Pyspark_version_Predicting_Customer_Churn_in_Telecommunication_

April 3, 2020

1 Predicting Customer Churn in Telecommunication Operators

Customer turnover refers to a decision made by the customer on the term of business relationship. Customer loyalty and customer turnover always add up to 100%. If a company has a 60% loyalty rate, then customer loss taxes are 40%. According to the 80/20 customer profitability rule, 20% of customers are generating 80% of revenue. Therefore, it is very important to predict the users who are likely to abandon the business relationship and the factors that affect how the customer's decisions. In this project, I predicted Customer Churn at a Telecommunications Operator using pyspark and frameworks (Pandas, Numpy, scipy and Scikit-Learn).

[199]:
```
# Importing libraries and frameworks
from pyspark.sql.functions import col, sum
from pyspark.sql import Row
import pyspark.sql.functions as f
from pyspark.sql.window import Window
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import PCA
from pyspark.ml.linalg import Vectors

from scipy.stats import skew, kurtosis

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
import operator as op
import folium
from IPython.display import display
from IPython.display import Image
from sklearn.metrics import classification_report, confusion_matrix

from pyspark.ml.feature import StandardScaler
from pyspark.ml.feature import MinMaxScaler
```

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator,␣
 ↪BinaryClassificationEvaluator
from pyspark.ml import Pipeline
from pyspark.sql.functions import monotonically_increasing_id, row_number
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

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

[200]:
```
# Creating Spark Session
spSession = SparkSession.builder.master("local").appName("local-SparkMLLib").
 ↪getOrCreate()
```

1.1 Importing dataset

[201]:
```
df_train = spark.read.csv("data/projeto4_telecom_treino.csv", header=True)
df_test = spark.read.csv("data/projeto4_telecom_teste.csv", header=True)
```

1.2 Exploratory Analysis

[202]:
```
# Checking train data
display(df_train.head(5))
```

```
[Row(_c0='1', state='KS', account_length='128', area_code='area_code_415', international_plan=
 Row(_c0='2', state='OH', account_length='107', area_code='area_code_415', international_plan=
 Row(_c0='3', state='NJ', account_length='137', area_code='area_code_415', international_plan=
 Row(_c0='4', state='OH', account_length='84', area_code='area_code_408', international_plan='
 Row(_c0='5', state='OK', account_length='75', area_code='area_code_415', international_plan='
```

[203]:
```
# Ckecking size of datasets
print((df_train.count(), len(df_train.columns)))
print((df_test.count(), len(df_train.columns)))
```

```
(3333, 21)
(1667, 21)
```

[204]:
```
# Checking for missing values on train and test datasets
df_train.select(*(sum(col(c).isNull().cast("int")).alias(c) for c in df_train.
 ↪columns)).show()
df_test.select(*(sum(col(c).isNull().cast("int")).alias(c) for c in df_train.
 ↪columns)).show()
```

```
+---+-----+--------------+---------+------------------+---------------+---------
------------+-----------------+---------------+----------------+----------------
```

```
-+---------------+----------------+-------------------+-----------------+-------
-----------+------------------+----------------+-----------------+--------------
---------------+-----+
|_c0|state|account_length|area_code|international_plan|voice_mail_plan|number_vm
ail_messages|total_day_minutes|total_day_calls|total_day_charge|total_eve_minute
s|total_eve_calls|total_eve_charge|total_night_minutes|total_night_calls|total_n
ight_charge|total_intl_minutes|total_intl_calls|total_intl_charge|number_custome
r_service_calls|churn|
+---+-----+--------------+---------+------------------+---------------+---------
------------+-----------------+---------------+----------------+----------------
-+---------------+----------------+-------------------+-----------------+-------
-----------+------------------+----------------+-----------------+--------------
---------------+-----+
|  0|    0|             0|        0|                 0|              0|
0|                0|              0|               0|                0|
0|               0|                  0|                0|                 0|
0|               0|                0|                            0|    0|
+---+-----+--------------+---------+------------------+---------------+---------
------------+-----------------+---------------+----------------+----------------
-+---------------+----------------+-------------------+-----------------+-------
-----------+------------------+----------------+-----------------+--------------
---------------+-----+

+---+-----+--------------+---------+------------------+---------------+---------
------------+-----------------+---------------+----------------+----------------
-+---------------+----------------+-------------------+-----------------+-------
-----------+------------------+----------------+-----------------+--------------
---------------+-----+
|_c0|state|account_length|area_code|international_plan|voice_mail_plan|number_vm
ail_messages|total_day_minutes|total_day_calls|total_day_charge|total_eve_minute
s|total_eve_calls|total_eve_charge|total_night_minutes|total_night_calls|total_n
ight_charge|total_intl_minutes|total_intl_calls|total_intl_charge|number_custome
r_service_calls|churn|
+---+-----+--------------+---------+------------------+---------------+---------
------------+-----------------+---------------+----------------+----------------
-+---------------+----------------+-------------------+-----------------+-------
-----------+------------------+----------------+-----------------+--------------
---------------+-----+
|  0|    0|             0|        0|                 0|              0|
0|                0|              0|               0|                0|
0|               0|                  0|                0|                 0|
0|               0|                0|                            0|    0|
+---+-----+--------------+---------+------------------+---------------+---------
------------+-----------------+---------------+----------------+----------------
-+---------------+----------------+-------------------+-----------------+-------
-----------+------------------+----------------+-----------------+--------------
---------------+-----+
```

[205]:
```
df_train.columns
```

[205]:
```
['_c0',
 'state',
 'account_length',
 'area_code',
 'international_plan',
 'voice_mail_plan',
 'number_vmail_messages',
 'total_day_minutes',
 'total_day_calls',
 'total_day_charge',
 'total_eve_minutes',
 'total_eve_calls',
 'total_eve_charge',
 'total_night_minutes',
 'total_night_calls',
 'total_night_charge',
 'total_intl_minutes',
 'total_intl_calls',
 'total_intl_charge',
 'number_customer_service_calls',
 'churn']
```

[206]:
```
# Compute numerical data summary statistics
df_train_num = df_train.select(['account_length', 'number_vmail_messages',␣
 ↪'total_day_minutes',
                        'total_day_calls', 'total_day_charge',␣
 ↪'total_eve_minutes',
                        'total_eve_calls', 'total_eve_charge',␣
 ↪'total_night_minutes',
                        'total_night_calls', 'total_night_charge',␣
 ↪'total_intl_minutes',
                        'total_intl_calls', 'total_intl_charge',␣
 ↪'number_customer_service_calls'])

# converting columns to numeric and calculating median
for col_name in df_train_num.columns:
    df_train_num = df_train_num.withColumn(col_name, df_train_num[col_name].
 ↪cast('float'))

summary_df = df_train_num.describe().toPandas()

s = ['account_length', 'number_vmail_messages', 'total_day_minutes',
                        'total_day_calls', 'total_day_charge',␣
 ↪'total_eve_minutes',
```

```
                         'total_eve_calls', 'total_eve_charge',␣
 ↪'total_night_minutes',
                         'total_night_calls', 'total_night_charge',␣
 ↪'total_intl_minutes',
                         'total_intl_calls', 'total_intl_charge',␣
 ↪'number_customer_service_calls']


df = pd.DataFrame([pd.to_numeric(summary_df[col]) for col in s]).T.round(2)
idx = 0
df.insert(loc=idx, column='summary', value=summary_df.summary.values)
```

[207]:

```
# Calculating percentiles
median = [df_train_num.approxQuantile(col, [0.5], 0.0)[0] for col in␣
 ↪df_train_num.columns]
first_quartile = [df_train_num.approxQuantile(col, [0.25], 0.0)[0] for col in␣
 ↪df_train_num.columns]
third_quartile = [df_train_num.approxQuantile(col, [0.75], 0.0)[0] for col in␣
 ↪df_train_num.columns]
```

[208]:

```
# Getting mean absolute deviation
def mad(col, axis=None):
    data = [int(row[col]) for row in df_train_num.select(col).collect()]
    return np.mean(np.absolute(data - np.mean(data, axis)), axis)

mad_lst = [mad(col) for col in df_train_num.columns]

# Getting skewness
skew_list = [skew([int(row[col]) for row in df_train_num.select(col).collect()])
            for col in df_train_num.columns]

# Getting kurtosis
kurt = [kurtosis([int(row[col]) for row in df_train_num.select(col).collect()])
            for col in df_train_num.columns]
```

[209]:

```
# summary_df.append(lst)
df.loc['5'] = ['25%'] + first_quartile
df.loc['6'] = ['50%'] + median
df.loc['7'] = ['75%'] + third_quartile
df.loc['8'] = ['mad'] + mad_lst
df.loc['9'] = ['skew'] + skew_list
df.loc['10'] = ['kurt'] + kurt
```

[210]:

```
display(df.set_index('summary').round(2))
```

```
         account_length  number_vmail_messages  total_day_minutes  \
summary
count           3333.00                3333.00            3333.00
mean             101.06                   8.10             179.78
```

```
stddev            39.82                  13.69                 54.47
min                1.00                   0.00                  0.00
max              243.00                  51.00                350.80
25%               74.00                   0.00                143.70
50%              101.00                   0.00                179.40
75%              127.00                  20.00                216.40
mad               31.82                  11.72                 43.52
skew               0.10                   1.26                 -0.03
kurt              -0.11                  -0.05                 -0.02

         total_day_calls  total_day_charge  total_eve_minutes  \
summary
count            3333.00           3333.00            3333.00
mean              100.44             30.56             200.98
stddev             20.07              9.26              50.71
min                 0.00              0.00               0.00
max               165.00             59.64             363.70
25%                87.00             24.43             166.60
50%               101.00             30.50             201.40
75%               114.00             36.79             235.30
mad                15.94              7.40              40.48
skew               -0.11             -0.03              -0.02
kurt                0.24             -0.03               0.02

         total_eve_calls  total_eve_charge  total_night_minutes  \
summary
count            3333.00           3333.00              3333.00
mean              100.11             17.08               200.87
stddev             19.92              4.31                50.57
min                 0.00              0.00                23.20
max               170.00             30.91               395.00
25%                87.00             14.16               167.00
50%               100.00             17.12               201.20
75%               114.00             20.00               235.30
mad                15.86              3.45                40.41
skew               -0.06             -0.03                 0.01
kurt                0.20              0.01                 0.08

         total_night_calls  total_night_charge  total_intl_minutes  \
summary
count              3333.00             3333.00             3333.00
mean                100.11                9.04               10.24
stddev               19.57                2.28                2.79
min                  33.00                1.04                0.00
max                 175.00               17.77               20.00
25%                  87.00                7.52                8.50
50%                 100.00                9.05               10.30
75%                 113.00               10.59               12.10
```

```
mad                   15.69               1.84                2.20
skew                   0.03               0.00               -0.21
kurt                  -0.07               0.05                0.45

         total_intl_calls  total_intl_charge  number_customer_service_calls
summary
count             3333.00            3333.00                        3333.00
mean                 4.48               2.76                           1.56
stddev               2.46               0.75                           1.32
min                  0.00               0.00                           0.00
max                 20.00               5.40                           9.00
25%                  3.00               2.30                           1.00
50%                  4.00               2.78                           1.00
75%                  6.00               3.27                           2.00
mad                  1.88               0.66                           1.05
skew                 1.32              -0.13                           1.09
kurt                 3.08               0.02                           1.73
```

[211]:

```
# Compute categorical data summary statistics
df_train_cat = df_train.select(['state', 'area_code', 'international_plan',
 ↪'voice_mail_plan', 'churn'])

count = [df_train.count()] * len(df_train_cat.columns)
unique = [df_train_cat.select(col).distinct().count() for col in df_train_cat.
 ↪columns]

top = []
freq = []

for col in df_train_cat.columns:
    frequency = df_train_cat.groupBy(col).count().orderBy('count',
 ↪ascending=False).head(1)[0]
    top.append(frequency[col])
    freq.append(frequency['count'])

desc = pd.DataFrame({}, columns = df_train_cat.columns, index=['count',
 ↪'unique', 'top', 'freq'])

desc.loc['count'] = count
desc.loc['unique'] = unique
desc.loc['top'] = top
desc.loc['freq'] = freq

display(desc)
```

```
       state      area_code international_plan voice_mail_plan churn
```

```
count    3333           3333              3333          3333  3333
unique     51              3                 2             2     2
top        WV   area_code_415               no            no    no
freq      106           1655              3010          2411  2850
```

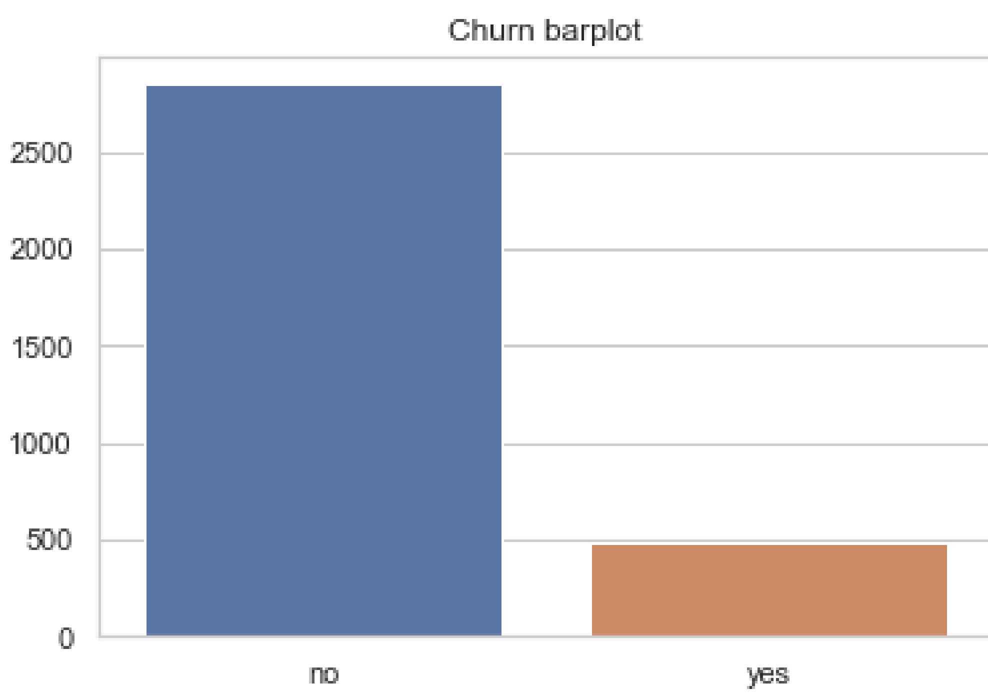
1.2.1 Univariate analysis

Checking churn variable distribution and proportion

[212]:
```
# churn values and proportion
df = df_train_cat.groupBy('churn').count().orderBy('count', ascending=False)
df = df.withColumn('percent', f.col('count')/f.sum('count').over(Window.
 ↪partitionBy()))
df = df.orderBy('percent', ascending=False)
df = df.toPandas().set_index('churn')
display(df)

# churn variable barplot
sns.set(style="whitegrid")
sns.barplot(x=['no','yes'], y=df['count'].values).set_title('Churn barplot')
```

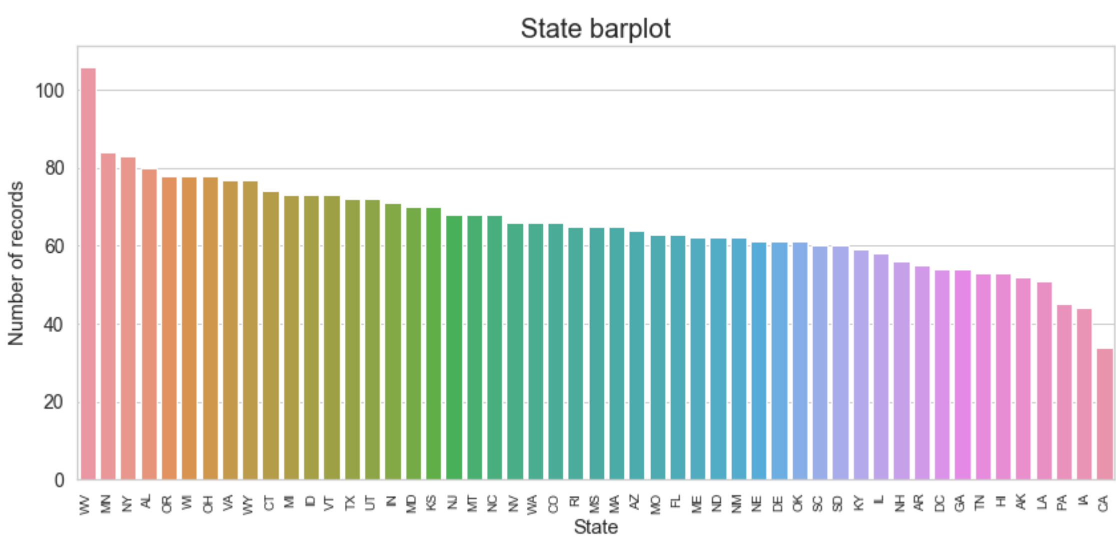
```
       count   percent
churn
no      2850  0.855086
yes      483  0.144914
```

[212]:
```
Text(0.5, 1.0, 'Churn barplot')
```



As noticed on table and graph above, churn feature is unbalanced. About 14% of customers stopped using the telecom service and 85% still using it. #### Categorical variables ##### State

[213]:
```
# State values and proportion
df = df_train_cat.groupBy('state').count().orderBy('count', ascending=False)
df = df.toPandas()

# State variable barplot
plt.figure(figsize=(14,6))

sns.barplot(x=df['state'],
            y=df['count'].values)

plt.title('State barplot', size=20)
plt.xlabel('State', size=15)
plt.ylabel('Number of records', size=15)
plt.yticks(fontsize=14)
plt.xticks(fontsize=10, rotation=90)
plt.show()
```

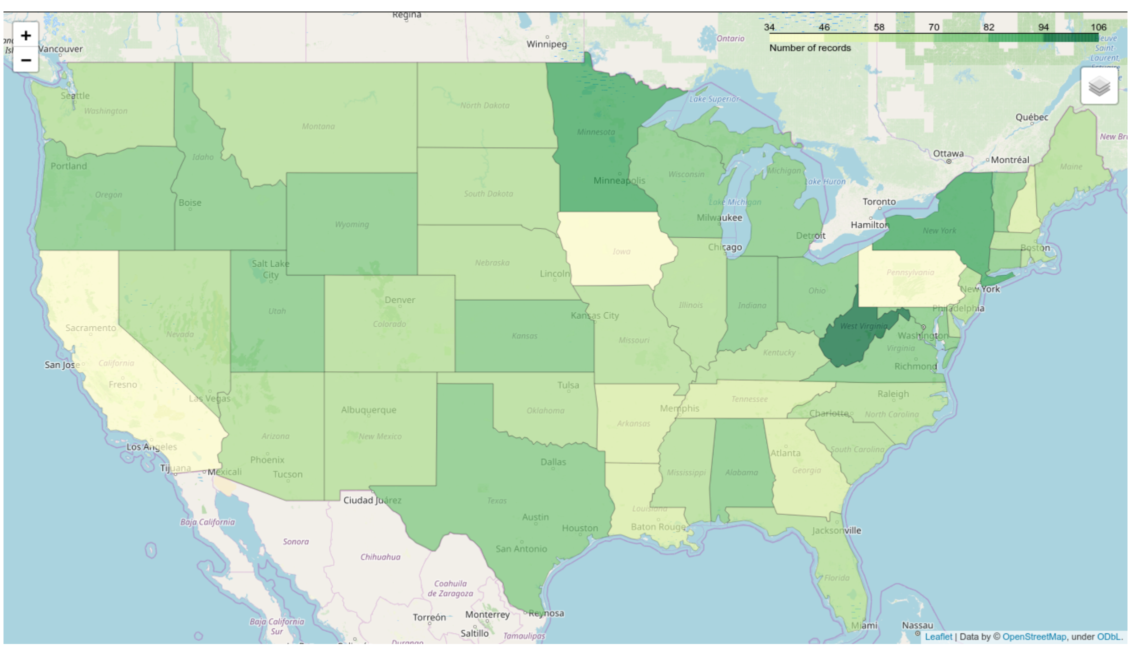


The state with the highest frequency is the West Virginia and the lowest frequency is the California. Other states with a large number of records are New York and Minnesota. ###### Number of records by State shown on the map below:

[142]:
```
# Map graph
# Load the shape of the zone (US states)
state_geo = os.path.join('', 'us-states.json')

# state data
```

```python
state_data = df_train_cat.groupBy('state').count().orderBy('count', ascending=False).toPandas()

# Initialize the map:
m = folium.Map(location=[37, -102], zoom_start=5)

# Add the color for the chloropleth:
m.choropleth(
 geo_data=state_geo,
 name='Number of records by state',
 data=state_data,
 columns=['state', 'count'],
 key_on='feature.id',
 fill_color='YlGn',
 fill_opacity=0.7,
 line_opacity=0.2,
 legend_name='Number of records'
)
folium.LayerControl().add_to(m)

# Save to html
m.save('#registers_by_map.html')

display(m)

# Loading map image
# Image(filename='records_by_state.png')
```

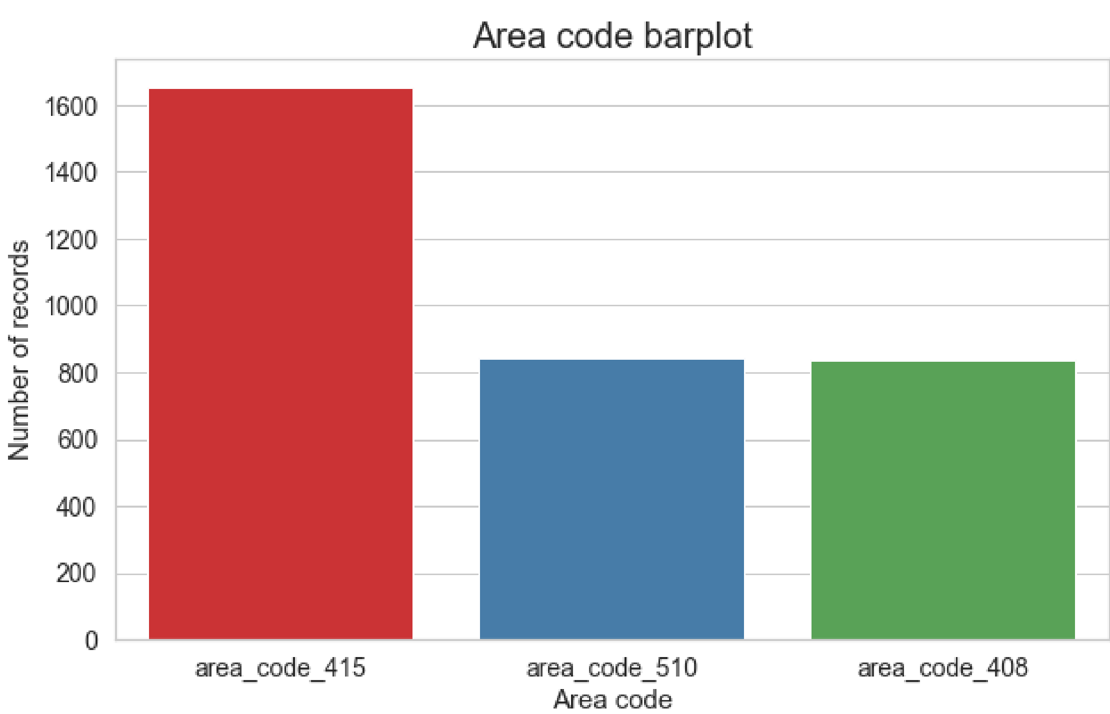
[142]:



Area code

[214]:
```
# area_code values and proportion
df = df_train_cat.groupBy('area_code').count().orderBy('count', ascending=False)
df = df.toPandas()
display(df)

# area_code variable barplot
plt.figure(figsize=(10,6))
sns.set(style="whitegrid")
sns.barplot(x=df['area_code'],
            y=df['count'],
           palette='Set1')

plt.title('Area code barplot', size=20)
plt.xlabel('Area code', size=15)
plt.ylabel('Number of records', size=15)
plt.yticks(fontsize=14)
plt.xticks(fontsize=14)
plt.show()
```

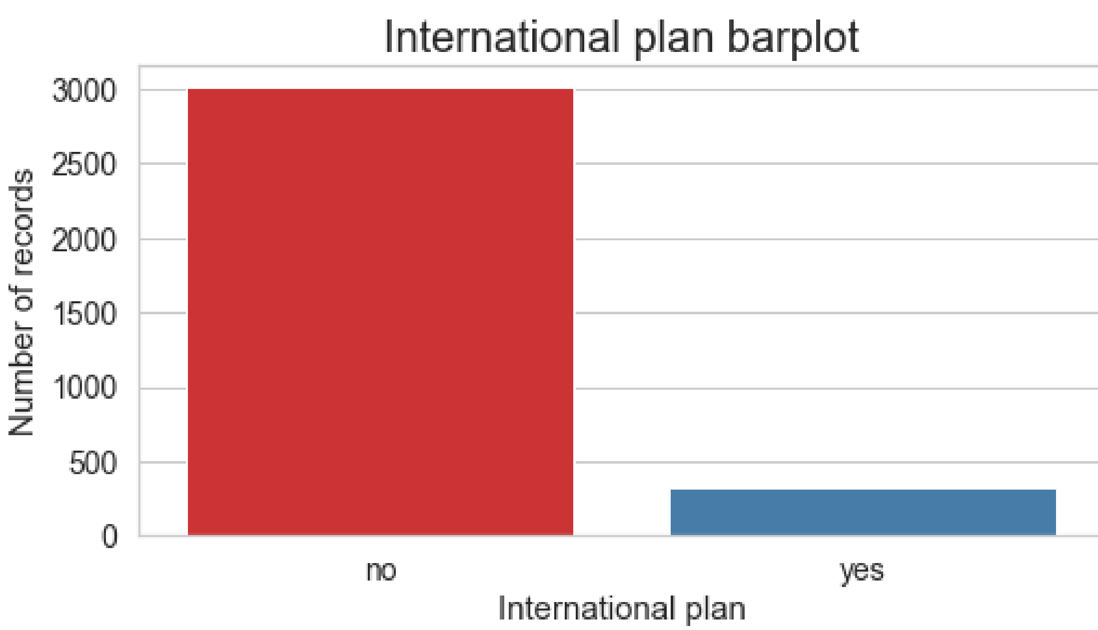
	area_code	count
0	area_code_415	1655
1	area_code_510	840
2	area_code_408	838



Code area 415 has the largest number of records. #### International plan

[215]:
```
# international_plan values and proportion
df = df_train_cat.groupBy('international_plan').count().orderBy('count',␣
↪ascending=False)
df = df.toPandas()
display(df)

# international_plan variable barplot
plt.figure(figsize=(8,4))
sns.set(style="whitegrid")
sns.barplot(x=df['international_plan'],
            y=df['count'],
           palette='Set1')

plt.title('International plan barplot', size=20)
plt.xlabel('International plan', size=15)
plt.ylabel('Number of records', size=15)
plt.yticks(fontsize=14)
plt.xticks(fontsize=14)
plt.show()
```

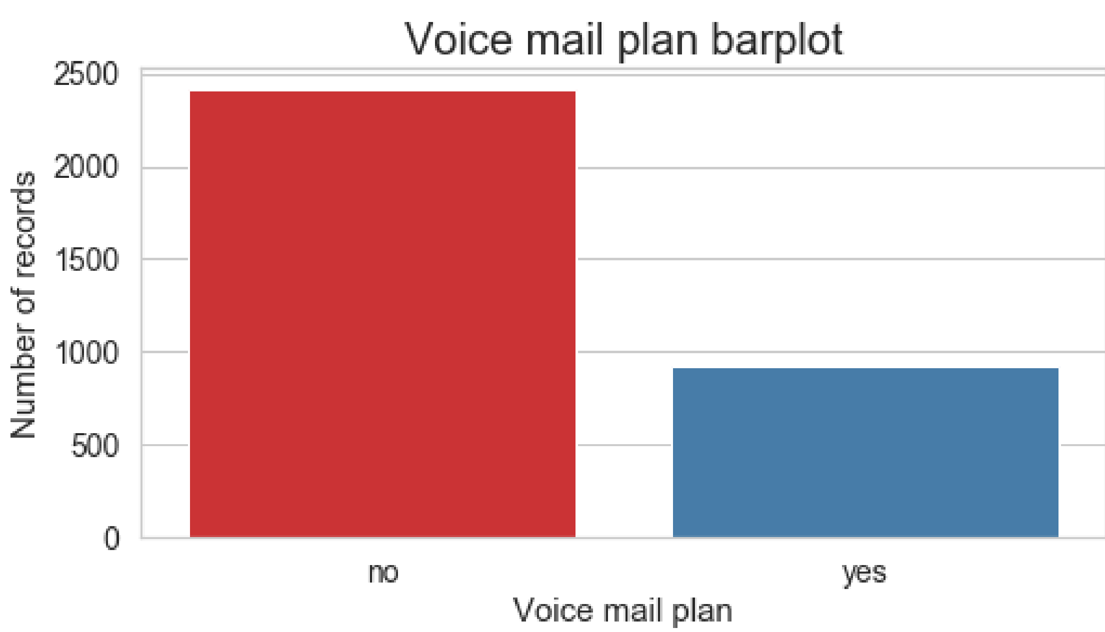
	international_plan	count
0	no	3010
1	yes	323



Most customers do not have international plan. #### Voice mail plan

[216]:
```
# voice_mail_plan values and proportion
df = df_train_cat.groupBy('voice_mail_plan').count().orderBy('count',␣
 ↪ascending=False)
df = df.toPandas()
display(df)

# voice_mail_plan variable barplot
plt.figure(figsize=(8,4))
sns.set(style="whitegrid")
sns.barplot(x=df['voice_mail_plan'],
            y=df['count'],
           palette='Set1')

plt.title('Voice mail plan barplot', size=20)
plt.xlabel('Voice mail plan', size=15)
plt.ylabel('Number of records', size=15)
plt.yticks(fontsize=14)
plt.xticks(fontsize=14)
plt.show()
```

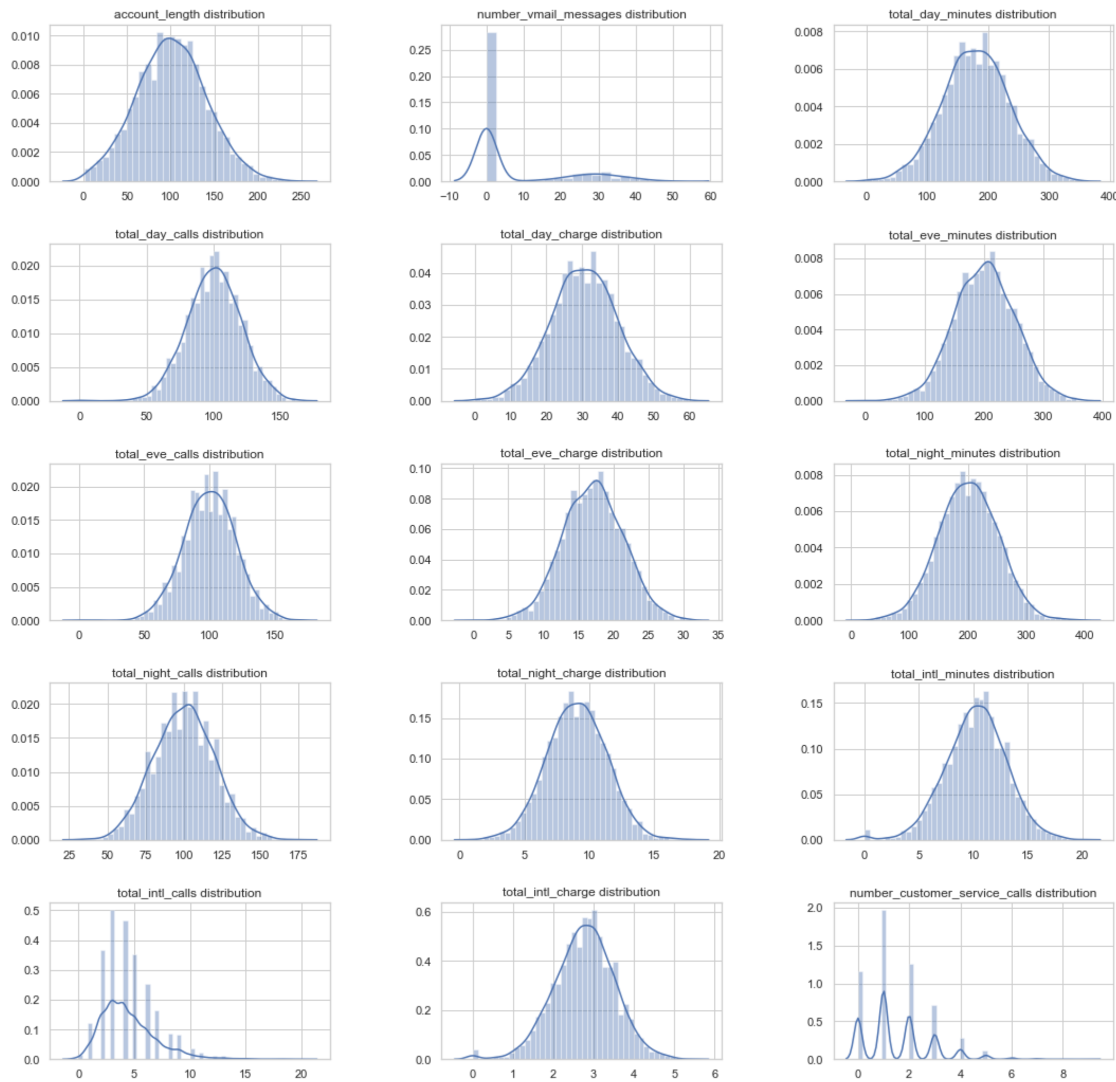
```
  voice_mail_plan  count
0              no   2411
1             yes    922
```



Most customers do not have voice mail plan. #### Numerical variables ##### Checking numerical features distributions

[217]:

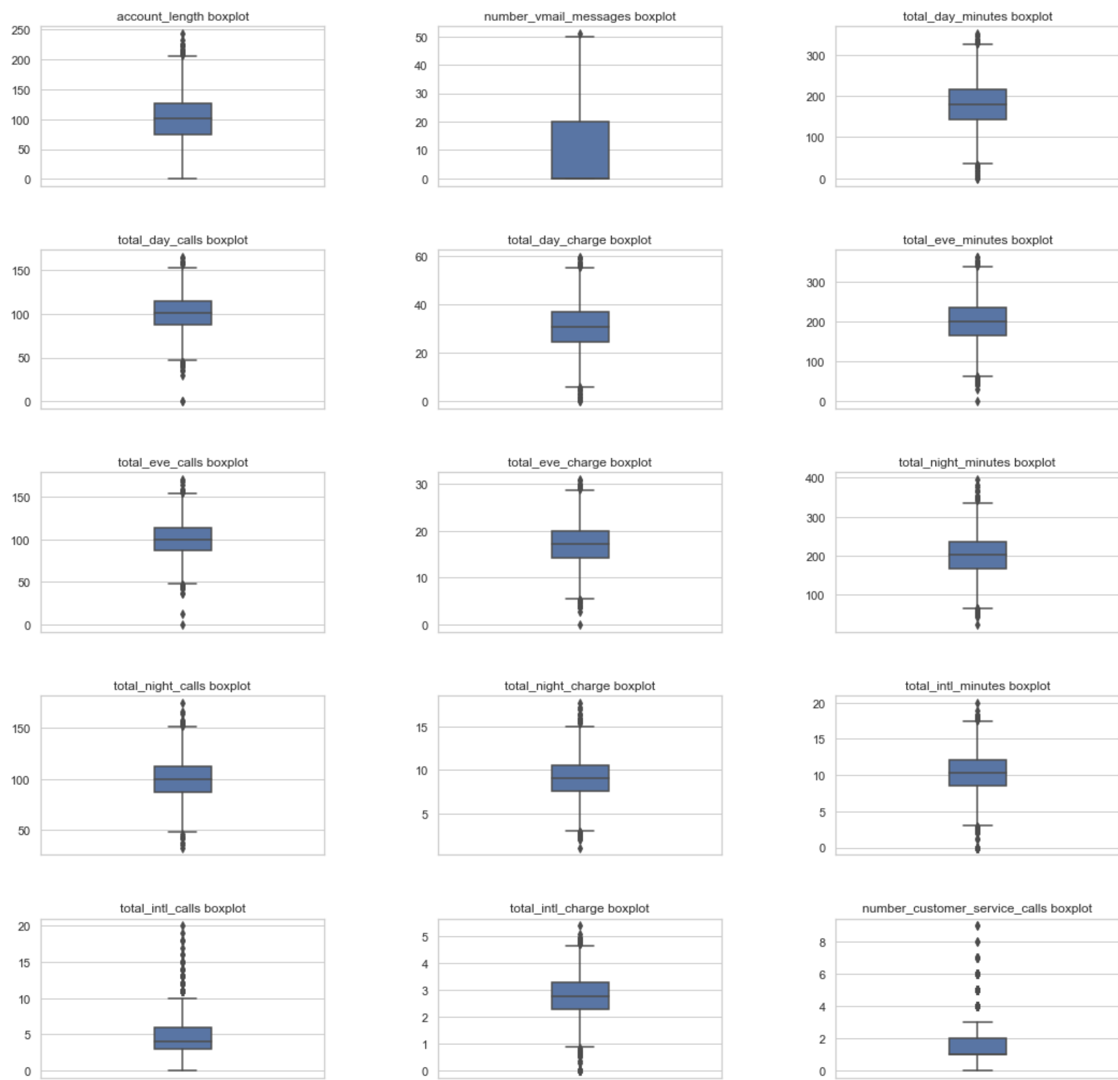
```
# Features histograms and kde
fig, axs = plt.subplots(ncols=3, nrows=5)
plt.subplots_adjust(hspace = 0.4, wspace = 0.4)
fig.set_size_inches(18, 18, forward=True)

count = 0

for i in range(5):
    for j in range(3):
        plt.sca(ax=axs[i][j])
        if count < len(df_train_num.columns):
            col = df_train_num.columns[count]
            y = sc.parallelize(df_train_num.select(col).collect())
            sns.distplot(y.collect()).set_title(col +' distribution')
        else:
            break

        count +=1
```



"account_lenght", " total_day_minutes"," total_day_calls"," total_day_charge"," total_eve_minutes"," total_eve_calls"," total_eve_charge"," total_night_minutes"," total_night_calls"," total_night_charge"," total_intl_minutes"," total_intl_charge" seem to have a normal distribution. "number_vmail_messages" has a bimodal distribution; "total_intl_calls" has a exponential distribution and "number_customer_service_calls" has a multimodal distribution.

[218]:
```
# Features boxplot
fig, axs = plt.subplots(ncols=3, nrows=5)
plt.subplots_adjust(hspace = 0.4, wspace = 0.4)
fig.set_size_inches(18, 18, forward=True)

count = 0

for i in range(5):
    for j in range(3):
```

```
        plt.sca(ax=axs[i][j])
        if count < len(df_train_num.columns):
            col = df_train_num.columns[count]
            y = sc.parallelize(df_train_num.select(col).collect())
            sns.boxplot(y=y.collect(), width=.20).set_title(col +' boxplot')
        else:
            break

        count +=1
```

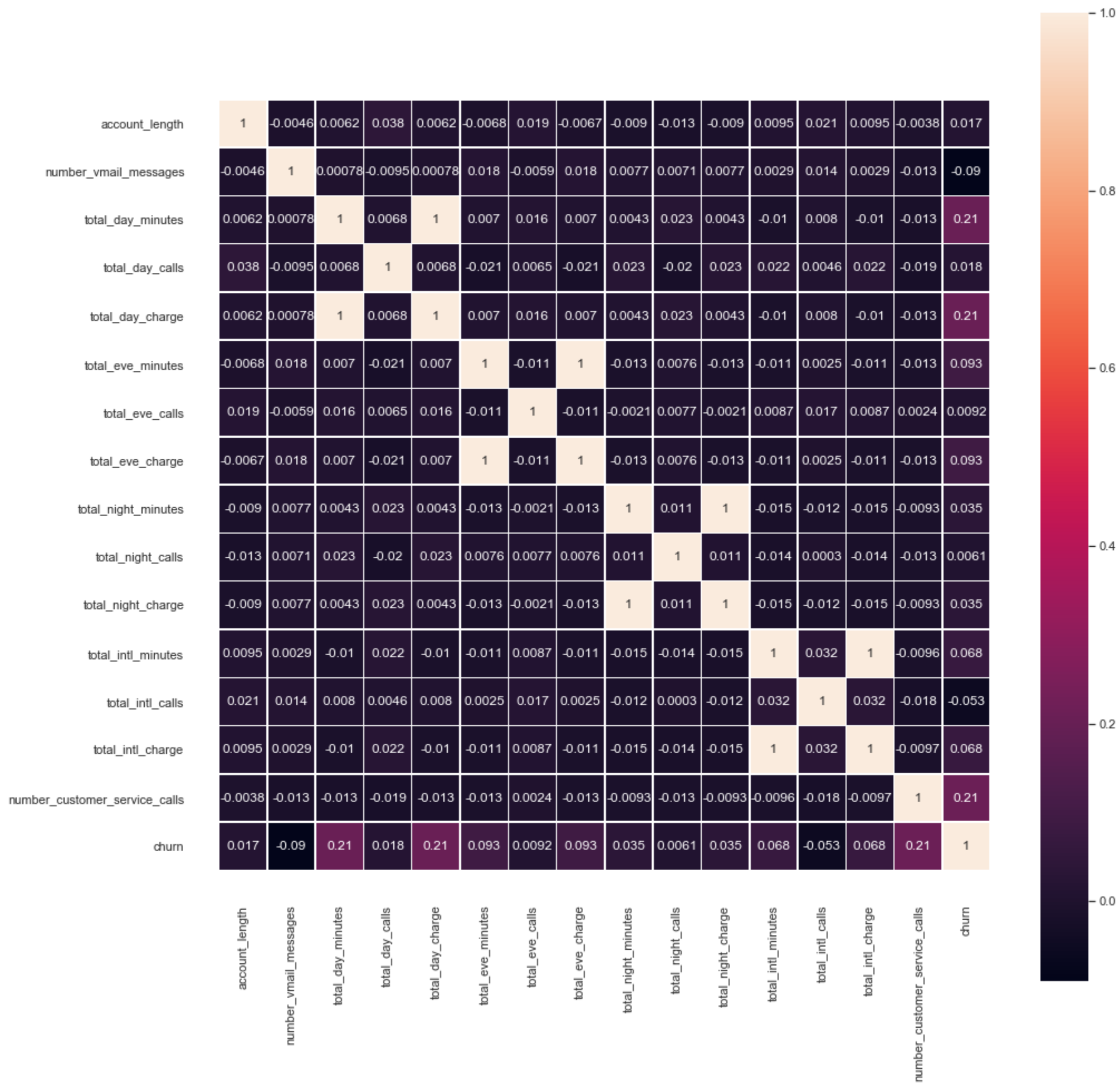


"total_intl_calls" and "number_customer_service" have a large number of outliers. ### Bivariate analysis #### Checking correlation between numerical variables

[219]:
```
# heat map of correlation values
# Adding churn column and converting to numeric format
```

```python
cols = df_train_num.columns + ['churn']
df = df_train.select(cols)

df = df.withColumn('churn', f.regexp_replace('churn', 'yes', '1'))
df = df.withColumn('churn', f.regexp_replace('churn', 'no', '0'))

# Converting to numeric
for col_name in df.columns:
    df = df.withColumn(col_name, df[col_name].cast('float'))

# convert to vector column first
vector_col = "corr_features"
assembler = VectorAssembler(inputCols=df.columns, outputCol=vector_col)
df_vector = assembler.transform(df).select(vector_col)

# get correlation matrix
corrmatrix = Correlation.corr(df_vector, vector_col).collect()[0][0]
corr = corrmatrix.toArray().tolist()

fig, ax = plt.subplots(figsize=(16,16))

g = sns.heatmap(corr, annot=True, ax=ax, square=True, linewidth=0.5)

plt.yticks(rotation=0)
g.set_xticklabels(df.columns, rotation=90)
g.set_yticklabels(df.columns)
ax.set_ylim([len(corr) + 0.5, 0])
ax.set_xlim([-0.5, len(corr)])
```

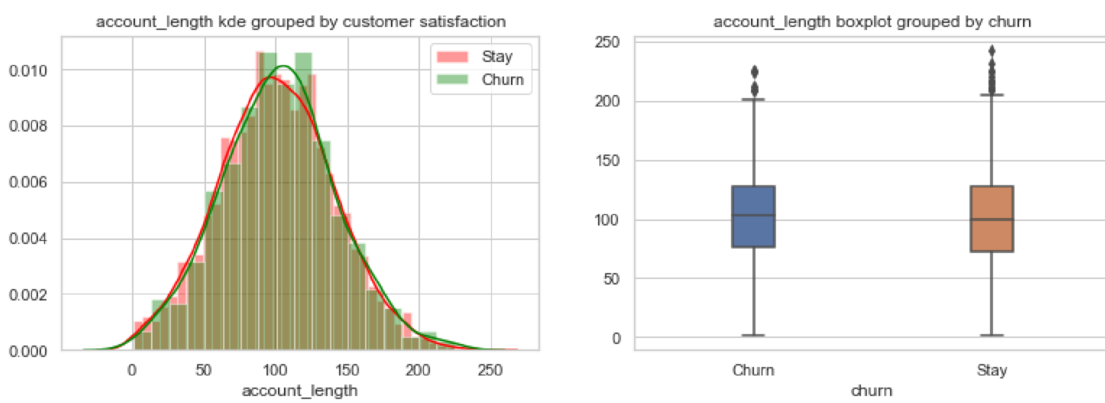
[219]:
```
(-0.5, 16)
```

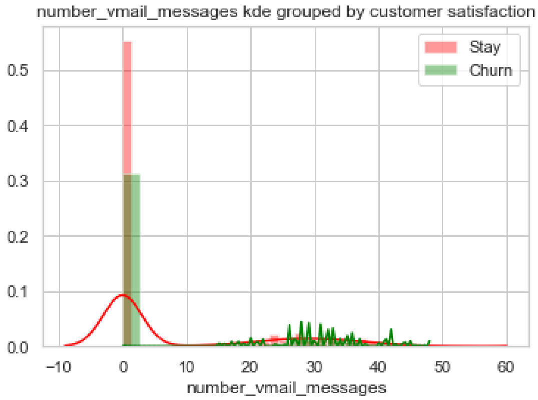
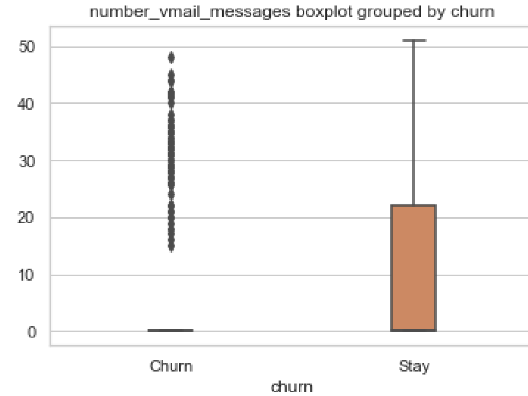
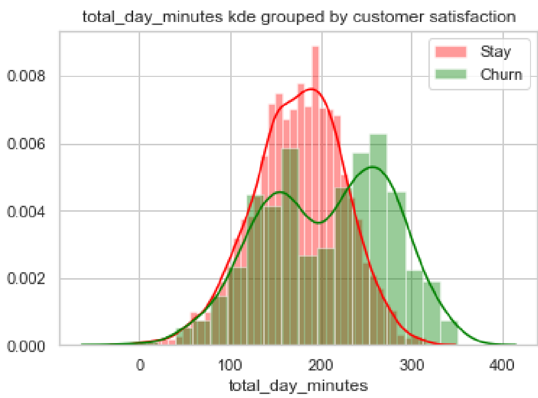
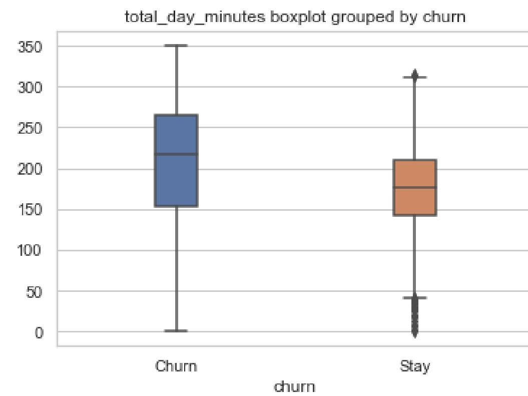
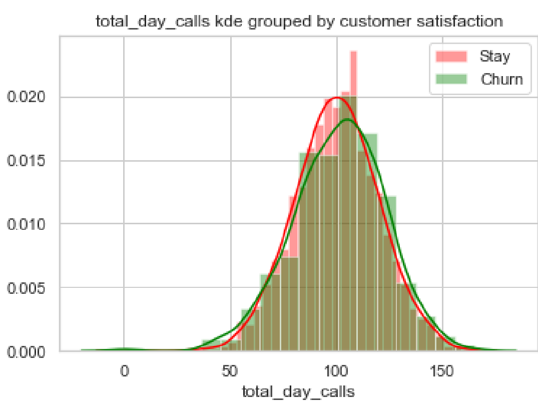
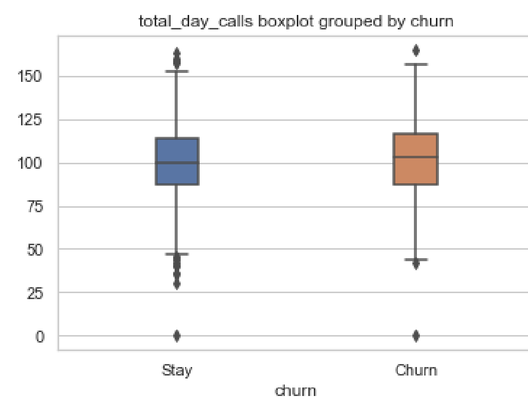


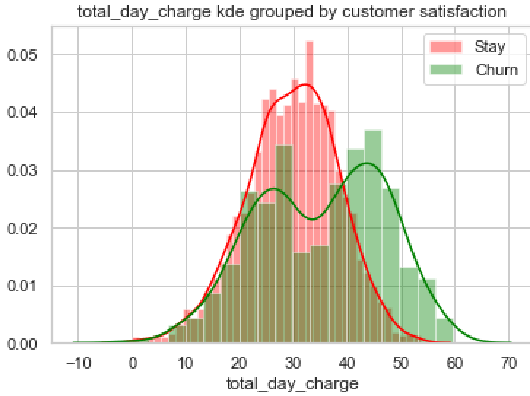
There is a high correlation between the following pairs of variables: "total_day_minutes" and "total_day_charge", "total_night_minutes" and "total_night_charge", "total_eve_minutes" and "total_eve_charge", "total_intl_minutes" and "total_intl_charge". #### Getting numerical features distribuition grouped by target variable (churn)

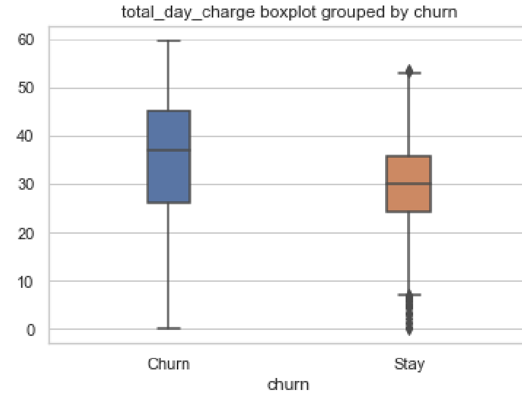
[220]:
```
# Getting numerical features distribuition grouped by churn variable
cols = df_train_num.columns + ['churn']
df = df_train.select(cols)

# Converting to numeric
for col_name in df.columns:
    if col_name != 'churn':
        df = df.withColumn(col_name, df[col_name].cast('float'))

for col in df.columns[:-1]:
```

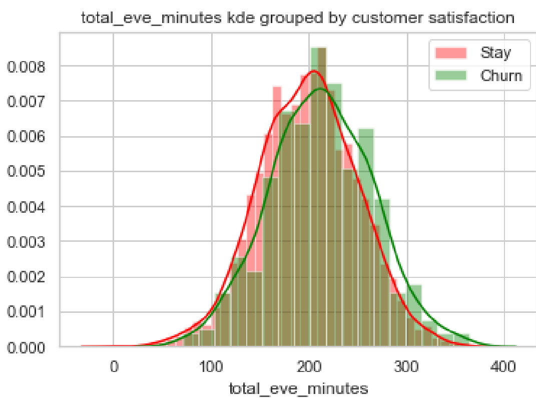
```
    y0 = sc.parallelize(df.filter(df.churn == 'no').select(col).collect())
    y1 = sc.parallelize(df.filter(df.churn == 'yes').select(col).collect())

    # Creating dictionary for values grouped by
    my_dict = {'Stay': y0.collect(), 'Churn': y1.collect()}

    # sort keys and values together
    sorted_keys, sorted_vals = zip(*sorted(my_dict.items(), key=op.
↪itemgetter(1)))

    fig, axs = plt.subplots(ncols=2)
    fig.set_size_inches(13, 4, forward=True)
    sns.distplot(y0.collect(), color='red', label='Stay', ax=axs[0], bins = 40)
    sns.distplot(y1.collect(), color='green', label='Churn', ax=axs[0], bins =␣
↪18)
    axs[0].legend()
    axs[0].set_xlabel(col)
    axs[0].set_title(col + ' kde grouped by customer satisfaction')
    sns.boxplot(data=sorted_vals, width=.18, ax=axs[1])
    axs[1].set_xlabel('churn')
    axs[1].set_xticklabels(sorted_keys)
    axs[1].set_title(col + ' boxplot grouped by churn')
    plt.show()
```

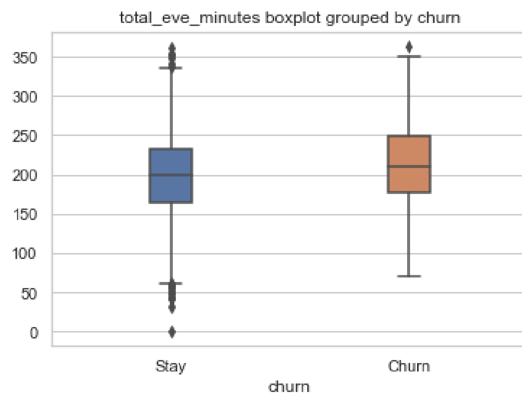


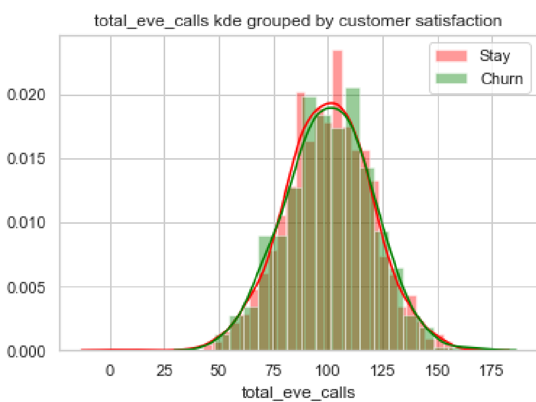

number_vmail_messages kde grouped by customer satisfaction
Stay
Churn
0.0
0.1
0.2
0.3
0.4
0.5
−10
0
10
20
30
40
50
60
number_vmail_messages

number_vmail_messages boxplot grouped by churn
0
10
20
30
40
50
Churn
Stay
churn

total_day_minutes kde grouped by customer satisfaction
Stay
Churn
0.000
0.002
0.004
0.006
0.008
0
100
200
300
400
total_day_minutes

total_day_minutes boxplot grouped by churn
0
50
100
150
200
250
300
350
Churn
Stay
churn

total_day_calls kde grouped by customer satisfaction
Stay
Churn
0.000
0.005
0.010
0.015
0.020
0
50
100
150
total_day_calls

total_day_calls boxplot grouped by churn
0
25
50
75
100
125
150
Stay
Churn
churn

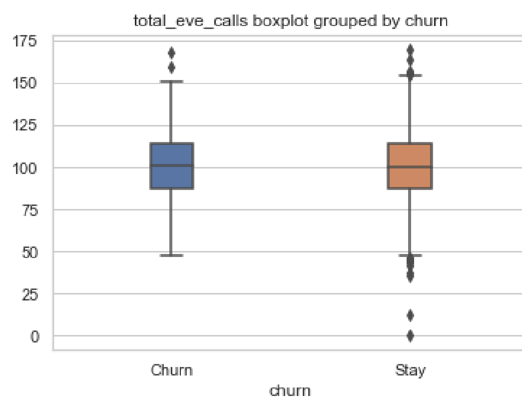

total_day_charge kde grouped by customer satisfaction
Stay
Churn
total_day_charge

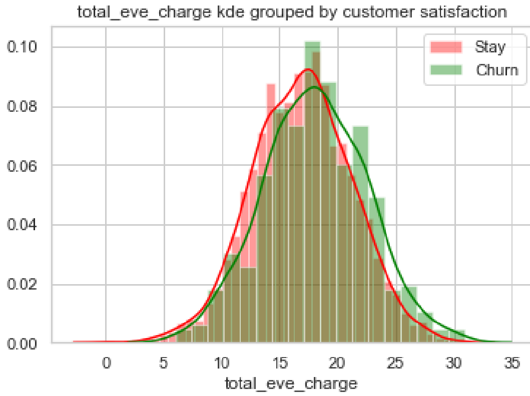

total_day_charge boxplot grouped by churn
Churn
Stay
churn

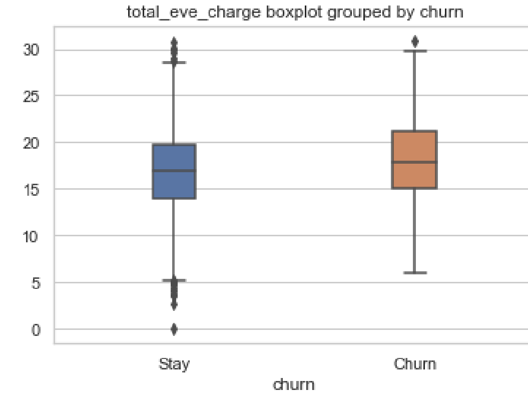

total_eve_minutes kde grouped by customer satisfaction
Stay
Churn
total_eve_minutes

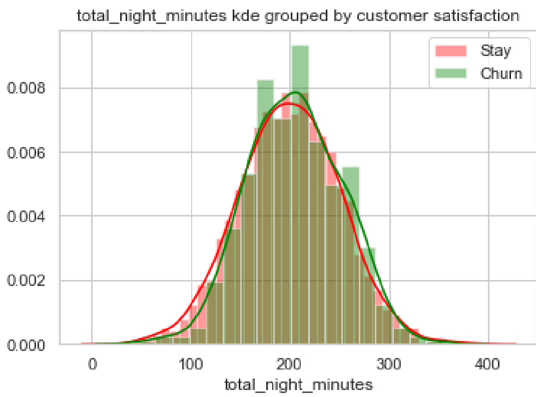

total_eve_minutes boxplot grouped by churn
Stay
Churn
churn

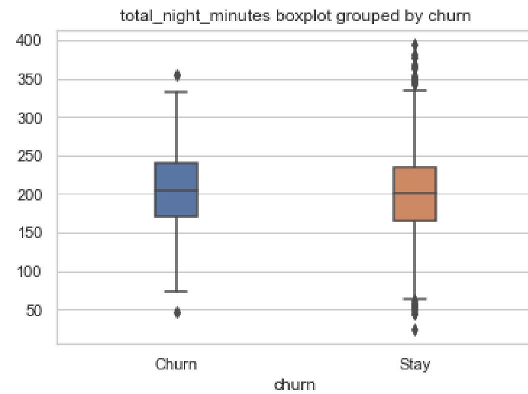

total_eve_calls kde grouped by customer satisfaction
Stay
Churn
total_eve_calls

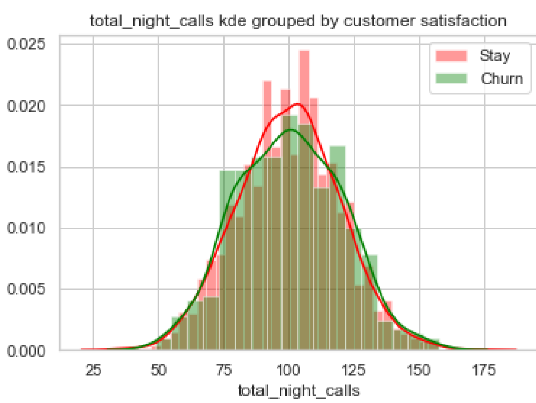

total_eve_calls boxplot grouped by churn
Churn
Stay
churn

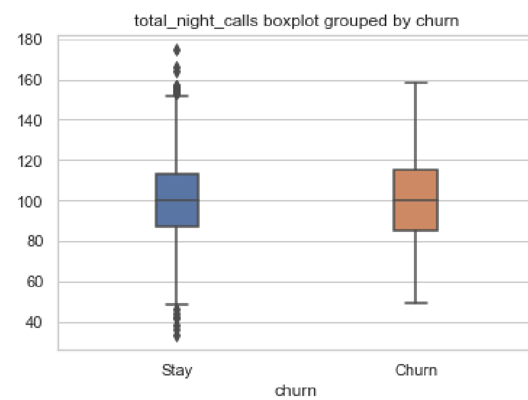

total_eve_charge kde grouped by customer satisfaction
Stay
Churn
total_eve_charge

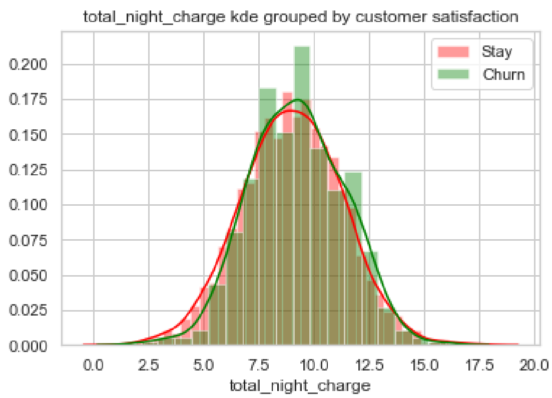

total_eve_charge boxplot grouped by churn
Stay
Churn
churn

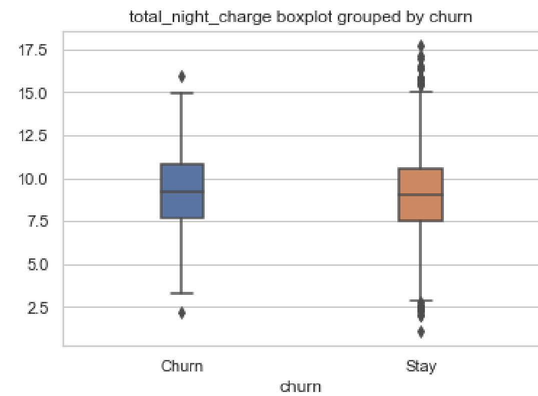

total_night_minutes kde grouped by customer satisfaction
Stay
Churn
total_night_minutes

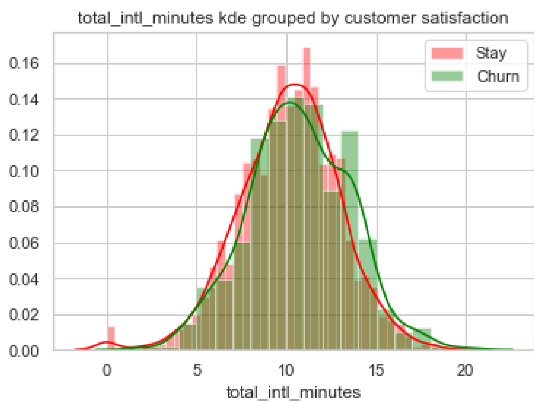

total_night_minutes boxplot grouped by churn
Churn
Stay
churn

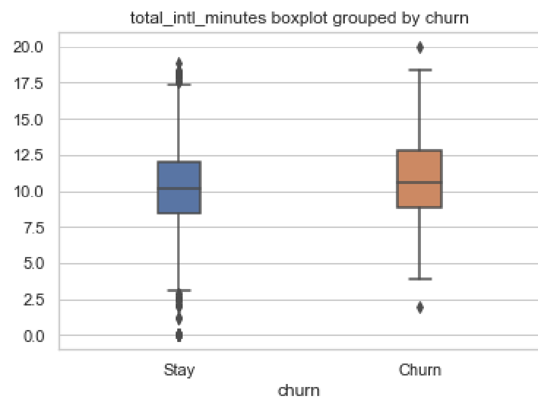

total_night_calls kde grouped by customer satisfaction
Stay
Churn
total_night_calls

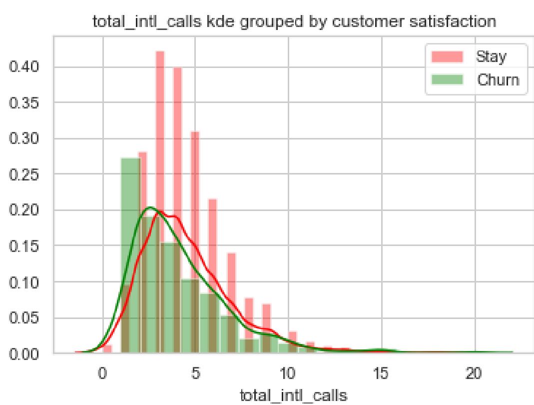

total_night_calls boxplot grouped by churn
Stay
Churn
churn

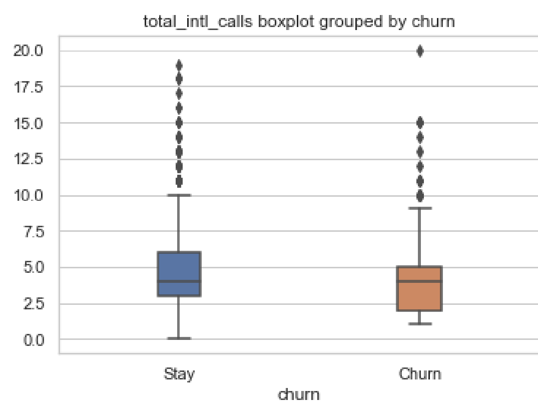

total_night_charge kde grouped by customer satisfaction
Stay
Churn
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0
total_night_charge

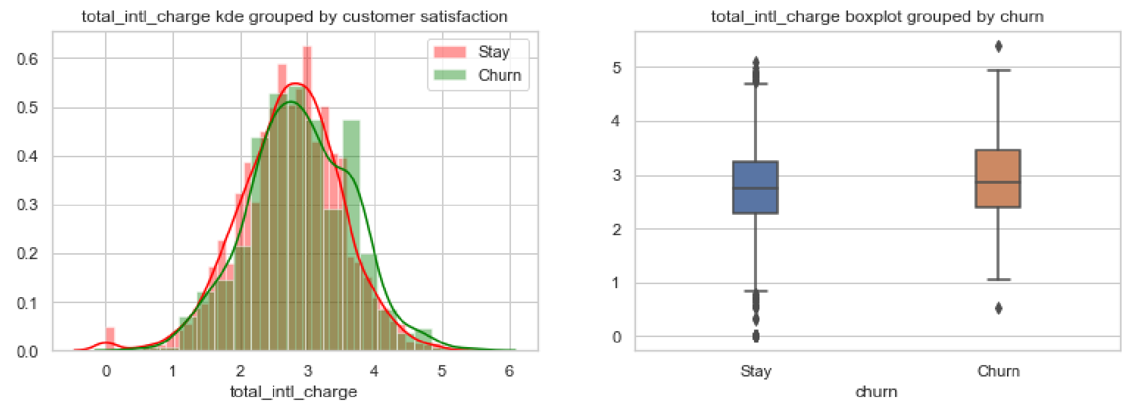

total_night_charge boxplot grouped by churn
17.5
15.0
12.5
10.0
7.5
5.0
2.5
Churn
Stay
churn

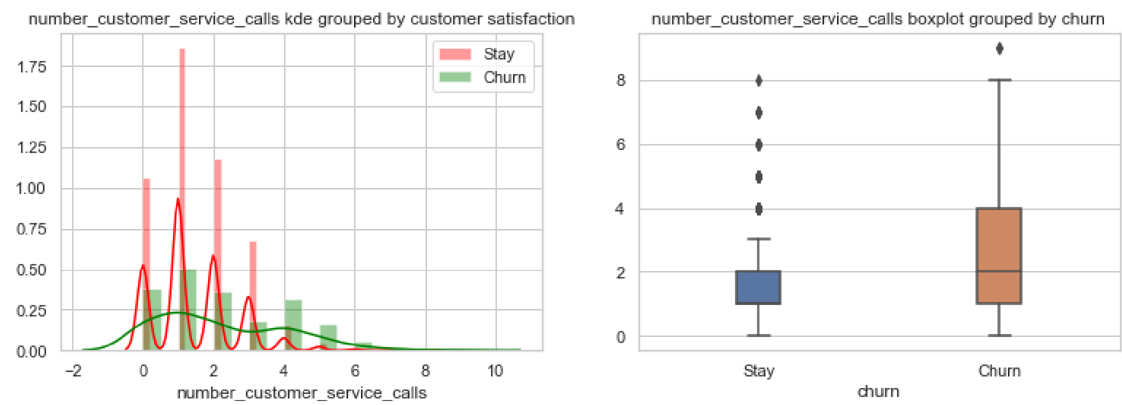

total_intl_minutes kde grouped by customer satisfaction
Stay
Churn
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0 5 10 15 20
total_intl_minutes


total_intl_minutes boxplot grouped by churn
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Stay
Churn
churn


total_intl_calls kde grouped by customer satisfaction
Stay
Churn
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0 5 10 15 20
total_intl_calls


total_intl_calls boxplot grouped by churn
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Stay
Churn
churn

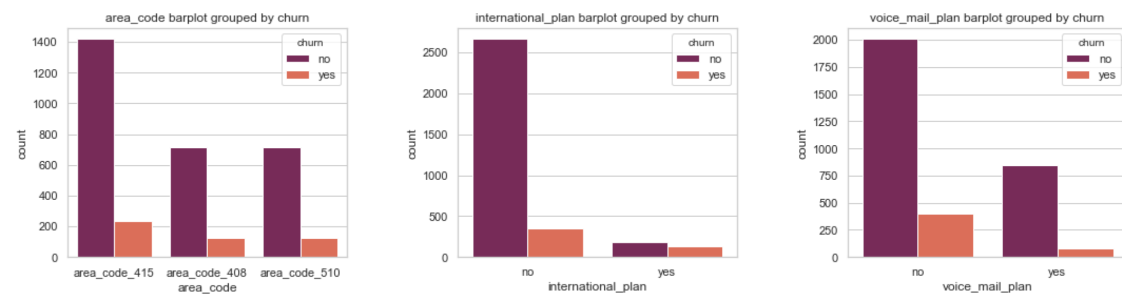




Getting categorical features barplot grouped by churn variable

[221]:
```
# categorical variables boxplot grouped by churn
cols = df_train_cat.columns
cols.remove('state')
df = df_train.select(cols)

fig, axs = plt.subplots(ncols=3)
plt.subplots_adjust(hspace = 0.4, wspace = 0.4)
fig.set_size_inches(18, 4, forward=True)

count = 0

for j in range(3):
    plt.sca(ax=axs[j])
    if count < len(df.columns):
        col = df.columns[count]
```

```
        df_g = df.groupBy(col, 'churn').count().orderBy('count',␣
↪ascending=False).toPandas()
        sns.barplot(x=col, y='count', hue="churn", data=df_g, palette="rocket")
        axs[j].set_xlabel(col)
        axs[j].set_title(col + ' barplot grouped by churn')
    else:
        break
    count +=1
```

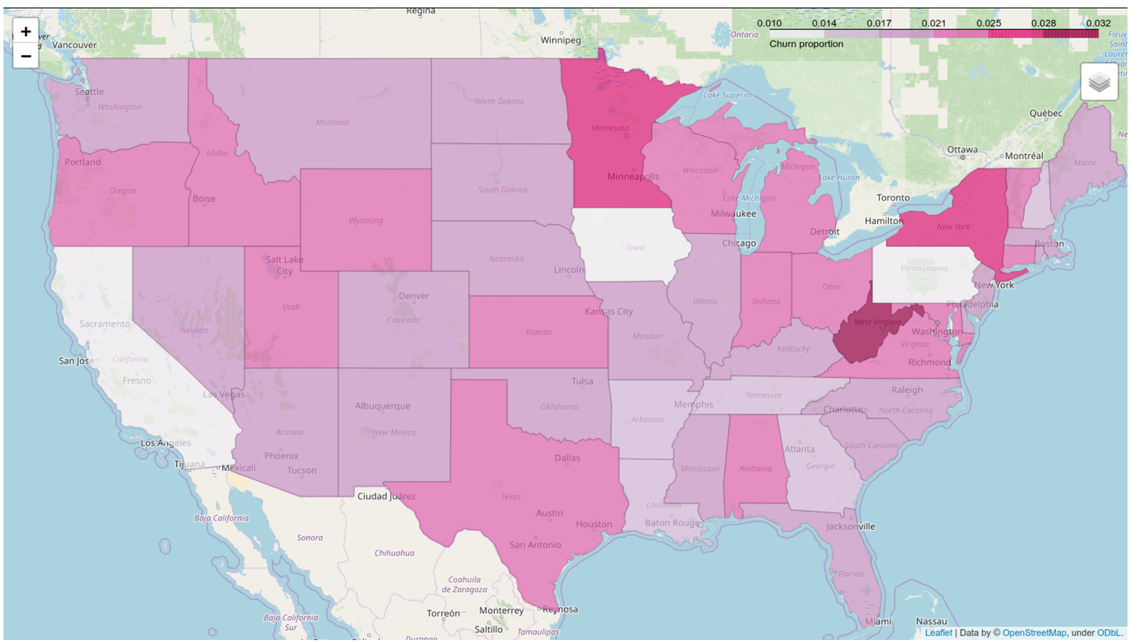


Churn by state

[140]:
```
# Map graph exibing churn proportion
# Load the shape of the zone (US states)
state_geo = os.path.join('', 'us-states.json')

# state data
# churn values and proportion
state_data = df_train_cat.groupBy('state').count().orderBy('count',␣
 ↪ascending=False) \
            .withColumn('percent', f.col('count')/f.sum('count').over(Window.
 ↪partitionBy())) \
            .orderBy('percent', ascending=False) \
            .toPandas()


# Initialize the map:
m = folium.Map(location=[37, -102], zoom_start=5)

# Add the color for the chloropleth:
m.choropleth(
 geo_data=state_geo,
 name='Churn proportion by state',
 data=state_data,
 columns=['state', 'percent'],
 key_on='feature.id',
 fill_color='PuRd',
```

```
 fill_opacity=0.7,
 line_opacity=0.2,
 legend_name='Churn proportion'
)
folium.LayerControl().add_to(m)

# Save to html
m.save('#churn_proportion.html')

display(m)

# # Loading map image
# Image(filename='churn_proportion_by_state.png')
```

[140]:



Checking categorical features

[222]:

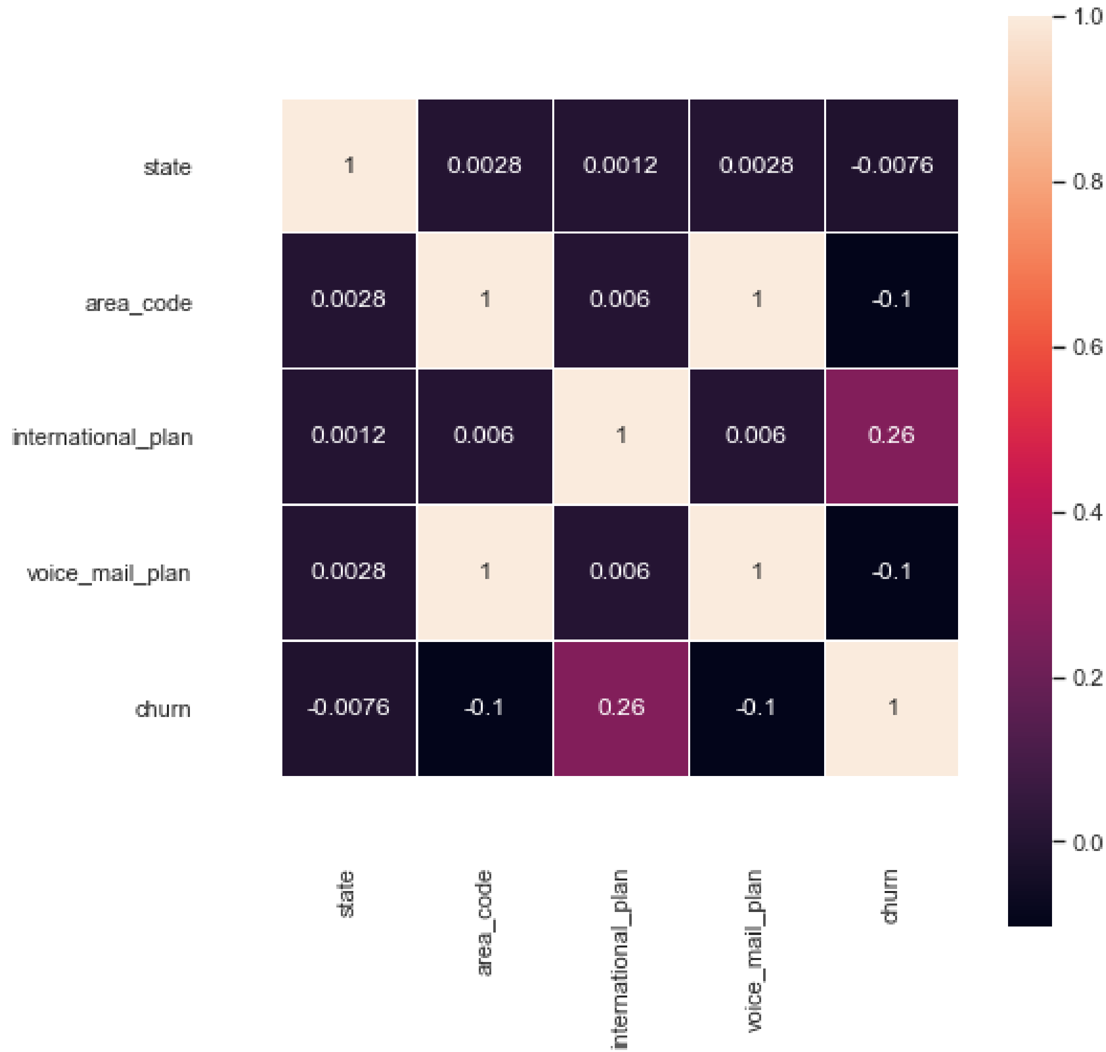
```
# Converting categoric features to numeric
df_new_train_cat = df_train_cat

# Converting to binary
for col in ['churn', 'international_plan', 'voice_mail_plan']:
    df_new_train_cat = df_new_train_cat.withColumn(col, f.regexp_replace(col,␣
 ↪'yes', '1'))
    df_new_train_cat = df_new_train_cat.withColumn(col, f.regexp_replace(col,␣
 ↪'no', '0'))
```

```
dic_state = {}
count = 1

# Changing state variable
for state in df_new_train_cat.select("state").distinct().collect():
    dic_state[state.state] = str(count)
    count += 1

for item in dic_state.items():
    df_new_train_cat = df_new_train_cat.withColumn('state', f.
 ↪regexp_replace('state', item[0], item[1]))


# Changing area_code variable
df_new_train_cat = df_new_train_cat.withColumn('area_code', f.
 ↪regexp_replace(col, 'area_code_408', '1'))
df_new_train_cat = df_new_train_cat.withColumn('area_code', f.
 ↪regexp_replace(col, 'area_code_510', '2'))
df_new_train_cat = df_new_train_cat.withColumn('area_code', f.
 ↪regexp_replace(col, 'area_code_415', '3'))

# Converting to numeric
for col_name in df_new_train_cat.columns:
    df_new_train_cat = df_new_train_cat.withColumn(col_name,␣
 ↪df_new_train_cat[col_name].cast('float'))
```

[223]:
```
# heat map of correlation values
df = df_new_train_cat

# convert to vector column first
vector_col = "corr_features"
assembler = VectorAssembler(inputCols=df.columns, outputCol=vector_col)
df_vector = assembler.transform(df).select(vector_col)

# get correlation matrix
corrmatrix = Correlation.corr(df_vector, vector_col).collect()[0][0]
corr = corrmatrix.toArray().tolist()

fig, ax = plt.subplots(figsize=(8,8))

g = sns.heatmap(corr, annot=True, ax=ax, square=True, linewidth=0.5)

plt.yticks(rotation=0)
g.set_xticklabels(df.columns, rotation=90)
g.set_yticklabels(df.columns)
ax.set_ylim([len(corr) + 0.5, 0])
```

```
ax.set_xlim([-0.5, len(corr)])
```

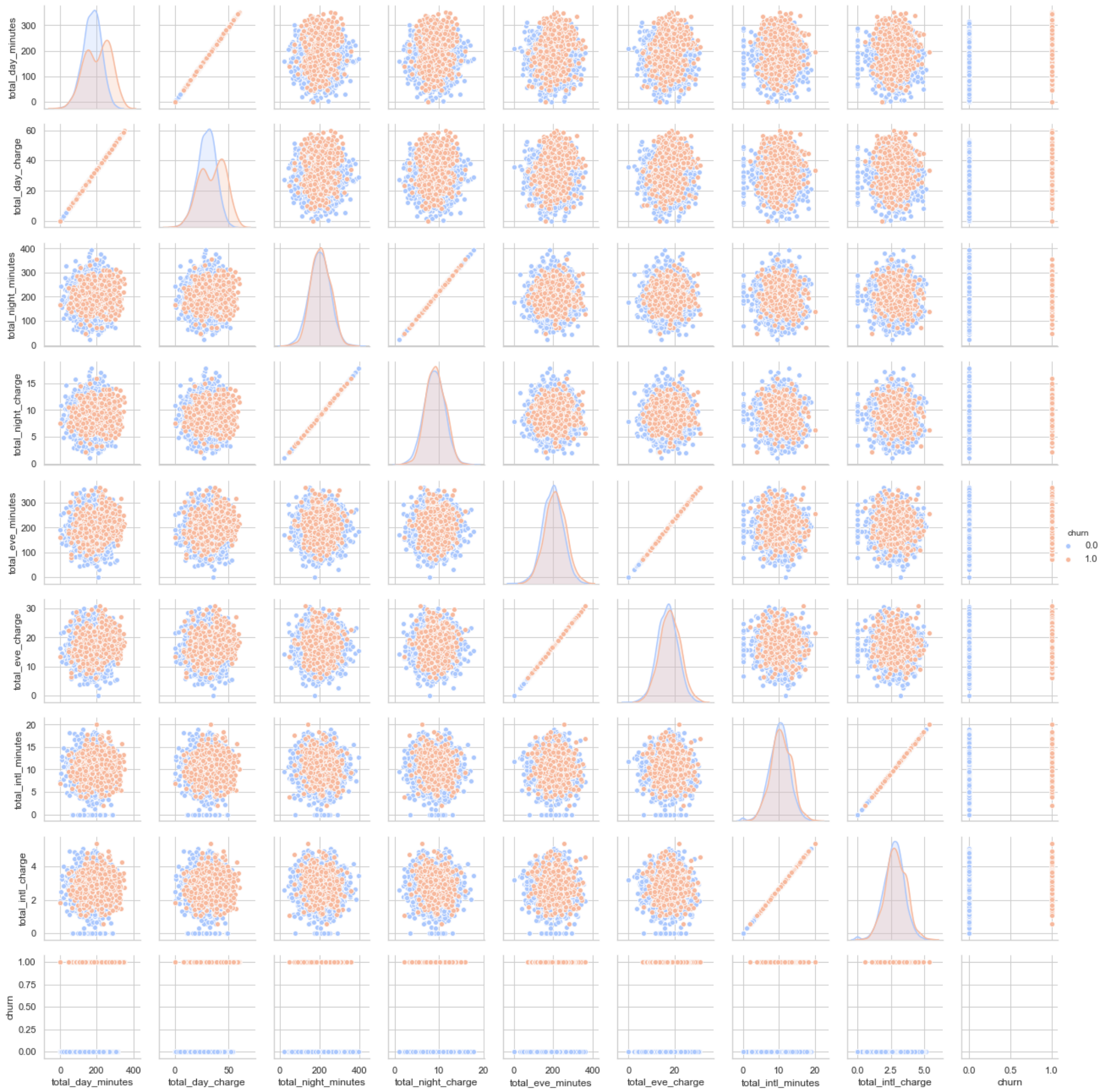
[223]: (-0.5, 5)



1.2.2 Multivariate analysis

[224]:
```
# Adding churn column and converting to numeric format
cols = ["total_day_minutes", "total_day_charge", "total_night_minutes",
        "total_night_charge", "total_eve_minutes", "total_eve_charge",
        "total_intl_minutes", "total_intl_charge", "churn"]

df = df_train.select(cols)

df = df.withColumn('churn', f.regexp_replace('churn', 'yes', '1'))
```

```
df = df.withColumn('churn', f.regexp_replace('churn', 'no', '0'))

# Converting to numeric
for col_name in df.columns:
    df = df.withColumn(col_name, df[col_name].cast('float'))

# paiplot
g = sns.pairplot(df.toPandas(), hue='churn', palette='coolwarm')
g.fig.set_size_inches(18,18)
```



[225]:
```
# Checking relation between categorical variables grouped by churn
values = df_train_cat.select('state', 'international_plan', 'voice_mail_plan',␣
 ↪'churn') \
```

```
                    .groupBy('international_plan', 'voice_mail_plan', 'churn').
↪count() \
                    .orderBy('count', ascending=False).toPandas()

display(values)
```

```
  international_plan voice_mail_plan churn  count
0                 no              no    no   1878
1                 no             yes    no    786
2                 no              no   yes    302
3                yes              no    no    130
4                yes              no   yes    101
5                yes             yes    no     56
6                 no             yes   yes     44
7                yes             yes   yes     36
```

1.3 Feature Engineering

1.3.1 Cleaning sataset

In this session, I converted categorical variables to numeric format, removed high correlated features and unnecessary variable.

[226]:
```
# Getting data on rdd format
trainRdd = sc.textFile("data/projeto4_telecom_treino.csv")
testRdd = sc.textFile("data/projeto4_telecom_teste.csv")
```

[227]:
```
# Removing first line of the file (head)
trainRdd2 = trainRdd.filter(lambda x: "state" not in x)
testRdd2 = testRdd.filter(lambda x: "state" not in x)

# Getting clean head
cols = trainRdd.collect()[0].replace("\"",'').split(",")
cols = [col for col in cols if "minute" not in col]
del cols[0]
```

[228]:
```
# Transformação e Limpeza
dic_state = {}
count = 1

# Changing state variable
for state in df_train.select("state").distinct().collect():
    dic_state[state.state] = count
    count += 1

dic_area_code = {'area_code_408': 1,
                 'area_code_510': 2,
                 'area_code_415': 3}
```

```python
dic_binary = {"no": 0,
              "yes": 1}

# Removing all minutes variables (highly correlated to charge features)
cols = trainRdd.collect()[0].replace("\"",'').split(",")

removal_index = [i for i in range(len(cols)) if 'minute' in cols[i]]

cols = trainRdd.collect()[0].replace("\"",'').split(",")
cols = [col for col in cols if "minute" not in col]
del cols[0]


# Function to transform and clean rdd data
def cleanRDD(autoStr):

    # checking indexing
    if isinstance(autoStr, int):
        return autoStr

    # Separate each index with a comma (column separator)
    attList = autoStr.replace("\"",'').split(",")

    # Changing and converting categorical variables to numeric
    attList[1] = dic_state[attList[1]]
    attList[3] = dic_area_code[attList[3]]
    attList[4] = dic_binary[attList[4]]
    attList[5] = dic_binary[attList[5]]
    attList[20] = dic_binary[attList[20]]

    # Converting numeric variable to numeric format
    attList[2] = pd.to_numeric(attList[2])
    attList[6:20] = pd.to_numeric(attList[6:20])

    # Removing high correlated features and "id" variable
    count = 0

    for i in removal_index:
        del attList[i-count]
        count +=1

    del attList[0]

    # Creating dictionary for store line values
    line_dict = {}
```

```
    for i in range(len(cols)):
        line_dict[cols[i]] = attList[i]

    print(line_dict)

    line = Row(**line_dict)

    return line
```

[229]:
```
# Cleaning train and test datasets
cleanTrainRDD = trainRdd2.map(cleanRDD)
cleanTestRDD = testRdd2.map(cleanRDD)
```

[230]:
```
# Converting to a LabeledPoint (target, vector [resources])
def transformVar(row):
    obj = (row["churn"], Vectors.dense([row[col] for col in cols if col !=␣
 ↪"churn"]))
    return obj
```

[231]:
```
# Use RDD, apply the function, convert to Dataframe and apply the select()␣
 ↪function
# train data
cleanTrainRDD2 = cleanTrainRDD.map(transformVar)
train_DF = spSession.createDataFrame(cleanTrainRDD2,["label", "features"])
train_DF.select("label","features").show(10)

# test data
cleanTestRDD2 = cleanTestRDD.map(transformVar)
test_DF = spSession.createDataFrame(cleanTestRDD2,["label", "features"])
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|    0|[44.0,128.0,3.0,0...|
|    0|[33.0,107.0,3.0,0...|
|    0|[5.0,137.0,3.0,0...|
|    0|[33.0,84.0,1.0,1...|
|    0|[49.0,75.0,3.0,1...|
|    0|[31.0,118.0,2.0,1...|
|    0|[43.0,121.0,2.0,0...|
|    0|[25.0,147.0,3.0,1...|
|    0|[3.0,117.0,1.0,0...|
|    0|[41.0,141.0,3.0,1...|
+-----+--------------------+
only showing top 10 rows
```

1.3.2 Applying MinMaxScaler to data

[232]:
```
# Calling MinMaxScaler function
minmax = MinMaxScaler(inputCol="features", outputCol="MinMaxScaledFeatures")

#Train data
# Compute summary statistics by fitting the StandardScaler
minmaxTrainModel = minmax.fit(train_DF)

# Normalize each feature to have unit standard deviation.
minmaxTrain = minmaxTrainModel.transform(train_DF)

#Test data
# Compute summary statistics by fitting the StandardScaler
minmaxTestModel = minmax.fit(test_DF)

# Normalize each feature to have unit standard deviation.
minmaxTest = minmaxTestModel.transform(test_DF)
```

1.3.3 Applying StandardScale to data

[233]:
```
# Calling StandardScaler function
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures",
                        withStd=True, withMean=False)

# Train data
# Compute summary statistics by fitting the StandardScaler
scalerTrainModel = scaler.fit(minmaxTrain)

# Normalize each feature to have unit standard deviation.
scaledTrainData = scalerTrainModel.transform(minmaxTrain)

# Test data
# Compute summary statistics by fitting the StandardScaler
scalerTestModel = scaler.fit(minmaxTest)

# Normalize each feature to have unit standard deviation.
scaledTestData = scalerTestModel.transform(minmaxTest)
```

1.4 Training models

To predict Customer Churn, I chose to use Logistic Regression to get information on whether customers are going to cancel their plan and their likelihood. ### Logistic Regression

[234]:
```
# Binomial model
lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
```

```
# Fit binomial model
lrModel = lr.fit(scaledTrainData.select('label', 'features'))

# Multinomial model
mlr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8,␣
↪family="multinomial")

# Fit multinomial model the model
mlrModel = mlr.fit(scaledTrainData.select('label', 'features'))

# binomial model predictions
binom_predictions = lrModel.transform(scaledTestData.select('label',␣
↪'features'))

# multinomial model predictions
multinom_predictions = mlrModel.transform(scaledTestData.select('label',␣
↪'features'))
```

[235]:
```
# Evaluating models
# Select (prediction, true label) and compute test error and accuracy

evaluator = MulticlassClassificationEvaluator(
    labelCol="label", predictionCol="prediction", metricName="accuracy")

# Evaluating binomial model
bin_accuracy = evaluator.evaluate(binom_predictions)
print("Test Error for binomial model = %g " % (1.0 - bin_accuracy))
print("Accuracy for binomial model = %g " % (bin_accuracy))

# Evaluating multinomial model
mult_accuracy = evaluator.evaluate(multinom_predictions)
print("Test Error for multinomial model = %g " % (1.0 - mult_accuracy))
print("Accuracy for multinomial model = %g " % (mult_accuracy))
```

```
Test Error for binomial model = 0.134373
Accuracy for binomial model = 0.865627
Test Error for multinomial model = 0.134373
Accuracy for multinomial model = 0.865627
```

1.5 Trying to optimizate model

1.5.1 Logistic regression optimization

Selecting most important features using PCA algorithm

[236]:
```
# Applying pipeline to select features and train model

# Binomial model
lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
```

```
# Multinomial model
mlr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8,␣
↪family="multinomial")

# Checking model accuracy by varying the number of features
for k in range(3,8):

    # Applying Dimension Reduction with PCA
    bankPCA = PCA(k = k, inputCol = "features", outputCol = "pcaFeatures")

    # Applying pipeline to select features and train model
    binom_pipeline = Pipeline(stages=[bankPCA, lr])
    multinom_pipeline = Pipeline(stages=[bankPCA, mlr])

    # Fit the pipeline to training documents.
    binom_model = binom_pipeline.fit(scaledTrainData)
    multinom_model = multinom_pipeline.fit(scaledTrainData)

    # Make predictions on test documents.
    binom_predictions = binom_model.transform(scaledTestData)
    multinom_predictions = multinom_model.transform(scaledTestData)

    bin_accuracy = evaluator.evaluate(binom_predictions)
    print("Accuracy for binomial model for %g features = %g " % (k,␣
↪bin_accuracy))

    # Evaluating multinomial model
    mult_accuracy = evaluator.evaluate(multinom_predictions)
    print("Accuracy for multinomial model for %g features = %g " % (k,␣
↪mult_accuracy))
```

```
Accuracy for binomial model for 3 features = 0.865627
Accuracy for multinomial model for 3 features = 0.865627
Accuracy for binomial model for 4 features = 0.865627
Accuracy for multinomial model for 4 features = 0.865627
Accuracy for binomial model for 5 features = 0.865627
Accuracy for multinomial model for 5 features = 0.865627
Accuracy for binomial model for 6 features = 0.865627
Accuracy for multinomial model for 6 features = 0.865627
Accuracy for binomial model for 7 features = 0.865627
Accuracy for multinomial model for 7 features = 0.865627
```

Decreasing the number of variables did not change the model's performance. ###### Using standard deviation dataset

[237]:
```
# binomial model
lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
```

```
sd_train = scaledTrainData.select('label', 'scaledFeatures').
 ↪withColumnRenamed("scaledFeatures","features")
sd_test = scaledTrainData.select('label', 'scaledFeatures').
 ↪withColumnRenamed("scaledFeatures","features")

# training model
sd_model = lr.fit(sd_train)

# Predction
sd_prediction = sd_model.transform(sd_test)

sd_accuracy = evaluator.evaluate(sd_prediction)
print("Accuracy for binomial model for standard scaled data = %g " %␣
 ↪(sd_accuracy))
```

```
Accuracy for binomial model for standard scaled data = 0.855086
```

Using min-max scaled deviation dataset

[239]:
```
# binomial model
lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

minmax_train = scaledTrainData.select('label', 'MinMaxScaledFeatures') \
    .withColumnRenamed("MinMaxScaledFeatures","features")

minmax_test = scaledTrainData.select('label', 'MinMaxScaledFeatures') \
    .withColumnRenamed("MinMaxScaledFeatures","features")

# training model
minmax_model = lr.fit(minmax_train)

# Predction
minmax_prediction = minmax_model.transform(minmax_test)

minmax_accuracy = evaluator.evaluate(minmax_prediction)
print("Accuracy for binomial model for min-max-standard scaled data = %g " %␣
 ↪(minmax_accuracy))
```

```
Accuracy for binomial model for min-max-standard scaled data = 0.855086
```

Applying standard scale to non-binary features

[241]:
```
# Converting to a LabeledPoint (target, vector [resources])
cols = trainRdd.collect()[0].replace("\"",'').split(",")
cols = trainRdd.collect()[0].replace("\"",'').split(",")
cols = [col for col in cols if "minute" not in col]
del cols[0]
```

```
drop_cols = ['churn', 'international_plan', 'voice_mail_plan']

def labeledPoint(row):
    obj = (row["churn"], row["international_plan"], \
           row["voice_mail_plan"], Vectors.dense([row[col] for col in cols if␣
 ↪col not in drop_cols]))
    return obj
```

[242]:
```
cleanTrainRDD3 = cleanTrainRDD.map(labeledPoint)
train_DF2 = spSession.createDataFrame(cleanTrainRDD3,["label",␣
 ↪'international_plan', \
                                                      'voice_mail_plan',␣
 ↪"features"])

# test data
cleanTestRDD3 = cleanTestRDD.map(labeledPoint)
test_DF2 = spSession.createDataFrame(cleanTestRDD3,["label",␣
 ↪'international_plan', \
                                                    'voice_mail_plan',␣
 ↪"features"])
```

[243]:
```
# Calling StandardScaler function
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures",
                        withStd=True, withMean=False)

# Train data
# Compute summary statistics by fitting the StandardScaler
scalerTrainModel = scaler.fit(train_DF2)

# Normalize each feature to have unit standard deviation.
part_scaledTrainData = scalerTrainModel.transform(train_DF2)

# Test data
# Compute summary statistics by fitting the StandardScaler
scalerTestModel = scaler.fit(test_DF2)

# Normalize each feature to have unit standard deviation.
part_scaledTestData = scalerTestModel.transform(test_DF2)
```

[251]:
```
# Converting partially scaled train data to labeledpoint format
df1 = part_scaledTrainData.select('label')
df_features = part_scaledTrainData.select('international_plan',␣
 ↪'voice_mail_plan', 'scaledFeatures')

# convert to vector column first
assembler = VectorAssembler(inputCols=df_features.columns, outputCol="features")
df2 = assembler.transform(df_features).select("features")
```

```
# since there is no common column between these two dataframes add row_index so␣
↪that it can be joined
df1=df1.withColumn('row_index', row_number().over(Window.
↪orderBy(monotonically_increasing_id())))
df2=df2.withColumn('row_index', row_number().over(Window.
↪orderBy(monotonically_increasing_id())))

sd_train_data = df1.join(df2, on=["row_index"]).drop("row_index")
```

[253]:

```
# Converting partially scaled test data to labeledpoint format
df1 = part_scaledTestData.select('label')
df_features = part_scaledTestData.select('international_plan',␣
↪'voice_mail_plan', 'scaledFeatures')

# convert to vector column first
vector_col = "corr_features"
assembler = VectorAssembler(inputCols=df_features.columns, outputCol="features")
df2 = assembler.transform(df_features).select("features")

# since there is no common column between these two dataframes add row_index so␣
↪that it can be joined
df1=df1.withColumn('row_index', row_number().over(Window.
↪orderBy(monotonically_increasing_id())))
df2=df2.withColumn('row_index', row_number().over(Window.
↪orderBy(monotonically_increasing_id())))

sd_test_data = df1.join(df2, on=["row_index"]).drop("row_index")
```

[256]:

```
# binomial model
lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# training model
sd_model = lr.fit(sd_train_data)

# Predction
sd_prediction = sd_model.transform(sd_test_data)

sd_accuracy = evaluator.evaluate(sd_prediction)
print("Accuracy for binomial model for standard scaled data = %g " %␣
↪(sd_accuracy))
```

```
Accuracy for binomial model for standard scaled data = 0.865627
```

Tuning Hyperparameters using GridSearch

[257]:

```
# Logistic Regression
lr = LogisticRegression(maxIter=10)
```

```python
# Principal component analysys
bankPCA = PCA(inputCol = "features", outputCol = "pcaFeatures")

# Configure an ML pipeline, which consists of two stages: pca and lr.
pipeline = Pipeline(stages=[bankPCA, lr])

paramGrid = ParamGridBuilder() \
    .addGrid(bankPCA.k, [3, 7]) \
    .addGrid(lr.regParam, [0.1, 0.01]) \
    .build()

crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(),
                          numFolds=2)  # use 3+ folds in practice

# Run cross-validation, and choose the best set of parameters.
cvModel = crossval.fit(sd_train_data)

# Make predictions on test data. cvModel uses the best model found (lrModel).
prediction = cvModel.transform(sd_test_data)

# Getting model accuracy
accuracy = evaluator.evaluate(prediction)
print("Accuracy for binomial model for standard scaled data = %g " % (accuracy))
```

```
Accuracy for binomial model for standard scaled data = 0.873425
```

1.6 Final model

The model increased by GridSearch exibed the best result. The final model presents accuracy of 0.87. The results are shown in the table below including churn probability predicted by model.

[258]:
```python
churn = prediction.select("label").collect()
pred = prediction.select("prediction").collect()
print(confusion_matrix(churn, pred), "\n")
print(classification_report(churn, pred))
```

```
[[1440    3]
 [ 208   16]]

              precision    recall  f1-score   support

           0       0.87      1.00      0.93      1443
           1       0.84      0.07      0.13       224

    accuracy                           0.87      1667
   macro avg       0.86      0.53      0.53      1667
```

```
weighted avg       0.87      0.87      0.82      1667
```

[198]:
```python
prediction.select('label', 'prediction', 'probability').show()
```

```
+-----+----------+--------------------+
|label|prediction|         probability|
+-----+----------+--------------------+
|    0|       0.0|[0.86376816665536...|
|    0|       0.0|[0.88532873786386...|
|    0|       0.0|[0.87647319860020...|
|    0|       0.0|[0.84784057070522...|
|    0|       0.0|[0.84458517609688...|
|    0|       0.0|[0.85207567418292...|
|    0|       0.0|[0.92398442044881...|
|    0|       0.0|[0.90448400359170...|
|    0|       0.0|[0.87781159692731...|
|    0|       0.0|[0.94079072133711...|
|    0|       0.0|[0.91706791347850...|
|    0|       0.0|[0.92113463540154...|
|    0|       0.0|[0.89080069370671...|
|    0|       0.0|[0.89626220335811...|
|    0|       0.0|[0.82641452697078...|
|    0|       0.0|[0.82669958196628...|
|    0|       0.0|[0.93789818541283...|
|    0|       0.0|[0.83086156635377...|
|    0|       0.0|[0.86405382203406...|
|    0|       0.0|[0.90861093261906...|
+-----+----------+--------------------+
only showing top 20 rows
```