Students' Information ¶

- Programme : RDS Y2 S1
- Tutorial Group : G2

Students' name and ID

1. ANG CHIA LING 20WMR08861
2. KOAY MEI SIEW 20WMR08866
3. LEE JING HUI 20WMR08872
4. QUAH TZE WEI 20WMR08885

Business Understanding

The telecommunication sector is made up of companies that make communication possible on a global scale, whether it is through the phone or the Internet, through airwaves or cables, through wires or wirelessly. These companies created the infrastructure that allows data in words, voice, audio or video to be sent anywhere in the world.

Telephone service companies, Internet service providers, pay TV companies, insurance firms, and alarm monitoring services, often use customer churn analysis and customer attrition rates as one of their key business metrics because the cost of retaining an existing customer is far less than acquiring a new one. Companies from these sectors often have customer service branches which attempt to win back defecting clients, because recovered long-term customers can be worth much more to a company than newly recruited clients.

Customer churn happens when customers quit using a company’s service or stop doing business with a company. By doing the prediction on customer churn rate, companies can easily monitor and be alert of churn rate. Thus, they will know what brings to their customer retention success rate and develop strategies to further improve the business.

The customer churn is an important activity in a growing and competitive telecommunication sector and is one of the greatest importance for a project manager. Due to the high cost of acquiring new customers, customer churn prediction has emerged as an indispensable part of telecom’s sectors’ strategic decision making and planning process.

In [1]:
```python
# Importing libraries
# General
import pandas as pd
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick

#Visualisation(EDA)
import plotly.graph_objs as go
import plotly.offline as py
sns.set(style = 'white')
%matplotlib inline

# Train test split
from sklearn.model_selection import train_test_split

# Data prepocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

# Modelling
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV


# Evaluation
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import plotly.figure_factory as ff   #visualization

# Confusion Matrix
import matplotlib.pyplot as plt
import itertools
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score

#ROC
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import recall_score

# K fold
from sklearn.model_selection import KFold

pd.set_option('mode.chained_assignment', None)
```

Data Understanding

In this assignment, we are going to use “Telcom_Customer_Churn” which is found in Kaggle to predict the customer churn rate. We are going to select the best modeling technique to predict our target attribute, which is the customer churn. The details of the dataset are described as below:-

Each row of the data will represent each customer, each column of the data represents the customers’ attributes which are described on the metadata of the customer. The dataset represents the details about:-

- Demographic data about customers – gender, whether they are senior citizens, having partners and dependents.
- Types of services signed up by each customer which included phone service, multiple lines, internet service, online security, online backup, device protection, technical support, streaming TV and movies.
- Customers who left within the telcom company which is also the target attributes,
- called Churn Account information of each customer – how long he/she have been a customer of telcom company(tenure), contract , paperless billing, payment method, monthly charges and total charges has been charged since they‘re one of the customers at telcom company.

There are a total of 7043 rows and 21 columns of data in this dataset,which means that this dataset contains 7043 unique customers. Within these 21 columns of key attributes, all attributes can be classified or transformed to categorical variables whereas tenure, monthly charges and total charges are classified as continuous variables and need to be scaled before modelling. Also, we can see that the “Total Charges” is needed to be transformed to numerical values as we found that it is declared as object data type earlier. And, we found that there are 11 of missing data in this dataset. Hence, we choose to remove the entire row of data whenever it finds a missing value. It is because removing an entire row of data with missing values will result in a more robust and accurate model and it only consists of 11 rows of data with missing values, thus it will not create a great loss on information of data. Furthermore, we have chosen to replace ‘no internet service’ and ‘no phone service’ in several columns to ‘no’ to ease data classification and ease our modelling section. All values in the attributes also have been changed to lowercase letters to ease the coding part.

In [2]:
```
# Reading datasets
data = pd.read_csv('telcom_churn.csv')
data.head()
```

Out[2]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	In
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	

2	3668-QPYBK	Male	0	No	No	2	Yes	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service
4	9237-HQITU	Female	0	No	No	2	Yes	No

5 rows × 21 columns

In [3]:
```
# Check the data types of all the columns
print("Data Types: \n",data.dtypes)
```

```
Data Types:
 customerID
object gender
object SeniorCitizen
int64
Partner
object Dependents
object tenure
int64 PhoneService
object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges
object Churn
object dtype: object
```

In [4]:
```
# Change data types to numeric value
data['TotalCharges'] = data['TotalCharges'].replace(r'\s+', np.nan,
regex=True) data['TotalCharges'] = pd.to_numeric(data['TotalCharges'])
```

In [5]:
```
# Display details of dataset
print ("Rows     : " ,data.shape[0])
print ("Columns  : " ,data.shape[1])
print ("\nFeatures : \n" ,data.columns.tolist())
print ("\nMissing values :  \n", data.isnull().sum())
print ("\nUnique values :  \n",data.nunique())
```

```
Rows     :  7043
Columns  :  21

Features :
 ['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'Online
Backup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
'Contract', 'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges', 'TotalChar
ges', 'Churn']

Missing values :
 customerID           0
gender               0
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64

Unique values :
 customerID          7043
gender                 2
SeniorCitizen          2
Partner                2
Dependents             2
tenure                73
PhoneService           2
MultipleLines          3
InternetService        3
OnlineSecurity         3
OnlineBackup           3
DeviceProtection       3
TechSupport            3
StreamingTV            3
```

In [7]:
```
# Remove customerID variable
data = data.drop('customerID', axis = 1)
data.head()
```

Out[7]:
```
StreamingMovies        3
Contract               3
PaperlessBilling       2
PaymentMethod          4
MonthlyCharges      1585
TotalCharges        6530
Churn                  2
dtype: int64
```

In [6]:
```
# Remove the missing values
data.dropna(inplace=True)
print ("\nMissing values :  \n", data.isnull().sum())
```

```
 Missing    values    :
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
0	Female	0	Yes	No	1	No	No phone service	DSL
1	Male	0	No	No	34	Yes	No	DSL
2	Male	0	No	No	2	Yes	No	DSL
3	Male	0	No	No	45	No	No phone service	DSL
4	Female	0	No	No	2	Yes	No	Fiber optic

In [8]:
```python
# Print all unique values for each columns
for item in data.columns:
    print(item)
    print (data[item].unique())
```

```
gender
['Female' 'Male']

SeniorCitizen
[0 1]
Partner
['Yes' 'No']
Dependents ['No'
'Yes'] tenure
[ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26 39]
PhoneService
['No' 'Yes']
MultipleLines
['No phone service' 'No' 'Yes']
InternetService
['DSL' 'Fiber optic' 'No']
OnlineSecurity
['No' 'Yes' 'No internet service'] OnlineBackup
['Yes' 'No' 'No internet service'] DeviceProtection
['No' 'Yes' 'No internet service'] TechSupport
['No' 'Yes' 'No internet service'] StreamingTV
['No' 'Yes' 'No internet service'] StreamingMovies
['No' 'Yes' 'No internet service']
Contract ['Month-to-month' 'One year'
'Two year']
PaperlessBilling
['Yes' 'No']
PaymentMethod
['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
MonthlyCharges
[29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
TotalCharges
[  29.85 1889.5   108.15 ...  346.45  306.6  6844.5 ] Churn
['No' 'Yes']
```

In [9]:
```python
# Convert all columns value to lowercase
for item in data.columns:
    try:
        data[item] = data[item].str.lower()
    except:
        print(item, "couldn't convert")
data.head()
```

```
SeniorCitizen couldn't convert
tenure couldn't convert MonthlyCharges
couldn't convert TotalCharges couldn't
convert
```

Out[9]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
0	female	0	yes	no	1	no	no phone service	dsl
1	male	0	no	no	34	yes	no	dsl
2	male	0	no	no	2	yes	no	dsl
3	male	0	no	no	45	no	no phone service	dsl
4	female	0	no	no	2	yes	no	fiber optic

```
In[10]: # Convert "no internet service" and "no phone service" into "no"

cols_replace_1 = ['OnlineSecurity', 'OnlineBackup', 'DeviceProtection','TechSuppo
for i in cols_replace_1 :
   data[i]  = data[i].replace({'no internet service' : 'no'})

cols_replace_2 = ['MultipleLines']
for i in cols_replace_2 :     data[i]  =
   data[i].replace({'no phone service' : 'no'})
cols_replace_3 = ['SeniorCitizen']
for i in cols_replace_3:
   data[i].replace(to_replace = 0, value = 'no', inplace = True)
   data[i].replace(to_replace = 1 , value = 'yes', inplace =True)
data.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
0	female	no	yes	no	1	no	no	dsl
1	male no	no	no	34	yes	no	dsl	
2	male no	no	no	2	yes	no	dsl	
3	male no	no	no	45	no	no	dsl	
4	female	no	no	no	2	yes	no	fiber optic

Exploratory Data Analysis(EDA)

Exploratory Data Analysis has been carried out to study the relationship between each attribute and customer churn rate. The pie charts below shows the pie plot for customer attrition based on different categories in different columns.

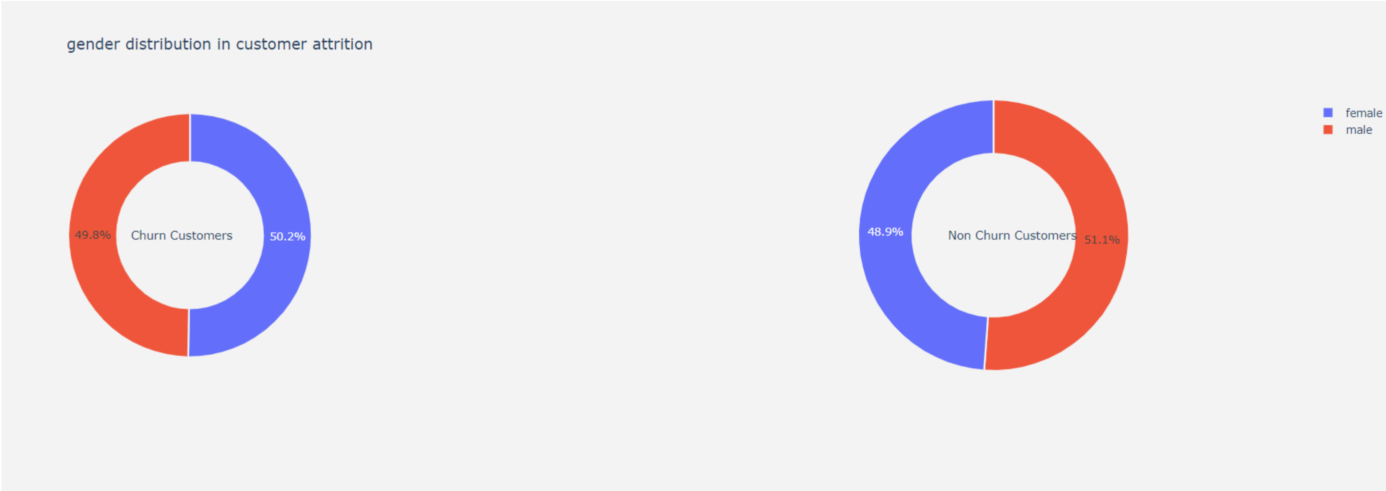
In[11]:
```python
# Split the dataset into 80% train data and 20% test data train_data,
test_data =  train_test_split(data, test_size=0.2, random_state=0)
train_data.head()
```

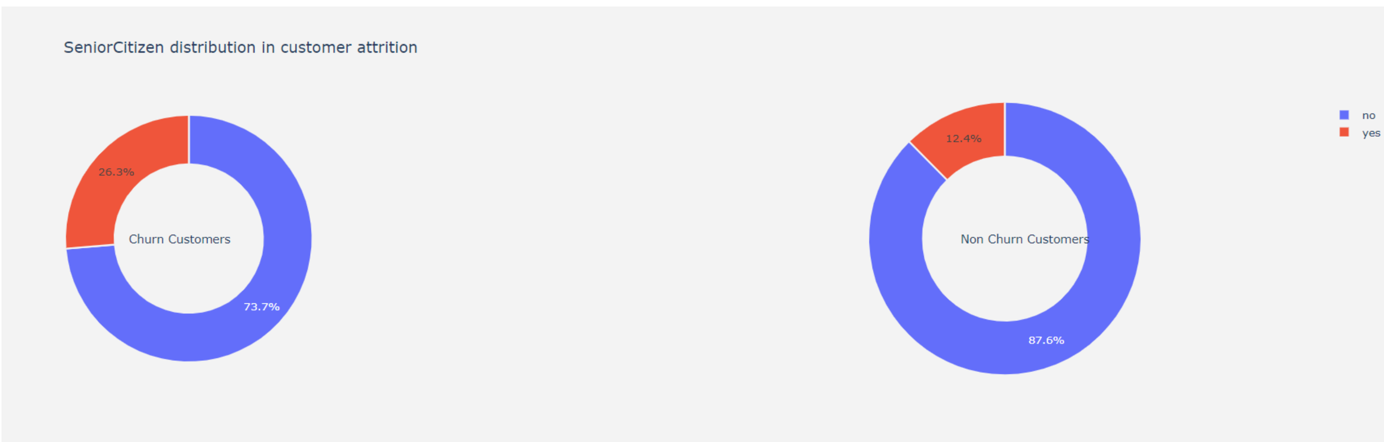
Out[11]:

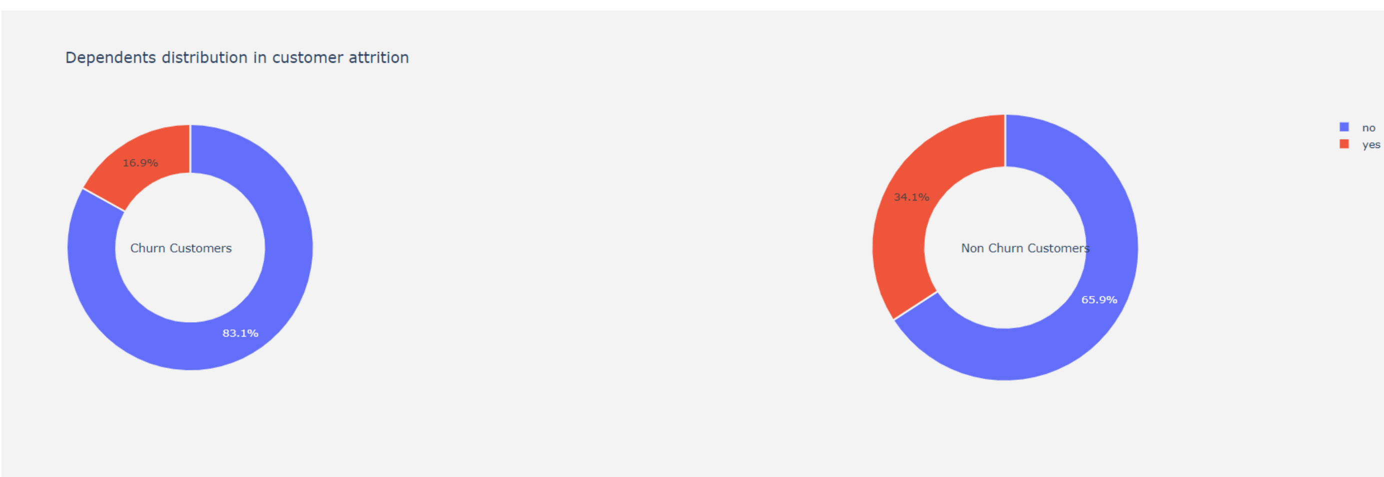
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetServ
2964	male	no	yes	no	24	yes	yes	
5113	female	no	yes	yes	71	yes	yes	fiber o
5363	male	no	yes	yes	70	yes	yes	
5074	female	no	no	yes	49	yes	no	
156	female	no	no	no	22	yes	yes	fiber o

In [12]:
```python
# Separating churn and non churn customers
cust_churn = train_data[train_data["Churn"] == "yes"]
cust_not_churn = train_data[train_data["Churn"] =="no"]

# Separating catagorical and numerical columns to do EDA
target_col = ["Churn"]
cat_cols   = train_data.nunique()[train_data.nunique() < 6].keys().tolist()
cat_cols   = [x for x in cat_cols if x not in target_col]
num_cols   = [x for x in train_data.columns if x not in cat_cols + target_col]
```

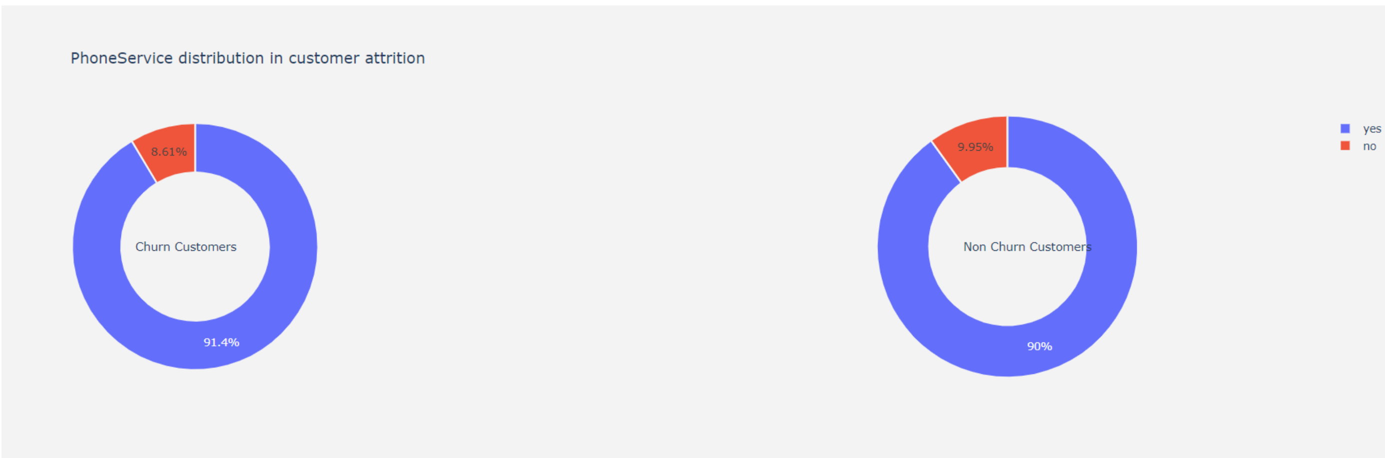
In [13]:
```
# Function for pie plot on data variables with customer
churn def plot_pie(column) :
   #Churn customers
    trace1 = go.Pie(values  = cust_churn[column].value_counts().values.tolist(),
                    labels = cust_churn[column].value_counts().keys().tolist(),
                    hoverinfo = "label+percent+name",
                    domain  = dict(x = [0,.20]),
                    name    = "Churn Customers",
                    marker  = dict(line = dict(width = 2,
                                              color = "rgb(243,243,243)")
                                 ),
                    hole    = .6
                  )
   #Non churn customers
    trace2 = go.Pie(values  = cust_not_churn[column].value_counts().values.tolist
                    labels = cust_not_churn[column].value_counts().keys().tolist
                    hoverinfo = "label+percent+name",
                    marker  = dict(line = dict(width = 2,
                                              color = "rgb(243,243,243)")
                                 ),
                    domain  = dict(x=[.52,1]),
                    hole    = .6,
                    name    = "Non Churn Customers"
                  )

    layout = go.Layout(dict(title = column + " distribution in customer attrition
                          plot_bgcolor = "rgb(243,243,243)",
                          paper_bgcolor = "rgb(243,243,243)",
                          annotations = [dict(text = "Churn Customers",
                          font = dict(size = 13),
                          showarrow = False,
                          x = .05, y = .5),
                    dict(text = "Non Churn Customers",
                          font = dict(size = 13),
                          showarrow = False,
                          x = .83,y = .5
                                          )
                                      ]
                          )
                     )
   data = [trace1,trace2
   fig=go.Figure(data=data,
         layout = layout)
   py.iplot(fig)

# To run plot pie functions for all categorical columns
for i in cat_cols :
   plot_pie(i)
```

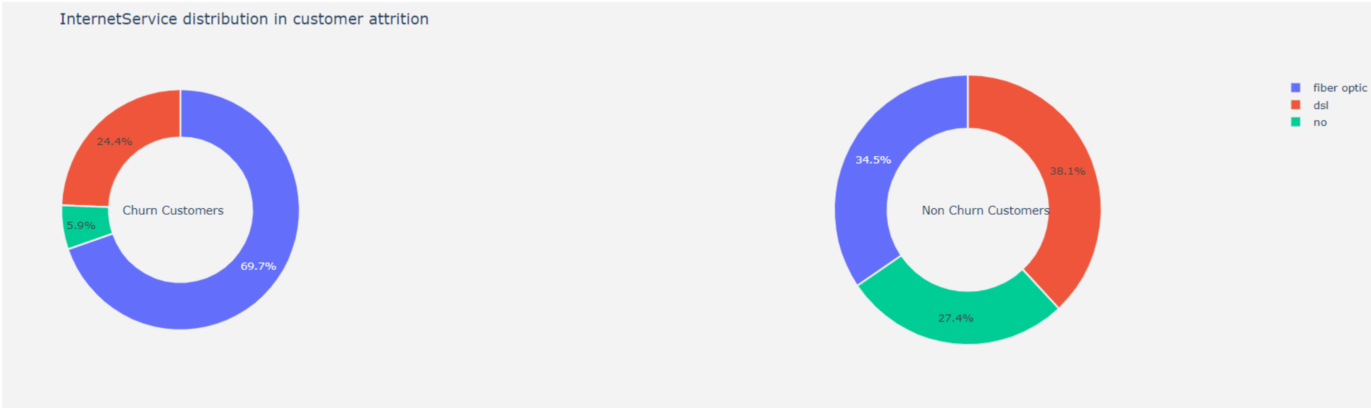

gender distribution in customer attrition
49.8%
Churn Customers
50.2%
48.9%
Non Churn Customers
51.1%
female
male

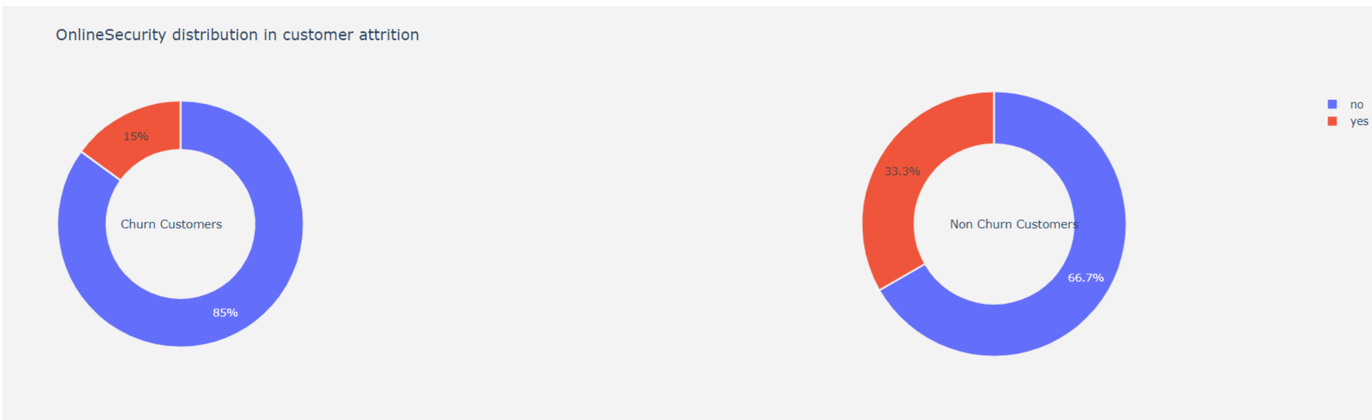

SeniorCitizen distribution in customer attrition
26.3%
Churn Customers
73.7%
12.4%
Non Churn Customers
87.6%
no
yes


Dependents distribution in customer attrition
16.9%
Churn Customers
83.1%
34.1%
Non Churn Customers
65.9%
no
yes


MultipleLines distribution in customer attrition
44.7%
Churn Customers
55.3%
40.7%
Non Churn Customers
59.3%
no
yes


PhoneService distribution in customer attrition
8.61%
Churn Customers
91.4%
9.95%
Non Churn Customers
90%
yes
no

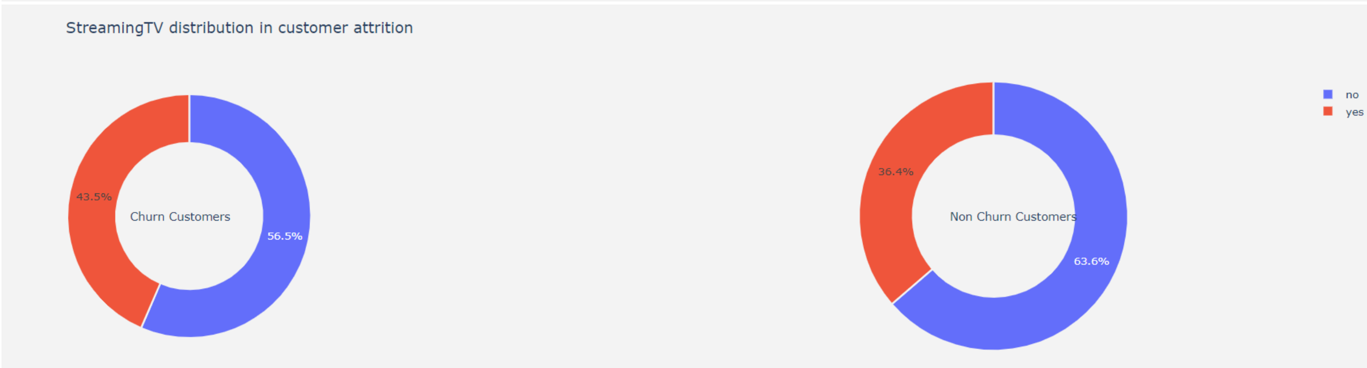

InternetService distribution in customer attrition
Churn Customers
69.7%
24.4%
5.9%
Non Churn Customers
34.5%
38.1%
27.4%
fiber optic
dsl
no

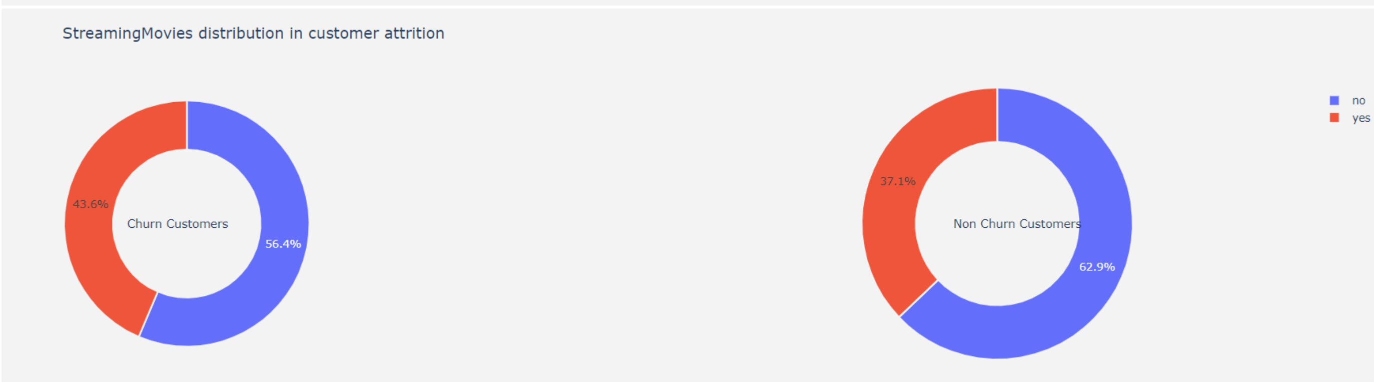

OnlineSecurity distribution in customer attrition
Churn Customers
85%
15%
Non Churn Customers
66.7%
33.3%
no
yes

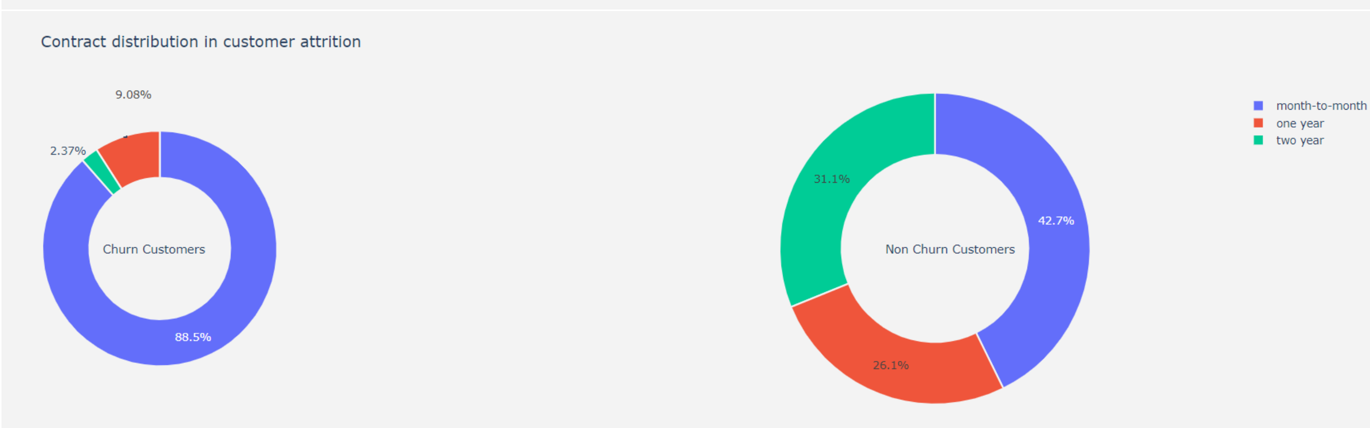

OnlineBackup distribution in customer attrition
Churn Customers
72.1%
27.9%
Non Churn Customers
62.6%
37.4%
no
yes


DeviceProtection distribution in customer attrition
Churn Customers
71.2%
28.8%
Non Churn Customers
63.5%
36.5%
no
yes

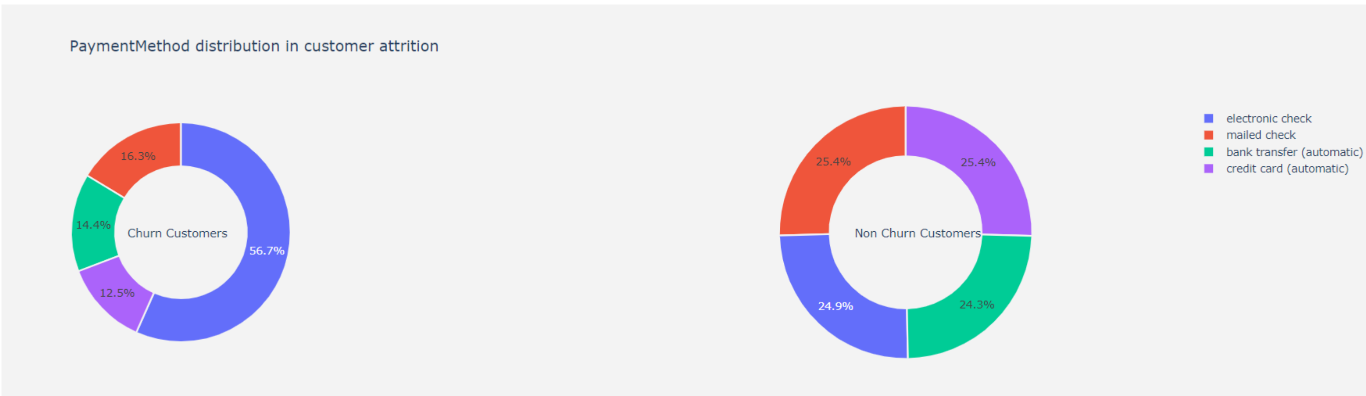

TechSupport distribution in customer attrition
16.3%
Churn Customers
83.7%
34%
Non Churn Customers
66%
no
yes


StreamingTV distribution in customer attrition
43.5%
Churn Customers
56.5%
36.4%
Non Churn Customers
63.6%
no
yes

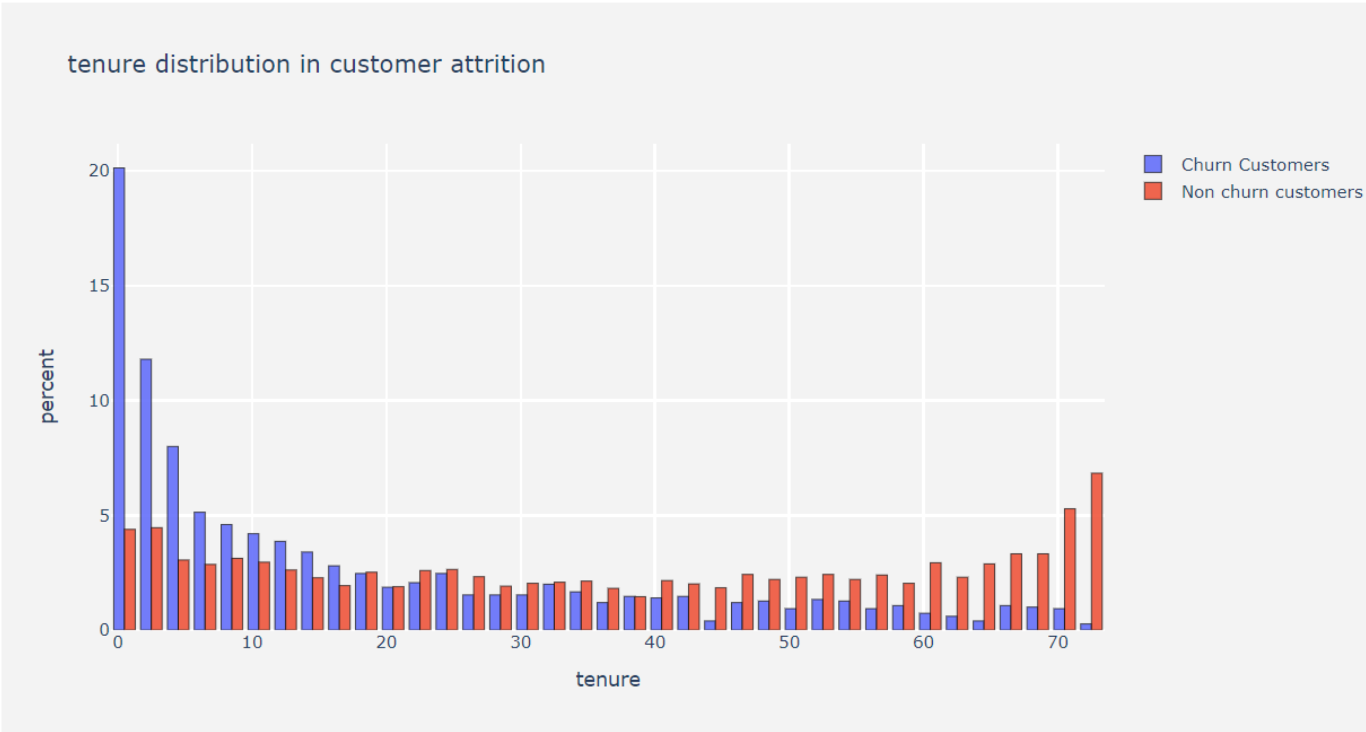

StreamingMovies distribution in customer attrition
43.6%
Churn Customers
56.4%
37.1%
Non Churn Customers
62.9%
no
yes

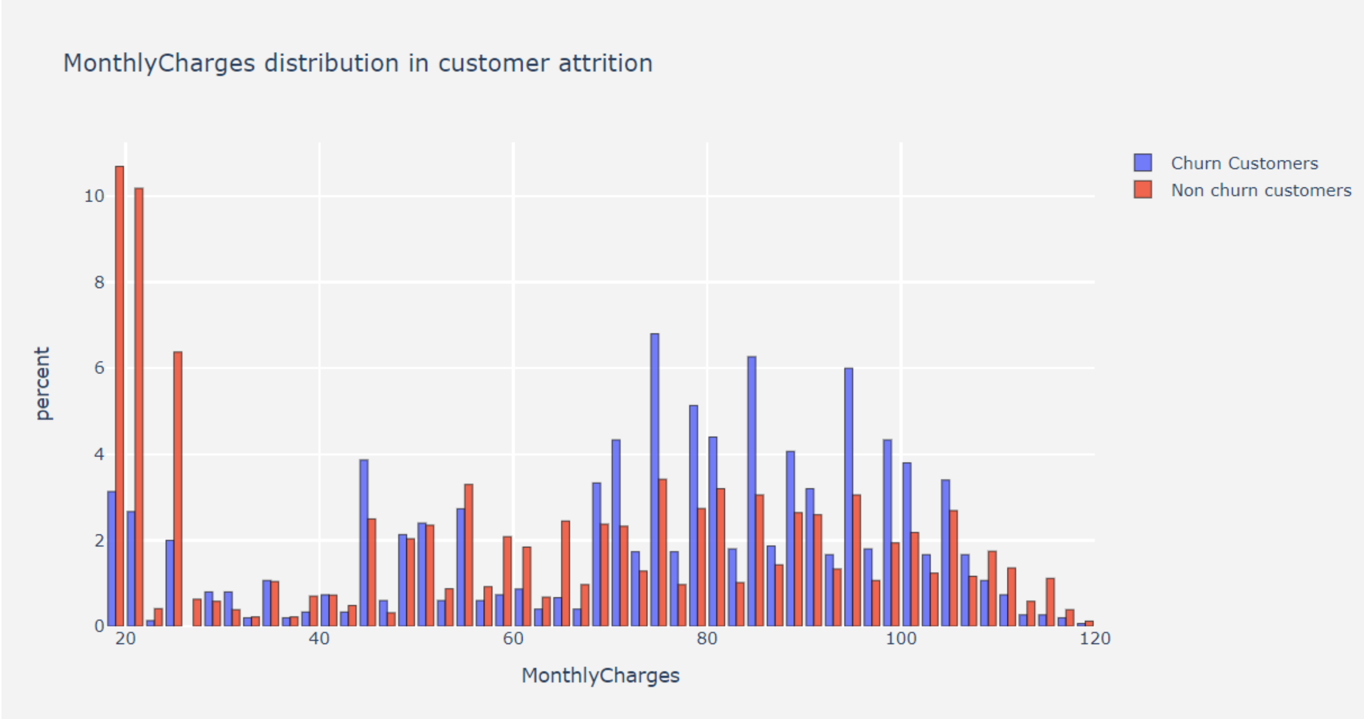

Contract distribution in customer attrition
9.08%
2.37%
Churn Customers
88.5%
31.1%
Non Churn Customers
42.7%
26.1%
month-to-month
one year
two year

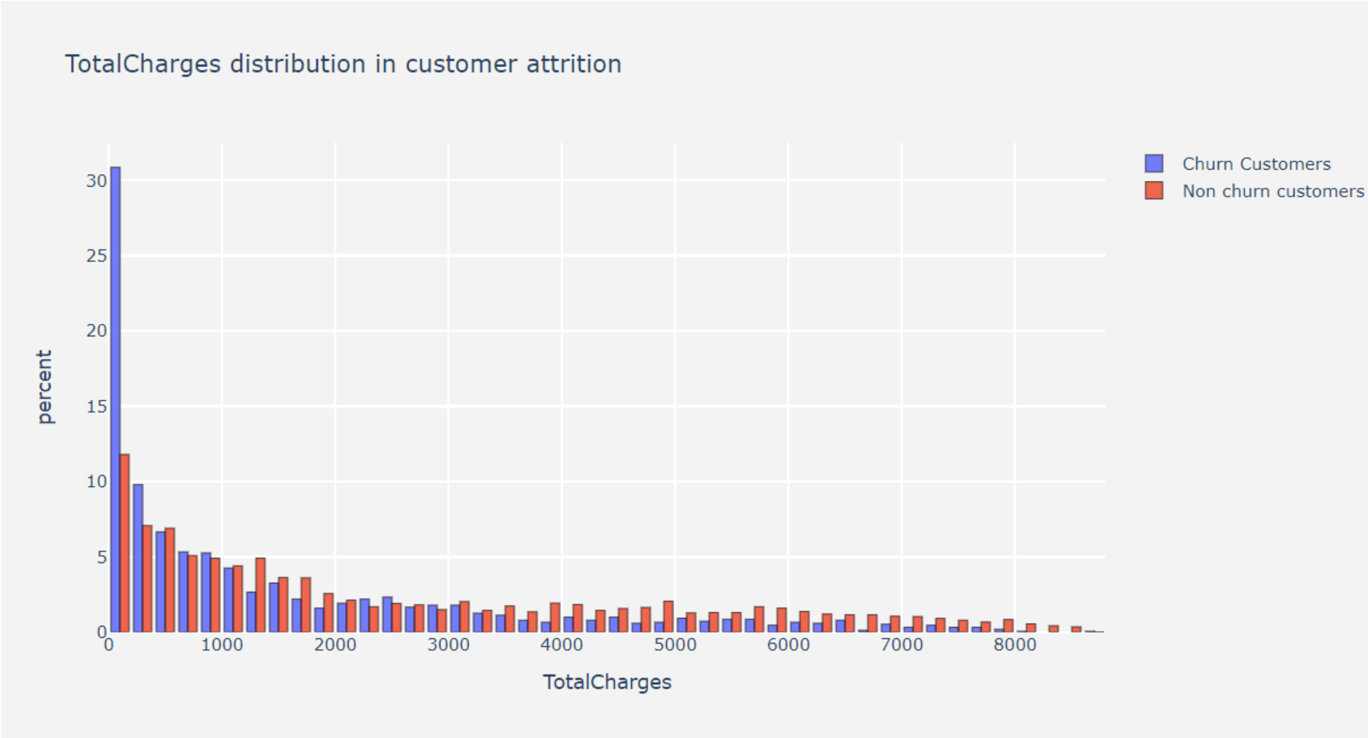

PaperlessBilling distribution in customer attrition
Churn Customers
24.9%
75.1%
Non Churn Customers
46.8%
53.2%
yes
no


PaymentMethod distribution in customer attrition
Churn Customers
16.3%
14.4%
12.5%
56.7%
Non Churn Customers
25.4%
25.4%
24.9%
24.3%
electronic check
mailed check
bank transfer (automatic)
credit card (automatic)

In [14]:
```
# Function for histogram on data variables with customer churn
def histogram(column) :
   trace1 = go.Histogram(x  = cust_churn[column],
                          histnorm= "percent",
                          name = "Churn Customers",
                          marker = dict(line = dict(width = .5,
                                                    color = "black"
                                                    )
                                       ),
                          opacity = .9
                          )

    trace2 = go.Histogram(x  = cust_not_churn[column],
                          histnorm = "percent",
                          name = "Non churn customers",
                          marker = dict(line = dict(width = .5,
                                                  color="black"
                                                   )
                                 ),
                          opacity = .9
                          )

    data = [trace1,trace2]
    layout = go.Layout(dict(title =column + " distribution in customer attrition
                            plot_bgcolor  = "rgb(243,243,243)",
                            paper_bgcolor = "rgb(243,243,243)",
                            xaxis  =  dict(gridcolor  =  'rgb(255,  255,  255)',
                                              title = column,
                                              zerolinewidth=1,
                                              ticklen=5,
                                              gridwidth=2
                                             ),
                             yaxis = dict(gridcolor = 'rgb(255, 255, 255)',
                                              title = "percent",
                                              zerolinewidth=1,
                                              ticklen=5,
                                              gridwidth=2
                                             ),
                            )
                       )
    fig  = go.Figure(data=data,layout=layout)

    py.iplot(fig)

# To run histogram functions for all numerical columns
for i in num_cols :
    histogram(i)
```


tenure distribution in customer attrition
Churn Customers
Non churn customers
percent
tenure
0
5
10
15
20
0
10
20
30
40
50
60
70


MonthlyCharges distribution in customer attrition
Churn Customers
Non churn customers
percent
MonthlyCharges
0
2
4
6
8
10
20
40
60
80
100
120



Gender

- This chart below shows the gender distribution in the customer attrition. It is shown that the percentage of female is more than male for churn customers but the percentage of male is slightly more than female for non churn customers.

Senior Citizen

- This chart is used to see the distribution of senior citizens in churn customer and non churn customer. Based on both pie charts, we can see that there is not much difference in the distribution of senior citizens between churn customers and non-churn customers. The percentage of senior citizens is less in both pie charts.

Partner

- The pie chart below shows the partner distribution in customer attrition. From here, we can see that the percentage of churn customers with no partner is more whereas the percentage of non churn customers with no partner is slightly lesser.

Dependents

- This pie chart shows the dependents distribution in customer attrition. The percentage of dependents in both churn and non churn customers is more compared to no dependents. Thus, the relationship between the attributes and customer churn rate is not strong.

Tenure

- By looking at the histogram below, we can see that a lot of churn customers have been with the telecom company for only less than a month. This could be potentially because different customers have different contracts. Thus by looking at the contract they are into, it could be easier for the customers to decide whether to stay or leave the company. The longer they are attached to the company, the lower the risk of customer churn to take place.

Phone Service

- Below shows the phone service distribution in customer attrition. Most of the customers in telecom have phone service. However, it seems no difference for the customer churn and customer non-churn. Thus, it can be concluded that there are no large impacts for phone service on customer churn.

Multiple Lines

- Among those customers with phone service, 44.7% of churn customers and 40.7% of non churn customers have multiple lines.

Internet Service

- The most popular internet service provider used by churn customers is fibre optic (69.7%) and then followed by dsl (24.4%) whereas 5.9% does not have internet service. This is interesting because even though fibre optic internet services are faster, customer churn are also most likely to happen because of it.

Online Security

- Most of the customers do not have online security. Thus, it does not strongly affect the customer churn.

Online Backup

- The ratio of customers with online backup is almost similar between churn customers and non churn customers. For both categories, the percentage of customers without online backup are higher compared to customers with online backup.

Online Backup

- The ratio of customers with online backup is almost similar between churn customers and non churn customers. For both categories, the percentage of customers without online backup are higher compared to customers with online backup.

Tech Support

- In both charts, customers without technical support are more than customers with technical support. Thus, it is less likely for this data to affect the customer churn.

Streaming TV

- Similarly, customers with or without a streaming tv is less likely to affect the customer churn. This is because the ratio of customers with streaming is more or less the same between churn and non churn customers.

Streaming movies

- 43.6% of churn customers stream movies while 56.4% of churn customers do not stream movies. Thus, there is no strong relationship between streaming movies and customer churn.

Contract

- From the pie chart, we can see that most of the churn customers are having month to month contracts followed by one year contracts. Churn customers with two years contracts are the least among all. Thus, we can conclude that when the contract period is short, the chances of customer churn to occur will be higher.

Paperless Billing

- The use of paperless billing in churn customers is slightly higher compared to non churn customers.

Payment Method

- Based on the payment method distribution, electronic check payment occupied 56.7% of the churn customer. Thus, it can be said that electronic check payment will have a huge impact in customer attrition.

Monthly Charges

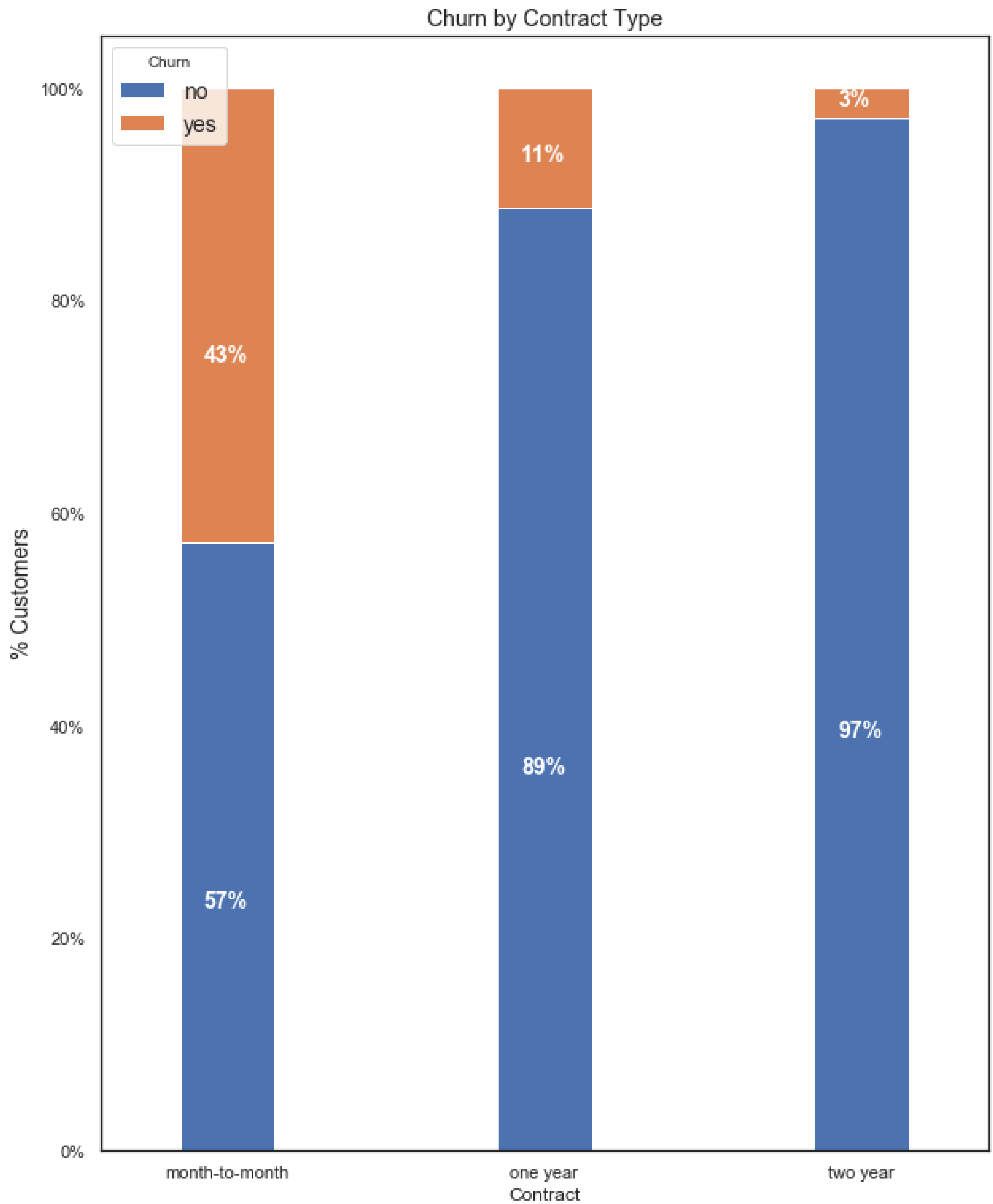
- From the histogram, the monthly charges for most of the non churn customers in Telecom is around 20, which is the lowest charge. This could potentially mean that the lower the charges is, the easier it is to retain the customers.

Total Charges

- Based on the histogram below, it can be seen that the churn is higher when the total charges are lower.

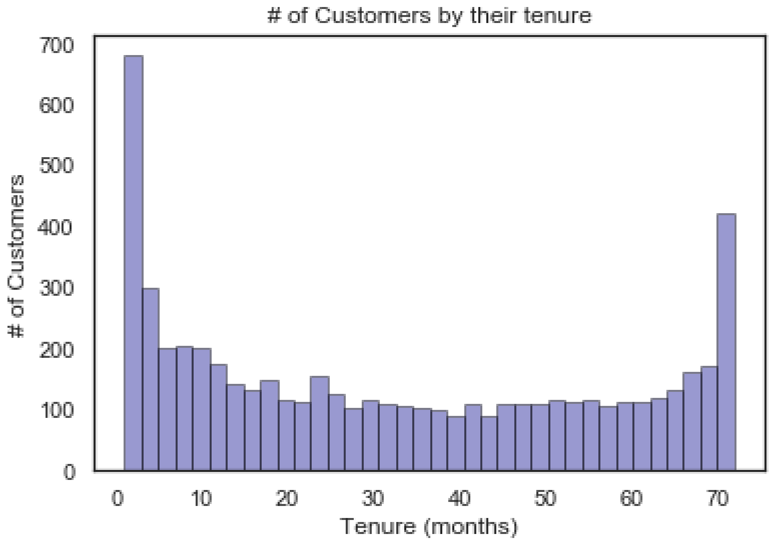
In [15]:

```
# Additional EDA for contract type with churn rate
contract_churn = train_data.groupby(['Contract','Churn']).size().unstack()

ax = (contract_churn.T*100.0 / contract_churn.T.sum()).T.plot(kind='bar',
                                                                width = 0.3,
                                                                stacked = True,
                                                                rot = 0,
                                                                figsize = (10,13
                                                                )

ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='best',prop={'size':14},title = 'Churn')
ax.set_ylabel('% Customers',size = 14)
ax.set_title('Churn by Contract Type',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.4*heig
                color = 'white',
               weight = 'bold',
               size = 14)
```



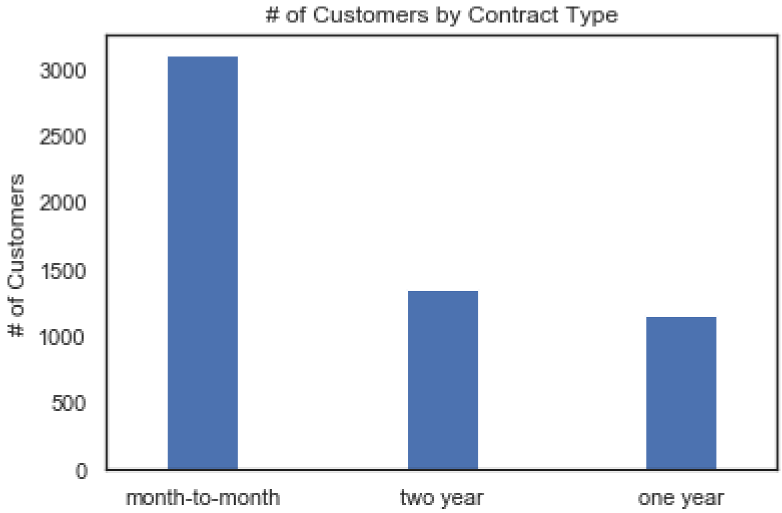
In[16]:
```
# number of customers by their tenure
ax = sns.distplot(train_data['tenure'], hist=True,kde=False,
     bins=int(180/5), color = 'darkblue',
     hist_kws={'edgecolor':'black'},
     kde_kws={'linewidth': 4})
ax.set_ylabel('# of Customers')
ax.set_xlabel('Tenure(months)')
ax.set_title('# of Customers by their tenure')
```

Out[16]: Text(0.5, 1.0, '# of Customers by their tenure')



In[17]:
```
# Cutomers by contract type
ax = train_data['Contract'].value_counts().plot(kind = 'bar',rot = 0, width = 0.3
ax.set_ylabel('# of Customers')
ax.set_title('# of Customers by Contract Type')
```

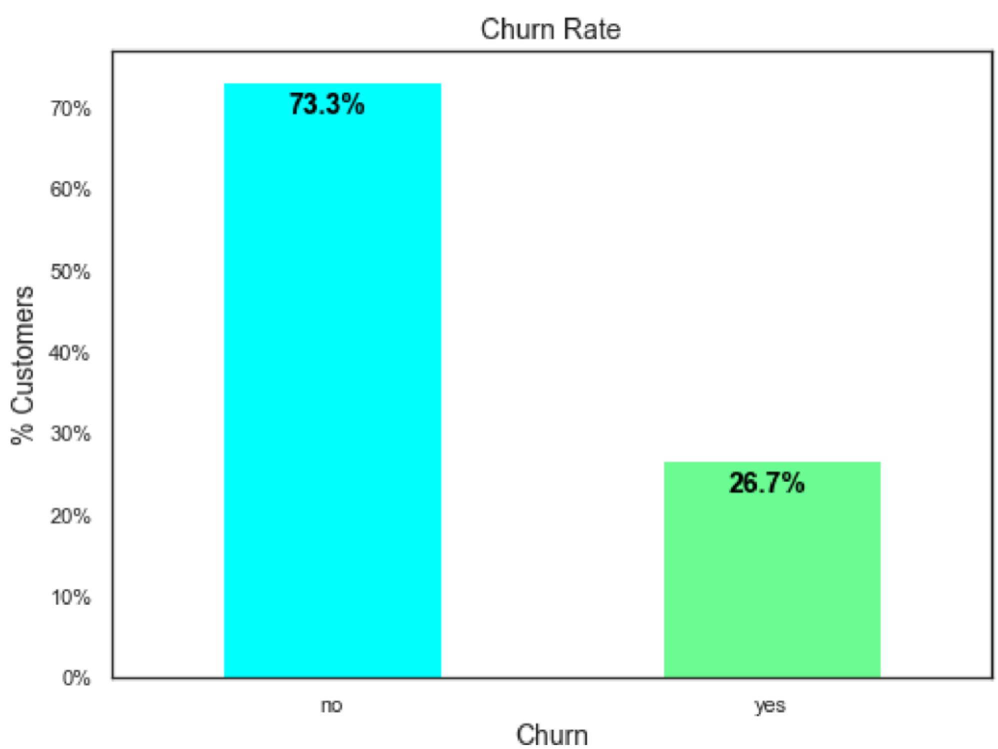
Out[17]: Text(0.5, 1.0, '# of Customers by Contract Type')



Balance the number of churners and nonchurners

In [18]:

```
# Showing original ratio for both churn and non-churn customer
colors = ['#00FFFF','#6AFB92']
ax = (train_data['Churn'].value_counts()*100.0 /len(train_data)).plot(kind='bar',
                                                          rot = 0, color = colors,
                                                         figsize = (8,6))
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('% Customers',size = 14)
ax.set_xlabel('Churn',size = 14)
ax.set_title('Churn Rate', size = 14)

# Create a list to collect the plt.patches data
totals = []

# Find the values and append to list
for i in ax.patches:
    totals.append(i.get_width())

# Set individual bar lables using above list
total = sum(totals)

for i in ax.patches:
    # get_width pulls left or right; get_y pushes up or down
    ax.text(i.get_x()+.15, i.get_height()-4.0, \
           str(round((i.get_height()/total), 1))+'%',
           fontsize=12,
           color='black',
           weight = 'bold',
           size = 14)
```



In [19]:
```
# Balance the number of the churners and non-churners
churners_number = len(train_data[train_data['Churn'] == 'yes'])
print("Number of churners: ", churners_number)

churners = (train_data[train_data['Churn'] == 'yes'])

non_churners = train_data[train_data['Churn'] == 'no'].sample(n=churners_number)
print("Number of non-churners: ", len(non_churners))
balance_train_data = churners.append(non_churners)
```

```
Number of churners:  1500
Number of non-churners:  1500
```

In [20]:
```
# Split X_train set, X_test set and y_train set,y_test set
X_train = balance_train_data.drop('Churn',axis=1)
X_test = test_data.drop('Churn',axis=1)
y_train = balance_train_data['Churn']
y_test = test_data['Churn']
X_test.head()
```

Out[20]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetServ
5561	male	no	no	no	1	yes	no	
5814	male	no	no	no	16	yes	no	
2645	female	no	no	no	1	yes	no	
3983	male	no	no	no	1	no	no	
6438	male	yes	no	no	1	yes	yes	fiber o

Data Preparation

After doing the EDA, we think that tenure, contract, InternetService, PaymentMethod and MonthlyCharges from our datasets are important in predicting the churn rate. This is because these five features are having a stronger association with the churn rate. Hence, we plan to construct one modelling with all features in our dataset excluding the customer ID and another one with only the five key features to determine which of them is more suitable to predict our customer churn.

Data Preprocessing

In [21]:

```python
# Categorical columns
encode_cat_cols   = X_train.nunique()[X_train.nunique() < 6].keys().tolist()

# Label encoding categorical columns
le = LabelEncoder()
for i in encode_cat_cols :
    X_train[i] = le.fit_transform(X_train[i])
    X_test[i] = le.transform(X_test[i])

# standard scaling the numerical columns
scaler = StandardScaler()

col_names_1 = ['MonthlyCharges']
features_train_1 = X_train[col_names_1]
features_test_1 = X_test[col_names_1]

X_train['MonthlyCharges'] = scaler.fit_transform(features_train_1)
X_test['MonthlyCharges'] = scaler.transform(features_test_1)

col_names_2 = ['tenure']
features_train_2 = X_train[col_names_2]
features_test_2 = X_test[col_names_2]

X_train['tenure'] = scaler.fit_transform(features_train_2)
X_test['tenure'] = scaler.transform(features_test_2)

col_names_3 = ['TotalCharges']
features_train_3 = X_train[col_names_3]
features_test_3 = X_test[col_names_3]

X_train['TotalCharges'] = scaler.fit_transform(features_train_3)
X_test['TotalCharges'] = scaler.transform(features_test_3)

# Label Encoding y_train and y_test
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)

X_train
```

Out[21]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetS
6498	0	0	0	0	-1.104475	1	0	
4062	0	0	0	0	-1.104475	1	0	
5901	1	0	1	1	0.898250	1	1	
381	1	0	0	0	-1.104475	1	0	
5779	0	0	0	0	0.481016	1	0	
...	...	...	...	...	...	...	...	
1537	0	0	1	1	1.857890	1	1	
3069	0	0	0	0	-0.728964	1	0	

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetS
200	0	0	1	0	-0.019666	1	0	
3021	1	0	0	0	-0.937582	1	0	
2555	0	0	0	0	-0.812411	1	0	

3000 rows × 19 columns

Modelling

Using All features

KNN

In [22]:
```
#KNN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

y_predict = knn.predict(X_test)
print(accuracy_score(y_test,y_predict))
confusion_matrix(y_test, y_predict)
```

```
0.7107320540156361
```

Out[22]:
```
array([[709, 329],
       [ 78, 291]], dtype=int64)
```

Logistic Regression

In [23]:
```
# Logistic Regression
lr = LogisticRegression()
lr.fit(X_train, y_train)

y_predict = lr.predict(X_test)
print(accuracy_score(y_predict, y_test))

confusion_matrix(y_test, y_predict)
```

```
0.736318407960199
```

Out[23]:
```
array([[739, 299],
       [ 72, 297]], dtype=int64)
```

Support Vector Machine (SVM)

In [24]:
```
#SVM
svc = SVC(probability=True)
svc.fit(X_train, y_train)

y_predict = svc.predict(X_test)
print(accuracy_score(y_test, y_predict))
confusion_matrix(y_test, y_predict)
```

```
0.7391613361762616
```

Out[24]:
```
array([[747, 291],
       [ 76, 293]], dtype=int64)
```

Random Forest Classifier

In [25]:
```
# Random Forest Classifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_predict = rf.predict(X_test)

print (accuracy_score(y_test, y_predict))
```

```
0.7427149964463398
```

Gaussian Naive Bayes

In [26]:
```
# Gaussian Naive Bayes
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_predict = gnb.predict(X_test)

# Evaluation
print(accuracy_score(y_test, y_predict))
```

```
0.7164179104477612
```

Modelling

Using Five Features

In [27]:
```
X_train_used = X_train[['Contract', 'tenure', 'InternetService', 'PaymentMethod',
X_test_used = X_test[['Contract', 'tenure', 'InternetService', 'PaymentMethod', '
```

KNN

In [28]:

```python
#KNN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_used, y_train)

y_predict = knn.predict(X_test_used)
print(accuracy_score(y_test,y_predict))
confusion_matrix(y_test, y_predict)
```

```
0.7171286425017769
```

Out[28]:

```
array([[738, 300],
       [ 98, 271]], dtype=int64)
```

Determine best parameters for KNN

In [29]:

```python
k_range = list(range(1,31))
weight_options = ["uniform", "distance"]

param_grid = dict(n_neighbors = k_range, weights = weight_options)
#print (param_grid)


grid_search = GridSearchCV(knn, param_grid, cv = 10, scoring = 'accuracy')
grid_search.fit(X_train_used,y_train)
best_params = grid_search.best_params_
print(best_params)
```

```
{'n_neighbors': 25, 'weights': 'uniform'}
```

Tuning parameters for KNN

In [30]:

```python
knn = KNeighborsClassifier(n_neighbors=best_params["n_neighbors"],
                           weights=best_params["weights"])
knn.fit(X_train_used, y_train)

y_predict = knn.predict(X_test_used)

# Evaluation
print(accuracy_score(y_test,y_predict))
confusion_matrix(y_test, y_predict)
```

```
0.7377398720682303
```

Out[30]:

```
array([[748, 290],
       [ 79, 290]], dtype=int64)
```

Logistic Regression

In [31]:
```
# Logistic Regression
lr = LogisticRegression()
lr.fit(X_train_used, y_train)

y_predict = lr.predict(X_test_used)
print(accuracy_score(y_predict, y_test))

confusion_matrix(y_test, y_predict)
```

```
0.7178393745557925
```

Out[31]:
```
array([[726, 312],
       [ 85, 284]], dtype=int64)
```

Determine best parameters for Logistic Regression

In [32]:
```
parameters = {'C': np.logspace(0, 4, 10), 'penalty':['l1','l2'] }

grid_search = GridSearchCV(estimator = lr,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10,
                           n_jobs = -1)
grid_search = grid_search.fit(X_train_used, y_train)
best_params = grid_search.best_params_
print(best_params)
```

```
{'C': 1.0, 'penalty': 'l2'}
```

Tuning parameters for Logistic Regression

In [33]:
```
lr = LogisticRegression(C=best_params["C"],
                        penalty=best_params["penalty"])
lr.fit(X_train_used, y_train)

y_predict = lr.predict(X_test_used)

# Evaluation
print(accuracy_score(y_predict, y_test))
confusion_matrix(y_test, y_predict)
```

```
0.7178393745557925
```

Out[33]:
```
array([[726, 312],
       [ 85, 284]], dtype=int64)
```

Support Vector Machine (SVM)

In [34]:
```
#SVM
svc = SVC(probability=True)
svc.fit(X_train_used, y_train)

y_predict = svc.predict(X_test_used)
print(accuracy_score(y_test, y_predict))
confusion_matrix(y_test, y_predict)
```

```
0.7569296375266524
```

Out[34]:
```
array([[771, 267],
       [ 75, 294]], dtype=int64)
```

Determine best parameters for SVM

In [35]:
```
parameters = [{'C': [1, 10, 100, 1000], 'kernel': ['linear','rbf'],
               'gamma': [0.1, 0.2, 0.3, 0.4, 0.5]}]

grid_search = GridSearchCV(estimator = svc,
param_grid = parameters,
scoring = 'accuracy',
cv = 5,                                   n_jobs = -1)

grid_search = grid_search.fit(X_train_used, y_train)
best_params = grid_search.best_params_
print(best_params)
```

```
{'C': 1, 'gamma': 0.5, 'kernel': 'rbf'}
```

Tuning parameters for SVM

In [36]:
```
# SVM
svc = SVC(C=best_params["C"],
          gamma=best_params["gamma"],
          kernel=best_params["kernel"],
          probability = True)
svc.fit(X_train_used, y_train)
y_predict = svc.predict(X_test_used)

# Evaluation
print(accuracy_score(y_test, y_predict))
confusion_matrix(y_test, y_predict)
```

```
0.7476901208244492
```

Out[36]:
```
array([[761, 277],
       [ 78, 291]], dtype=int64)
```

Random Forest Classifier

In [37]:
```
# Before tuning parameters
rf = RandomForestClassifier()
rf.fit(X_train_used, y_train)
y_predict = rf.predict(X_test_used)

print (accuracy_score(y_test, y_predict))
```

```
0.7327647476901208
```

Determine best parameters for Random Forest

In [38]:
```
# Create the parameter grid based on the results of random search
param_grid = {
    'max_depth': [80, 90, 100, 110],
    'max_features': ['auto', 'log2'],
    'max_leaf_nodes': [10, 20, 30],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]

}
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)

# Fit the grid search to the data
grid_search = grid_search.fit(X_train_used, y_train)
best_params = grid_search.best_params_
print(best_params)
```

```
Fitting 3 folds for each of 864 candidates, totalling 2592 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:    7.5s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:   36.6s
[Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 640 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done 1005 tasks      | elapsed:  4.2min
[Parallel(n_jobs=-1)]: Done 1450 tasks      | elapsed:  6.1min
[Parallel(n_jobs=-1)]: Done 1977 tasks      | elapsed:  8.4min
[Parallel(n_jobs=-1)]: Done 2584 tasks      | elapsed: 10.9min
[Parallel(n_jobs=-1)]: Done 2592 out of 2592 | elapsed: 11.0min finished

{'max_depth': 100, 'max_features': 'log2', 'max_leaf_nodes': 20, 'min_samples_l
eaf': 5, 'min_samples_split': 12, 'n_estimators': 100}
```

Tuning parameters for Random Forest

In [39]:
```
# After tuning parameter
rf = RandomForestClassifier(max_depth=best_params["max_depth"],
                            max_features=best_params["max_features"],
                            max_leaf_nodes=best_params["max_leaf_nodes"],
                            min_samples_leaf=best_params["min_samples_leaf"],
                            min_samples_split=best_params["min_samples_split”],
                            n_estimators=best_params["n_estimators"])

rf.fit(X_train_used, y_train) y_predict =

rf.predict(X_test_used)

print (accuracy_score(y_test, y_predict))


# Making the Confusion Matrix cm =
confusion_matrix(y_test, y_predict)

# Calculate the Accuracy
print(accuracy_score(y_test, y_predict))
```

```
0.7533759772565742
0.7533759772565742
```

Gaussian Naive Bayes

In [40]:
```
# Gaussian Naive Bayes
gnb = GaussianNB()
gnb.fit(X_train_used, y_train)
y_predict = gnb.predict(X_test_used)

# Evaluation
print(accuracy_score(y_test, y_predict))
```

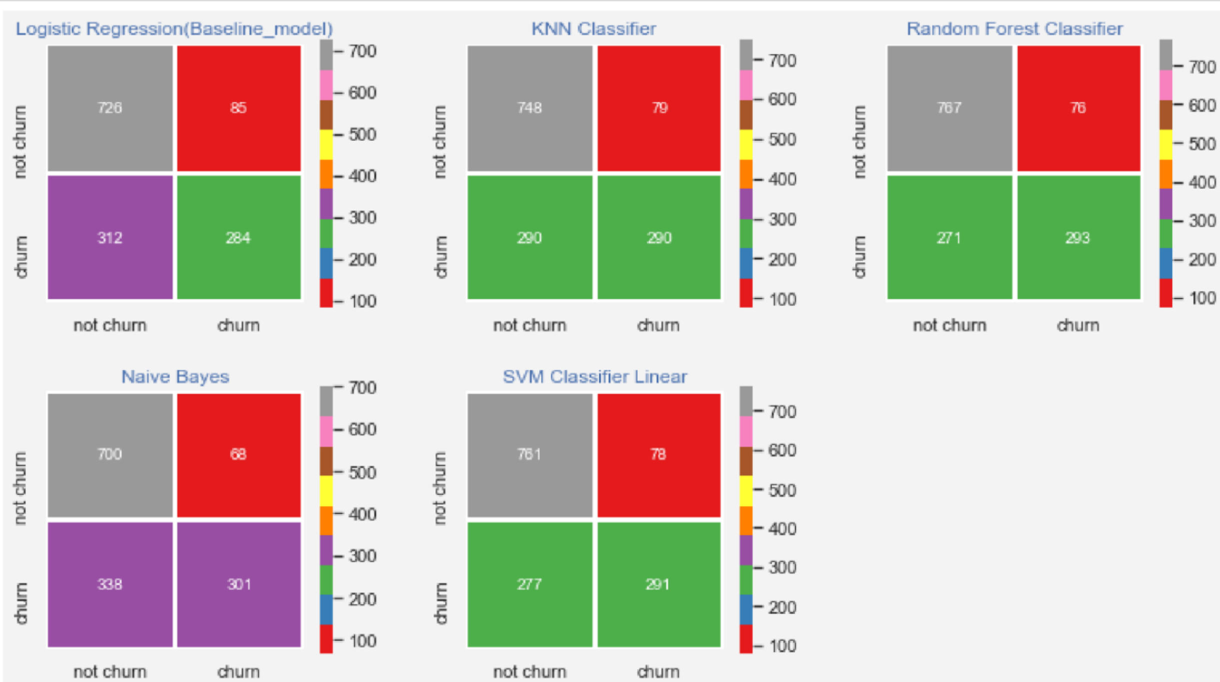
```
0.7114427860696517
```

After the modelling processes, we found out that the accuracy score of the model that uses only five key features is slightly lower than using all features. Since we can obtain an almost similar result from both of the modelling, we decided to use only the five features to predict our customer churn as more features will cause a longer training time.

Evaluation

Confusion Matrix

In [41]:
```
def telecom_churn_prediction_alg(X_train_used,X_test_used,y_train,y_test):

  model.fit(X_train_used,y_train)
  predictions  = model.predict(X_test_used)
  accuracy     = accuracy_score(y_test,predictions)

#confusion matrix
#conf_matrix = confusion_matrix(y_test,predictions)

lst = [lr,knn,rf,gnb,svc]

length = len(lst)

mods   = ['Logistic Regression(Baseline_model)','KNN Classifier',
          'Random Forest Classifier',"Naive Bayes",'SVM Classifier Linear']

fig = plt.figure(figsize=(13,15))
fig.set_facecolor("#F3F3F3")
for i,j,k in itertools.zip_longest(lst,range(length),mods) :
    plt.subplot(4,3,j+1)
    predictions = i.predict(X_test_used)
    conf_matrix = confusion_matrix(predictions,y_test)
    sns.heatmap(conf_matrix,annot=True,fmt = "d",square = True,
                xticklabels=["not churn","churn"],
                yticklabels=["not churn","churn"],
                linewidths = 2,linecolor = "w",cmap = "Set1")
    plt.title(k,color = "b")
    plt.subplots_adjust(wspace = .3,hspace = .3)
```



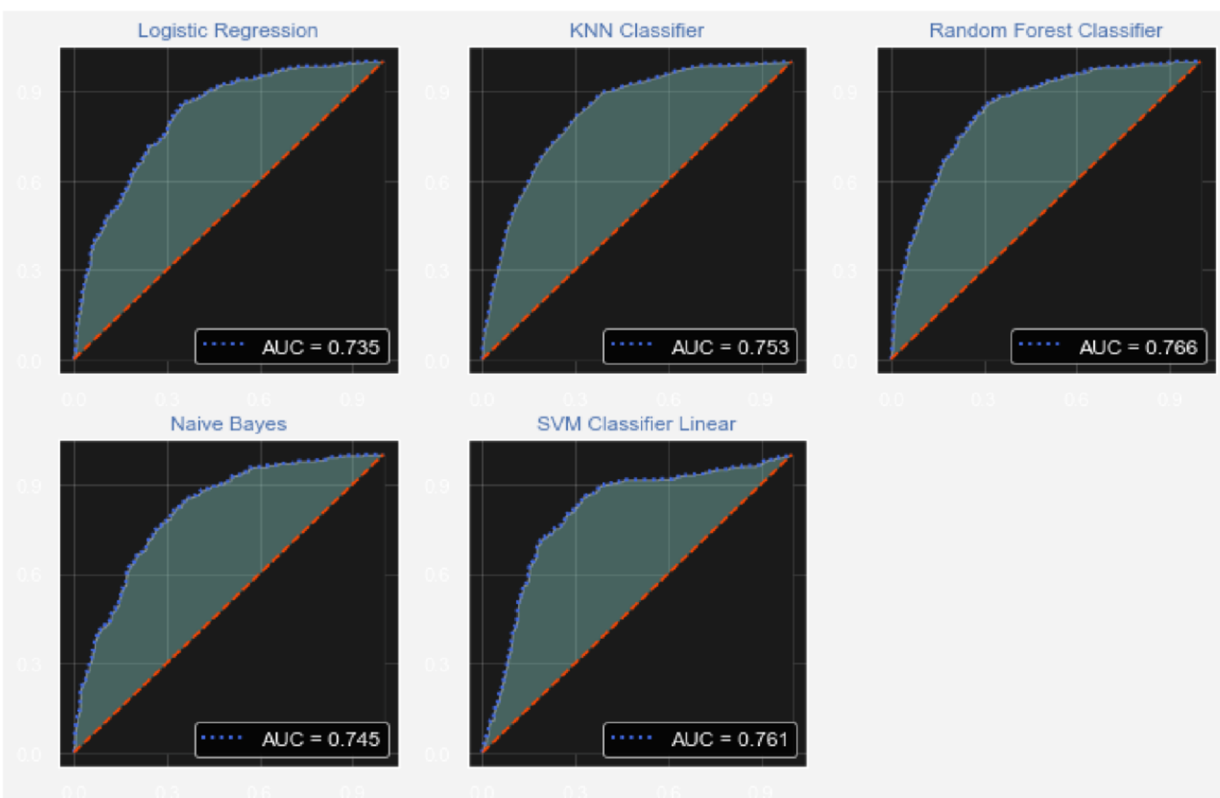
In the business world, we think that we should avoid the Type II error which is the False Negative

(FP). In our case, false negative means that we predicted not churn, but it actually is a churn. We should avoid this kind of error. This is because if we assume a particular customer will not churn, then we will not take any action to retain that customer. However, they eventually change their service provider, this will lead to a higher churn rate as proper action is not taken on time.

The result above shows the confusion matrix of the five modelling algorithms. Based on the confusion matrix, we can see that the Random Forest algorithm has the lowest FN and Naive Bayes has the highest FN.

Receiver Operating Characteristic (ROC)

In [42]:
```python
lst = [lr,knn,rf,gnb,svc]

length = len(lst)

mods   = ['Logistic Regression','KNN Classifier',
          'Random Forest Classifier',"Naive Bayes",'SVM Classifier Linear']

plt.style.use("dark_background")
fig = plt.figure(figsize=(12,16))
fig.set_facecolor("#F3F3F3")
for i,j,k in itertools.zip_longest(lst,range(length),mods) :
    qx = plt.subplot(4,3,j+1)
    predictions    = i.predict(X_test_used)
    probabilities = i.predict_proba(X_test_used)
    fpr,tpr,thresholds = roc_curve(y_test,probabilities[:,1])
    plt.plot(fpr,tpr,linestyle = "dotted",
             color = "royalblue",linewidth = 2,
             label = "AUC = " + str(np.around(roc_auc_score(y_test,predictions),
    plt.plot([0,1],[0,1],linestyle = "dashed",
             color = "orangered",linewidth = 1.5)
    plt.fill_between(fpr,tpr,alpha = .4)
    plt.fill_between([0,1],[0,1],color = "k")
    plt.legend(loc = "lower right",
               prop = {"size" : 12})
    qx.set_facecolor("k")
    plt.grid(True,alpha = .15)
    plt.title(k,color = "b")
    plt.xticks(np.arange(0,1,.3))
    plt.yticks(np.arange(0,1,.3))
```



When AUC is between 0.5 and 1, there is a higher possibility that the model will be able to differentiate the churn and not churn customers. This is because the model can detect a higher number of true positive and true negative rather than false negative and false positive. By referring to the five ROC curves above, we can clearly see that the AUC of the five models have reached a value between 0.5 and 1. However, Random Forest model has the highest Area Under Curve (AUC).

Modal Comparison

In [43]:
```
#gives model report in dataframe
def model_report(model,x_train_used,X_test_used,y_train,y_test,name):
   model.fit(x_train_used,y_train)
   predictions  = model.predict(X_test_used)
   accuracy     = accuracy_score(y_test,predictions)
   recallscore  = recall_score(y_test,predictions)
   precision    = precision_score(y_test,predictions)
   roc_auc      = roc_auc_score(y_test,predictions)
   f1score      = f1_score(y_test,predictions)

    df = pd.DataFrame({"Model"            : [name],
                       "Accuracy_score"   : [accuracy],
                      "Recall_score"     : [recallscore],
                      "Precision"        : [precision],
                      "f1_score"         : [f1score],
                      "Area_under_curve": [roc_auc],
                     })
    return df

#outputs for every model
model1 = model_report(lr,X_train_used,X_test_used,y_train,y_test, "Logistic Regre

model2 = model_report(knn,X_train_used,X_test_used,y_train,y_test,"KNN Classifier
rfc = RandomForestClassifier(n_estimators = 1000,
                             random_state = 123,
                             max_depth = 9,
                             criterion = "gini")
model3 = model_report(rf,X_train_used,X_test_used,y_train,y_test,"Random Forest C
model4 = model_report(gnb,X_train_used,X_test_used,y_train,y_test,"Naive Bayes")
model5 = model_report(svc,X_train_used,X_test_used,y_train,y_test,"SVM Classifier
#concat all models
model_performances = pd.concat([model1,model2,model3,
                                model4,model5],axis =  0).reset_index()
model_performances = model_performances.drop(columns = "index",axis =1)
table  = ff.create_table(np.round(model_performances,4))
py.iplot(table)
```

Model	Accuracy_score	Recall_score	Precision	f1_score	Area_under_curve
Logistic Regression	0.7178	0.7696	0.4765	0.5886	0.7345
KNN Classifier	0.7377	0.7859	0.5	0.6112	0.7533
Random Forest Classifier	0.752	0.8022	0.5175	0.6291	0.7681
Naive Bayes	0.7114	0.8157	0.471	0.5972	0.745
SVM Classifier	0.7477	0.7886	0.5123	0.6211	0.7609

The table above shows the accuracy score, recall score, precision, F1 score and Area under the curve of all the five modelling techniques. From the table, we can conclude that Random Forest model has the highest accuracy score.

Deployment

Among the five modelling techniques that we have constructed, we decided to choose the Random Forest because it has the highest values for the accuracy_score, precision, f1_score and area_under_curve.

Choosing an appropriate modelling is very important to predict customer churn rate. This will help us to know more about whether the customer will be churn or not in the future by fitting them into our modelling. By knowing which of the customer will be churn after predicting using a suitable modelling technique, telecom companies are able to take immediate actions such as customizing them different promotional activities to gain back their retention and prevent them from churning.

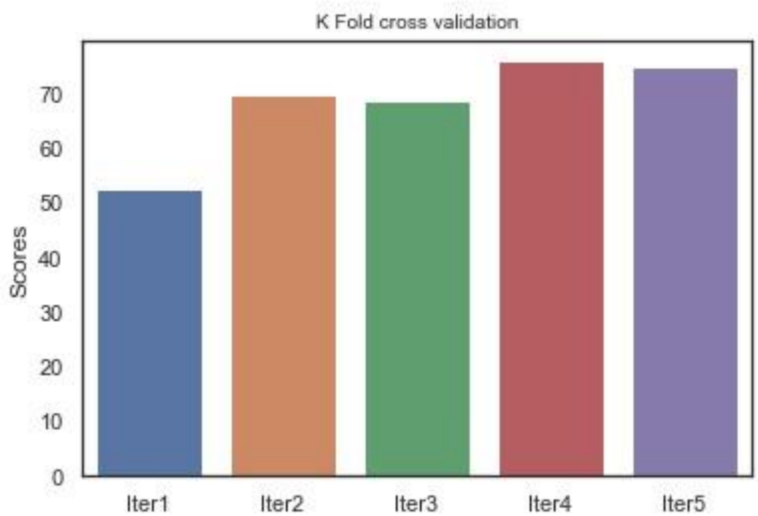
K fold cross validaiton

In [44]:

```
X = X_train_used.append(X_test_used)
y = np.concatenate((y_train,y_test))

# KFold Cross Validation approach
kf = KFold(n_splits=5,shuffle=False)
kf.split(X)

# Initialize the accuracy of the models to blank list. The accuracy of each model
accuracy_model = []

# Iterate over each train-test split
for train_index, test_index in kf.split(X):
    # Split train-test
    X_train_used, X_test_used = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]
    # Train the model
    model = rf.fit(X_train_used, y_train)
    # Append to accuracy_model the accuracy of the model
    accuracy_model.append(accuracy_score(y_test, model.predict(X_test_used), norm

# Print the accuracy
 print(accuracy_model)
```

```
[52.721088435374156, 69.84126984126983, 68.89897843359817, 76.04994324631102, 7
5.14188422247446]
```

In [45]:
```
# Visualize accuracy for each iteration

scores = pd.DataFrame(accuracy_model,columns=['Scores'])

sns.set(style="white", rc={"lines.linewidth": 3})
sns.barplot(x=['Iter1','Iter2','Iter3','Iter4','Iter5'],y="Scores",data=scores)
plt.title('K Fold cross validation', fontsize=10)

plt.show()
sns.set()
```



We use the K-fold cross validation to check whether the selected model has the overfit problem or not. We need to make sure that the model will provide us a consistent result when predicting different sets of data. When the model provides us an accurate result during our modelling process but provides a bad result when using another set of data to do the modelling, this means that this model is imperfect because overfitting may occur as the model only fits a certain dataset.

From the bar chart above, we can clearly see that the accuracy score of the five iterations are in the acceptable range. So, we can conclude that Random Forest model do not have overfit problem.

Conclusion

We have selected the Random Forest Classifier model to predict our customer churn rate. Random Forests is a powerful algorithm in Machine Learning. Primarily, Random Forest will take the input data from the initial dataset and some features variables will be randomly chosen to grow the tree. So, in our model, there are 5 key variables that have a strong correlation with churn rate that are used to grow the tree. Every tree in the forest should not be cut off unless it reaches the end of the tree and predicts its churn and not churn. In this way, Random Forest can create strong classifiers to our churn rate and make predictions accurately. It can be used to solve classification and regression problems. Basically, our assignment is counted as a classification problem as the model needs to classify the churn rate to “Yes” or “No”. Most of our columns are categorical variables and only two columns which are “Monthly Charges” and “Total Charges”.

And, we can see that Random Forest works well with both categorical and continuous variables which can be found in our datasets, and hence we have selected Random Forest as our model.

The Random Forest modelling technique has the ability to handle big data with numerous variables that might go up to thousands which is greatly needed for our data. It can automatically balance the data sets when a class is more infrequent than other classes in the data. The method also handles variables fast, making it suitable for complicated tasks. Apart from that, the Random Forest is also considered one of the most stable algorithms among all models because it will not seriously affect the overall algorithm even if a new data point is introduced in the dataset. The newly introduced data will mostly only impact one tree as it is very hard for it to impact all the trees. On the other hand, there were some limitations when using the Random Forest model. One of the limitations is time consuming. When determining the best parameters for the Random Forest using GridSearchCV, it took quite a long time for the tuning to complete, an approximately 15-30 minutes is required in our case. This is because Random Forest consists of many parameters that we can tune, hence there will be a large number of different parameter lists which are required to be tried out by the GridSearchCV.

Another limitation is that our accuracy for the Random Forest is only around 0.7 and each time when we run the model, there will be a slight difference in the result. So we were unable to make sure that our churn prediction for every customer is 100% accurate. This means that sometimes it may predict the wrong customer churn. In conclusion, Random Forest is a fast, simple and a flexible model to use but not without some limitations.

Reference

https://www.sv-europe.com/crisp-dm-methodology/ (https://www.sv-europe.com/crisp-dmmethodology/) https://www.profitwell.com/blog/churn-prediction (https://www.profitwell.com/blog/churn-prediction)

https://baremetrics.com/academy/churn-prediction-can-improve-business (https://baremetrics.com/academy/churn-prediction-can-improve-business)

https://www.appier.com/blog/churnprediction/#:~:text=Customer%20churn%20prediction%20can%20help,it%20comes%20to%20custom (https://www.appier.com/blog/churnprediction/#:~:text=Customer%20churn%20prediction%20can%20help,it%20comes%20to%20custom

https://pdfs.semanticscholar.org/e6d1/b56a913c45acbec34534cef4c1dfbc8d4663.pdf (https://pdfs.semanticscholar.org/e6d1/b56a913c45acbec34534cef4c1dfbc8d4663.pdf)

https://analyticsindiamag.com/5-ways-handle-missing-values-machine-learning-datasets/ (https://analyticsindiamag.com/5-ways-handle-missing-values-machine-learning-datasets/)

https://www.kaggle.com/blastchar/telco-customer-churn (https://www.kaggle.com/blastchar/telcocustomer-churn)

https://www.kaggle.com/farazrahman/telco-customer-churn-logisticregression (https://www.kaggle.com/farazrahman/telco-customer-churn-logisticregression)

https://www.kaggle.com/pavanraj159/telecom-customer-churn-prediction (https://www.kaggle.com/pavanraj159/telecom-customer-churn-prediction)

https://www.kaggle.com/devarshraval/churn-prediction-neural-network-and-ml-models (https://www.kaggle.com/devarshraval/churn-prediction-neural-network-and-ml-models)

https://financetrain.com/k-fold-cross-validation-example-python-scikit-learn/ (https://financetrain.com/k-fold-cross-validation-example-python-scikit-learn/)