In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sys
import seaborn as sns
from scipy.stats import chi2
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter)
import Regression

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
telco = pd.read_csv('Telco-Customer-Churn.csv')
```

In [3]:
```
telco["Churn"] = np.where((telco['Churn'] =='Yes'),1,0)
telco = telco[telco["TotalCharges"]!=' '].reset_index()
telco["TotalCharges"] = telco["TotalCharges"].astype("float64")
telco["SeniorCitizen"] = telco["SeniorCitizen"].astype("object")
```

In [4]:
```
catName = ["gender","SeniorCitizen","Partner","Dependents","PhoneService","MultipleLines","Contract","PaperlessBillir
intName = ["tenure","MonthlyCharges","TotalCharges"]
```

In [5]:
```
yName = "Churn"

trainData = telco[catName+intName+[yName]]

# Generate a column of Intercept
X0_train = trainData[[yName]].copy()
X0_train.insert(0, 'Intercept', 1.0)
X0_train.drop(columns = [yName], inplace = True)

y_train = trainData[yName].copy()
n_sample = trainData.shape[0]
```

In [6]:
```python
# Reorder the categories of the categorical variables in ascending frequency
for pred in catName:
    u = trainData[pred].astype('category').copy()
    u_freq = u.value_counts(ascending = True)
    trainData[pred] = u.cat.reorder_categories(list(u_freq.index)).copy()
```

