In [1]:
```
# Importa os pacotes principais
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
from sklearn.model_selection import train_test_split
%matplotlib inline
```

In [2]:
```
# Carrega o dataset
dftreino = pd.read_csv('projeto4_telecom_treino.csv')
dfteste = pd.read_csv('projeto4_telecom_teste.csv')
dftreino['dadosTreino'] = 1
dfteste['dadosTreino'] = 0
df = pd.concat([dftreino, dfteste])
```

Análise exploratória

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

Out[3]:

	Unnamed: 0	state	account_length	area_code	international_plan	voice_mail_plan	number_
0	1	KS	128	area_code_415	no	yes	
1	2	OH	107	area_code_415	no	yes	
2	3	NJ	137	area_code_415	no	no	
3	4	OH	84	area_code_408	yes	no	
4	5	OK	75	area_code_415	yes	no	

5 rows × 22 columns

In [4]:
```
# Tratamentos iniciais
# Retira a coluna de index
df = df.drop('Unnamed: 0', axis=1)
# Modifica as colunas com no e yes para 0 e 1
colunas = ['international_plan','voice_mail_plan','churn']
df[colunas] = df[colunas].applymap(lambda x: 0 if x=='no' else 1)
dftreino[colunas] = dftreino[colunas].applymap(lambda x: 0 if x=='no' else 1)
```

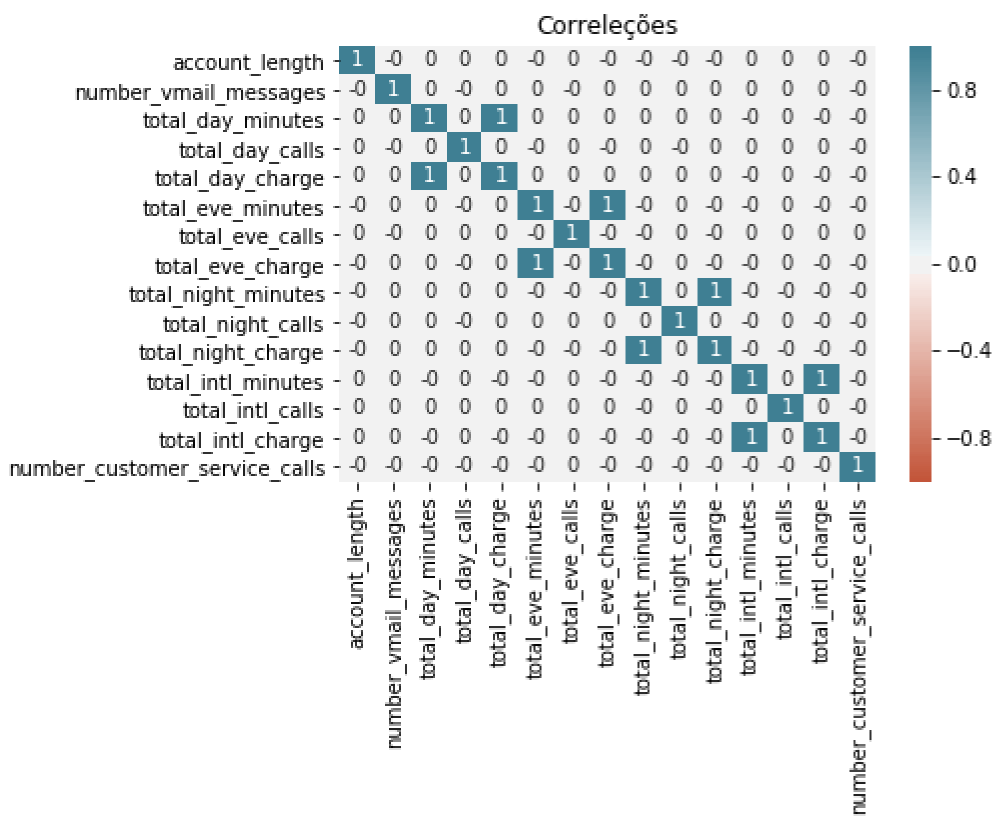
In [5]:
```
# Distribuição de classes. O Data set está desbalanceado, precisará de tratame
nto no momomento de modelagem
churnDist = round(dftreino.groupby('churn').size() / dftreino['churn'].count()
* 100, ndigits=1)
churnDist
```

Out[5]:
```
churn
0    85.5
1    14.5
dtype: float64
```

In [7]:
```
# Cálculo de correlação
colunas = ['account_length', 'number_vmail_messages', 'total_day_minutes', 'to
tal_day_calls', 'total_day_charge', 'total_eve_minutes', 'total_eve_calls', 't
otal_eve_charge', 'total_night_minutes', 'total_night_calls', 'total_night_cha
rge', 'total_intl_minutes', 'total_intl_calls', 'total_intl_charge', 'number_c
ustomer_service_calls']
dfCorrMatrix = np.corrcoef(dftreino[colunas], rowvar = False)
dfCorrMatrix = np.nan_to_num(dfCorrMatrix)
dfCorrMatrix = np.around(dfCorrMatrix,decimals=1)
```

In [8]:
```
corrGrafico = sns.heatmap(
    dfCorrMatrix, annot=True,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200))
corrGrafico.set(title='Correleções')
corrGrafico.set_xticklabels(colunas, rotation=90)
corrGrafico.set_yticklabels(colunas, rotation=0)
```

