Churn problem

Import necessary libraries

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

In [2]:
```
import pandas as pd
```

In [3]:
```
import seaborn as sns
```

In [4]:
```
import matplotlib.ticker as mtick
```

In [5]:
```
import matplotlib.pyplot as plt
```

In [6]:
```
sns.set(style = 'white')
```

In [7]:
```
import os
```

In [8]:
```
from sklearn.preprocessing import StandardScaler
```

In [9]:
```
from sklearn.linear_model import LogisticRegression
```

In [10]:
```
from sklearn import metrics
```

In [11]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [12]:
```
from sklearn.svm import SVC
```

In [13]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [14]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [15]:
```
from xgboost import XGBClassifier
```

In [16]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [17]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
```

In [18]:
```
from sklearn.preprocessing import KBinsDiscretizer
from imblearn.pipeline import make_pipeline
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.ensemble import (RandomForestClassifier, AdaBoostClassifier,
 GradientBoostingClassifier, ExtraTreesClassifier)
```

In [19]:
```
from sklearn.metrics import accuracy_score, roc_curve, f1_score, precision_score, recall_score, confusion_matrix
from sklearn.metrics import roc_auc_score
```

In [22]:
```
#Load the dataset
telecom_cust = pd.read_csv("C:\\Users\\30698\\Downloads\\churn-train.csv")
```

In [23]:
```
telecom_cust.head()
```

Out[23]:

vice	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
DSL	Yes	No	Yes	No	No	No	Month-to-month	No	'Bank transfer (automatic)'	33.60	2117.2	No
ptic'	No	Yes	Yes	Yes	No	No	'Two year'	No	'Bank transfer (automatic)'	90.45	6565.85	No
ptic'	No	No	No	No	Yes	No	Month-to-month	Yes	'Electronic check'	84.00	424.75	No
DSL	Yes	Yes	Yes	Yes	No	No	'Two year'	No	'Bank transfer (automatic)'	67.40	3306.85	No
No	'No internet service'	'No internet service'	'No internet service'	'No internet service'	'No internet service'	'No internet service'	Month-to-month	Yes	'Bank transfer (automatic)'	19.70	168.9	No

Data Preprocessing

In [24]:
```
telecom_cust.columns.values
```

Out[24]:
```
array(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
       'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',
       'TotalCharges', 'Churn'], dtype=object)
```

In [25]:
```
telecom_cust.dtypes
```

Out[25]:
```
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 [27]:
```
#convert Total charges to numeric variable
telecom_cust.TotalCharges = pd.to_numeric(telecom_cust.TotalCharges, errors='coerce')
print(telecom_cust.isnull().values.any())
telecom_cust.isnull().sum()
#there are 6 missing values in the dataset
```

```
True
```

Out[27]:
```
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        6
Churn               0
dtype: int64
```

In [29]:
```python
#for test set
test = pd.read_csv("C:\\Users\\30698\\Downloads\\churn-test.csv")
test.TotalCharges = pd.to_numeric(test.TotalCharges, errors='coerce')
print(test.isnull().values.any())
test.isnull().sum()
```

```
True
```

Out[29]:
```
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        5
Churn               0
dtype: int64
```

In [30]:
```python
#removing missing values
telecom_cust.dropna(inplace = True)
test.dropna(inplace = True)
```

In [31]:
```python
print(telecom_cust.isnull().values.any())
```

```
False
```

In [32]:
```python
df=telecom_cust
```

In [33]:
```python
df['Churn'].replace(to_replace='Yes', value=1, inplace=True)
df['Churn'].replace(to_replace='No',  value=0, inplace=True)
```

In [34]:
```python
test['Churn'].replace(to_replace='Yes', value=1, inplace=True)
test['Churn'].replace(to_replace='No',  value=0, inplace=True)
```

In [37]:
```python
#convert categorical variables to dummies
df_dummies = pd.get_dummies(df)
df_dummies.head()
```

Out[37]:

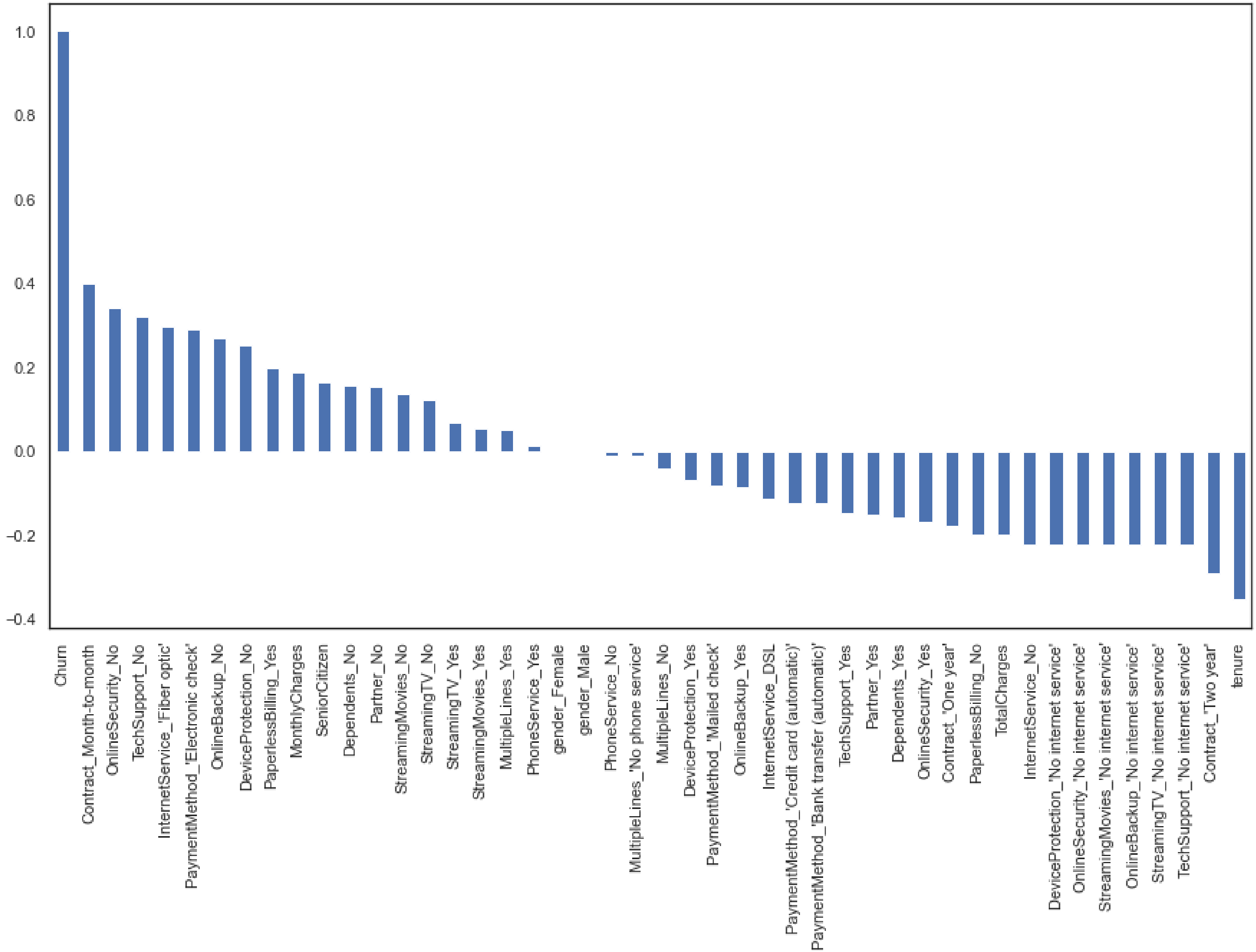
	SeniorCitizen	tenure	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	...	StreamingMovies_Yes	Contract_'One year'	Cc
0	0	61	33.60	2117.20	0	0	1	0	1	0	...	0	0	
1	0	72	90.45	6565.85	0	0	1	0	1	0	...	0	0	
2	0	5	84.00	424.75	0	1	0	1	0	1	...	0	0	
3	0	49	67.40	3306.85	0	1	0	1	0	1	...	0	0	
4	0	8	19.70	168.90	0	0	1	1	0	1	...	0	0	

5 rows × 46 columns

In [38]:
```python
test_dummies = pd.get_dummies(test)
```

In [39]:
```
plt.figure(figsize=(15,8))
df_dummies.corr()['Churn'].sort_values(ascending = False).plot(kind='bar')
```

Out[39]: <AxesSubplot:>



We observe a positive correlation of Churn with month-to-month contract,no tech support,no online security and fiber optic internet service. This means that most churn customers are those who have contract only for a month (this is logical as after the end of their contract is most possible to leave their "product" than others who have longer contracts), and those who dont have services like tech support and online security and also those who use fiber optic internet servise because probably it is more expensive. We also observe that churn customers are negatively correlated with tenure and two year contract This means that a customer who uses the "product" for a long time or has a two year contract is less possible to become a churn customer

In [35]:
```
#Data Analysis
```

In [41]:
```
df['gender'].value_counts()*100.0 /len(df)
#Data are balanced regarding the gender(about half male,half female)
```

Out[41]:
```
Male      50.245255
Female    49.754745
Name: gender, dtype: float64
```

In [42]:
```
df['SeniorCitizen'].value_counts()*100.0 /len(telecom_cust)
#Most of the customers in the dataset are younger people.
```

Out[42]:
```
0    83.663894
1    16.336106
Name: SeniorCitizen, dtype: float64
```

In [43]:
```
df['Dependents'].value_counts()*100.0 /len(telecom_cust)
#only 30% of customers have dependents
```

Out[43]:
```
No     70.804009
Yes    29.195991
Name: Dependents, dtype: float64
```

In [44]:
```
df['Partner'].value_counts()*100.0 /len(telecom_cust)
#half of customers have partner
```

Out[44]:
```
No     52.121988
Yes    47.878012
Name: Partner, dtype: float64
```

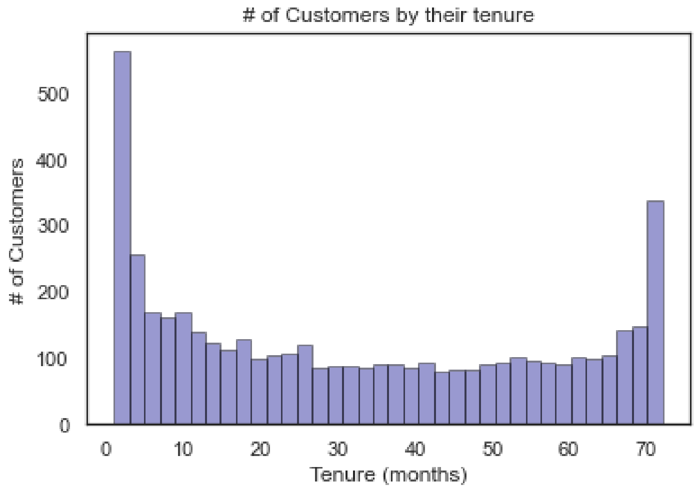
In [45]:
```
df['Churn'].value_counts()*100.0 /len(telecom_cust)
#The dataset is imbalanced.Only 26% are churn customers.
```

Out[45]:
```
0    73.299211
1    26.700789
Name: Churn, dtype: float64
```

In [46]:
```
import warnings
warnings.filterwarnings('ignore')
```

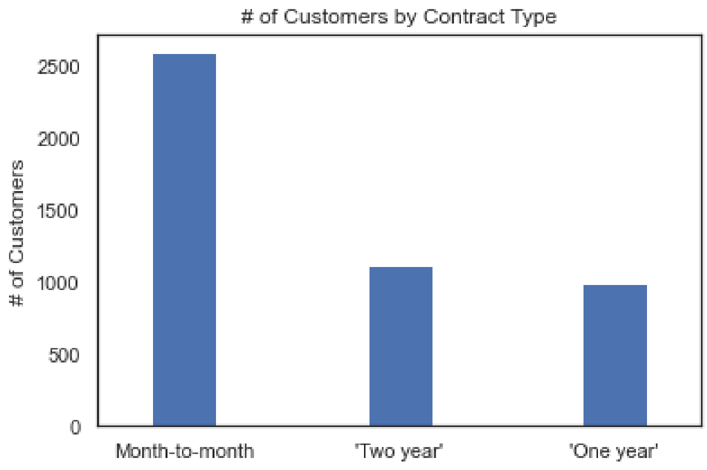
In [47]:
```
ax = sns.distplot(df['tenure'], hist=True, kde=False,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
ax.set_ylabel('# of Customers')
ax.set_xlabel('Tenure (months)')
ax.set_title('# of Customers by their tenure')
#A lot of customers stayed only one month with the company and many stayed for 72 months.
```

Out[47]:
```
Text(0.5, 1.0, '# of Customers by their tenure')
```



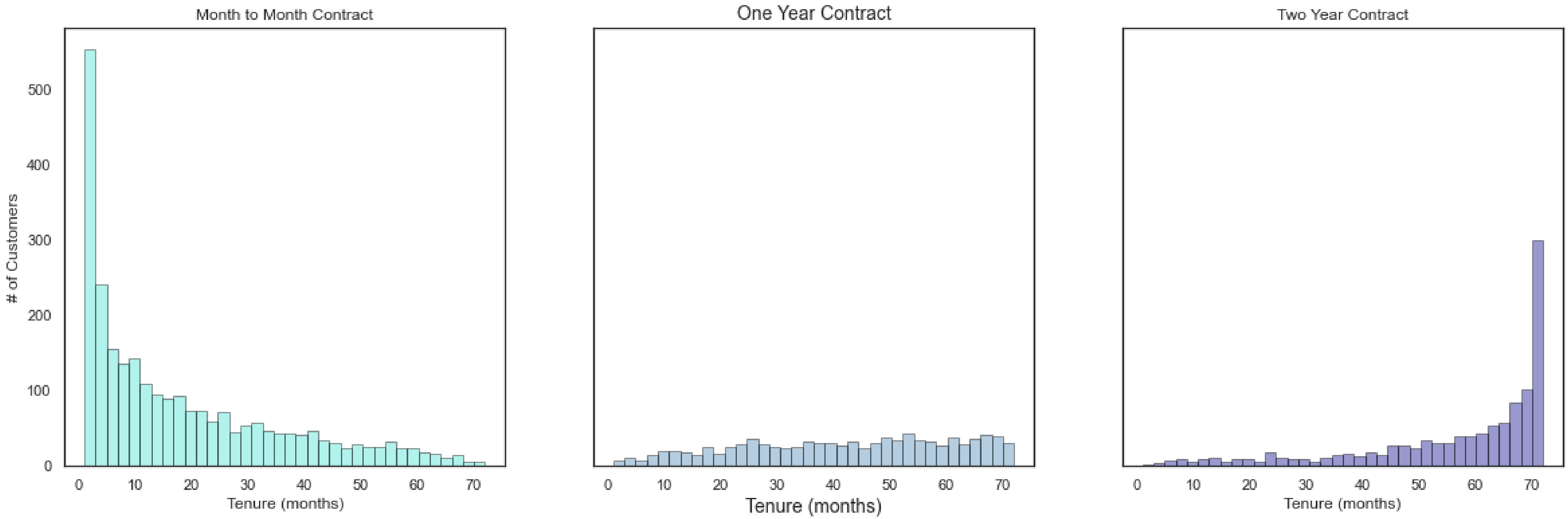
In [48]:
```python
ax = df['Contract'].value_counts().plot(kind = 'bar',rot = 0, width = 0.3)
ax.set_ylabel('# of Customers')
ax.set_title('# of Customers by Contract Type')
#Most of the customers are in the one month contract
```

Out[48]:
```
Text(0.5, 1.0, '# of Customers by Contract Type')
```



In [49]:
```
#The tenure of customers based on their contract type
fig, (ax1,ax2,ax3) = plt.subplots(nrows=1, ncols=3, sharey = True, figsize = (20,6))

ax = sns.distplot(df[df['Contract']=='Month-to-month']['tenure'],
                   hist=True, kde=False,
                   bins=int(180/5), color = 'turquoise',
                   hist_kws={'edgecolor':'black'},
                   kde_kws={'linewidth': 4},
                 ax=ax1)
ax.set_ylabel('# of Customers')
ax.set_xlabel('Tenure (months)')
ax.set_title('Month to Month Contract')

ax = sns.distplot(df[df['Contract']=="'One year'"]['tenure'],
                   hist=True, kde=False,
                   bins=int(180/5), color = 'steelblue',
                   hist_kws={'edgecolor':'black'},
                   kde_kws={'linewidth': 4},
                 ax=ax2)
ax.set_xlabel('Tenure (months)',size = 14)
ax.set_title('One Year Contract',size = 14)

ax = sns.distplot(df[df['Contract']=="'Two year'"]['tenure'],
                   hist=True, kde=False,
                   bins=int(180/5), color = 'darkblue',
                   hist_kws={'edgecolor':'black'},
                   kde_kws={'linewidth': 4},
                 ax=ax3)

ax.set_xlabel('Tenure (months)')
ax.set_title('Two Year Contract')
#We observe that most of the customers with the one month contract stay just for one month in the telecom company.On the other
#hand most of those with the two year contract stay for 72 months in the company.
#So,customers taking a longer contract are more loyal to the company and tend to stay with it for a longer period of time.
```

Out[49]:
```
Text(0.5, 1.0, 'Two Year Contract')
```

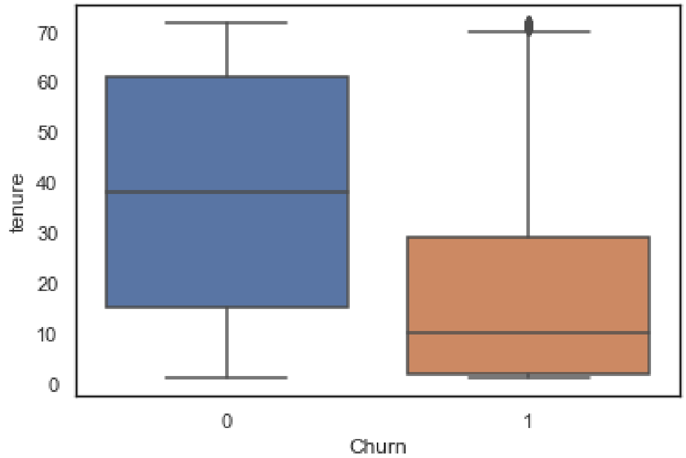



```
In [50]: df['Contract'].unique()
```

```
Out[50]: array(['Month-to-month', "'Two year'", "'One year'"], dtype=object)
```

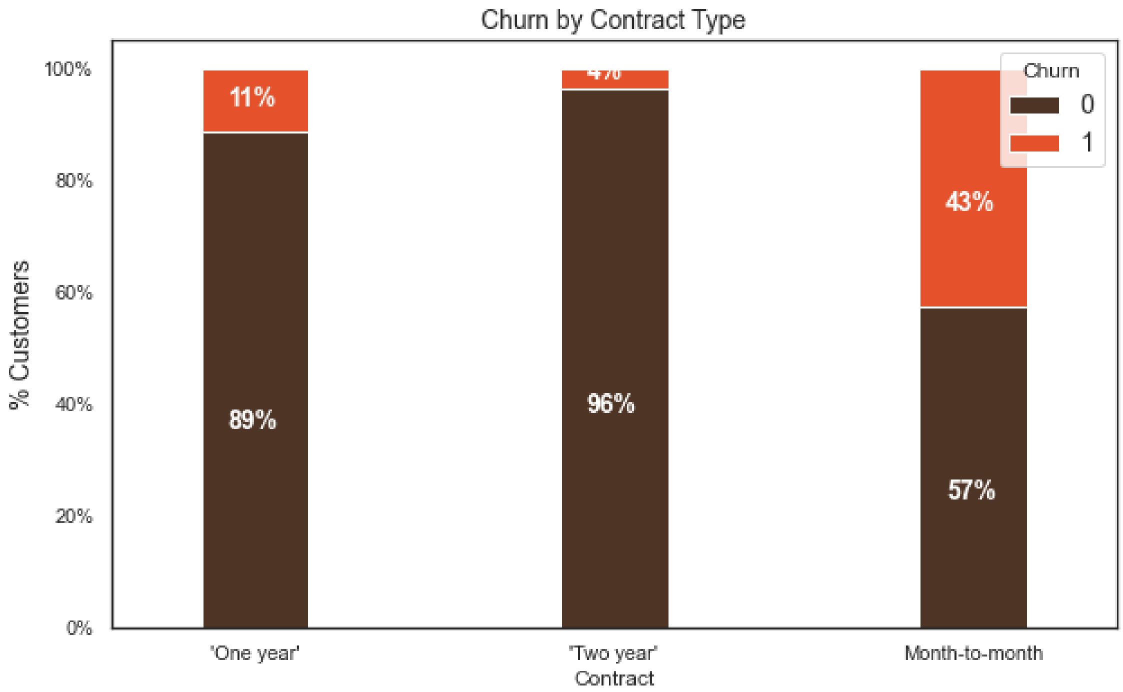
```
In [51]: #Churn vs Tenure
         sns.boxplot(x = df.Churn, y = df.tenure)
         #Loyal customers tend to stay for a longer tenure with the company.
```

```
Out[51]: <AxesSubplot:xlabel='Churn', ylabel='tenure'>
```



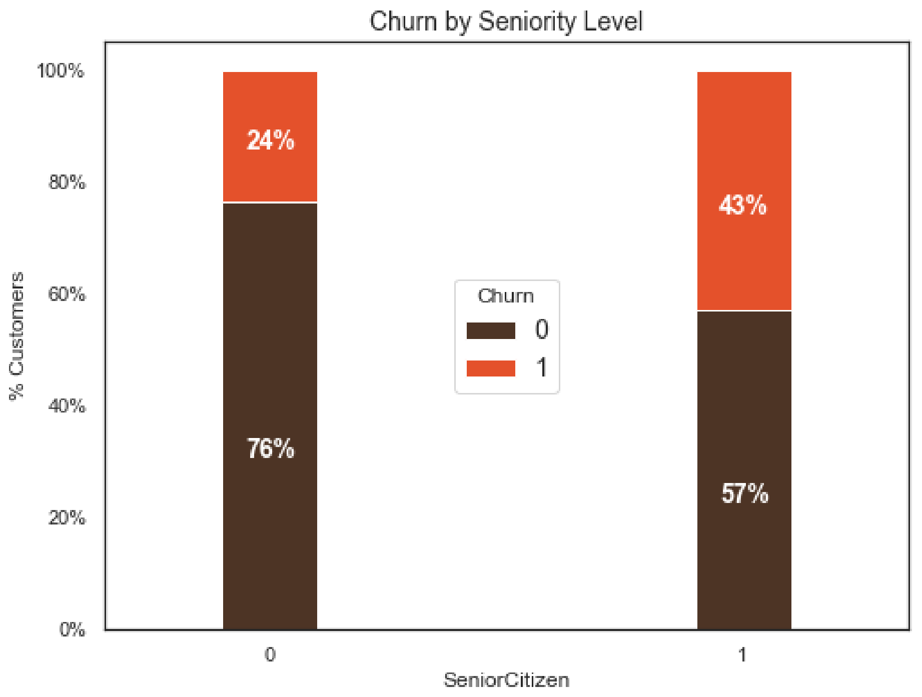
In [52]:

```python
colors = ['#4D3425','#E4512B']
contract_churn = df.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 = (10,6),
                                                                color = colors)
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*height),
                color = 'white',
               weight = 'bold',
               size = 14)
#Customers who have the one month contract have a very high churn rate.
```

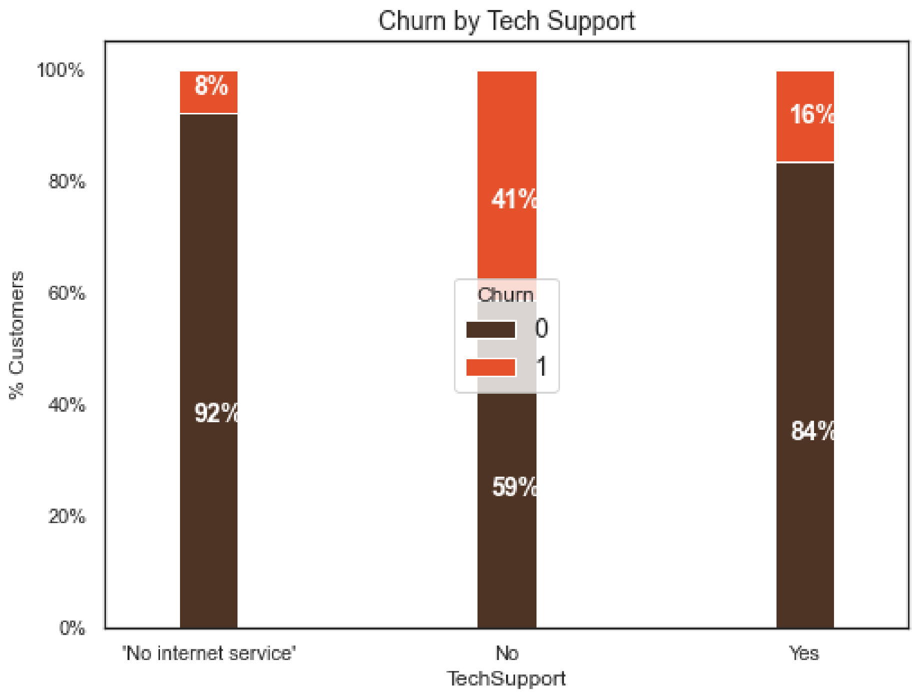


In [53]:

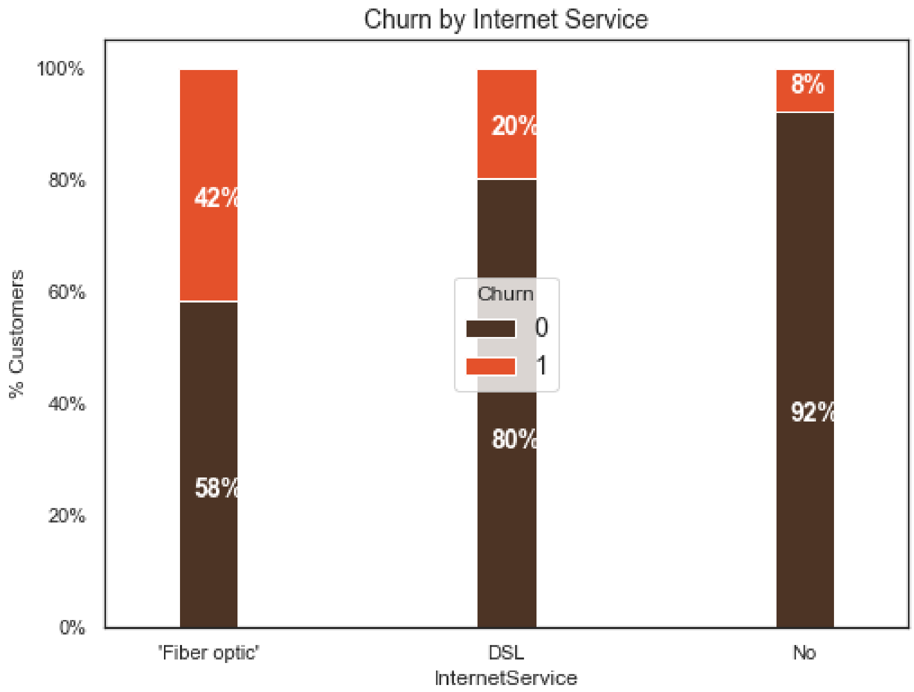
```
colors = ['#4D3425','#E4512B']
seniority_churn = df.groupby(['SeniorCitizen','Churn']).size().unstack()

ax = (seniority_churn.T*100.0 / seniority_churn.T.sum()).T.plot(kind='bar',
                                                                width = 0.2,
                                                                stacked = True,
                                                                rot = 0,
                                                                figsize = (8,6),
                                                                color = colors)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='center',prop={'size':14},title = 'Churn')
ax.set_ylabel('% Customers')
ax.set_title('Churn by Seniority Level',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*height),
                color = 'white',
               weight = 'bold',size =14)
#Senior Citizens have higher churn rate than younger customers.(15%more)
```



In [54]:
```
colors = ['#4D3425','#E4512B']
seniority_churn = df.groupby(['TechSupport','Churn']).size().unstack()

ax = (seniority_churn.T*100.0 / seniority_churn.T.sum()).T.plot(kind='bar',
                                                                width = 0.2,
                                                                stacked = True,
                                                                rot = 0,
                                                                figsize = (8,6),
                                                                color = colors)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='center',prop={'size':14},title = 'Churn')
ax.set_ylabel('% Customers')
ax.set_title('Churn by Tech Support',size = 14)

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*height),
                color = 'white',
               weight = 'bold',size =14)
#The abcense of Tech support increases significantly the churn rate.
```



In [55]:
```
colors = ['#4D3425','#E4512B']
seniority_churn = df.groupby(['InternetService','Churn']).size().unstack()

ax = (seniority_churn.T*100.0 / seniority_churn.T.sum()).T.plot(kind='bar',
                                                                width = 0.2,
                                                                stacked = True,
                                                                rot = 0,
                                                                figsize = (8,6),
                                                                color = colors)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='center',prop={'size':14},title = 'Churn')
ax.set_ylabel('% Customers')
ax.set_title('Churn by Internet Service',size = 14)

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*height),
                color = 'white',
               weight = 'bold',size =14)
#The use of the fiber optic service increases significantly the churn rate.
```



In [56]:
```
df_dummies.head()
```

Out[56]:

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	...	StreamingMovies_Yes	Contract_'One year'	Co
0	0	61	33.60	2117.20	0	0	1	0	1	0	...	0	0	
1	0	72	90.45	6565.85	0	0	1	0	1	0	...	0	0	
2	0	5	84.00	424.75	0	1	0	1	0	1	...	0	0	
3	0	49	67.40	3306.85	0	1	0	1	0	1	...	0	0	
4	0	8	19.70	168.90	0	0	1	1	0	1	...	0	0	

5 rows × 46 columns

In [57]:
```
#There is high range in the values so we need to scale them in order to avoid the high values to prevail in our model.
scal=StandardScaler()
```

In [58]:
```
col=['tenure','MonthlyCharges','TotalCharges']
```

In [59]:
```
df_dummies[col]=scal.fit_transform(df_dummies[col])
```

In [60]:
```
test_dummies[col]=scal.fit_transform(test_dummies[col])
```

In [63]:
```
y=df_dummies['Churn']
X=df_dummies.drop('Churn',axis=1)
```

In [65]:
```
#sampling.
#We will use the oversampling technique because the dataset is class imbalanced.
from imblearn.over_sampling import SMOTE
smote = SMOTE()
x_smote, y_smote = smote.fit_sample(X,y)
```

In [68]:
```python
ada = AdaBoostClassifier()
ada.fit(X,y)
scores = cross_val_score(ada, X, y, cv=10, scoring='accuracy')
#We could change the scoring to recall but we will take it separately later
print(scores)
print(scores.mean())
pred=cross_val_predict(ada,X,y, cv=10)
print(confusion_matrix(y,pred))
print(classification_report(y, pred))
#Using Recall for imbalanced data
recall_score(y,pred)
```

```
[0.82089552 0.79317697 0.79744136 0.79530917 0.79530917 0.79317697
 0.79957356 0.78464819 0.81449893 0.80128205]
0.7995311902028319
[[3087  350]
 [ 590  662]]
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      3437
           1       0.65      0.53      0.58      1252

    accuracy                           0.80      4689
   macro avg       0.75      0.71      0.73      4689
weighted avg       0.79      0.80      0.79      4689
```

Out[68]:
```
0.5287539936102237
```

In [135]:
```
#The performance of adaboost classifier using the data after oversampling increases drammatically the recall score.
ada2 = AdaBoostClassifier()
ada2.fit(x_smote, y_smote)
scores = cross_val_score(ada2, x_smote, y_smote, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(ada2,x_smote, y_smote, cv=10)
print(confusion_matrix(y_smote,pred))
print(classification_report(y_smote, pred))
recall_score(y_smote,pred)
```

```
[0.7747093  0.7630814  0.73837209 0.76744186 0.81659389 0.82387191
 0.84425036 0.82678311 0.84861718 0.83697234]
0.8040693443011409
[[2655  782]
 [ 565 2872]]
              precision    recall  f1-score   support

           0       0.82      0.77      0.80      3437
           1       0.79      0.84      0.81      3437

    accuracy                           0.80      6874
   macro avg       0.81      0.80      0.80      6874
weighted avg       0.81      0.80      0.80      6874
```

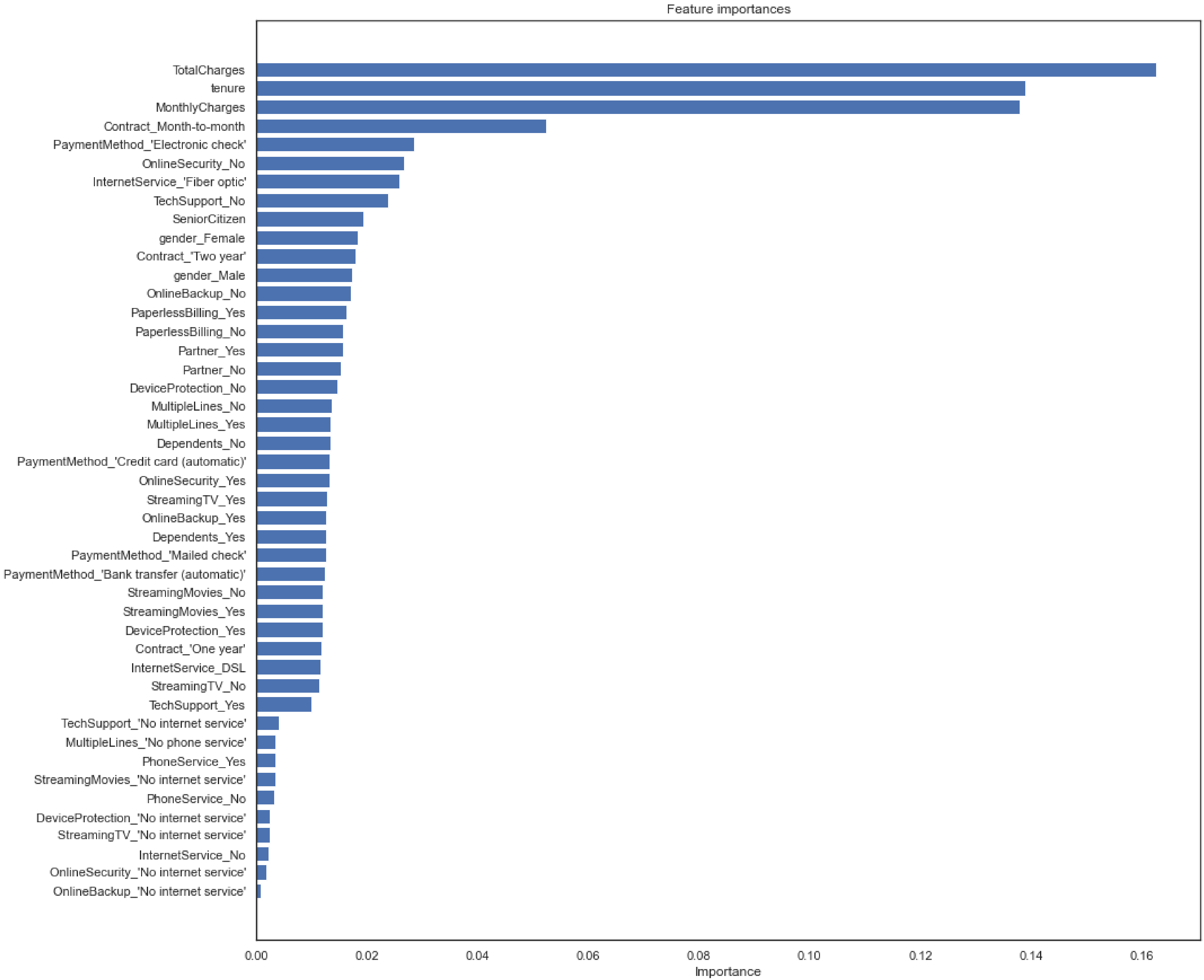
Out[135]: 0.8356124527203957

Feature selection

In [69]:

```
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X,y)
rf_features=rf.feature_importances_
sorted_idx = np.argsort(rf_features)
print(sorted_idx )
plt.figure(figsize=(15, 15))
plt.barh(range(len(sorted_idx)), rf_features[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), X.columns.values[sorted_idx])
plt.xlabel('Importance')
plt.title('Feature importances')
plt.draw()
plt.show()
#Here we can see the most important and less important features.Random Forest does feature selection
#so we dont have to do cross validation during the training of this model.
#As we can see the most important features to predict a churn customer are:Total charges,tenure,monthly charges,
#month to month contract,absence of tech support,fiber optic internet service.
#we can drop the least important features:'PhoneService_No', 'PhoneService_Yes', 'MultipleLines_'No phone service'',
#'InternetService_No', 'OnlineSecurity_'No internet service'','OnlineBackup_'No internet service'',
#'DeviceProtection_'No internet service''  'TechSupport_'No internet service'',  'StreamingTV_'No internet service'',
#'StreamingMovies_'No internet service''
```

```
[21 18 17 30 24 10 33 11 12 27 29 31 16 36 26 35 34 41 44  9 23 32 20 42
  8 14 13 25  6  7 39 40 22  5 37  4  0 28 15 19 43 38  2  1  3]
```


Feature importances
TotalCharges
tenure
MonthlyCharges
Contract_Month-to-month
PaymentMethod_'Electronic check'
OnlineSecurity_No
InternetService_'Fiber optic'
TechSupport_No
SeniorCitizen
gender_Female
Contract_'Two year'
gender_Male
OnlineBackup_No
PaperlessBilling_Yes
PaperlessBilling_No
Partner_Yes
Partner_No
DeviceProtection_No
MultipleLines_No
MultipleLines_Yes
Dependents_No
PaymentMethod_'Credit card (automatic)'
OnlineSecurity_Yes
StreamingTV_Yes
OnlineBackup_Yes
Dependents_Yes
PaymentMethod_'Mailed check'
PaymentMethod_'Bank transfer (automatic)'
StreamingMovies_No
StreamingMovies_Yes
DeviceProtection_Yes
Contract_'One year'
InternetService_DSL
StreamingTV_No
TechSupport_Yes
TechSupport_'No internet service'
MultipleLines_'No phone service'
PhoneService_Yes
StreamingMovies_'No internet service'
PhoneService_No
DeviceProtection_'No internet service'
StreamingTV_'No internet service'
InternetService_No
OnlineSecurity_'No internet service'
OnlineBackup_'No internet service'
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
Importance

In [70]:
```
fs_dataset=df_dummies.drop(columns=['PhoneService_No','PhoneService_Yes', "MultipleLines_'No phone service'",'InternetService_No', "OnlineSecurity_
"StreamingMovies_'No internet service'"])
y_f=fs_dataset['Churn']
X_f=fs_dataset.drop('Churn',axis=1)
```

In [131]:
```
from sklearn.feature_selection import RFE
rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=15)
rfe.fit(X,y)
rfe.fit(X_f,y_f)
```

Out[131]:
```
RFE
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

In [82]:
```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
clf = ExtraTreesClassifier(n_estimators=15)
clf = clf.fit(X, y)
model = SelectFromModel(clf, prefit=True)
```

In [87]:
```
pip=make_pipeline(model,smote,ada)
pip.fit(X,y)
y_predict = pip.predict(X)
accuracy_score(pip.predict(X), y)
print(roc_auc_score(pip.predict(X),y))
print(confusion_matrix(pip.predict(X), y))
print(classification_report(pip.predict(X), y))
recall_score(y,y_predict)
```

```
0.7197301795082549
[[2524  239]
 [ 913 1013]]
              precision    recall  f1-score   support

           0       0.73      0.91      0.81      2763
           1       0.81      0.53      0.64      1926

    accuracy                           0.75      4689
   macro avg       0.77      0.72      0.73      4689
weighted avg       0.77      0.75      0.74      4689
```

Out[87]:
```
0.8091054313099042
```

In [80]:
```
pip=make_pipeline(rfe,smote,ada)
pip.fit(X,y)
y_predict = pip.predict(X)
accuracy_score(pip.predict(X), y)
print(roc_auc_score(pip.predict(X),y))
print(confusion_matrix(pip.predict(X), y))
print(classification_report(pip.predict(X), y))
recall_score(y,y_predict)
```

```
0.7261116364326875
[[2641  283]
 [ 796  969]]
              precision    recall  f1-score   support

           0       0.77      0.90      0.83      2924
           1       0.77      0.55      0.64      1765

    accuracy                           0.77      4689
   macro avg       0.77      0.73      0.74      4689
weighted avg       0.77      0.77      0.76      4689
```

Out[80]:
```
0.7739616613418531
```

In [81]:
```
gr = GradientBoostingClassifier()
gr.fit(X,y)
scores = cross_val_score(gr, X,y, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(gr,X,y, cv=10)
print(confusion_matrix(y,pred))
print(classification_report(y, pred))
recall_score(y,pred)
```

```
[0.79957356 0.79957356 0.7782516  0.78464819 0.81663113 0.78678038
 0.80383795 0.78464819 0.80383795 0.81623932]
0.7974021832230788
[[3103  334]
 [ 615  637]]
              precision    recall  f1-score   support

           0       0.83      0.90      0.87      3437
           1       0.66      0.51      0.57      1252

    accuracy                           0.80      4689
   macro avg       0.75      0.71      0.72      4689
weighted avg       0.79      0.80      0.79      4689
```

Out[81]:
```
0.5087859424920128
```

In [76]:
```
#after sampling
gr = GradientBoostingClassifier()
gr.fit(x_smote,y_smote)
scores = cross_val_score(gr, x_smote,y_smote, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(gr,x_smote,y_smote, cv=10)
print(confusion_matrix(y_smote,pred))
print(classification_report(y_smote, pred))
recall_score(y_smote,pred)
```

```
[0.78052326 0.77180233 0.74709302 0.77034884 0.8588064  0.86899563
 0.88646288 0.86899563 0.88646288 0.86608443]
0.8305575302122474
[[2775  662]
 [ 503 2934]]
              precision    recall  f1-score   support

           0       0.85      0.81      0.83      3437
           1       0.82      0.85      0.83      3437

    accuracy                           0.83      6874
   macro avg       0.83      0.83      0.83      6874
weighted avg       0.83      0.83      0.83      6874
```

Out[76]:
```
0.853651440209485
```

In [99]:
```
pip=make_pipeline(rfe,smote,gr)
pip.fit(X,y)
y_predict = pip.predict(X)
accuracy_score(pip.predict(X), y)
print(roc_auc_score(pip.predict(X),y))
print(confusion_matrix(pip.predict(X), y))
print(classification_report(pip.predict(X), y))
recall_score(y,y_predict)
```

```
0.7354042156547894
[[2689  280]
 [ 748  972]]
              precision    recall  f1-score   support

           0       0.78      0.91      0.84      2969
           1       0.78      0.57      0.65      1720

    accuracy                           0.78      4689
   macro avg       0.78      0.74      0.75      4689
weighted avg       0.78      0.78      0.77      4689
```

Out[99]:
```
0.7763578274760383
```

In [88]:
```
pip=make_pipeline(model,smote,gr)
pip.fit(X,y)
y_predict = pip.predict(X)
accuracy_score(pip.predict(X), y)
print(roc_auc_score(pip.predict(X),y))
print(confusion_matrix(pip.predict(X), y))
print(classification_report(pip.predict(X), y))
recall_score(y,y_predict)
```

```
0.7386242291455964
[[2663  252]
 [ 774 1000]]
              precision    recall  f1-score   support

           0       0.77      0.91      0.84      2915
           1       0.80      0.56      0.66      1774

    accuracy                           0.78      4689
   macro avg       0.79      0.74      0.75      4689
weighted avg       0.78      0.78      0.77      4689
```

Out[88]:
```
0.7987220447284346
```

In [125]:
```
rf=RandomForestClassifier()
rf.fit(X,y)
scores = cross_val_score(rf, X,y, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(rf,X,y, cv=10)
print(confusion_matrix(y,pred))
print(classification_report(y, pred))
recall_score(y,pred)
```

```
[0.7771855  0.76226013 0.7761194  0.80063966 0.80576307]
0.7843935528941662
[[3082  355]
 [ 652  600]]
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      3437
           1       0.63      0.48      0.54      1252

    accuracy                           0.79      4689
   macro avg       0.73      0.69      0.70      4689
weighted avg       0.77      0.79      0.78      4689
```

Out[125]:
```
0.4792332268370607
```

In [126]:
```
rf=RandomForestClassifier()
rf.fit(x_smote,y_smote)
scores = cross_val_score(rf, x_smote,y_smote, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(rf,x_smote,y_smote, cv=10)
print(confusion_matrix(y_smote,pred))
print(classification_report(y_smote, pred))
recall_score(y_smote,pred)
```

```
[0.78690909 0.784      0.88872727 0.90472727 0.91048035]
0.8549687971417228
[[2892  545]
 [ 420 3017]]
              precision    recall  f1-score   support

           0       0.87      0.84      0.86      3437
           1       0.85      0.88      0.86      3437

    accuracy                           0.86      6874
   macro avg       0.86      0.86      0.86      6874
weighted avg       0.86      0.86      0.86      6874
```

Out[126]:
```
0.8778004073319755
```

In [93]:
```
clf = LogisticRegression(class_weight="balanced")
clf.fit(X,y)
scores = cross_val_score(clf, X,y, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(clf,X,y, cv=10)
print(confusion_matrix(y,pred))
print(classification_report(y, pred))
#we also need to use the right metric for imbalanced data(recall)
recall_score(y,pred)
```

```
[0.75692964 0.74840085 0.74626866 0.71002132 0.72921109 0.73987207
 0.76119403 0.75053305 0.76972281 0.74358974]
0.7455743261713411
[[2504  933]
 [ 260  992]]
              precision    recall  f1-score   support

           0       0.91      0.73      0.81      3437
           1       0.52      0.79      0.62      1252

    accuracy                           0.75      4689
   macro avg       0.71      0.76      0.72      4689
weighted avg       0.80      0.75      0.76      4689
```

Out[93]:
```
0.792332268370607
```

In [94]:
```
pip=make_pipeline(rfe,smote,clf)
pip.fit(X,y)
y_predict = pip.predict(X)
accuracy_score(pip.predict(X), y)
print(roc_auc_score(pip.predict(X),y))
print(confusion_matrix(pip.predict(X), y))
print(classification_report(pip.predict(X), y))
print(recall_score(y,y_predict))
```

```
0.71129221295567
[[2516  264]
 [ 921  988]]
              precision    recall  f1-score   support

           0       0.73      0.91      0.81      2780
           1       0.79      0.52      0.63      1909

    accuracy                           0.75      4689
   macro avg       0.76      0.71      0.72      4689
weighted avg       0.76      0.75      0.73      4689

0.7891373801916933
```

In [95]:
```
pip=make_pipeline(model,smote,clf)
pip.fit(X,y)
y_predict = pip.predict(X)
accuracy_score(pip.predict(X), y)
print(roc_auc_score(pip.predict(X),y))
print(confusion_matrix(pip.predict(X), y))
print(classification_report(pip.predict(X), y))
print(recall_score(y,y_predict))
```

```
0.7079926836844121
[[2519  277]
 [ 918  975]]
              precision    recall  f1-score   support

           0       0.73      0.90      0.81      2796
           1       0.78      0.52      0.62      1893

    accuracy                           0.75      4689
   macro avg       0.76      0.71      0.71      4689
weighted avg       0.75      0.75      0.73      4689

0.7787539936102237
```

In [102]:
```
#knn suffers from imbalanced data (knn is a distance based model).We will use weighted knn.#The intuition behind
#weighted kNN, is to give more weight to the points which are nearby and less weight to the points
#which are farther away.So the performance of the model improves when k increases using weighted knn.
```

In [96]:
```
k=[1,5,11,21]
for i in k:
    neigh=KNeighborsClassifier(n_neighbors=i,weights ='distance')
    neigh.fit(X,y)
    scores = cross_val_score(neigh, X, y, cv=10, scoring='accuracy')
    print(scores)
    print(scores.mean())
    pred=cross_val_predict(neigh, X,y, cv=10)
    print(confusion_matrix(y,pred))
    print(classification_report(y, pred))
    print(recall_score(y,pred))
```

```
[0.69296375 0.70362473 0.70149254 0.69722814 0.68230277 0.68230277
 0.73134328 0.7249467  0.73987207 0.75854701]
0.7114623767608843
[[2711  726]
 [ 627  625]]
              precision    recall  f1-score   support

           0       0.81      0.79      0.80      3437
           1       0.46      0.50      0.48      1252

    accuracy                           0.71      4689
   macro avg       0.64      0.64      0.64      4689
weighted avg       0.72      0.71      0.71      4689

0.4992012779552716
[0.75266525 0.75266525 0.73134328 0.74200426 0.75906183 0.7249467
 0.7761194  0.74413646 0.77185501 0.77777778]
0.7532575219142383
[[2892  545]
```

In [98]:
```
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X,y)
scores = cross_val_score(clf, X, y, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(clf,X,y, cv=10)
print(confusion_matrix(y,pred))
print(classification_report(y, pred))
#Decision trees is not good for large data.Decission Trees tend to overfit when there is a large number of splits
recall_score(y,pred)
```

```
[0.74840085 0.73987207 0.74200426 0.70149254 0.71002132 0.72921109
 0.75053305 0.73560768 0.73347548 0.73504274]
0.7325661071929729
[[2757  680]
 [ 594  658]]
              precision    recall  f1-score   support

           0       0.82      0.80      0.81      3437
           1       0.49      0.53      0.51      1252

    accuracy                           0.73      4689
   macro avg       0.66      0.66      0.66      4689
weighted avg       0.73      0.73      0.73      4689
```

Out[98]:
```
0.5255591054313099
```

In [106]:
```
from sklearn import svm
clf = svm.SVC(kernel='linear')
clf.fit(X,y)
scores = cross_val_score(clf, X,y, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(clf,X,y, cv=10)
print(confusion_matrix(y,pred))
print(classification_report(y, pred))
recall_score(y,pred)
```

```
[0.82302772 0.78678038 0.76972281 0.79317697 0.80810235 0.7803838
 0.80170576 0.79957356 0.78678038 0.8034188 ]
0.7952672534762086
[[3072  365]
 [ 595  657]]
              precision    recall  f1-score   support

           0       0.84      0.89      0.86      3437
           1       0.64      0.52      0.58      1252

    accuracy                           0.80      4689
   macro avg       0.74      0.71      0.72      4689
weighted avg       0.79      0.80      0.79      4689
```

Out[106]:
```
0.5247603833865815
```

In [108]:
```
from sklearn import svm
clf2 = svm.SVC(kernel='linear')
clf2.fit(x_smote,y_smote)
scores = cross_val_score(clf2,x_smote,y_smote, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(clf2,x_smote,y_smote, cv=10)
print(confusion_matrix(y_smote,pred))
print(classification_report(y_smote, pred))
recall_score(y_smote,pred)
```

```
[0.72819767 0.73546512 0.7122093  0.7877907  0.90247453 0.90684134
 0.91703057 0.90684134 0.92139738 0.90684134]
0.842508928269185
[[2971  466]
 [ 617 2820]]
              precision    recall  f1-score   support

           0       0.83      0.86      0.85      3437
           1       0.86      0.82      0.84      3437

    accuracy                           0.84      6874
   macro avg       0.84      0.84      0.84      6874
weighted avg       0.84      0.84      0.84      6874
```

Out[108]:
```
0.8204829793424498
```

In [107]:
```
pip=make_pipeline(rfe,smote,clf)
pip.fit(X,y)
y_predict = pip.predict(X)
accuracy_score(pip.predict(X), y)
print(roc_auc_score(pip.predict(X),y))
print(confusion_matrix(pip.predict(X), y))
print(classification_report(pip.predict(X), y))
print(recall_score(y,y_predict))
accuracy_score(y,y_predict)
```

```
0.6960097263387475
[[2248  193]
 [1189 1059]]
              precision    recall  f1-score   support

           0       0.65      0.92      0.76      2441
           1       0.85      0.47      0.61      2248

    accuracy                           0.71      4689
   macro avg       0.75      0.70      0.69      4689
weighted avg       0.75      0.71      0.69      4689

0.8458466453674122
```

Out[107]:
```
0.7052676476860738
```

In [109]:
```
pip=make_pipeline(model,smote,clf)
pip.fit(X,y)
y_predict = pip.predict(X)
accuracy_score(pip.predict(X), y)
print(roc_auc_score(pip.predict(X),y))
print(confusion_matrix(pip.predict(X), y))
print(classification_report(pip.predict(X), y))
print(recall_score(y,y_predict))
accuracy_score(y,y_predict)
```

```
0.6946770521073926
[[2211  183]
 [1226 1069]]
              precision    recall  f1-score   support

           0       0.64      0.92      0.76      2394
           1       0.85      0.47      0.60      2295

    accuracy                           0.70      4689
   macro avg       0.75      0.69      0.68      4689
weighted avg       0.75      0.70      0.68      4689

0.8538338658146964
```

Out[109]:
```
0.6995094902964385
```

In [110]:
```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X,y)
scores = cross_val_score(clf, X,y, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(clf,X,y, cv=10)
print(confusion_matrix(y,pred))
print(classification_report(y, pred))
recall_score(y,pred)
```

```
[0.71855011 0.73347548 0.6652452  0.66098081 0.66950959 0.68230277
 0.71002132 0.67803838 0.70788913 0.6965812 ]
0.6922593989758169
[[2205 1232]
 [ 211 1041]]
              precision    recall  f1-score   support

           0       0.91      0.64      0.75      3437
           1       0.46      0.83      0.59      1252

    accuracy                           0.69      4689
   macro avg       0.69      0.74      0.67      4689
weighted avg       0.79      0.69      0.71      4689
```

Out[110]:
```
0.8314696485623003
```

In [105]:
```
clf2 = GaussianNB()
clf2.fit(x_smote,y_smote)
scores = cross_val_score(clf2, x_smote,y_smote, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(clf2,x_smote,y_smote, cv=10)
print(confusion_matrix(y_smote,pred))
print(classification_report(y_smote, pred))
recall_score(y_smote,pred)
```

```
[0.77616279 0.74854651 0.72093023 0.73255814 0.75691412 0.75691412
 0.77292576 0.73508006 0.77438137 0.76128093]
0.7535694035408417
[[2269 1168]
 [ 526 2911]]
              precision    recall  f1-score   support

           0       0.81      0.66      0.73      3437
           1       0.71      0.85      0.77      3437

    accuracy                           0.75      6874
   macro avg       0.76      0.75      0.75      6874
weighted avg       0.76      0.75      0.75      6874
```

Out[105]:
```
0.8469595577538551
```

In [111]:
```
pip=make_pipeline(rfe,smote,clf)
pip.fit(X,y)
y_predict = pip.predict(X)
accuracy_score(pip.predict(X), y)
print(roc_auc_score(pip.predict(X),y))
print(confusion_matrix(pip.predict(X), y))
print(classification_report(pip.predict(X), y))
print(recall_score(y,y_predict))
accuracy_score(y,y_predict)
```

```
0.7070689302071212
[[2550  297]
 [ 887  955]]
              precision    recall  f1-score   support

           0       0.74      0.90      0.81      2847
           1       0.76      0.52      0.62      1842

    accuracy                           0.75      4689
   macro avg       0.75      0.71      0.71      4689
weighted avg       0.75      0.75      0.74      4689

0.762779552715655
```

Out[111]:
```
0.7474941352100661
```

In [ ]:
```
#Recall is the metric that we have to take into account in this specific problem..
#we are interested in the proportion of churns identified correctly by the total number of churns
```

In [116]:

```
clf = LogisticRegression(class_weight="balanced")
clf.fit(X_f,y_f)
scores = cross_val_score(clf,X_f,y_f, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(clf,X_f,y_f, cv=10)
print(confusion_matrix(y_f,pred))
print(classification_report(y_f, pred))
recall_score(y_f,pred)
#(logistic regression with data from which we droped the least important
#features using random forest "feature selection")
```

```
[0.75692964 0.74626866 0.74626866 0.70788913 0.72921109 0.73773987
 0.76119403 0.75053305 0.76972281 0.74358974]
0.7449346673227271
[[2501  936]
 [ 260  992]]
              precision    recall  f1-score   support

           0       0.91      0.73      0.81      3437
           1       0.51      0.79      0.62      1252

    accuracy                           0.74      4689
   macro avg       0.71      0.76      0.72      4689
weighted avg       0.80      0.74      0.76      4689
```

Out[116]:

```
0.792332268370607
```

In [117]:
```
pip=make_pipeline(rfe,smote,clf)
pip.fit(X,y)
y_predict = pip.predict(X)
accuracy_score(pip.predict(X), y)
print(roc_auc_score(pip.predict(X),y))
print(confusion_matrix(pip.predict(X), y))
print(classification_report(pip.predict(X), y))
print(recall_score(y,y_predict))
#good model
```

```
0.711134676684494
[[2530  272]
 [ 907  980]]
              precision    recall  f1-score   support

           0       0.74      0.90      0.81      2802
           1       0.78      0.52      0.62      1887

    accuracy                           0.75      4689
   macro avg       0.76      0.71      0.72      4689
weighted avg       0.75      0.75      0.74      4689

0.7827476038338658
```

In [119]:
```
gr = GradientBoostingClassifier()
gr.fit(X_f,y_f)
scores = cross_val_score(gr, X_f,y_f, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(clf,X_f,y_f, cv=10)
print(confusion_matrix(y_f,pred))
print(classification_report(y_f, pred))
recall_score(y_f,pred)
```

```
[0.80170576 0.80597015 0.78464819 0.78464819 0.81876333 0.79530917
 0.80383795 0.78464819 0.80383795 0.81623932]
0.799960818617535
[[2501  936]
 [ 260  992]]
              precision    recall  f1-score   support

           0       0.91      0.73      0.81      3437
           1       0.51      0.79      0.62      1252

    accuracy                           0.74      4689
   macro avg       0.71      0.76      0.72      4689
weighted avg       0.80      0.74      0.76      4689
```

Out[119]: 0.792332268370607

In [121]:
```
pip=make_pipeline(model,smote,gr)
pip.fit(X,y)
y_predict = pip.predict(X)
accuracy_score(pip.predict(X), y)
print(roc_auc_score(pip.predict(X),y))
print(confusion_matrix(pip.predict(X), y))
print(classification_report(pip.predict(X), y))
print(recall_score(y,y_predict))
```

```
0.7400622670052407
[[2660  245]
 [ 777 1007]]
              precision    recall  f1-score   support

           0       0.77      0.92      0.84      2905
           1       0.80      0.56      0.66      1784

    accuracy                           0.78      4689
   macro avg       0.79      0.74      0.75      4689
weighted avg       0.79      0.78      0.77      4689

0.8043130990415336
```

In [122]:
```
pip=make_pipeline(rfe,smote,gr)
pip.fit(X,y)
y_predict = pip.predict(X)
accuracy_score(pip.predict(X), y)
print(roc_auc_score(pip.predict(X),y))
print(confusion_matrix(pip.predict(X), y))
print(classification_report(pip.predict(X), y))
print(recall_score(y,y_predict))
```

```
0.7450380143889006
[[2798  321]
 [ 639  931]]
              precision    recall  f1-score   support

           0       0.81      0.90      0.85      3119
           1       0.74      0.59      0.66      1570

    accuracy                           0.80      4689
   macro avg       0.78      0.75      0.76      4689
weighted avg       0.79      0.80      0.79      4689

0.7436102236421726
```

In [132]:
```
#Now,we will use the data set after smote sampling and after the drop of the least important
#features that was found with the Random Forest Algorithm
xf_smote, yf_smote = smote.fit_sample(X_f,y_f)
```

In [133]:
```python
gr = GradientBoostingClassifier()
gr.fit(xf_smote,yf_smote)
scores = cross_val_score(gr, xf_smote,yf_smote, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(gr,xf_smote,yf_smote, cv=10)
print(confusion_matrix(yf_smote,pred))
print(classification_report(yf_smote, pred))
recall_score(yf_smote,pred)
```

```
[0.78633721 0.75872093 0.74854651 0.76744186 0.86608443 0.8558952
 0.88500728 0.86899563 0.86608443 0.86754003]
0.8270653498527469
[[2740  697]
 [ 492 2945]]
              precision    recall  f1-score   support

           0       0.85      0.80      0.82      3437
           1       0.81      0.86      0.83      3437

    accuracy                           0.83      6874
   macro avg       0.83      0.83      0.83      6874
weighted avg       0.83      0.83      0.83      6874
```

Out[133]:
```
0.8568519057317427
```

In [134]:
```
ada = AdaBoostClassifier()
ada.fit(xf_smote, yf_smote)
scores = cross_val_score(ada, xf_smote, yf_smote, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(ada,xf_smote, y_smote, cv=10)
print(confusion_matrix(yf_smote,pred))
print(classification_report(yf_smote, pred))
recall_score(yf_smote,pred)
```

```
[0.79069767 0.7630814  0.74709302 0.74709302 0.82532751 0.82387191
 0.85298399 0.83697234 0.83988355 0.83551674]
0.8062521157035982
[[2643  794]
 [ 538 2899]]
              precision    recall  f1-score   support

           0       0.83      0.77      0.80      3437
           1       0.78      0.84      0.81      3437

    accuracy                           0.81      6874
   macro avg       0.81      0.81      0.81      6874
weighted avg       0.81      0.81      0.81      6874
```

Out[134]:
```
0.8434681408204829
```

In [127]:
```
# using the following model in the test set
BEST=make_pipeline(model,smote,gr)
BEST.fit(X,y)
y_predict = BEST.predict(X)
accuracy_score(BEST.predict(X), y)
print(roc_auc_score(BEST.predict(X),y))
print(confusion_matrix(BEST.predict(X), y))
print(classification_report(BEST.predict(X), y))
print(recall_score(y,y_predict))
```

```
0.7368794188261255
[[2648  249]
 [ 789 1003]]
              precision    recall  f1-score   support

           0       0.77      0.91      0.84      2897
           1       0.80      0.56      0.66      1792

    accuracy                           0.78      4689
   macro avg       0.79      0.74      0.75      4689
weighted avg       0.78      0.78      0.77      4689

0.8011182108626198
```

In [136]:
```
BEST2 = GradientBoostingClassifier()
BEST2.fit(xf_smote,yf_smote)
scores = cross_val_score(BEST2, xf_smote,yf_smote, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
pred=cross_val_predict(BEST2,xf_smote,yf_smote, cv=10)
print(confusion_matrix(yf_smote,pred))
print(classification_report(yf_smote, pred))
recall_score(yf_smote,pred)
```

```
[0.78633721 0.75872093 0.74854651 0.76744186 0.86608443 0.8558952
 0.88500728 0.86899563 0.86608443 0.86899563]
0.8272109102603162
[[2740  697]
 [ 492 2945]]
              precision    recall  f1-score   support

           0       0.85      0.80      0.82      3437
           1       0.81      0.86      0.83      3437

    accuracy                           0.83      6874
   macro avg       0.83      0.83      0.83      6874
weighted avg       0.83      0.83      0.83      6874
```

Out[136]:
```
0.8568519057317427
```

In [128]:
```
#selecting 300 customers randomly
test_data_300 = test_dummies.sample(n=300)
churn_predictions = BEST.predict(test_data_300.drop('Churn', axis=1))
number_true_churn_customers = np.count_nonzero(churn_predictions == test_data_300['Churn'])
print('The number of true churn customers among the selected 300 customers is:', number_true_churn_customers)
```

```
The number of true churn customers among the selected 300 customers is: 209
```

In [129]:
```
print(sum(churn_predictions))
```

```
142
```

In [ ]:
```
#Expected(cost)=10*142+64*(209-142)=5708 EUROS
```

In [138]:
```
#SECOND WAY(with BEST2)
test2=test_dummies.drop(columns=['PhoneService_No','PhoneService_Yes', "MultipleLines_'No phone service'",'InternetService_No', "OnlineSecurity_'N
"StreamingMovies_'No internet service'"])
```

In [139]:
```
#selecting 300 customers randomly
test2_data_300 = test2.sample(n=300)
churn_predictions = BEST2.predict(test2_data_300.drop('Churn', axis=1))
number_true_churn_customers = np.count_nonzero(churn_predictions == test2_data_300['Churn'])
print('The number of true churn customers among the selected 300 customers is:', number_true_churn_customers)
```

```
The number of true churn customers among the selected 300 customers is: 232
```

In [140]:
```
print(sum(churn_predictions))
#9282Euros the cost with the second way
```

```
103
```

Results

Descriptive Task Characteristics of loyal and churn customers Customers taking a longer contract(2 years contruct)are more loyal to the company and tend to stay with it for a longer period of time.Customers who dont churn tend to stay for a longer tenure with the telecom company Customers who have month to month contract have a very high churn rate. Senior citizens have almost double the churn rate than younger population When monthly charges are high a big percent of customers churn.The absence of online security is also a reason that make customers to churn. Most churn customers are those who have contract only for a month (this is logical as after the end of their contract is most possible to leave their "product" than others who have longer contracts), and those who dont have services like tech support and online security and also those who use fiber optic internet servise(which surprisingly means faster internet) because probably it is more expensive and also because the most of churn customers are senior citizens who probably dont need very fast internet. Proposed system The system I would suggest to the Telecom company is to always make a two-year contract with its customers, to have DSL internet service and to include tech support in the contract.

Predictive Task The predictive model used for our test set is Gradient Boosting Algorithm with smote sampling and tree based feature selection. The expected cost for the company in one month will be 5709 Euros