Question 1

a) For each categorical predictor, generate a bar chart that shows the odds of Churn for each category. Please order the categories in ascending odds of Churn. Also, please comment on each categorical predictor on whether it may affect the target variable.

def convert_object_to_category(df): for i in catName: df[i] = df[i].astype("category") freq = df[i].value_counts(ascending = True) one_pm = df[i].cat.reorder_categories(list(freq.index)) series.append(one_pm)
return pd.concat(series,axis=1)

In [7]:
```
cate_pred = trainData[catName].join(y_train)
cate_pred
```

Out[7]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	Contract	PaperlessBilling	Churn
0	Female	0	Yes	No	No	No phone service	Month-to-month	Yes	0
1	Male	0	No	No	Yes	No	One year	No	0
2	Male	0	No	No	Yes	No	Month-to-month	Yes	1
3	Male	0	No	No	No	No phone service	One year	No	0
4	Female	0	No	No	Yes	No	Month-to-month	Yes	1
...	...	...	...	...	...	...	...	...	...
7027	Male	0	Yes	Yes	Yes	Yes	One year	Yes	0
7028	Female	0	Yes	Yes	Yes	Yes	One year	Yes	0
7029	Female	0	Yes	Yes	No	No phone service	Month-to-month	Yes	0
7030	Male	1	Yes	No	Yes	Yes	Month-to-month	Yes	1
7031	Male	0	No	No	Yes	No	Two year	Yes	0

7032 rows × 9 columns

In [8]:

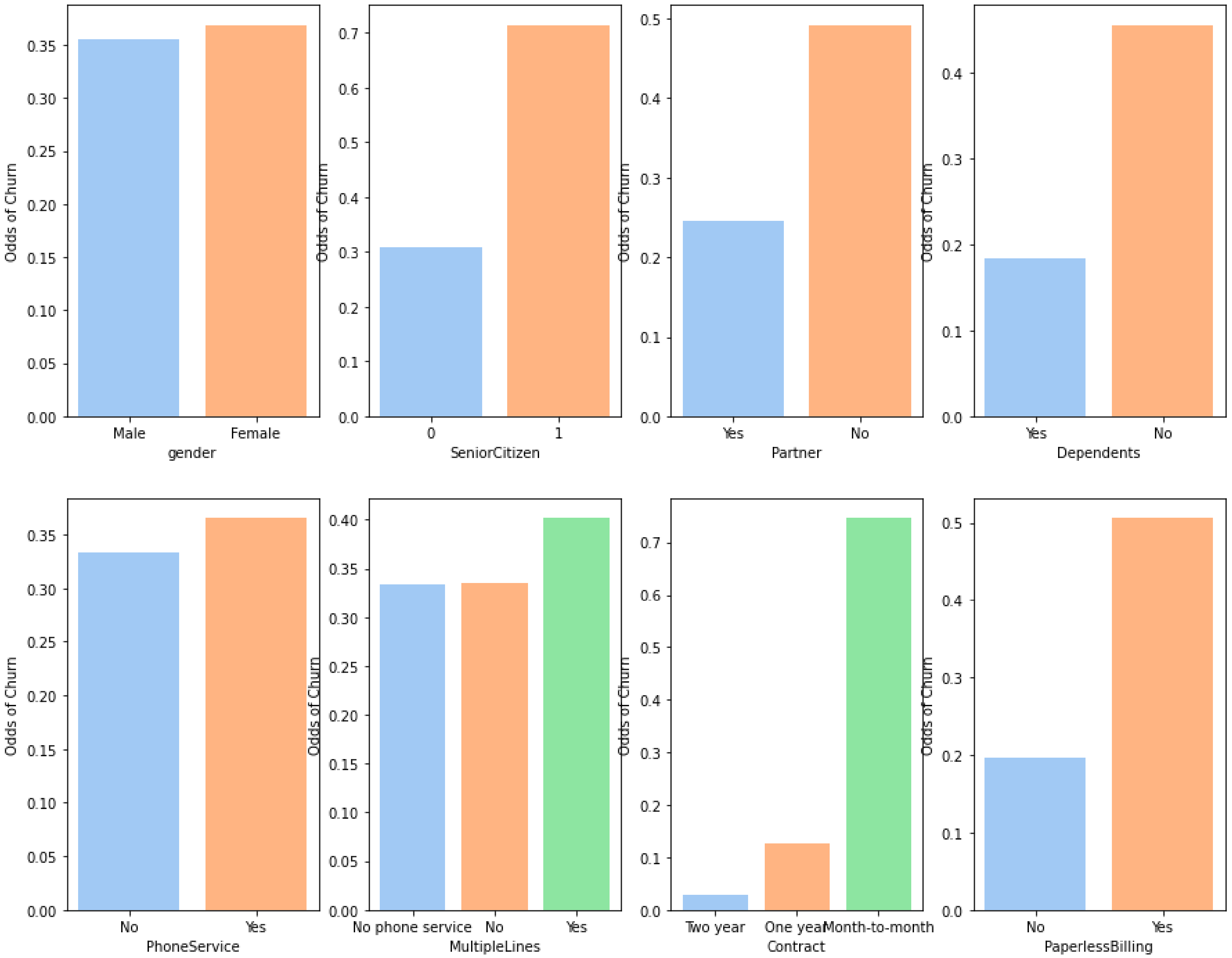
```
#  Categorical Data
cate_columns = cate_pred.iloc[:,:-1].columns

a = 2  # number of rows
b = 4 # number of columns
c = 1  # initialize plot counter

fig = plt.figure(figsize=(15,12))

for yvar in cate_columns:
    plt.subplot(a, b, c)
    # plt.title('{}, subplot: {}{}{}'.format(i, a, b, c))
    plt.xlabel(yvar)
    xtab = pd.crosstab(index = cate_pred[yvar], columns = cate_pred['Churn'])
    xtab.reset_index(inplace = True)
    xtab['N'] = xtab[0] + xtab[1]
    xtab['Odds'] = xtab[1] / xtab[0]
    xtab.sort_values(by = 'Odds', inplace = True)
    # palette "BuGn" is awesome
    plt.bar(xtab[yvar],xtab['Odds'],color = sns.color_palette("pastel"))
    plt.xlabel(yvar)
    plt.ylabel('Odds of Churn')
    plt.xticks(xtab[yvar])
    c = c + 1

plt.show()
```



Odds of Churn
gender
Male
Female
SeniorCitizen
0
1
Partner
Yes
No
Dependents
PhoneService
MultipleLines
No phone service
Contract
Two year
One year
Month-to-month
PaperlessBilling

Categorical variables of SeniorCitizen, Partner, Dependents, Contract, Paperless will may affect on target variable. Gender, Phone Service, Mutiplelines won't affect much since they have similar odds of churn among each its categoires.

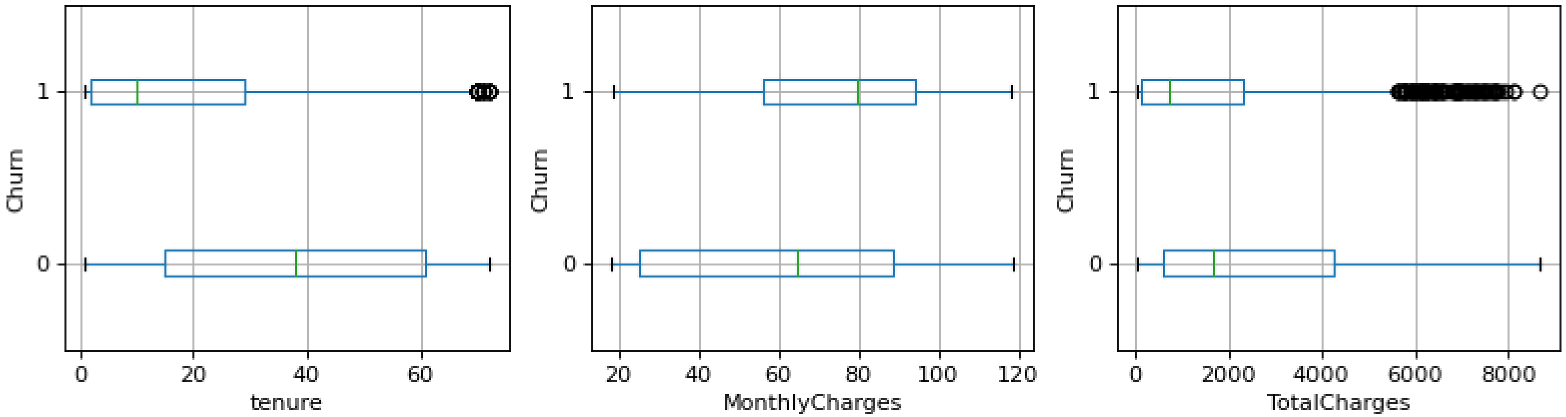
b) For each interval predictor, generate a grouped boxplot that shows the distribution of the interval predictor. The grouping variable, in this case, is the target variable. Also, please comment on each interval predictor on whether it may affect the target variable.

In [9]:

```
interval_pred = trainData[intName].join(trainData["Churn"])
```

In [10]:

```
plt.rcParams["figure.figsize"] = (10,3)
fig, ax = plt.subplots(ncols = 3,nrows=1,dpi = 80)
for i in range(len(intName)):
    interval_pred.boxplot(column = intName[i], by = 'Churn', ax = ax.flatten()[i], vert = False)
    ax.flatten()[i].set_xlabel(intName[i])
    ax.flatten()[i].set_ylabel("Churn")
    ax.flatten()[i].yaxis.grid(True)
    ax.flatten()[i].set_title("")
    plt.suptitle("")
plt.tight_layout()
```



Tenure may have the largest effect on odds of churn, while monthlycharges and total charges may have effect but not as much as tenure

tenure.

Question 2

a) Please provide a summary report of the Forward Selection. The report should include (1) the step number, (2) the predictor entered, (3) the number of non-aliased parameters in the current model, (4) the log-likelihood value of the current model, (5) the Deviance Chi-squares statistic between the current and the previous models, (6) the corresponding Deviance Degree of Freedom, and (7) the corresponding Chi-square significance.

In [11]:

```python
maxIter = 20
tolS = 1e-7
stepSummary = pd.DataFrame()

# Intercept only model
resultList = Regression.BLogisticModel (X0_train, y_train, offset = None, maxIter = maxIter, tolSweep = tolS)

llk0 = resultList[3]
df0 = len(resultList[4])
stepSummary = stepSummary.append([['Intercept', ' ', df0, llk0, np.NaN, np.NaN, np.NaN]], ignore_index = True)
stepSummary.columns = ['Predictor', 'Type', 'ModelDF', 'ModelLLK', 'DevChiSq', 'DevDF', 'DevSig']

print('======= Step Detail =======')
print('Step = ', 0)
print('Step Statistics:')
print(stepSummary)

cName = catName.copy()
iName = intName.copy()
```

```
======= Step Detail =======
Step =  0
Step Statistics:
   Predictor Type  ModelDF     ModelLLK  DevChiSq  DevDF  DevSig
0  Intercept             1 -4071.677573       NaN    NaN     NaN
```

In [12]:

```
entryThreshold = 0.001
nPredictor = len(catName) + len(intName)

for step in range(nPredictor):
    enterName = ''
    stepDetail = pd.DataFrame()

    # Enter the next predictor
    for X_name in cName:
        X_train = pd.get_dummies(trainData[[X_name]])
        X_train = X0_train.join(X_train)
        resultList = Regression.BLogisticModel (X_train, y_train, offset = None, maxIter = maxIter, tolSweep = tolS)
        llk1 = resultList[3]
        df1 = len(resultList[4])
        devChiSq = 2.0 * (llk1 - llk0)
        devDF = df1 - df0
        devSig = chi2.sf(devChiSq, devDF)
        stepDetail = stepDetail.append([[X_name, 'categorical', df1, llk1, devChiSq, devDF, devSig]], ignore_index =

    for X_name in iName:
        X_train = trainData[[X_name]]
        X_train = X0_train.join(X_train)
        resultList = Regression.BLogisticModel (X_train, y_train, offset = None, maxIter = maxIter, tolSweep = tolS)
        llk1 = resultList[3]
        df1 = len(resultList[4])
        devChiSq = 2.0 * (llk1 - llk0)
        devDF = df1 - df0
        devSig = chi2.sf(devChiSq, devDF)
        stepDetail = stepDetail.append([[X_name, 'interval', df1, llk1, devChiSq, devDF, devSig]], ignore_index = Tru

    stepDetail.columns = ['Predictor', 'Type', 'ModelDF', 'ModelLLK', 'DevChiSq', 'DevDF', 'DevSig']


     # Find a predictor to enter, if any
    stepDetail.sort_values(by = 'DevSig', axis = 0, ascending = True, inplace = True)
    enterRow = stepDetail.iloc[0].copy()
    minPValue = enterRow['DevSig']
    if (minPValue <= entryThreshold):
        stepSummary = stepSummary.append([enterRow], ignore_index = True)
        df0 = enterRow['ModelDF']
        llk0 = enterRow['ModelLLK']
```

```python
        enterName = enterRow['Predictor']
        enterType = enterRow['Type']
        if (enterType == 'categorical'):
            X_train = pd.get_dummies(trainData[[enterName]].astype('category'))
            X0_train = X0_train.join(X_train)
            cName.remove(enterName)
        elif (enterType == 'interval'):
            X_train = trainData[[enterName]]
            X0_train = X0_train.join(X_train)
            iName.remove(enterName)
    else:
        break
```

In [13]:

```python
# Print debugging output
print('======= Step Detail =======')
print('Step = ', step+1)
print('Step Statistics:')
print(stepDetail)
print('Enter predictor = ', enterName)
print('Minimum P-Value =', minPValue)
print('\n')
```

```
======= Step Detail =======
Step =  8
Step Statistics:
      Predictor         Type  ModelDF     ModelLLK  DevChiSq  DevDF    DevSig
2    Dependents  categorical       11 -2993.825597  6.395833      1  0.011439
1       Partner  categorical       11 -2996.660886  0.725256      1  0.394425
0        gender  categorical       11 -2997.010328  0.026373      1  0.870993
3  PhoneService  categorical       10 -2997.023514  0.000000      0       NaN
Enter predictor = 
Minimum P-Value = 0.011438851234830882
```

In [14]:
```
# End of forward selection
stepSummary
```

Out[14]:

	Predictor	Type	ModelDF	ModelLLK	DevChiSq	DevDF	DevSig
0	Intercept		1	-4071.677573	NaN	NaN	NaN
1	Contract	categorical	3	-3381.260348	1380.834450	2.0	1.430899e-300
2	MonthlyCharges	interval	4	-3241.782940	278.954817	1.0	1.268706e-62
3	tenure	interval	5	-3071.742414	340.081052	1.0	6.126591e-76
4	MultipleLines	categorical	7	-3034.375474	74.733880	2.0	5.912133e-17
5	PaperlessBilling	categorical	8	-3015.225743	38.299461	1.0	6.067939e-10
6	SeniorCitizen	categorical	9	-3003.996546	22.458394	1.0	2.147450e-06
7	TotalCharges	interval	10	-2997.023514	13.946064	1.0	1.881312e-04

b) Please show a table of the complete set of parameters of your final model (including the aliased parameters). Besides the parameter estimates, please also include the standard errors, and the 95% asymptotic confidence intervals. Conventionally, aliased parameters have missing standard errors and confidence intervals.

In [15]:
```python
# Final model
resultList = Regression.BLogisticModel (X0_train, y_train, offset = None, maxIter = maxIter, tolSweep = tolS)
resultList[0]
```

Out[15]:

	Estimate	Standard Error	Lower 95% CI	Upper 95% CI
Intercept	-1.305712	0.141101	-1.582266	-1.029159
Contract_One year	-0.926626	0.103280	-1.129051	-0.724201
Contract_Two year	-1.842079	0.171081	-2.177390	-1.506767
Contract_Month-to-month	0.000000	0.000000	0.000000	0.000000
MonthlyCharges	0.023901	0.001994	0.019992	0.027810
tenure	-0.061276	0.006041	-0.073116	-0.049436
MultipleLines_No phone service	0.939374	0.121371	0.701492	1.177256
MultipleLines_Yes	0.222655	0.079157	0.067511	0.377799
MultipleLines_No	0.000000	0.000000	0.000000	0.000000
PaperlessBilling_No	-0.432488	0.072438	-0.574464	-0.290512
PaperlessBilling_Yes	0.000000	0.000000	0.000000	0.000000
SeniorCitizen_1	0.383080	0.080771	0.224771	0.541389
SeniorCitizen_0	0.000000	0.000000	0.000000	0.000000
TotalCharges	0.000250	0.000068	0.000117	0.000383

Quesetion 3

a) Please calculate the McFadden’s R-squared, the Cox-Snell’s R-squared, the Nagelkerke’s R-squared, and the Tjur’s Coefficient of Discrimination

In [16]:
```
model_llk_final = resultList[3]
model_llk0 = -4071.677573

R_MF = 1.0 - (model_llk_final / model_llk0)
R_MF
```

Out[16]: 0.2639339779748823

In [17]:
```
R_CS = np.exp(model_llk0 - model_llk_final)
R_CS = 1.0 - np.power(R_CS, (2.0 / n_sample))
R_CS
```

Out[17]: 1.0

In [18]:
```
upbound = 1.0 - np.power(np.exp(model_llk0), (2.0 / n_sample))
R_N = R_CS / upbound
R_N
```

Out[18]: 1.0

In [19]:
```
predprob_event = resultList[6][1]
abc = predprob_event[y_train == 1]
S1 = np.mean(predprob_event[y_train == 1])
S0 = np.mean(predprob_event[y_train == 0])

R_TJ = S1 - S0
R_TJ
```

Out[19]: 0.28203430255131934

b) Please calculate the Area Under Curve statistic and the Root Average Squared Error.

In [20]:
```
predProbY =  resultList[6][1]
predProbY
```

Out[20]:
```
0       0.572803
1       0.051418
2       0.471491
3       0.046888
4       0.574307
          ...
7027    0.277643
7028    0.107605
7029    0.438723
7030    0.712954
7031    0.049414
Name: 1, Length: 7032, dtype: float64
```

In [21]:
```
y_pred = np.where(predProbY >= 0.5, 1, 0)
y_pred
```

Out[21]:
```
array([1, 0, 0, ..., 0, 1, 0])
```

In [22]:

```python
def binary_model_metric (target, valueEvent, valueNonEvent, predProbEvent, eventProbThreshold = 0.5):
    '''
    Calculate metrics for a binary classification model
    Parameter
    ---------
    target: Panda Series that contains values of target variable
    valueEvent: Formatted value of target variable that indicates an event
    valueNonEvent: Formatted value of target variable that indicates a non-event
    predProbEvent: Panda Series that contains predicted probability that the event will occur
    eventProbThreshold: Threshold for event probability to indicate a success

    Return
    ------
    outSeries: Pandas Series that contain the following statistics
               ASE: Average Squared Error
               RASE: Root Average Squared Error
               MCE: Misclassification Rate
               AUC: Area Under Curve
   '''
    # Number of observations
    nObs = len(target)
    # Aggregate observations by the target values and the predicted probabilities
    aggrProb = pd.crosstab(predProbEvent, target, dropna = True)
    # Calculate the root average square error
    ase = (np.sum(aggrProb[valueEvent] * (1.0 - aggrProb.index)**2) +np.sum(aggrProb[valueNonEvent] * (0.0 - aggrProb
    if (ase > 0.0):
        rase = np.sqrt(ase)
    else:
        rase = 0.0
    # Calculate the misclassification error rate
    nFP = np.sum(aggrProb[valueEvent].iloc[aggrProb.index < eventProbThreshold])
    nFN = np.sum(aggrProb[valueNonEvent].iloc[aggrProb.index >= eventProbThreshold])
    mce = (nFP + nFN) / nObs
    # Calculate the number of concordant, discordant, and tied pairs
    nConcordant = 0.0
    nDiscordant = 0.0
    nTied = 0.0
    # Loop over the predicted event probabilities from the Event column
    predEP = aggrProb.index
    eventFreq = aggrProb[valueEvent]
    for i in range(len(predEP)):
```

```
        eProb = predEP[i]
        eFreq = eventFreq.loc[eProb]
        if (eFreq > 0.0):
            nConcordant = nConcordant + np.sum(eFreq * aggrProb[valueNonEvent].iloc[eProb > aggrProb.index])
            nDiscordant = nDiscordant + np.sum(eFreq * aggrProb[valueNonEvent].iloc[eProb < aggrProb.index])
            nTied = nTied + np.sum(eFreq * aggrProb[valueNonEvent].iloc[eProb == aggrProb.index])
    auc = 0.5 + 0.5 * (nConcordant - nDiscordant) / (nConcordant + nDiscordant + nTied)
    outSeries = pd.Series({'ASE': ase, 'RASE': rase, 'MCE': mce, 'AUC': auc})
    return(outSeries)
```

In [23]:

```
binary_model_metric(y_train,1,0,predprob_event,0.5)
```

Out[23]:

```
ASE     0.139227
RASE    0.373132
MCE     0.202645
AUC     0.837631
dtype: float64
```

c) According to the F1 Score, please suggest the probability threshold for Churn. Using this threshold, what is the misclassification rate?

In [24]:

```python
threshold_list = []
TP_list = []
TN_list = []
FP_list = []
FN_list = []
Misclassification_rate_list = []
F1_list = []


for threshold in np.arange(0.1,0.9,0.1):
    each_y_pred = np.where(predProbY >= threshold, 1, 0)
    TP = sum((y_train==1) & (each_y_pred==1))
    TN = sum((y_train==0) & (each_y_pred==0))
    FP = sum((y_train==1) & (each_y_pred==0))
    FN = sum((y_train==0) & (each_y_pred==1))

    precision = TP/(TP+FP)
    recall = TP/(TP+FN)

    Misclassification_rate = (FP+FN)/(TP+TN+FP+FN)
    F1 = 2*(precision*recall) / (precision+recall)

    threshold_list.append(threshold)

    TP_list.append(TP)
    TN_list.append(TN)
    FP_list.append(FP)
    FN_list.append(FN)
    Misclassification_rate_list.append(Misclassification_rate)

    F1_list.append(F1)

F1_by_threshold = pd.DataFrame({"threshold":threshold_list,"TP":TP_list,"TN":TN_list,"FP":FP_list,"FN":FN_list,
                                "Misclassification_rate":Misclassification_rate_list,"F1":F1_list})
F1_by_threshold
```

Out[24]:

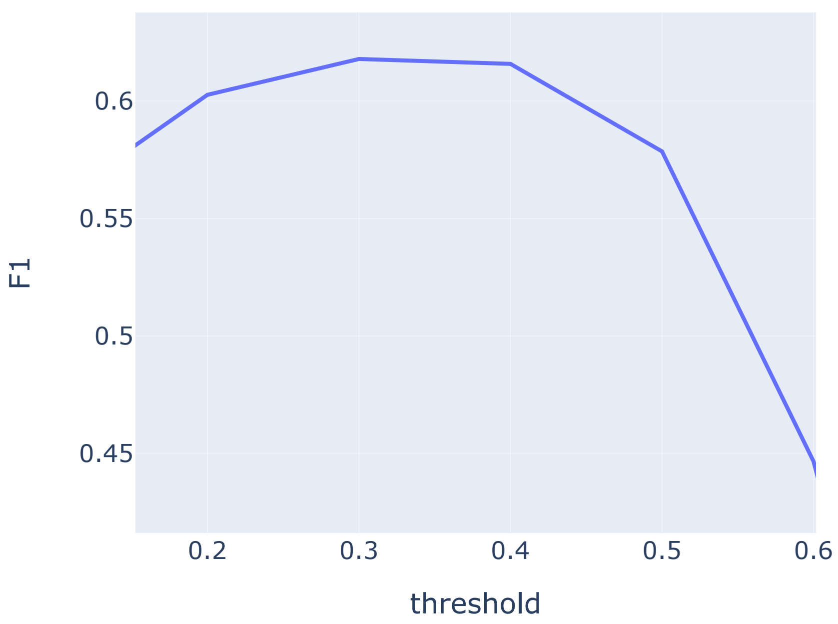
	threshold	TP	TN	FP	FN	Misclassification_rate	F1
0	0.1	1771	2448	98	2715	0.400028	0.557356

	threshold	TP	TN	FP	FN	Misclassification_rate	F1
1	0.2	1643	3222	226	1941	0.308163	0.602604
2	0.3	1447	3795	422	1368	0.254551	0.617848
3	0.4	1254	4213	615	950	0.222554	0.615762
4	0.5	978	4629	891	534	0.202645	0.578527
5	0.6	602	4938	1267	225	0.212173	0.446588
6	0.7	215	5115	1654	48	0.242036	0.201689
7	0.8	21	5160	1848	3	0.263225	0.022187

In [25]:

```
import plotly.express as px

fig = px.line(F1_by_threshold, x="threshold", y="F1",width=500, height=400)
fig.show()
```



In [26]:

```python
F1_by_threshold.sort_values("F1",ascending=False).iloc[0]
# so we choose threshold of 0.3 which F1 is 0.617848
# Accorrding to this F1, misclassification rate is 0.254551
```

Out[26]:

```
threshold                        0.300000
TP                            1447.000000
TN                            3795.000000
FP                             422.000000
FN                            1368.000000
Misclassification_rate           0.254551
F1                               0.617848
Name: 2, dtype: float64
```