
 'average': False,
 'l1_ratio': 0.503491101662517}
```

In [169]:

```
# build a classifier
clf = SGDClassifier(loss='hinge', penalty='elasticnet',
                    fit_intercept=True)
param_dist = {'alpha': [0.0006626483822186211], 'average': [False], 'l1

# run randomized search
n_iter_search = 20
random_search = RandomizedSearchCV(clf, param_distributions=param_dist,
                                   n_iter=n_iter_search)

start = time()
rs=random_search.fit(scaled_X_train, y_train)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time() - start), n_iter_search))
report(random_search.cv_results_)
```

```
RandomizedSearchCV took 0.08 seconds for 20 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.858 (std: 0.002)
Parameters: {'l1_ratio': 0.503491101662517, 'average': False, 'alpha': 0.00
06626483822186211}
```

In [170]:

```
y_pred_train = rs.predict(scaled_X_train)
y_pred_test = rs.predict(scaled_X_test)


model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_pred
roc_curve_and_auc(rs, scaled_X_train, scaled_X_test, y_train, y_test)

plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)
```

```
Test set
              precision    recall  f1-score   support

       False       0.85      1.00      0.92       566
        True       0.00      0.00      0.00       101

    accuracy                           0.85       667
   macro avg       0.42      0.50      0.46       667
weighted avg       0.72      0.85      0.78       667

-----------------------------------------------------

Cohen's Kappa for train and test set:
  0.0045 0.0
f2 score for train and test set:
  0.0033 0.0
roc auc score for train and test set:
  0.5013 0.5
Mean Cross Validation Score:
 0.6972
```

Contents

19.2 Results & Comments

This model performed poorly.

20 Data Interpretation

20.1 Most Important features Accoring to DecisionTree Model

Contents

In [189]:

```
dt_param_grid2 = {
    'criterion': ['gini'],
    'max_depth': [ 6],
    'min_samples_split': [4],
    'min_samples_leaf': [10]
}

dt_grid_search2 = GridSearchCV(tree_clf, dt_param_grid2, cv=10, return_

# Fit to the data

clf=dt_grid_search2 .fit(scaled_X_train, y_train)



#Examine best parameters

# Mean training score
dt_gs_training_score2 = np.mean(clf.cv_results_['mean_train_score'])

# Mean test score
dt_gs_testing_score2 = clf.score(scaled_X_test, y_test)


print(f"Mean Training Score: {dt_gs_training_score2 :.2%}")
print(f"Mean Test Score: {dt_gs_testing_score2 :.2%}")

# Model Performance
# Test set predictions
y_pred_test =clf.predict(scaled_X_test)
y_pred_train =clf.predict(scaled_X_train)

print(classification_report(y_test,y_pred_test))

model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_pred



plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)

plot_feature_importances(tree_clf)
```

```
              True       0.89      0.67      0.77       101

    accuracy                           0.94       667
   macro avg       0.92      0.83      0.87       667
weighted avg       0.94      0.94      0.93       667

MODEL EVALUATION METRICS:
 ------------------------------------------------------

Confusion Matrix for train & test set:

  Predicted  No Churn  Churn     All
0    Actual       NaN    NaN     NaN
1  No Churn    2262.0   22.0  2284.0
2     Churn     104.0  278.0   382.0
```

Contents

```
3       All    2366.0  300.0  2666.0

  Predicted  No Churn  Churn    All
0    Actual       NaN    NaN    NaN
1  No Churn     558.0    8.0  566.0
2     Churn      33.0   68.0  101.0
```

Contents

20.1.1 Top 5 Important Features Affecting Churn Rate

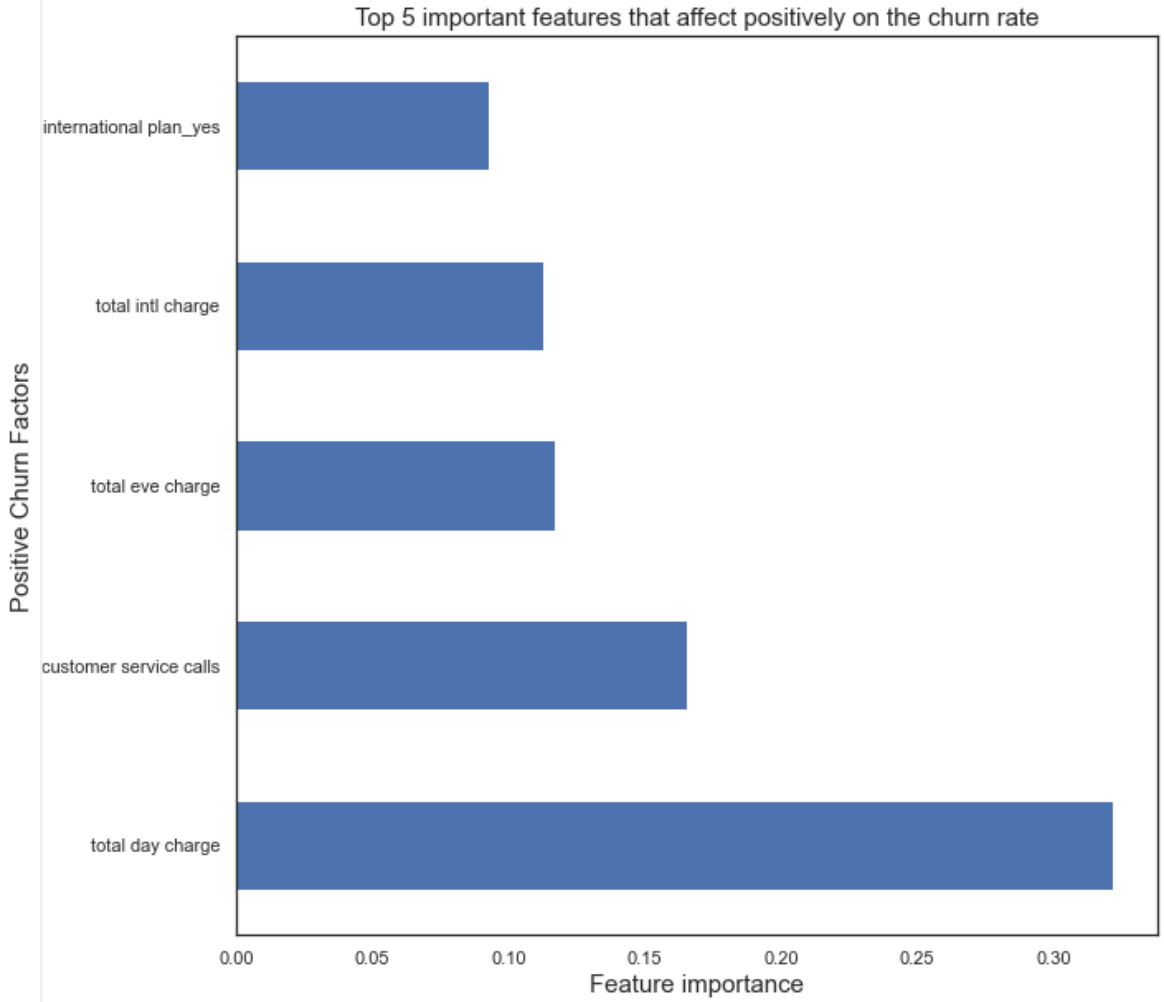
In [196]:

```
feature_importance = tree_clf.feature_importances_
print (tree_clf.feature_importances_)
feat_importances = pd.Series(tree_clf.feature_importances_, index=X.col
feat_importances = feat_importances.nlargest(5)
feat_importances.plot(kind='barh' , figsize=(10,10), color="b")

plt.xlabel('Feature importance', fontsize=15)
plt.ylabel('Positive Churn Factors', fontsize=15)
plt.title('Top 5 important features that affect positively on the churr
```

```
[0.         0.         0.         0.32193796 0.0134261  0.11687289
 0.00422496 0.02131784 0.08281552 0.11242512 0.1652558  0.
 0.0928464  0.         0.0688774 ]
```

Out[196]:

```
Text(0.5, 1.0, 'Top 5 important features that affect positively on the chur
n rate')
```



20.2 Top Positive and Negative Features According to Logistic Regression

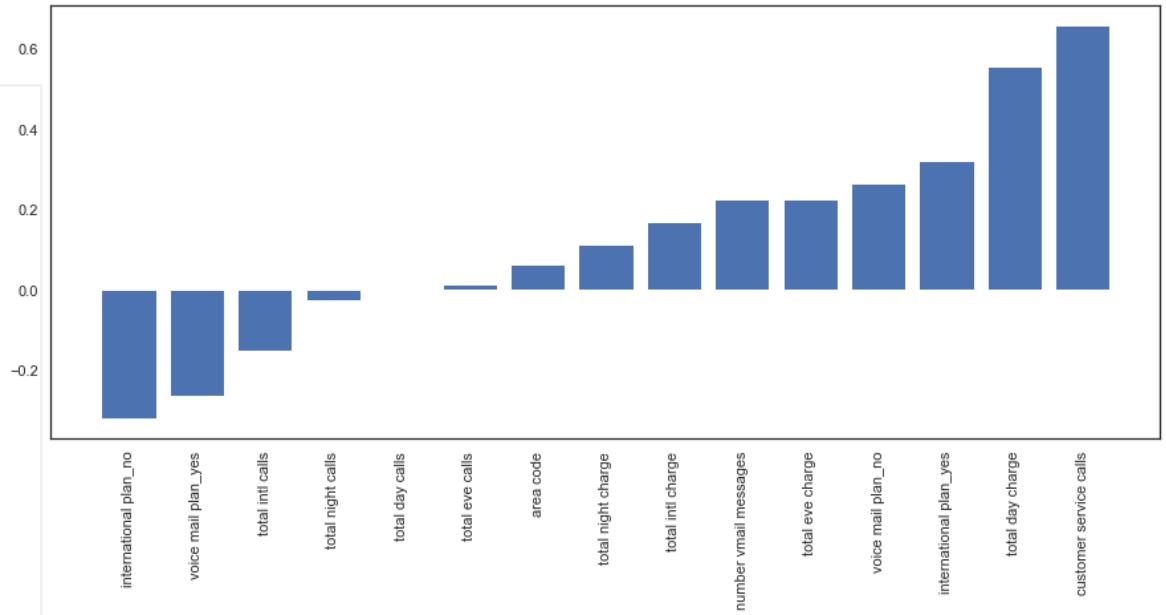
In [178]:

```
lr = LogisticRegression(fit_intercept=False, C=100,
                        solver='liblinear', class_weight='balanced')
lr.fit(scaled_X_train, y_train)
y_pred_train = lr.predict(scaled_X_train)
y_pred_test = lr.predict(scaled_X_test)


model_evaluation(scaled_X_train, scaled_X_test, y_train, y_test, y_pred

roc_curve_and_auc(lr, scaled_X_train, scaled_X_test, y_train, y_test)

plot_pr_rc_curve(y_test, y_pred_test, y_train, y_pred_train)
```

Contents


```
0   Actual       NaN     NaN     NaN
1  No Churn    1426.0   858.0  2284.0
2     Churn      48.0   334.0   382.0
3       All    1474.0  1192.0  2666.0

  Predicted  No Churn  Churn    All
0   Actual       NaN    NaN    NaN
1  No Churn     357.0  209.0  566.0
2     Churn      10.0   91.0  101.0
3       All     367.0  300.0  667.0

------------------------------------------------------

Classification Report for train & test set

Train set
              precision    recall  f1-score   support

       False       0.97      0.62      0.76      2284
        True       0.28      0.87      0.42       382
```

In [179]:
```
1 plot_coefficients(lr)
```



Contents

In [180]:
```
1 lr.coef_.round(2)
```

Out[180]:
```
array([[ 0.06,  0.23,  0.  ,  0.56,  0.01,  0.23, -0.03,  0.11, -0.15,
         0.17,  0.66, -0.32,  0.32,  0.26, -0.26]])
```

In [181]:
```
1 from collections import OrderedDict
2 coefficients =pd.Series(lr.coef_[0], index=X.columns.values).to_dict()
3
4 abs_coefficients = {k: abs(v) for k, v in coefficients.items()}
5 coefs_ranked =OrderedDict(sorted(coefficients.items(), key=lambda kv: k
6 top_5 = list(coefs_ranked)[:5]
7 print("The 5 most important coefficients are:")
8 print(" / ".join(top_5))
```

```
The 5 most important coefficients are:
international plan_no / voice mail plan_yes / total intl calls / total nigh
t calls / total day calls
```

In [182]:
```
1 coefs_pos = {k: abs(v) for k, v in coefs_ranked.items() if v > 0}
2 coefs_neg = {k: abs(v) for k, v in coefs_ranked.items() if v < 0}
3 coefs_pos = OrderedDict(sorted(coefs_pos.items(), key=lambda kv: kv[1],
```

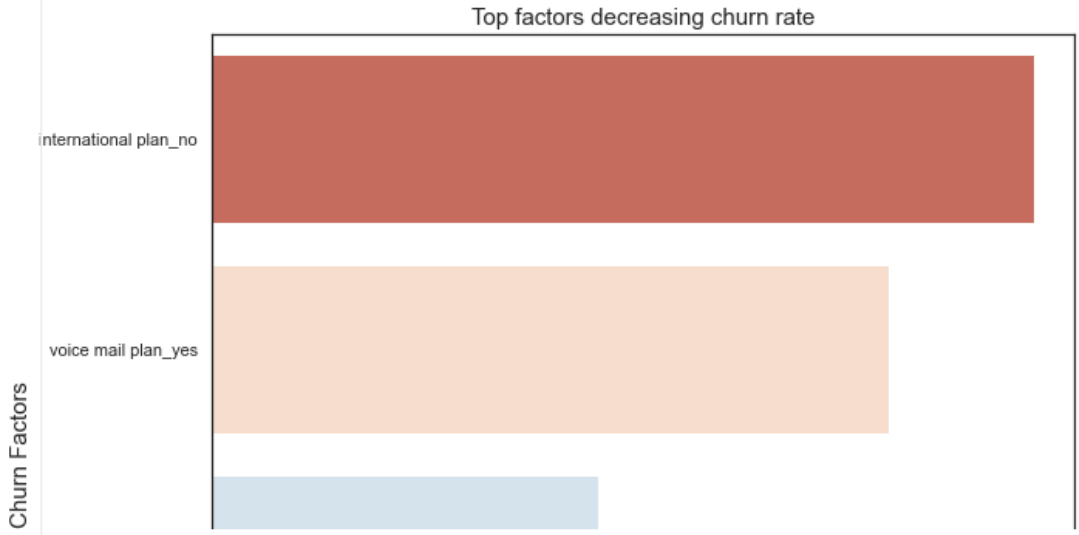
20.2.1 Top Negative Factors affectig Churn Rate

In [183]:

```
sns.set(rc={"figure.figsize":(10,10)})
sns.set_style("white")
sns.barplot(x=list(coefs_neg.values()), y=list(coefs_neg.keys()), palet
plt.title('Top factors decreasing churn rate', fontsize=15)
plt.xlabel("Importance Index", fontsize=15)
plt.ylabel('Churn Factors', fontsize=15)
print("\n\nBiggest Negative Factors on Churn")
```

Contents


```
Biggest Negative Factors on Churn
```



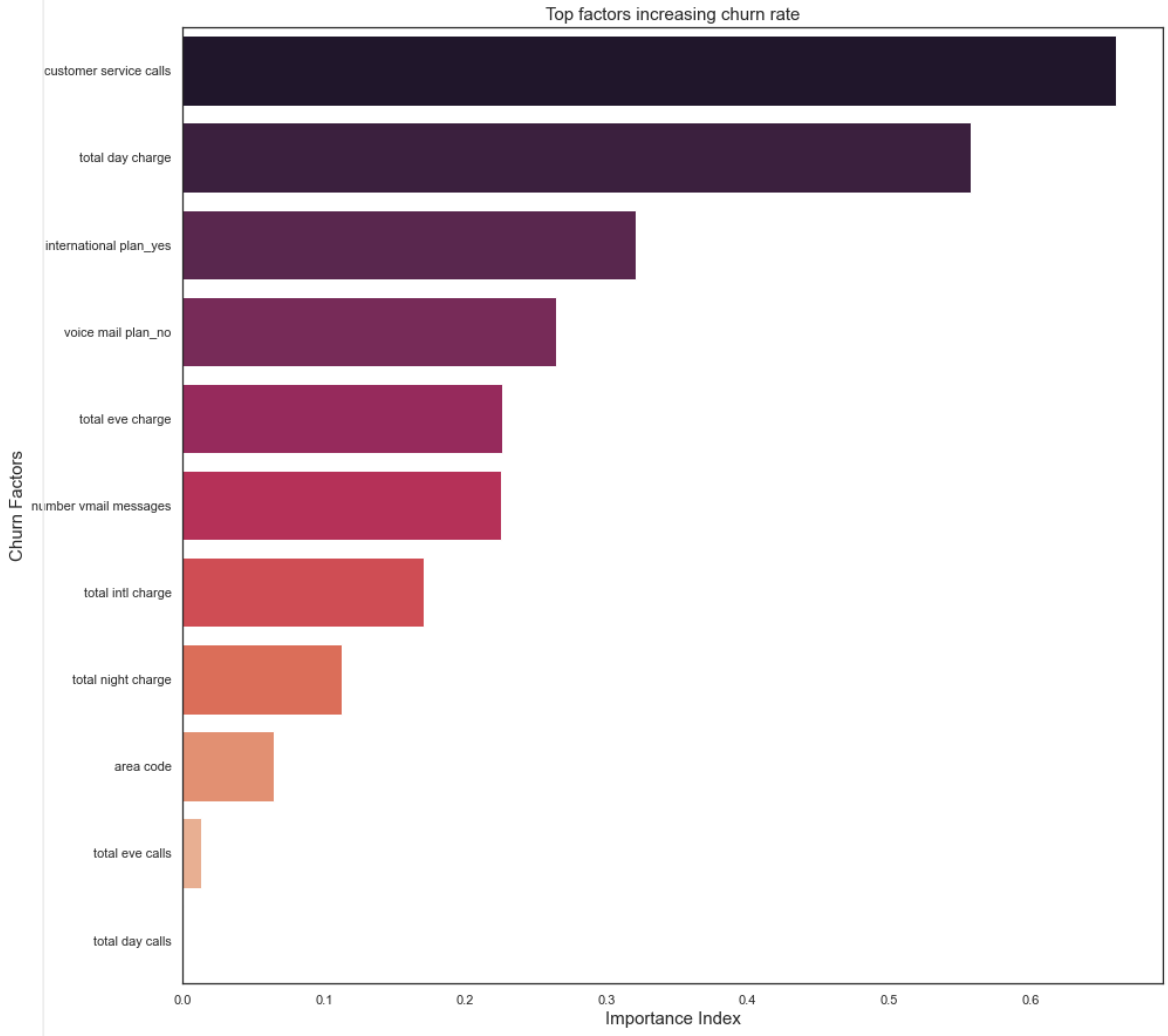
20.2.2 Top Positive Fators Affectig Churn Rate

In [184]:

```
sns.set(rc={"figure.figsize":(15,15)})
sns.set_style("white")
sns.barplot(x=list(coefs_pos.values()), y=list(coefs_pos.keys()), palet
plt.title('Top factors increasing churn rate', fontsize=15)
plt.xlabel("Importance Index", fontsize=15)
plt.ylabel('Churn Factors', fontsize=15)
print("\n\nBiggest Positive Factors on Churn rate")
```

Contents


```
Biggest Positive Factors on Churn rate
```



Contents

21 Conclusion

The best performing machine learnng model to predict the churn rate is for Stacking Classifier with recall of 74% and precision of 93%. However, we can not interpret this model and extract the important features.

NB Guassian, Adaboost and Gradient Boost Classifier had the optimal recall and precision of 100%

We decided to use DecisionTree classifier which performed well with 65% recall and 87% precision to intepret the data and extract the features.

Adaboost and GradientBoost, DecisionTree and RandoForest intpreted the data similarly.

Factors that contibute to churn of customers are:

```
1)Total day charge of calls,
2)customer service calls,
3)Total evening charge,
4)Total International Charge,
5)Total International calls,
```

Factors that contribute to the retention of customers and churn prevention are:

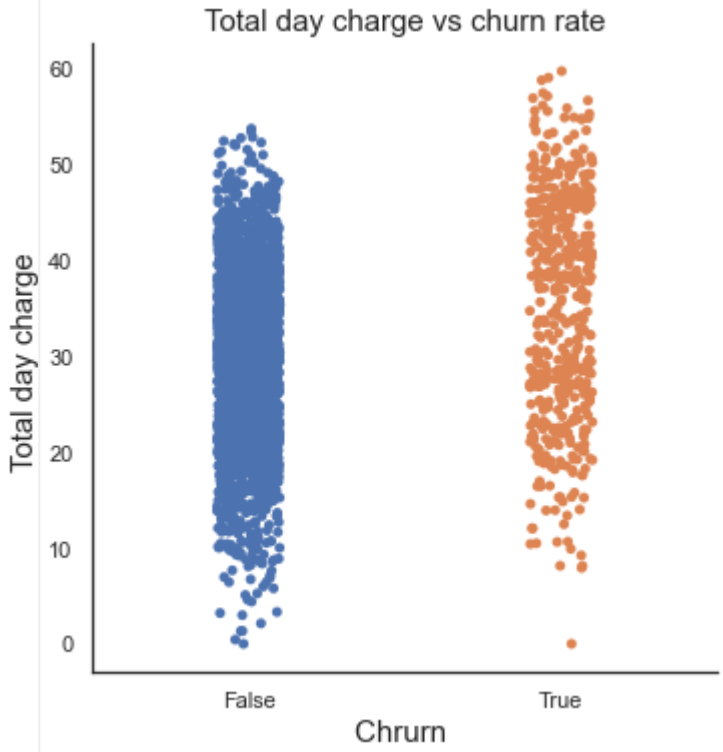
```
1) International plan = no
2) Voice mail plan = yes
3) Total International calls
```

21.1 Increasing churn factors

21.1.1 Total day charge

In [10]:

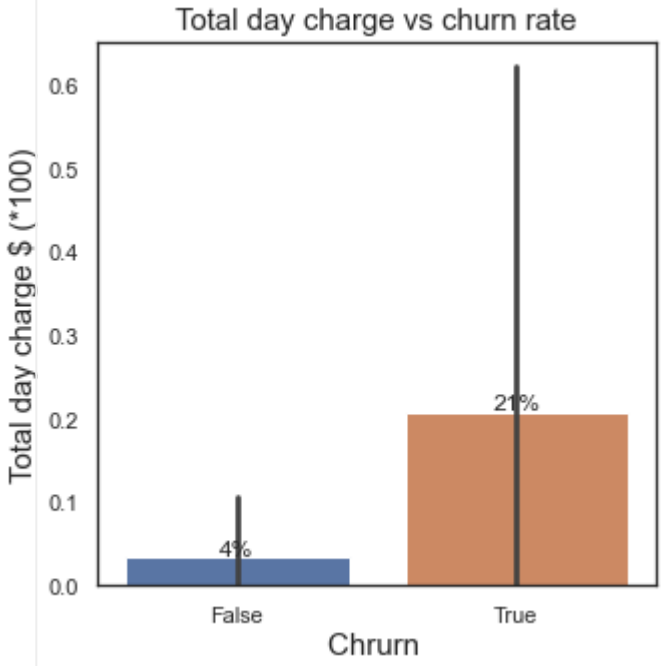
```
sns.set_style("white")
g=sns.catplot(data=df, x="churn", y="total day charge")
plt.title('Total day charge vs churn rate', fontsize=15)
plt.xlabel("Chrurn ", fontsize=15)
plt.ylabel('Total day charge ', fontsize=15)

spots = zip(g.ax.patches)
for spot in spots:
    height = spot[0].get_height()
    g.ax.text(spot[0].get_x(), height+3, '{:1.2f}'.format(churn.value_c
```



Contents

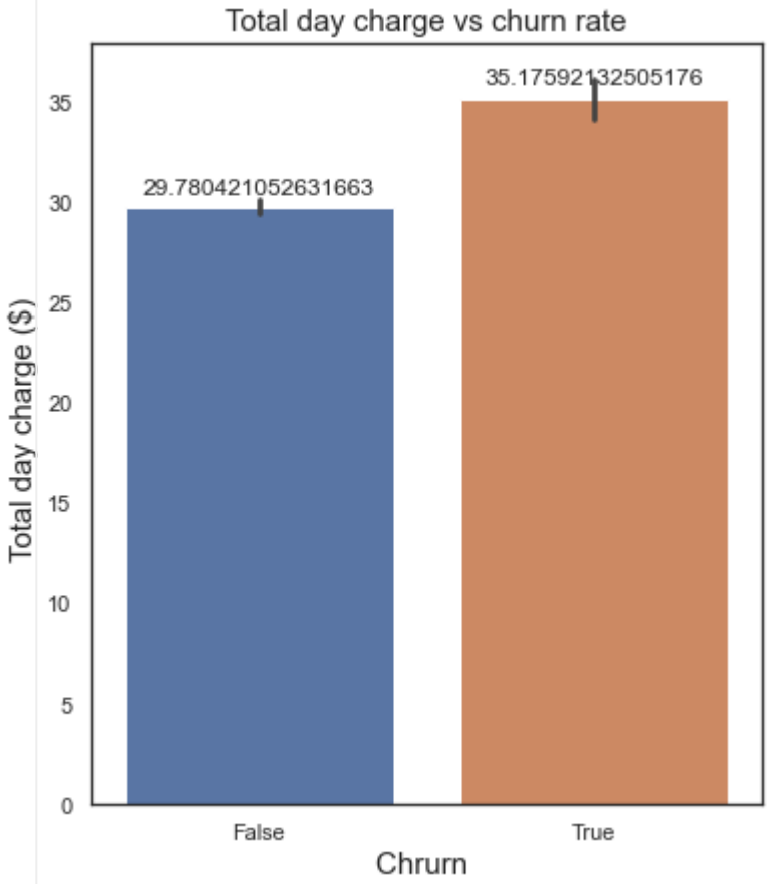
In [266]:

```
sns.set(rc={"figure.figsize":(5,5)})
sns.set_style("white")
ax=sns.barplot( x="churn", y="total day charge", data=df, estimator=lam
sns.set(rc={"figure.figsize":(5,5)})
plt.title('Total day charge vs churn rate', fontsize=15)
plt.xlabel("Chrurn ", fontsize=15)
plt.ylabel('Total day charge $ (*100)', fontsize=15)
for p in ax.patches:
    width = p.get_width()
    height = p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height:.0%}', (x + width/2, y + height*1.02), ha='ce
```



In [12]:

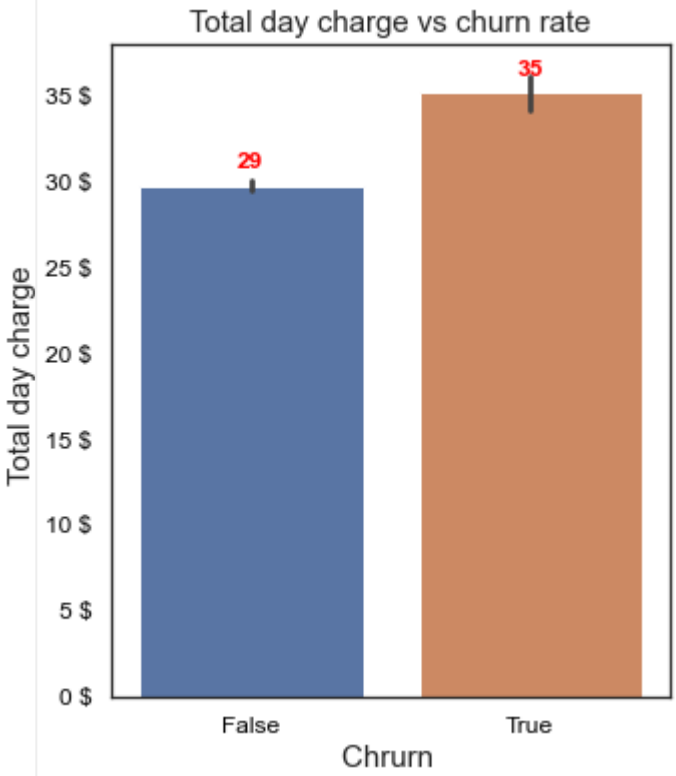
```
sns.set(rc={"figure.figsize":(6,7)})
sns.set_style("white")
ax=sns.barplot(data=df, x="churn", y="total day charge")
plt.title('Total day charge vs churn rate', fontsize=15)
plt.xlabel("Chrurn ", fontsize=15)
plt.ylabel('Total day charge ($) ', fontsize=15)
for p in ax.patches:
    width = p.get_width()
    height = p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height}', (x + width/2, y + height*1.02), ha='center
```



Contents

In [42]:

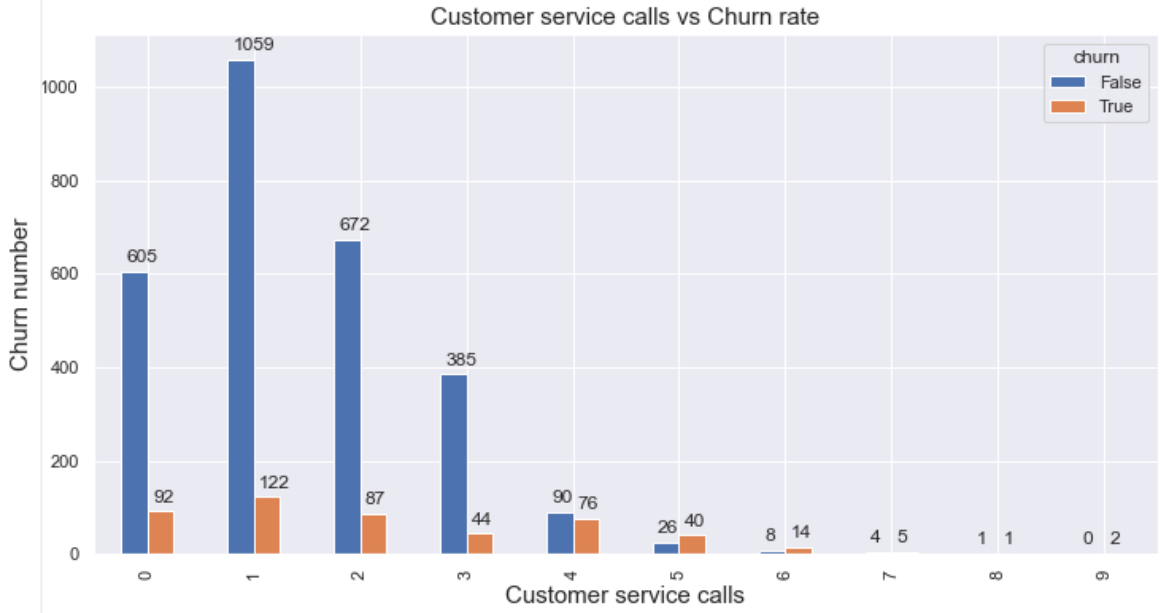
```
sns.set(rc={"figure.figsize":(5,6)})
sns.set_style("white")
ax=sns.barplot(data=df, x="churn", y="total day charge")
plt.title('Total day charge vs churn rate', fontsize=15)
plt.xlabel("Chrurn ", fontsize=15)
plt.ylabel('Total day charge ', fontsize=15)

yticks = np.arange(0, len(df['total day charge']) ,1)




ylabel=["0 $", "5 $", "10 $", "15 $", "20 $", "25 $", "30 $", "35 $"]
xlabel=["False", "True"]
ax.set_yticklabels(ylabel,  c= 'black', fontsize=12)
ax.set_xticklabels(xlabel,  c= 'black', fontsize=12)

for i in ax.patches:
        ax.text(i.get_x()+.4,i.get_height()+1,int(i.get_height()), colo


```

21.1.2 Customer service calls

In [216]:
```
cs=df.groupby(["customer service calls", "churn"]).size().unstack().plo
for i in cs.patches:
    cs.text(i.get_x()+0.05, i.get_height()+20,int(i.get_height()))

plt.xlabel('Customer service calls', fontsize=15)
plt.ylabel('Churn number', fontsize=15)
plt.title('Customer service calls vs Churn rate', fontsize=15)
```

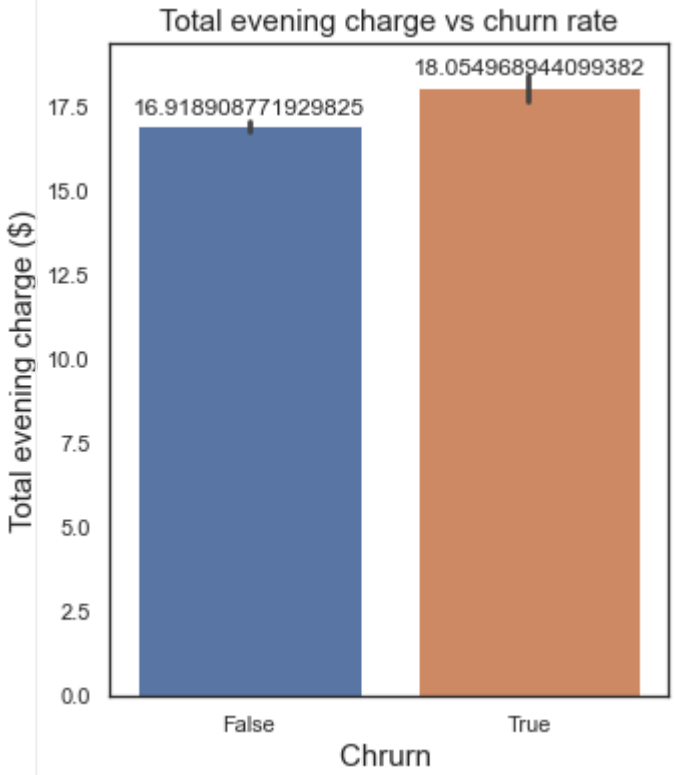
Out[216]: Text(0.5, 1.0, 'Customer service calls vs Churn rate')



21.1.3 Total evening charge

In [300]:

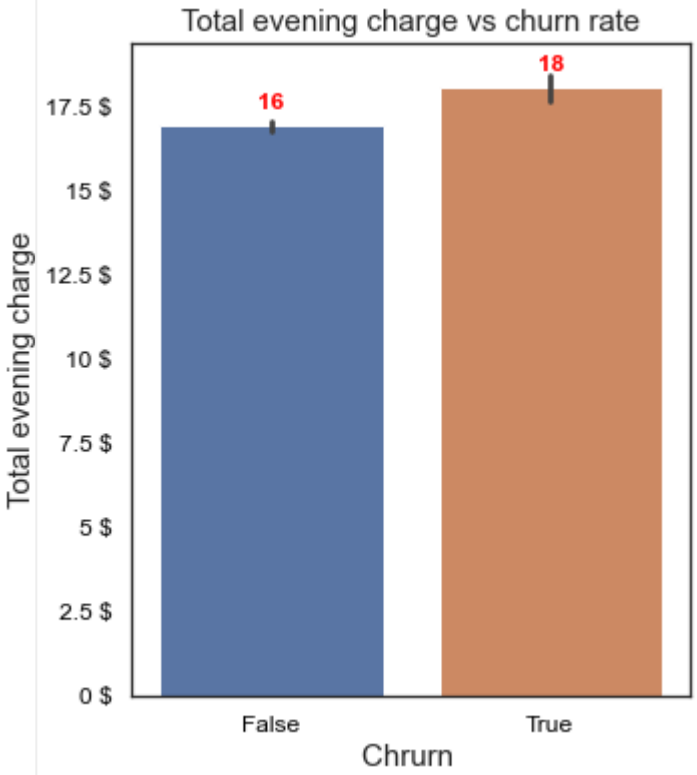
```
sns.set(rc={"figure.figsize":(5,6)})
sns.set_style("white")
ax=sns.barplot(data=df, x="churn", y="total eve charge")
#ax2=ax.twiny()
plt.title('Total evening charge vs churn rate', fontsize=15)
plt.xlabel("Chrurn ", fontsize=15)
plt.ylabel('Total evening charge ($)', fontsize=15)

#ax2.set_yticklabels(('$', '$', '$', '$', '$'))
for p in ax.patches:
    width = p.get_width()
    height = p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height}', (x + width/2, y + height*1.02), ha='center
```



Contents

Contents

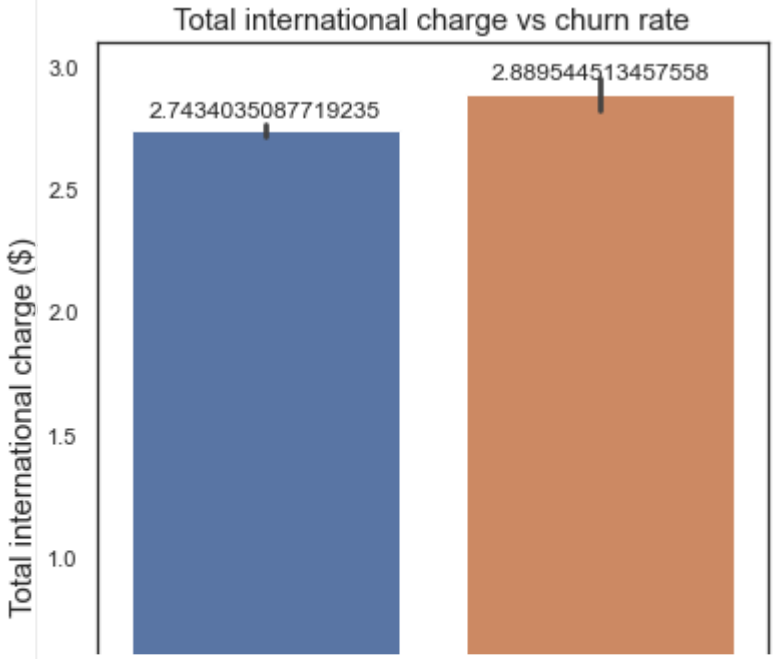

```
sns.set(rc={"figure.figsize":(5,6)})
sns.set_style("white")
ax=sns.barplot(data=df, x="churn", y="total eve charge")

yticks = np.arange(0, len(df['total eve charge']) ,1)


plt.title('Total evening charge vs churn rate', fontsize=15)
plt.xlabel("Chrurn ", fontsize=15)
plt.ylabel('Total evening charge', fontsize=15)

ylabel=["0 $", "2.5 $", "5 $", "7.5 $", "10 $", "12.5 $", "15 $", "17.5 $
xlabel=["False", "True"]
ax.set_yticklabels(ylabel,  c= 'black', fontsize=12)
ax.set_xticklabels(xlabel,  c= 'black', fontsize=12)

for i in ax.patches:
        ax.text(i.get_x()+.4,i.get_height()+.5,int(i.get_height()), color
```



21.1.4 Total internatioanl charge

In [223]:

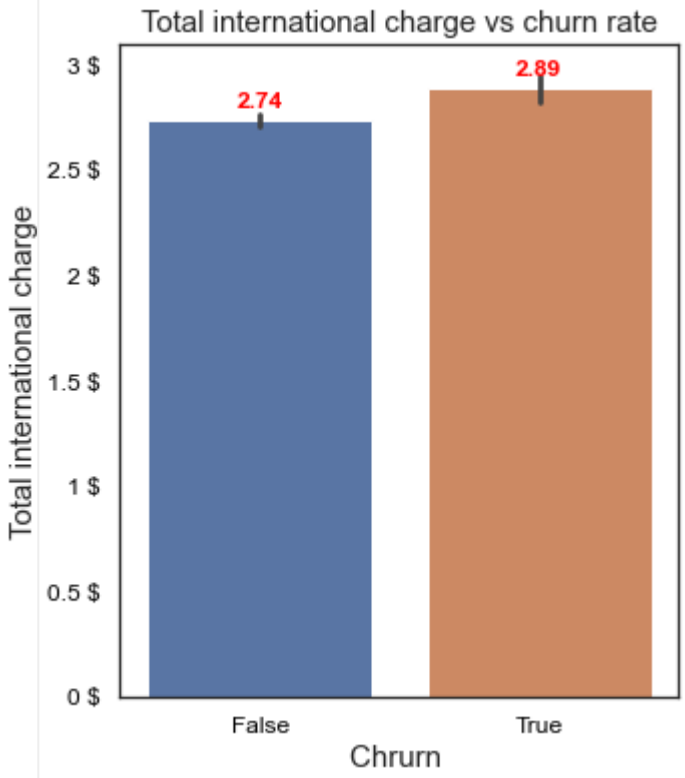
```
sns.set(rc={"figure.figsize":(6,7)})
sns.set_style("white")
ax=sns.barplot(data=df, x="churn", y="total intl charge")
plt.title('Total international charge vs churn rate', fontsize=15)
plt.xlabel("Chrurn ", fontsize=15)
plt.ylabel('Total international charge ($) ', fontsize=15)

for p in ax.patches:
    width = p.get_width()
    height = p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height}', (x + width/2, y + height*1.02), ha='center
```



Contents

In [32]:

```
sns.set(rc={"figure.figsize":(5,6)})
sns.set_style("white")
ax=sns.barplot(data=df, x="churn", y="total intl charge")
plt.title('Total international charge vs churn rate', fontsize=15)
plt.xlabel("Chrurn ", fontsize=15)
plt.ylabel('Total international charge', fontsize=15)

yticks = np.arange(0, len(df['total intl charge']) ,1)




ylabel=["0 $", "0.5 $", "1 $", "1.5 $", "2 $", "2.5 $", "3 $"]
xlabel=["False", "True"]
ax.set_yticklabels(ylabel,  c= 'black', fontsize=12)
ax.set_xticklabels(xlabel,  c= 'black', fontsize=12)

for p in ax.patches:
    width = p.get_width()
    height = p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{round(height,2)}', (x + width/2, y + height*1.02), c

```

Contents

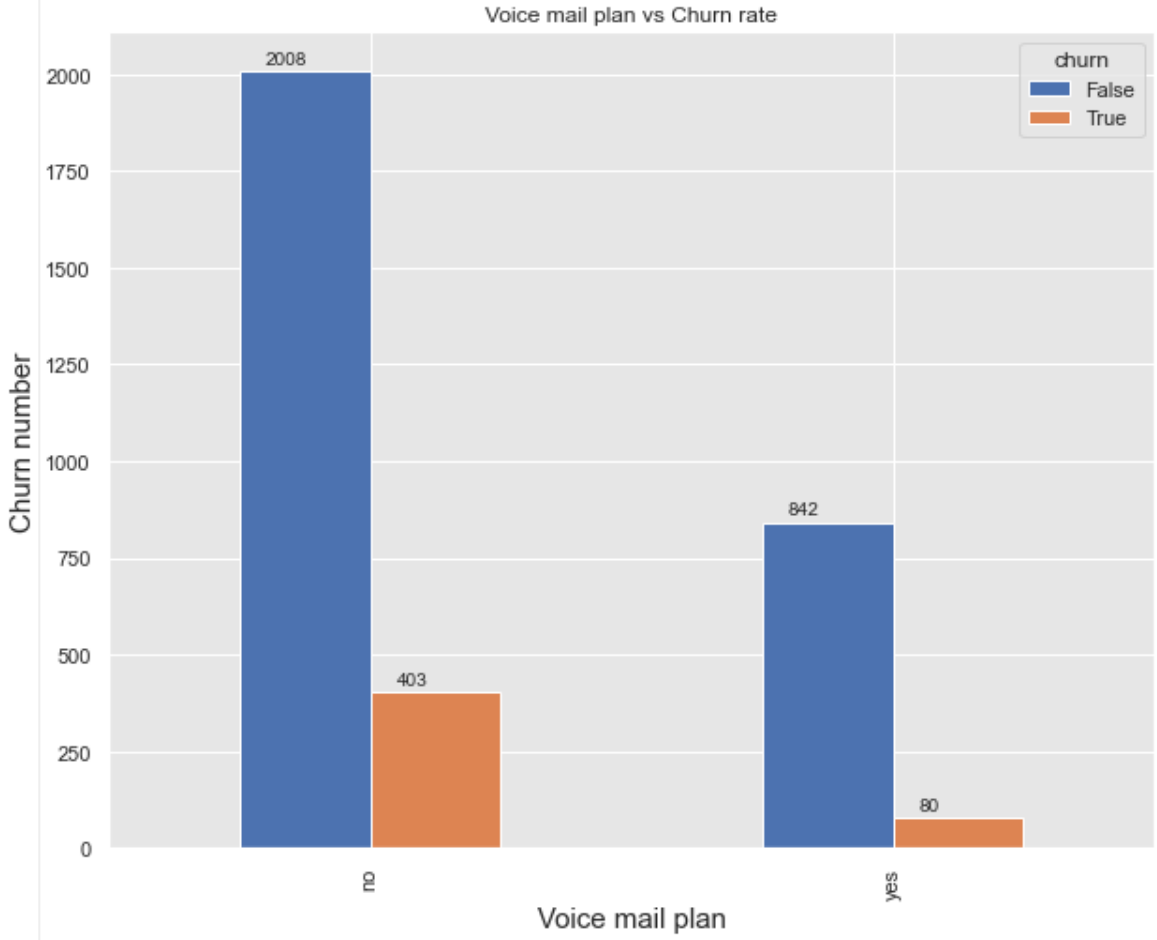
21.2 Decreasing churn factors

21.2.1 Voice mail plan

In [256]:

```
ax=df.groupby(["voice mail plan","churn"]).size().unstack().plot(kind='
for i in ax.patches:
    ax.text(i.get_x()+0.05, i.get_height()+20,str(i.get_height()))

plt.xlabel('Voice mail plan', fontsize=15)
plt.ylabel('Churn number', fontsize=15)
plt.title('Voice mail plan vs Churn rate')
```

Out[256]: Text(0.5, 1.0, 'Voice mail plan vs Churn rate')



In [228]:

```
df.columns
```

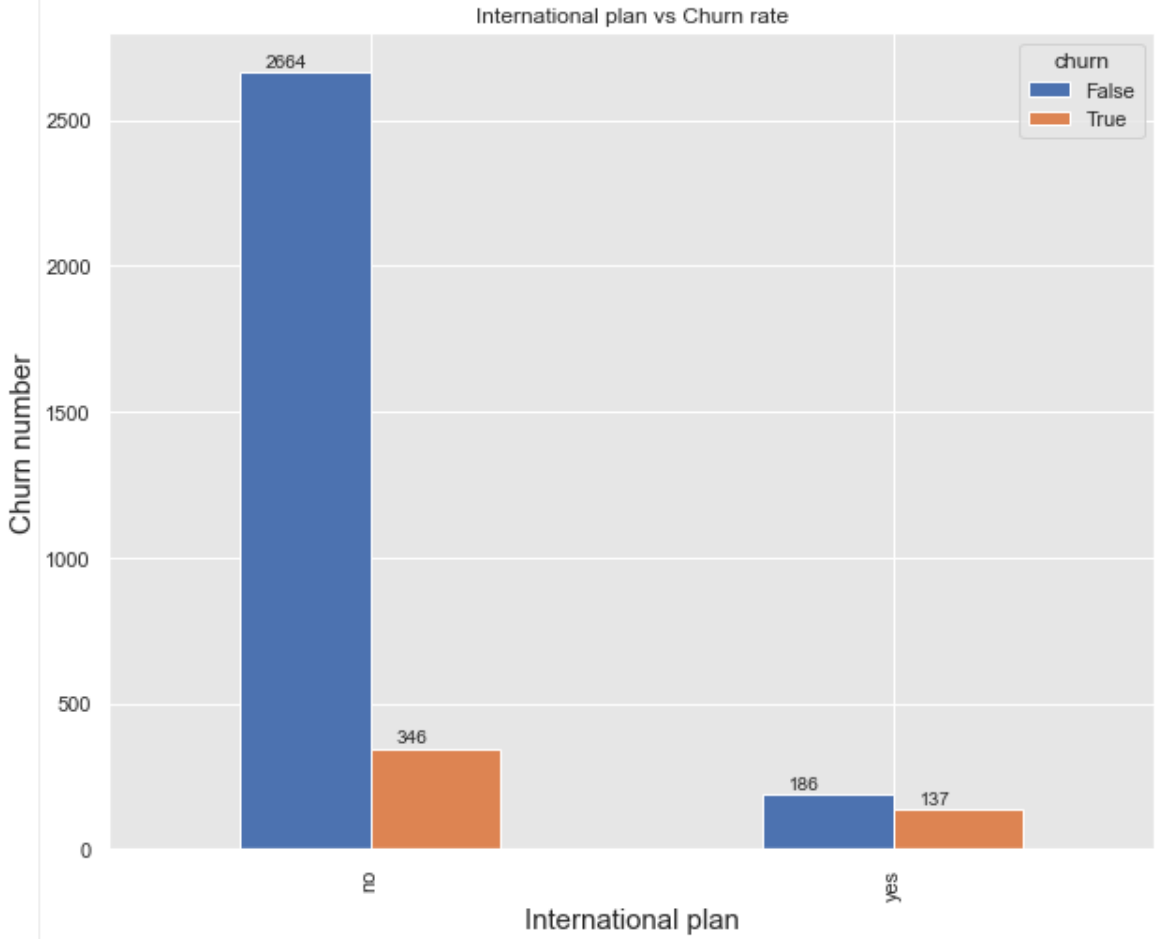
Out[228]:

```
Index(['area code', 'number vmail messages', 'total day calls',
       'total day charge', 'total eve calls', 'total eve charge',
       'total night calls', 'total night charge', 'total intl calls',
       'total intl charge', 'customer service calls', 'churn',
       'international plan_no', 'international plan_yes', 'voice mail plan_
no',
       'voice mail plan_yes'],
      dtype='object')
```

21.2.2 International plan

In [113]:
```
ac=df.groupby(['international plan',"churn"]).size().unstack().plot(kir
for i in ac.patches:
    ac.text(i.get_x()+0.05, i.get_height()+20,str(i.get_height()))
plt.xlabel('International plan', fontsize=15)
plt.ylabel('Churn number', fontsize=15)
plt.title('International plan vs Churn rate')
```

Out[113]: Text(0.5, 1.0, 'International plan vs Churn rate')



22 Recomendations

1) Decrease call charge for day and special charges for evening calls.

2) Have a better international plan.

3) Customer service calls is a measure usually for bad experience. Look what are the main causes of complaints and improve the quality of phone service.

4) Keep up the good work with the voice mail plan which is the best feature that keeps customers in the company.

5) Compete with other companies on this strongly recommended feature"voice mail plan" and make better offers for the customers. This can attract customers from competeing rivals

23 Future Work

More important data is needed to predict the churn of customers.

1) Competitor Information

Customer churn usually occurs when a rival company makes a good offer to our customers Comparison of other companies phone charges (day, night, international) and internet service will be good fator to be used in the models to predict the churn of customers.

2) Internet package information

Internet speed informaction, charge of internet per usage, the internet technology (fiber optic), internet package offer details including speed, data usage, TV streaming ...ect. All this information is important to include in our customer churn models.

3) Contract Information

Month to month contract, 6 month contract, a year contact or 2 years contarct. This information is important to know and definitely affects the churn or retention of customers.

5) Payment method

How the customers pay for the sevice: mailed check, automatic credit card payments, electronic check, automatic bank transfers. The less complex the payment method is, the more likely customers will not churn???

6) Complaint data

Text data from e-mails and customer service phone calls can be used to find the possible complaints and solve them before they cause customer churn.

7) Tenure data

Tenure data of customers is important throughout the years which can show us loyality and churn of customers.

In [ ]:
```
1
```