Project - Python_ML - Case Study - Telecom Churn Analysis

Part 1 - Data Exploration and Pre-processsing

Importing the required libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

1) load the given dataset

In [2]:

```
df = pd.read_csv('Project_Py_ml_Case_Study.csv')
```

In [3]:

```
pd.set_option('display.max_columns', None)
```

In [4]:

```
df.head()
```

Out[4]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Tot ir cal
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	

2) print all the column names

In [5]:

```
df.columns
```

Out[5]:

```
Index(['State', 'Account length', 'Area code', 'International plan',
       'Voice mail plan', 'Number vmail messages', 'Total day minutes',
       'Total day calls', 'Total day charge', 'Total eve minutes',
       'Total eve calls', 'Total eve charge', 'Total night minutes',
       'Total night calls', 'Total night charge', 'Total intl minutes',
       'Total intl calls', 'Total intl charge', 'Customer service calls',
       'Churn'],
      dtype='object')
```

3) describe the data

In [6]:

```
df.describe()
```

Out[6]:

	Account length	Area code	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Tot
count	2666.000000	2666.000000	2666.000000	2666.00000	2666.000000	2666.000000	2666.000000	2666.000000	2666.000000	2666.000000	2666
mean	100.620405	437.438860	8.021755	179.48162	100.310203	30.512404	200.386159	100.023631	17.033072	201.168942	100
std	39.563974	42.521018	13.612277	54.21035	19.988162	9.215733	50.951515	20.161445	4.330864	50.780323	19
min	1.000000	408.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	43.700000	33
25%	73.000000	408.000000	0.000000	143.40000	87.000000	24.380000	165.300000	87.000000	14.050000	166.925000	87
50%	100.000000	415.000000	0.000000	179.95000	101.000000	30.590000	200.900000	100.000000	17.080000	201.150000	100
75%	127.000000	510.000000	19.000000	215.90000	114.000000	36.700000	235.100000	114.000000	19.980000	236.475000	113

max	243.000000	510.000000	50.000000	350.80000	160.000000	59.640000	363.700000	170.000000	30.910000	395.000000	166

4) find all the Null values

In [7]:

```
df.isnull().sum()
```

Out[7]:

```
State                     0
Account length            0
Area code                 0
International plan        0
Voice mail plan           0
Number vmail messages     0
Total day minutes         0
Total day calls           0
Total day charge          0
Total eve minutes         0
Total eve calls           0
Total eve charge          0
Total night minutes       0
Total night calls         0
Total night charge        0
Total intl minutes        0
Total intl calls          0
Total intl charge         0
Customer service calls    0
Churn                     0
dtype: int64
```

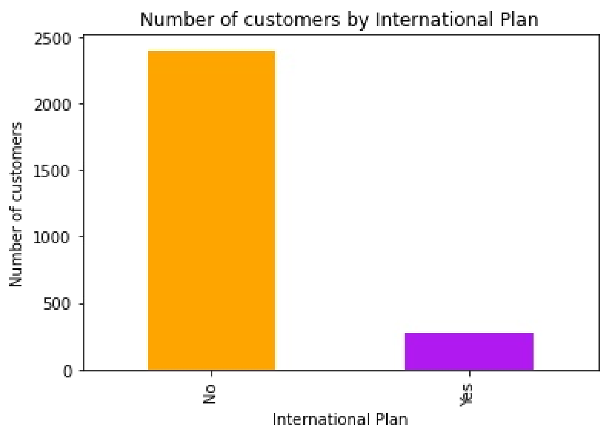
5) plot the customers who have international plans

In [8]:

```
df['International plan'].value_counts().plot(kind='bar', color=['orange','#b01af0']);
plt.title('Number of customers by International Plan')
plt.xlabel('International Plan')
plt.ylabel('Number of customers')
```

Out[8]:

```
Text(0, 0.5, 'Number of customers')
```



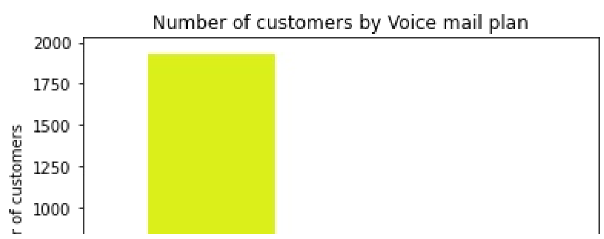
6) plot the customers who have Voice mail plan

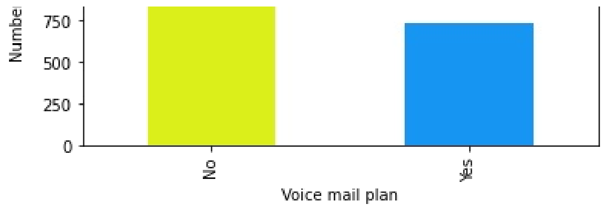
In [9]:

```
df['Voice mail plan'].value_counts().plot(kind='bar', color=['#dbf01a','#1696f2'])
plt.title('Number of customers by Voice mail plan')
plt.xlabel('Voice mail plan')
plt.ylabel('Number of customers')
```

Out[9]:

```
Text(0, 0.5, 'Number of customers')
```

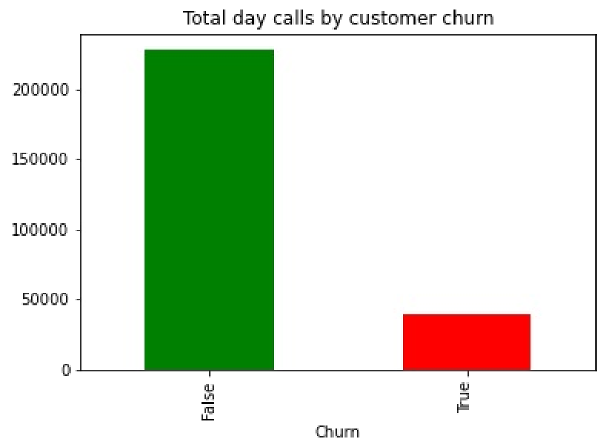




7) Plot the total day calls

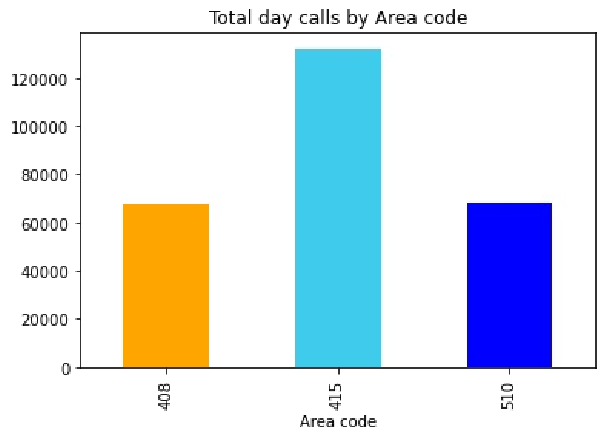
In [10]:

```
df.groupby('Churn')['Total day calls'].sum().plot(kind='bar', color=['green','red'])
plt.title('Total day calls by customer churn');
```



In [11]:

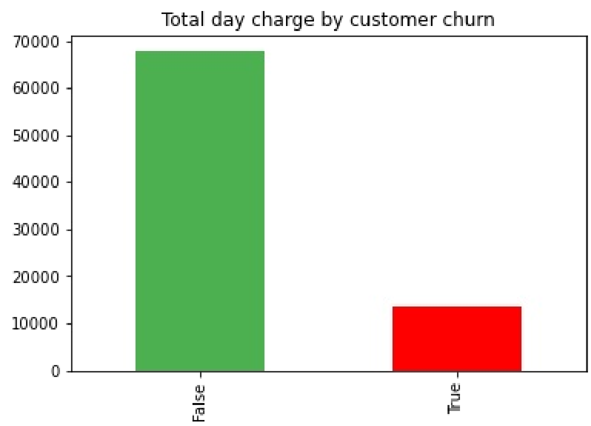
```
df.groupby('Area code')['Total day calls'].sum().plot(kind='bar', color = ['orange','#3fcbeb','blue']);
plt.title('Total day calls by Area code');
```



8) Plot the total day charge

In [12]:

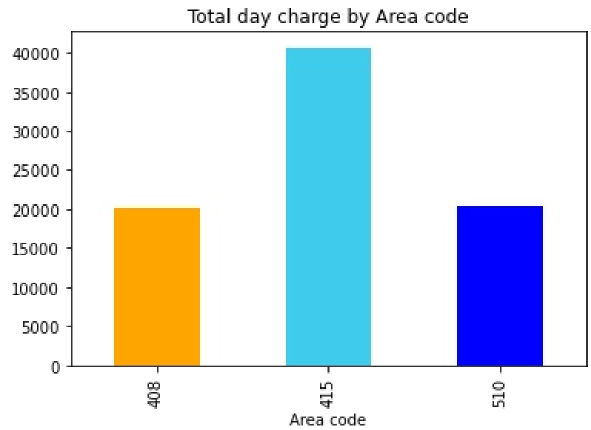
```
df.groupby('Churn')['Total day charge'].sum().plot(kind='bar', color=['#4CAF50', 'red']);
plt.title('Total day charge by customer churn');
```



Churn

In [13]:

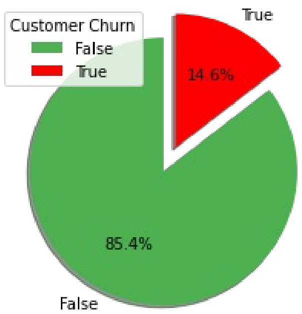
```
df.groupby('Area code')['Total day charge'].sum().plot(kind='bar', color=['orange','#3fcbeb','blue']);
plt.title('Total day charge by Area code');
```



9) Display pie chart for value count in Churn column

In [14]:

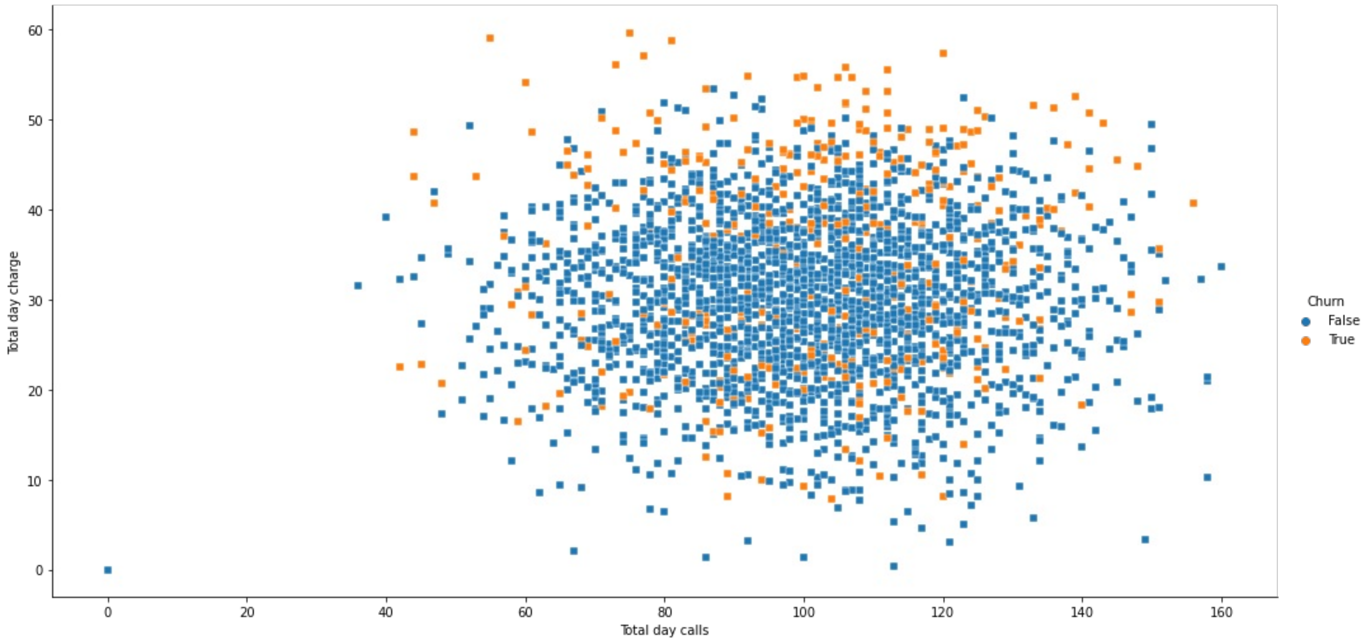
```
x=np.array(df['Churn'].value_counts())

plt.pie(x, labels=df['Churn'].unique(), data=df, autopct='%1.1f%%',startangle = 90, explode =[0,0.2], shadow = Tr
plt.legend(title='Customer Churn')
plt.show()
```



10) Display a scatter plot between total day calls and total day charges

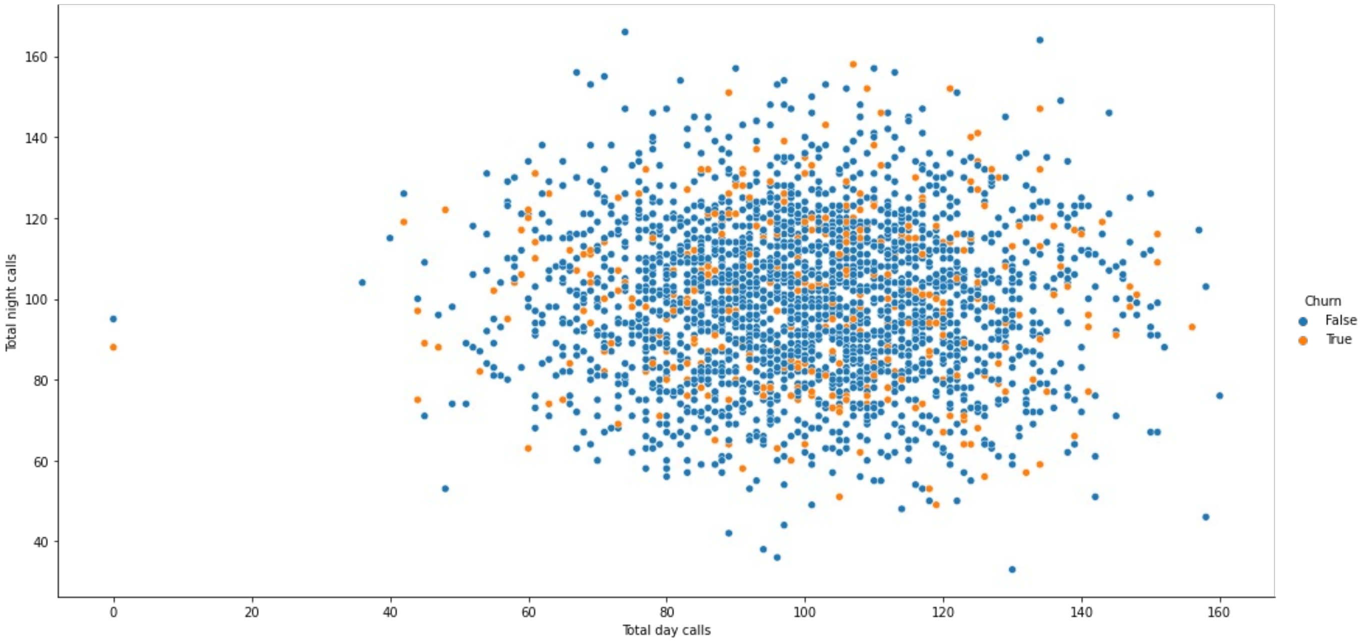
In [15]:

```
sns.relplot(x='Total day calls', y='Total day charge', hue='Churn', marker='s', data=df, height =7, aspect=2);
```



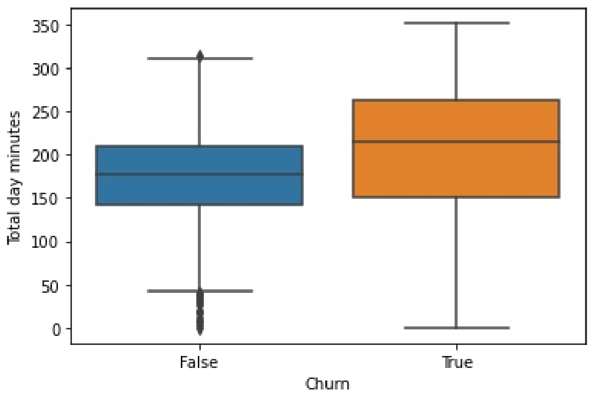
11) Display a scatter plot between total day calls and total night calls

In [16]:
```
sns.relplot(x='Total day calls', y='Total night calls', hue='Churn', data=df, height =7, aspect=2);
```



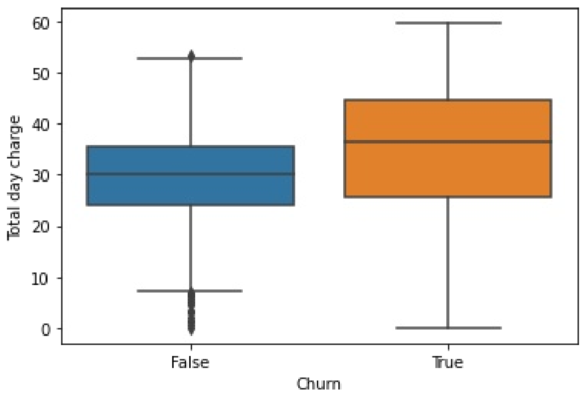
12) Display a boxplot of Total day minutes with respect to Churn

In [17]:
```
sns.boxplot(x='Churn', y='Total day minutes', data=df);
```



13) Display a boxplot of Total day charge with respect to Churn

In [18]:
```
sns.boxplot(x='Churn', y='Total day charge', data=df);
```



Part 2 - Working with models

1) Perform encoding on churn

In [19]:
```python
from sklearn.preprocessing import LabelEncoder
```

In [20]:
```python
le = LabelEncoder()
```

In [21]:
```python
df['Churn'] = le.fit_transform(df['Churn'])
```

In [22]:
```python
df.head()
```

Out[22]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Tot ir cal
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	

2) Perform encoding on International Plan

In [23]:
```python
df['International plan'] = le.fit_transform(df['International plan'])
```

In [24]:
```python
df.head()
```

Out[24]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Tot ir cal
0	KS	128	415	0	Yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	
1	OH	107	415	0	Yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	
2	NJ	137	415	0	No	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	
3	OH	84	408	1	No	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	
4	OK	75	415	1	No	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	

3) Perform encoding on voice mail plan

In [25]:
```python
df['Voice mail plan'] = le.fit_transform(df['Voice mail plan'])
```

In [26]:
```python
df.head()
```

Out[26]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Tot ir cal
0	KS	128	415	0	1	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	
1	OH	107	415	0	1	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	
2	NJ	137	415	0	0	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	
3	OH	84	408	1	0	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	
4	OK	75	415	1	0	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	

4) Check the correlation among all the columns

In [27]:
```python
correl = df.corr()
correl
```

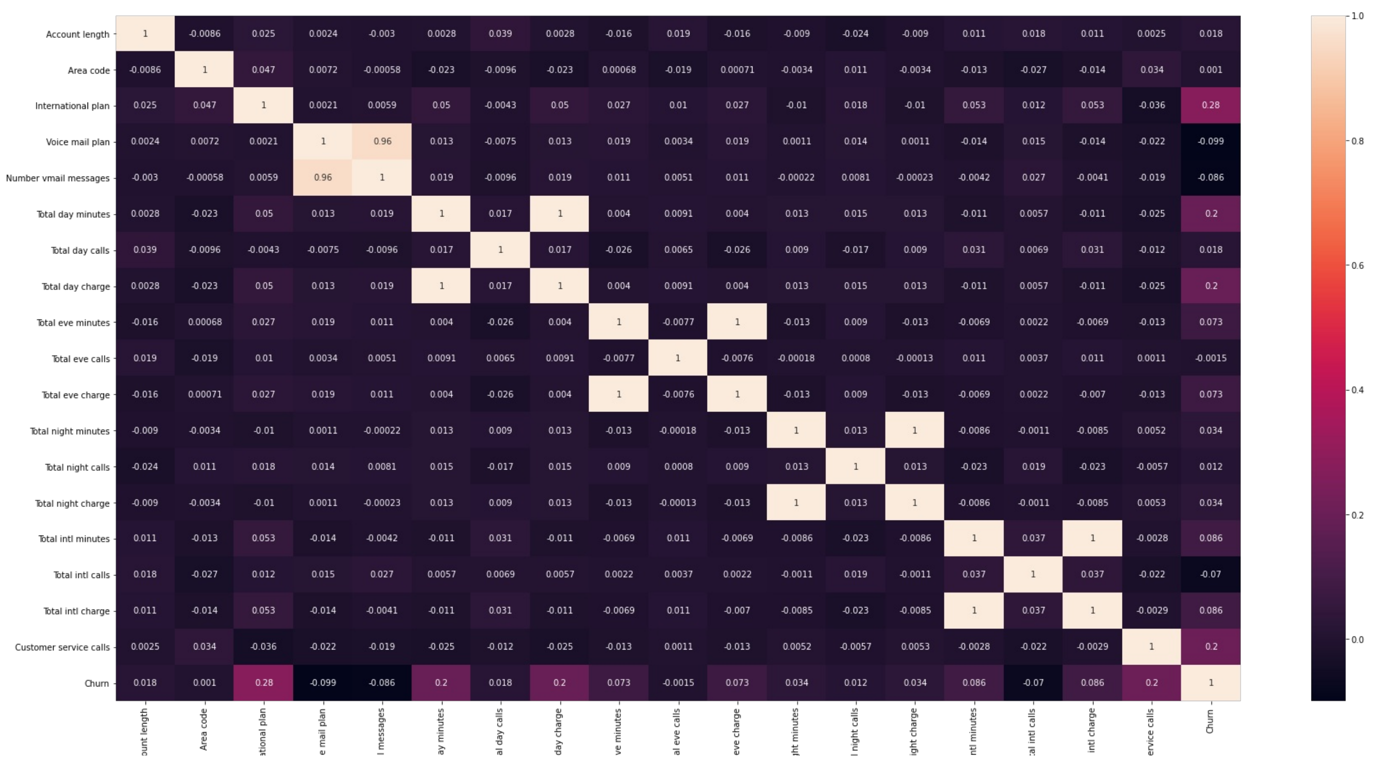
Out[27]:

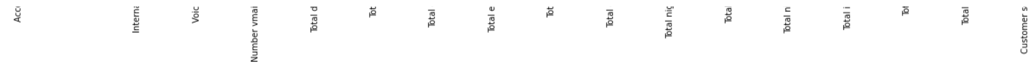
	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes
Account length	1.000000	-0.008620	0.024500	0.002448	-0.002996	0.002847	0.038862	0.002843	-0.015923	0.018552	-0.015909	-0.008994
Area code	-0.008620	1.000000	0.047099	0.007180	-0.000584	-0.023134	-0.009629	-0.023130	0.000679	-0.018602	0.000707	-0.003353
International plan	0.024500	0.047099	1.000000	0.002131	0.005858	0.049550	-0.004277	0.049555	0.026616	0.010277	0.026623	-0.010310
Voice mail plan	0.002448	0.007180	0.002131	1.000000	0.957159	0.013438	-0.007541	0.013439	0.019132	0.003404	0.019147	0.001065
Number vmail messages	-0.002996	-0.000584	0.005858	0.957159	1.000000	0.019027	-0.009622	0.019027	0.011401	0.005131	0.011418	-0.000224
Total day minutes	0.002847	-0.023134	0.049550	0.013438	0.019027	1.000000	0.016780	1.000000	0.003999	0.009059	0.003992	0.013491
Total day calls	0.038862	-0.009629	-0.004277	-0.007541	-0.009622	0.016780	1.000000	0.016787	-0.026003	0.006473	-0.026006	0.008986
Total day charge	0.002843	-0.023130	0.049555	0.013439	0.019027	1.000000	0.016787	1.000000	0.004008	0.009056	0.004002	0.013495
Total eve minutes	-0.015923	0.000679	0.026616	0.019132	0.011401	0.003999	-0.026003	0.004008	1.000000	-0.007654	1.000000	-0.013414
Total eve calls	0.018552	-0.018602	0.010277	0.003404	0.005131	0.009059	0.006473	0.009056	-0.007654	1.000000	-0.007642	-0.000175
Total eve charge	-0.015909	0.000707	0.026623	0.019147	0.011418	0.003992	-0.026006	0.004002	1.000000	-0.007642	1.000000	-0.013428
Total night minutes	-0.008994	-0.003353	-0.010310	0.001065	-0.000224	0.013491	0.008986	0.013495	-0.013414	-0.000175	-0.013428	1.000000
Total night calls	-0.024007	0.011455	0.018081	0.013985	0.008124	0.015054	-0.016776	0.015057	0.009017	0.000797	0.009030	0.012736
Total night charge	-0.008999	-0.003382	-0.010316	0.001066	-0.000229	0.013464	0.008972	0.013468	-0.013450	-0.000135	-0.013464	0.999999
Total intl minutes	0.011369	-0.013418	0.053162	-0.013963	-0.004156	-0.011042	0.031036	-0.011046	-0.006915	0.011012	-0.006923	-0.008607
Total intl calls	0.017627	-0.027423	0.011549	0.015196	0.027013	0.005687	0.006928	0.005688	0.002160	0.003710	0.002169	-0.001110
Total intl charge	0.011383	-0.013534	0.053037	-0.013931	-0.004136	-0.010934	0.031133	-0.010938	-0.006947	0.011000	-0.006955	-0.008510
Customer service calls	0.002455	0.034442	-0.035955	-0.022054	-0.018787	-0.024543	-0.011945	-0.024548	-0.013192	0.001058	-0.013196	0.005236
Churn	0.017728	0.001019	0.277489	-0.099291	-0.086474	0.195688	0.018290	0.195689	0.072906	-0.001539	0.072893	0.033639

In [28]:

```
plt.rcParams['figure.figsize'] = (30,15)
sns.heatmap(correl, annot=True)
```

Out[28]: <AxesSubplot:>





5) Create features and target data. Only select features data that are highly correlated with target data.

6) select target data (Churn)

In [29]:
```
x = df[['International plan','Total day minutes','Total day charge','Customer service calls']] #feature data
y = df['Churn'] # target data
```

7) Check the shape of both training data and testing data

In [30]:
```
from sklearn.model_selection import train_test_split
```

In [31]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

In [32]:
```
x_train.shape, x_test.shape
```

Out[32]:
```
((2132, 4), (534, 4))
```

In [33]:
```
y_train.shape, y_test.shape
```

Out[33]:
```
((2132,), (534,))
```

8) Apply Logistic regression

In [34]:
```
from sklearn.linear_model import LogisticRegression
```

In [35]:
```
lr = LogisticRegression()
```

In [36]:
```
lr.fit(x_train, y_train)
```

Out[36]:
```
LogisticRegression()
```

In [37]:
```
lr.score(x_train, y_train)
```

Out[37]:
```
0.8602251407129456
```

In [38]:
```
lr.score(x_test, y_test)
```

Out[38]:
```
0.8539325842696629
```

In [39]:
```
y_predict = lr.predict(x_test)
```

9) Display confusion matrix

In [40]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [41]:
```
cm = confusion_matrix(y_test, y_predict)
cm
```

Out[41]:
```
array([[444,  11],
```

```
       [ 67,  12]], dtype=int64)
```

In [42]:
```python
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       455
           1       0.52      0.15      0.24        79

    accuracy                           0.85       534
   macro avg       0.70      0.56      0.58       534
weighted avg       0.82      0.85      0.82       534
```

10) Perform Hyper parameter tuning

In [43]:
```python
from sklearn.model_selection import RandomizedSearchCV
```

In [55]:
```python
penalty = ['l1', 'l2', 'elasticnet', 'none']
solver = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
max_iter = [100,200,300,400,500, 600,700,800,900,1000,1100,1200]
multi_class = ['auto', 'ovr', 'multinomial']
```

In [56]:
```python
random_grid = {'penalty': penalty,
               'solver': solver,
               'max_iter': max_iter,
               'multi_class': multi_class}
```

In [57]:
```python
random_grid
```

Out[57]:
```
{'penalty': ['l1', 'l2', 'elasticnet', 'none'],
 'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
 'max_iter': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200],
 'multi_class': ['auto', 'ovr', 'multinomial']}
```

In [59]:
```python
lr_random = RandomizedSearchCV(estimator = lr,
                               param_distributions = random_grid,
                               scoring='neg_mean_squared_error',
                               n_iter = 10, cv = 5,
                               verbose=2,
                               random_state=42, n_jobs = 1)
```

In [60]:
```python
lr_random.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_iter=600, multi_class=multinomial, penalty=l1, solver=newton-cg; total time=   0.0s
[CV] END max_iter=600, multi_class=multinomial, penalty=l1, solver=newton-cg; total time=   0.0s
[CV] END max_iter=600, multi_class=multinomial, penalty=l1, solver=newton-cg; total time=   0.0s
[CV] END max_iter=600, multi_class=multinomial, penalty=l1, solver=newton-cg; total time=   0.0s
[CV] END max_iter=600, multi_class=multinomial, penalty=l1, solver=newton-cg; total time=   0.0s
[CV] END max_iter=500, multi_class=multinomial, penalty=elasticnet, solver=newton-cg; total time=   0.0s
[CV] END max_iter=500, multi_class=multinomial, penalty=elasticnet, solver=newton-cg; total time=   0.0s
[CV] END max_iter=500, multi_class=multinomial, penalty=elasticnet, solver=newton-cg; total time=   0.0s
[CV] END max_iter=500, multi_class=multinomial, penalty=elasticnet, solver=newton-cg; total time=   0.0s
[CV] END max_iter=500, multi_class=multinomial, penalty=elasticnet, solver=newton-cg; total time=   0.0s
[CV] END max_iter=100, multi_class=multinomial, penalty=elasticnet, solver=saga; total time=   0.0s
[CV] END max_iter=100, multi_class=multinomial, penalty=elasticnet, solver=saga; total time=   0.0s
[CV] END max_iter=100, multi_class=multinomial, penalty=elasticnet, solver=saga; total time=   0.0s
[CV] END max_iter=100, multi_class=multinomial, penalty=elasticnet, solver=saga; total time=   0.0s
[CV] END max_iter=100, multi_class=multinomial, penalty=elasticnet, solver=saga; total time=   0.0s
[CV] END max_iter=400, multi_class=auto, penalty=none, solver=sag; total time=   0.0s
[CV] END max_iter=400, multi_class=auto, penalty=none, solver=sag; total time=   0.0s
[CV] END max_iter=400, multi_class=auto, penalty=none, solver=sag; total time=   0.0s
[CV] END max_iter=400, multi_class=auto, penalty=none, solver=sag; total time=   0.0s
[CV] END max_iter=400, multi_class=auto, penalty=none, solver=sag; total time=   0.0s
[CV] END max_iter=800, multi_class=ovr, penalty=elasticnet, solver=sag; total time=   0.0s
[CV] END max_iter=800, multi_class=ovr, penalty=elasticnet, solver=sag; total time=   0.0s
[CV] END max_iter=800, multi_class=ovr, penalty=elasticnet, solver=sag; total time=   0.0s
[CV] END max_iter=800, multi_class=ovr, penalty=elasticnet, solver=sag; total time=   0.0s
```

```
[CV] END max_iter=800, multi_class=ovr, penalty=elasticnet, solver=sag; total time=   0.0s
[CV] END max_iter=200, multi_class=auto, penalty=l2, solver=newton-cg; total time=   0.0s
[CV] END max_iter=200, multi_class=auto, penalty=l2, solver=newton-cg; total time=   0.0s
[CV] END max_iter=200, multi_class=auto, penalty=l2, solver=newton-cg; total time=   0.0s
[CV] END max_iter=200, multi_class=auto, penalty=l2, solver=newton-cg; total time=   0.0s
[CV] END max_iter=200, multi_class=auto, penalty=l2, solver=newton-cg; total time=   0.0s
[CV] END max_iter=200, multi_class=auto, penalty=l1, solver=sag; total time=   0.0s
[CV] END max_iter=200, multi_class=auto, penalty=l1, solver=sag; total time=   0.0s
[CV] END max_iter=200, multi_class=auto, penalty=l1, solver=sag; total time=   0.0s
[CV] END max_iter=200, multi_class=auto, penalty=l1, solver=sag; total time=   0.0s
[CV] END max_iter=200, multi_class=auto, penalty=l1, solver=sag; total time=   0.0s
[CV] END max_iter=300, multi_class=ovr, penalty=l2, solver=newton-cg; total time=   0.0s
[CV] END max_iter=300, multi_class=ovr, penalty=l2, solver=newton-cg; total time=   0.0s
[CV] END max_iter=300, multi_class=ovr, penalty=l2, solver=newton-cg; total time=   0.0s
[CV] END max_iter=300, multi_class=ovr, penalty=l2, solver=newton-cg; total time=   0.0s
[CV] END max_iter=300, multi_class=ovr, penalty=l2, solver=newton-cg; total time=   0.0s
[CV] END max_iter=500, multi_class=multinomial, penalty=elasticnet, solver=liblinear; total time=   0.0s
[CV] END max_iter=500, multi_class=multinomial, penalty=elasticnet, solver=liblinear; total time=   0.0s
[CV] END max_iter=500, multi_class=multinomial, penalty=elasticnet, solver=liblinear; total time=   0.0s
[CV] END max_iter=500, multi_class=multinomial, penalty=elasticnet, solver=liblinear; total time=   0.0s
[CV] END max_iter=500, multi_class=multinomial, penalty=elasticnet, solver=liblinear; total time=   0.0s
[CV] END max_iter=700, multi_class=ovr, penalty=elasticnet, solver=newton-cg; total time=   0.0s
[CV] END max_iter=700, multi_class=ovr, penalty=elasticnet, solver=newton-cg; total time=   0.0s
[CV] END max_iter=700, multi_class=ovr, penalty=elasticnet, solver=newton-cg; total time=   0.0s
[CV] END max_iter=700, multi_class=ovr, penalty=elasticnet, solver=newton-cg; total time=   0.0s
[CV] END max_iter=700, multi_class=ovr, penalty=elasticnet, solver=newton-cg; total time=   0.0s
```

Out[60]:

```
RandomizedSearchCV(cv=5, estimator=LogisticRegression(), n_jobs=1,
                   param_distributions={'max_iter': [100, 200, 300, 400, 500,
                                                     600, 700, 800, 900, 1000,
                                                     1100, 1200],
                                        'multi_class': ['auto', 'ovr',
                                                        'multinomial'],
                                        'penalty': ['l1', 'l2', 'elasticnet',
                                                    'none'],
                                        'solver': ['newton-cg', 'lbfgs',
                                                   'liblinear', 'sag',
                                                   'saga']},
                   random_state=42, scoring='neg_mean_squared_error',
                   verbose=2)
```

In [61]:

```
lr_random.best_params_
```

Out[61]:

```
{'solver': 'newton-cg',
 'penalty': 'l2',
 'multi_class': 'auto',
 'max_iter': 200}
```

11) Create a model

In [63]:

```
lr_new = LogisticRegression(solver= 'newton-cg',
 penalty= 'l2',
 multi_class= 'auto',
 max_iter= 200)
```

In [65]:

```
lr_new.fit(x_train, y_train)
```

Out[65]:

```
LogisticRegression(max_iter=200, solver='newton-cg')
```

12) Check the model score of both training and testing data

In [66]:

```
lr_new.score(x_train, y_train)
```

Out[66]:

```
0.8602251407129456
```

In [67]:

```
lr_new.score(x_test, y_test)
```

Out[67]: 0.853932584269662９

In [68]:
```
y_newpred = lr_new.predict(x_test)
```

In [69]:
```
print(classification_report(y_test, y_newpred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       455
           1       0.52      0.15      0.24        79

    accuracy                           0.85       534
   macro avg       0.70      0.56      0.58       534
weighted avg       0.82      0.85      0.82       534
```

13) Perform cross validation technique with SVM Classifier

In [70]:
```
from sklearn.svm import SVC
```

In [71]:
```
from sklearn.model_selection import KFold,cross_val_score
```

In [72]:
```
models = []

models.append(('SVM', SVC()))



results = []
names = []
for name,model in models:
    kfold = KFold(n_splits=10)
    cv_result =cross_val_score(model,x_train,y_train,cv=kfold)
    names.append(name)
    results.append(cv_result)
for i in range(len(names)):
    print(names[i],results[i].mean())
```

```
SVM 0.8592733973937081
```

14) Perform hyperparameter tuning with different classifier models

In [73]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [74]:
```
model = [DecisionTreeClassifier, KNeighborsClassifier, RandomForestClassifier]

for mod in model:
    reg = mod()
    reg = reg.fit(x_train,y_train)
    print(mod , 'accuracy',reg.score(x_test,y_test))
```

```
<class 'sklearn.tree._classes.DecisionTreeClassifier'> accuracy 0.846441947565543
<class 'sklearn.neighbors._classification.KNeighborsClassifier'> accuracy 0.8558052434456929
<class 'sklearn.ensemble._forest.RandomForestClassifier'> accuracy 0.8520599250936329
```

In [86]:
```
rfc = RandomForestClassifier()
rfc.fit(x_train, y_train)
```

Out[86]: RandomForestClassifier()

In [87]:
```
y_pred = rfc.predict(x_test)
```

In [88]:
```
rfc.score(x_train, y_train)
```
Out[88]:
```
0.9901500938086304
```

In [89]:
```
rfc.score(x_test, y_test)
```
Out[89]:
```
0.850187265917603
```

In [90]:
```
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
min_samples_split = [2, 5, 10, 15, 100]
min_samples_leaf = [1, 2, 5, 10]
```

In [91]:
```
random_grid_rf = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
```

In [92]:
```
random_grid_rf
```
Out[92]:
```
{'n_estimators': [100,
  200,
  300,
  400,
  500,
  600,
  700,
  800,
  900,
  1000,
  1100,
  1200],
 'max_features': ['auto', 'sqrt'],
 'max_depth': [5, 10, 15, 20, 25, 30],
 'min_samples_split': [2, 5, 10, 15, 100],
 'min_samples_leaf': [1, 2, 5, 10]}
```

In [93]:
```
rfc_random = RandomizedSearchCV(estimator = rfc,
                               param_distributions = random_grid_rf,
                               scoring='neg_mean_squared_error',
                               n_iter = 10, cv = 5,
                               verbose=2,
                               random_state=42, n_jobs = 1)
```

In [94]:
```
rfc_random.fit(x_train,y_train)
```
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=
1.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=
1.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=
1.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=
1.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=
1.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time
=   2.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time
=   2.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time
=   2.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time
=   2.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time
=   2.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
```

```
= 0.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
= 0.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
= 0.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
= 0.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
= 0.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=
0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=
0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=
0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=
0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=
0.6s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=
1.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=
1.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=
1.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=
1.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=
1.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=
1.9s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=
2.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=
2.0s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=
2.0s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=
2.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time
= 1.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time
= 1.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time
= 1.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time
= 1.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time
= 1.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
0.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
0.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
0.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
0.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
0.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
0.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
0.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
0.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
1.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
0.9s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=
1.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=
1.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=
1.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=
1.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=
1.3s
```

Out[94]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=1,
                   param_distributions={'max_depth': [5, 10, 15, 20, 25, 30],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 5, 10],
```

```
                                        'min_samples_split': [2, 5, 10, 15,
                                                              100],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000, 1100,
                                                         1200]},
                   random_state=42, scoring='neg_mean_squared_error',
                   verbose=2)
```

In [95]:
```
rfc_random.best_params_
```

Out[95]:
```
{'n_estimators': 1100,
 'min_samples_split': 15,
 'min_samples_leaf': 10,
 'max_features': 'sqrt',
 'max_depth': 5}
```

In [98]:
```
rfc_new = RandomForestClassifier(n_estimators= 1100,
 min_samples_split= 15,
 min_samples_leaf= 10,
 max_features= 'sqrt',
 max_depth= 5)
rfc_new.fit(x_train, y_train)
```

Out[98]:
```
RandomForestClassifier(max_depth=5, max_features='sqrt', min_samples_leaf=10,
                       min_samples_split=15, n_estimators=1100)
```

In [99]:
```
y_pred_new = rfc_new.predict(x_test)
```

In [100…
```
rfc_new.score(x_train, y_train)
```

Out[100…
```
0.8977485928705441
```

In [101…
```
rfc_new.score(x_test, y_test)
```

Out[101…
```
0.8857677902621723
```

In [103…
```
print(confusion_matrix(y_test, y_pred_new))
```

```
[[442  13]
 [ 48  31]]
```

In [104…
```
print(classification_report(y_test, y_pred_new))
```

```
              precision    recall  f1-score   support

           0       0.90      0.97      0.94       455
           1       0.70      0.39      0.50        79

    accuracy                           0.89       534
   macro avg       0.80      0.68      0.72       534
weighted avg       0.87      0.89      0.87       534
```

15) Perform k-means clustering on dataset and divide it into four clusters

In [105…
```
from sklearn.cluster import KMeans
```

In [107…

```
df.drop('State', axis=1, inplace=True)
df.head()
```

Out[107…

	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls	T cha
0	128	415	0	1	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2
1	107	415	0	1	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3
2	137	415	0	0	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3
3	84	408	1	0	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1
4	75	415	1	0	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2

In [108…

```
km = KMeans(n_clusters=4)
km.fit(df)
```

Out[108… KMeans(n_clusters=4)

In [110…

```
y_predicted = km.fit_predict(df)
y_predicted
```

Out[110… array([0, 1, 0, ..., 0, 2, 0])

In [111…

```
df['cluster']=y_predicted
df
```

Out[111…

	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls
0	128	415	0	1	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3
1	107	415	0	1	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3
2	137	415	0	0	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5
3	84	408	1	0	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7
4	75	415	1	0	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2661	79	415	0	0	0	134.7	98	22.90	189.7	68	16.12	221.4	128	9.96	11.8	5
2662	192	415	0	1	36	156.2	77	26.55	215.5	126	18.32	279.1	83	12.56	9.9	6
2663	68	415	0	0	0	231.1	57	39.29	153.4	55	13.04	191.3	123	8.61	9.6	4
2664	28	510	0	0	0	180.8	109	30.74	288.8	58	24.55	191.9	91	8.64	14.1	6
2665	74	415	0	1	25	234.4	113	39.85	265.9	82	22.60	241.4	77	10.86	13.7	4

2666 rows × 20 columns

In [112…

```
km.cluster_centers_
```

Out[112…

```
array([[1.00031161e+02, 4.15873938e+02, 1.09065156e-01, 2.96033994e-01,
        8.90084986e+00, 2.38116856e+02, 9.98456091e+01, 4.04803683e+01,
        1.97854816e+02, 1.00300283e+02, 1.68178045e+01, 1.95705807e+02,
        9.99164306e+01, 8.80682720e+00, 1.02652975e+01, 4.51699717e+00,
        2.77225212e+00, 1.58215297e+00, 2.32294618e-01],
       [9.80287278e+01, 4.12904241e+02, 8.75512996e-02, 2.55813953e-01,
        7.50068399e+00, 1.50837073e+02, 1.00673051e+02, 2.56428181e+01,
        1.71694802e+02, 1.00179207e+02, 1.45941997e+01, 2.32922572e+02,
        1.00162791e+02, 1.04817647e+01, 1.03064295e+01, 4.45554036e+00,
        2.78325581e+00, 1.54582763e+00, 1.09439124e-01],
       [9.99671362e+01, 5.10000000e+02, 1.18935837e-01, 2.83255086e-01,
        8.09233177e+00, 1.74723631e+02, 1.00084507e+02, 2.97035837e+01,
        1.98037872e+02, 9.91815336e+01, 1.68337246e+01, 2.01925509e+02,
        1.00547731e+02, 9.08660407e+00, 1.01964006e+01, 4.35211268e+00,
        2.75341158e+00, 1.64475743e+00, 1.29890454e-01],
       [1.05244068e+02, 4.15054237e+02, 8.98305085e-02, 2.64406780e-01,
        7.53898305e+00, 1.49961356e+02, 1.00661017e+02, 2.54939661e+01,
        2.41506610e+02, 1.00411864e+02, 2.05282881e+01, 1.67544576e+02,
```

```
        9.97847458e+01, 7.53955932e+00, 1.01611864e+01, 4.54745763e+00,
        2.74394915e+00, 1.47118644e+00, 1.03389831e-01]])
```

In [113…

```python
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
df4 = df[df.cluster==3]
```

In [114…

```python
df1
```

Out[114…

	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls
0	128	415	0	1	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3
2	137	415	0	0	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5
3	84	408	1	0	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7
8	141	415	1	1	37	258.6	84	43.96	222.0	111	18.87	326.4	97	14.69	11.2	5
16	73	415	0	0	0	224.4	90	38.15	159.5	88	13.56	192.8	74	8.68	13.0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2648	181	408	0	0	0	229.9	130	39.08	144.4	93	12.27	262.4	110	11.81	14.2	4
2653	163	415	1	0	0	197.2	90	33.52	188.5	113	16.02	211.1	94	9.50	7.8	8
2657	62	408	0	0	0	321.1	105	54.59	265.5	122	22.57	180.5	72	8.12	11.5	2
2663	68	415	0	0	0	231.1	57	39.29	153.4	55	13.04	191.3	123	8.61	9.6	4
2665	74	415	0	1	25	234.4	113	39.85	265.9	82	22.60	241.4	77	10.86	13.7	4

706 rows × 20 columns

In [115…

```python
df2
```

Out[115…

	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls
1	107	415	0	1	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3
4	75	415	1	0	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3
7	147	415	1	0	0	157.0	79	26.69	103.1	94	8.76	211.8	96	9.53	7.1	6
9	74	415	0	0	0	187.7	127	31.91	163.4	148	13.89	196.0	94	8.82	9.1	5
10	168	408	0	0	0	128.8	96	21.90	104.9	71	8.92	141.1	128	6.35	11.2	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2651	149	415	0	1	18	148.5	106	25.25	114.5	106	9.73	178.3	98	8.02	6.5	4
2655	89	415	0	0	0	115.4	99	19.62	209.9	115	17.84	280.9	112	12.64	15.9	6
2659	78	408	0	0	0	193.4	99	32.88	116.9	88	9.94	243.3	109	10.95	9.3	4
2661	79	415	0	0	0	134.7	98	22.90	189.7	68	16.12	221.4	128	9.96	11.8	5
2662	192	415	0	1	36	156.2	77	26.55	215.5	126	18.32	279.1	83	12.56	9.9	6

731 rows × 20 columns

In [116…

```python
df3
```

Out[116…

	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls
5	118	510	1	0	0	223.4	98	37.98	220.6	101	18.75	203.9	118	9.18	6.3	6
6	121	510	0	1	24	218.2	88	37.09	348.5	108	29.62	212.6	118	9.57	7.5	7
11	95	510	0	0	0	156.6	88	26.62	247.6	75	21.05	192.3	115	8.65	12.3	5
14	93	510	0	0	0	190.7	114	32.42	218.2	111	18.55	129.6	121	5.83	8.1	3
15	76	510	0	1	33	189.7	66	32.25	212.8	65	18.09	165.7	108	7.46	10.0	5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2640	75	510	1	0	0	153.2	78	26.04	210.8	99	17.92	153.5	100	6.91	7.8	3

2641	71	510	1	0	0	186.1	114	31.64	198.6	140	16.88	206.5	80	9.29	13.8	5
2652	103	510	0	1	29	164.1	111	27.90	219.1	96	18.62	220.3	108	9.91	12.3	9
2656	122	510	1	0	0	140.0	101	23.80	196.4	77	16.69	120.1	133	5.40	9.7	4
2664	28	510	0	0	0	180.8	109	30.74	288.8	58	24.55	191.9	91	8.64	14.1	6

639 rows × 20 columns

In [117…:

```
df4
```

Out[117…:

	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls
12	62	415	0	0	0	120.7	70	20.52	307.2	76	26.11	203.0	99	9.14	13.1	6
13	85	408	0	1	27	196.4	139	33.39	280.9	90	23.88	89.3	75	4.02	13.8	4
17	147	415	0	0	0	155.1	117	26.37	239.7	93	20.37	208.8	133	9.40	10.6	4
21	174	415	0	0	0	124.3	76	21.13	277.1	112	23.55	250.7	115	11.28	15.5	5
23	54	408	0	0	0	134.3	73	22.83	155.5	100	13.22	102.1	68	4.59	14.7	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2647	128	415	0	0	0	147.7	94	25.11	283.3	83	24.08	188.3	124	8.47	6.9	5
2650	89	415	0	0	0	178.7	81	30.38	233.7	74	19.86	131.9	120	5.94	9.1	4
2654	52	415	0	0	0	124.9	131	21.23	300.5	118	25.54	192.5	106	8.66	11.6	4
2658	117	415	0	0	0	118.4	126	20.13	249.3	97	21.19	227.0	56	10.22	13.6	3
2660	96	415	0	0	0	106.6	128	18.12	284.8	87	24.21	178.9	92	8.05	14.9	7

590 rows × 20 columns

In [118…:

```
from sklearn.metrics import silhouette_score
```

In [119…:

```
score = silhouette_score(df, y_predicted)
score
```

Out[119…: 0.1461521115598611

16) Apply PCA give n components value to 3 show we only get 3 columns after applying PCA

In [34]:

```
from sklearn.preprocessing import StandardScaler
```

In [35]:

```
ss = StandardScaler()
```

In [36]:

```
x = ss.fit_transform(x)
```

In [40]:

```
df5 = pd.DataFrame(x, columns = ['International plan','Total day minutes','Total day charge','Customer service ca
df5.head()
```

Out[40]:

	International plan	Total day minutes	Total day charge	Customer service calls
0	-0.335690	1.579670	1.579942	-0.429172
1	-0.335690	-0.329918	-0.330194	-0.429172
2	-0.335690	1.179302	1.179465	-1.191955
3	2.978938	2.212509	2.212675	0.333610
4	2.978938	-0.235822	-0.235772	1.096392

In [41]:

```
from sklearn.decomposition import PCA
```

In [42]:

```
pca = PCA(n_components=3)
```

In [43]:

```
principal_components = pca.fit_transform(df5)
```

In [44]:

```
df6 = pd.DataFrame(principal_components, columns=['principal_component1', 'principal_component2', 'principal_comp
df6
```

Out[44]:

	principal_component1	principal_component2	principal_component3
0	-2.219156	0.087260	-0.596811
1	0.473185	-0.116485	-0.526572
2	-1.682836	-0.508415	-1.106757
3	-3.317512	-1.561759	2.309008
4	0.162151	-1.269996	2.923725
...	...	...	...
2661	1.200762	0.383545	0.016347
2662	0.641945	0.425833	0.001769
2663	-1.278526	1.126256	0.475604
2664	0.001506	0.474299	-0.014939
2665	-1.448753	-0.526130	-1.100650

2666 rows × 3 columns

In [46]:

```
df7 = pd.DataFrame(y, columns = ['Churn'])
df7.head()
```

Out[46]:

	Churn
0	0
1	0
2	0
3	0
4	0

In [48]:

```
df8 = pd.concat([df6, df7], axis=1 )
df8.head(50)
```

Out[48]:

	principal_component1	principal_component2	principal_component3	Churn
0	-2.219156	0.087260	-0.596811	0
1	0.473185	-0.116485	-0.526572	0
2	-1.682836	-0.508415	-1.106757	0
3	-3.317512	-1.561759	2.309008	0
4	0.162151	-1.269996	2.923725	0
5	-1.397124	-2.817264	1.311232	0
6	-0.942471	1.100825	0.484371	0
7	0.330044	-2.947969	1.356292	0
8	-2.312344	-2.748004	1.287356	0
9	-0.234044	-0.618054	-1.068960	0
10	1.325829	-0.181009	-0.504328	0
11	0.659546	0.979591	0.526166	0
12	1.621207	1.461906	1.075192	0
13	-0.432243	-0.047966	-0.550193	0
14	-0.227599	1.046726	0.503021	0
15	-0.257905	-0.061159	-0.545645	0
16	-1.160503	0.007146	-0.569192	0
17	0.613705	-0.682208	-1.046844	0
18	3.165628	1.900120	1.639422	1
19	-0.111724	-0.627311	-1.065769	0

20	2.538390	-0.827860	-0.996631	0
21	1.499569	0.916021	0.548081	0
22	-0.892002	-0.568262	-1.086125	0
23	1.239476	0.935704	0.541295	0
24	-0.293789	-0.613533	-1.070519	0
25	2.498394	0.285345	0.050201	0
26	-1.204795	0.010498	-0.570348	0
27	-0.781520	1.088645	0.488570	0
28	-1.815630	0.056723	-0.586284	1
29	0.049228	-0.639491	-1.061570	0
30	-0.989594	1.104391	0.483142	0
31	1.273811	-0.177073	-0.505685	0
32	-0.805088	-1.196799	2.898491	0
33	1.024275	-0.158189	-0.512195	0
34	1.447550	0.919958	0.546724	0
35	-0.088935	-2.916263	1.345361	1
36	1.458024	0.364076	0.023059	0
37	2.500543	0.840272	0.574195	0
38	0.670702	-0.131432	-0.521419	0
39	1.559996	0.356359	0.025719	0
40	-0.159446	0.486479	-0.019138	0
41	1.212001	-0.172395	-0.507297	0
42	-0.258504	1.049065	0.502215	0
43	-1.061361	-0.000357	-0.566606	0
44	0.537908	0.433707	-0.000945	0
45	0.861360	2.074497	1.579306	1
46	0.111121	-0.089085	-0.536018	0
47	1.375783	-0.184790	-0.503024	0
48	-0.427947	1.061888	0.497795	1
49	-1.287718	0.016773	-0.572511	0