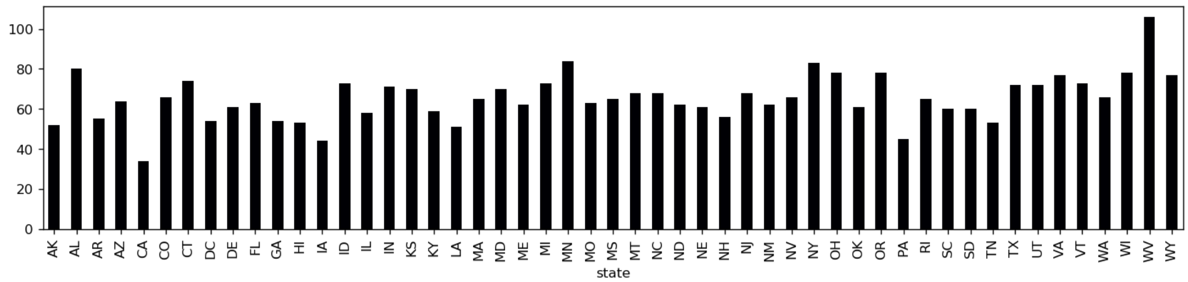
Out[8]:
```
[Text(0,0.5,'account_length'),
 Text(0,1.5,'number_vmail_messages'),
 Text(0,2.5,'total_day_minutes'),
 Text(0,3.5,'total_day_calls'),
 Text(0,4.5,'total_day_charge'),
 Text(0,5.5,'total_eve_minutes'),
 Text(0,6.5,'total_eve_calls'),
 Text(0,7.5,'total_eve_charge'),
 Text(0,8.5,'total_night_minutes'),
 Text(0,9.5,'total_night_calls'),
 Text(0,10.5,'total_night_charge'),
 Text(0,11.5,'total_intl_minutes'),
 Text(0,12.5,'total_intl_calls'),
 Text(0,13.5,'total_intl_charge'),
 Text(0,14.5,'number_customer_service_calls')]
```



In [9]:
```
# Vamos excluir as variáveis com grande correlação
df = df.drop(['total_day_charge','total_eve_charge', 'total_night_charge', 'to
tal_intl_charge'], axis=1)
```

In [10]:
```
# Distribuição do dataset por estados parece balanceada.
dftemp = dftreino.groupby('state').size()
fig = plt.figure(figsize=(15, 3), dpi = 120)
dftemp.plot('bar', colormap='magma')
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22f8d801d08>
```

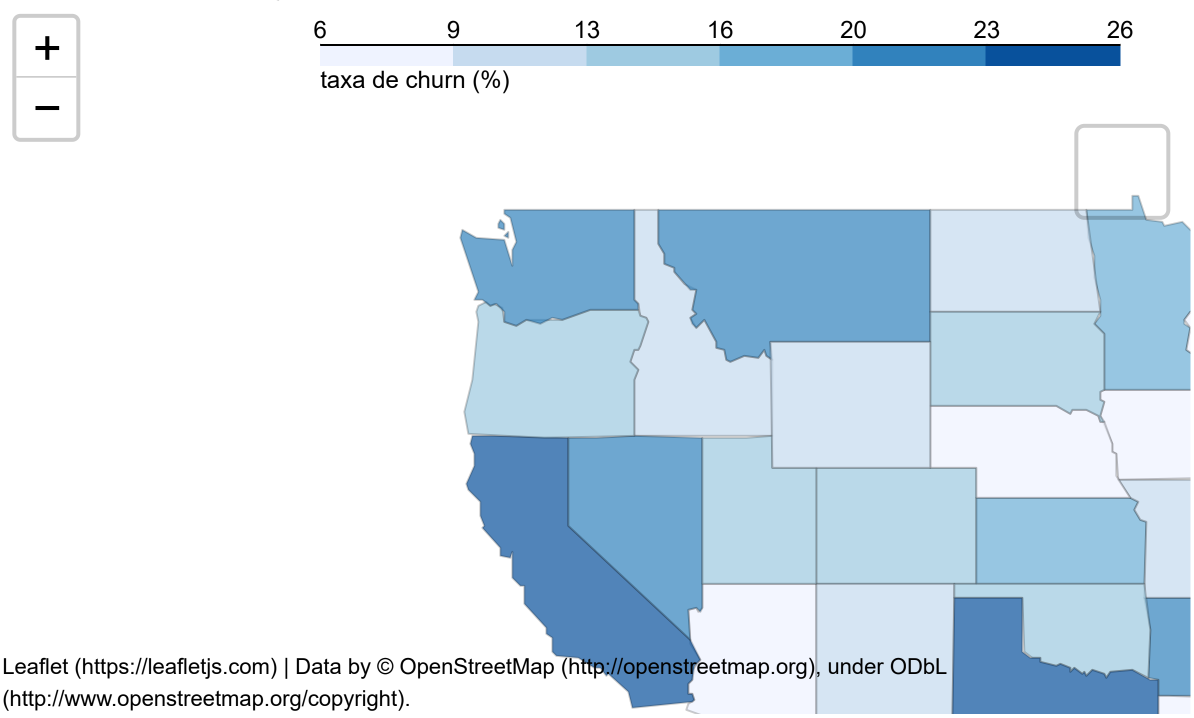


In [11]:
```
# Vamos análisar a distrbuição de churn por estado e colocar isso em um mapa p
ara melhor vizualização
import folium
import os
estados = os.path.join('us-states.json')
```

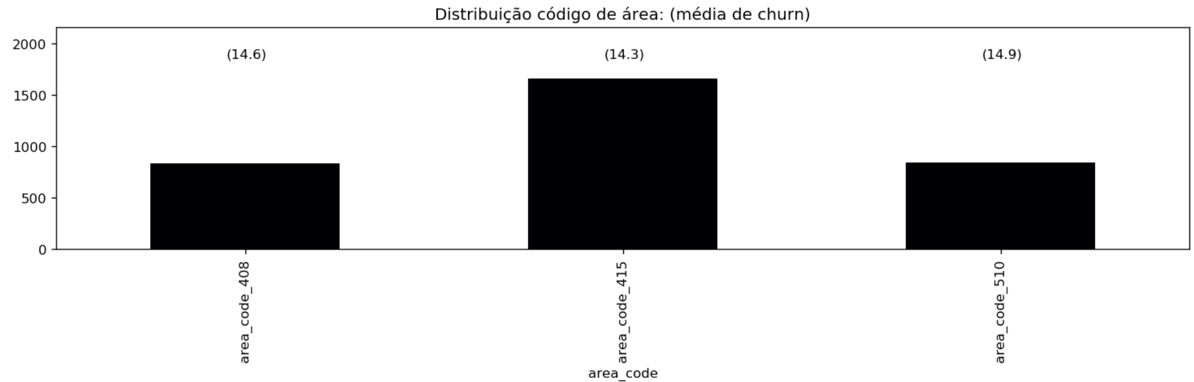
In [14]:

```python
# Criando os dados
# Califórnia, Texas, New York e Carolina do Sul possuem maior taxa de churn (~
26%)

dftemp = dftreino.groupby('state')['churn'].sum() / dftreino.groupby('state').
size() * 100

# Criando o mapa
m = folium.Map(location=[37, -102], zoom_start=4)
m.choropleth(
 geo_data=estados,
 name='choropleth',
 data=dftemp,
 columns=['state', 'account_length'],
 key_on='feature.id',
 fill_color='Blues',
 fill_opacity=0.7,
 line_opacity=0.2,
 legend_name='taxa de churn (%)'
)
folium.LayerControl().add_to(m)
m
```

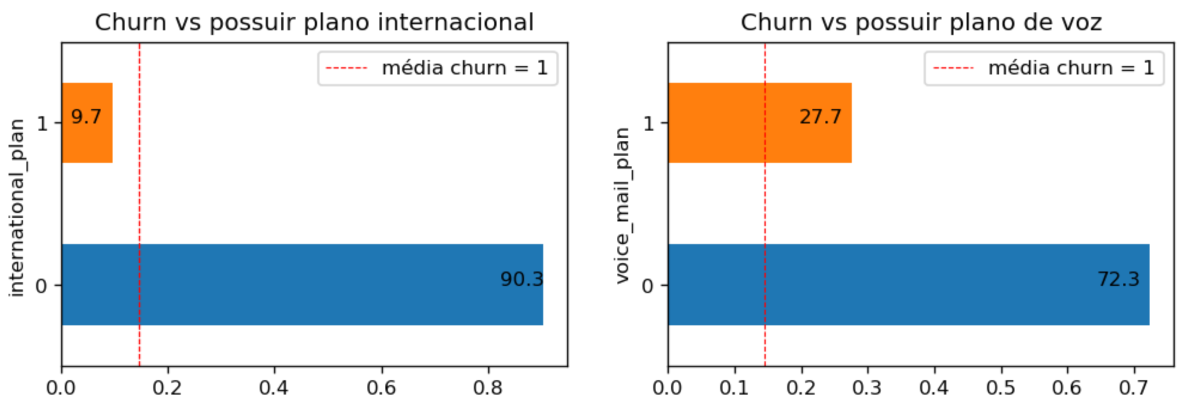
Out[14]:



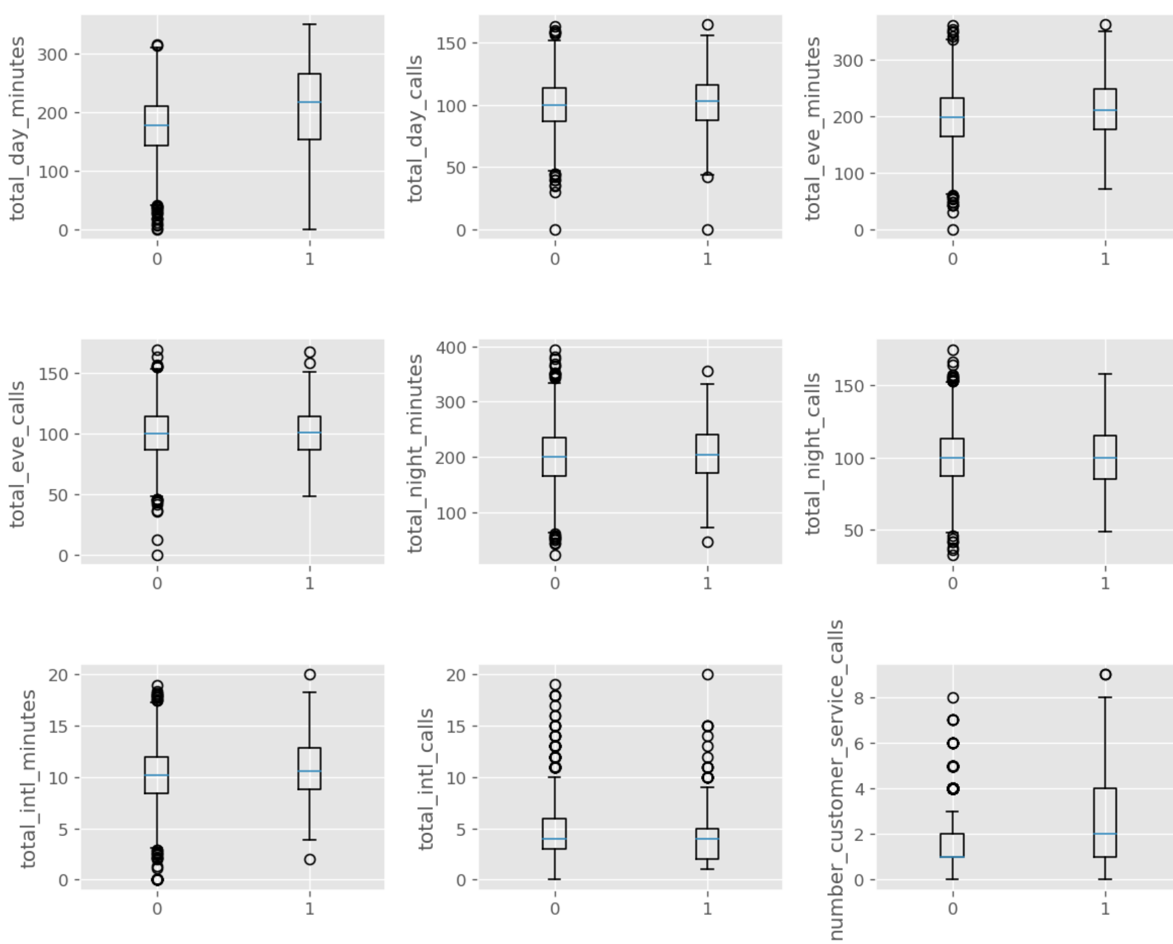
In [15]:
```
# Distribuição por código de area
# Não há grande diferença na média de churn dado o código de area.
# Isso significa que porovavelmente essa variável não é boa preditora, pois po
ssue pouca variabilidade

dftemp = dftreino.groupby('area_code').size()
dftemp2 = round(dftreino.groupby('area_code')['churn'].sum() / dftreino.groupb
y('area_code').size() * 100, 1)
fig = plt.figure(figsize=(15, 3), dpi = 120)
dftemp.plot(kind='bar', colormap='magma', ylim=[0,dftemp.max()+500] ,title='Di
stribuição código de área: (média de churn)')
for i in range(3):
    plt.text(i-0.05,dftemp.max()+200,'(' + str(dftemp2[i]) + ')')
```



In [17]:
```
# Análise de churn caso o cliente possua plano internacional ou plano de voz.
 Dadas as médias diferentes, é provável que essas
# variáveis sejam boas preditoras

fig, axes = plt.subplots(1,2, figsize=(10, 3), dpi = 120, squeeze = False)

dftemp = dftreino.groupby('international_plan').size() / dftreino['internation
al_plan'].count()
dftemp.plot('barh',ax=axes[0,0], title='Churn vs possuir plano internacional')
axes[0,0].vlines(churnDist[1]/100,-1,2, color='red', linestyle='--', linewidth
= 0.7)

dftemp2 = dftreino.groupby('voice_mail_plan').size() / dftreino['voice_mail_pl
an'].count()
dftemp2.plot('barh',ax=axes[0,1], title='Churn vs possuir plano de voz')

for i in range(2):
        axes[0,0].text(dftemp[i]-0.08,i,round(dftemp[i]*100,1))
        axes[0,1].text(dftemp2[i]-0.08,i,round(dftemp2[i]*100,1))
        axes[0,i].vlines(churnDist[1]/100,-1,2, color='red', linestyle='--', l
inewidth = 0.7, label='Média de churn')
        axes[0,i].legend(['média churn = 1'])
```



In [18]:
```
# Vamos análisar as variáveis numéricas separas por churn. É possível verifica
r que muitas variáveis não possuem grande
# diferenciação entre os grupos, indicando que podem não ser boas variáveis pr
editoras

plt.style.use('ggplot')
fig, axes = plt.subplots(3,3, figsize=(10, 8), dpi = 120, squeeze = False)

# Define as variáveis
variaveis = ['total_day_minutes','total_day_calls','total_eve_minutes',
             'total_eve_calls','total_night_minutes','total_night_calls',
             'total_intl_minutes','total_intl_calls','number_customer_service_
calls']

# Plota as variáveis
k = 0
for linha in axes:
    for ax in linha:
            listtemp = [dftreino.loc[dftreino['churn'] == 0,variaveis[k]], dft
reino.loc[dftreino['churn'] == 1,variaveis[k]]]
            ax.boxplot(listtemp)
            #ax.set_xlabel('churn'); # x label
            ax.set_ylabel(variaveis[k]); # y label
            ax.set_xticklabels([0,1])
            k += 1

plt.tight_layout()
```



In [19]:
```
# Por fim, vamos analisar as distribuições em pares. Podemos ver que muitas se
parações não são lineares, o que
# dificulta a modelagem preditiva
sns.set(style="ticks")
sns.pairplot(df, hue="churn")
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureW
arning: Using a non-tuple sequence for multidimensional indexing is deprecate
d; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be in
terpreted as an array index, `arr[np.array(seq)]`, which will result either i
n an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:4
88: RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdetool
s.py:34: RuntimeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
```

Out[19]: <seaborn.axisgrid.PairGrid at 0x22f8f337708>

Modelagem Preditiva

In [21]:
```
# Dataset que utilizaremos para a modelagem preditiva
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()

# Transforma as variáveis de texto em variáveis dummies
dados = pd.get_dummies(df)
colunas = dados.columns

dadosMPN = min_max_scaler.fit_transform(dados)
dadosMPN = pd.DataFrame(dadosMPN, columns = colunas)
#dadosMPN = preprocessing.scale(dadosMP)
```

In [22]:
```
# Separa os datasets em Treino e Teste
dadosTreino = dadosMPN[dadosMPN['dadosTreino'] == 1]
dadosTeste = dadosMPN[dadosMPN['dadosTreino'] == 0]
dadosTreino = dadosTreino.drop(['dadosTreino'], axis=1)
dadosTeste = dadosTeste.drop(['dadosTreino'], axis=1)

dadosTreinoX = dadosTreino.drop(['churn'], axis=1)
dadosTreinoY = dadosTreino['churn']

dadosTesteX = dadosTeste.drop(['churn'], axis=1)
dadosTesteY = dadosTeste['churn']
```

In [23]:
```
# Regressão Logística
import statsmodels.api as sm
from sklearn import linear_model
```

In [24]:
```
# Gera os coeficientes e valores p para podermos entender quais variáveis são
 mais importantes
modeloRL_sm=sm.Logit(dadosTreinoY,dadosTreinoX)
modeloRL_smResult=modeloRL_sm.fit()
modeloRL_smResult.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.310794
         Iterations 7
```

Out[24]:

Model:	Logit	Pseudo R-squared:	0.249
Dependent Variable:	churn	AIC:	2203.7528
Date:	2020-02-22 14:57	BIC:	2607.1202
No. Observations:	3333	Log-Likelihood:	-1035.9
Df Model:	65	LL-Null:	-1379.1
Df Residuals:	3267	LLR p-value:	1.4720e-104
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.9
account_length	0.2312	0.3472	0.6660	0.5054	-0.4493	0.9
international_plan	2.1884	0.1533	14.2743	0.0000	1.8879	2.4
voice_mail_plan	-2.1062	0.5930	-3.5515	0.0004	-3.2685	-0.9
number_vmail_messages	1.9507	0.9676	2.0160	0.0438	0.0542	3.8
total_day_minutes	4.6119	0.3900	11.8264	0.0000	3.8476	5.3
total_day_calls	0.6684	0.4716	1.4173	0.1564	-0.2559	1.5
total_eve_minutes	2.8258	0.4307	6.5603	0.0000	1.9816	3.6
total_eve_calls	0.1720	0.4912	0.3503	0.7261	-0.7906	1.1
total_night_minutes	1.5505	0.4546	3.4104	0.0006	0.6594	2.4
total_night_calls	0.0382	0.5120	0.0746	0.9405	-0.9653	1.0
total_intl_minutes	1.6733	0.4218	3.9669	0.0001	0.8466	2.5
total_intl_calls	-1.7907	0.5138	-3.4854	0.0005	-2.7977	-0.7
number_customer_service_calls	4.8333	0.3686	13.1134	0.0000	4.1109	5.5
state_AK	-1.2660	3287813.8752	-0.0000	1.0000	-6443998.0492	6443995.5
state_AL	-0.9391	3287813.8752	-0.0000	1.0000	-6443997.7223	6443995.8
state_AR	-0.3591	3287813.8752	-0.0000	1.0000	-6443997.1423	6443996.4
state_AZ	-1.1537	3287813.8752	-0.0000	1.0000	-6443997.9368	6443995.6
state_CA	0.5614	3287813.8752	0.0000	1.0000	-6443996.2218	6443997.3
state_CO	-0.6021	3287813.8752	-0.0000	1.0000	-6443997.3852	6443996.1
state_CT	-0.2478	3287813.8752	-0.0000	1.0000	-6443997.0310	6443996.5
state_DC	-0.5721	3287813.8752	-0.0000	1.0000	-6443997.3553	6443996.2
state_DE	-0.5057	3287813.8752	-0.0000	1.0000	-6443997.2889	6443996.2
state_FL	-0.6742	3287813.8752	-0.0000	1.0000	-6443997.4574	6443996.1
state_GA	-0.5958	3287813.8752	-0.0000	1.0000	-6443997.3790	6443996.1
state_HI	-1.4858	3287813.8752	-0.0000	1.0000	-6443998.2690	6443995.2
state_IA	-1.0360	3287813.8752	-0.0000	1.0000	-6443997.8192	6443995.7
state_ID	-0.3873	3287813.8752	-0.0000	1.0000	-6443997.1705	6443996.3
state_IL	-1.4805	3287813.8752	-0.0000	1.0000	-6443998.2637	6443995.3

state_IN	-0.8271	3287813.8752	-0.0000	1.0000	-6443997.6103	6443995.95
state_KS	-0.1954	3287813.8752	-0.0000	1.0000	-6443996.9786	6443996.58
state_KY	-0.4599	3287813.8752	-0.0000	1.0000	-6443997.2431	6443996.32
state_LA	-0.7020	3287813.8752	-0.0000	1.0000	-6443997.4852	6443996.08
state_MA	-0.1009	3287813.8752	-0.0000	1.0000	-6443996.8840	6443996.68
state_MD	-0.1258	3287813.8752	-0.0000	1.0000	-6443996.9089	6443996.65
state_ME	0.0808	3287813.8752	0.0000	1.0000	-6443996.7024	6443996.86
state_MI	0.1191	3287813.8752	0.0000	1.0000	-6443996.6641	6443996.90
state_MN	-0.0977	3287813.8752	-0.0000	1.0000	-6443996.8809	6443996.68
state_MO	-0.6615	3287813.8752	-0.0000	1.0000	-6443997.4447	6443996.12
state_MS	0.0913	3287813.8752	0.0000	1.0000	-6443996.6919	6443996.87
state_MT	0.6010	3287813.8752	0.0000	1.0000	-6443996.1822	6443997.38
state_NC	-0.6718	3287813.8752	-0.0000	1.0000	-6443997.4550	6443996.11
state_ND	-1.1232	3287813.8752	-0.0000	1.0000	-6443997.9063	6443995.66
state_NE	-0.9429	3287813.8752	-0.0000	1.0000	-6443997.7261	6443995.84
state_NH	-0.0910	3287813.8752	-0.0000	1.0000	-6443996.8742	6443996.69
state_NJ	0.3243	3287813.8752	0.0000	1.0000	-6443996.4589	6443997.10
state_NM	-0.7909	3287813.8752	-0.0000	1.0000	-6443997.5741	6443995.99
state_NV	-0.0117	3287813.8752	-0.0000	1.0000	-6443996.7949	6443996.77
state_NY	-0.0981	3287813.8752	-0.0000	1.0000	-6443996.8813	6443996.68
state_OH	-0.5850	3287813.8752	-0.0000	1.0000	-6443997.3681	6443996.19
state_OK	-0.3905	3287813.8752	-0.0000	1.0000	-6443997.1736	6443996.39
state_OR	-0.4827	3287813.8752	-0.0000	1.0000	-6443997.2659	6443996.30
state_PA	-0.1151	3287813.8752	-0.0000	1.0000	-6443996.8983	6443996.66
state_RI	-1.3744	3287813.8752	-0.0000	1.0000	-6443998.1576	6443995.40
state_SC	0.5094	3287813.8752	0.0000	1.0000	-6443996.2738	6443997.29
state_SD	-0.4317	3287813.8752	-0.0000	1.0000	-6443997.2149	6443996.35
state_TN	-0.9849	3287813.8752	-0.0000	1.0000	-6443997.7681	6443995.79
state_TX	0.3811	3287813.8752	0.0000	1.0000	-6443996.4021	6443997.16
state_UT	-0.2173	3287813.8752	-0.0000	1.0000	-6443997.0004	6443996.56
state_VA	-1.7047	3287813.8752	-0.0000	1.0000	-6443998.4879	6443995.07
state_VT	-1.1694	3287813.8752	-0.0000	1.0000	-6443997.9526	6443995.61
state_WA	0.1456	3287813.8752	0.0000	1.0000	-6443996.6376	6443996.92
state_WI	-0.9891	3287813.8752	-0.0000	1.0000	-6443997.7723	6443995.79
state_WV	-0.6808	3287813.8752	-0.0000	1.0000	-6443997.4640	6443996.10
state_WY	-0.9594	3287813.8752	-0.0000	1.0000	-6443997.7426	6443995.82
area_code_area_code_408	-8.4286	3287813.8752	-0.0000	1.0000	-6444005.2118	6443988.35

area_code_area_code_415	-8.5093	3287813.8752	-0.0000	1.0000	-6444005.2925	6443988.27
area_code_area_code_510	-8.5382	3287813.8752	-0.0000	1.0000	-6444005.3214	6443988.24

In [25]:

```
# Como vimos acima, alguns variáveis não seriam boas pretidoras (código de áre
a) e outras teriam bom potencial preditivo
# (plano internacional e plano de voz), o que se confirmou na tabela acima dad
o os valores de p

# Filtrando apenas as variáveis com p value < 5%
valoresP = modeloRL_smResult.pvalues
valoresPFiltered = valoresP[valoresP > 0.05]

dadosTreinoXFinal = dadosTreinoX.drop(columns=valoresPFiltered.index, axis=1)
dadosTreinoYFinal = dadosTreinoY.drop(columns=valoresPFiltered.index, axis=1)

dadosTesteXFinal = dadosTreinoX.drop(columns=valoresPFiltered.index, axis=1)
dadosTesteYFinal = dadosTreinoY.drop(columns=valoresPFiltered.index, axis=1)
```

In [26]:

```
# Importa pacotes do sklearn para modelagem preditiva
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

In [27]:

```
# Separando os dados
X_train, X_valid, y_train, y_valid = train_test_split(dadosTreinoXFinal, dados
TreinoYFinal, test_size=0.3)
```

In [28]:

```
# Contruindo o modelo
modeloRL = linear_model.LogisticRegression()
modeloRL.fit(X_train,y_train)

# Acurácia
acuracia = cross_val_score(modeloRL, X_train, y_train, cv = 5, scoring = 'roc_
auc', n_jobs = -1)
print('Acurácia média: %0.2f' % np.mean(acuracia))

predicoes = modeloRL.predict(X_valid)
# Classification report
print(classification_report(y_valid, predicoes))
# Confusion Matrix
print(confusion_matrix(y_valid, predicoes))
```

```
Acurácia média: 0.82
             precision    recall  f1-score   support

        0.0       0.86      0.98      0.92       846
        1.0       0.52      0.14      0.22       154

avg / total       0.81      0.85      0.81      1000

[[826  20]
 [132  22]]
```

In [29]:
```
# Como os dados estão desbalanceados, o modelo possui muitos falsos negativos.
Vamos tentar balancear os dados.

# Construindo o modelo
modeloRLB = linear_model.LogisticRegression(class_weight='balanced')
modeloRLB.fit(X_train,y_train)

# Acurácia
acuracia = cross_val_score(modeloRLB, X_train, y_train, cv = 5, scoring = 'roc
_auc', n_jobs = -1)
print('Acurácia média: %0.2f' % np.mean(acuracia))

predicoes = modeloRLB.predict(X_valid)
# Classification report
print(classification_report(y_valid, predicoes))
# Confusion Matrix
print(confusion_matrix(y_valid, predicoes))
```

```
Acurácia média: 0.82
             precision    recall  f1-score   support

        0.0       0.94      0.73      0.82       846
        1.0       0.34      0.74      0.46       154

avg / total       0.85      0.73      0.77      1000

[[621 225]
 [ 40 114]]
```

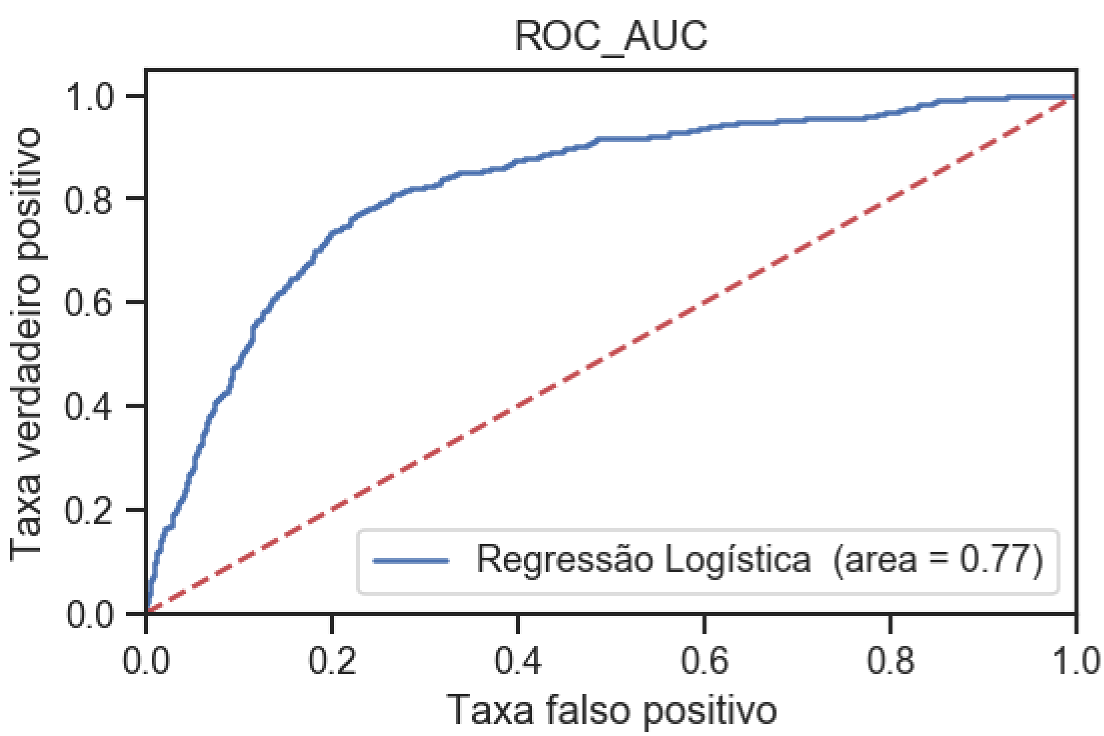
O modelo balanceado consegui identificar de forma correta 5x mais que o modelo não balanceado. Apesar do número de falsos positivos ter aumentado, o mais importante é maximizar a identificação dos clientes churn.

In [30]:
```
# Resultados com os dados de teste finais

predicoes = modeloRLB.predict(dadosTesteXFinal)
# Classification report
print(classification_report(dadosTesteYFinal, predicoes))
# Confusion Matrix
print(confusion_matrix(dadosTesteYFinal, predicoes))
```

```
             precision    recall  f1-score   support

        0.0       0.95      0.77      0.85      2850
        1.0       0.36      0.77      0.49       483

avg / total       0.87      0.77      0.80      3333

[[2200  650]
 [ 111  372]]
```

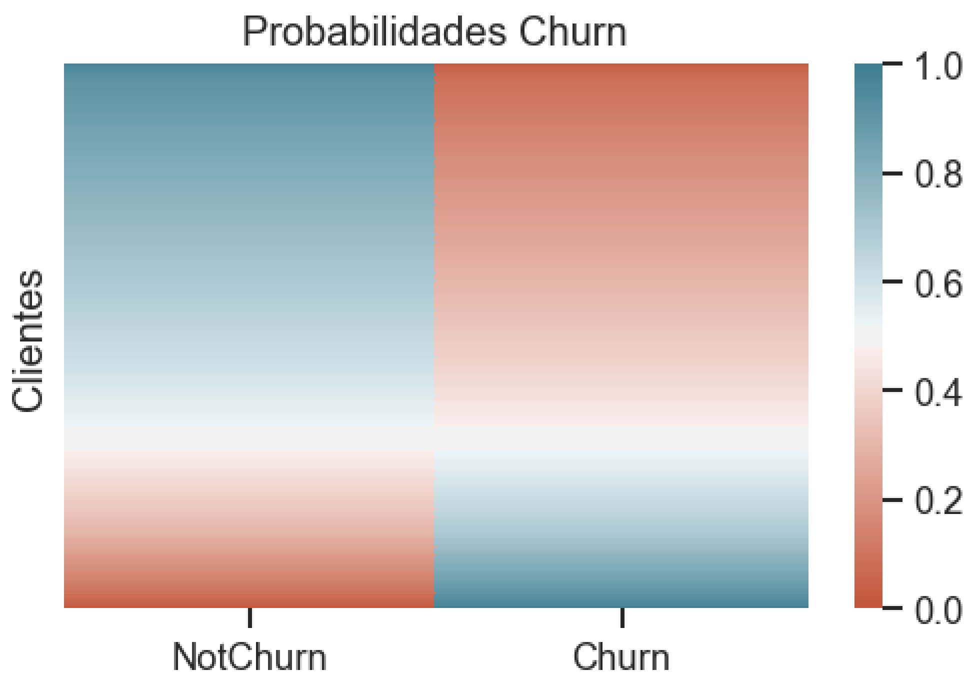
In [31]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
```

In [33]:
```
# Montando a roc_auc
fig = plt.figure(figsize=(5, 3), dpi = 120)
RL_roc_auc = roc_auc_score(dadosTesteYFinal, modeloRLB.predict(dadosTesteXFina
l))
fpr, tpr, thresholds = roc_curve(dadosTesteYFinal, modeloRLB.predict_proba(dad
osTesteXFinal)[:,1])
plt.plot(fpr, tpr, label='Regressão Logística  (area = %0.2f)' % RL_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Taxa falso positivo')
plt.ylabel('Taxa verdadeiro positivo')
plt.title('ROC_AUC')
plt.legend(loc="lower right")
plt.show()
```



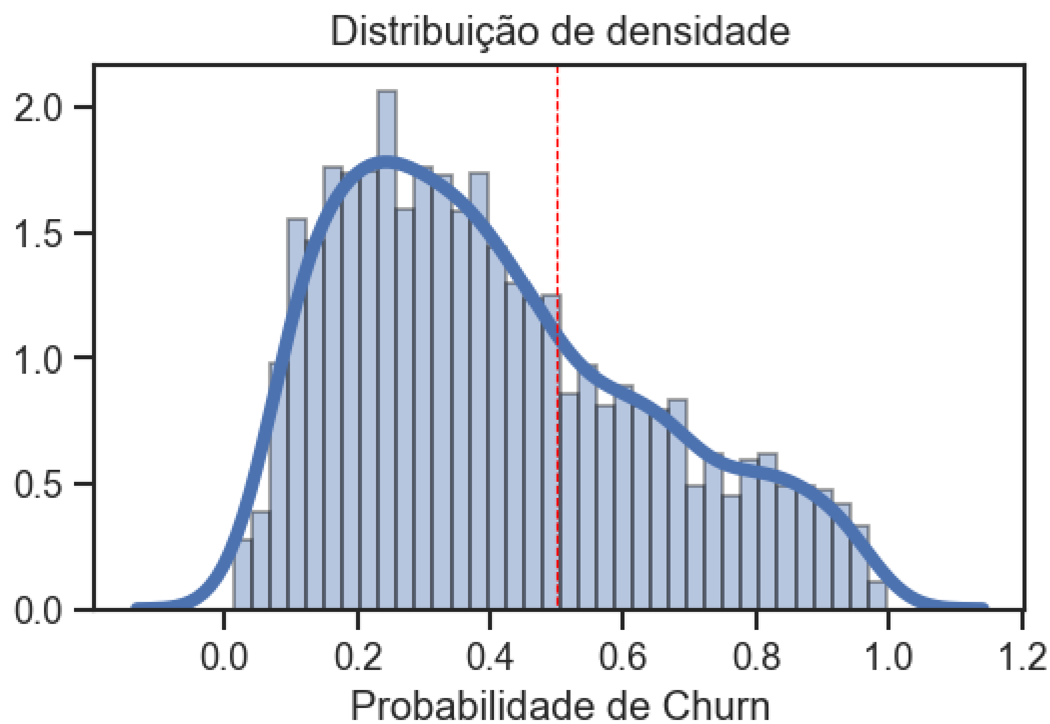
In [43]:
```
# Vamos analisar as probabilidades dos clientes serem classificados como Churn
ou não
predicoesProb = modeloRLB.predict_proba(dadosTesteXFinal)
predicoesProb = pd.DataFrame(predicoesProb,columns=['NotChurn','Churn'])
predicoesProb = predicoesProb.sort_values(by='Churn')
fig = plt.figure(figsize=(5, 3), dpi = 120)
ax = sns.heatmap(
    predicoesProb, annot=False,
    vmin=0, vmax=1, center=0.5,
    cmap=sns.diverging_palette(20, 220, n=200))
ax.set(title='Probabilidades Churn')
ax.set_yticks([])
ax.set_ylabel('Clientes')
```

Out[43]:
```
Text(34.5,0.5,'Clientes')
```



In [44]:
```
# Distribuição de probabilidades
fig = plt.figure(figsize=(5, 3), dpi = 120)
predicoesProb = modeloRLB.predict_proba(dadosTesteXFinal)
predicoesProb = pd.DataFrame(predicoesProb,columns=['NotChurn','Churn'])
ax = sns.distplot(predicoesProb['Churn'], hist=True, kde=True,
             bins=int(180/5),
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
plt.axvline(0.5, 2,0,color='red', linestyle='--', linewidth = 0.7)
plt.title('Distribuição de densidade')
ax.set_xlabel('Probabilidade de Churn')
```

Out[44]:
```
Text(0.5,0,'Probabilidade de Churn')
```



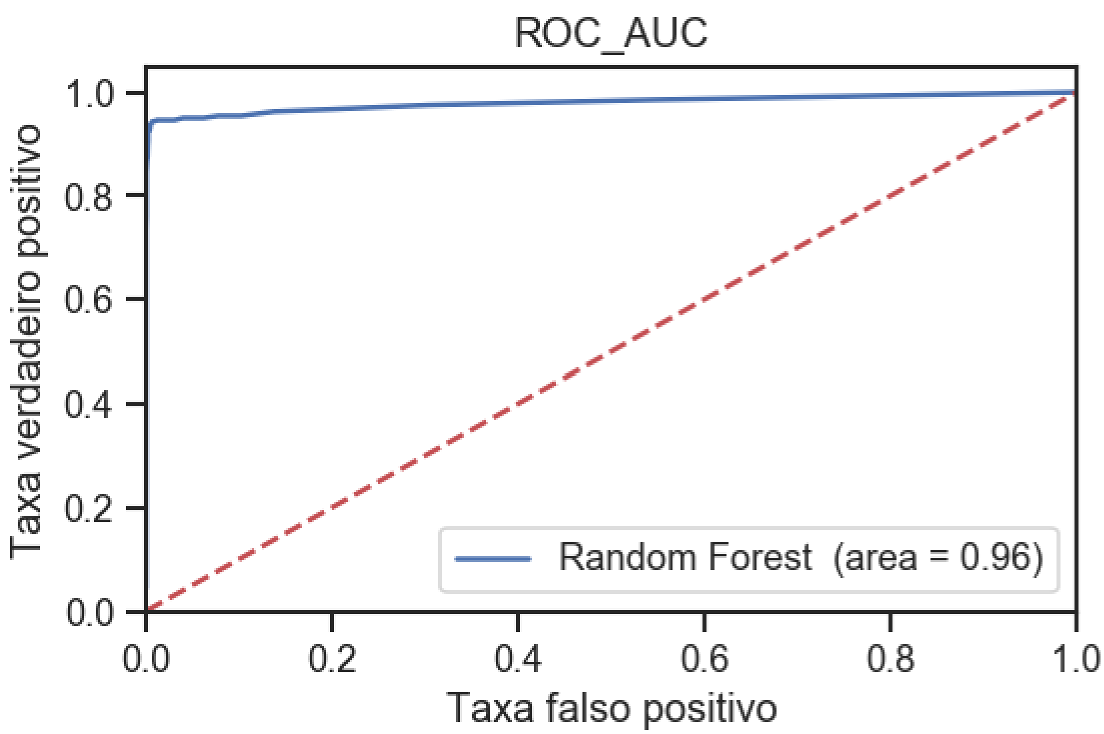
Bônus

Como vimos acima, muitas variáveis não possuem separação linear. Com isso, um classificador não linear poderia obter melhores resultados que um modelo de regressão. Abaixo, vamos construir um modelo simples de árvore de decisão para verificarmos a diferença de acurácia entre os modelos

In [47]:
```
from sklearn.ensemble import RandomForestClassifier
modeloRF = RandomForestClassifier(n_estimators=50)
modeloRF.fit(X_train, y_train)

# Acurácia
acuracia = cross_val_score(modeloRF, X_train, y_train, cv = 5, scoring = 'roc_auc', n_jobs = -1)
print('Acurácia média: %0.2f' % np.mean(acuracia))

predicoes = modeloRF.predict(dadosTesteXFinal)
# Classification report
print(classification_report(dadosTesteYFinal, predicoes))
# Confusion Matrix
print(confusion_matrix(dadosTesteYFinal, predicoes))
```

```
Acurácia média: 0.91
             precision    recall  f1-score   support

        0.0       0.99      1.00      0.99      2850
        1.0       0.98      0.92      0.95       483

avg / total       0.99      0.99      0.99      3333

[[2843    7]
 [  37  446]]
```

In [48]:
```
# Montando a roc_auc
fig = plt.figure(figsize=(5, 3), dpi = 120)
RL_roc_auc = roc_auc_score(dadosTesteYFinal, modeloRF.predict(dadosTesteXFinal
))
fpr, tpr, thresholds = roc_curve(dadosTesteYFinal, modeloRF.predict_proba(dado
sTesteXFinal)[:,1])
plt.plot(fpr, tpr, label='Random Forest  (area = %0.2f)' % RL_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Taxa falso positivo')
plt.ylabel('Taxa verdadeiro positivo')
plt.title('ROC_AUC')
plt.legend(loc="lower right")
plt.show()
```



Como podemos verificar, o modelo de árvore de decisão conseguiu uma acurácia muito maior que o modelo de regressão logística.

In [ ]: