Telecom Churn Case Study

- With 21 predictor variables we need to predict whether a particular customer will switch to another telecom provider or not. In telecom terminology, this is referred to as churning and not churning, respectively.

Step 1: Importing and Merging Data

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

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

In [2]:
```
# Importing all datasets
churn_data = pd.read_csv("churn_data.csv")
customer_data = pd.read_csv("customer_data.csv")
internet_data = pd.read_csv("internet_data.csv")
```

Combining all data files into one consolidated dataframe

In [3]:
```
# Merge datasets on 'customerID'
df_1 = pd.merge(churn_data, customer_data, how='inner', on='customerID')
telecom = pd.merge(df_1, internet_data, how='inner', on='customerID')
```

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

Out[4]:

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges
0	7590-VHVEG	1	No	Month-to-month	Yes	Electronic check	29.85
1	5575-GNVDE	34	Yes	One year	No	Mailed check	56.95
2	3668-QPYBK	2	Yes	Month-to-month	Yes	Mailed check	53.85
3	7795-CFOCW	45	No	One year	No	Bank transfer (automatic)	42.30
4	9237-HQITU	2	Yes	Month-to-month	Yes	Electronic check	70.70

5 rows × 21 columns

Step 2: Data Cleaning and Transformation

In [5]:
```
# Converting some binary variables (Yes/No) to 0/1
binary_vars =  ['PhoneService', 'PaperlessBilling', 'Churn', 'Partner', 'Dependents'
def binary_map(x):
    return x.map({'Yes': 1, "No": 0})
telecom[binary_vars] = telecom[binary_vars].apply(binary_map)
```

In [6]:
```
telecom.head()
```

Out[6]:

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges
0	7590-VHVEG	1	0	Month-to-month	1	Electronic check	29.85
1	5575-GNVDE	34	1	One year	0	Mailed check	56.95
2	3668-QPYBK	2	1	Month-to-month	1	Mailed check	53.85
3	7795-CFOCW	45	0	One year	0	Bank transfer (automatic)	42.30
4	9237-HQITU	2	1	Month-to-month	1	Electronic check	70.70

5 rows × 21 columns

Dummy Variable Creation

In [7]:
```
# Creating a dummy variable for some of the categorical variables and dropping the f
dummy1 = pd.get_dummies(telecom[['Contract', 'PaymentMethod', 'gender', 'InternetSer

# Adding the results to the master dataframe
telecom = pd.concat([telecom, dummy1], axis=1)
```

In [8]:
```
telecom.head()
```

Out[8]:

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges
0	7590-VHVEG	1	0	Month-to-month	1	Electronic check	29.85
1	5575-GNVDE	34	1	One year	0	Mailed check	56.95
2	3668-QPYBK	2	1	Month-to-month	1	Mailed check	53.85
3	7795-CFOCW	45	0	One year	0	Bank transfer (automatic)	42.30
4	9237-HQITU	2	1	Month-to-month	1	Electronic check	70.70

5 rows × 29 columns

In [9]:
```
# Creating dummy variables for the remaining categorical variables and dropping the

# Creating dummy variables for the variable 'MultipleLines'
ml = pd.get_dummies(telecom['MultipleLines'], prefix='MultipleLines')
# Dropping MultipleLines_No phone service column
ml1 = ml.drop(['MultipleLines_No phone service'], 1)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,ml1], axis=1)

os = pd.get_dummies(telecom['OnlineSecurity'], prefix='OnlineSecurity')
os1 = os.drop(['OnlineSecurity_No internet service'], 1)
telecom = pd.concat([telecom,os1], axis=1)

ob = pd.get_dummies(telecom['OnlineBackup'], prefix='OnlineBackup')
ob1 = ob.drop(['OnlineBackup_No internet service'], 1)
telecom = pd.concat([telecom,ob1], axis=1)

dp = pd.get_dummies(telecom['DeviceProtection'], prefix='DeviceProtection')
dp1 = dp.drop(['DeviceProtection_No internet service'], 1)
telecom = pd.concat([telecom,dp1], axis=1)

ts = pd.get_dummies(telecom['TechSupport'], prefix='TechSupport')
ts1 = ts.drop(['TechSupport_No internet service'], 1)
telecom = pd.concat([telecom,ts1], axis=1)

st =pd.get_dummies(telecom['StreamingTV'], prefix='StreamingTV')
st1 = st.drop(['StreamingTV_No internet service'], 1)
telecom = pd.concat([telecom,st1], axis=1)

smo = pd.get_dummies(telecom['StreamingMovies'], prefix='StreamingMovies')
smo1 = smo.drop(['StreamingMovies_No internet service'], 1)
telecom = pd.concat([telecom,smo1], axis=1)
```

In [10]:
```
telecom.head()
```

Out[10]:

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges
0	7590-VHVEG	1	0	Month-to-month	1	Electronic check	29.85
1	5575-GNVDE	34	1	One year	0	Mailed check	56.95
2	3668-QPYBK	2	1	Month-to-month	1	Mailed check	53.85
3	7795-CFOCW	45	0	One year	0	Bank transfer (automatic)	42.30
4	9237-HQITU	2	1	Month-to-month	1	Electronic check	70.70

5 rows × 43 columns

Dropping the repeated variables

In [11]:
```
# We have created dummies for the below variables, so we can drop them
telecom = telecom.drop(['Contract','PaymentMethod','gender','MultipleLines','Interne
       'TechSupport', 'StreamingTV', 'StreamingMovies'], 1)
```

In [12]:
```
telecom = telecom[~(telecom.TotalCharges == " ")]
telecom.TotalCharges = telecom.TotalCharges.astype(float, errors="ignore")
```

In [13]:
```
print(telecom['TotalCharges'].dtypes)
```

```
float64
```

Checking for Outliers

In [14]:
```
# Checking for outliers in the continuous variables at 25%, 50%, 75%, 90%, 95% and 9
num_telecom = telecom[['tenure','MonthlyCharges','SeniorCitizen','TotalCharges']]
num_telecom.describe(percentiles=[.25, .5, .75, .90, .95, .99])
```

Out[14]:

	tenure	MonthlyCharges	SeniorCitizen	TotalCharges
count	7032.000000	7032.000000	7032.000000	7032.000000
mean	32.421786	64.798208	0.162400	2283.300441
std	24.545260	30.085974	0.368844	2266.771362
min	1.000000	18.250000	0.000000	18.800000
25%	9.000000	35.587500	0.000000	401.450000
50%	29.000000	70.350000	0.000000	1397.475000
75%	55.000000	89.862500	0.000000	3794.737500
90%	69.000000	102.645000	1.000000	5976.640000
95%	72.000000	107.422500	1.000000	6923.590000
99%	72.000000	114.734500	1.000000	8039.883000
max	72.000000	118.750000	1.000000	8684.800000

Checking for Missing Values and Inputing Them

In [15]:
```
# Adding up the missing values (column-wise)
telecom.isnull().sum()
```

Out[15]:
```
customerID                               0
tenure                                   0
PhoneService                             0
PaperlessBilling                         0
MonthlyCharges                           0
TotalCharges                             0
Churn                                    0
SeniorCitizen                            0
Partner                                  0
Dependents                               0
Contract_One year                        0
Contract_Two year                        0
PaymentMethod_Credit card (automatic)    0
PaymentMethod_Electronic check           0
PaymentMethod_Mailed check               0
gender_Male                              0
InternetService_Fiber optic              0
InternetService_No                       0
MultipleLines_No                         0
```

```
MultipleLines_Yes                    0
OnlineSecurity_No                    0
OnlineSecurity_Yes                   0
OnlineBackup_No                      0
OnlineBackup_Yes                     0
DeviceProtection_No                  0
DeviceProtection_Yes                 0
TechSupport_No                       0
TechSupport_Yes                      0
StreamingTV_No                       0
StreamingTV_Yes                      0
StreamingMovies_No                   0
StreamingMovies_Yes                  0
dtype: int64
```

Step 3: Test-Train Split

In [16]:
```python
from sklearn.model_selection import train_test_split

y = telecom['Churn']
X = telecom.drop(['Churn','customerID'], axis=1)
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=
```

Step 4: Feature Scaling

In [17]:
```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train[['tenure','MonthlyCharges','TotalCharges']] = scaler.fit_transform(X_train[[
X_test[['tenure', 'MonthlyCharges', 'TotalCharges']] = scaler.transform(X_test[['ten
```

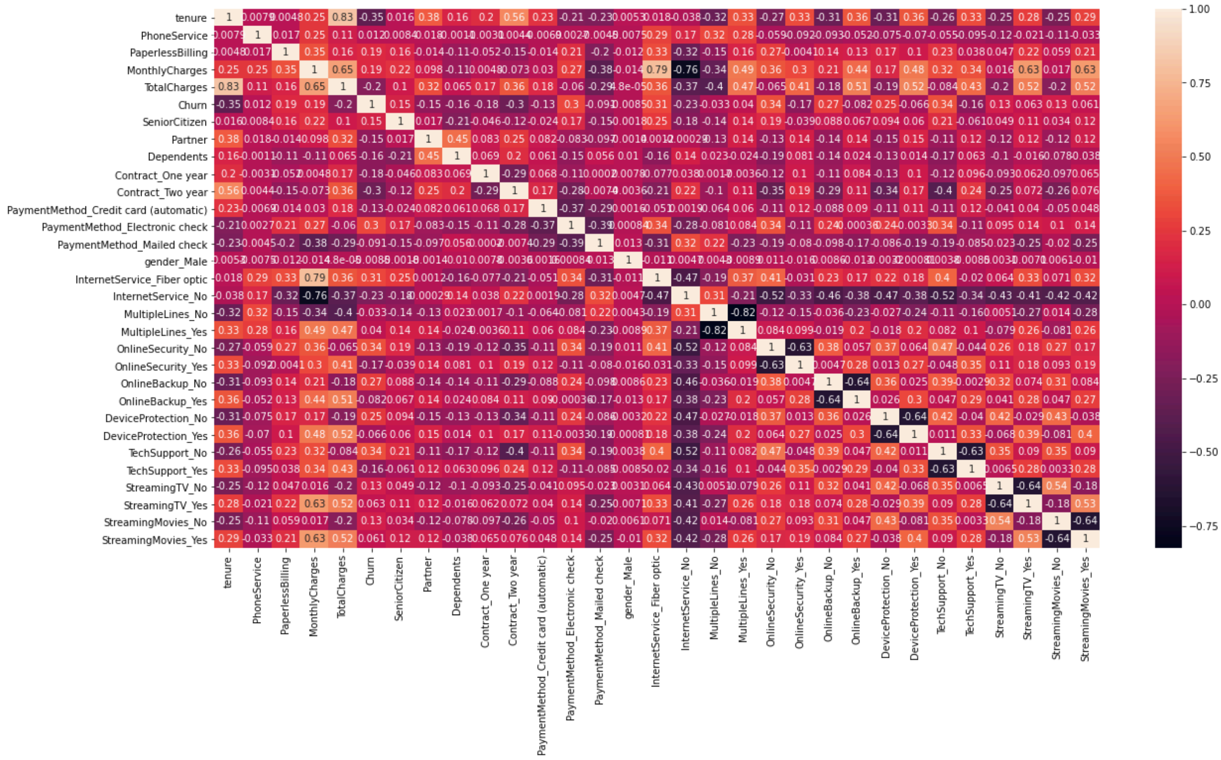
In [18]:
```python
### Checking the Churn Rate
churn = (sum(telecom['Churn'])/len(telecom['Churn'].index))*100
churn
```

Out[18]:
```
26.578498293515356
```

- "Our data showed a 27% churn rate, indicating moderate class imbalance. Traditional accuracy would be misleading here, so we focused on recall, precision, and ROC-AUC to build a more reliable model. We adjusted the cutoff threshold to 0.3 to prioritize detecting churners, ensuring business actions are directed at high-risk customers for proactive retention.

Step 5: Looking at Correlations

In [19]:
```python
plt.figure(figsize = (20,10))        # Size of the figure
sns.heatmap(telecom.corr(),annot = True)
plt.show()
```

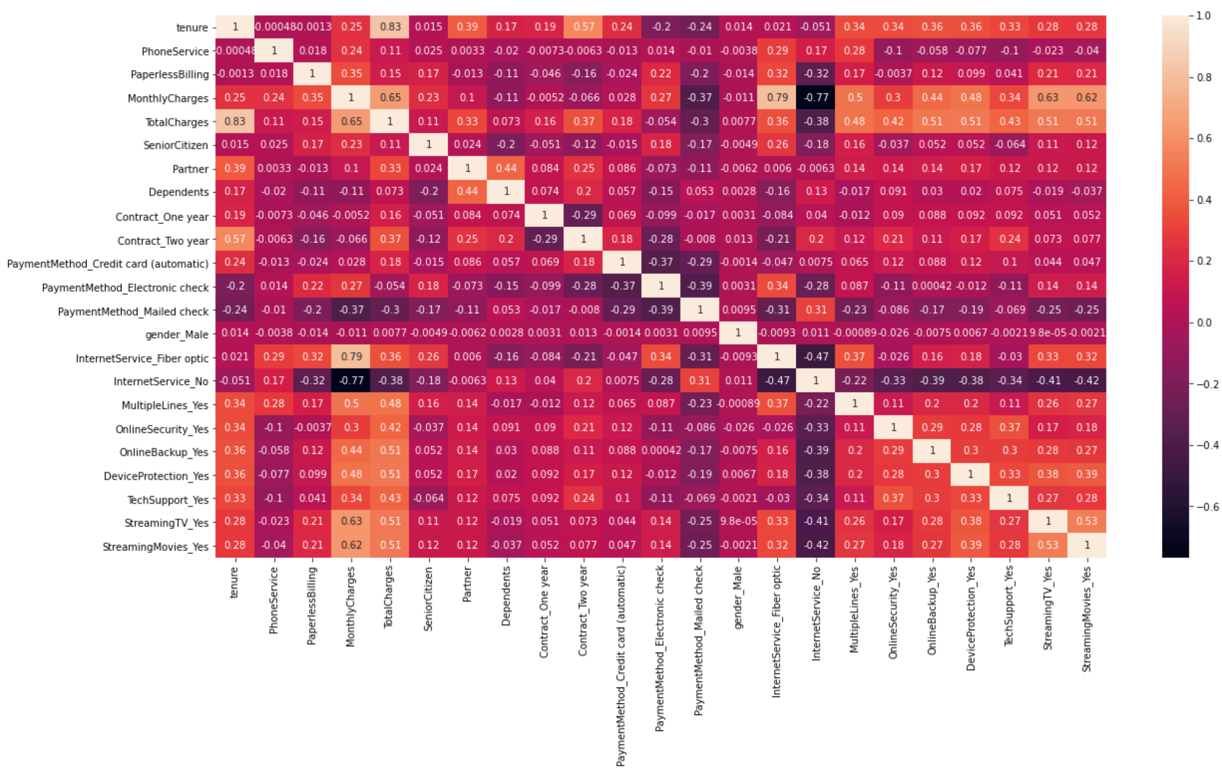


Dropping highly correlated dummy variables

In [20]:
```
X_test = X_test.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','Devi
                      'StreamingTV_No','StreamingMovies_No'], 1)
X_train = X_train.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','De
                        'StreamingTV_No','StreamingMovies_No'], 1)
```

- Checking the Correlation Matrix After dropping highly correlated variables now let's check the correlation matrix again.

In [21]:
```
plt.figure(figsize = (20,10))
sns.heatmap(X_train.corr(),annot = True)
plt.show()
```



Step 6: Model Building

Building a GLM (statsmodels) on full data

In [22]:
```
import statsmodels.api as sm

#Generalized Linear Model for logistic regression.
logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

Out[22]:

Generalized Linear Model Regression Results

Dep. Variable:	Churn	**No. Observations:**	4922
Model:	GLM	**Df Residuals:**	4898
Model Family:	Binomial	**Df Model:**	23
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-2004.7
Date:	Sat, 05 Jul 2025	**Deviance:**	4009.4
Time:	16:02:57	**Pearson chi2:**	6.07e+03
No. Iterations:	7		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.9382	1.546	-2.547	0.011	-6.969	-0.908
tenure	-1.5172	0.189	-8.015	0.000	-1.888	-1.146
PhoneService	0.9507	0.789	1.205	0.228	-0.595	2.497
PaperlessBilling	0.3254	0.090	3.614	0.000	0.149	0.502
MonthlyCharges	-2.1806	1.160	-1.880	0.060	-4.454	0.092
TotalCharges	0.7332	0.198	3.705	0.000	0.345	1.121
SeniorCitizen	0.3984	0.102	3.924	0.000	0.199	0.597
Partner	0.0374	0.094	0.399	0.690	-0.146	0.221
Dependents	-0.1430	0.107	-1.332	0.183	-0.353	0.067
Contract_One year	-0.6578	0.129	-5.106	0.000	-0.910	-0.405
Contract_Two year	-1.2455	0.212	-5.874	0.000	-1.661	-0.830
PaymentMethod_Credit card (automatic)	-0.2577	0.137	-1.883	0.060	-0.526	0.011
PaymentMethod_Electronic check	0.1615	0.113	1.434	0.152	-0.059	0.382
PaymentMethod_Mailed check	-0.2536	0.137	-1.845	0.065	-0.523	0.016
gender_Male	-0.0346	0.078	-0.442	0.658	-0.188	0.119
InternetService_Fiber optic	2.5124	0.967	2.599	0.009	0.618	4.407
InternetService_No	-2.7792	0.982	-2.831	0.005	-4.703	-0.855
MultipleLines_Yes	0.5623	0.214	2.628	0.009	0.143	0.982
OnlineSecurity_Yes	-0.0245	0.216	-0.113	0.910	-0.448	0.399

OnlineBackup_Yes	0.1740	0.212	0.822	0.411	-0.241	0.589
DeviceProtection_Yes	0.3229	0.215	1.501	0.133	-0.099	0.744
TechSupport_Yes	-0.0305	0.216	-0.141	0.888	-0.455	0.394
StreamingTV_Yes	0.9598	0.396	2.423	0.015	0.183	1.736
StreamingMovies_Yes	0.8484	0.396	2.143	0.032	0.072	1.624

Feature Selection Using RFE

In [23]:
```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
```

In [24]:
```
from sklearn.feature_selection import RFE

rfe = RFE(logreg, 15)             # running RFE with 15 variables as output
rfe = rfe.fit(X_train, y_train)
```

In [25]:
```
rfe.support_ # it returns a boolean array indicating which features were selected by
```

Out[25]:
```
array([ True, False,  True,  True,  True,  True, False, False,  True,
        True,  True, False,  True, False,  True,  True,  True, False,
       False, False,  True,  True,  True])
```

In [26]:
```
col = X_train.columns[rfe.support_]
```

In [27]:
```
X_train.columns[~rfe.support_]
```

Out[27]:
```
Index(['PhoneService', 'Partner', 'Dependents',
       'PaymentMethod_Electronic check', 'gender_Male', 'OnlineSecurity_Yes',
       'OnlineBackup_Yes', 'DeviceProtection_Yes'],
      dtype='object')
```

Building GLM using only the RFE-selected features

In [28]:
```
X_train_sm = sm.add_constant(X_train[col])
logm2 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
res = logm2.fit()
res.summary()
```

Out[28]:

Generalized Linear Model Regression Results

Dep. Variable:	Churn	**No. Observations:**	4922
Model:	GLM	**Df Residuals:**	4906
Model Family:	Binomial	**Df Model:**	15
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-2011.1
Date:	Sat, 05 Jul 2025	**Deviance:**	4022.2
Time:	16:02:59	**Pearson chi2:**	6.25e+03
No. Iterations:	7		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]

const	-2.2462	0.189	-11.879	0.000	-2.617	-1.876
tenure	-1.5596	0.187	-8.334	0.000	-1.926	-1.193
PaperlessBilling	0.3436	0.090	3.832	0.000	0.168	0.519
MonthlyCharges	-0.9692	0.199	-4.878	0.000	-1.359	-0.580
TotalCharges	0.7421	0.197	3.764	0.000	0.356	1.128
SeniorCitizen	0.4296	0.100	4.312	0.000	0.234	0.625
Contract_One year	-0.6830	0.128	-5.342	0.000	-0.934	-0.432
Contract_Two year	-1.2931	0.211	-6.138	0.000	-1.706	-0.880
PaymentMethod_Credit card (automatic)	-0.3724	0.113	-3.308	0.001	-0.593	-0.152
PaymentMethod_Mailed check	-0.3723	0.111	-3.345	0.001	-0.591	-0.154
InternetService_Fiber optic	1.5865	0.216	7.342	0.000	1.163	2.010
InternetService_No	-1.6897	0.216	-7.830	0.000	-2.113	-1.267
MultipleLines_Yes	0.3779	0.104	3.640	0.000	0.174	0.581
TechSupport_Yes	-0.2408	0.109	-2.210	0.027	-0.454	-0.027
StreamingTV_Yes	0.5796	0.114	5.102	0.000	0.357	0.802
StreamingMovies_Yes	0.4665	0.111	4.197	0.000	0.249	0.684

Calculating VIF to check for multicollinearity among the RFE-selected features

In [29]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
vif['Features'] = X_train[col].columns
vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_tra
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[29]:

	Features	VIF
2	MonthlyCharges	14.85
3	TotalCharges	10.42
0	tenure	7.38
9	InternetService_Fiber optic	5.61
10	InternetService_No	5.27
6	Contract_Two year	3.14
13	StreamingTV_Yes	2.79
14	StreamingMovies_Yes	2.79
1	PaperlessBilling	2.76
11	MultipleLines_Yes	2.38
12	TechSupport_Yes	1.95
5	Contract_One year	1.85

	Features	VIF
8	PaymentMethod_Mailed check	1.73
7	PaymentMethod_Credit card (automatic)	1.45
4	SeniorCitizen	1.33

In [30]:
```
col = col.drop('TotalCharges', 1)
col
```

Out[30]:
```
Index(['tenure', 'PaperlessBilling', 'MonthlyCharges', 'SeniorCitizen',
       'Contract_One year', 'Contract_Two year',
       'PaymentMethod_Credit card (automatic)', 'PaymentMethod_Mailed check',
       'InternetService_Fiber optic', 'InternetService_No',
       'MultipleLines_Yes', 'TechSupport_Yes', 'StreamingTV_Yes',
       'StreamingMovies_Yes'],
      dtype='object')
```

In [31]:
```
# Let's re-run the model using the selected variables
X_train_sm = sm.add_constant(X_train[col])
logm3 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
res = logm3.fit()
res.summary()
```

Out[31]:

Generalized Linear Model Regression Results

Dep. Variable:	Churn	**No. Observations:**	4922
Model:	GLM	**Df Residuals:**	4907
Model Family:	Binomial	**Df Model:**	14
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-2018.5
Date:	Sat, 05 Jul 2025	**Deviance:**	4037.1
Time:	16:03:01	**Pearson chi2:**	5.25e+03
No. Iterations:	7		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-2.1697	0.186	-11.663	0.000	-2.534	-1.805
tenure	-0.9137	0.065	-13.982	0.000	-1.042	-0.786
PaperlessBilling	0.3332	0.089	3.726	0.000	0.158	0.508
MonthlyCharges	-0.7106	0.184	-3.854	0.000	-1.072	-0.349
SeniorCitizen	0.4407	0.100	4.404	0.000	0.245	0.637
Contract_One year	-0.6821	0.127	-5.374	0.000	-0.931	-0.433
Contract_Two year	-1.2558	0.208	-6.034	0.000	-1.664	-0.848
PaymentMethod_Credit card (automatic)	-0.3774	0.113	-3.348	0.001	-0.598	-0.156
PaymentMethod_Mailed check	-0.3207	0.110	-2.917	0.004	-0.536	-0.105
InternetService_Fiber optic	1.5264	0.213	7.166	0.000	1.109	1.944
InternetService_No	-1.5165	0.208	-7.278	0.000	-1.925	-1.108
MultipleLines_Yes	0.3872	0.104	3.739	0.000	0.184	0.590

TechSupport_Yes	-0.2426	0.109	-2.224	0.026	-0.456	-0.029
StreamingTV_Yes	0.5779	0.113	5.126	0.000	0.357	0.799
StreamingMovies_Yes	0.4667	0.110	4.226	0.000	0.250	0.683

In [32]:
```
vif = pd.DataFrame()
vif['Features'] = X_train[col].columns
vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_tra
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[32]:

	Features	VIF
2	MonthlyCharges	10.63
8	InternetService_Fiber optic	5.44
9	InternetService_No	5.15
5	Contract_Two year	3.13
12	StreamingTV_Yes	2.79
13	StreamingMovies_Yes	2.79
1	PaperlessBilling	2.76
0	tenure	2.38
10	MultipleLines_Yes	2.38
11	TechSupport_Yes	1.94
4	Contract_One year	1.85
7	PaymentMethod_Mailed check	1.69
6	PaymentMethod_Credit card (automatic)	1.45
3	SeniorCitizen	1.33

In [33]:
```
col = col.drop('MonthlyCharges', 1)
col
```

Out[33]:
```
Index(['tenure', 'PaperlessBilling', 'SeniorCitizen', 'Contract_One year',
       'Contract_Two year', 'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Mailed check', 'InternetService_Fiber optic',
       'InternetService_No', 'MultipleLines_Yes', 'TechSupport_Yes',
       'StreamingTV_Yes', 'StreamingMovies_Yes'],
      dtype='object')
```

In [34]:
```
# Let's re-run the model using the selected variables
X_train_sm = sm.add_constant(X_train[col])
logm4 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
res = logm4.fit()
res.summary()
```

Out[34]:

Generalized Linear Model Regression Results

Dep. Variable:	Churn	**No. Observations:**	4922
Model:	GLM	**Df Residuals:**	4908
Model Family:	Binomial	**Df Model:**	13
Link Function:	logit	**Scale:**	1.0000

Method:	IRLS	**Log-Likelihood:**	-2025.9
Date:	Sat, 05 Jul 2025	**Deviance:**	4051.9
Time:	16:03:02	**Pearson chi2:**	5.25e+03
No. Iterations:	7		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-1.6577	0.127	-13.094	0.000	-1.906	-1.410
tenure	-0.9426	0.065	-14.480	0.000	-1.070	-0.815
PaperlessBilling	0.3455	0.089	3.877	0.000	0.171	0.520
SeniorCitizen	0.4597	0.100	4.613	0.000	0.264	0.655
Contract_One year	-0.7218	0.127	-5.702	0.000	-0.970	-0.474
Contract_Two year	-1.2987	0.208	-6.237	0.000	-1.707	-0.891
PaymentMethod_Credit card (automatic)	-0.3874	0.113	-3.442	0.001	-0.608	-0.167
PaymentMethod_Mailed check	-0.3307	0.110	-3.020	0.003	-0.545	-0.116
InternetService_Fiber optic	0.8052	0.097	8.272	0.000	0.614	0.996
InternetService_No	-0.9726	0.155	-6.261	0.000	-1.277	-0.668
MultipleLines_Yes	0.2097	0.092	2.279	0.023	0.029	0.390
TechSupport_Yes	-0.4046	0.101	-4.019	0.000	-0.602	-0.207
StreamingTV_Yes	0.3390	0.094	3.619	0.000	0.155	0.523
StreamingMovies_Yes	0.2428	0.093	2.598	0.009	0.060	0.426

In [35]:

```
vif = pd.DataFrame()
vif['Features'] = X_train[col].columns
vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_tra
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[35]:

	Features	VIF
4	Contract_Two year	2.98
7	InternetService_Fiber optic	2.67
12	StreamingMovies_Yes	2.54
11	StreamingTV_Yes	2.51
1	PaperlessBilling	2.45
9	MultipleLines_Yes	2.24
0	tenure	2.04
8	InternetService_No	2.03
10	TechSupport_Yes	1.92
3	Contract_One year	1.78
6	PaymentMethod_Mailed check	1.63

	Features	VIF
5	PaymentMethod_Credit card (automatic)	1.42
2	SeniorCitizen	1.31

Step 7: Evaluation of model on train set

In [36]:
```
y_train_pred = res.predict(X_train_sm).values.reshape(-1)
```

In [37]:
```
y_train_pred_final = pd.DataFrame({'Churn': y_train.values, 'Churn_Prob': y_train_pr
y_train_pred_final['CustID'] = y_train.index
y_train_pred_final.head()
```

Out[37]:

	Churn	Churn_Prob	CustID
0	0	0.245817	879
1	0	0.265361	5790
2	1	0.669410	6498
3	1	0.630970	880
4	1	0.682916	2784

In [38]:
```
y_train_pred_final['predicted'] = y_train_pred_final.Churn_Prob.map(lambda x: 1 if x
y_train_pred_final.head()
```

Out[38]:

	Churn	Churn_Prob	CustID	predicted
0	0	0.245817	879	0
1	0	0.265361	5790	0
2	1	0.669410	6498	1
3	1	0.630970	880	1
4	1	0.682916	2784	1

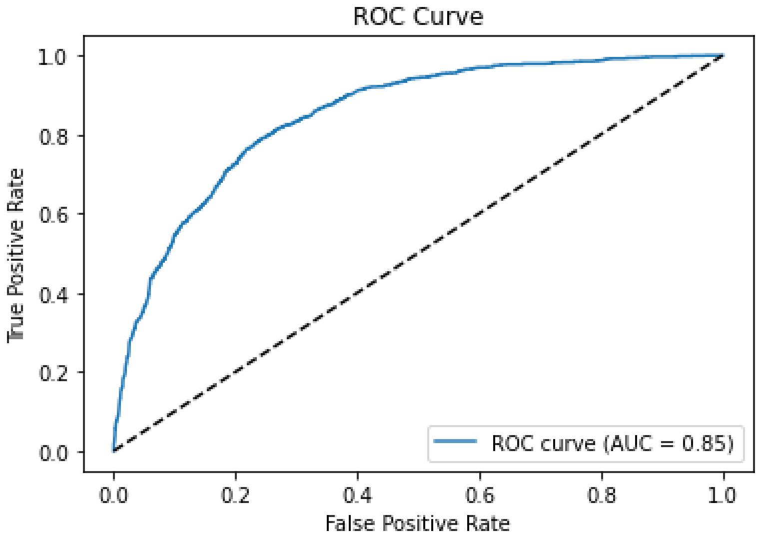
In [39]:
```
from sklearn import metrics

confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.pr
accuracy = metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.predi
print(confusion)
print("Accuracy:", accuracy)
```

```
[[3278  357]
 [ 597  690]]
Accuracy: 0.8061763510767981
```

In [40]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
recall = recall_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
f1 = f1_score(y_train_pred_final.Churn, y_train_pred_final.predicted)

print("1. Precision:", precision)
print("2. Recall:", recall)
print("3. F1-Score:", f1)
```

```
1. Precision: 0.6590257879656161
2. Recall: 0.5361305361305362
3. F1-Score: 0.5912596401028278
```

In [41]:
```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_train_pred_final.Churn, y_train_pred_final.Churn_
auc_score = roc_auc_score(y_train_pred_final.Churn, y_train_pred_final.Churn_Prob)

plt.plot(fpr, tpr, label='ROC curve (AUC = %0.2f)' % auc_score)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



Finding Optimal Cutoff Point

In [42]:
```
# Let's create columns with different probability cutoffs
numbers = [float(x)/10 for x in range(10)]
for i in numbers:
    y_train_pred_final[i]= y_train_pred_final.Churn_Prob.map(lambda x: 1 if x > i el
y_train_pred_final.head()
```

Out[42]:

	Churn	Churn_Prob	CustID	predicted	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	0	0.245817	879	0	1	1	1	0	0	0	0	0	0	0
1	0	0.265361	5790	0	1	1	1	0	0	0	0	0	0	0
2	1	0.669410	6498	1	1	1	1	1	1	1	1	0	0	0
3	1	0.630970	880	1	1	1	1	1	1	1	1	0	0	0
4	1	0.682916	2784	1	1	1	1	1	1	1	1	0	0	0

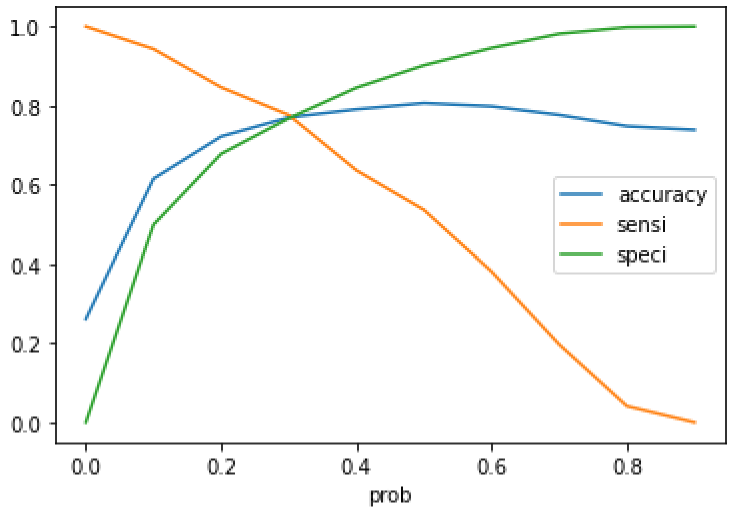
In [43]:
```
# Now let's calculate accuracy sensitivity and specificity for various probability c
cutoff_df = pd.DataFrame( columns = ['prob','accuracy','sensi','speci'])
from sklearn.metrics import confusion_matrix


num = [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
for i in num:
```

```
    cm1 = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final[i] )
    total1=sum(sum(cm1))
    accuracy = (cm1[0,0]+cm1[1,1])/total1

    speci = cm1[0,0]/(cm1[0,0]+cm1[0,1])
    sensi = cm1[1,1]/(cm1[1,0]+cm1[1,1])
    cutoff_df.loc[i] =[ i ,accuracy,sensi,speci]
print(cutoff_df)
```

```
     prob  accuracy     sensi     speci
0.0   0.0  0.261479  1.000000  0.000000
0.1   0.1  0.614994  0.943279  0.498762
0.2   0.2  0.721861  0.846154  0.677854
0.3   0.3  0.770012  0.776224  0.767813
0.4   0.4  0.790532  0.636364  0.845117
0.5   0.5  0.806176  0.536131  0.901788
0.6   0.6  0.798050  0.380730  0.945805
0.7   0.7  0.776310  0.196581  0.981568
0.8   0.8  0.747867  0.041181  0.998074
0.9   0.9  0.738521  0.000000  1.000000
```

In [44]:

```
# Let's plot accuracy sensitivity and specificity for various probabilities.
cutoff_df.plot.line(x='prob', y=['accuracy','sensi','speci'])
plt.show()
```



From the curve above, 0.3 is the optimum point to take it as a cutoff probability.

In [45]:

```
y_train_pred_final['final_predicted'] = y_train_pred_final.Churn_Prob.map( lambda x:
y_train_pred_final.head()
```

Out[45]:

	Churn	Churn_Prob	CustID	predicted	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	final_predi
0	0	0.245817	879	0	1	1	1	0	0	0	0	0	0	0	
1	0	0.265361	5790	0	1	1	1	0	0	0	0	0	0	0	
2	1	0.669410	6498	1	1	1	1	1	1	1	1	0	0	0	
3	1	0.630970	880	1	1	1	1	1	1	1	1	0	0	0	
4	1	0.682916	2784	1	1	1	1	1	1	1	1	0	0	0	

In [46]:

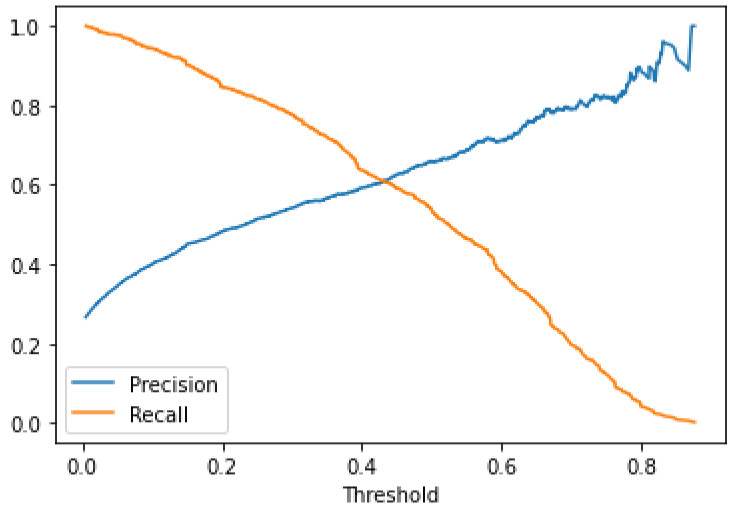
```
# Confusion matrix
metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.final_predicte
```

Out[46]:
```
array([[2791,  844],
       [ 288,  999]], dtype=int64)
```

In [47]:
```
print(confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.final_predicted)
print(accuracy_score(y_train_pred_final.Churn, y_train_pred_final.final_predicted))
print(precision_score(y_train_pred_final.Churn, y_train_pred_final.final_predicted))
print(recall_score(y_train_pred_final.Churn, y_train_pred_final.final_predicted))
print(f1_score(y_train_pred_final.Churn, y_train_pred_final.final_predicted))
```

```
[[2791  844]
 [ 288  999]]

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-47-3fca15a55fb2> in <module>
      1 print(confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.final_pre
dicted))
----> 2 print(accuracy_score(y_train_pred_final.Churn, y_train_pred_final.final_predi
cted))
      3 print(precision_score(y_train_pred_final.Churn, y_train_pred_final.final_pred
icted))
      4 print(recall_score(y_train_pred_final.Churn, y_train_pred_final.final_predict
ed))
      5 print(f1_score(y_train_pred_final.Churn, y_train_pred_final.final_predicted))

NameError: name 'accuracy_score' is not defined
```

In [48]:
```
from sklearn.metrics import precision_recall_curve

p, r, thresholds = precision_recall_curve(y_train_pred_final.Churn, y_train_pred_fin
plt.plot(thresholds, p[:-1], label="Precision")
plt.plot(thresholds, r[:-1], label="Recall")
plt.xlabel('Threshold')
plt.legend()
plt.show()
```



Step 8: Evaluation of model on Test set

In [49]:
```
# Scaling numeric columns
X_test[['tenure', 'MonthlyCharges', 'TotalCharges']] = scaler.transform(X_test[['ten

# Selecting final columns (same as col from training)
X_test = X_test[col]
X_test_sm = sm.add_constant(X_test)
```

In [50]:
```
y_test_pred = res.predict(X_test_sm)
```

In [51]:
```
y_test_df = pd.DataFrame(y_test)
y_test_df['CustID'] = y_test_df.index
y_test_df.head()
```

Out[51]:

	Churn	CustID
942	0	942
3730	1	3730
1761	0	1761
2283	1	2283
1872	0	1872

In [52]:
```
y_pred_1 = pd.DataFrame(y_test_pred)
y_pred_1.reset_index(drop=True, inplace=True)
y_test_df.reset_index(drop=True, inplace=True)
y_pred_1.head()
```

Out[52]:

	0
0	0.648571
1	0.752059
2	0.072410
3	0.613676
4	0.063482

In [53]:
```
y_pred_final = pd.concat([y_test_df, y_pred_1], axis=1)
y_pred_final = y_pred_final.rename(columns={0: 'Churn_Prob'})
y_pred_final.head()
```

Out[53]:

	Churn	CustID	Churn_Prob
0	0	942	0.648571
1	1	3730	0.752059
2	0	1761	0.072410
3	1	2283	0.613676
4	0	1872	0.063482

In [54]:
```
y_pred_final['final_predicted'] = y_pred_final.Churn_Prob.map(lambda x: 1 if x > 0.3
y_pred_final[['Churn', 'Churn_Prob', 'final_predicted']].head()
```

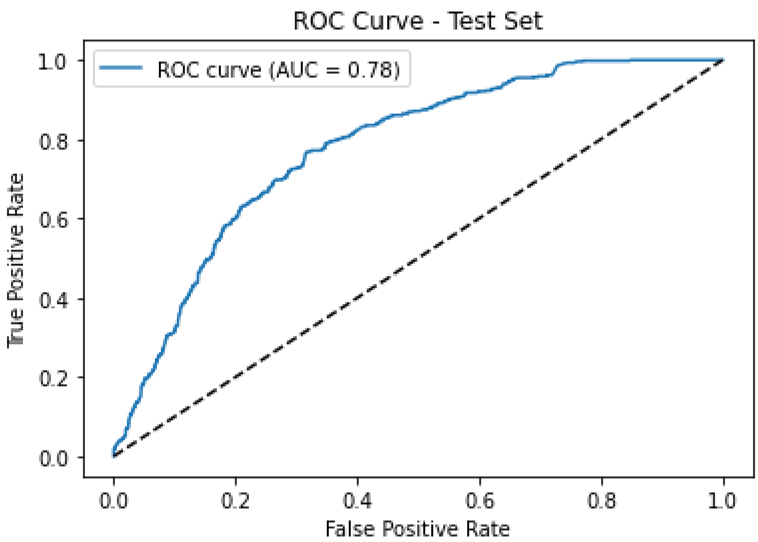
Out[54]:

	Churn	Churn_Prob	final_predicted
0	0	0.648571	1
1	1	0.752059	1
2	0	0.072410	0
3	1	0.613676	1
4	0	0.063482	0

In [55]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,

print("Confusion Matrix:\n", confusion_matrix(y_pred_final.Churn, y_pred_final.final
print("Accuracy:", accuracy_score(y_pred_final.Churn, y_pred_final.final_predicted))
print("Precision:", precision_score(y_pred_final.Churn, y_pred_final.final_predicted
print("Recall:", recall_score(y_pred_final.Churn, y_pred_final.final_predicted))
print("F1-Score:", f1_score(y_pred_final.Churn, y_pred_final.final_predicted))
print("AUC:", roc_auc_score(y_pred_final.Churn, y_pred_final.Churn_Prob))
```

```
Confusion Matrix:
 [[684 844]
 [ 57 525]]
Accuracy: 0.5729857819905213
Precision: 0.38349159970781593
Recall: 0.9020618556701031
F1-Score: 0.5381855458739108
AUC: 0.7781031287670247
```

In [56]:
```
fpr, tpr, thresholds = roc_curve(y_pred_final.Churn, y_pred_final.Churn_Prob)
plt.plot(fpr, tpr, label='ROC curve (AUC = %0.2f)' % roc_auc_score(y_pred_final.Chur
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Test Set')
plt.legend()
plt.show()
```



In [ ]:

In [ ]:

In [ ]: