Telecom Indusrty Data

Introduction: This is the dataset of telecome industry which tell us that people preffered to call more in the night as compare to evening and day. Also telecom industry dataset tell us that there is a positive relation betwen charges and the customers who preffered to churn into another company.

We are going to analyse the telecom data which contains 21 columns Namely 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 Minutes, Total Eve Calls, Total Eve Charge, Total Night Minutes, Total Night Calls, Total Night Charge, Total Intl Minutes, Total Intl Calls, Total Intl Charge, Customer Service Calls, Churn and 3333 rows

Objective: Build a model that will help to identiy that customers will churn or not into another company by considering the various factors.

Chrun Call Estimation

- 1. Import Libraries
- 2. Import Data
- 3. Data Eye/Sanity Check
- 4. Data Understanding-Info and describe
- 5. Data Cleaning and Pre Processing - (i.e. NA value.Missing value,Junk values/Wrong)
- 6. Looking and undersatbd features, Data Distribution(Plotting, Table (GroupBy))
- 7. Correlation of features and aslo target variale:
- 8. KNN Analysis

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
d=pd.read_csv(r'C:\Users\lxm\Downloads\data\Telecom.csv')
```

In [3]:
```
d
```

Out[3]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	t c
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	
...	...	...	...	...	...	...	...	...	...	...	...	
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...	
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...	
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...	
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...	
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...	

3333 rows × 21 columns

In [4]:
```
d.head()
```

Out[4]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	t c
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	

5 rows × 21 columns

In [5]:
```
d.tail()
```

Out[5]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	t c
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...	
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...	
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...	
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...	
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...	

5 rows × 21 columns

Shape of the dataset

In [6]:
```
d.shape
```

Out[6]:
```
(3333, 21)
```

In [7]:
```
d.columns
```

Out[7]:
```
Index(['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 minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls', 'churn'],
      dtype='object')
```

Information about dataset

In [8]:
```
d.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
```

Statistical Information about Dataset

In [9]:
```
d.describe().T
```

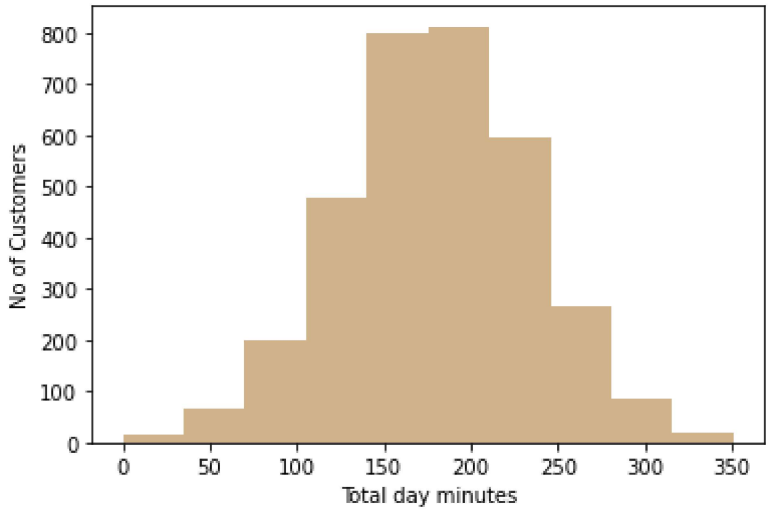
Out[9]:

	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

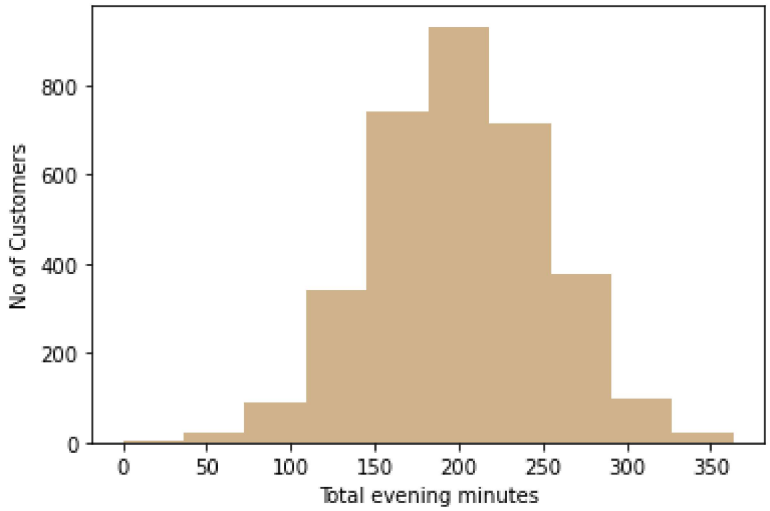
In [10]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

Calculate Histogram Time spend time in day call

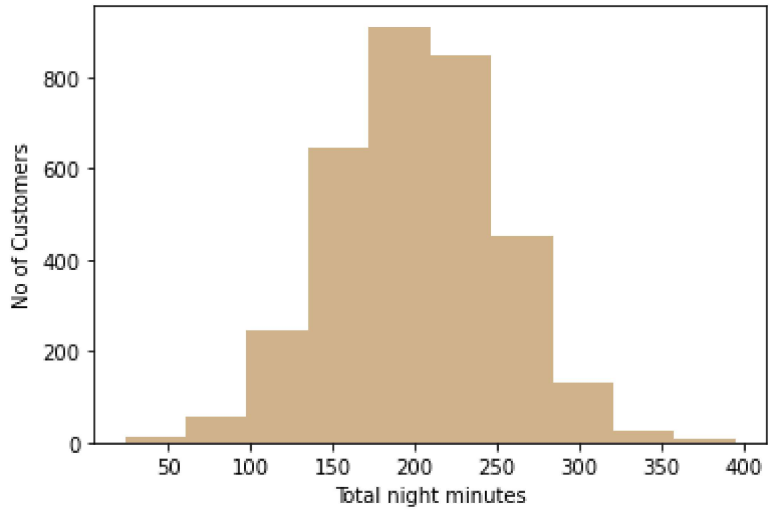
In [11]:
```
plt.hist(d['total day minutes'],bins=10,facecolor='tan')
plt.xlabel('Total day minutes')
plt.ylabel('No of Customers')
plt.show()
```



In [12]:
```
plt.hist(d['total eve minutes'],bins=10,facecolor='tan')
plt.xlabel('Total evening minutes')
plt.ylabel('No of Customers')
plt.show()
```



In [13]:
```
plt.hist(d['total night minutes'],bins=10,facecolor='tan')
plt.xlabel('Total night minutes')
plt.ylabel('No of Customers')
plt.show()
```



Call wise charges

In [14]:
```
d1=d.groupby('total day charge').max()
d1
```

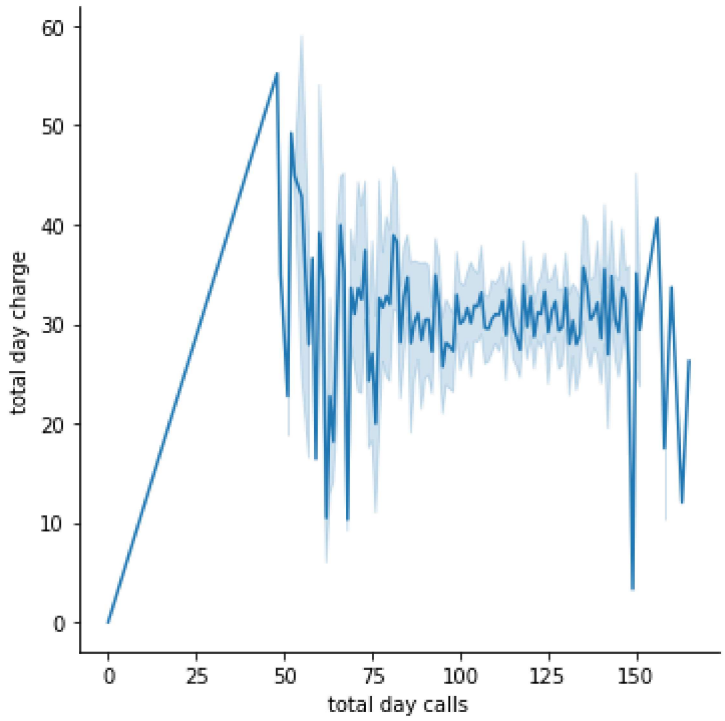
Out[14]:

total day charge	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total eve minutes	t c
0.00	VT	101	510	413-7655	no	no	0	0.0	0	192.1	
0.44	OK	127	510	403-1128	no	yes	27	2.6	113	254.0	
1.33	OH	134	415	406-4158	no	no	0	7.8	86	171.4	
1.34	WI	70	415	405-9233	no	no	0	7.9	100	136.4	
2.13	OR	98	415	378-6772	yes	no	0	12.5	67	256.6	
...	...	...	...	...	...	...	...	...	...	...	
57.04	MO	112	415	373-2053	no	no	0	335.5	77	212.5	
57.36	OH	83	415	370-9116	no	no	0	337.4	120	227.4	
58.70	OH	115	510	348-1163	yes	no	0	345.3	81	203.4	
58.96	NY	64	415	345-9140	yes	no	0	346.8	55	249.5	
59.64	CO	154	415	343-5709	no	no	0	350.8	75	216.5	

1667 rows × 20 columns

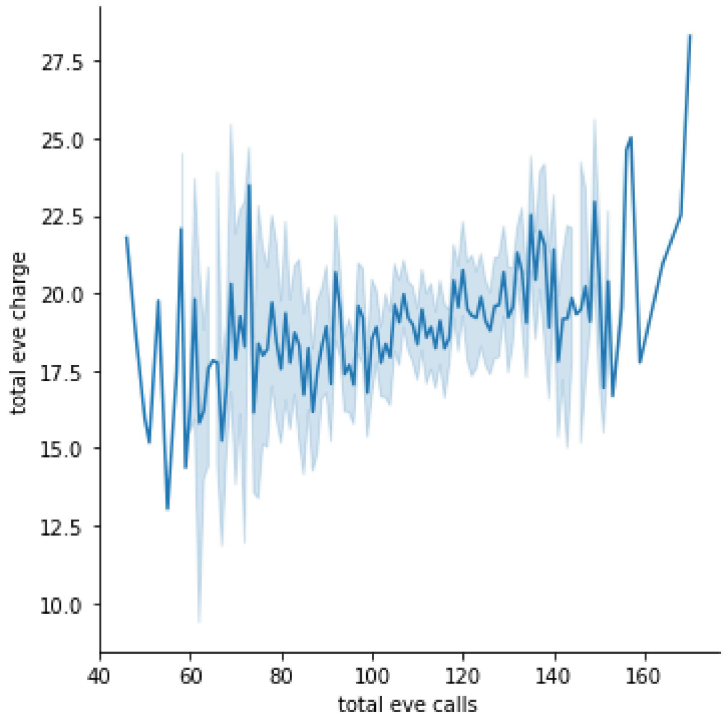
In [15]:
```
sns.relplot(x='total day calls',y='total day charge', data=d1, kind='line')
plt.plot()
```

Out[15]: []



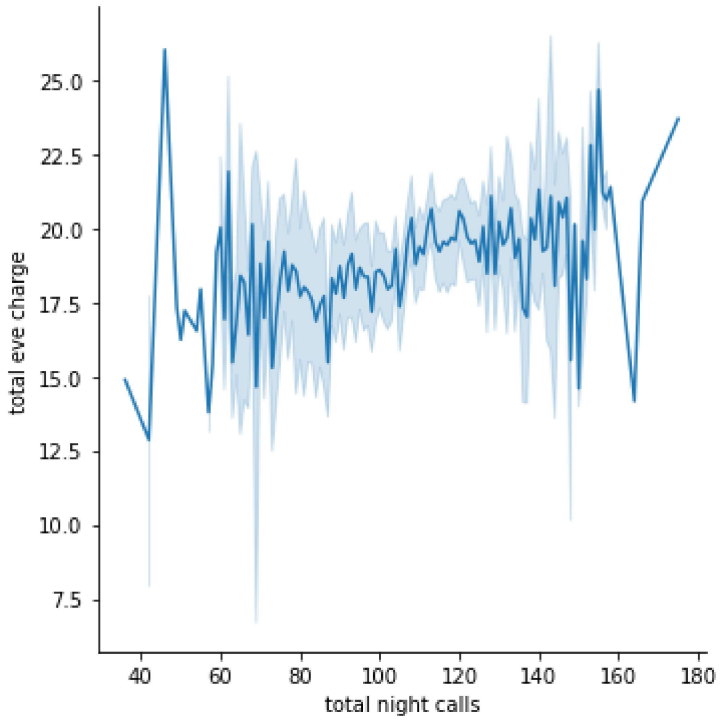
In [16]:
```
sns.relplot(x='total eve calls',y='total eve charge',data=d1,kind='line')
plt.plot()
```

Out[16]: []



In [17]:
```
sns.relplot(x='total night calls',y='total eve charge',data=d1, kind='line')
plt.plot()
```

Out[17]: []



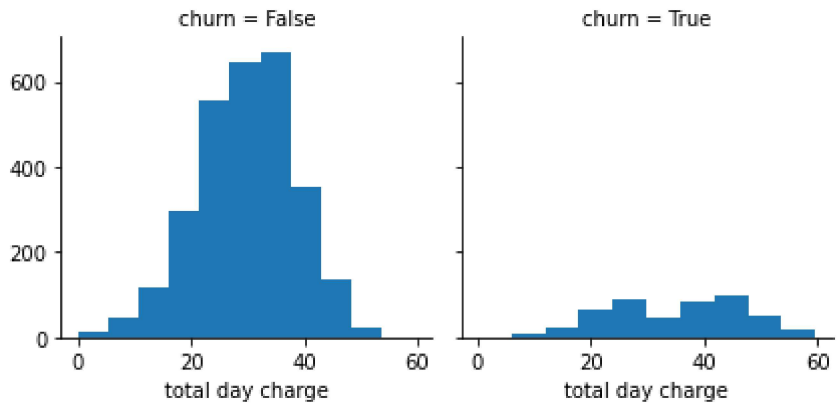
In [18]:
```
d['churn'].value_counts()
```

Out[18]:
```
False    2850
True      483
Name: churn, dtype: int64
```

How do we categorized the churn and not churn for the time charges on day call?

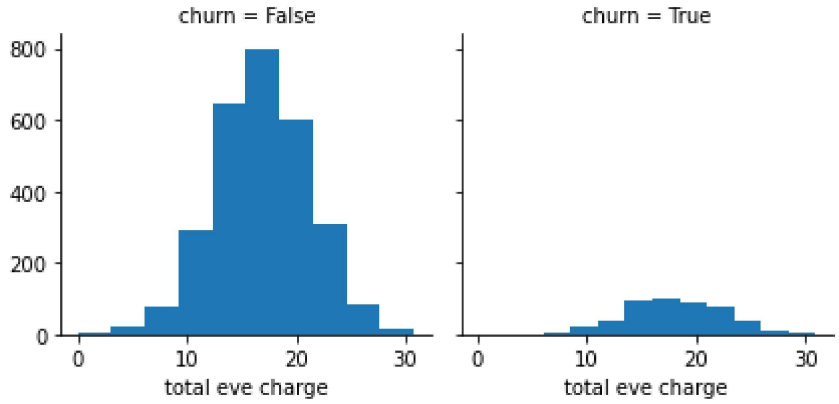
In [19]:
```
g=sns.FacetGrid(d,col='churn')
g.map(plt.hist,"total day charge")
```

Out[19]: <seaborn.axisgrid.FacetGrid at 0x22ba7c98310>



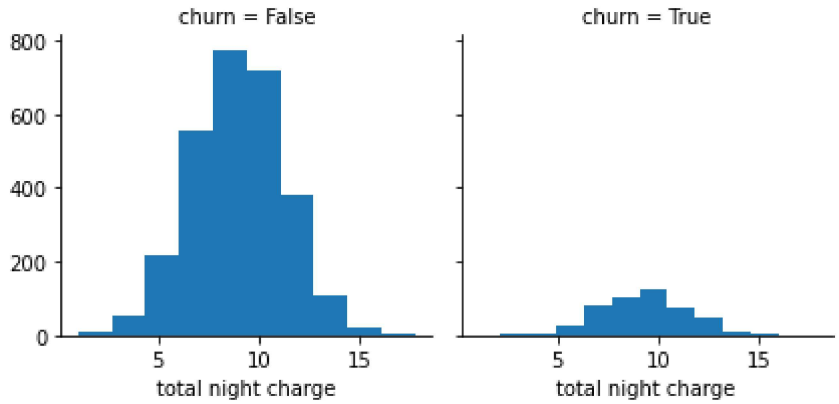
In [20]:
```
g=sns.FacetGrid(d,col='churn')
g.map(plt.hist,"total eve charge")
```

Out[20]: <seaborn.axisgrid.FacetGrid at 0x22ba25807f0>



In [21]:
```
g=sns.FacetGrid(d,col='churn')
g.map(plt.hist,"total night charge")
```

Out[21]:
```
<seaborn.axisgrid.FacetGrid at 0x22ba7f4e580>
```



Find the customers who did opt a voice mail plane

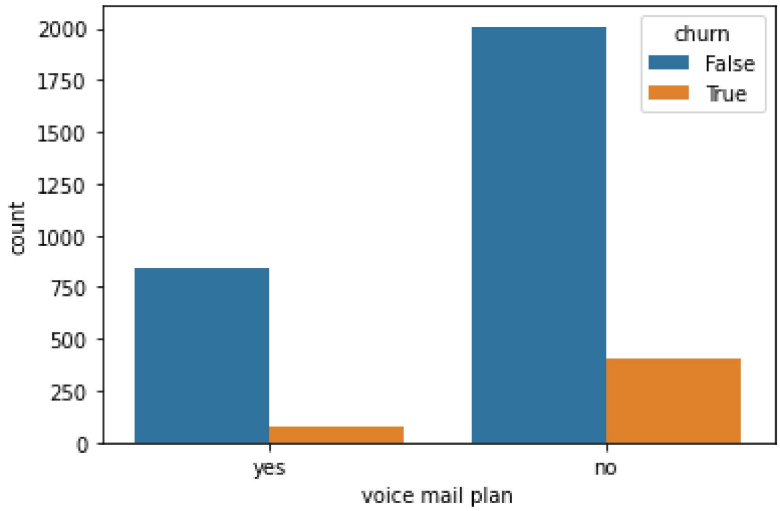
In [22]:
```
d['voice mail plan'].value_counts()
```

Out[22]:
```
no     2411
yes     922
Name: voice mail plan, dtype: int64
```

Produce a countploat of above result

In [23]:
```
sns.countplot(x='voice mail plan',hue='churn',data=d)
```

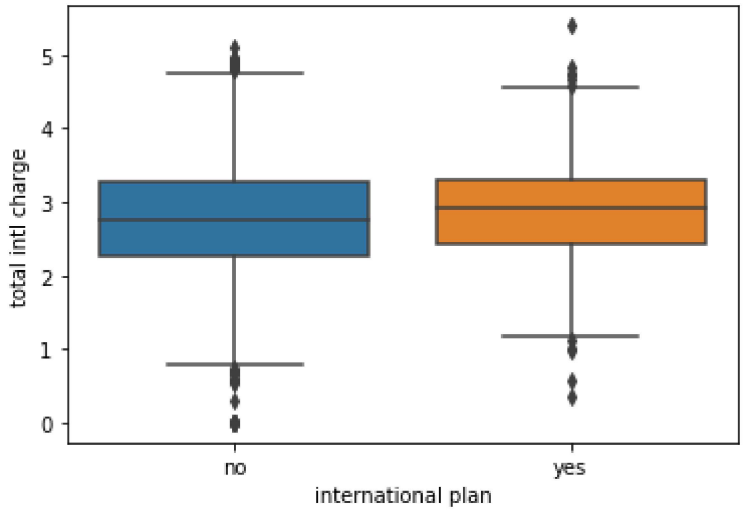
Out[23]:
```
<AxesSubplot:xlabel='voice mail plan', ylabel='count'>
```



Create a boxplot for a international plan and International charges

In [24]:
```
sns.boxplot(x='international plan',y='total intl charge',data=d)
```

Out[24]:
```
<AxesSubplot:xlabel='international plan', ylabel='total intl charge'>
```



Create a cross tab of area code to find the churner or not curner

In [25]:
```
pd.crosstab(d['area code'], d['voice mail plan'])
```

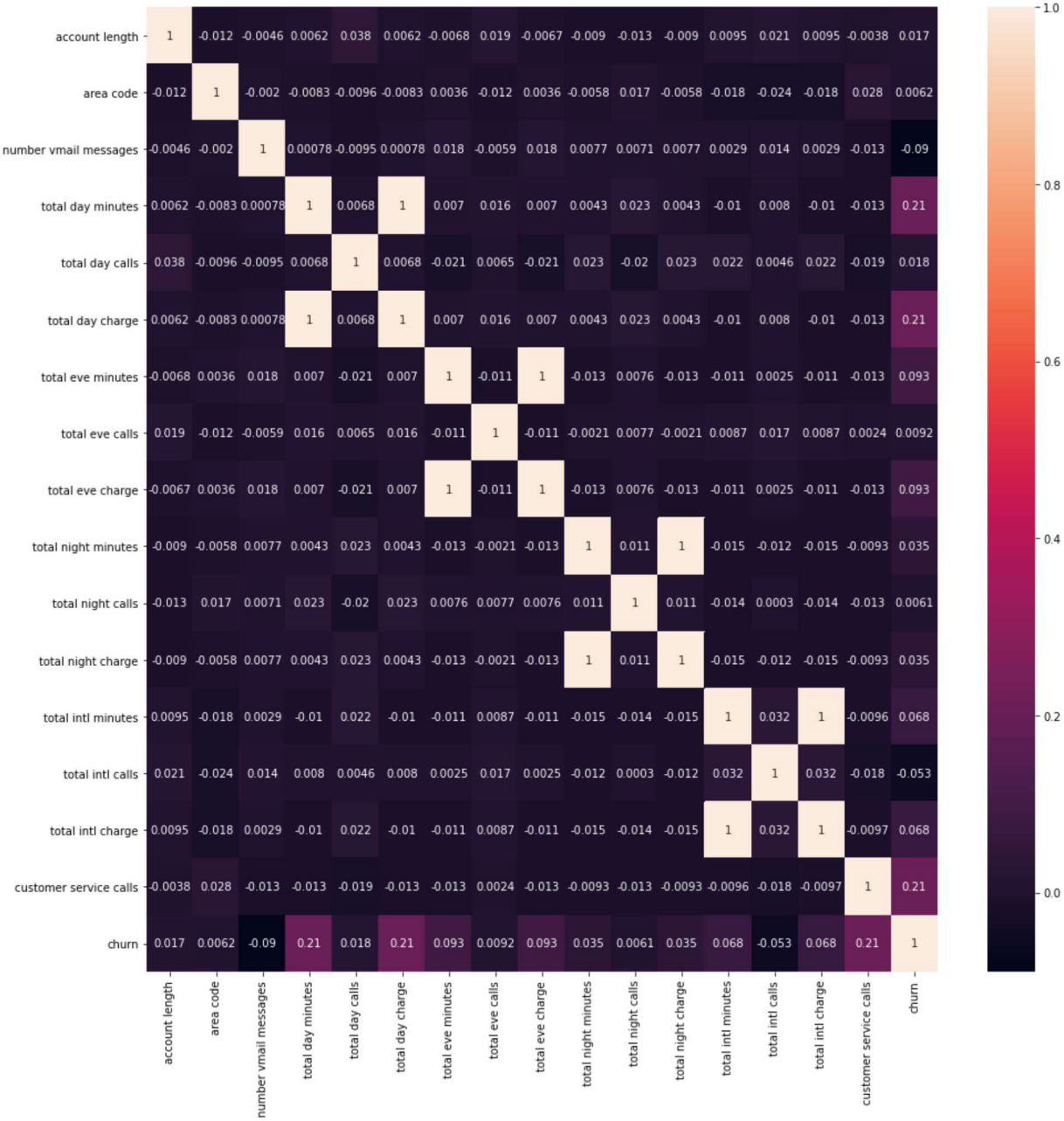
Out[25]:

voice mail plan	no	yes
area code		
408	618	220
415	1184	471
510	609	231

Correlation

In [26]:
```
plt.figure(figsize=(16,16))
corr=d.corr()
sns.heatmap(corr,annot=True)
```

Out[26]: <AxesSubplot:>



Conclusion

People prefer to call in the night as compare to evening and morning.

The highest time spend in day call is350.80, evening 363.70, and night call is 395 minutes.

people prefer to call in day is maximum 165, n evening 170 and in night 175.

There is a a positive relation between calls minutes and charges on call.

There are 483 people who churn their plan and convert into the another company and 2850 has not churn.

There are 922 people who have te voice mail plane and 2411 peoples do not have plane.

There is apositive relation between churn and the total eve minutes, call and charges

KNN

Remove Extraneous columns

In [27]:
```
d.drop(['state','area code','phone number'],axis=1,inplace=True)
```

In [28]:
```
d.columns
```

Out[28]:
```
Index(['account length', 'international plan', 'voice mail plan',
       'number vmail messages', 'total day minutes', 'total day calls',
       'total day charge', 'total eve minutes', 'total eve calls',
       'total eve charge', 'total night minutes', 'total night calls',
       'total night charge', 'total intl minutes', 'total intl calls',
       'total intl charge', 'customer service calls', 'churn'],
      dtype='object')
```

Some of the data is categorical data and some are floats. These features need to be numerical encoded using one of the method

In [29]:
```
d[['international plan', 'voice mail plan','churn']]
```

Out[29]:

	international plan	voice mail plan	churn
0	no	yes	False
1	no	yes	False
2	no	no	False
3	yes	no	False
4	yes	no	False
...	...	...	...
3328	no	yes	False
3329	no	no	False
3330	no	no	False
3331	yes	no	False
3332	no	yes	False

3333 rows × 3 columns

Import LabelBinarizer

In [30]:
```
from sklearn.preprocessing import LabelBinarizer

lb=LabelBinarizer()

for col in ['international plan','voice mail plan','churn']:
    d[col]=lb.fit_transform(d[col])
```

In [31]:
```
d[['international plan', 'voice mail plan','churn']]
```

Out[31]:

	international plan	voice mail plan	churn
0	0	1	0
1	0	1	0
2	0	0	0
3	1	0	0
4	1	0	0
...	...	...	...
3328	0	1	0
3329	0	0	0
3330	0	0	0
3331	1	0	0
3332	0	1	0

3333 rows × 3 columns

Data Preprocessing

In [32]:
```
import warnings
warnings.filterwarnings('ignore',module='sklearn')

from sklearn.preprocessing import MinMaxScaler

msc=MinMaxScaler()

data=pd.DataFrame(msc.fit_transform(d),columns=d.columns)
```

In [33]:
```
data
```

Out[33]:

	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls
0	0.524793	0.0	1.0	0.490196	0.755701	0.666667	0.755701	0.542755	0.582353
1	0.438017	0.0	1.0	0.509804	0.460661	0.745455	0.460597	0.537531	0.605882
2	0.561983	0.0	0.0	0.000000	0.693843	0.690909	0.693830	0.333242	0.647059
3	0.342975	1.0	0.0	0.000000	0.853478	0.430303	0.853454	0.170195	0.517647
4	0.305785	1.0	0.0	0.000000	0.475200	0.684848	0.475184	0.407754	0.717647
...	...	...	...	...	...	...	...	...	...
3328	0.789256	0.0	1.0	0.705882	0.445268	0.466667	0.445171	0.592521	0.741176
3329	0.276860	0.0	0.0	0.000000	0.658780	0.345455	0.658786	0.421776	0.323529
3330	0.111570	0.0	0.0	0.000000	0.515393	0.660606	0.515426	0.794061	0.341176
3331	0.756198	1.0	0.0	0.000000	0.609464	0.636364	0.609490	0.438823	0.494118
3332	0.301653	0.0	1.0	0.490196	0.668187	0.684848	0.668176	0.731097	0.482353

3333 rows × 18 columns

Fit a K Nearest Neighbor model with a values on the dataset and predict the ouctome on the same data

Get a list of columns that don't contain the label

In [34]:
```
x_cols=[x for x in d.columns if x!='churn']
```

Slit the data into two dataframes

In [35]:
```
x_data=data[x_cols]
y_data=data['churn']
```

In [36]:
```
x_data
```

Out[36]:

	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls
0	0.524793	0.0	1.0	0.490196	0.755701	0.666667	0.755701	0.542755	0.582353
1	0.438017	0.0	1.0	0.509804	0.460661	0.745455	0.460597	0.537531	0.605882
2	0.561983	0.0	0.0	0.000000	0.693843	0.690909	0.693830	0.333242	0.647059
3	0.342975	1.0	0.0	0.000000	0.853478	0.430303	0.853454	0.170195	0.517647
4	0.305785	1.0	0.0	0.000000	0.475200	0.684848	0.475184	0.407754	0.717647
...	...	...	...	...	...	...	...	...	...
3328	0.789256	0.0	1.0	0.705882	0.445268	0.466667	0.445171	0.592521	0.741176
3329	0.276860	0.0	0.0	0.000000	0.658780	0.345455	0.658786	0.421776	0.323529
3330	0.111570	0.0	0.0	0.000000	0.515393	0.660606	0.515426	0.794061	0.341176
3331	0.756198	1.0	0.0	0.000000	0.609464	0.636364	0.609490	0.438823	0.494118
3332	0.301653	0.0	1.0	0.490196	0.668187	0.684848	0.668176	0.731097	0.482353

3333 rows × 17 columns

In [37]:
```
y_data
```

Out[37]:
```
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0
       ...
3328    0.0
3329    0.0
3330    0.0
3331    0.0
3332    0.0
Name: churn, Length: 3333, dtype: float64
```

Import Train Test Split

In [38]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_data,y_data,test_size=0.3)
```

In [39]:
```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[39]:
```
((2333, 17), (1000, 17), (2333,), (1000,))
```

Import KNeighborsClassifier

In [40]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [41]:
```
res={}
for k in range(3,20):
    Knn= KNeighborsClassifier(n_neighbors=k)
    Knn.fit(x_train,y_train)
    print("Training Accuracy for k={} is {}:".format(k,Knn.score(x_train,y_train)
    print("Testing Accuracy for K={} is {}:".format(k,Knn.score(x_test,y_test)))
    res[k]=Knn.score(x_test,y_test)
```

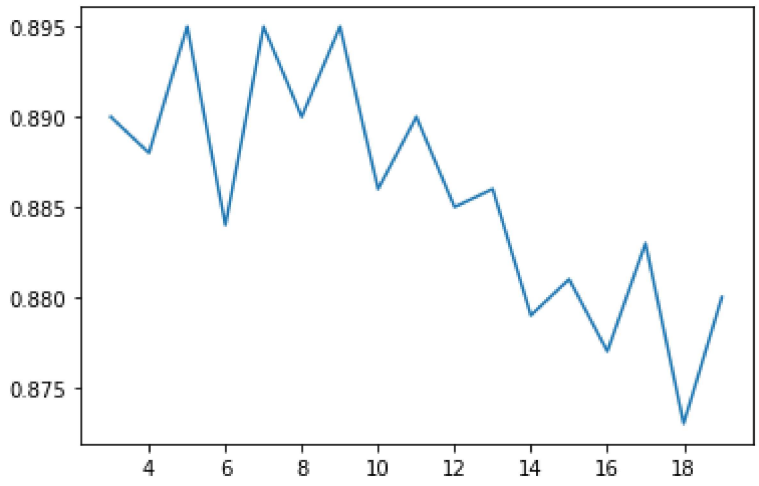
```
Training Accuracy for k=3 is 0.9412773253321903:
Testing Accuracy for K=3 is 0.89:
Training Accuracy for k=4 is 0.914273467638234:
Testing Accuracy for K=4 is 0.888:
Training Accuracy for k=5 is 0.9245606515216459:
Testing Accuracy for K=5 is 0.895:
Training Accuracy for k=6 is 0.906986712387484:
Testing Accuracy for K=6 is 0.884:
Training Accuracy for k=7 is 0.9211315902271753:
Testing Accuracy for K=7 is 0.895:
Training Accuracy for k=8 is 0.9009858551221603:
Testing Accuracy for K=8 is 0.89:
Training Accuracy for k=9 is 0.9112730390055722:
Testing Accuracy for K=9 is 0.895:
Training Accuracy for k=10 is 0.901843120445778:
Testing Accuracy for K=10 is 0.886:
Training Accuracy for k=11 is 0.9082726103729104:
Testing Accuracy for K=11 is 0.89:
Training Accuracy for k=12 is 0.8958422631804543:
Testing Accuracy for K=12 is 0.885:
Training Accuracy for k=13 is 0.9027003857693956:
Testing Accuracy for K=13 is 0.886:
Training Accuracy for k=14 is 0.8924132018859837:
Testing Accuracy for K=14 is 0.879:
Training Accuracy for k=15 is 0.9009858551221603:
Testing Accuracy for K=15 is 0.881:
Training Accuracy for k=16 is 0.8898414059151307:
Testing Accuracy for K=16 is 0.877:
Training Accuracy for k=17 is 0.8971281611658808:
Testing Accuracy for K=17 is 0.883:
Training Accuracy for k=18 is 0.8902700385769395:
Testing Accuracy for K=18 is 0.873:
Training Accuracy for k=19 is 0.8945563651950279:
Testing Accuracy for K=19 is 0.88:
```

In [42]:
```
res
```

Out[42]:
```
{3: 0.89,
 4: 0.888,
 5: 0.895,
 6: 0.884,
 7: 0.895,
 8: 0.89,
 9: 0.895,
 10: 0.886,
 11: 0.89,
 12: 0.885,
 13: 0.886,
 14: 0.879,
 15: 0.881,
 16: 0.877,
 17: 0.883,
 18: 0.873,
 19: 0.88}
```

In [43]:
```
plt.plot(res.keys(),res.values())
```

Out[43]:
```
[<matplotlib.lines.Line2D at 0x22ba8ebb8b0>]
```



In [44]:
```
Knn= KNeighborsClassifier(n_neighbors=19)
Knn.fit(x_train,y_train)
```

Out[44]:
```
KNeighborsClassifier(n_neighbors=19)
```

In [45]:
```
y_pred=Knn.predict(x_test)
```

In [46]:
```
y_pred.shape
```

Out[46]:
```
(1000,)
```

In [47]:
```
from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(y_pred,y_test))
```

```
[[855 119]
 [  1  25]]
```

In [48]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.88      0.93       974
         1.0       0.17      0.96      0.29        26

    accuracy                           0.88      1000
   macro avg       0.59      0.92      0.61      1000
weighted avg       0.98      0.88      0.92      1000
```

Conclusion ¶

- By using the KNN method we come to the point that k=19 are good fit into the model and try to give the high accuracy as compare to others which is 0.89.
- The confusion matrix tell us that 858 peoples are churn and the dataset actually said that it is churn and 37 do not churn tell by dataset and the machine as well, also model predict that 100 peoples churn but actually it is not churn and 5 people not churn which is actually churn.

In [ ]:
```
```