Background

You are provided with a sample dataset of a telecom company's customers and it's expected to done the following tasks:

- Perform exploratory analysis and extract insights from the dataset.
- Split the dataset into train/test sets and explain your reasoning.
- Build a predictive model to predict which customers are going to churn and discuss the reason why you choose a particular algorithm.
- Establish metrics to evaluate model performance.
- Discuss the potential issues with deploying the model into production.

Data Description

The customer churn data is given in the file sony_churn.csv. The detailed explanation is as follows:

Column Name	Column Type	Column Description
State	String	The state where a customer comes from
Account length	Integer	Number of days a customer has been using services
Area code	Integer	The area where a customer comes from
Phone number	Alphanumeric	The phone number of a customer
International plan	String	The status of customer international plan
Voicemail plan	String	The status of customer voicemail plan
No. vmail msgs	Integer	Number of voicemail message sent by a customer
Total day minutes	Float	Total call minutes spent by a customer during day time
Total day calls	Integer	Total number of calls made by a customer during day time
Total day charge	Float	Total amount charged to a customer during day time
Total eve minutes	Float	Total call minutes spent by a customer during evening time
Total eve calls	Integer	Total number of calls made by a customer during evening time
Total eve charge	Float	Total amount charged to a customer during evening time
Total night minutes	Float	Total call minutes spent by a customer during night time

Column Name	Column Type	Column Description
Total night calls	Integer	Total number of calls made by a customer during night time
Total night charge	Float	Total amount charged to a customer during night time
Total intl minutes	Float	Total international call minutes spent by a customer
Total intl calls	Integer	Total number of international calls made by a customer
Total int charge	Float	Total international call amount charged to a customer
Customer service calls	Integer	Total number of customer service calls made by a customer
Churn	Boolean	Whether a customer is churned or not

Exploratory Data Analysis (EDA)

In [1]:
```
import pandas as pd
import numpy as np
# set random seed to have reproducible results
# sklearn uses numpy random seed
np.random.seed(42)
```

In [8]:
```
import warnings
warnings.filterwarnings("ignore") # suppress warnings for readability
```

In [2]:
```
#read dataset
df = pd.read_csv("../data/sony_churn.csv")
df.head()
```

Out[2]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	t… cha…
0	KS	128	415	382-4657	no	yes	25	265.1	110	4…
1	OH	107	415	371-7191	no	yes	26	161.6	123	2…
2	NJ	137	415	358-1921	no	no	0	243.4	114	4…
3	OH	84	408	375-9999	yes	no	0	299.4	71	5…
4	OK	75	415	330-6626	yes	no	0	166.7	113	2…

5 rows × 21 columns

In [3]:
```python
# check fundamentals
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
```

In [4]:
```python
# see if every row is unique to one customer
df["phone number"].nunique()
```

Out[4]: 3333

In [5]:
```python
# check other uniques
df["area code"].nunique()
```

Out[5]: 3

We will prefer to leave state values out of the dataset in order to not have issues with high dimensionality. We can start to process other categorical features.

In [6]:
```python
df["state"].nunique()
```

Out[6]: 51

In [7]:
```python
area_code_dummies = pd.get_dummies(df["area code"])
area_code_dummies = area_code_dummies.add_prefix('area_code_')
area_code_dummies
```

Out[7]:

	area_code_408	area_code_415	area_code_510
0	False	True	False
1	False	True	False
2	False	True	False
3	True	False	False
4	False	True	False
...	...	...	...
3328	False	True	False
3329	False	True	False
3330	False	False	True
3331	False	False	True
3332	False	True	False

3333 rows × 3 columns

In [9]:

```
df["voice mail plan"].loc[df["voice mail plan"] == "no"] = 0
df["voice mail plan"].loc[df["voice mail plan"] == "yes"] = 1
df["voice mail plan"] = df["voice mail plan"].astype("int64")
df["voice mail plan"]
```

Out[9]:

```
0       1
1       1
2       0
3       0
4       0
       ..
3328    1
3329    0
3330    0
3331    0
3332    1
Name: voice mail plan, Length: 3333, dtype: int64
```

In [10]:

```
df["international plan"].loc[df["international plan"] == "no"] = 0
df["international plan"].loc[df["international plan"] == "yes"] = 1
df["international plan"] = df["international plan"].astype("int64")
df["international plan"]
```

Out[10]:
```
0       0
1       0
2       0
3       1
4       1
       ..
3328    0
3329    0
3330    0
3331    1
3332    0
Name: international plan, Length: 3333, dtype: int64
```

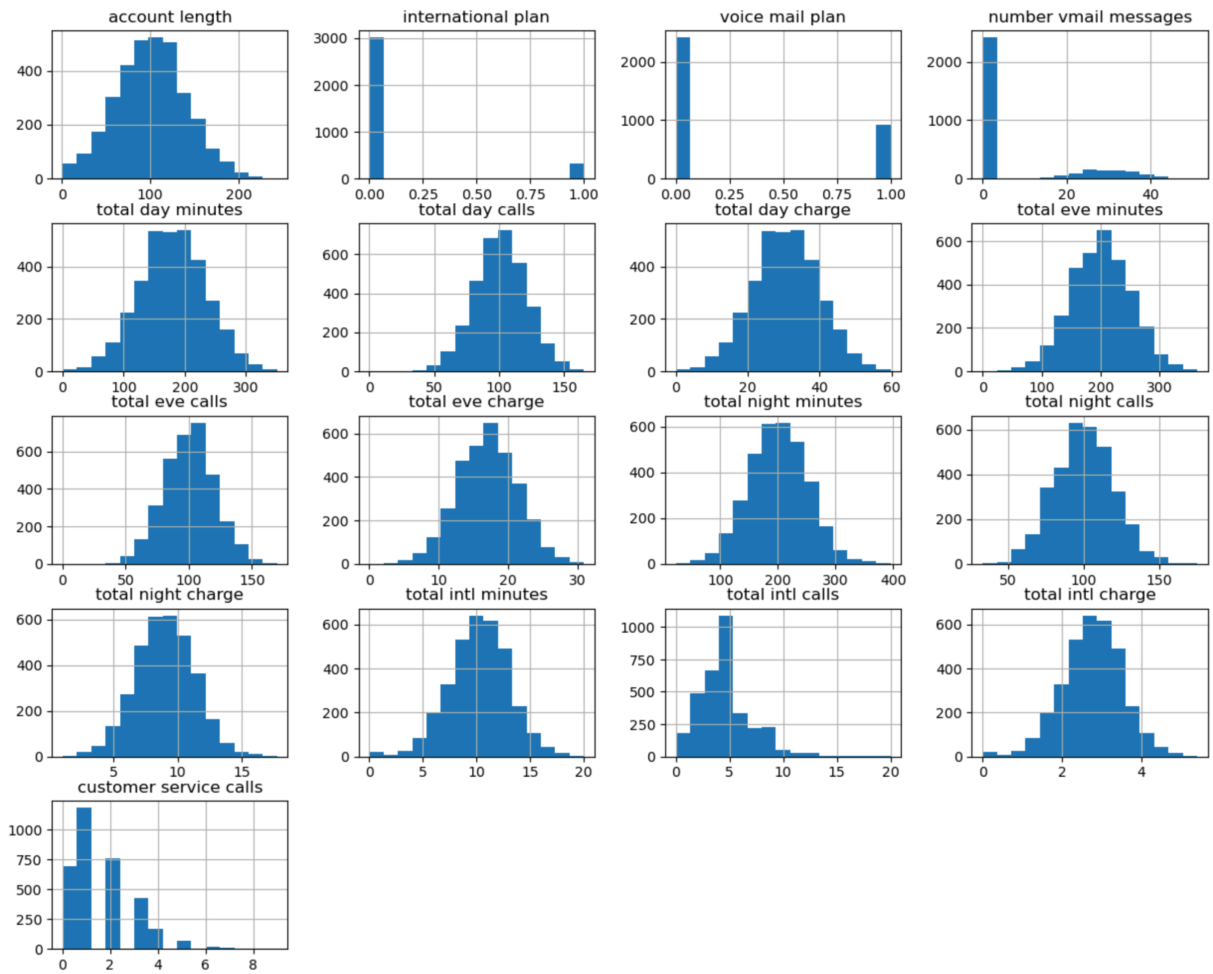
In [11]:
```
# form final dataset
df_final = df.drop(columns=["phone number", "state", "area code"])
df_final = pd.concat([df_final,area_code_dummies], axis=1)
df_final
```

Out[11]:

	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	tota eve calls
0	128	0	1	25	265.1	110	45.07	197.4	99
1	107	0	1	26	161.6	123	27.47	195.5	103
2	137	0	0	0	243.4	114	41.38	121.2	110
3	84	1	0	0	299.4	71	50.90	61.9	88
4	75	1	0	0	166.7	113	28.34	148.3	122
...	...	...	...	...	...	...	...	...	..
3328	192	0	1	36	156.2	77	26.55	215.5	126
3329	68	0	0	0	231.1	57	39.29	153.4	55
3330	28	0	0	0	180.8	109	30.74	288.8	58
3331	184	1	0	0	213.8	105	36.35	159.6	84
3332	74	0	1	25	234.4	113	39.85	265.9	82

3333 rows × 21 columns

In [12]:
```
import matplotlib.pyplot as plt

# check distribution of values
df_final.hist(figsize=(15,12),bins = 15)
plt.show()
```



In [13]:
```
# check classes ratio
df_final.groupby(['churn'])['churn'].count()
```

Out[13]:
```
churn
False    2850
True      483
Name: churn, dtype: int64
```

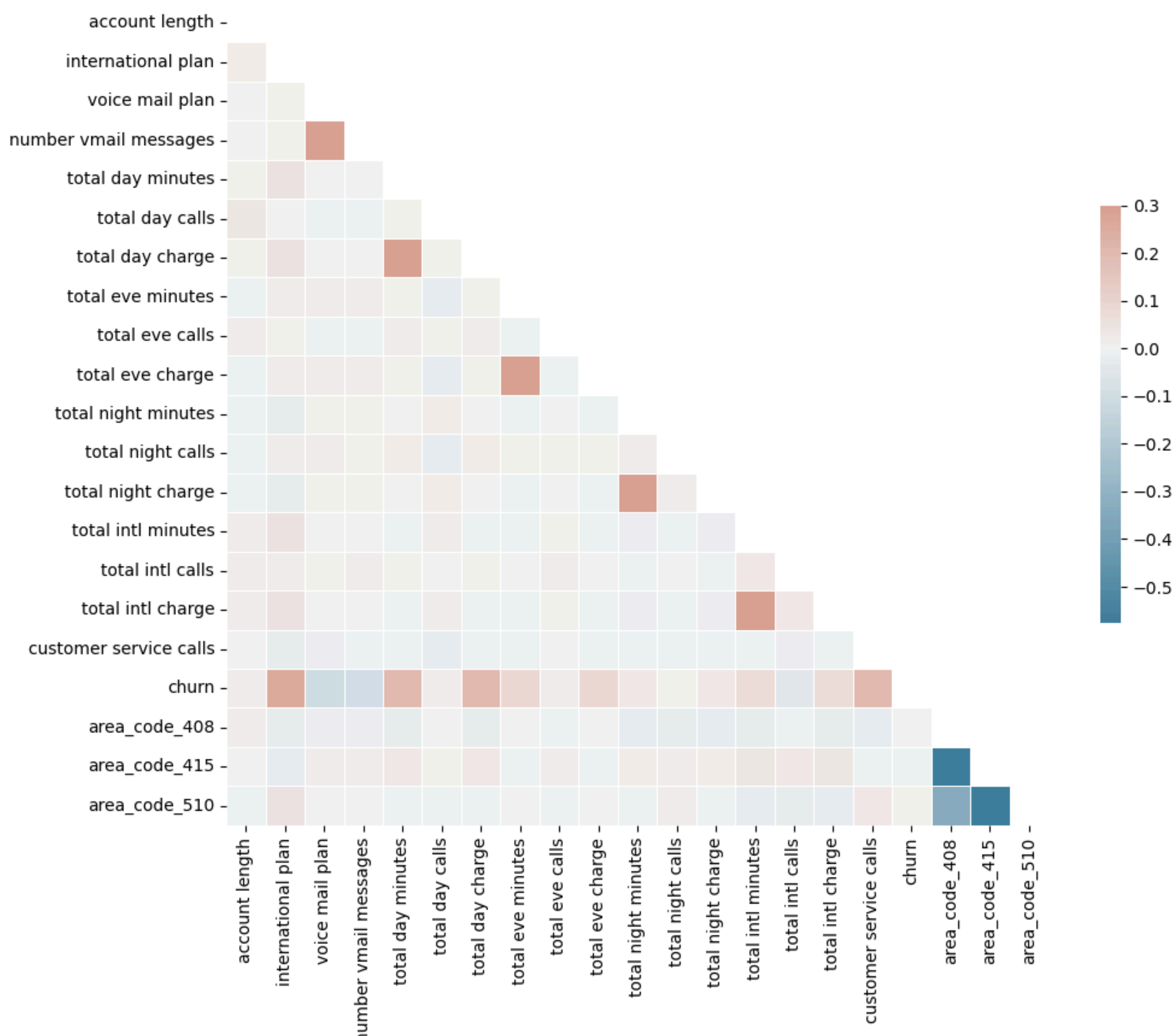
The distributions tell us:

- Most customers don't use voice mail service and international plans.
- Half of the customers live in area code 415.
- The company earns more by total day calls (check total day charge).
- We have an imbalanced dataset which could be tricky when choosing evaluation metrics.

In [14]:
```
# some insights into the relationship between features
# observe the correlation.

import matplotlib.pyplot as plt
import seaborn as sns

# it could take some time to run this cell since we are calculating correlat
# to have a better visualization, we will take only one triangle
```

```python
# because other trangle is only its symmetry (i.e a x b and b x a)

# Generate a mask for the upper triangle
corr = df_final.corr()
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

```
Out[14]: <Axes: >
```



From the correlation matrix, we observe the following things:

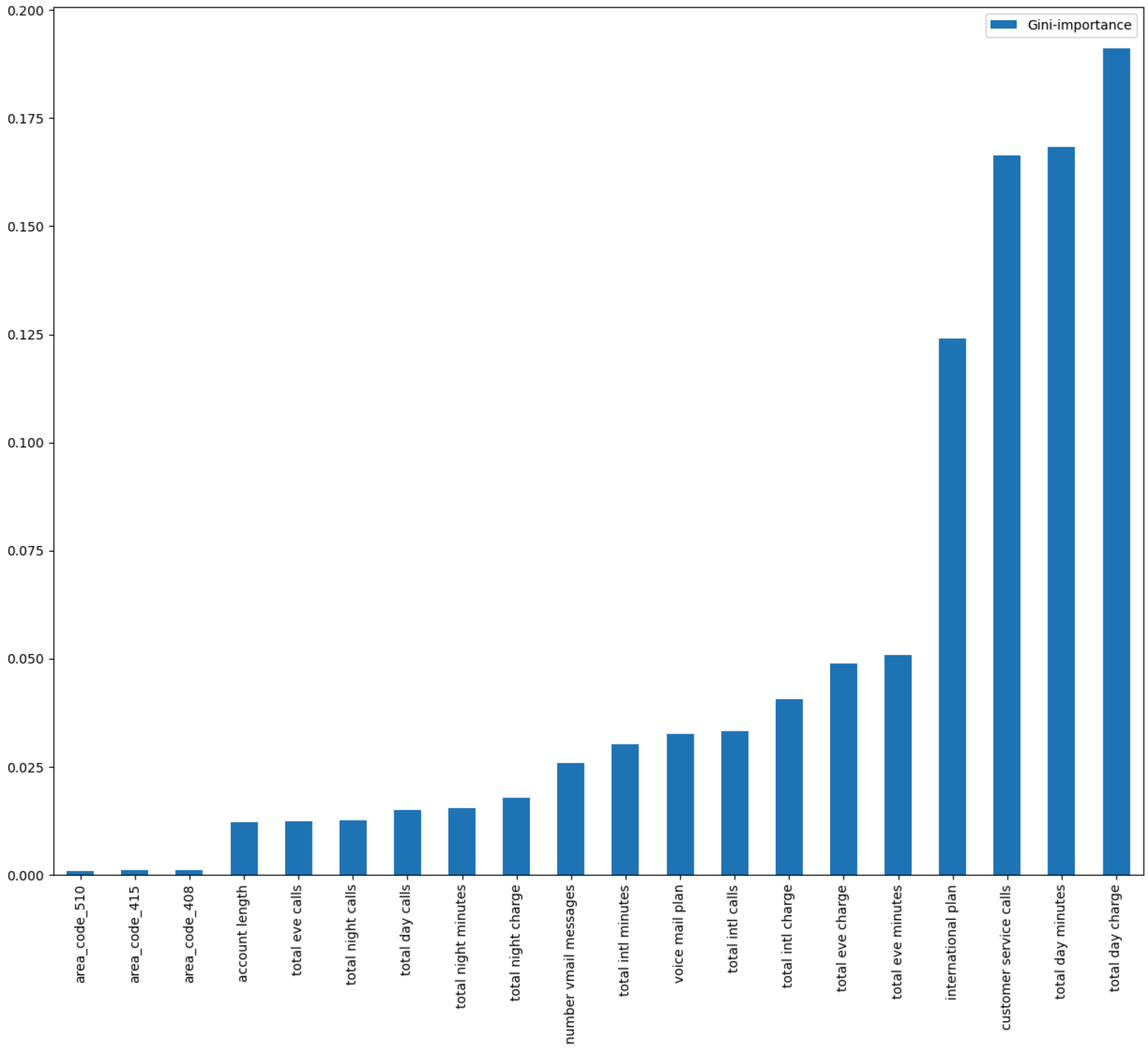
- There is a positive correlation between:
  - total day charge, total day minutes, and churn

- total eve minutes and total eve charge
- total night minutes and total night charge
- total intl minutes and total intl charge
- total customer service calls and churn
- number vmail messages and voice mail
- international plan and churn

- There is a negative correlation between:

    - churn and voice mail plan
    - churn and number vmail messages
    - churn and total intl calls

In [15]:
```
"""check feature importances via random forest classifier"""

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

le = preprocessing.LabelEncoder()
# apply label encoder for churn since its values are also categories
y = le.fit_transform(df_final["churn"])

# drop label column
X = df_final.drop(columns=["churn"])

# train-test split
X = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# selected features are selected in multicollinearity check part
feature_names = [f"feature {i}" for i in range((X.shape[1]))]
forest =  RandomForestClassifier(max_depth=5)
forest.fit(X_train, y_train)
feats = {} # a dict to hold feature_name: feature_importance
for feature, importance in zip(df_final.drop(columns=["churn"]).columns, for
    feats[feature] = importance #add the name/value pair

importances = pd.DataFrame.from_dict(feats, orient='index').rename(columns={
importances.sort_values(by='Gini-importance').plot(kind='bar', rot=90, figsi
plt.show()
```



Gini-importance shows us which features would be most useful if we build a tree-based model with given features. According to the analysis above, the most important three features of churn are: total day charge, total day minutes, and customer service calls.

Train/Test Split

In this notebook, we will mostly apply machine learning methods for the given problem. Therefore, we will prefer to use an 80%-20% split since it is used as the most common ratio in applications (not including Deep Learning). Furthermore, we have an imbalanced dataset in terms of class distributions. We can use stratify option of train_test_split( ) function of sklearn to split data to train and test datasets with the same distribution and be sure that samples of the test or train dataset are not only formed by the majority class.

In [17]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
# apply label encoder for churn since its values are also categories
```

```
y = le.fit_transform(df_final["churn"])
X = df_final.drop(columns=["churn"])
```

In [18]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
```

Predictive Model

In [19]:
```
# Apply classifiers and decide to pick one to use in production based on the
# Hyperparameters of the given classifiers are chosen as trial-error

from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

names = [
    "Nearest Neighbors",
    "Linear SVM",
    "RBF SVM",
    "Gaussian Process",
    "Decision Tree",
    "Random Forest",
    "Neural Net",
    "AdaBoost",
    "Naive Bayes",
    "QDA",
    "XGBoost",
    "LightGBM"
]

classifiers = [

    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0), random_state=42),
    DecisionTreeClassifier(max_depth=5, random_state=42),
    RandomForestClassifier(max_depth=5, random_state=42),
    MLPClassifier(alpha=1, max_iter=1000, random_state=42),
    AdaBoostClassifier(random_state=42),
    GaussianNB(),
    QuadraticDiscriminantAnalysis(),
    XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', seed=0),
```

```
    LGBMClassifier(random_state=42),
]
```

Metrics

This is a classification task, and the most commonly used metric is accuracy. But, we have an imbalanced dataset, which means we need to be careful about our evaluations. F1 score balances the precision and recall so we can have a good metric even for imbalanced datasets. Hence, we will use accuracy and the F1 score while comparing the performance of different algorithms.

In [20]:
```
from sklearn.metrics import f1_score
```

Model Results

Classical Machine Learning Models

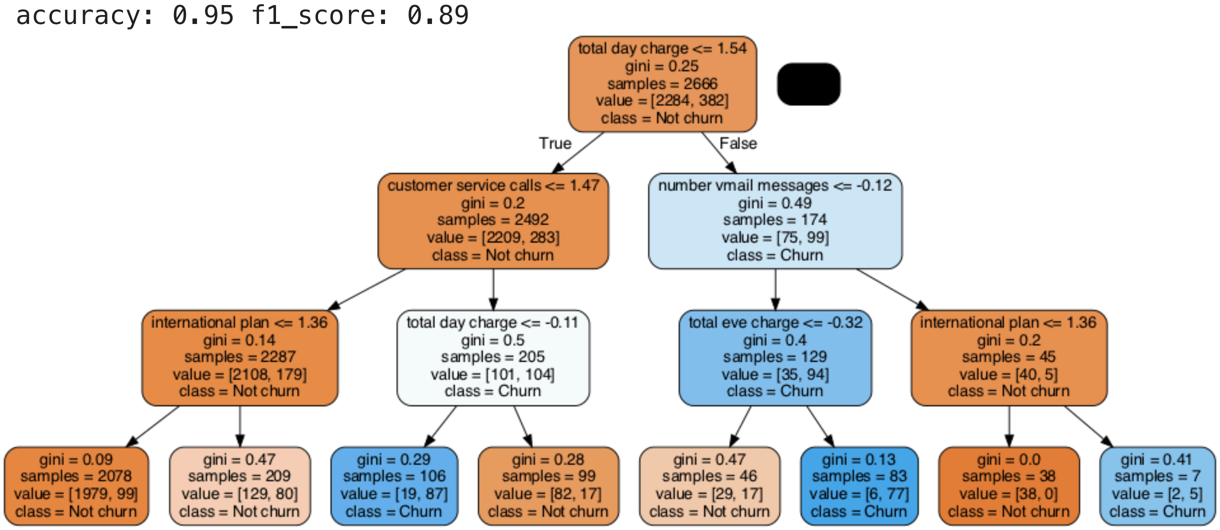
In [21]:
```
for name, clf in zip(names, classifiers):
        clf.fit(X_train, y_train)
        acc_score = clf.score(X_test, y_test)
        y_pred = clf.predict(X_test)
        f_score = f1_score(y_test, y_pred, average='macro')
        print("accuracy:", "{:.2f}".format(acc_score), "f1_score:", "{:.2f}"
```

```
accuracy: 0.89 f1_score: 0.72 Model: Nearest Neighbors
accuracy: 0.85 f1_score: 0.46 Model: Linear SVM
accuracy: 0.85 f1_score: 0.46 Model: RBF SVM
accuracy: 0.93 f1_score: 0.85 Model: Gaussian Process
accuracy: 0.94 f1_score: 0.86 Model: Decision Tree
accuracy: 0.90 f1_score: 0.73 Model: Random Forest
accuracy: 0.93 f1_score: 0.85 Model: Neural Net
accuracy: 0.88 f1_score: 0.70 Model: AdaBoost
accuracy: 0.85 f1_score: 0.70 Model: Naive Bayes
accuracy: 0.87 f1_score: 0.78 Model: QDA
accuracy: 0.96 f1_score: 0.92 Model: XGBoost
[LightGBM] [Info] Number of positive: 382, number of negative: 2284
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of te
sting was 0.000416 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2401
[LightGBM] [Info] Number of data points in the train set: 2666, number of us
ed features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.143286 -> initscore=-1.788
263
[LightGBM] [Info] Start training from score -1.788263
accuracy: 0.95 f1_score: 0.89 Model: LightGBM
```

In [24]:
```
"""visualize the Decision Tree and see how tree-based algorithms decide for

from sklearn import tree
```

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from IPython.display import SVG,display, Image
import pydotplus

#Function attributes
#maximum_depth  - depth of tree
#criterion_type - ["gini" or "entropy"]
#split_type     - ["best" or "random"]

def plot_decision_tree(maximum_depth,criterion_type,split_type) :


    #model
    clf = DecisionTreeClassifier(max_depth=3)
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    print("accuracy:", "{:.2f}".format(acc_score), "f1_score:", "{:.2f}".for

    #plot decision tree
    graph = tree.export_graphviz(clf,out_file=None,
                                    rounded=True,proportion = False,
                                    feature_names = df_final.drop(columr
                                    precision  = 2,
                                    class_names=["Not churn","Churn"],
                                    filled = True,

                    )

    pydot_graph = pydotplus.graph_from_dot_data(graph)
    pydot_graph.set_size('"10,10"')
    plt = Image(pydot_graph.create_png())
    display(plt)

plot_decision_tree(3,"gini","best")
```

```
accuracy: 0.95 f1_score: 0.89
```



Deployment Issues

After the deployment of our ML model to production, we need to continue to monitor its performance since it could degrade over time due to internal or external reasons. It is recommended to update our models periodically, such as training with recent data to avoid common problems. There are two significant problems with the MLOps cycle:

- Data drift: Data drift is the situation where the model’s input distribution changes. It could be caused by broken data ingestion or serving pipeline, or a change in the nature of your problem. We can resolve this issue by fixing the broken data engineering pipelines where applicable or by training our model with more data including more recent data points if there is no deterioration in the data quality.
- Concept drift: Concept drift is the situation when the functional relationship between the model inputs and outputs changes. The context has changed, but the model doesn’t know about the change. Its learned patterns do not hold anymore. Hence, we need to learn a new model and even use another algorithm if our particular algorithm's performance is not good enough to use in production.