Customer Churn Analysis

Overview

Customer churn is the percentage of customers who stopped purchasing your business's products or services during a certain period of time. Businesses typically treat a customer as churned once a particular amount of time has elapsed since the customer's last interaction with the site or service. The full cost of customer churn includes both lost revenue and the marketing costs involved with replacing those customers with new ones. Reduction customer churn is important because cost of acquiring a new customer is higher than retaining an existing one. Reducing customer churn is a key business goal of every business. This case is related to telecom industry where particular organizations want to know that for given certain parameters whether a person will churn or not.

Business Understanding

Syriatel is telecommunications company that has concerns regarding churn rate.Using customer account data,we will analyze what features from the data are most important in prediciting customer churn, or whether or not they will leave the company. In order to do this the notebook will provide classification models aimed at producing the highest possible recall metric.Predicting as many positives as possible out of actual positives from dataset is the goal here, thus recall has been chosen as one of the performance matrices along with an accuracy score

Importing Libraries

In [1]:
```
# import the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV, cross_v
from sklearn.preprocessing import StandardScaler,OneHotEncoder,LabelEncoder

from sklearn.metrics import confusion_matrix, plot_confusion_matrix, classi
from sklearn.metrics import accuracy_score, recall_score, f1_score, roc_auc

from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier, plot_importance

import warnings
warnings.filterwarnings('ignore')

# Plotting pretty figures and avoid blurry images
%config InlineBackend.figure_format = 'retina'
# Larger scale for plots in notebooks
sns.set_context('notebook')
```

Data Understanding

In [2]:
```
df = pd.read_csv('data/data.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	total day charge	...	total eve calls
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122

5 rows × 21 columns

Data Description

The dataset contains relevant customer account data including:

`state` : The state the customer is from

`account lenght` : The account length of the customer

`area code` : the customer's area code

`international plan` : Whether or not the customer has an international plan

`voice mail plan` : Whether or not the customer has a voicemail plan

The total day, evening, night, and international minutes of the customer

The total day, evening, night, and international calls of the customer

The total day, evening, night, and international charge of the customer

`customer service call` : The total customer service calls of the customer

`churn` : Whether or not the customer was 'true' or 'false' churn (true churn meaning that they have left the company, this will be the primary focus of the models)

In [3]:
```
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]:
```
df.describe().T
```

Out[4]:

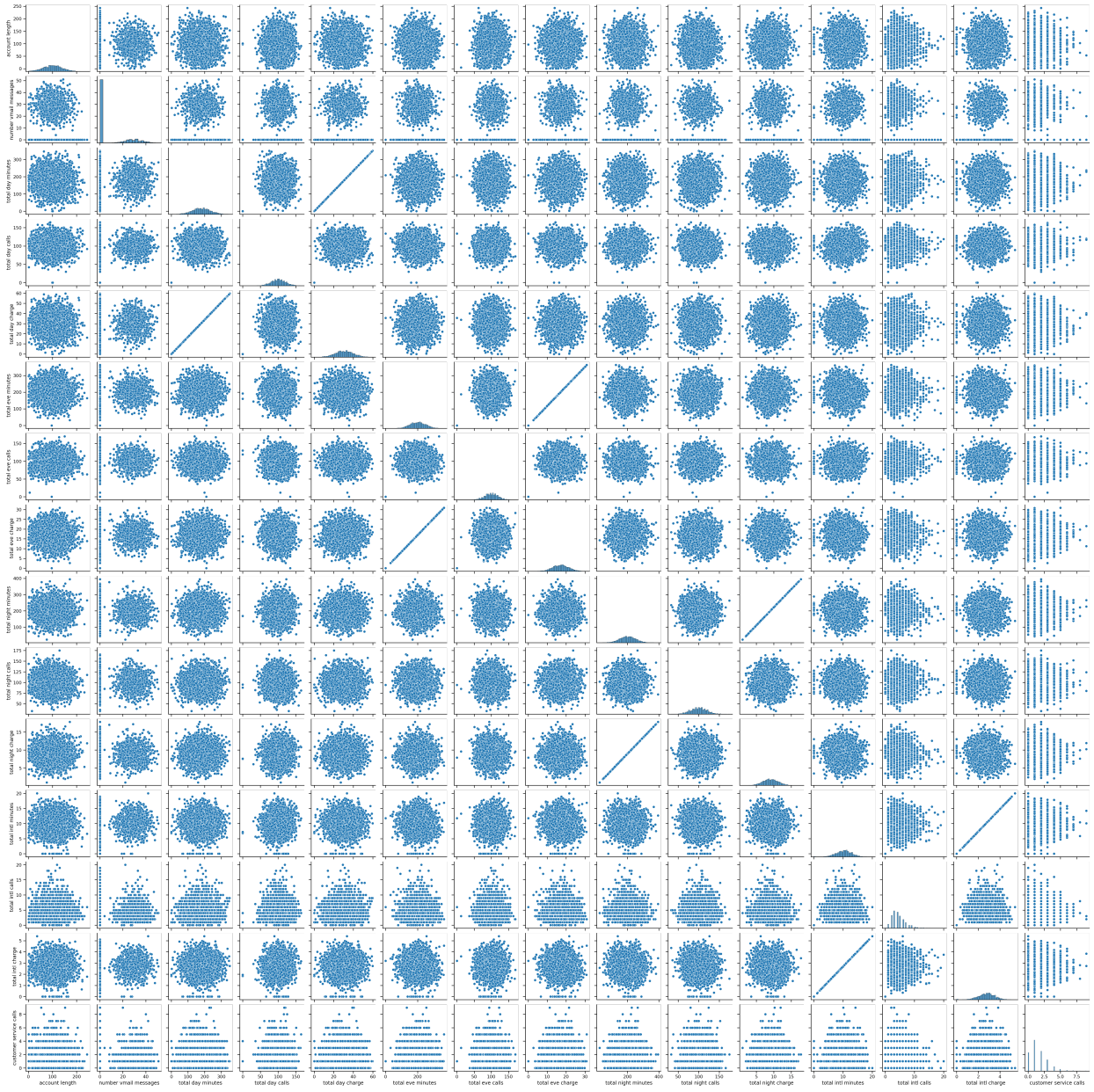
	count	mean	std	min	25%	50%	75%	max
account length	3333.0	101.064806	39.822106	1.00	74.00	101.00	127.00	243.00
area code	3333.0	437.182418	42.371290	408.00	408.00	415.00	510.00	510.00
number vmail messages	3333.0	8.099010	13.688365	0.00	0.00	0.00	20.00	51.00
total day minutes	3333.0	179.775098	54.467389	0.00	143.70	179.40	216.40	350.80
total day calls	3333.0	100.435644	20.069084	0.00	87.00	101.00	114.00	165.00
total day charge	3333.0	30.562307	9.259435	0.00	24.43	30.50	36.79	59.64
total eve minutes	3333.0	200.980348	50.713844	0.00	166.60	201.40	235.30	363.70
total eve calls	3333.0	100.114311	19.922625	0.00	87.00	100.00	114.00	170.00
total eve charge	3333.0	17.083540	4.310668	0.00	14.16	17.12	20.00	30.91
total night minutes	3333.0	200.872037	50.573847	23.20	167.00	201.20	235.30	395.00
total night calls	3333.0	100.107711	19.568609	33.00	87.00	100.00	113.00	175.00
total night charge	3333.0	9.039325	2.275873	1.04	7.52	9.05	10.59	17.77
total intl minutes	3333.0	10.237294	2.791840	0.00	8.50	10.30	12.10	20.00
total intl calls	3333.0	4.479448	2.461214	0.00	3.00	4.00	6.00	20.00
total intl charge	3333.0	2.764581	0.753773	0.00	2.30	2.78	3.27	5.40
customer service calls	3333.0	1.562856	1.315491	0.00	1.00	1.00	2.00	9.00

Data Exploration

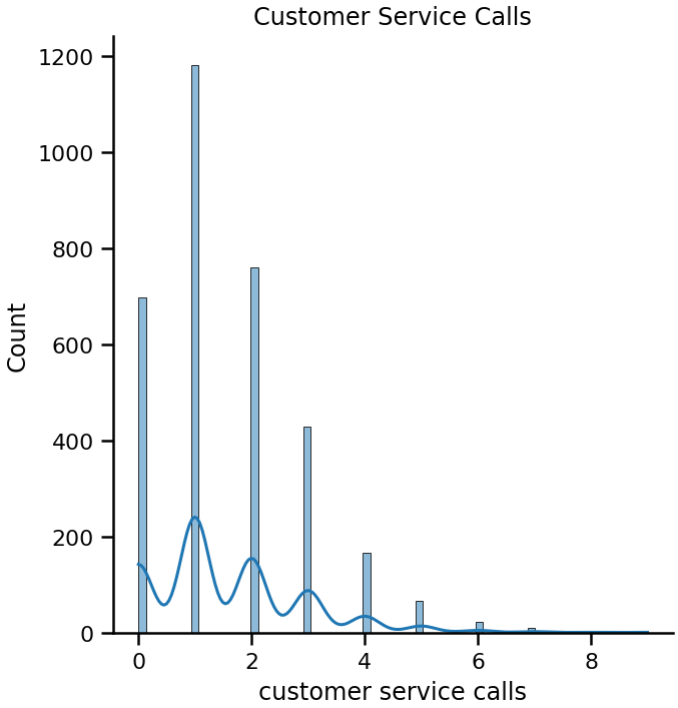
In [5]:
```
# I will drop irrevalent columns
df = df.drop(['phone number' , 'area code'] , axis=1)
```

In [6]:
```
# df_cont1 = df.drop(columns=['state', 'international plan', 'voice mail pl
# All contunious variables
df_cont = df.select_dtypes(include=[np.number])
# All categorical variables
df_cat = df.select_dtypes(exclude=[np.number])
df_cat = df_cat.drop('churn' , axis=1)
```

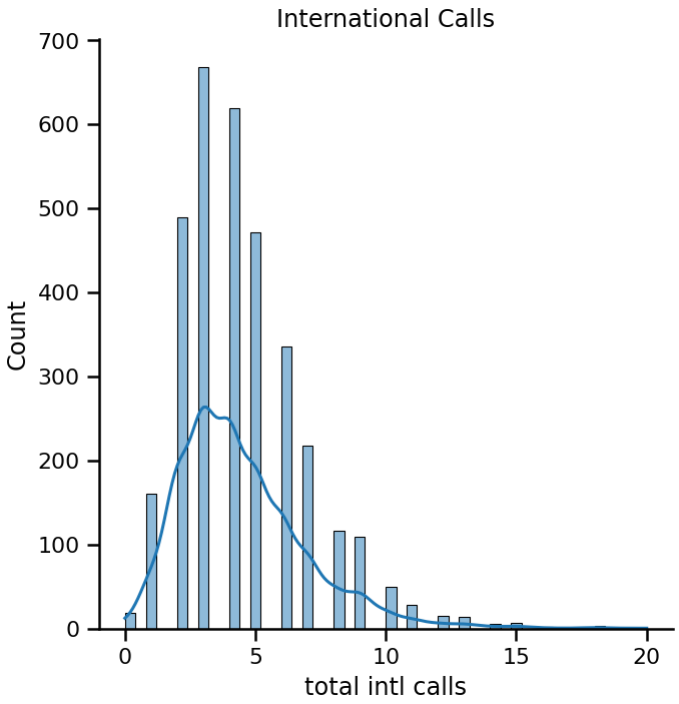
In [7]:
```
sns.pairplot(df_cont);
```



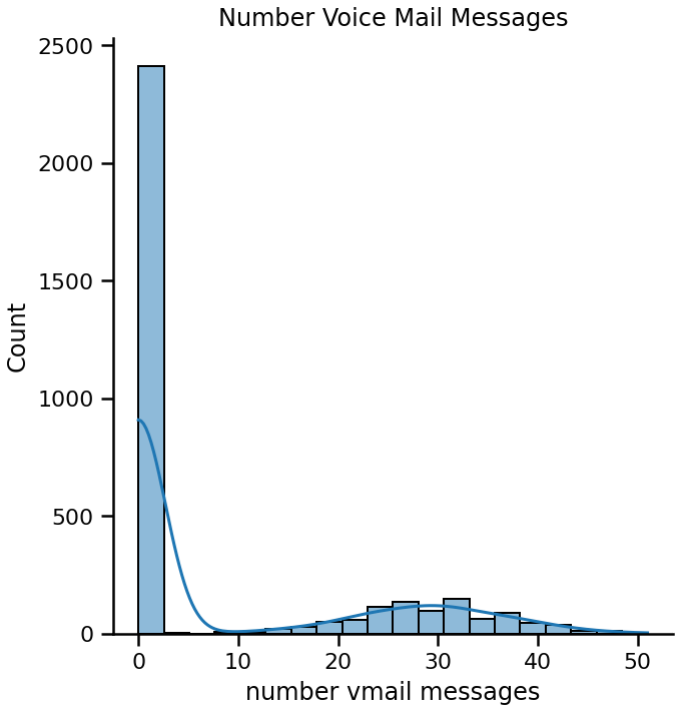
In [8]:
```
# Looking at the features that dont have normal distrubiton
sns.displot(x=df['customer service calls'],kind='hist',kde=True).set(title=
```



In [9]:
```
# Looking at the features that dont have normal distrubiton
sns.displot(x=df['total intl calls'],kind='hist',kde=True).set(title='Inter
```



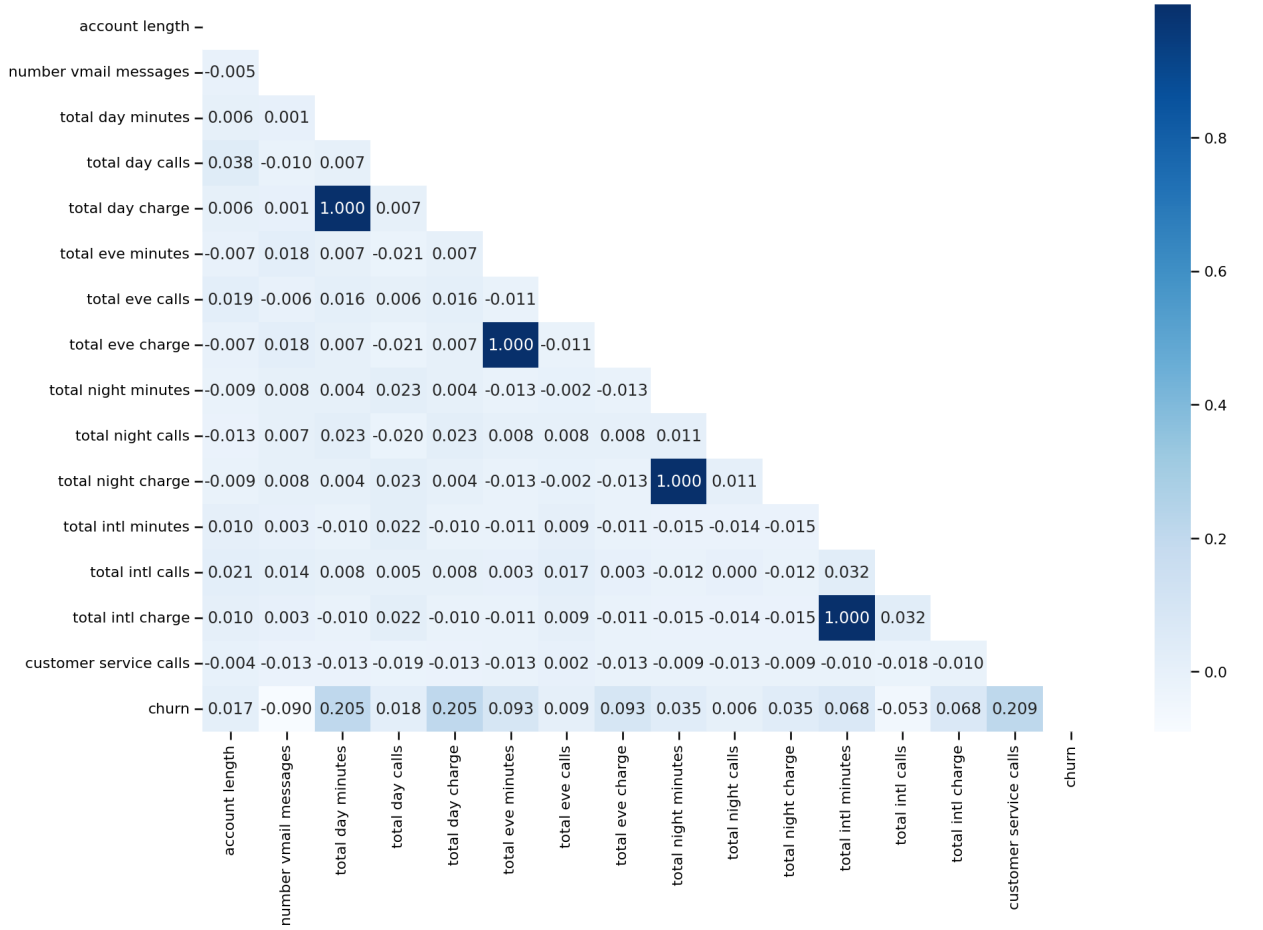
In [10]:
```
# Looking at the features that dont have normal distrubiton
sns.displot(x=df['number vmail messages'],kind='hist',kde=True).set(title='
```



- All feature columns have a normal distrubution except for `customer service calls`, `total intl calls`, `number vmail messages`.
- Features with a 45 degree angle above graphs (`total day charge` and `total day minutes` etc.) are positively correlated.It's sign of multicollinearity. I might remove one of the features further to get better dataset

- `number vmail messages` does not appear evenly distributed with a large portion of customers having 0 messages. This would make sense as many customers probably delete their voicemail messages and likely would not be a predicitor of churn.
- `total intl calls` appears somewhat evenly distributed except for outliers over about 12. This would make sense as several customers are likely international customers making a lot of international calls. This likely would not have an effect on churn.
- `customer service calls` does not appear to be evenly distributed. This feature could be relevant at predicting churn as cutomers who are unhappy with their service would likely make more customer service calls.

In [11]:
```
plt.figure(figsize = (15,10));
mask = np.triu(df.corr());
sns.heatmap(df.corr(), cmap = "Blues",annot = True, fmt = '.3f',mask = mask
```



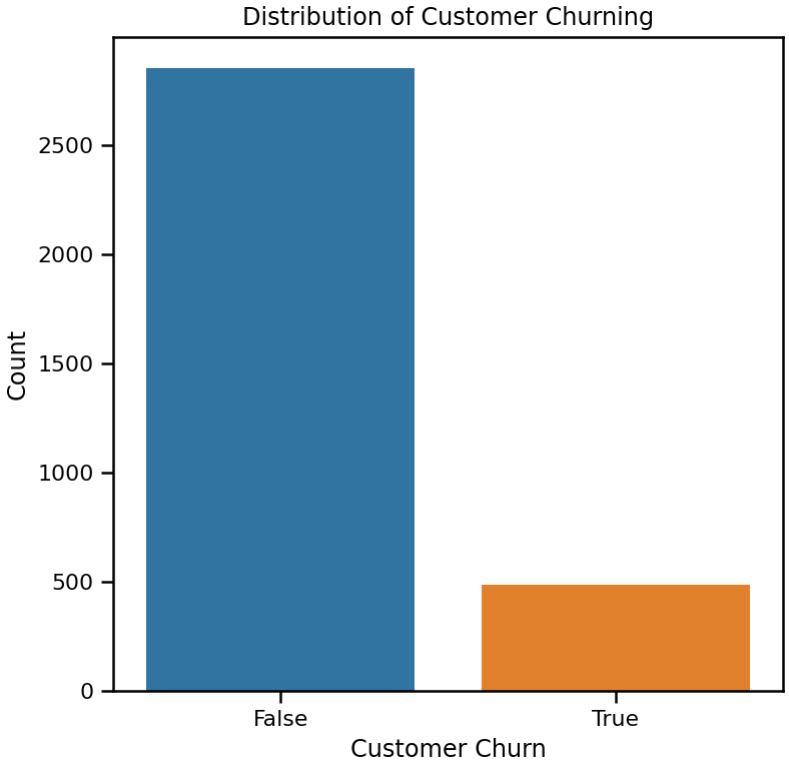
- `total day charge` and `total day minutes`
- `total eve charge` and `total eve minutes`
- `total night charge` and `total night minutes`
- `total intl charge` and `total intl minutes` are highly correlated.

Will need to drop one of these features when building models to prevent multicollinearity.

In [12]:
```
# Check the balance of target values
print(df['churn'].value_counts())
print()
print('Check percentages:')
print(df['churn'].value_counts(normalize = True))
```

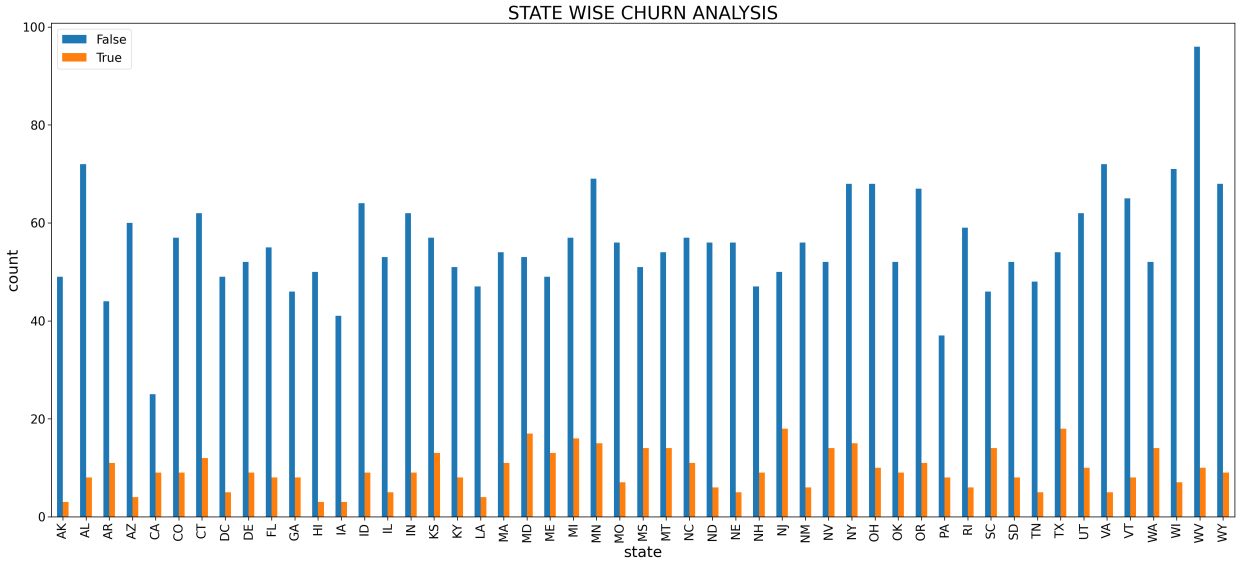
```
False    2850
True      483
Name: churn, dtype: int64

Check percentages:
False    0.855086
True     0.144914
Name: churn, dtype: float64
```

In [13]:
```
plt.figure(figsize=(6,6));
sns.countplot(x= df.churn);
plt.xlabel('Customer Churn');
plt.ylabel('Count');
plt.title('Distribution of Customer Churning');
plt.show()
```

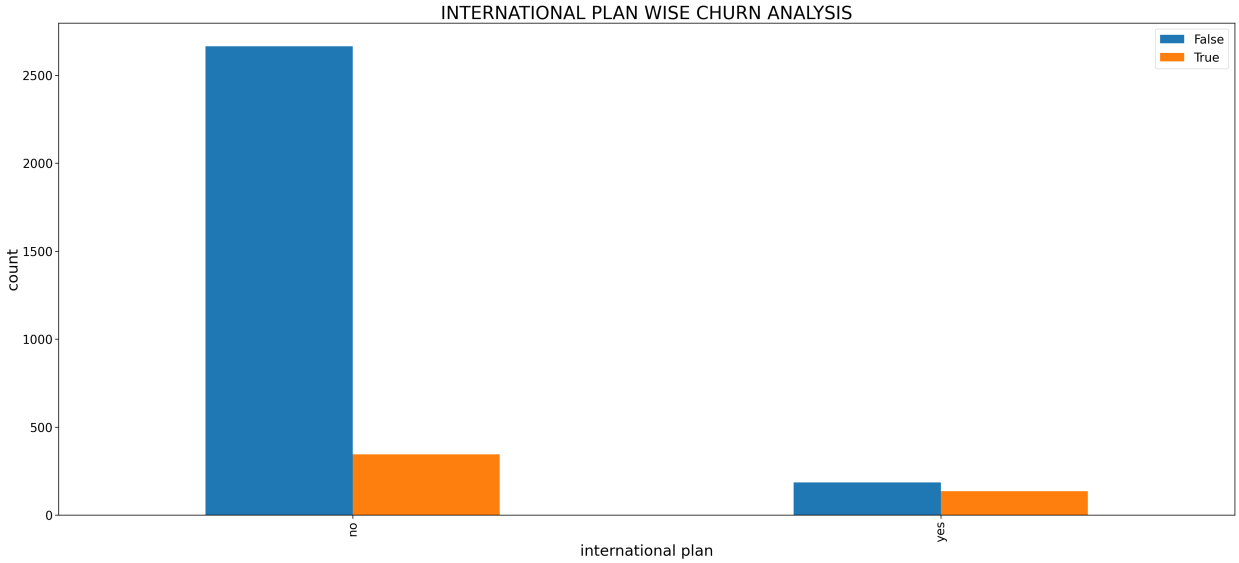


- There is imbalance , we will have to deal with during the modeling phase

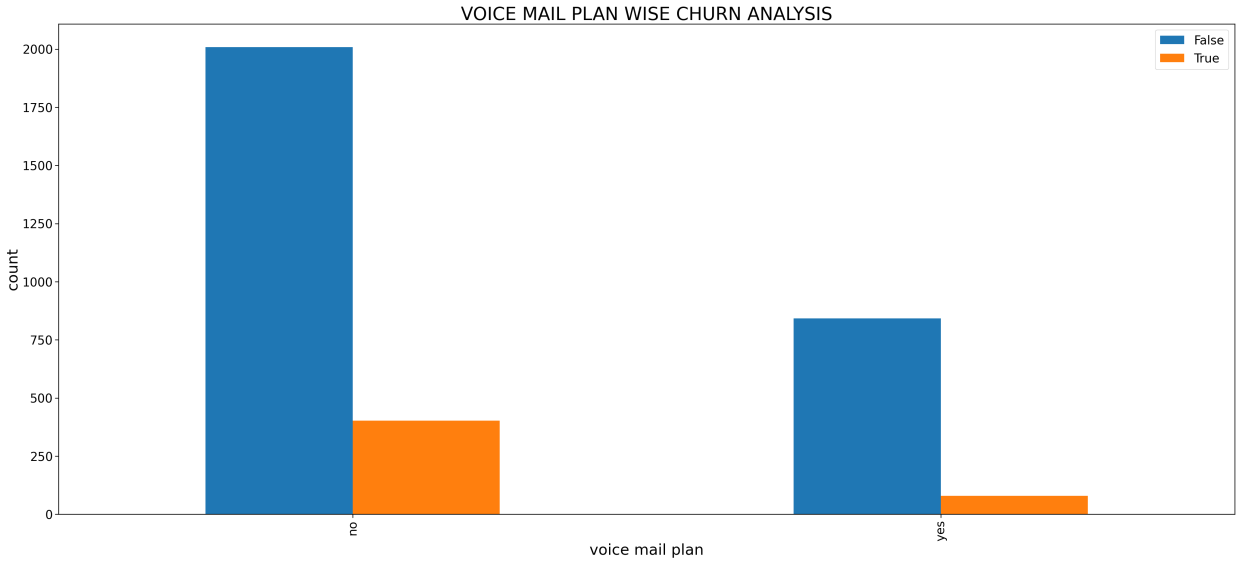
In [14]:
```
# relational bar graph for categorical variables
for c in df_cat.columns:
    df.groupby([c , 'churn']).size().unstack(level = -1).plot(kind = 'bar'
    plt.xlabel(c, fontsize = 25)
    plt.ylabel('count' ,  fontsize = 25)
    plt.xticks(fontsize = 20)
    plt.yticks(fontsize = 20)
    plt.title('{X} wise Churn Analysis'.format(X=c).upper() , fontsize = 30
    plt.legend(fontsize = 20)
    plt.show()
    plt.savefig('{X}.png'.format(X=c).upper())
```



<Figure size 432x288 with 0 Axes>



```
<Figure size 432x288 with 0 Axes>
```

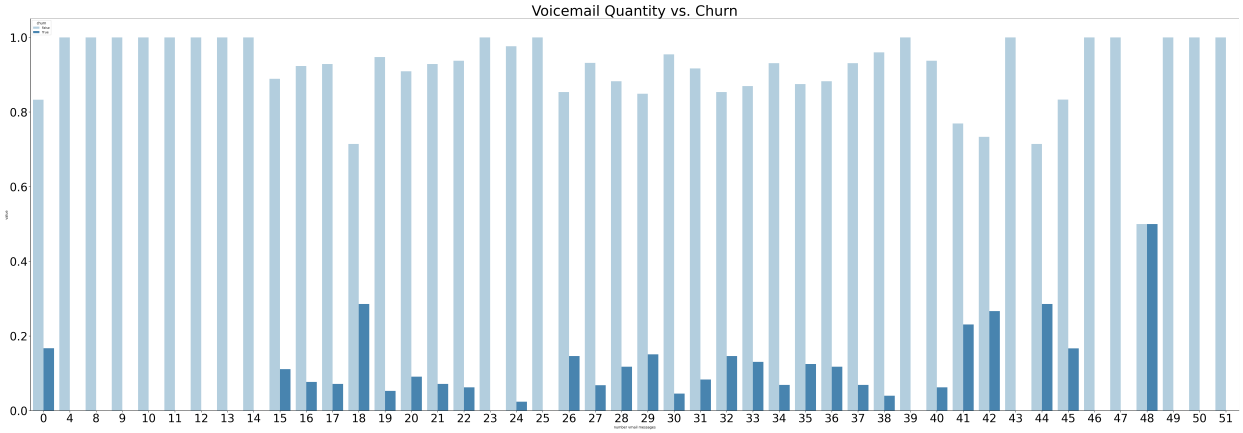



```
<Figure size 432x288 with 0 Axes>
```

- Maryland, New Jersey, and Texas have the highest churn rates. Alaska & Hawaii had the lowest churn rate. But there isn't a clear reason why certain states perform better than others
- `international plan` contains a similar number of 'yes's despite the much greater number of False-churns from the data. This could be an idication that customers with True-churn are more likely to have an international plan. This could mean that international is a good predictor of churn.
- `voice mail plan` appears to be similar across churn despite the unbalanced data, and is likely not a good predictor of churn.

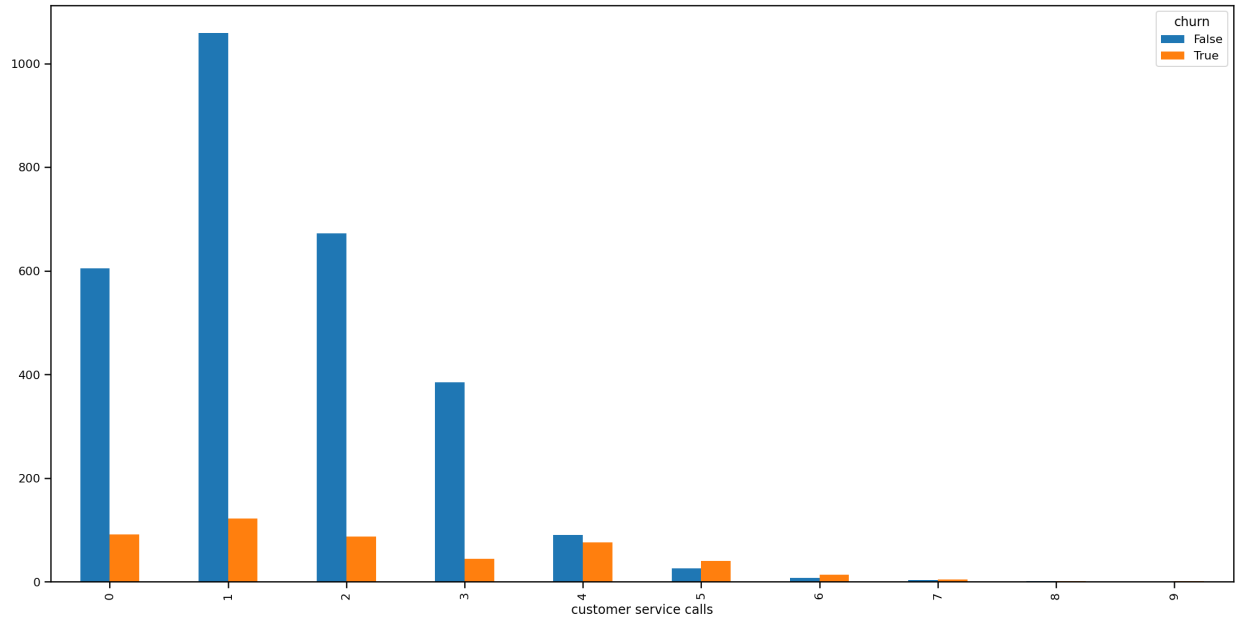
In [15]:
```
num_vmail = df.groupby('number vmail messages')['churn'].value_counts(norma
num_vmail = pd.DataFrame(num_vmail)
num_vmail.columns = ['value']
num_vmail = num_vmail.reset_index()
```

In [16]:
```
sns.catplot(data = num_vmail, kind = 'bar', x = 'number vmail messages', y
            hue = 'churn', palette = 'Blues', alpha=1, height=20, aspect=3
plt.title('Voicemail Quantity vs. Churn', fontsize = 50);
plt.xticks(fontsize = 40);
plt.yticks(fontsize = 40);
```



- Customers who got more voicemail messages tended to churn more then customers who got less, however, this relationship is not perfectly defined.

In [17]:
```
df.groupby(['customer service calls' , 'churn']).size().unstack(level = -1)
```



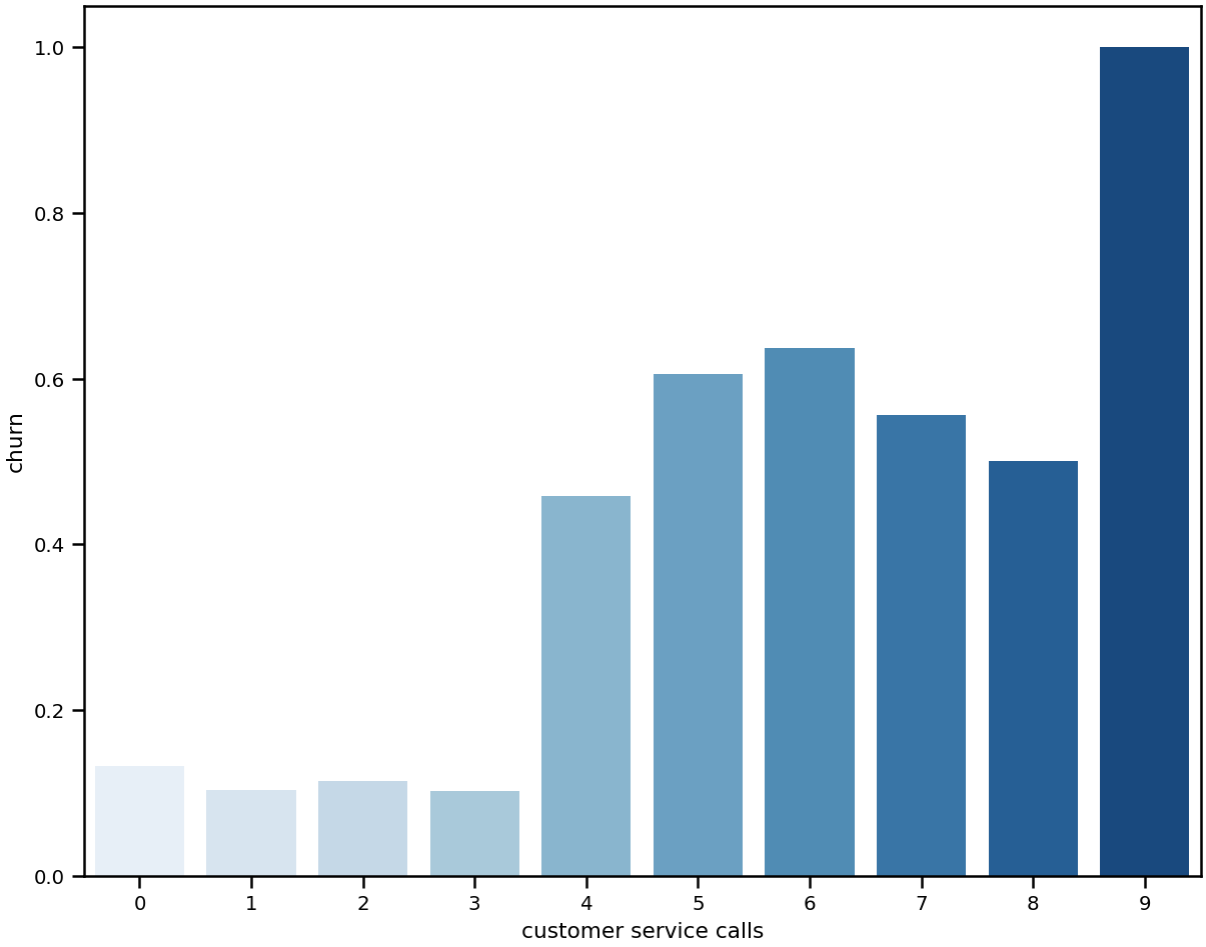
In [18]:
```
customer_service_calls =round(df.groupby('customer service calls')['churn']
customer_service_calls
```

Out[18]:

customer service calls	churn	
0	False	0.87
	True	0.13
1	False	0.90
	True	0.10
2	False	0.89
	True	0.11
3	False	0.90
	True	0.10
4	False	0.54
	True	0.46
5	True	0.61
	False	0.39
6	True	0.64
	False	0.36
7	True	0.56
	False	0.44
8	False	0.50
	True	0.50
9	True	1.00

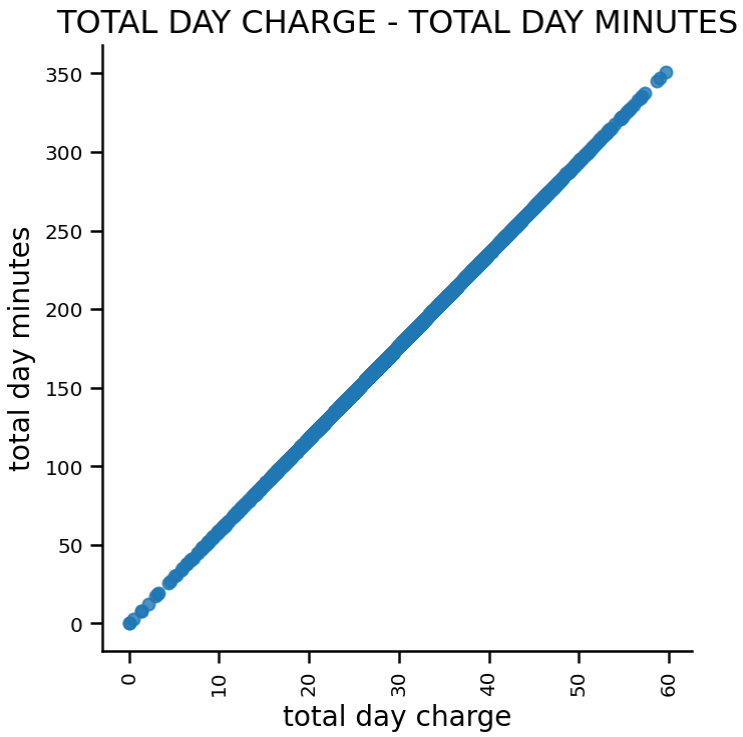
Name: churn, dtype: float64

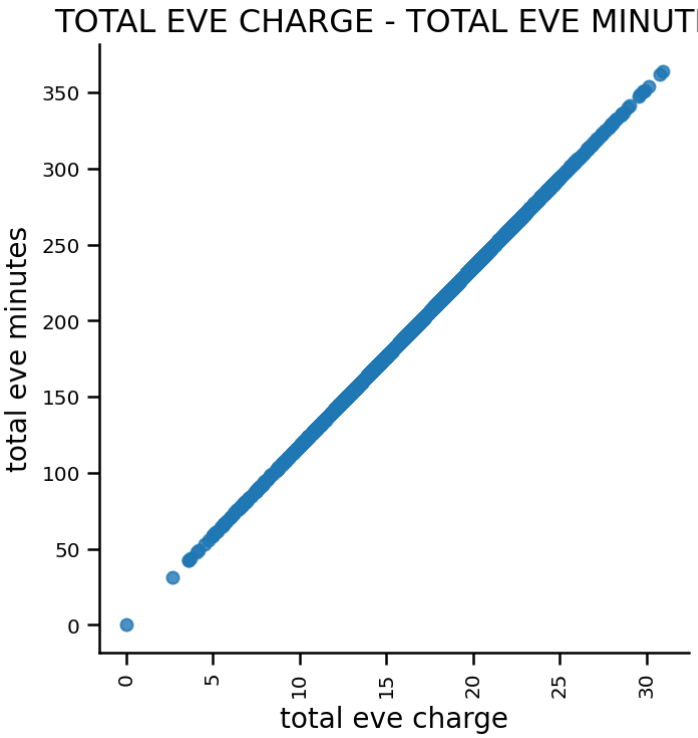
In [19]:
```
sns.set_context('notebook', font_scale = 0.9)
plt.figure(figsize=(10, 8))
sns.barplot(x='customer service calls', y='churn', data=df, palette = 'Blue
```

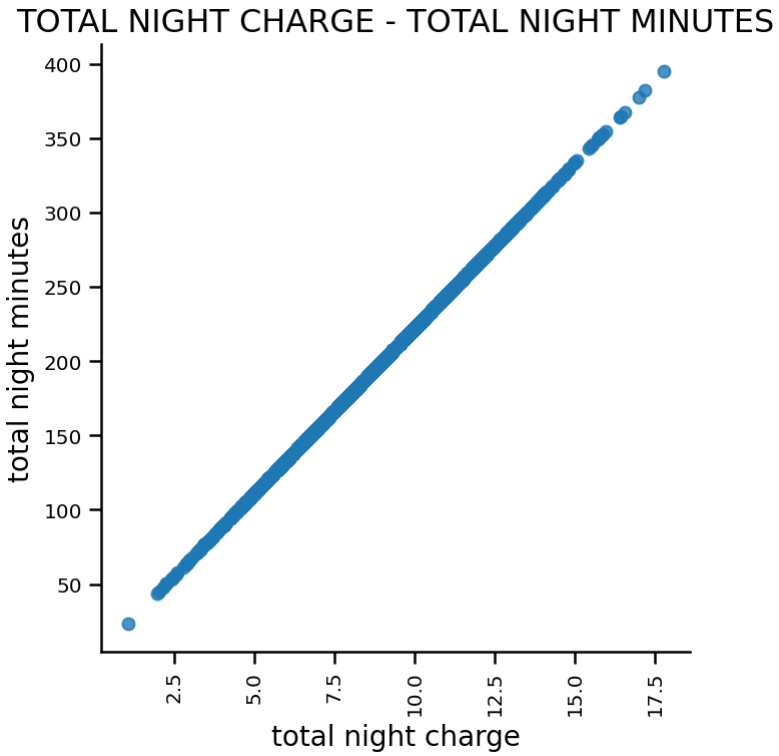


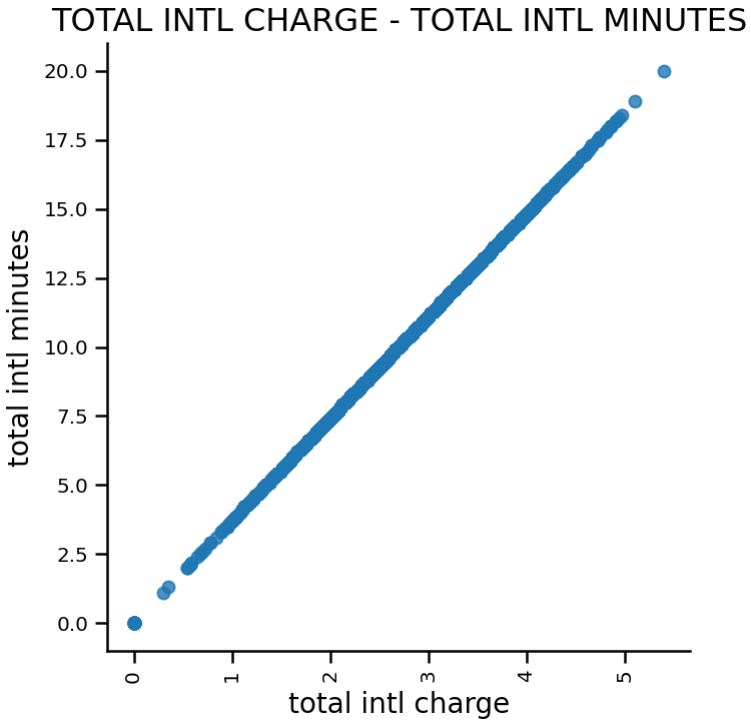
It seems like the greater the number of customer service calls there are the greater likelihood of churn. It looks like once people hit 3 calls, thats when there was a significant rise of customer churn. So 3 calls could be considered as a warning thresold for the company. We will definitely use this column in modeling as there seems to be a distinct relationship.

In [20]:
```
df_charge = df[['total day charge','total eve charge' , 'total night charge
df_min = df[['total day minutes', 'total eve minutes' , 'total night minute

# relational scatter graph for continuous variables
for c ,m  in zip(df_charge , df_min):
#     fig = plt.figure();
    fig = sns.lmplot(c,m, data=df,fit_reg=False);
    plt.xlabel(c,fontsize= 14);
    plt.ylabel(m,fontsize= 14);
    plt.xticks(fontsize=10, rotation=90);
    plt.yticks(fontsize=10);
    plt.title("{X} - {Y}".format(X=c,Y=m).upper(),fontsize = 16);
    plt.show();
```




TOTAL EVE CHARGE - TOTAL EVE MINUTES
total eve minutes
0
50
100
150
200
250
300
350
0
5
10
15
20
25
30
total eve charge


TOTAL NIGHT CHARGE - TOTAL NIGHT MINUTES
total night minutes
400
350
300
250
200
150
100
50
2.5
5.0
7.5
10.0
12.5
15.0
17.5
total night charge


TOTAL INTL CHARGE - TOTAL INTL MINUTES
total intl minutes
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
1
2
3
4
5
total intl charge

Feature Engineering

In [21]:
```
# Create new feature adding together all 'charge' categories for a new 'tot
df['total charge'] = (df['total day charge'] + df['total eve charge'] + df[
```

In [22]:
```
df.groupby('churn').mean().reset_index()
```

Out[22]:

	churn	account length	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total ch
0	False	100.793684	8.604561	175.175754	100.283158	29.780421	199.043298	100.038596	16.91
1	True	102.664596	5.115942	206.914079	101.335404	35.175921	212.410145	100.561077	18.05

Higher average charge and minutes for True-churn. This would make sense as customer with more minutes would accur a higher charge thus causing them look for a cheaper plan.

In [23]:
```
for i in df_cont.columns:
    q75, q25 = np.percentile(df_cont.loc[:,i], [75 ,25])
    iqr = q75 - q25
    lower = q25 - (iqr*1.5)
    upper = q75 + (iqr*1.5)
    lower_outliers = len(df_cont.loc[df_cont.loc[:,i] < lower,i])
    upper_outliers = len(df_cont.loc[df_cont.loc[:,i] > upper,i])
    print('{columns} ---> {X}  outliers'.format(columns = i, X = (upper_out
```

```
account length ---> 18  outliers
number vmail messages ---> 1  outliers
total day minutes ---> 25  outliers
total day calls ---> 23  outliers
total day charge ---> 25  outliers
total eve minutes ---> 24  outliers
total eve calls ---> 20  outliers
total eve charge ---> 24  outliers
total night minutes ---> 30  outliers
total night calls ---> 22  outliers
total night charge ---> 30  outliers
total intl minutes ---> 46  outliers
total intl calls ---> 78  outliers
total intl charge ---> 49  outliers
customer service calls ---> 267  outliers
```

- In this case i didnt need to remove outliers.

Data Preparation

In [24]:
```
df['voice mail plan'].value_counts()
```

Out[24]:
```
no     2411
yes     922
Name: voice mail plan, dtype: int64
```

In [25]:
```
df['international plan'].value_counts()
```

Out[25]:
```
no     3010
yes     323
Name: international plan, dtype: int64
```

In [26]:
```
df['churn'].value_counts()
```

Out[26]:
```
False    2850
True      483
Name: churn, dtype: int64
```

In [27]:
```
# converting columns with catgeorical information such as churn, internatio
df['vmail plan'] = df['voice mail plan'].map(lambda x:1 if x == 'yes' else
df['int_plan'] = df['international plan'].map(lambda x:1 if x == 'yes' else
df['churn_'] = df['churn'].map(lambda x:1 if x == True else 0)
df = df.drop(['international plan' , 'voice mail plan' , 'churn'] , axis=1)
df.head()
```

Out[27]:

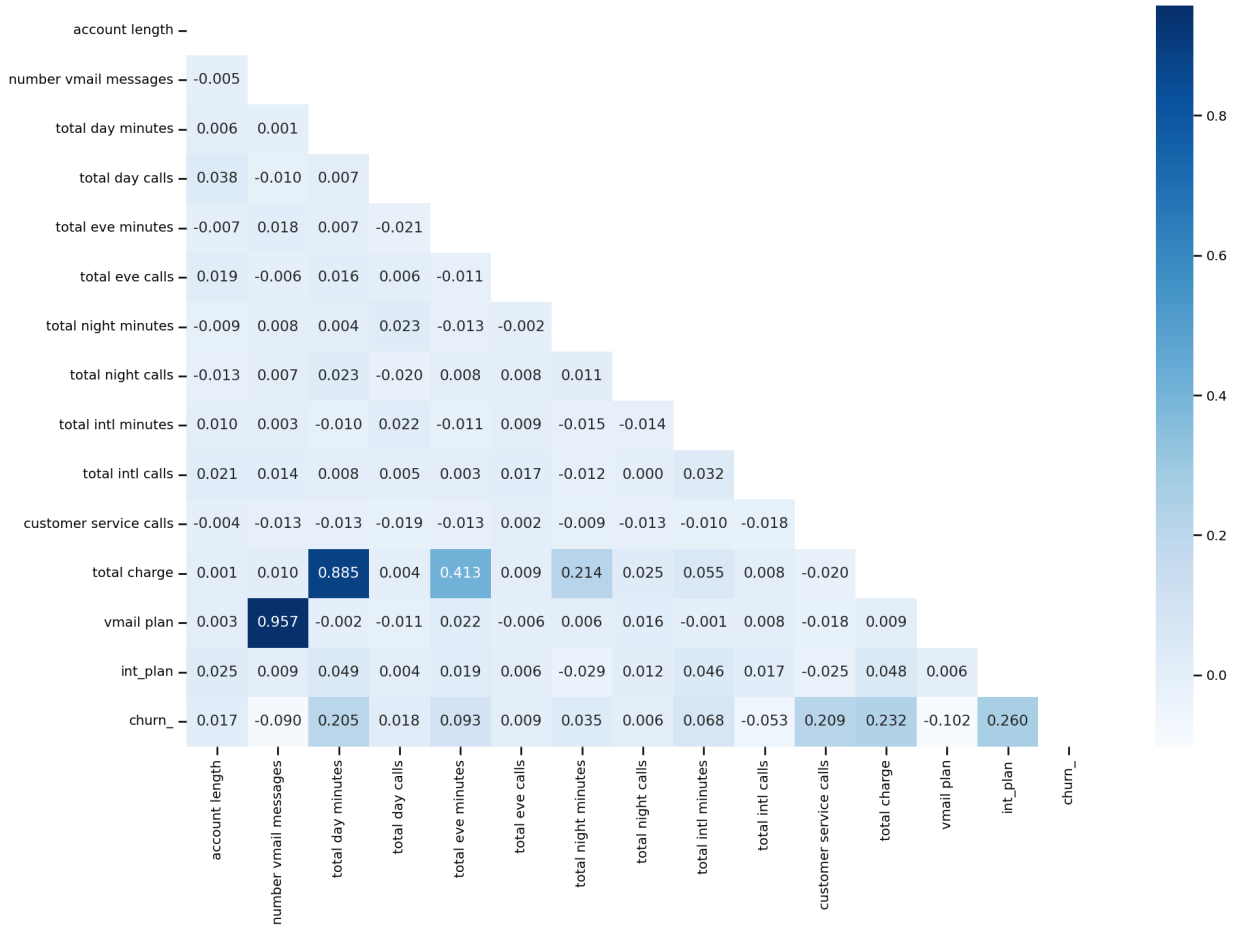
	state	account length	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	tot nig charg
0	KS	128	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.(
1	OH	107	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.4
2	NJ	137	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.3
3	OH	84	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.8
4	OK	75	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.4

In [28]:
```
df.info()
```

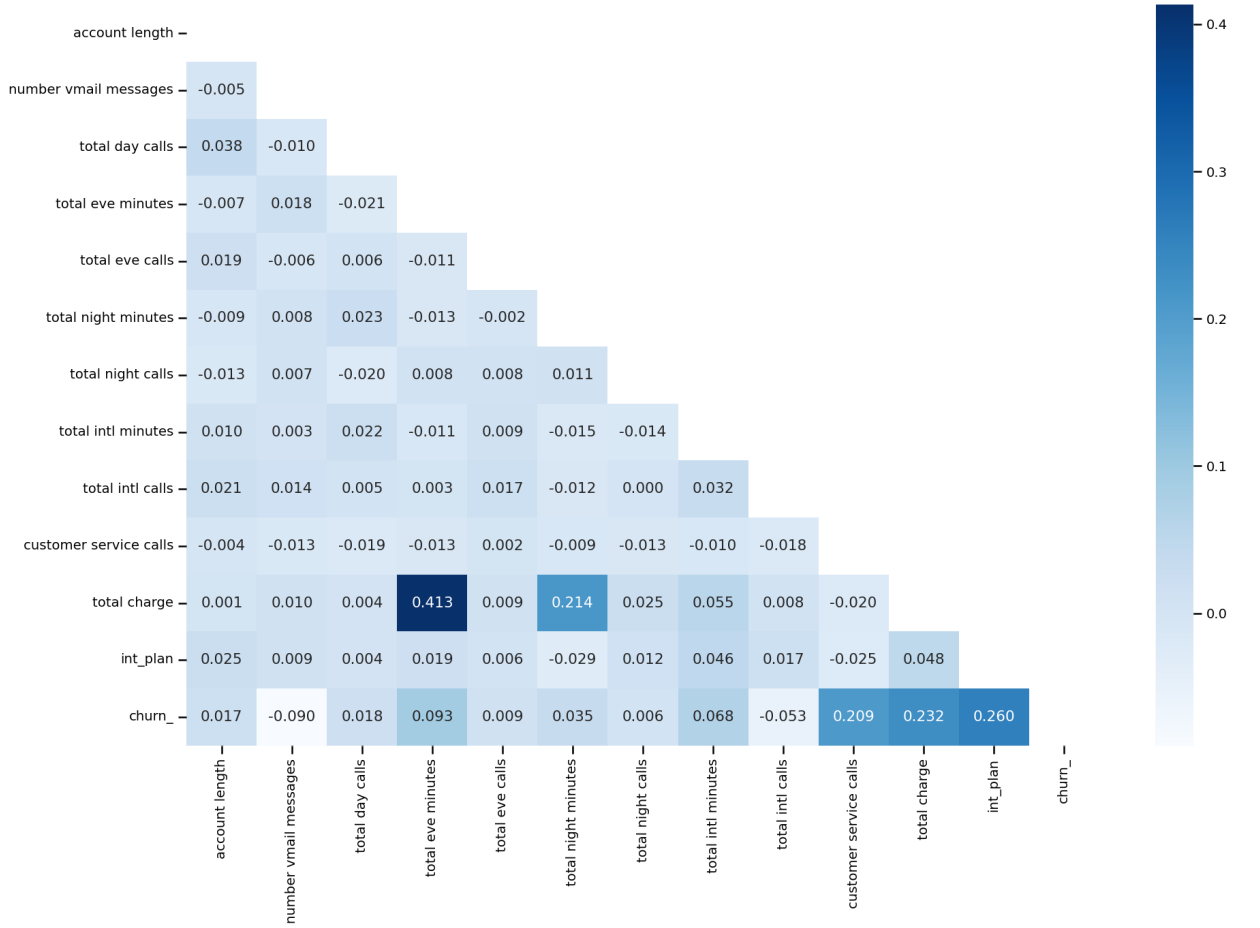
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   number vmail messages   3333 non-null   int64  
 3   total day minutes       3333 non-null   float64
 4   total day calls         3333 non-null   int64  
 5   total day charge        3333 non-null   float64
 6   total eve minutes       3333 non-null   float64
 7   total eve calls         3333 non-null   int64  
 8   total eve charge        3333 non-null   float64
 9   total night minutes     3333 non-null   float64
 10  total night calls       3333 non-null   int64  
 11  total night charge      3333 non-null   float64
 12  total intl minutes      3333 non-null   float64
 13  total intl calls        3333 non-null   int64  
 14  total intl charge       3333 non-null   float64
 15  customer service calls  3333 non-null   int64  
 16  total charge            3333 non-null   float64
 17  vmail plan              3333 non-null   int64  
 18  int_plan                3333 non-null   int64  
 19  churn_                  3333 non-null   int64  
dtypes: float64(9), int64(10), object(1)
memory usage: 520.9+ KB
```

In [29]:
```
#Befor start modeling I will drop columns that have multicollinearty

df.drop(['total day charge', 'total eve charge', 'total night charge', 'tot
```

In [30]:
```
plt.figure(figsize = (15,10));
mask = np.triu(df.corr());
sns.heatmap(df.corr(), cmap = "Blues",annot = True, fmt = '.3f',mask = mask
```



In [31]:
```
# voice mail plan drop yapilcak mi?
df = df.drop(['vmail plan' , 'total day minutes'] ,  axis=1)
plt.figure(figsize = (15,10));
mask = np.triu(df.corr());
sns.heatmap(df.corr(), cmap = "Blues",annot = True, fmt = '.3f',mask = mask
```



In [32]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   number vmail messages   3333 non-null   int64  
 3   total day calls         3333 non-null   int64  
 4   total eve minutes       3333 non-null   float64
 5   total eve calls         3333 non-null   int64  
 6   total night minutes     3333 non-null   float64
 7   total night calls       3333 non-null   int64  
 8   total intl minutes      3333 non-null   float64
 9   total intl calls        3333 non-null   int64  
 10  customer service calls  3333 non-null   int64  
 11  total charge            3333 non-null   float64
 12  int_plan                3333 non-null   int64  
 13  churn_                  3333 non-null   int64  
dtypes: float64(4), int64(9), object(1)
memory usage: 364.7+ KB
```

Spliting Data into Training and Test sets

In [33]:
```
X = df.drop('churn_' , axis =1)
y = df['churn_']
```

In [34]:
```
X_train , X_test , y_train , y_test = train_test_split(X, y , random_state=
print(X_train.shape, y_train.shape)
print(X_test.shape , y_test.shape)
```

```
(2499, 13) (2499,)
(834, 13) (834,)
```

Encoding States

In [35]:
```
ohe = OneHotEncoder(sparse=False, handle_unknown='ignore')
X_tr_cat = X_train[['state']]
X_tr_ohe = pd.DataFrame(ohe.fit_transform(X_tr_cat),columns=ohe.get_feature

X_tr_num = X_train.drop('state', axis = 1)
X_tr = X_tr_num.join(X_tr_ohe)
X_tr.head()
```

Out[35]:

	account length	number vmail messages	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes	total intl calls	customer service calls	...	state
367	45	0	127	253.4	108	255.0	100	18.0	3	1	...	
3103	115	0	111	227.0	108	313.2	113	13.2	1	2	...	
549	121	31	63	205.6	117	196.7	85	10.1	5	4	...	
2531	180	0	134	180.5	113	184.2	87	10.1	4	1	...	
2378	112	0	122	164.5	94	140.3	101	12.6	7	3	...	

5 rows × 63 columns

In [36]:
```
X_tr.shape
```

Out[36]:
```
(2499, 63)
```

In [37]:
```
y_train.shape
```

Out[37]:
```
(2499,)
```

In [38]:
```
X_test_cat = X_test[['state']]
X_test_ohe = pd.DataFrame(ohe.fit_transform(X_test_cat),columns=ohe.get_fea
                          index=X_test_cat.index)

X_test_num = X_test.drop('state', axis = 1)
X_te = X_test_num.join(X_test_ohe)
X_te.head()
```

Out[38]:

	account length	number vmail messages	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes	total intl calls	customer service calls	...	state
438	113	0	93	330.6	106	189.4	123	13.5	3	1	...	
2674	67	0	117	217.4	124	188.4	141	12.8	6	0	...	
1345	98	0	0	159.6	130	167.1	88	6.8	1	4	...	
1957	147	0	79	204.1	91	156.2	113	10.2	2	1	...	
2148	96	0	102	224.7	73	227.7	91	10.0	7	1	...	

5 rows × 63 columns

In [39]:
```
X_te.shape
```

Out[39]:
```
(834, 63)
```

In [40]:
```
y_test.shape
```

Out[40]:
```
(834,)
```

Smote Imbalanced Data

As our data are imbalanced, we used oversampling method during the model building process. After preprocessing the data, we build six model including vanilla and tuned model.

In [41]:
```
#SMOTE(Synthetic Minority Over-sampling Technique)
#Before smote y_train
y_train.value_counts()
```

Out[41]:
```
0    2141
1     358
Name: churn_, dtype: int64
```

In [42]:
```
y_test.value_counts()
```

Out[42]:
```
0    709
1    125
Name: churn_, dtype: int64
```

In [43]:
```
smote = SMOTE(random_state=42)
X_tr_sm, y_tr_sm = smote.fit_resample(X_tr,y_train)
X_tr_sm.shape , y_tr_sm.shape
```

Out[43]:
```
((4282, 63), (4282,))
```

In [44]:
```
y_tr_sm.value_counts()
```

Out[44]:
```
0    2141
1    2141
Name: churn_, dtype: int64
```

Scaling Data

In [45]:
```
X_tr_sm.head()
```

Out[45]:

	account length	number vmail messages	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes	total intl calls	customer service calls	...	state_SI
0	45	0	127	253.4	108	255.0	100	18.0	3	1	...	0.
1	115	0	111	227.0	108	313.2	113	13.2	1	2	...	0.
2	121	31	63	205.6	117	196.7	85	10.1	5	4	...	0.
3	180	0	134	180.5	113	184.2	87	10.1	4	1	...	0.
4	112	0	122	164.5	94	140.3	101	12.6	7	3	...	0.

5 rows × 63 columns

In [46]:
```
# Scale testing and training features
scaler = StandardScaler()
scaler.fit(X_tr_sm)
X_tr_sm_scaled = scaler.transform(X_tr_sm)
X_tr_sm_scaled= pd.DataFrame(X_tr_sm_scaled, columns=X_tr_sm.columns)
X_test_scaled = scaler.transform(X_te)
X_test_scaled = pd.DataFrame(X_test_scaled , columns = X_te.columns)
```

Metric to use : Recall or Precision

- Precision: Of all the users that the algorithm predicts will churn, how many of them do actually churn?

It might have become obvious that precision and churn are inversely related. For instance, if an algorithm predicts that every single user will churn, it would have perfect recall. But clearly, not every user will churn from your business – and your algorithm's precision would end up being very low, and it wouldn't be useful for your business.

- Recall : What percentage of users that end up churning does the algorithm successfully find?

Recall would be a better metric for this dataset because with churn rate, we can implement more customer retention strategies, and misidentifying someone as cancelling and hitting them with a strategy to keep them engaged would be more beneficial than missing someone who exited and not hitting them with a strategy to keep them subscribed to the service.

Model 1 : Logistic Regression

Having a false negative (Type II Error) means our model is predicting that the customer isn't going to cancel but they really are. Our priority is given to the metric of `Recall` .

In [47]:
```
logreg = LogisticRegression()
logreg.fit(X_tr_sm_scaled, y_tr_sm)
```

Out[47]:
```
▾ LogisticRegression

LogisticRegression()
```

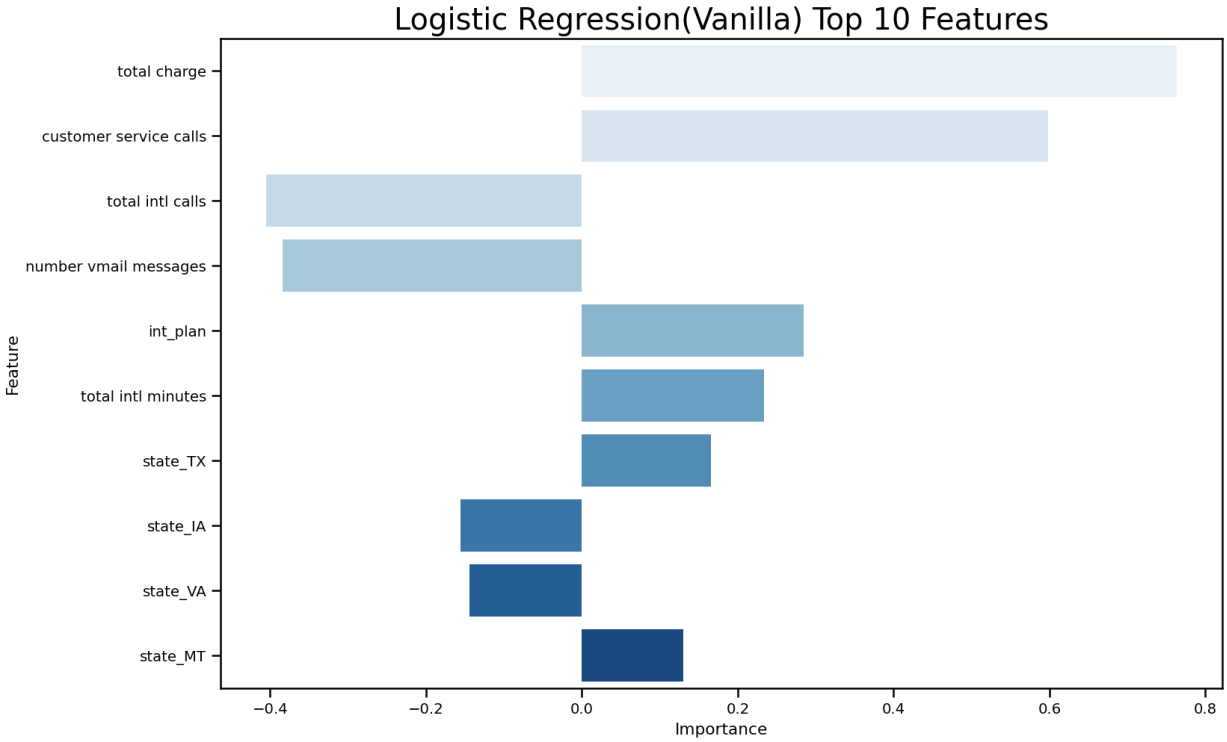
In [48]:
```
feat_import_lr = pd.DataFrame(zip(X_tr_sm_scaled.columns, logreg.coef_[0,:]
feat_import_lr.sort_values(by='Importance', key=abs, ascending=False)
```

Out[48]:

	Feature	Importance
10	total charge	0.762831
9	customer service calls	0.598284
8	total intl calls	-0.404954
1	number vmail messages	-0.383928
11	int_plan	0.284495
...	...	...
41	state_NE	0.006199
61	state_WV	-0.005790
39	state_NC	0.004842
42	state_NH	0.002721
35	state_MN	0.002672

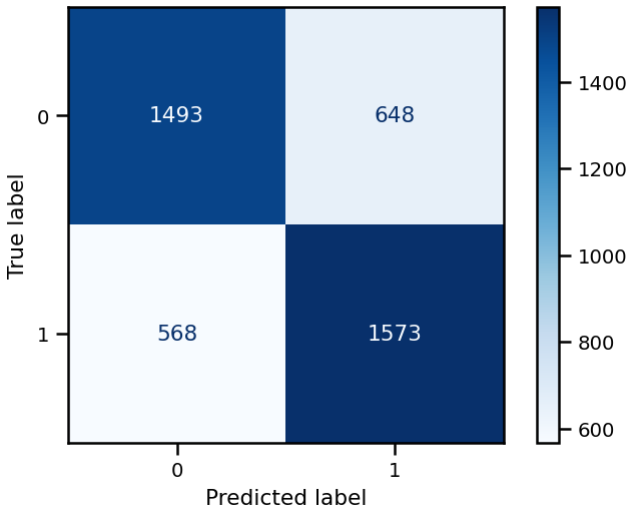
63 rows × 2 columns

In [49]:
```
fig, ax = plt.subplots(figsize=(12,8))
sns.barplot(x='Importance', y='Feature',
            data=feat_import_lr.sort_values(by='Importance', key=abs, ascen
           palette='Blues');
plt.title('Logistic Regression(Vanilla) Top 10 Features' , fontdict={'fonts
```

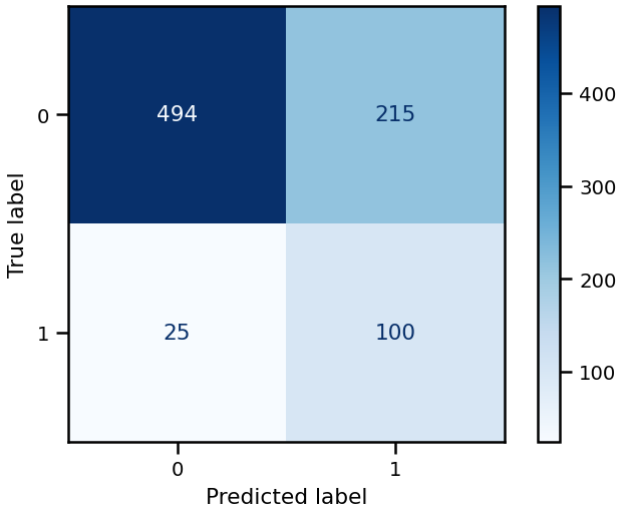


In [50]:
```
y_pred = logreg.predict(X_test_scaled)
```

In [51]:
```
plot_confusion_matrix(logreg, X_tr_sm_scaled, y_tr_sm , cmap='Blues');
```



In [52]:
```
plot_confusion_matrix(logreg, X_test_scaled, y_test , cmap='Blues');
```



- We want to reduce that 25 because those are our False Negatives (Type II Error) means our model is predicting that the customer isn't going to cancel but they really are.

- According to the testing data confusion matrix, out of all the true positives,Logistic Regression predicts 100 out of 125 correctly. This would seem to indicate a `recall` score which should be reflected in the classification report.

In [53]:
```
print(classification_report(y_tr_sm, logreg.predict(X_tr_sm_scaled)))
```

```
              precision    recall  f1-score   support

           0       0.72      0.70      0.71      2141
           1       0.71      0.73      0.72      2141

    accuracy                           0.72      4282
   macro avg       0.72      0.72      0.72      4282
weighted avg       0.72      0.72      0.72      4282
```

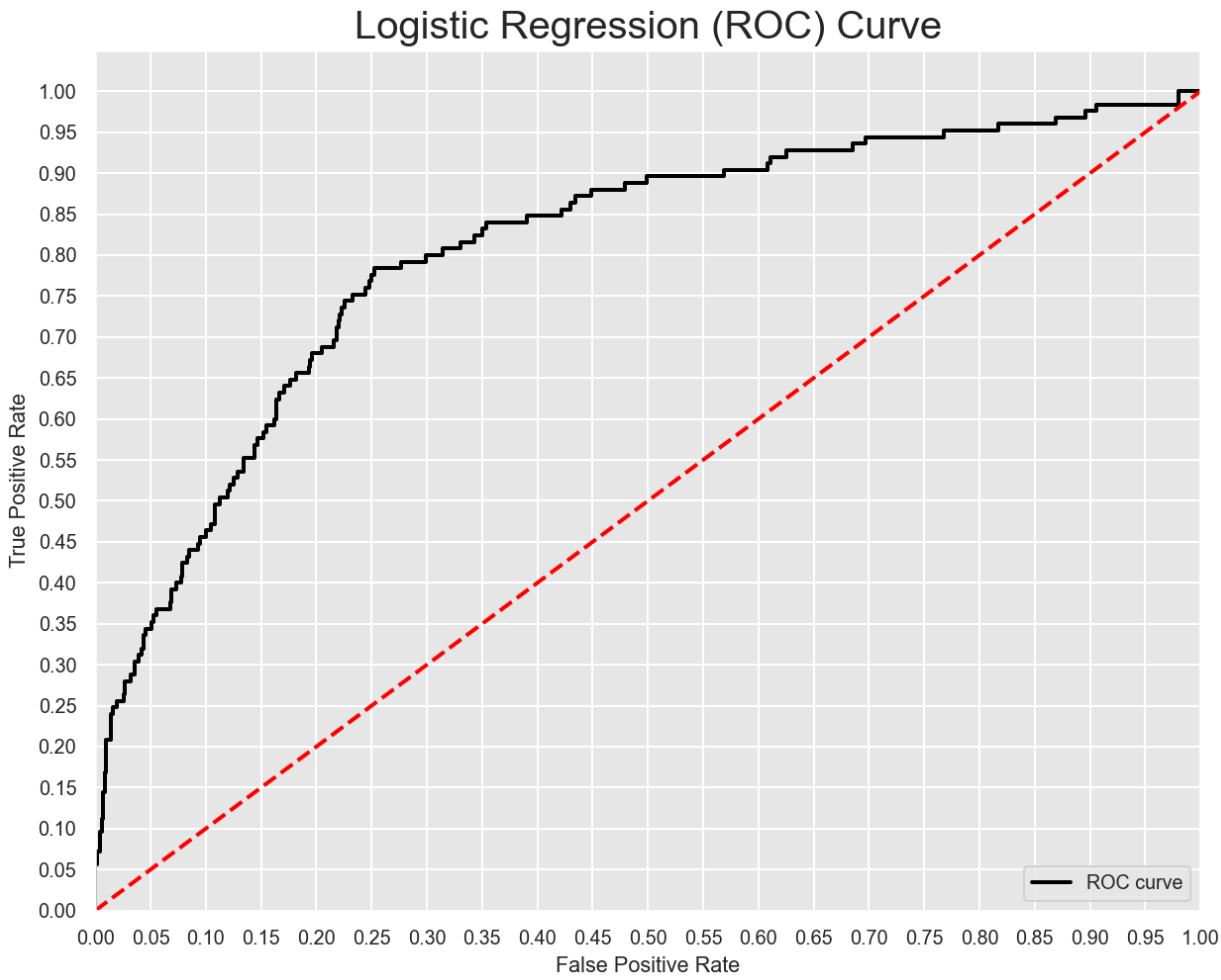
In [54]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.70      0.80       709
           1       0.32      0.80      0.45       125

    accuracy                           0.71       834
   macro avg       0.63      0.75      0.63       834
weighted avg       0.86      0.71      0.75       834
```

- According to the testing data classification report Logistic Regression shows a 80% recall score for true-churn

In [55]:
```
# Probability scores for test set
y_score = logreg.fit(X_tr_sm_scaled, y_tr_sm).decision_function(X_test_scal

# False positive rate and true positive rate
fpr, tpr, thresholds = roc_curve(y_test, y_score)

# Seaborn's beautiful styling
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

# Print AUC
print('AUC: {}'.format(auc(fpr, tpr)))

# Plot the ROC curve
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr, tpr, color = 'black',
         lw=lw, label='ROC curve')
plt.plot([0,1], [0,1], color = 'red', lw=lw, linestyle ='--')
plt.xlim([0.0,1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression (ROC) Curve' , fontdict={'fontsize': 20})
plt.legend(loc='lower right')
plt.show()
```

```
AUC: 0.8073568406205924
```



In [56]:
```
logreg.score(X_test_scaled, y_test)
```

Out[56]:
```
0.7122302158273381
```

Model 1 Summary:

- Recall Score (Test Data) : 80%
- AUC : 81%
- Mean accuracy : 71%

Model 2: Logistic Regression(Tuned Parameters)

GridsearchCV

In [57]:
```
# Select parameter options for gridsearchcv
param_grid = [
    {'penalty' : ['l1', 'l2', 'elasticnet'],
    'C' : np.logspace(-3,3,7),
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
    'max_iter' : [100, 1000,2500, 5000]
    }]
```

In [58]:
```
clf = GridSearchCV(logreg, param_grid = param_grid, cv = 3, verbose=True, n
clf.fit(X_tr_sm_scaled, y_tr_sm)
```

```
Fitting 3 folds for each of 420 candidates, totalling 1260 fits
```

Out[58]:

GridSearchCV

```
GridSearchCV(cv=3, estimator=LogisticRegression(), n_jobs=-1,
             param_grid=[{'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00,
1.e+01, 1.e+02, 1.e+03]),
                          'max_iter': [100, 1000, 2500, 5000],
                          'penalty': ['l1', 'l2', 'elasticnet'],
                          'solver': ['lbfgs', 'newton-cg', 'liblinear',
'sag',
                                     'saga']}],
             verbose=True)
```

estimator: LogisticRegression

LogisticRegression

In [59]:
```
# Best classifer estimates
clf.best_estimator_
```

Out[59]:

LogisticRegression

```
LogisticRegression(C=10.0, penalty='l1', solver='liblinear')
```

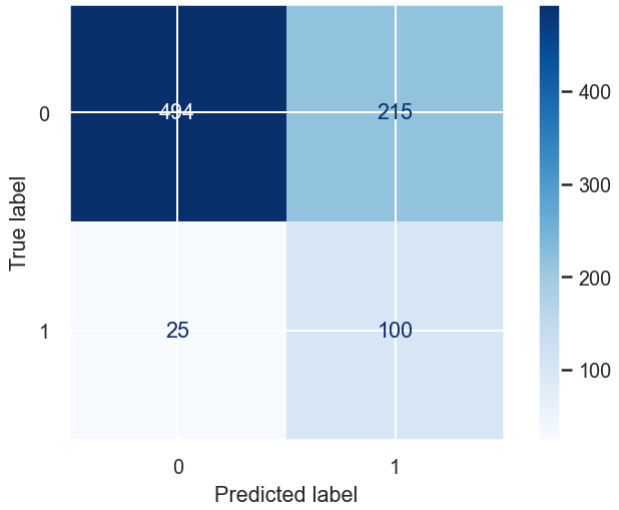
In [60]:
```
# Create tunned logistic regression model
logreg2 = LogisticRegression(C=10.0 , penalty='l1', solver='liblinear')
logreg2.fit(X_tr_sm_scaled, y_tr_sm)
```

Out[60]:

LogisticRegression

```
LogisticRegression(C=10.0, penalty='l1', solver='liblinear')
```

In [61]:
```
plot_confusion_matrix(logreg2, X_test_scaled, y_test, cmap='Blues');
sns.set_style('dark')
```



In [62]:
```
print(classification_report(y_tr_sm, logreg2.predict(X_tr_sm_scaled)))
```

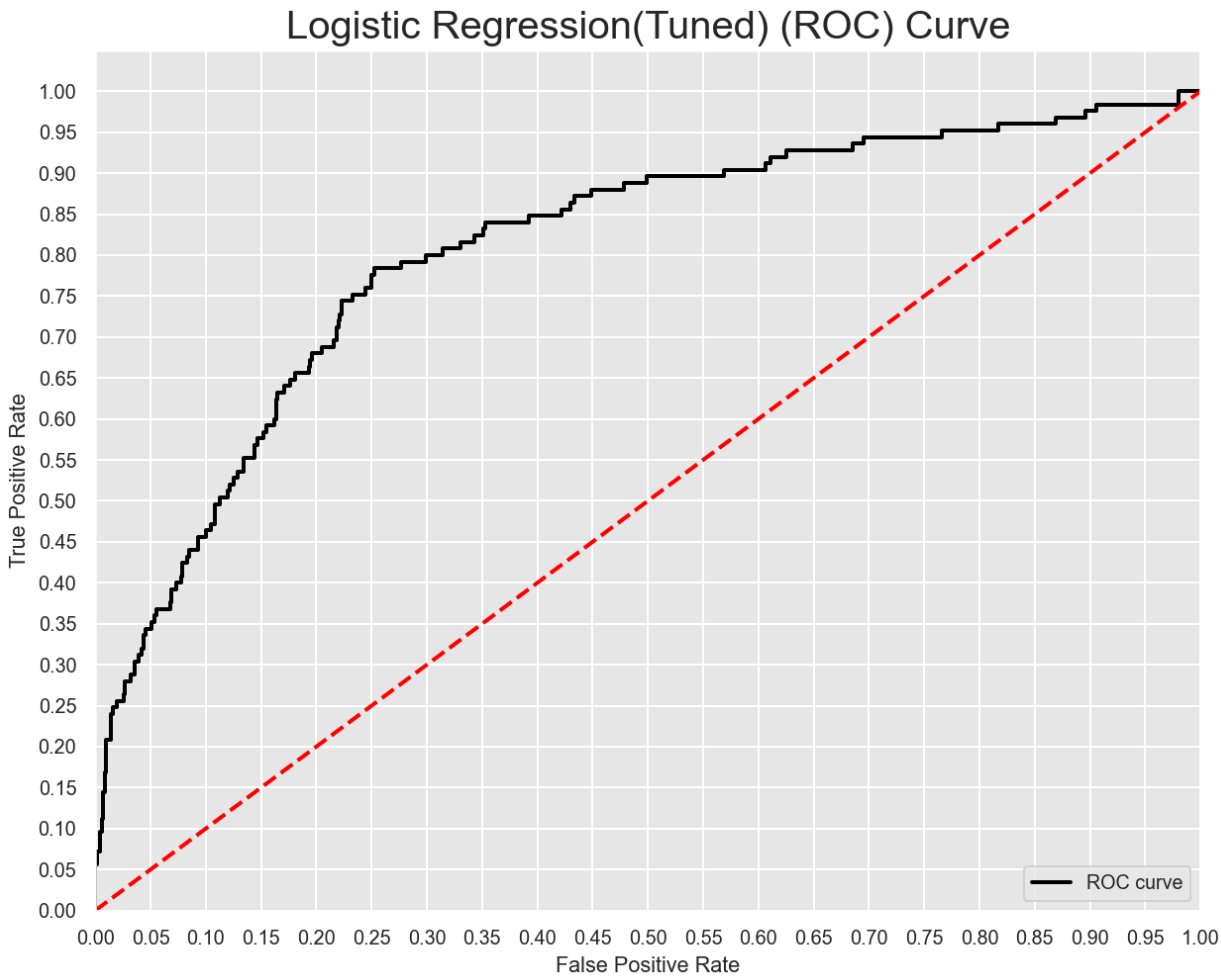
```
              precision    recall  f1-score   support

           0       0.72      0.70      0.71      2141
           1       0.71      0.73      0.72      2141

    accuracy                           0.72      4282
   macro avg       0.72      0.72      0.72      4282
weighted avg       0.72      0.72      0.72      4282
```

In [63]:
```
print(classification_report(y_test, logreg2.predict(X_test_scaled)))
```

```
              precision    recall  f1-score   support

           0       0.95      0.70      0.80       709
           1       0.32      0.80      0.45       125

    accuracy                           0.71       834
   macro avg       0.63      0.75      0.63       834
weighted avg       0.86      0.71      0.75       834
```

In [64]:
```
# Probability scores for test set
y_score = logreg2.fit(X_tr_sm_scaled, y_tr_sm).decision_function(X_test_sca

# False positive rate and true positive rate
fpr, tpr, thresholds = roc_curve(y_test, y_score)

# Seaborn's beautiful styling
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

# Print AUC
print('AUC: {}'.format(auc(fpr, tpr)))

# Plot the ROC curve
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr, tpr, color = 'black',
         lw=lw, label='ROC curve')
plt.plot([0,1], [0,1], color = 'red', lw=lw, linestyle ='--')
plt.xlim([0.0,1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression(Tuned) (ROC) Curve' , fontdict={'fontsize':
plt.legend(loc='lower right')
plt.show()
```

```
AUC: 0.8074358251057828
```



In [65]:
```
logreg2.score(X_test_scaled, y_test)
```

Out[65]:
```
0.7122302158273381
```

Model 2 Summary:

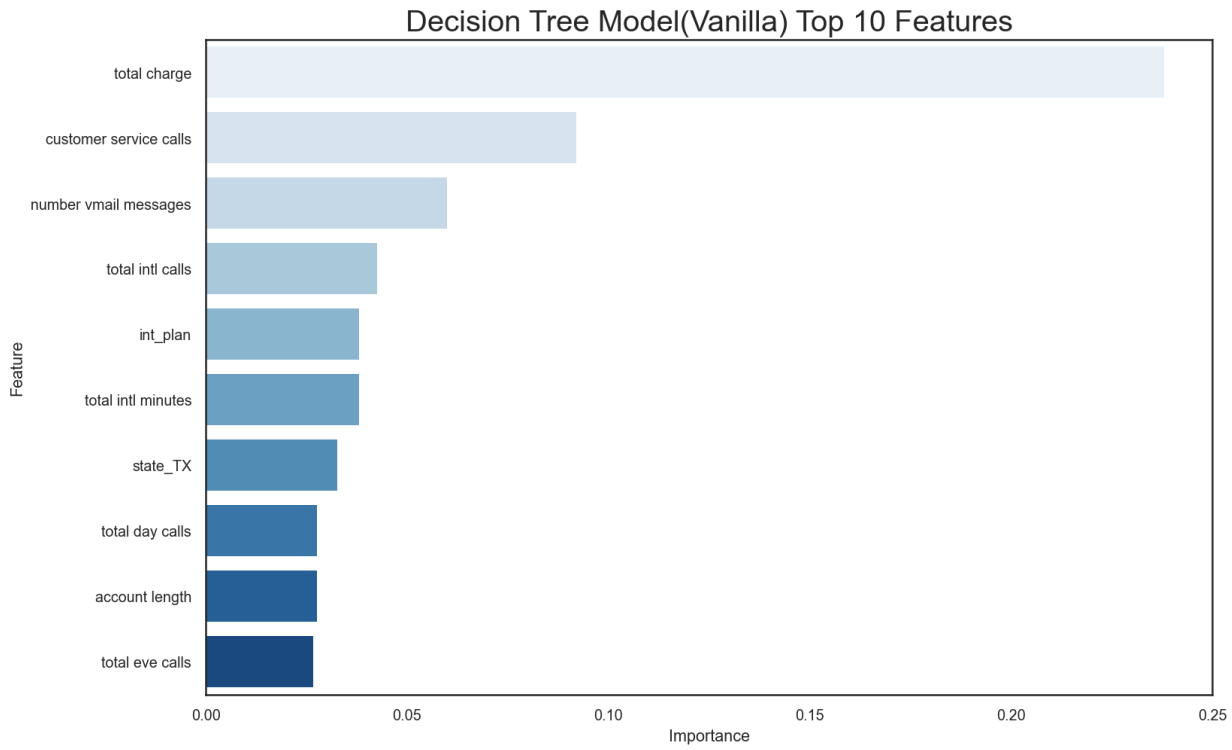
- Recall Score (Test Data) : 80%
- AUC : 81%
- Mean accuracy : 71%

Model 3: Decision Tree

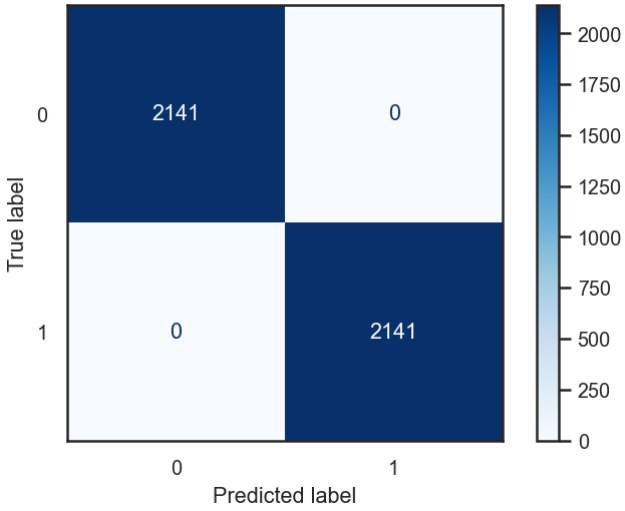
In [66]:
```
dt = DecisionTreeClassifier()
dt.fit(X_tr_sm_scaled , y_tr_sm)
```

Out[66]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

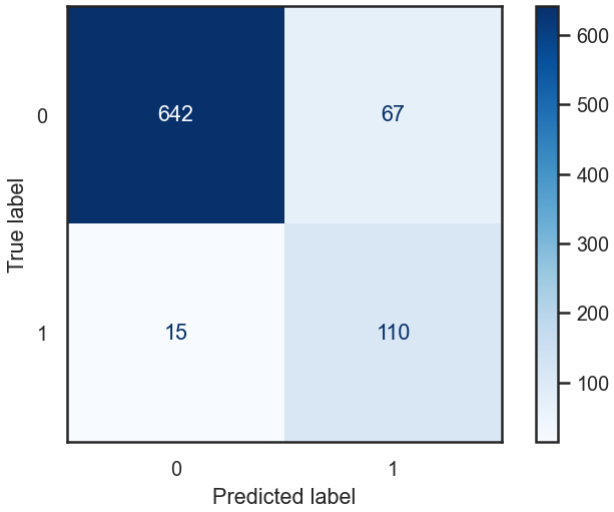
In [67]:
```
feat_import_df = pd.DataFrame(zip(X_tr_sm_scaled.columns, dt.feature_import
                              columns=['Feature', 'Importance'])
sns.set_style('white')
fig, ax = plt.subplots(figsize=(12,8))
sns.barplot(x='Importance', y='Feature',
            data=feat_import_df.sort_values(by='Importance', key=abs, ascen
           palette = 'Blues');
plt.title('Decision Tree Model(Vanilla) Top 10 Features' , fontdict={'fonts
```



In [68]:
```
plot_confusion_matrix(dt , X_tr_sm_scaled , y_tr_sm , cmap='Blues');
```



In [69]:
```
plot_confusion_matrix(dt, X_test_scaled, y_test , cmap='Blues');
```



- According to the testing data confusion matrix, out of all the true positives,Decision Tree predicts 110 out of 125 correctly. This would seem to indicate a `recall` score which should be reflected in the classification report.

In [70]:
```
print(classification_report(y_tr_sm, dt.predict(X_tr_sm_scaled)))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2141
           1       1.00      1.00      1.00      2141

    accuracy                           1.00      4282
   macro avg       1.00      1.00      1.00      4282
weighted avg       1.00      1.00      1.00      4282
```

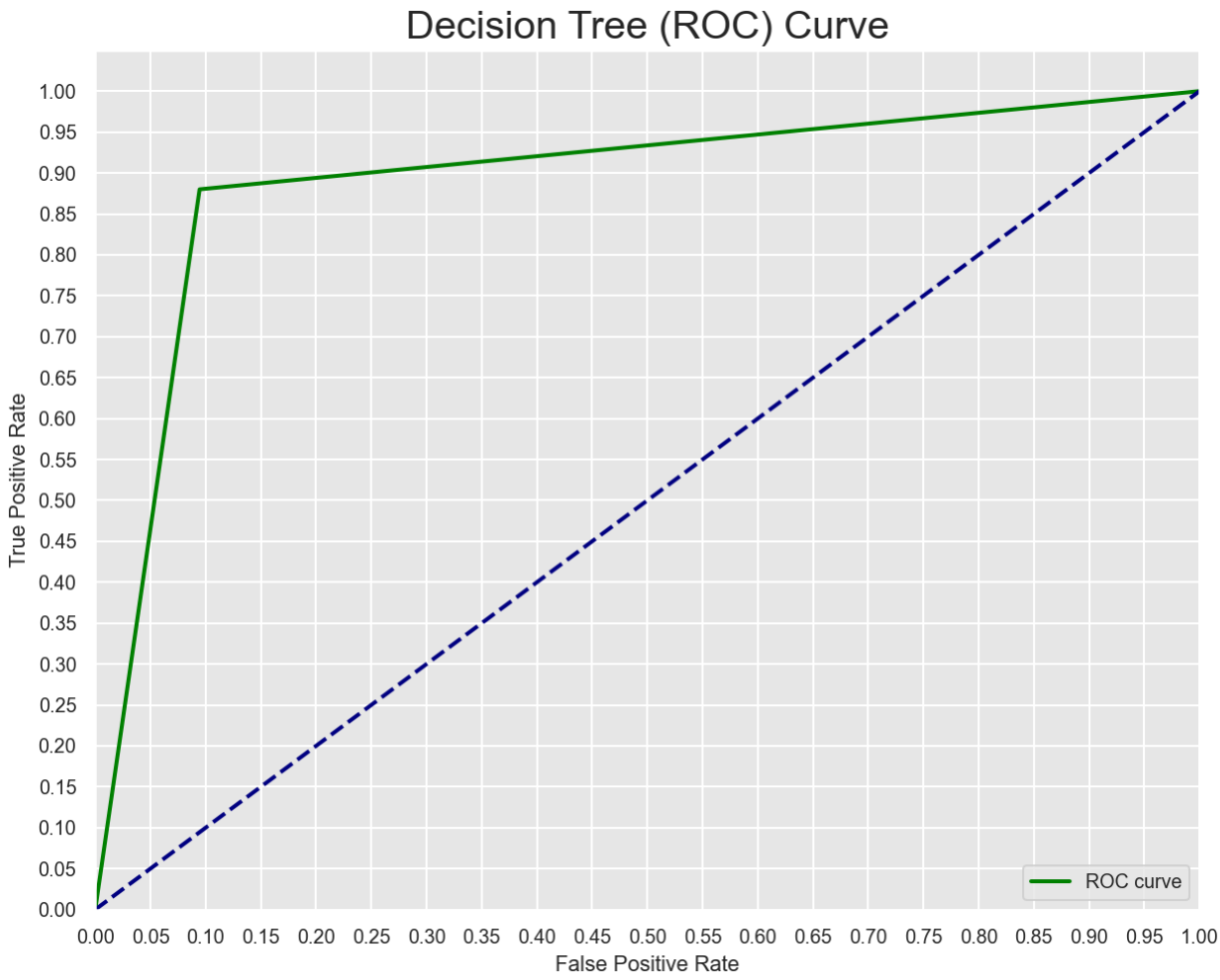
- According to the training data classification report Decision Tree model shows 100% recall score.This would likely indicate that the model is overfitting the data and will need to tune the decision tree parameters for a more reliable model.

In [71]:
```
print(classification_report(y_test, dt.predict(X_test_scaled)))
```

```
              precision    recall  f1-score   support

           0       0.98      0.91      0.94       709
           1       0.62      0.88      0.73       125

    accuracy                           0.90       834
   macro avg       0.80      0.89      0.83       834
weighted avg       0.92      0.90      0.91       834
```

In [72]:
```
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})
fpr, tpr, thresholds = roc_curve(y_test, dt.predict(X_test_scaled))
print('AUC: {}'.format(auc(fpr, tpr)))
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr, tpr, color='green',
         lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Decision Tree (ROC) Curve' , fontdict={'fontsize': 20})
plt.legend(loc='lower right')
plt.show()
```

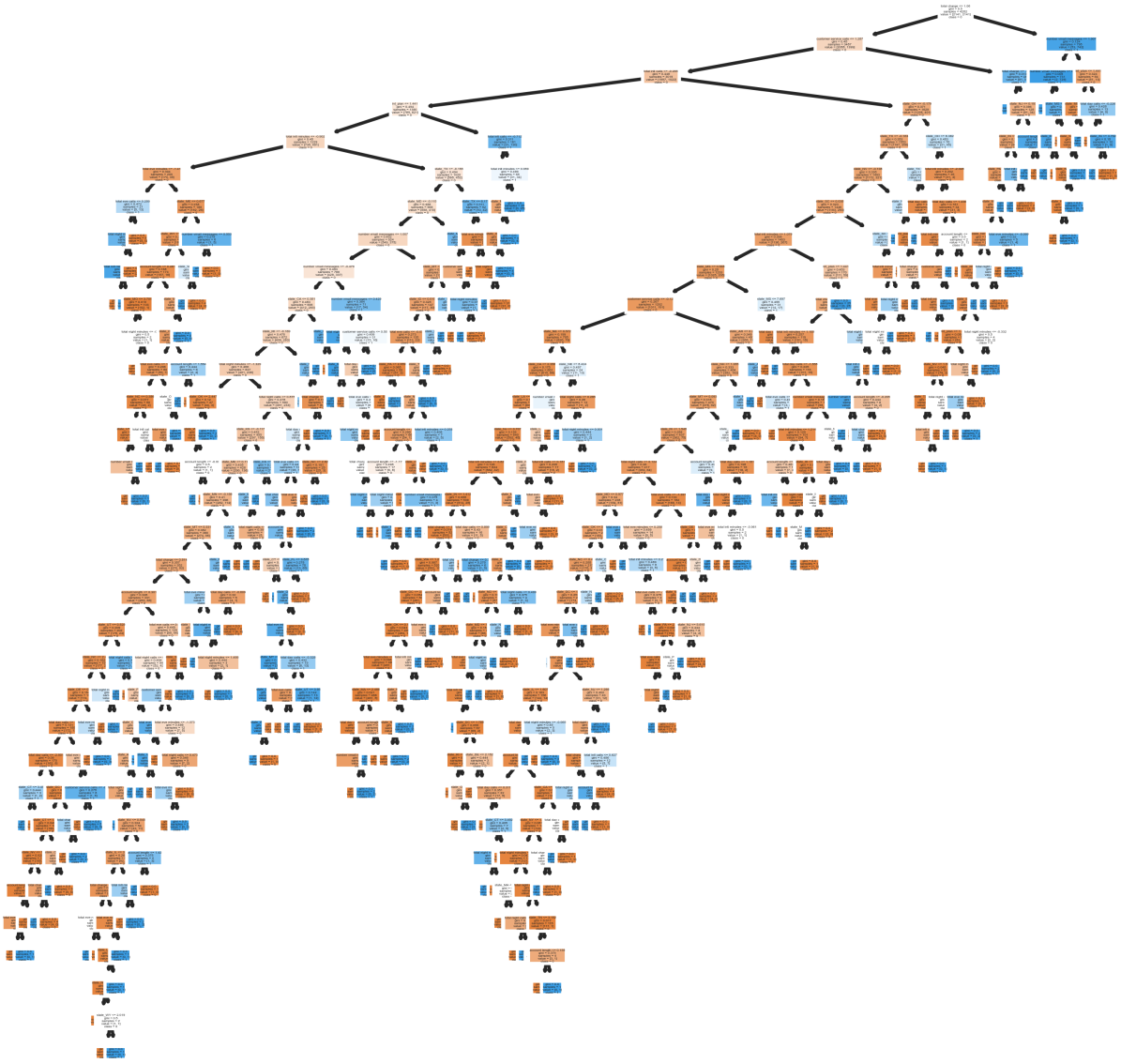
```
AUC: 0.8927503526093089
```



In [73]:
```
dt.score(X_test_scaled, y_test)
```

Out[73]:
```
0.9016786570743405
```

In [74]:
```
# Graph Decision tree
fig, axes = plt.subplots(nrows = 1,ncols = 1, figsize = (6,6), dpi=300)
tree.plot_tree(dt,
               feature_names = X_tr_sm_scaled.columns,
               class_names=np.unique(y_tr_sm).astype('str'),
               filled = True)
plt.show()
```



Model 3 Summary:

- Recall Score (Test Data) : 88%
- AUC : 90%
- Mean Accuracy : 91%

Model 4: Decision Tree (Tuned Parameters)

GridsearchCV

In [75]:
```
# Select parameter options for gridsearchcv
param_grid_dt = [
    {'max_depth' : [1, 10, 10],
    'min_samples_split' : [1, 10, 10],
    'min_samples_leaf' : [0.1, 0.5, 5]
    }]
```

In [76]:
```
clf_dt = GridSearchCV(dt , param_grid= param_grid_dt ,cv = 3, verbose=True,
clf_dt.fit(X_tr_sm_scaled ,y_tr_sm)
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
```

Out[76]:

GridSearchCV

```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid=[{'max_depth': [1, 10, 10],
                          'min_samples_leaf': [0.1, 0.5, 5],
                          'min_samples_split': [1, 10, 10]}],
             verbose=True)
```

estimator: DecisionTreeClassifier

DecisionTreeClassifier

In [77]:
```
clf_dt.best_estimator_
```

Out[77]:

DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=5, min_samples_spl
it=10)
```

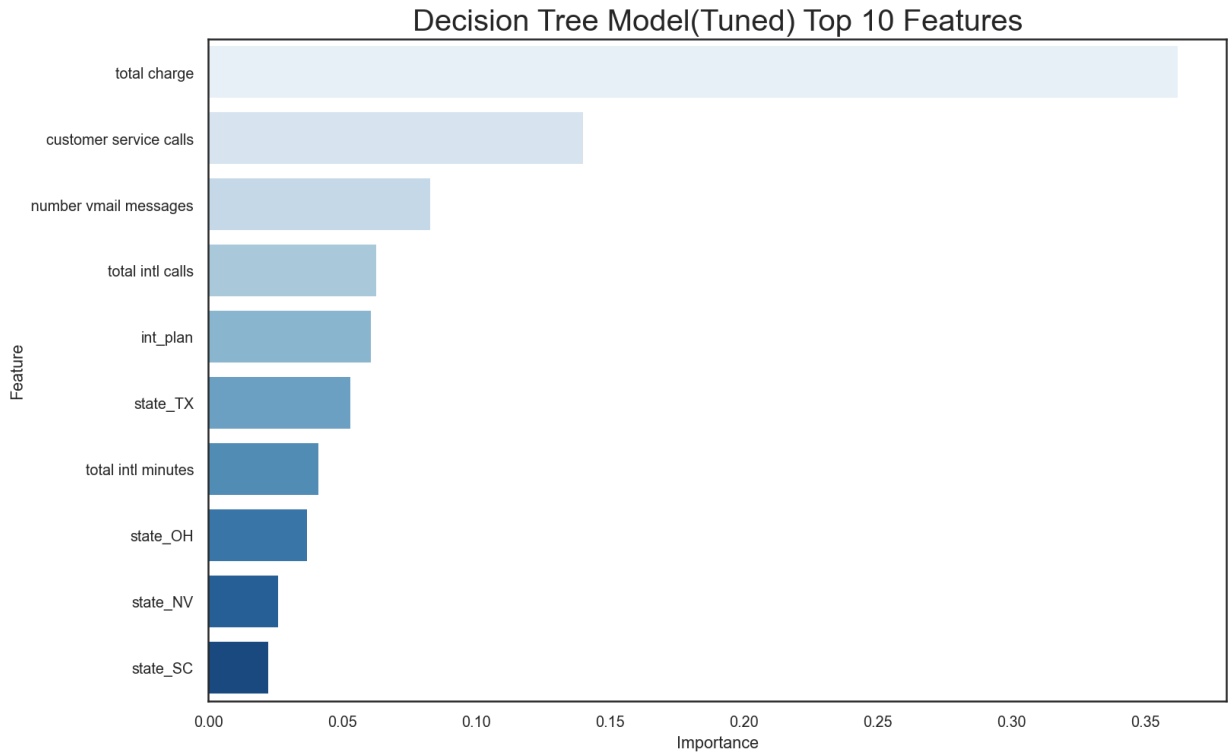
In [78]:
```
# Create tunned Decision Tree model

dt2 = DecisionTreeClassifier(max_depth=10 , min_samples_leaf=5 , min_sample
dt2.fit(X_tr_sm_scaled , y_tr_sm)
```

Out[78]:

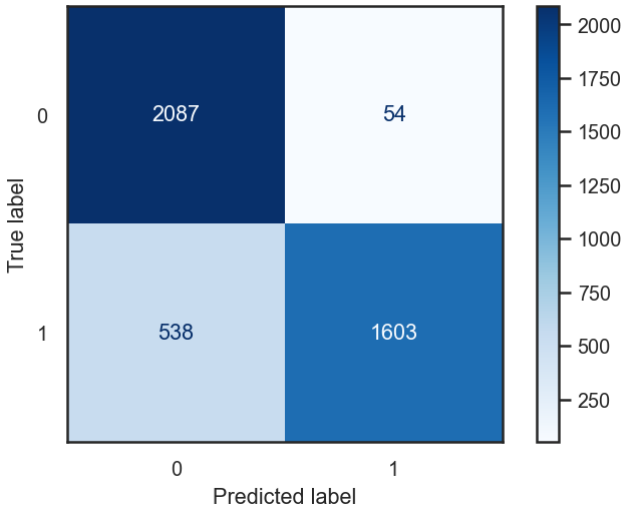
DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=5, min_samples_spl
it=10)
```

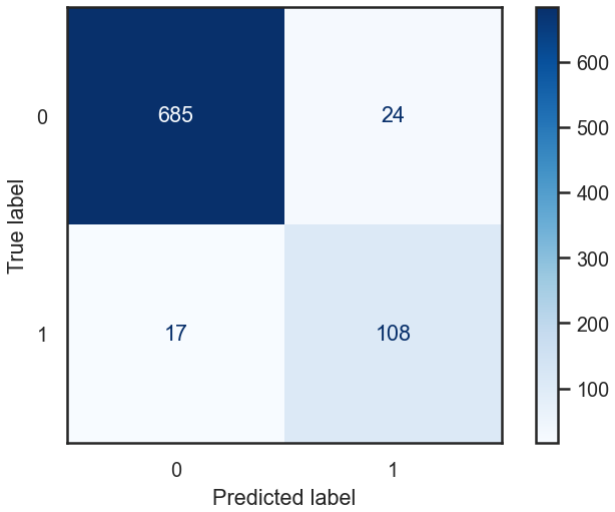
In [79]:
```
feat_imps_df2 = pd.DataFrame(zip(X_tr_sm_scaled.columns, dt2.feature_import
                             columns=['Feature', 'Importance'])
sns.set_style('white')
fig, ax = plt.subplots(figsize=(12,8))
sns.barplot(x='Importance', y='Feature',
            data=feat_imps_df2.sort_values(by='Importance', key=abs, ascend
           palette = 'Blues');
plt.title('Decision Tree Model(Tuned) Top 10 Features' , fontdict={'fontsiz
```



In [80]:
```
plot_confusion_matrix(dt2 , X_tr_sm_scaled , y_tr_sm , cmap='Blues');
```



In [81]:
```
plot_confusion_matrix(dt2 , X_test_scaled , y_test , cmap='Blues');
```



In [82]:
```
print(classification_report(y_tr_sm , dt2.predict(X_tr_sm_scaled)))
```

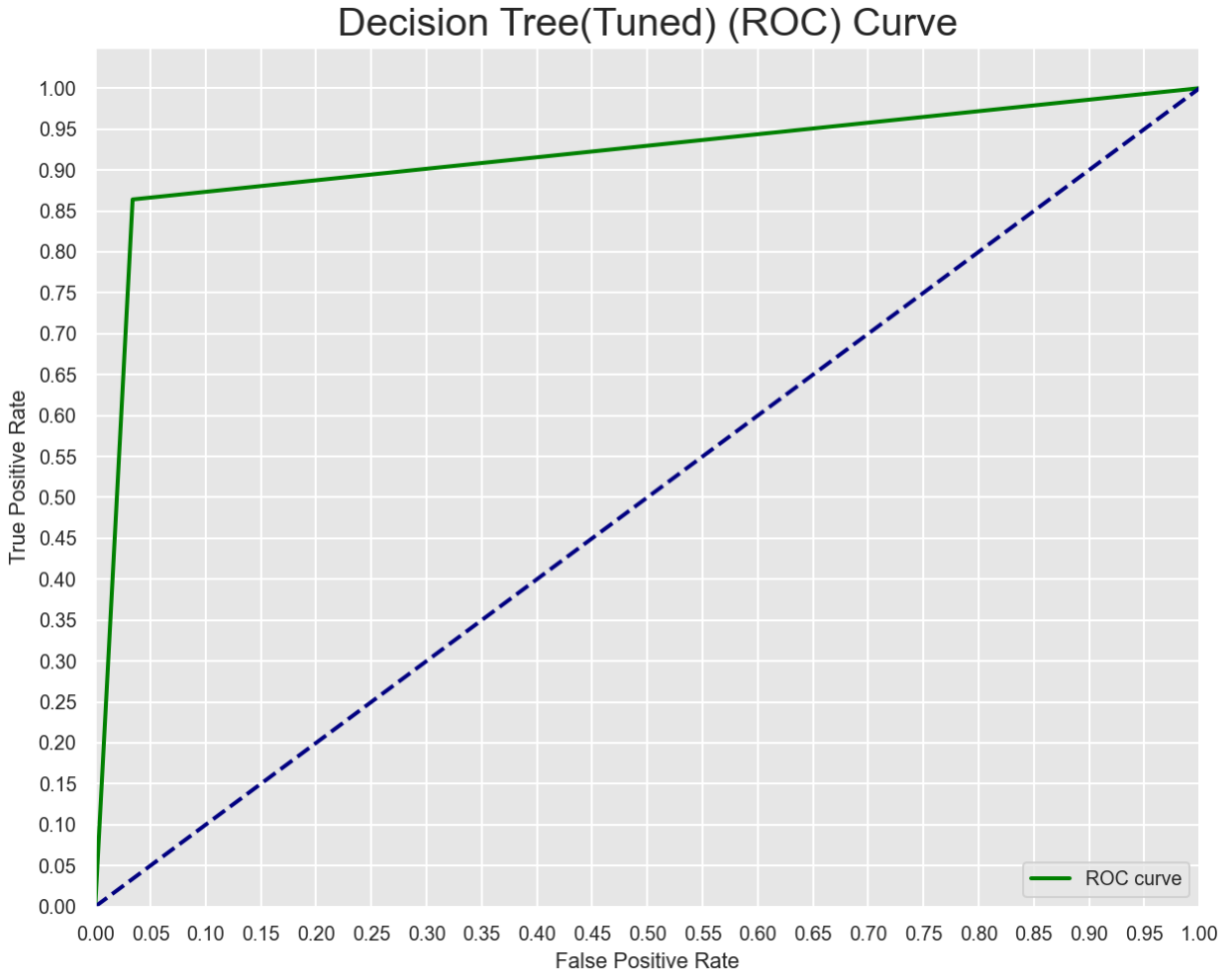
```
              precision    recall  f1-score   support

           0       0.80      0.97      0.88      2141
           1       0.97      0.75      0.84      2141

    accuracy                           0.86      4282
   macro avg       0.88      0.86      0.86      4282
weighted avg       0.88      0.86      0.86      4282
```

In [83]:
```
print(classification_report(y_test , dt2.predict(X_test_scaled)))
```

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       709
           1       0.82      0.86      0.84       125

    accuracy                           0.95       834
   macro avg       0.90      0.92      0.91       834
weighted avg       0.95      0.95      0.95       834
```

In [84]:
```
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})
fpr, tpr, thresholds = roc_curve(y_test, dt2.predict(X_test_scaled))
print('AUC: {}'.format(auc(fpr, tpr)))
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr, tpr, color='green',
         lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Decision Tree(Tuned) (ROC) Curve' , fontdict={'fontsize': 20})
plt.legend(loc='lower right')
plt.show()
```

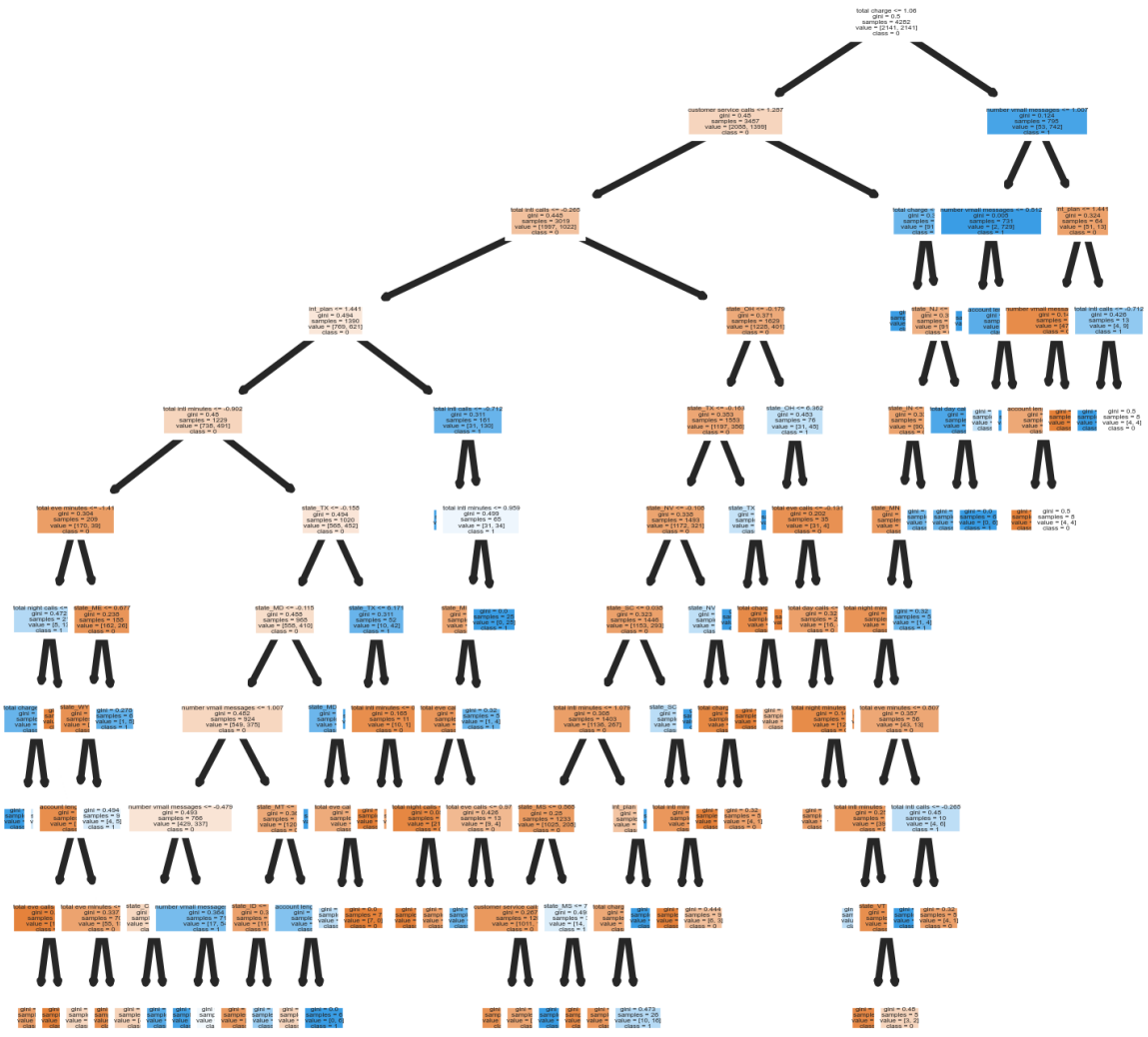
```
AUC: 0.9150747531734837
```



In [85]:
```
dt2.score(X_test_scaled, y_test)
```

Out[85]:
```
0.9508393285371702
```

In [86]:
```
# Graph Decision tree
fig, axes = plt.subplots(nrows = 1,ncols = 1, figsize = (3,3), dpi=300)
tree.plot_tree(dt2,
               feature_names = X_tr_sm_scaled.columns,
               class_names=np.unique(y_tr_sm).astype('str'),
               filled = True)
plt.show()
```



- Parametre tunned Decision Tree Model does not appear to be overfitting based.

Model 4 Summary:

- Recall Score (Test Data) : 86%
- AUC : 92%

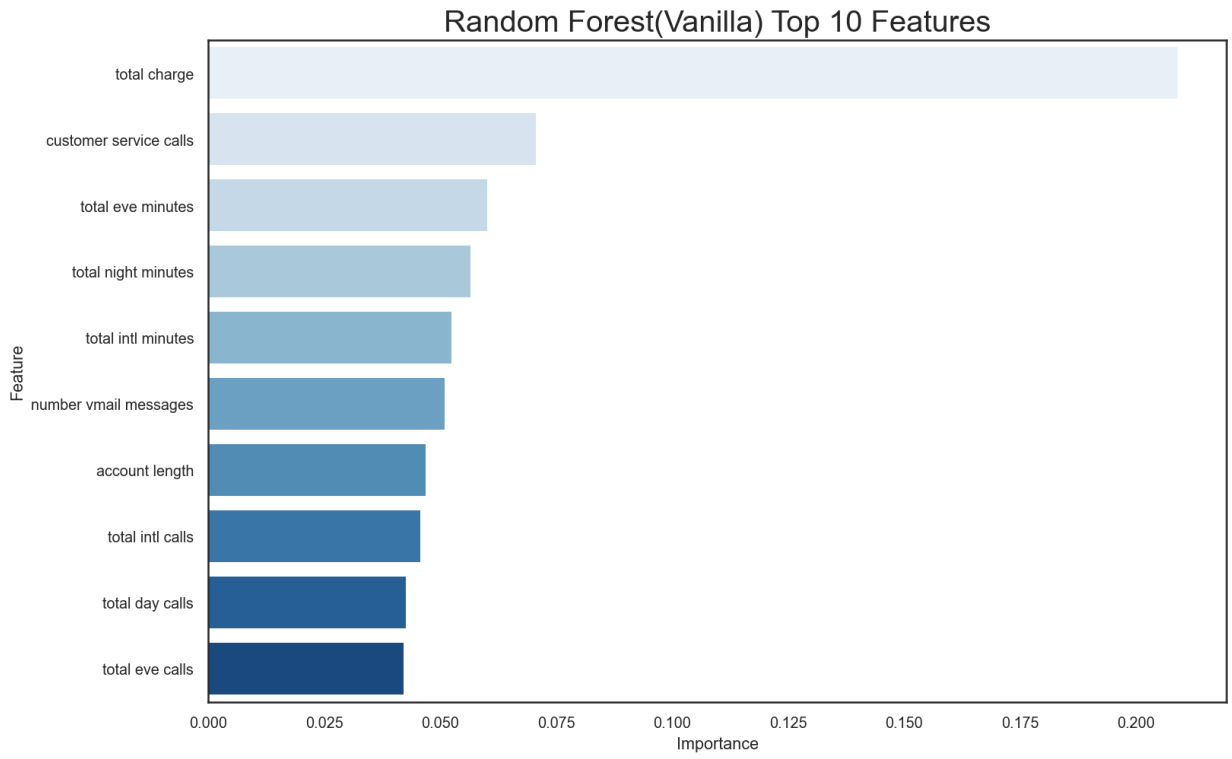
- Mean Accuracy : 95%

Model 5: Random Forest

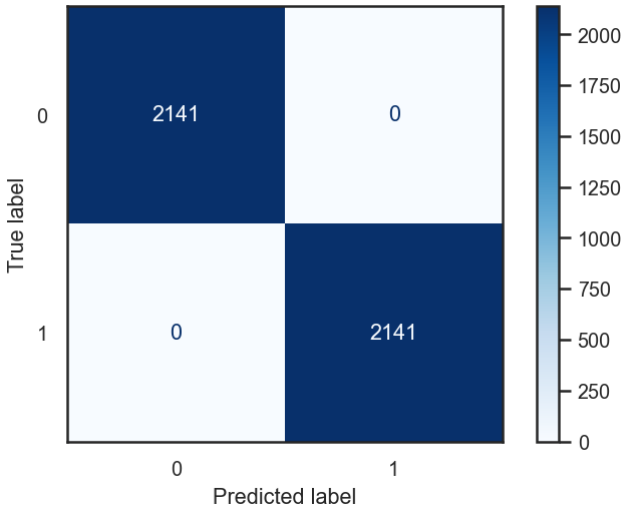
In [87]:
```
rf = RandomForestClassifier(random_state=42)
rf.fit(X_tr_sm_scaled , y_tr_sm)
```

Out[87]:
```
▾        RandomForestClassifier

RandomForestClassifier(random_state=42)
```

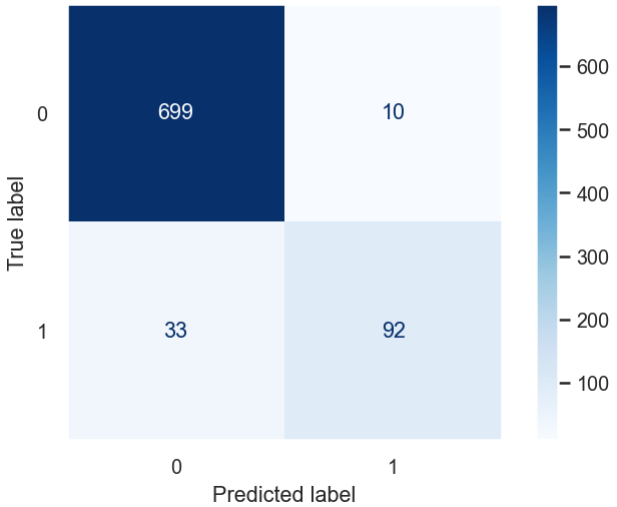
In [88]:
```
feat_import_rf = pd.DataFrame(zip(X_tr_sm_scaled.columns, rf.feature_import
                                  columns=['Feature', 'Importance'])
sns.set_style('white')
fig, ax = plt.subplots(figsize=(12,8))
sns.barplot(x='Importance', y='Feature',
            data=feat_import_rf.sort_values(by='Importance', ascending=Fals
            ax=ax, palette = 'Blues');
plt.title('Random Forest(Vanilla) Top 10 Features' , fontdict={'fontsize' :
```



In [89]:
```
plot_confusion_matrix(rf , X_tr_sm_scaled , y_tr_sm , cmap= 'Blues');
sns.set_style('dark')
```



In [90]:
```
plot_confusion_matrix(rf , X_test_scaled , y_test , cmap= 'Blues');
```



In [91]:
```
print(classification_report(y_tr_sm , rf.predict(X_tr_sm_scaled)))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2141
           1       1.00      1.00      1.00      2141

    accuracy                           1.00      4282
   macro avg       1.00      1.00      1.00      4282
weighted avg       1.00      1.00      1.00      4282
```

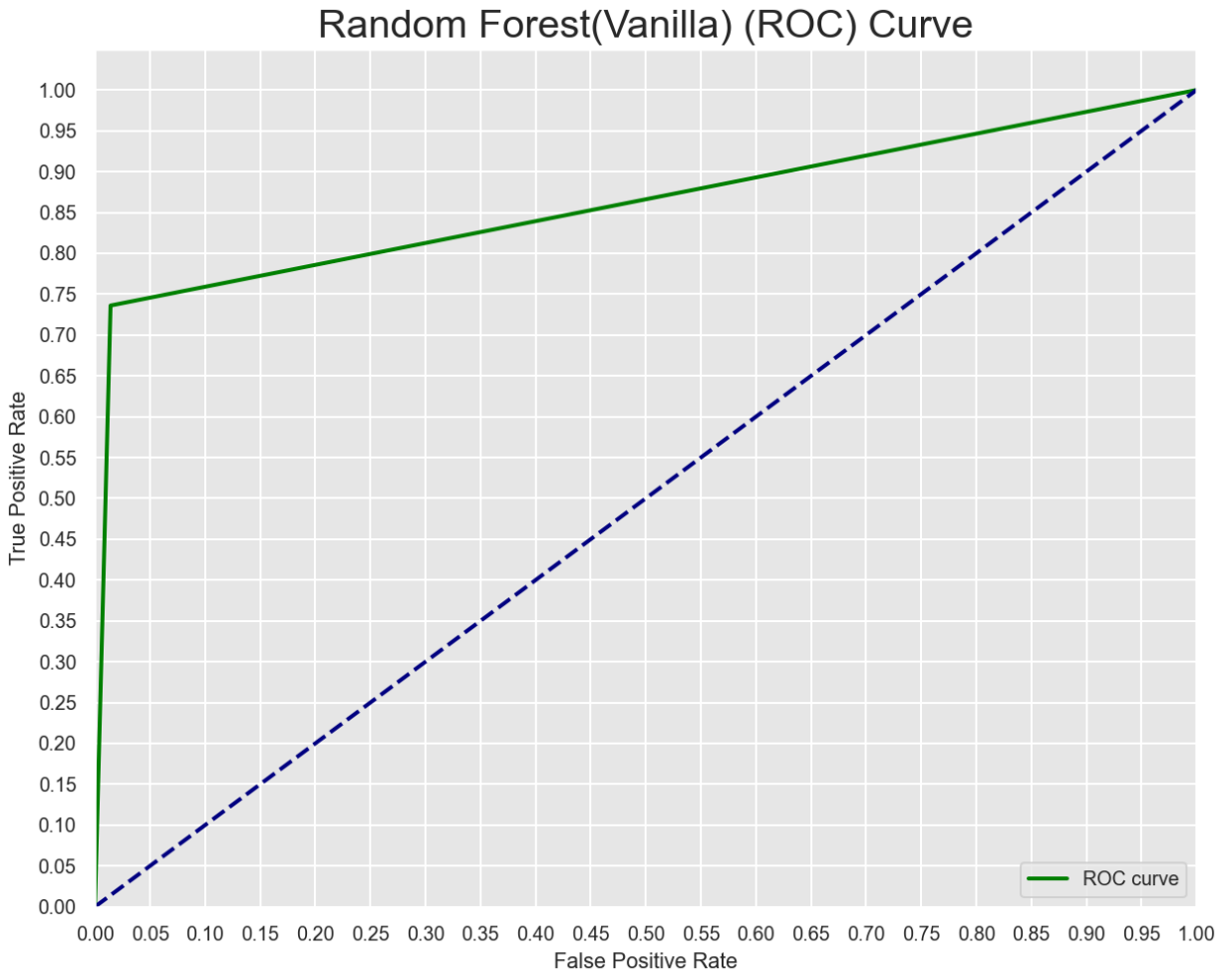
- Random Forest Model appears to be overfitting based on perfect training data results

In [92]:
```
print(classification_report(y_test , rf.predict(X_test_scaled)))
```

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       709
           1       0.90      0.74      0.81       125

    accuracy                           0.95       834
   macro avg       0.93      0.86      0.89       834
weighted avg       0.95      0.95      0.95       834
```

In [93]:
```
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})
fpr, tpr, thresholds = roc_curve(y_test, rf.predict(X_test_scaled))
print('AUC: {}'.format(auc(fpr, tpr)))
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr, tpr, color='green',
         lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest(Vanilla) (ROC) Curve' , fontdict={'fontsize': 20})
plt.legend(loc='lower right')
plt.show()
```

```
AUC: 0.8609478138222849
```



In [94]:
```
rf.score(X_test_scaled , y_test)
```

Out[94]:
```
0.9484412470023981
```

Model 5 Summary

- Recall(0n test set) : 74%
- AUC : 86%
- Mean Accuracy : 95%

Model 6 : Random Forest(Tuned Parameters)

GridSearchCV

In [95]:
```
param_grid_rf = [
   {'min_samples_leaf' : [1, 2, 4],
    'n_estimators' : [100, 200, 300],
    'min_samples_split' : [2, 5, 10],
     'max_depth' : [10, 20, 30, 40, 50]
    }]
```

In [96]:
```
clf_rf = GridSearchCV(rf , param_grid= param_grid_rf , cv = 3 , n_jobs= -1
```

In [97]:
```
clf_rf.fit(X_tr_sm_scaled , y_tr_sm)
```

```
Fitting 3 folds for each of 135 candidates, totalling 405 fits
```

Out[97]:

▸ **GridSearchCV**

▸ **estimator: RandomForestClassifier**

▸ RandomForestClassifier

In [98]:
```
clf_rf.best_estimator_
```

Out[98]:

▾ RandomForestClassifier

```
RandomForestClassifier(max_depth=40, n_estimators=300, random_state=42)
```

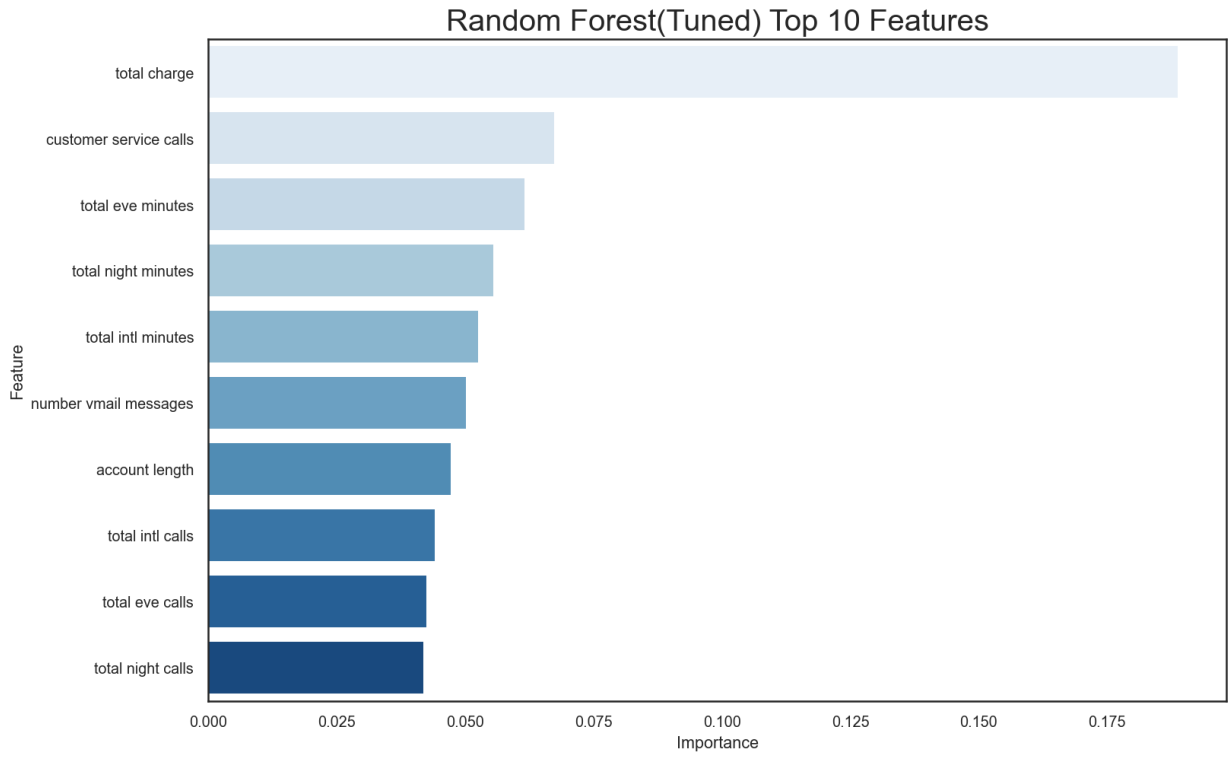
In [99]:
```
rf2 = RandomForestClassifier(max_depth= 50 , n_estimators = 400 ,max_featur

rf2.fit(X_tr_sm_scaled , y_tr_sm)
```

Out[99]:

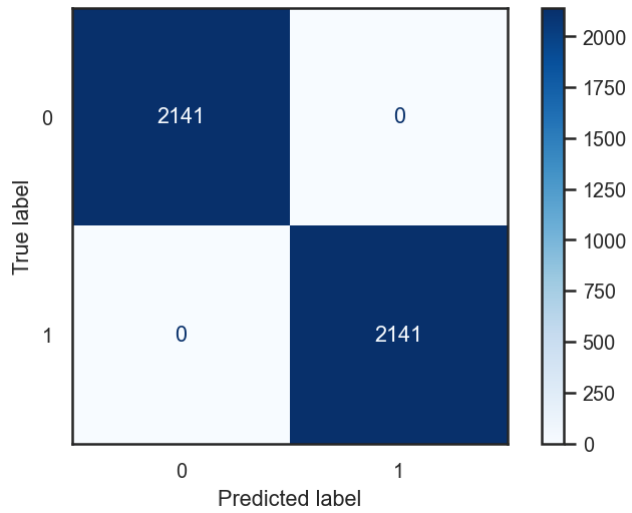
▾ RandomForestClassifier

```
RandomForestClassifier(max_depth=50, max_features='log2', n_estimators=4
00,
                       random_state=42)
```

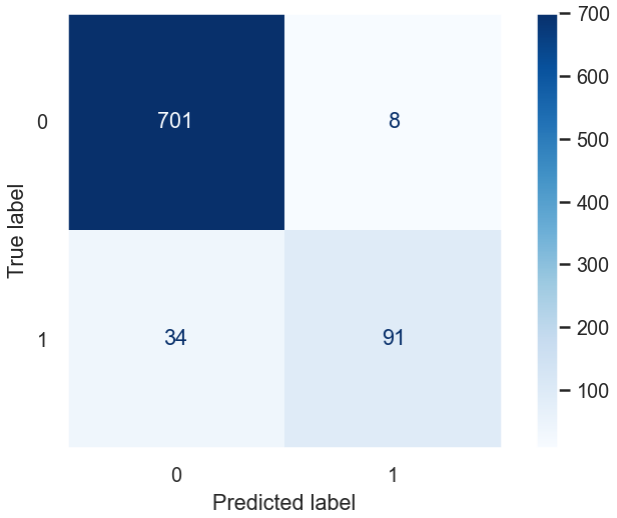
In [100]:
```
feat_import_rf2 = pd.DataFrame(zip(X_tr_sm_scaled.columns, rf2.feature_impo
                               columns=['Feature', 'Importance'])
sns.set_style('white')
fig, ax = plt.subplots(figsize=(12,8))
sns.barplot(x='Importance', y='Feature',
            data=feat_import_rf2.sort_values(by='Importance', ascending=Fal
            ax=ax, palette = 'Blues');
plt.title('Random Forest(Tuned) Top 10 Features' , fontdict={'fontsize' : 2
```



In [101]:
```
plot_confusion_matrix(rf2 , X_tr_sm_scaled , y_tr_sm , cmap = 'Blues')
sns.set_style( style = 'dark')
```



In [102]:
```
plot_confusion_matrix(rf2 , X_test_scaled , y_test , cmap = 'Blues')
sns.set_style( style = 'dark')
```



In [103]:
```
print(classification_report(y_tr_sm , rf2.predict(X_tr_sm_scaled)))
```

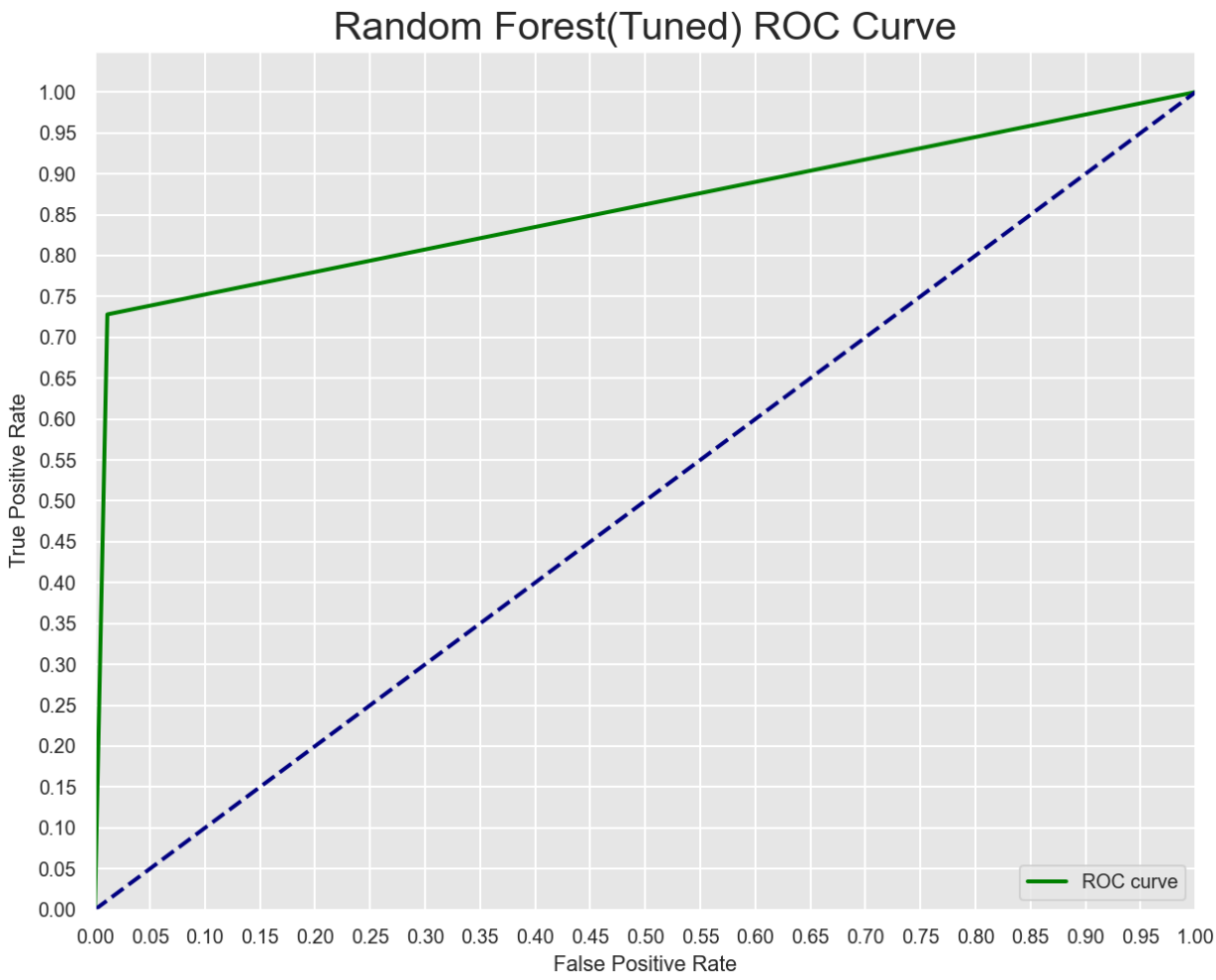
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2141
           1       1.00      1.00      1.00      2141

    accuracy                           1.00      4282
   macro avg       1.00      1.00      1.00      4282
weighted avg       1.00      1.00      1.00      4282
```

In [104]:
```
print(classification_report(y_test , rf2.predict(X_test_scaled)))
```

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       709
           1       0.92      0.73      0.81       125

    accuracy                           0.95       834
   macro avg       0.94      0.86      0.89       834
weighted avg       0.95      0.95      0.95       834
```

In [105]:
```
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})
fpr, tpr, thresholds = roc_curve(y_test, rf2.predict(X_test_scaled))
print('AUC: {}'.format(auc(fpr, tpr)))
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr, tpr, color='green',
         lw=lw, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest(Tuned) ROC Curve' , fontdict={'fontsize': 20})
plt.legend(loc='lower right')
plt.show()
```

```
AUC: 0.8583582510578279
```



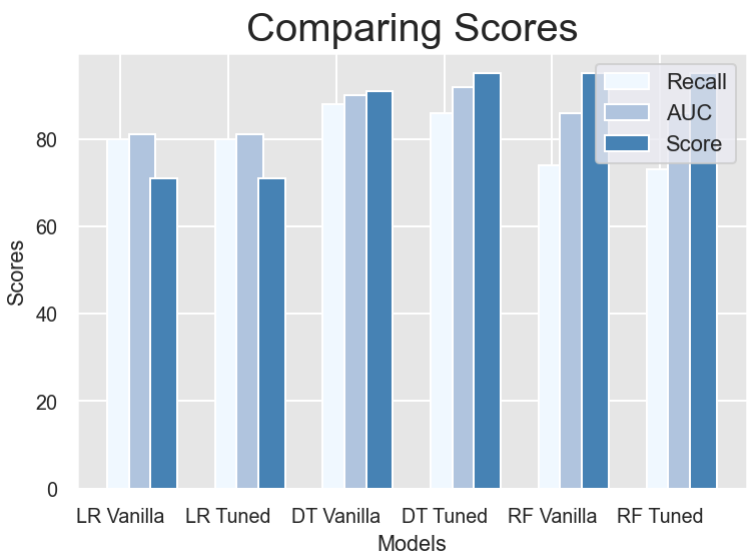
In [106]:
```
rf2.score(X_test_scaled , y_test)
```

Out[106]:
```
0.9496402877697842
```

Model 6 Summary

- Recall(0n test set) : 73%
- AUC : 86%
- Mean Accuracy : 95%

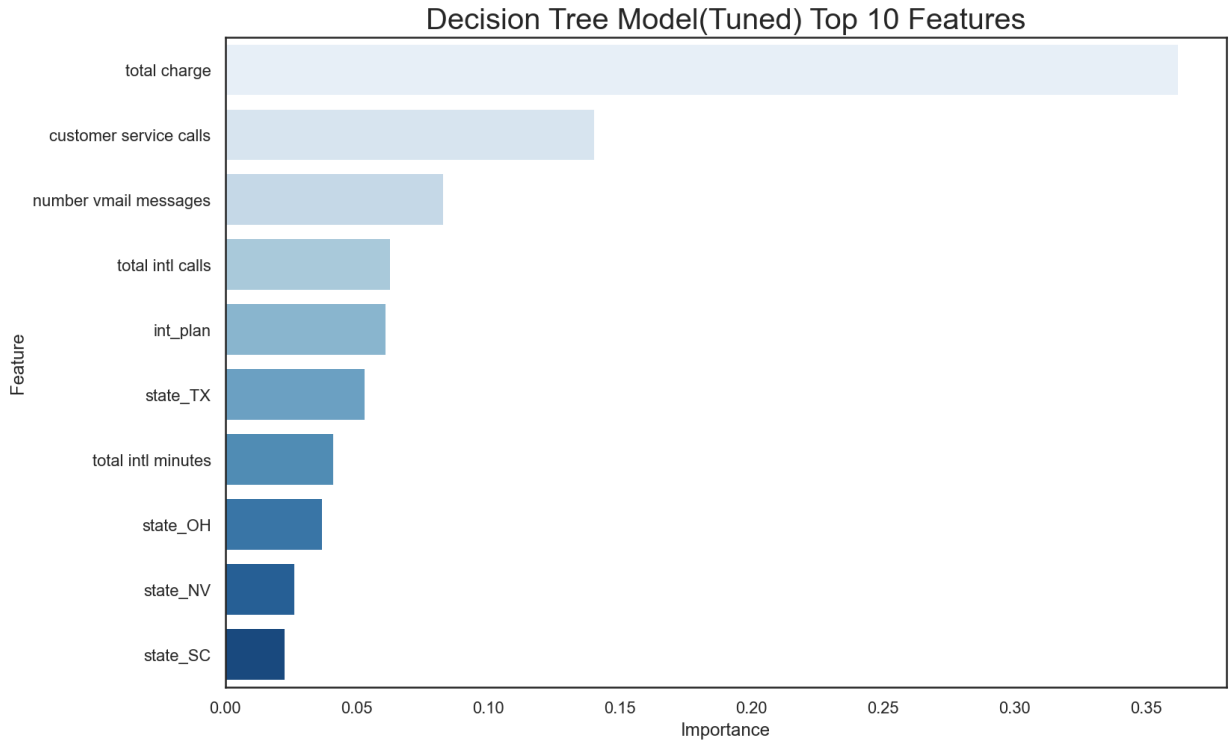
In [107]:
```
Recall = [80 , 80 , 88 , 86 , 74, 73]
AUC = [81 , 81 , 90 ,92 ,86 ,86]
Accuracy= [71 , 71 , 91 ,95 , 95, 95]
models=['LR Vanilla', 'LR Tuned','DT Vanilla','DT Tuned' , 'RF Vanilla' , '

X_axis = np.arange(len(models))

bar1 = plt.bar(X_axis , Recall, 0.25, label = 'Recall' ,color = 'aliceblue'
bar2 = plt.bar(X_axis+0.2 , AUC, 0.25, label = 'AUC', color = 'lightsteelbl
bar3 = plt.bar(X_axis+0.4 , Accuracy, 0.25, label = 'Score', color = 'steel

plt.xticks(X_axis, models)
plt.xlabel("Models")
plt.ylabel("Scores")
plt.title("Comparing Scores", fontdict={'fontsize' : 20})

sns.set(rc = {'figure.figsize':(8,8)})

plt.legend(loc = 0)
plt.show()
```



- After looking at all models it is obvious the DT Tuned model works the best as it has the highest AUC as well as other test metrics - including recall and accuracy. We would use this

DT tuned model to predict which of the given customers in the data set will churn. Now Let's check the top 10 important factor on this model.

In [108]:
```
feat_imps_df2 = pd.DataFrame(zip(X_tr_sm_scaled.columns, dt2.feature_import
                                 columns=['Feature', 'Importance'])
sns.set_style('white')
fig, ax = plt.subplots(figsize=(12,8))
sns.barplot(x='Importance', y='Feature',
            data=feat_imps_df2.sort_values(by='Importance', key=abs, ascend
           palette = 'Blues');
plt.title('Decision Tree Model(Tuned) Top 10 Features' , fontdict={'fontsiz
```



Conclusion & Recommendation

- Based on the highest score model in this notebook(Model 4: Decision Tree Tuned) I would recommend the following three business strategies to SyriaTel as a way to prevent losing customers to churn in the future.

1- We recommend the SyriaTell company to work on total charge and making promotions on this feature. We want to make sure that they get these offers to all people especially those who call customer service.

2- As the above bar suggests, the number of customer service call is the another important reason to churn. The more customer service calls, the more people leave the plan.

3- Another important factor is total internatiol calls . The Customer may not be happy for the

international service or international call charge might be high. SyriaTel should offer discount to customers especially those who make a lot of international calls

Next Step

1- We can get more competitors' datasets to see if the reason of higher churn because of the competition at these states TX , OH , NV , SC.

2- We can test on more models such as Naive Bayes , K-Nearast Neigbor and Gradient Boosting to see if those models work well.

3-We can get more data on what specific reason customers called customer service for