In [1]:

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

Classification models:

to predict weather or not a person would leave based off some input features

target: To perform the logistic, Knn, dicision tree, random forsets, some more bagging, boosting algos on Teleco-churn dataset to compare bw the them...

In [2]:

```
df = pd.read_csv("churn_logistic.csv")
```

In [3]:

```
df.head()
```

Out[3]:

	Account Length	VMail Message	Day Mins	Eve Mins	Night Mins	Intl Mins	CustServ Calls	Intl Plan	VMail Plan	Day Calls	...	Eve Calls	Eve Charge	Night Calls	Night Charge	Intl Calls	Intl Charge	State	Area Code	Phone	Chı
0	128	25	265.1	197.4	244.7	10.0	1	0	1	110	...	99	16.78	91	11.01	3	2.70	KS	415	382-4657	
1	107	26	161.6	195.5	254.4	13.7	1	0	1	123	...	103	16.62	103	11.45	3	3.70	OH	415	371-7191	
2	137	0	243.4	121.2	162.6	12.2	0	0	0	114	...	110	10.30	104	7.32	5	3.29	NJ	415	358-1921	
3	84	0	299.4	61.9	196.9	6.6	2	1	0	71	...	88	5.26	89	8.86	7	1.78	OH	408	375-9999	
4	75	0	166.7	148.3	186.9	10.1	3	1	0	113	...	122	12.61	121	8.41	3	2.73	OK	415	330-6626	

5 rows × 21 columns

In [4]:

```
df.shape
```

Out[4]:

```
(5700, 21)
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5700 entries, 0 to 5699
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Account Length  5700 non-null   int64  
 1   VMail Message   5700 non-null   int64  
 2   Day Mins        5700 non-null   float64
 3   Eve Mins        5700 non-null   float64
 4   Night Mins      5700 non-null   float64
 5   Intl Mins       5700 non-null   float64
 6   CustServ Calls  5700 non-null   int64  
 7   Intl Plan       5700 non-null   int64  
 8   VMail Plan      5700 non-null   int64  
 9   Day Calls       5700 non-null   int64  
 10  Day Charge      5700 non-null   float64
 11  Eve Calls       5700 non-null   int64  
 12  Eve Charge      5700 non-null   float64
 13  Night Calls     5700 non-null   int64  
 14  Night Charge    5700 non-null   float64
 15  Intl Calls      5700 non-null   int64  
 16  Intl Charge     5700 non-null   float64
 17  State           5700 non-null   object 
 18  Area Code       5700 non-null   int64  
 19  Phone           5700 non-null   object 
 20  Churn           5700 non-null   int64  
dtypes: float64(8), int64(11), object(2)
memory usage: 935.3+ KB
```

Obs: No missing data

Let's explore and understand the data

In [6]:

```
df
```

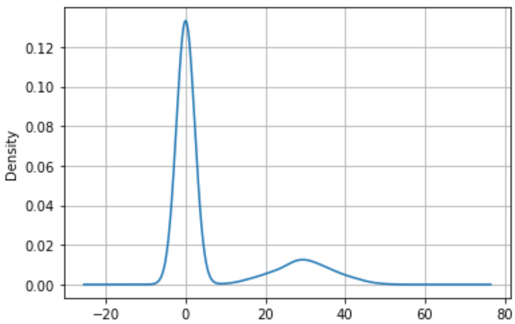
Out[6]:

	Account Length	VMail Message	Day Mins	Eve Mins	Night Mins	Intl Mins	CustServ Calls	Intl Plan	VMail Plan	Day Calls	...	Eve Calls	Eve Charge	Night Calls	Night Charge	Intl Calls	Intl Charge	State	Area Code	Phone
0	128	25	265.1	197.4	244.7	10.0	1	0	1	110	...	99	16.78	91	11.01	3	2.70	KS	415	382-4657
1	107	26	161.6	195.5	254.4	13.7	1	0	1	123	...	103	16.62	103	11.45	3	3.70	OH	415	371-7191
2	137	0	243.4	121.2	162.6	12.2	0	0	0	114	...	110	10.30	104	7.32	5	3.29	NJ	415	358-1921
3	84	0	299.4	61.9	196.9	6.6	2	1	0	71	...	88	5.26	89	8.86	7	1.78	OH	408	375-9999
4	75	0	166.7	148.3	186.9	10.1	3	1	0	113	...	122	12.61	121	8.41	3	2.73	OK	415	330-6626
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5695	224	0	171.5	160.0	212.4	5.0	1	1	0	99	...	103	13.60	102	9.56	2	1.35	DE	510	361-6563
5696	131	0	131.6	179.3	251.2	15.5	1	0	0	95	...	109	15.24	129	11.30	3	4.19	MS	415	333-9002
5697	132	0	291.2	234.2	191.7	8.9	1	0	0	104	...	132	19.91	87	8.63	3	2.40	MI	408	389-4608
5698	100	0	113.3	197.9	284.5	11.7	4	0	0	96	...	89	16.82	93	12.80	2	3.16	MT	415	341-4873
5699	147	0	274.0	231.8	283.6	6.2	0	0	0	92	...	82	19.70	83	12.76	1	1.67	MD	408	376-4292

5700 rows × 21 columns

In [7]:

```
df['VMail Message'].plot(kind='kde')
plt.grid()
```



we have a lot of users send only zero voice message!

In [8]:

```
df
```

Out[8]:

	Account Length	VMail Message	Day Mins	Eve Mins	Night Mins	Intl Mins	CustServ Calls	Intl Plan	VMail Plan	Day Calls	...	Eve Calls	Eve Charge	Night Calls	Night Charge	Intl Calls	Intl Charge	State	Area Code	Phone
0	128	25	265.1	197.4	244.7	10.0	1	0	1	110	...	99	16.78	91	11.01	3	2.70	KS	415	382-4657
1	107	26	161.6	195.5	254.4	13.7	1	0	1	123	...	103	16.62	103	11.45	3	3.70	OH	415	371-7191
2	137	0	243.4	121.2	162.6	12.2	0	0	0	114	...	110	10.30	104	7.32	5	3.29	NJ	415	358-1921
3	84	0	299.4	61.9	196.9	6.6	2	1	0	71	...	88	5.26	89	8.86	7	1.78	OH	408	375-9999
4	75	0	166.7	148.3	186.9	10.1	3	1	0	113	...	122	12.61	121	8.41	3	2.73	OK	415	330-6626
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5695	224	0	171.5	160.0	212.4	5.0	1	1	0	99	...	103	13.60	102	9.56	2	1.35	DE	510	361-6563
5696	131	0	131.6	179.3	251.2	15.5	1	0	0	95	...	109	15.24	129	11.30	3	4.19	MS	415	333-9002
5697	132	0	291.2	234.2	191.7	8.9	1	0	0	104	...	132	19.91	87	8.63	3	2.40	MI	408	389-4608
5698	100	0	113.3	197.9	284.5	11.7	4	0	0	96	...	89	16.82	93	12.80	2	3.16	MT	415	341-4873
5699	147	0	274.0	231.8	283.6	6.2	0	0	0	92	...	82	19.70	83	12.76	1	1.67	MD	408	376-4292

5700 rows × 21 columns

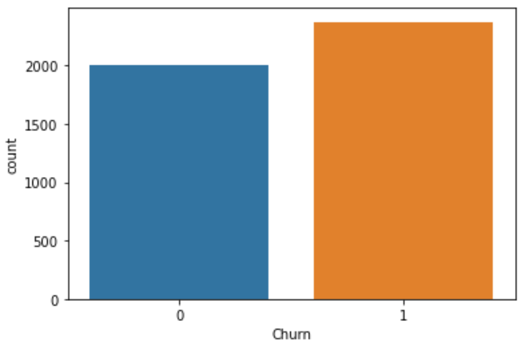
In [9]:

```
sns.countplot(df[df['VMail Message'] == 0].Churn)
```

```
C:\Users\Ashutosh Patidar\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments with
out an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[9]:

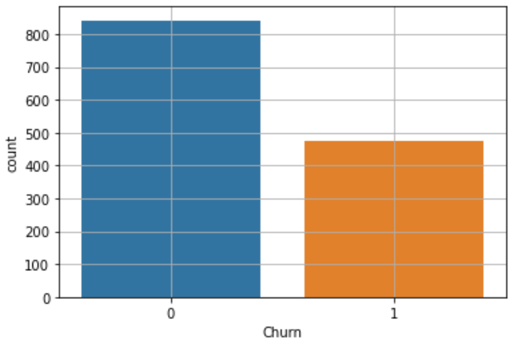
```
<AxesSubplot:xlabel='Churn', ylabel='count'>
```



In [10]:

```
sns.countplot(df[df['VMail Message'] != 0].Churn)
plt.grid()
```

```
C:\Users\Ashutosh Patidar\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments with
out an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



not significant impact on churning!

In [11]:

```
df
```

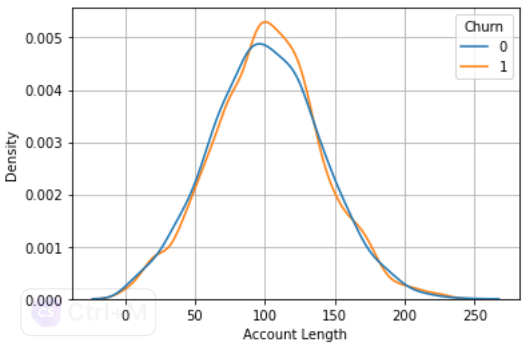
Out[11]:

	Account Length	VMail Message	Day Mins	Eve Mins	Night Mins	Intl Mins	CustServ Calls	Intl Plan	VMail Plan	Day Calls	...	Eve Calls	Eve Charge	Night Calls	Night Charge	Intl Calls	Intl Charge	State	Area Code	Phone
0	128	25	265.1	197.4	244.7	10.0	1	0	1	110	...	99	16.78	91	11.01	3	2.70	KS	415	382-4657
1	107	26	161.6	195.5	254.4	13.7	1	0	1	123	...	103	16.62	103	11.45	3	3.70	OH	415	371-7191
2	137	0	243.4	121.2	162.6	12.2	0	0	0	114	...	110	10.30	104	7.32	5	3.29	NJ	415	358-1921
3	84	0	299.4	61.9	196.9	6.6	2	1	0	71	...	88	5.26	89	8.86	7	1.78	OH	408	375-9999
4	75	0	166.7	148.3	186.9	10.1	3	1	0	113	...	122	12.61	121	8.41	3	2.73	OK	415	330-6626
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5695	224	0	171.5	160.0	212.4	5.0	1	1	0	99	...	103	13.60	102	9.56	2	1.35	DE	510	361-6563
5696	131	0	131.6	179.3	251.2	15.5	1	0	0	95	...	109	15.24	129	11.30	3	4.19	MS	415	333-9002
5697	132	0	291.2	234.2	191.7	8.9	1	0	0	104	...	132	19.91	87	8.63	3	2.40	MI	408	389-4608
5698	100	0	113.3	197.9	284.5	11.7	4	0	0	96	...	89	16.82	93	12.80	2	3.16	MT	415	341-4873
5699	147	0	274.0	231.8	283.6	6.2	0	0	0	92	...	82	19.70	83	12.76	1	1.67	MD	408	376-4292

5700 rows × 21 columns

In [12]:

```
sns.kdeplot(df['Account Length'],hue = df['Churn'])
plt.grid()
plt.show()
```



Obs: no significant impact of account length on churning

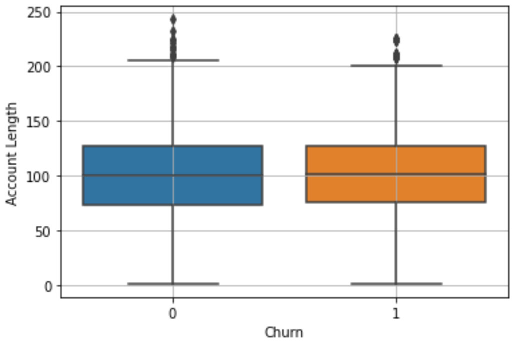
In [13]:

```
import warnings

warnings.filterwarnings('ignore')
```

In [14]:

```
sns.boxplot(x = df['Churn'], y = df['Account Length'])
plt.grid()
```



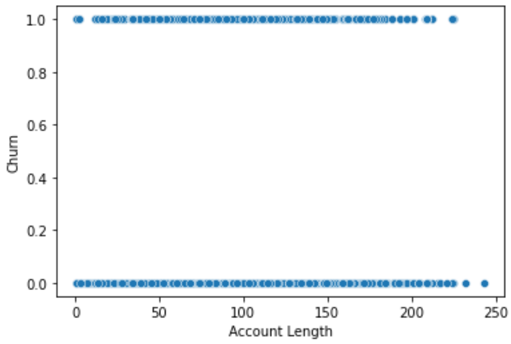
Obs: outliers detected

In [15]:

```
sns.scatterplot(y = df['Churn'], x = df['Account Length'])
```

Out[15]:

```
<AxesSubplot:xlabel='Account Length', ylabel='Churn'>
```



let's remove the outliers!

In [16]:

```
iqr = np.percentile(df['Account Length'], 75) - np.percentile(df['Account Length'], 25)
mid = np.percentile(df['Account Length'], 50)

df[df['Account Length'] > mid + 1.5 * iqr]['Account Length'].size
```

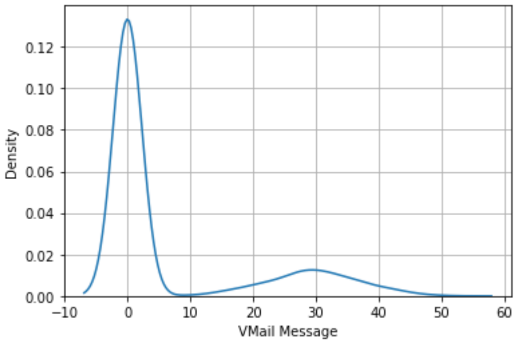
Out[16]:

```
150
```

Account Length has 150 outliers

In [17]:

```
sns.kdeplot(df['VMail Message'])
plt.grid()
```



In [18]:

```
df
```

Out[18]:

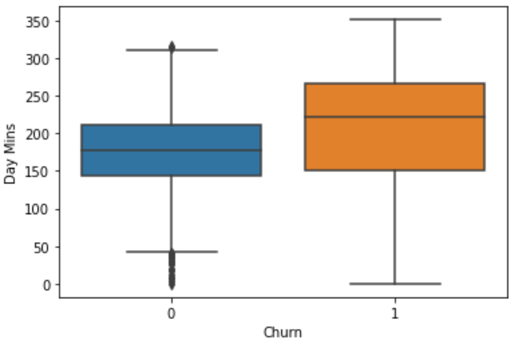
	Account Length	VMail Message	Day Mins	Eve Mins	Night Mins	Intl Mins	CustServ Calls	Intl Plan	VMail Plan	Day Calls	...	Eve Calls	Eve Charge	Night Calls	Night Charge	Intl Calls	Intl Charge	State	Area Code	Phone	Churn
0	128	25	265.1	197.4	244.7	10.0	1	0	1	110	...	99	16.78	91	11.01	3	2.70	KS	415	382-4657	0
1	107	26	161.6	195.5	254.4	13.7	1	0	1	123	...	103	16.62	103	11.45	3	3.70	OH	415	371-7191	0
2	137	0	243.4	121.2	162.6	12.2	0	0	0	114	...	110	10.30	104	7.32	5	3.29	NJ	415	358-1921	0
3	84	0	299.4	61.9	196.9	6.6	2	1	0	71	...	88	5.26	89	8.86	7	1.78	OH	408	375-9999	0
4	75	0	166.7	148.3	186.9	10.1	3	1	0	113	...	122	12.61	121	8.41	3	2.73	OK	415	330-6626	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5695	224	0	171.5	160.0	212.4	5.0	1	1	0	99	...	103	13.60	102	9.56	2	1.35	DE	510	361-	1

In [19]:

```
sns.boxplot(x = df.Churn, y = df['Day Mins'])
```

Out[19]:

<AxesSubplot:xlabel='Churn', ylabel='Day Mins'>

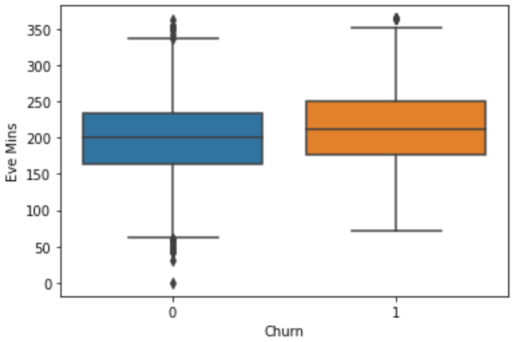


In [20]:

```
sns.boxplot(x = df.Churn, y = df['Eve Mins'])
```

Out[20]:

```
<AxesSubplot:xlabel='Churn', ylabel='Eve Mins'>
```

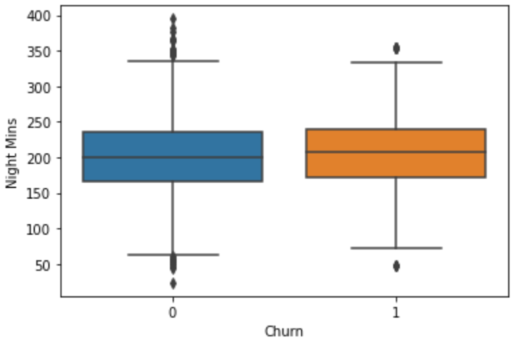


In [21]:

```
sns.boxplot(x = df.Churn, y = df['Night Mins'])
```

Out[21]:

```
<AxesSubplot:xlabel='Churn', ylabel='Night Mins'>
```

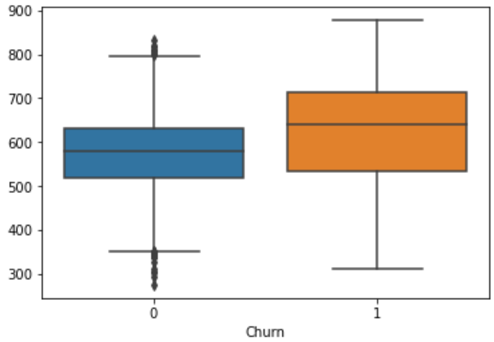


In [22]:

```
total_mins = df['Day Mins'] + df['Eve Mins'] + df['Night Mins']
sns.boxplot(x = df.Churn, y = total_mins)
```

Out[22]:

```
<AxesSubplot:xlabel='Churn'>
```

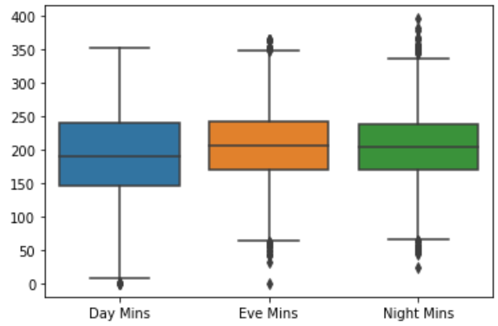


In [23]:

```
df_sel = df[['Day Mins', 'Eve Mins', 'Night Mins']]
sns.boxplot(data = df_sel)
```

Out[23]:

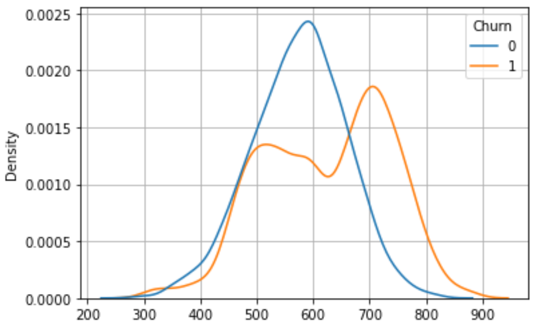
```
<AxesSubplot:>
```



Obs: outliers detected! and talking mins isn't significantly impacting weather or not a person churn

In [24]:

```
sns.kdeplot(total_mins , hue=df.Churn)
plt.grid()
```

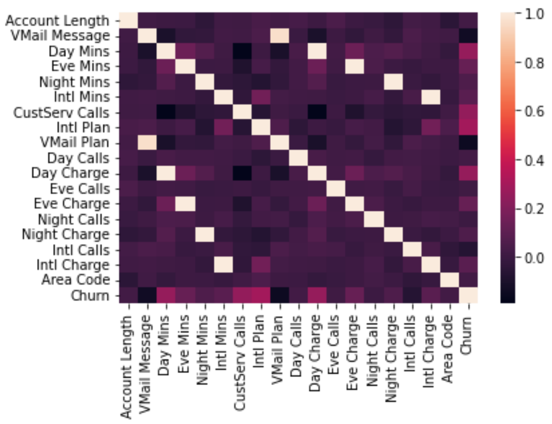


In [25]:

```
sns.heatmap(df.corr())
```

Out[25]:

```
<AxesSubplot:>
```



In [26]:

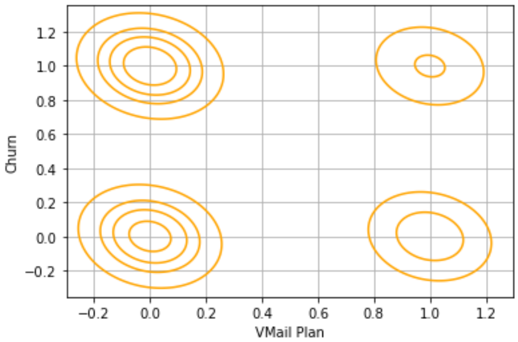
```
from sklearn.metrics import confusion_matrix
confusion_matrix(df['VMail Plan'], df['Churn'])
```

Out[26]:

```
array([[2008, 2374],
       [ 842,  476]], dtype=int64)
```

In [27]:

```
sns.kdeplot(df['VMail Plan'], df['Churn'], levels=5, color='orange')
plt.grid()
```



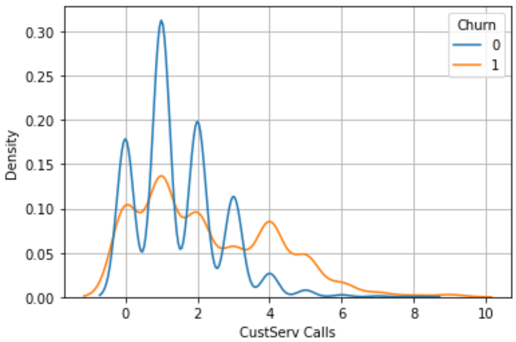
by observing above matrix we can clearly see that the persons not taking the vmail plans are churning the most, which don't make any sense hence vmail can be also considered unnecessary

In [28]:

```
df.drop(['Phone'], axis = 1, inplace = True)
```

In [29]:

```
sns.kdeplot(df['CustServ Calls'], hue = df.Churn)
plt.grid()
```



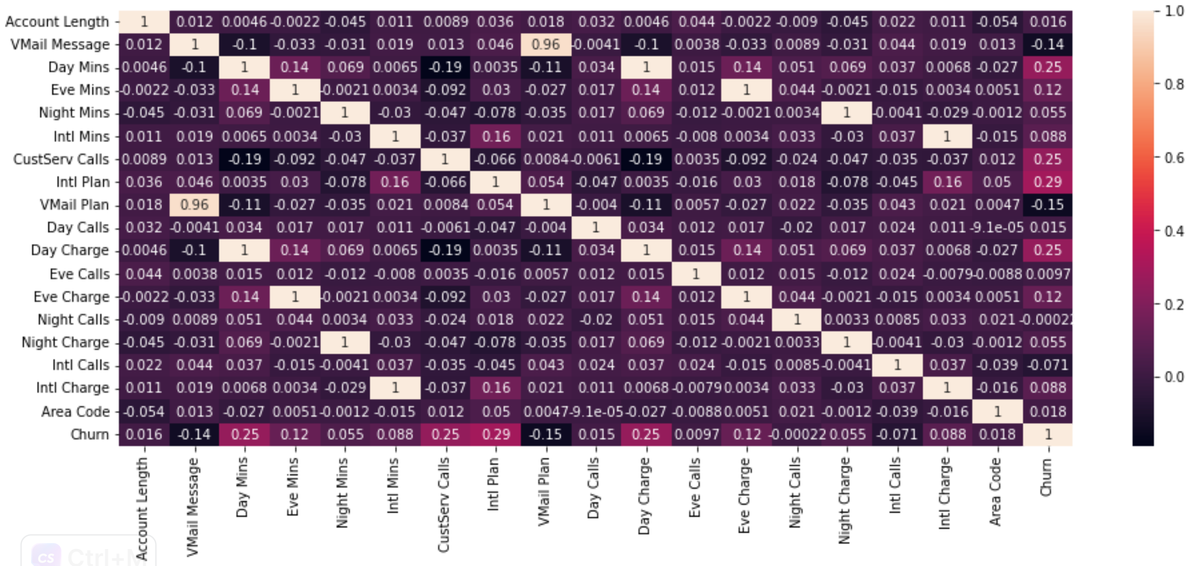
obs: the persons who does a little higher no of customer calls are likely to churn: hence it's a good feature!

In [30]:

```
plt.figure(figsize=(16,6))
sns.heatmap(df.corr(), annot=True)
```

Out[30]:

```
<AxesSubplot:>
```



also we can notice that CustServ calls has a comparatively good depencency with the curining of the customers... with correlation = 0.29

In [31]:

```
df
```

Out[31]:

	Account Length	VMail Message	Day Mins	Eve Mins	Night Mins	Intl Mins	CustServ Calls	Intl Plan	VMail Plan	Day Calls	Day Charge	Eve Calls	Eve Charge	Night Calls	Night Charge	Intl Calls	Intl Charge	State	Area Code	Ch
0	128	25	265.1	197.4	244.7	10.0	1	0	1	110	45.07	99	16.78	91	11.01	3	2.70	KS	415	
1	107	26	161.6	195.5	254.4	13.7	1	0	1	123	27.47	103	16.62	103	11.45	3	3.70	OH	415	
2	137	0	243.4	121.2	162.6	12.2	0	0	0	114	41.38	110	10.30	104	7.32	5	3.29	NJ	415	
3	84	0	299.4	61.9	196.9	6.6	2	1	0	71	50.90	88	5.26	89	8.86	7	1.78	OH	408	
4	75	0	166.7	148.3	186.9	10.1	3	1	0	113	28.34	122	12.61	121	8.41	3	2.73	OK	415	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
5695	224	0	171.5	160.0	212.4	5.0	1	1	0	99	29.16	103	13.60	102	9.56	2	1.35	DE	510	
5696	131	0	131.6	179.3	251.2	15.5	1	0	0	95	22.37	109	15.24	129	11.30	3	4.19	MS	415	
5697	132	0	291.2	234.2	191.7	8.9	1	0	0	104	49.50	132	19.91	87	8.63	3	2.40	MI	408	
5698	100	0	113.3	197.9	284.5	11.7	4	0	0	96	19.26	89	16.82	93	12.80	2	3.16	MT	415	
5699	147	0	274.0	231.8	283.6	6.2	0	0	0	92	46.58	82	19.70	83	12.76	1	1.67	MD	408	

5700 rows × 20 columns

In [32]:

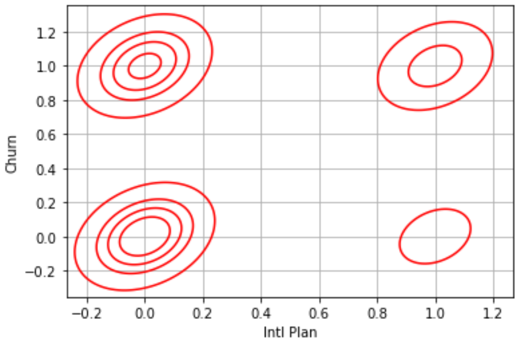
```
from sklearn.metrics import confusion_matrix
confusion_matrix(df['Intl Plan'], df['Churn'])
```

Out[32]:

```
array([[2664, 2032],
       [ 186,  818]], dtype=int64)
```

In [33]:

```
sns.kdeplot(df['Intl Plan'], df['Churn'], levels=5, color='red')
plt.grid()
```



In [34]:

```
# sns.kdeplot(df['Churn'], df['Account Length'], levels=5, color='red')
```

In [35]:

```
df
```

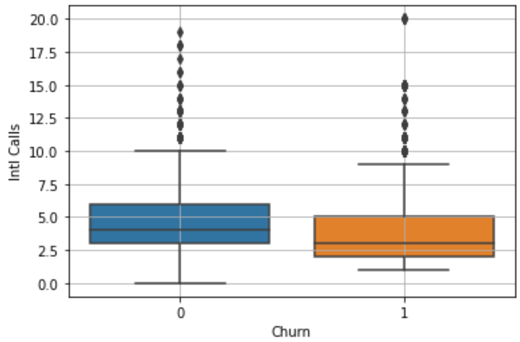
Out[35]:

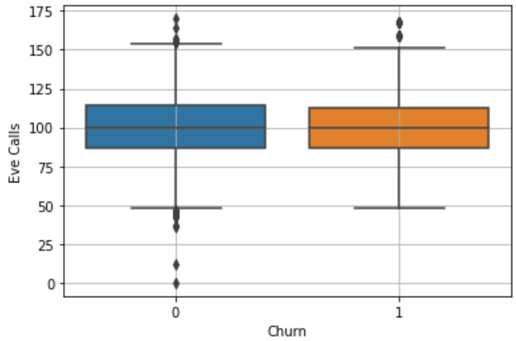
	Account Length	VMail Message	Day Mins	Eve Mins	Night Mins	Intl Mins	CustServ Calls	Intl Plan	VMail Plan	Day Calls	Day Charge	Eve Calls	Eve Charge	Night Calls	Night Charge	Intl Calls	Intl Charge	State	Area Code	Ch
0	128	25	265.1	197.4	244.7	10.0	1	0	1	110	45.07	99	16.78	91	11.01	3	2.70	KS	415	
1	107	26	161.6	195.5	254.4	13.7	1	0	1	123	27.47	103	16.62	103	11.45	3	3.70	OH	415	
2	137	0	243.4	121.2	162.6	12.2	0	0	0	114	41.38	110	10.30	104	7.32	5	3.29	NJ	415	
3	84	0	299.4	61.9	196.9	6.6	2	1	0	71	50.90	88	5.26	89	8.86	7	1.78	OH	408	
4	75	0	166.7	148.3	186.9	10.1	3	1	0	113	28.34	122	12.61	121	8.41	3	2.73	OK	415	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
5695	224	0	171.5	160.0	212.4	5.0	1	1	0	99	29.16	103	13.60	102	9.56	2	1.35	DE	510	
5696	131	0	131.6	179.3	251.2	15.5	1	0	0	95	22.37	109	15.24	129	11.30	3	4.19	MS	415	
5697	132	0	291.2	234.2	191.7	8.9	1	0	0	104	49.50	132	19.91	87	8.63	3	2.40	MI	408	
5698	100	0	113.3	197.9	284.5	11.7	4	0	0	96	19.26	89	16.82	93	12.80	2	3.16	MT	415	
5699	147	0	274.0	231.8	283.6	6.2	0	0	0	92	46.58	82	19.70	83	12.76	1	1.67	MD	408	

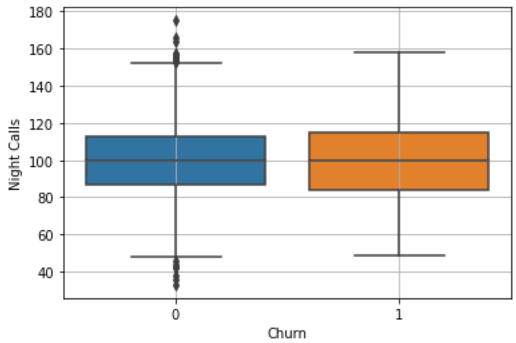
5700 rows × 20 columns

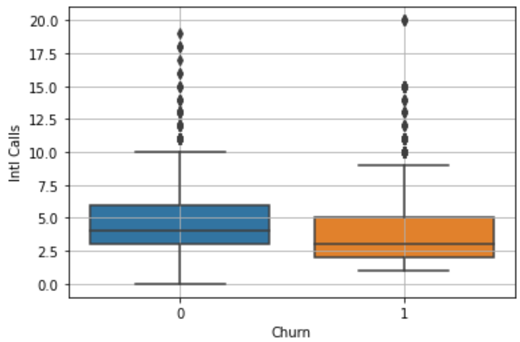
In [36]:

```
derived = df['Day Calls'] + df['Eve Calls'] + df['Night Calls'] + df['Intl Calls']
sns.boxplot(x = df.Churn, y =  df['Intl Calls'] )
plt.grid()
plt.show()
sns.boxplot(x = df.Churn, y =  df['Eve Calls'] )
plt.grid()
plt.show()
sns.boxplot(x = df.Churn, y =  df['Night Calls'] )
plt.grid()
plt.show()
sns.boxplot(x = df.Churn, y =   df['Intl Calls'] )
plt.grid()
plt.show()
sns.boxplot(x = df.Churn, y =   derived )
```



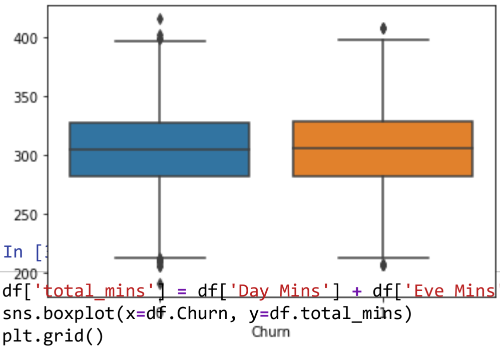






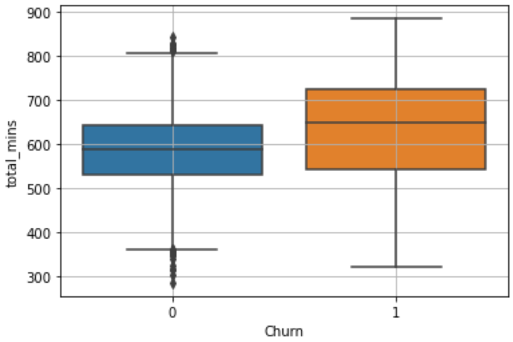
Out[36]:

```
<AxesSubplot:xlabel='Churn'>
```



In [

```
df['total_mins'] = df['Day Mins'] + df['Eve Mins'] + df['Night Mins'] + df['Intl Mins']
sns.boxplot(x=df.Churn, y=df.total_mins)
plt.grid()
```



significant feature

removing some of the unnecessary columns based off above observations

In [38]:

```
df.columns
```

Out[38]:

```
Index(['Account Length', 'VMail Message', 'Day Mins', 'Eve Mins', 'Night Mins',
       'Intl Mins', 'CustServ Calls', 'Intl Plan', 'VMail Plan', 'Day Calls',
       'Day Charge', 'Eve Calls', 'Eve Charge', 'Night Calls', 'Night Charge',
       'Intl Calls', 'Intl Charge', 'State', 'Area Code', 'Churn',
       'total_mins'],
      dtype='object')
```

In [39]:

```
df.drop(['Day Mins', 'Eve Mins', 'Night Mins',
'Intl Mins','Day Calls',
       'Day Charge', 'Eve Calls', 'Eve Charge', 'Night Calls', 'Night Charge', 'Intl Charge'], axis = 1, inplace = True)
df.head()
```

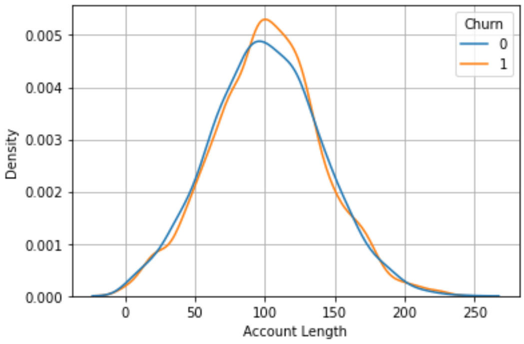
Out[39]:

	Account Length	VMail Message	CustServ Calls	Intl Plan	VMail Plan	Intl Calls	State	Area Code	Churn	total_mins
0	128	25	1	0	1	3	KS	415	0	717.2
1	107	26	1	0	1	3	OH	415	0	625.2
2	137	0	0	0	0	5	NJ	415	0	539.4
3	84	0	2	1	0	7	OH	408	0	564.8
4	75	0	3	1	0	3	OK	415	0	512.0

looking for anymore feature having no impact of churning of the customer

In [40]:

```
sns.kdeplot(df['Account Length'], hue = df.Churn )
plt.grid()
```



obs: account length has no significant impact on the curning of the customer

In [41]:

```
df.drop(columns=['Account Length'], inplace = True)
df.head()
```

Out[41]:

	VMail Message	CustServ Calls	Intl Plan	VMail Plan	Intl Calls	State	Area Code	Churn	total_mins
0	25	1	0	1	3	KS	415	0	717.2
1	26	1	0	1	3	OH	415	0	625.2
2	0	0	0	0	5	NJ	415	0	539.4
3	0	2	1	0	7	OH	408	0	564.8
4	0	3	1	0	3	OK	415	0	512.0

In [42]:

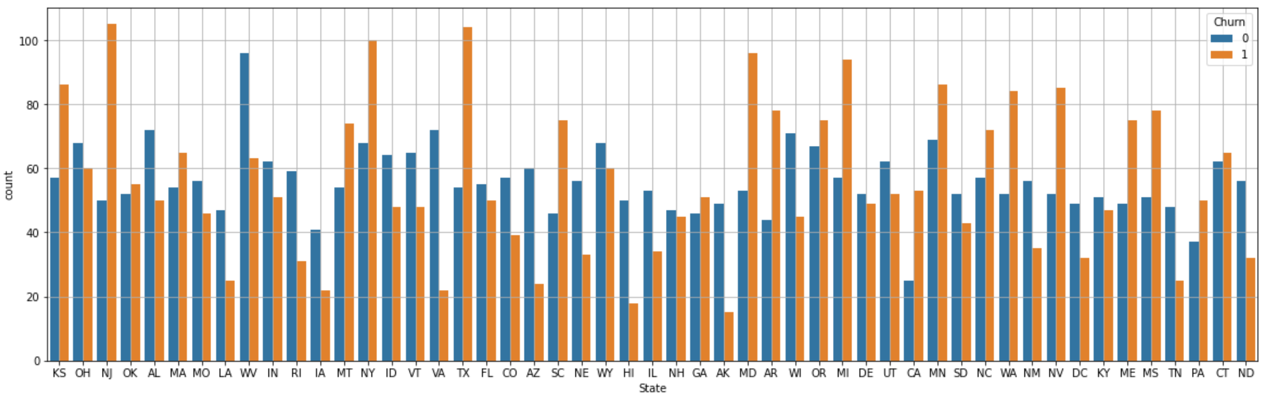
```
df['State'].unique()
```

Out[42]:

```
array(['KS', 'OH', 'NJ', 'OK', 'AL', 'MA', 'MO', 'LA', 'WV', 'IN', 'RI',
       'IA', 'MT', 'NY', 'ID', 'VT', 'VA', 'TX', 'FL', 'CO', 'AZ', 'SC',
       'NE', 'WY', 'HI', 'IL', 'NH', 'GA', 'AK', 'MD', 'AR', 'WI', 'OR',
       'MI', 'DE', 'UT', 'CA', 'MN', 'SD', 'NC', 'WA', 'NM', 'NV', 'DC',
       'KY', 'ME', 'MS', 'TN', 'PA', 'CT', 'ND'], dtype=object)
```

In [43]:

```
plt.figure(figsize=(20,6))
sns.countplot(df.State, hue = df.Churn)
plt.grid()
```



we need to do target mean encoding for the state feature

In [44]:

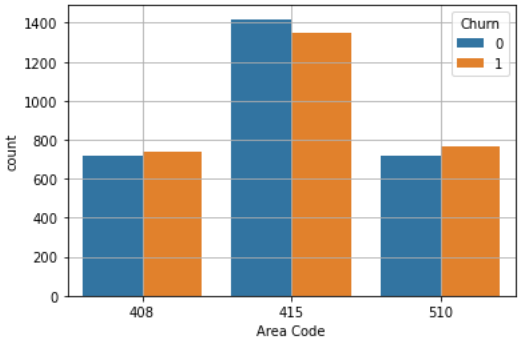
```
df.head()
```

Out[44]:

	VMail Message	CustServ Calls	Intl Plan	VMail Plan	Intl Calls	State	Area Code	Churn	total_mins
0	25	1	0	1	3	KS	415	0	717.2
1	26	1	0	1	3	OH	415	0	625.2
2	0	0	0	0	5	NJ	415	0	539.4
3	0	2	1	0	7	OH	408	0	564.8
4	0	3	1	0	3	OK	415	0	512.0

In [45]:

```
sns.countplot(df['Area Code'], hue = df.Churn)
plt.grid()
```



no significant effect of area code on churing of the customer

to perform EDA and check for multicoliniarity, outliers, encodings and starnge behaviour in the data

In [46]:

```
df.drop(columns=['Area Code'], inplace=True)
```

In [47]:

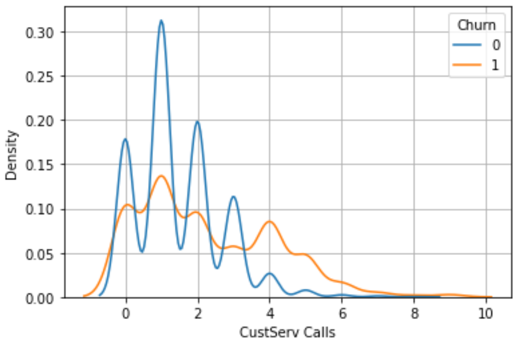
```
df.head()
```

Out[47]:

	VMail Message	CustServ Calls	Intl Plan	VMail Plan	Intl Calls	State	Churn	total_mins
0	25	1	0	1	3	KS	0	717.2
1	26	1	0	1	3	OH	0	625.2
2	0	0	0	0	5	NJ	0	539.4
3	0	2	1	0	7	OH	0	564.8
4	0	3	1	0	3	OK	0	512.0

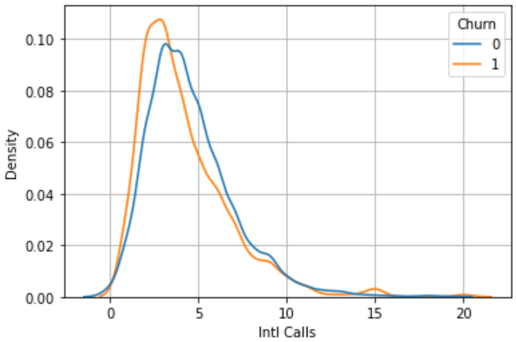
In [48]:

```
sns.kdeplot(df['CustServ Calls'], hue = df.Churn)
plt.grid()
```



In [49]:

```
sns.kdeplot(df['Intl Calls'], hue = df.Churn)
plt.grid()
```



let's perform the target mean endcoding on the state feature

In [50]:

```
df.head()
```

Out[50]:

	VMail Message	CustServ Calls	Intl Plan	VMail Plan	Intl Calls	State	Churn	total_mins
0	25	1	0	1	3	KS	0	717.2
1	26	1	0	1	3	OH	0	625.2
2	0	0	0	0	5	NJ	0	539.4
3	0	2	1	0	7	OH	0	564.8
4	0	3	1	0	3	OK	0	512.0

In [51]:

```
dct = {}
for i in df['State'].unique():
    dct[i] = len(df[df.State == i].Churn == 1)
dct
```

Out[51]:

```
{'KS': 143,
 'OH': 128,
 'NJ': 155,
 'OK': 107,
 'AL': 122,
 'MA': 119,
 'MO': 102,
 'LA': 72,
 'WV': 159,
 'IN': 113,
 'RI': 90,
 'IA': 63,
 'MT': 128,
 'NY': 168,
 'ID': 112,
 'VT': 113,
 'VA': 94,
 'TX': 158,
 'FL': 105,
 'CO': 96,
 'AZ': 84,
 'SC': 121,
 'NE': 89,
 'WY': 128,
 'HI': 68,
 'IL': 87,
 'NH': 92,
 'GA': 97,
 'AK': 64,
 'MD': 149,
 'AR': 122,
 'WI': 116,
 'OR': 142,
 'MI': 151,
 'DE': 101,
 'UT': 114,
 'CA': 78,
 'MN': 155,
 'SD': 95,
 'NC': 129,
 'WA': 136,
 'NM': 91,
 'NV': 137,
 'DC': 81,
 'KY': 98,
 'ME': 124,
 'MS': 129,
 'TN': 73,
 'PA': 87,
 'CT': 127,
 'ND': 88}
```

replacing the value

In [52]:

```
new = []
for i in df.State:
    new.append(dct[i])
new
```

Out[52]:

```
[143,
 128,
 155,
 128,
 107,
 122,
 119,
 102,
 72,
 159,
 113,
 90,
 63,
 128,
 63,
 168,
 112,
 113,
```

In [53]:

```
print(len(dct))
print(len(new))
```

```
51
5700
```

In [54]:

```
new = pd.DataFrame(new)
new['State_enc'] = new
# new.drop(columns=[')'])
new.head()
```

Out[54]:

	0	State_enc
0	143	143
1	128	128
2	155	155
3	128	128
4	107	107

In [55]:

```
df['State_enc'] = new['State_enc']
df.columns
```

Out[55]:

```
Index(['VMail Message', 'CustServ Calls', 'Intl Plan', 'VMail Plan',
       'Intl Calls', 'State', 'Churn', 'total_mins', 'State_enc'],
      dtype='object')
```

In [56]:

```
df.head()
```

Out[56]:

	VMail Message	CustServ Calls	Intl Plan	VMail Plan	Intl Calls	State	Churn	total_mins	State_enc
0	25	1	0	1	3	KS	0	717.2	143
1	26	1	0	1	3	OH	0	625.2	128
2	0	0	0	0	5	NJ	0	539.4	155
3	0	2	1	0	7	OH	0	564.8	128
4	0	3	1	0	3	OK	0	512.0	107

In [57]:

```
df.drop(columns='State', inplace = True)
df.head()
```

Out[57]:

	VMail Message	CustServ Calls	Intl Plan	VMail Plan	Intl Calls	Churn	total_mins	State_enc
0	25	1	0	1	3	0	717.2	143
1	26	1	0	1	3	0	625.2	128
2	0	0	0	0	5	0	539.4	155
3	0	2	1	0	7	0	564.8	128
4	0	3	1	0	3	0	512.0	107

In [58]:

```
X = df.drop(columns=['Churn'])
y = df.Churn
```

In [59]:

```
X.head()
```

Out[59]:

	VMail Message	CustServ Calls	Intl Plan	VMail Plan	Intl Calls	total_mins	State_enc
0	25	1	0	1	3	717.2	143
1	26	1	0	1	3	625.2	128
2	0	0	0	0	5	539.4	155
3	0	2	1	0	7	564.8	128
4	0	3	1	0	3	512.0	107

Standerizing the data

In [60]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
X = pd.DataFrame(X)
X.head()
```

Out[60]:

	0	1	2	3	4	5	6
0	0.490196	0.111111	0.0	1.0	0.15	0.720659	0.761905
1	0.509804	0.111111	0.0	1.0	0.15	0.567505	0.619048
2	0.000000	0.000000	0.0	0.0	0.25	0.424671	0.876190
3	0.000000	0.222222	1.0	0.0	0.35	0.466955	0.619048
4	0.000000	0.333333	1.0	0.0	0.15	0.379058	0.419048

splitting the data into train and test

In [61]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=42)
```

logistic regression:

In [62]:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

In [63]:

```
from sklearn.metrics import confusion_matrix, classification_report
confusion_matrix(y_test,y_pred)
```

Out[63]:

```
array([[444, 141],
       [125, 430]], dtype=int64)
```

In [64]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.76      0.77       585
           1       0.75      0.77      0.76       555

    accuracy                           0.77      1140
   macro avg       0.77      0.77      0.77      1140
weighted avg       0.77      0.77      0.77      1140
```

KNN:

In [65]:

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(X_train, y_train)
y_pred_knn = model.predict(X_test)
confusion_matrix(y_test,y_pred_knn)
```

Out[65]:

```
array([[476, 109],
       [ 20, 535]], dtype=int64)
```

In [66]:

```
print(classification_report(y_test,y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.96      0.81      0.88       585
           1       0.83      0.96      0.89       555

    accuracy                           0.89      1140
   macro avg       0.90      0.89      0.89      1140
weighted avg       0.90      0.89      0.89      1140
```

Obs: knn is perfroming better than logistic regression in terms of accuracy

here knn is performing really good in terms of recall...ie it's predicting almost 96% true positives out of all actual postives...which is required here in this business case

In [67]:

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(X_train, y_train)
y_pred_knn = model.predict(X_test)
confusion_matrix(y_test,y_pred_knn)
```

Out[67]:

```
array([[476, 109],
       [ 20, 535]], dtype=int64)
```

let's check with different values of k

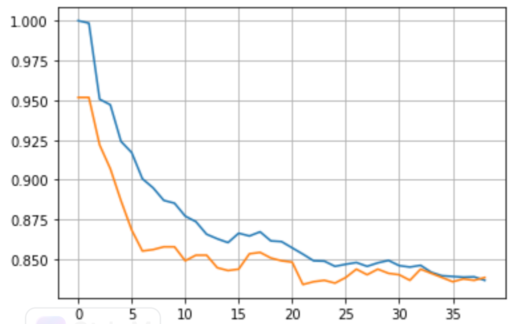
In [68]:

```
train_score = []
test_score = []
from sklearn.metrics import accuracy_score
for i in range(1,40):
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(X_train, y_train)
    y1_pred_knn = model.predict(X_train)
    y2_pred_knn = model.predict(X_test)
    train_score.append(accuracy_score(y_train,y1_pred_knn))
    test_score.append(accuracy_score(y_test,y2_pred_knn))
```

plotting with accuracy score

In [69]:

```
plt.plot(train_score)
plt.plot(test_score)
plt.grid()
```



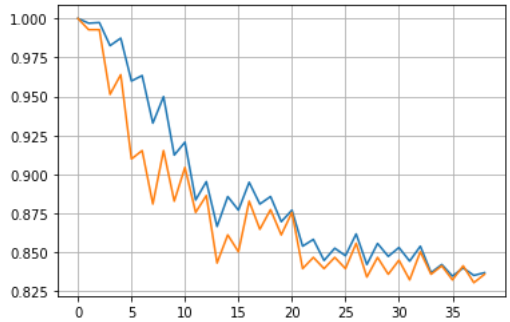
In [70]:

```
train_rec_score = []
test_rec_score = []
from sklearn.metrics import recall_score
for i in range(1,40):
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(X_train, y_train)
    y1_pred_knn = model.predict(X_train)
    y2_pred_knn = model.predict(X_test)
    train_rec_score.append(recall_score(y_train,y1_pred_knn))
    test_rec_score.append(recall_score(y_test,y2_pred_knn))
```

plotting with recall score

In [71]:

```
plt.plot(train_rec_score)
plt.plot(test_rec_score)
plt.grid()
```



but here we can observe some contradiction, how could the test data has this high accuracy with k = 1

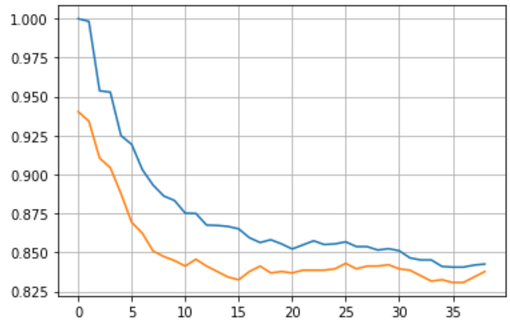
let's replot the same graph with splitting the data once again with different random state

In [72]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=50)

train_score = []
test_score = []
from sklearn.metrics import accuracy_score
for i in range(1,40):
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(X_train, y_train)
    y1_pred_knn = model.predict(X_train)
    y2_pred_knn = model.predict(X_test)
    train_score.append(accuracy_score(y_train,y1_pred_knn))
    test_score.append(accuracy_score(y_test,y2_pred_knn))

plt.plot(train_score)
plt.plot(test_score)
plt.grid()
```



kind of similar nature: could be explained based off the nature of the data

logistic regression with different random state

In [73]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=43)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred_new = model.predict(X_test)

from sklearn.metrics import classification_report
print(classification_report(y_pred_new,y_test))
```

```
              precision    recall  f1-score   support

           0       0.78      0.79      0.78       575
           1       0.78      0.77      0.78       565

    accuracy                           0.78      1140
   macro avg       0.78      0.78      0.78      1140
weighted avg       0.78      0.78      0.78      1140
```

now let's plot the ruc-auc curve for the logistic regression model

In [74]:

```
y_prob = model.predict_proba(X_test)
from sklearn.metrics import roc_auc_score
# tpr, fpr, thresholds = roc_auc_score(y_test,pd.DataFrame(y_prob[:,1]))
auc = roc_auc_score(y_test, y_prob[:,1])
auc
```

Out[74]:

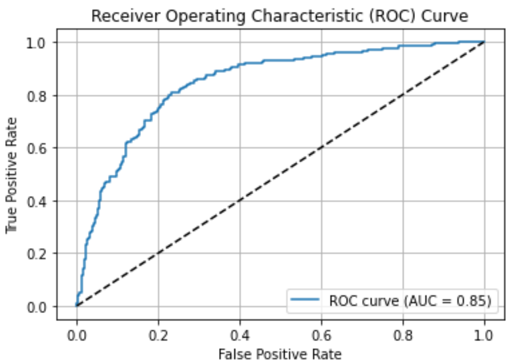
```
0.8458520058520058
```

In [75]:

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob[:,1])
```

In [76]:

```
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (AUC = {:.2f})'.format(auc))
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line representing random guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



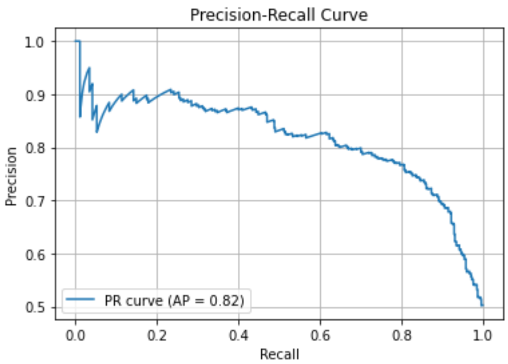
ROC score: 0.85 is quite good!

In [77]:

```
from sklearn.metrics import precision_recall_curve, average_precision_score

y_prob = model.predict_proba(X_test)
precision, recall, _ = precision_recall_curve(y_test, y_prob[:, 1])
average_precision = average_precision_score(y_test, y_prob[:, 1])

# Plot PR curve
plt.figure()
plt.plot(recall, precision, label='PR curve (AP = {:.2f})'.format(average_precision))
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.grid()
plt.show()
```



observation: descent performance

Dicision Trees:

first see the default observation

In [78]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=41)
from sklearn.tree import DecisionTreeClassifier
dic_tree = DecisionTreeClassifier()
dic_tree.fit(X_train, y_train)
y_pred_dic = dic_tree.predict(X_test)
print(classification_report(y_test, y_pred_dic))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97       591
           1       0.94      1.00      0.97       549

    accuracy                           0.97      1140
   macro avg       0.97      0.97      0.97      1140
weighted avg       0.97      0.97      0.97      1140
```

In [79]:

```
!pip install graphviz
```

```
Requirement already satisfied: graphviz in c:\users\ashutosh patidar\anaconda3\lib\site-packages (0.20.1)
```

In [80]:

```
from sklearn.tree import export_graphviz
import graphviz
```

In [81]:

```
!pip install pydotplus
```

```
Requirement already satisfied: pydotplus in c:\users\ashutosh patidar\anaconda3\lib\site-packages (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in c:\users\ashutosh patidar\anaconda3\lib\site-packages (from pydotplus)
(3.0.4)
```

now lets apply hypterparameter tuning

In [82]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 3]
}

dic_tree = DecisionTreeClassifier()
grid_search = GridSearchCV(dic_tree, param_grid, cv=5)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=43)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
best_params
# best_model
```

Out[82]:

```
{'criterion': 'entropy',
 'max_depth': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2}
```

In [83]:

```
y_best = best_model.predict(X_test)
print(classification_report(y_test,y_best))
```

```
              precision    recall  f1-score   support

           0       1.00      0.91      0.95       585
           1       0.91      1.00      0.95       555

    accuracy                           0.95      1140
   macro avg       0.96      0.95      0.95      1140
weighted avg       0.96      0.95      0.95      1140
```

observed excellent accuracy!

In [84]:

```
X_train.head()
```

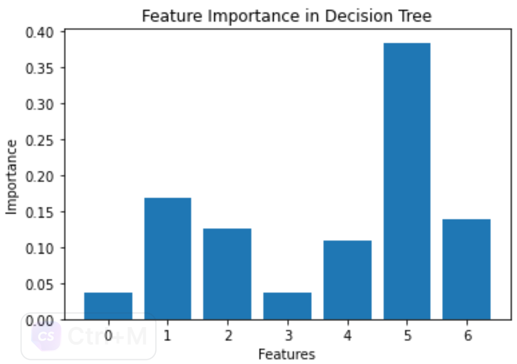
Out[84]:

	0	1	2	3	4	5	6
2751	0.000000	0.000000	0.0	0.0	0.20	0.606626	0.838095
811	0.000000	0.000000	0.0	0.0	0.30	0.409023	0.400000
1072	0.490196	0.111111	0.0	1.0	0.30	0.250874	0.371429
2519	0.000000	0.222222	0.0	0.0	0.25	0.393874	0.371429
5032	0.000000	0.000000	1.0	0.0	0.30	0.346929	0.533333

In [85]:

```
importance = best_model.feature_importances_

plt.bar(range(len(importance)), importance)
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance in Decision Tree')
plt.show()
```



In [86]:

```
X.head()
```

Out[86]:

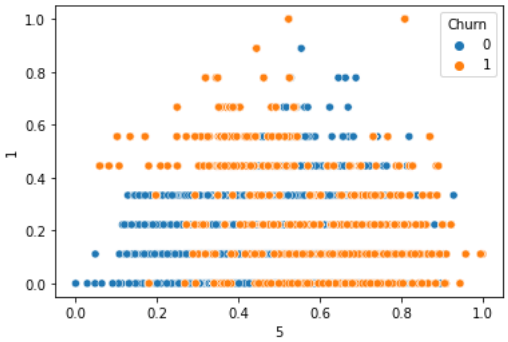
	0	1	2	3	4	5	6
0	0.490196	0.111111	0.0	1.0	0.15	0.720659	0.761905
1	0.509804	0.111111	0.0	1.0	0.15	0.567505	0.619048
2	0.000000	0.000000	0.0	0.0	0.25	0.424671	0.876190
3	0.000000	0.222222	1.0	0.0	0.35	0.466955	0.619048
4	0.000000	0.333333	1.0	0.0	0.15	0.379058	0.419048

In [87]:

```
sns.scatterplot(X[5],X[1], hue = y)
```

Out[87]:

<AxesSubplot:xlabel='5', ylabel='1'>



In [88]:

```
df.columns
```

Out[88]:

```
Index(['VMail Message', 'CustServ Calls', 'Intl Plan', 'VMail Plan',
       'Intl Calls', 'Churn', 'total_mins', 'State_enc'],
      dtype='object')
```

Random Forest:

default observation

In [94]:

```
from sklearn.ensemble import RandomForestClassifier
import numpy as np
clf = RandomForestClassifier()
clf.fit(X_train, y_train)

predictions = clf.predict(X_test)
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      0.95      0.97       585
           1       0.95      1.00      0.97       555

    accuracy                           0.97      1140
   macro avg       0.97      0.97      0.97      1140
weighted avg       0.97      0.97      0.97      1140
```

performing quite well

now lest see by using hyperparameter tuning

In [91]:

```
clf = RandomForestClassifier()

param_grid = {
    'n_estimators': [100, 200, 300],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 3]
}

grid_search = GridSearchCV(clf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
best_params
```

In [98]:

```
predicts = best_model.predict(X_test)
print(classification_report(y_test,predicts))
```

Now usnig some boosting algorithms!

with default hyperparameters

In [100]:

```
from sklearn.ensemble import AdaBoostClassifier
adaboost = AdaBoostClassifier()
adaboost.fit(X_train, y_train)
y_pred_ada = adaboost.predict(X_test)
print(classification_report(y_test,y_pred_ada))
```

```
              precision    recall  f1-score   support

           0       0.84      0.86      0.85       585
           1       0.85      0.83      0.84       555

    accuracy                           0.84      1140
   macro avg       0.84      0.84      0.84      1140
weighted avg       0.84      0.84      0.84      1140
```

Obs: not performing quite well lets use hyperparameter tuning

In [101]:

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.1, 0.5, 1.0]
}

adaboost = AdaBoostClassifier()

grid_search = GridSearchCV(adaboost, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_adaboost = grid_search.best_estimator_
grid_search.best_params_
```

Out[101]:

```
{'learning_rate': 0.5, 'n_estimators': 50}
```

In [102]:

```
y_pred_ada = best_adaboost.predict(X_test)
print(classification_report(y_test, y_pred_ada))
```

```
              precision    recall  f1-score   support

           0       0.85      0.85      0.85       585
           1       0.84      0.84      0.84       555

    accuracy                           0.85      1140
   macro avg       0.85      0.85      0.85      1140
weighted avg       0.85      0.85      0.85      1140
```

obs: imporoved a little but not much improvement!

let's use some more techniques

In [103]:

```
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
gbc.fit(X_train, y_train)
y_test_gbc = gbc.predict(X_test)
print(classification_report(y_test,y_test_gbc))
```

```
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       585
           1       0.88      0.87      0.87       555

    accuracy                           0.88      1140
   macro avg       0.88      0.88      0.88      1140
weighted avg       0.88      0.88      0.88      1140
```

hypterparameter tuning

In [105]:

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.1, 0.5, 1.0],
    'max_depth': [3, 5, 7]
}

gb_clf = GradientBoostingClassifier()

grid_search = GridSearchCV(gb_clf, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_gb_clf = grid_search.best_estimator_
grid_search.best_params_
```

Out[105]:

```
{'learning_rate': 1.0, 'max_depth': 7, 'n_estimators': 200}
```

In [106]:

```
y_pred_gb = best_gb_clf.predict(X_test)
print(classification_report(y_test, y_pred_gb))
```

```
              precision    recall  f1-score   support

           0       1.00      0.95      0.97       585
           1       0.95      1.00      0.97       555

    accuracy                           0.97      1140
   macro avg       0.97      0.97      0.97      1140
weighted avg       0.97      0.97      0.97      1140
```

Obs: so far the gradient boosting clf worked the best overall!

In [108]:

```
from sklearn.ensemble import BaggingClassifier
bc = BaggingClassifier()
bc.fit(X_train, y_train)
bc_pred = bc.predict(X_test)
print(classification_report(y_test,bc_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.93      0.96       585
           1       0.93      1.00      0.96       555

    accuracy                           0.96      1140
   macro avg       0.97      0.96      0.96      1140
weighted avg       0.97      0.96      0.96      1140
```

In [109]:

```
bagging = BaggingClassifier()
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_samples': [0.5, 0.7, 1.0],
    'max_features': [0.5, 0.7, 1.0]
}

grid_search = GridSearchCV(bagging, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_bagging = grid_search.best_estimator_
grid_search.best_params_
```

Out[109]:

```
{'max_features': 0.7, 'max_samples': 1.0, 'n_estimators': 100}
```

In [110]:

```
y_pred_bagging = best_bagging.predict(X_test)
print(classification_report(y_test, y_pred_bagging))
```

```
              precision    recall  f1-score   support

           0       1.00      0.95      0.97       585
           1       0.95      1.00      0.97       555

    accuracy                           0.97      1140
   macro avg       0.98      0.98      0.97      1140
weighted avg       0.98      0.97      0.97      1140
```

it too seems working pretty well

In [112]:

```
!pip install xgboost
```

```
^C
```

In [114]:

```
import xgboost as xgb
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
params = {
    'objective': 'binary:logistic',
    'eval_metric': 'logloss',
    'eta': 0.1,
    'max_depth': 3
}
num_rounds = 100
xgb_model = xgb.train(params, dtrain, num_rounds)
y_pred = xgb_model.predict(dtest)
y_pred_binary = [1 if p >= 0.5 else 0 for p in y_pred]
print(classification_report(y_test, y_pred_binary))
```

```
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       585
           1       0.87      0.88      0.87       555

    accuracy                           0.88      1140
   macro avg       0.88      0.88      0.88      1140
weighted avg       0.88      0.88      0.88      1140
```

using stacking

In [115]:

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier

base_models = [
    ('logistic_regression', LogisticRegression()),
    ('k_nearest_neighbors', KNeighborsClassifier()),
    ('support_vector_machine', SVC()),
    ('naive_bayes', GaussianNB()),
    ('decision_tree', DecisionTreeClassifier())
]

stacking = StackingClassifier(
    estimators=base_models,
    final_estimator=LogisticRegression()
)
stacking.fit(X_train, y_train)
```

Out[115]:

```
StackingClassifier(estimators=[('logistic_regression', LogisticRegression()),
                               ('k_nearest_neighbors', KNeighborsClassifier()),
                               ('support_vector_machine', SVC()),
                               ('naive_bayes', GaussianNB()),
                               ('decision_tree', DecisionTreeClassifier())],
                   final_estimator=LogisticRegression())
```

In [116]:

```
y_pred_stacking = stacking.predict(X_test)
print(classification_report(y_test, y_pred_stacking))
```

```
              precision    recall  f1-score   support

           0       1.00      0.95      0.97       585
           1       0.95      1.00      0.97       555

    accuracy                           0.97      1140
   macro avg       0.97      0.97      0.97      1140
weighted avg       0.97      0.97      0.97      1140
```

tuning

In [120]:

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier

base_models = [
    ('logistic_regression', LogisticRegression()),
    ('k_nearest_neighbors', KNeighborsClassifier()),
    ('support_vector_machine', SVC()),
    ('naive_bayes', GaussianNB()),
    ('decision_tree', DecisionTreeClassifier())
]

stacking = StackingClassifier(
    estimators=base_models,
    final_estimator=LogisticRegression()
)
param_grid = {
    'logistic_regression__C': [0.1, 1, 10],
    'k_nearest_neighbors__n_neighbors': [3, 5, 7],
    'support_vector_machine__C': [0.1, 1, 10],
    'decision_tree__max_depth': [None, 5, 10]
}

grid_search = GridSearchCV(stacking, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_stacking = grid_search.best_estimator_
best_stacking.best_params_
```

In [ ]:

```
y_pred_stacking = best_stacking.predict(X_test)
print(classification_report(y_test, y_pred_stacking))
```

predictions are not there becuase it's taking a lot of time!

conclusion: some algorithms worked quite well like: decision trees(fasted with best accuracy and recall), random forest(time consuming), gradientBoosting classifier.

If anywhere in the notebook, if you find any point of disscussion pls comment down below!

thanks and have a good day!

In [ ]:

In [ ]:

In [ ]:

In [ ]: