Logistic Regression for Predictive Modeling

Data preparation

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_val_predict, train_test_split
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('/Users/ebeth/Desktop/Churn Data/churn_clean.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	CaseOrder	Customer_id	Interaction	UID	City	Sta
0	1	K409198	aa90260b-4141-4a24-8e36-b04ce1f4f77b	e885b299883d4f9fb18e39c75155d990	Point Baker	
1	2	S120509	fb76459f-c047-4a9d-8af9-e0f7d4ac2524	f2de8bef964785f41a2959829830fb8a	West Branch	
2	3	K191035	344d114c-3736-4be5-98f7-c72c281e2d35	f1784cfa9f6d92ae816197eb175d3c71	Yamhill	
3	4	D90850	abfa2b40-2d43-4994-b15a-989b8c79e311	dc8a365077241bb5cd5ccd305136b05e	Del Mar	
4	5	K662701	68a861fd-0d20-4e51-a587-8a90407ee574	aabb64a116e83fdc4befc1fbab1663f9	Needville	

5 rows × 50 columns

In [4]:
```
df.shape
```

Out[4]: (10000, 50)

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CaseOrder             10000 non-null  int64  
 1   Customer_id           10000 non-null  object 
 2   Interaction           10000 non-null  object 
 3   UID                   10000 non-null  object 
 4   City                  10000 non-null  object 
 5   State                 10000 non-null  object 
 6   County                10000 non-null  object 
 7   Zip                   10000 non-null  int64  
 8   Lat                   10000 non-null  float64
 9   Lng                   10000 non-null  float64
 10  Population            10000 non-null  int64  
 11  Area                  10000 non-null  object 
 12  TimeZone              10000 non-null  object 
 13  Job                   10000 non-null  object 
 14  Children              10000 non-null  int64  
 15  Age                   10000 non-null  int64  
 16  Income                10000 non-null  float64
 17  Marital               10000 non-null  object 
 18  Gender                10000 non-null  object 
 19  Churn                 10000 non-null  object 
 20  Outage_sec_perweek    10000 non-null  float64
 21  Email                 10000 non-null  int64  
 22  Contacts              10000 non-null  int64  
 23  Yearly_equip_failure  10000 non-null  int64  
 24  Techie                10000 non-null  object 
 25  Contract              10000 non-null  object 
 26  Port_modem            10000 non-null  object 
 27  Tablet                10000 non-null  object 
 28  InternetService       10000 non-null  object 
 29  Phone                 10000 non-null  object 
 30  Multiple              10000 non-null  object 
 31  OnlineSecurity        10000 non-null  object 
 32  OnlineBackup          10000 non-null  object 
 33  DeviceProtection      10000 non-null  object 
 34  TechSupport           10000 non-null  object 
 35  StreamingTV           10000 non-null  object 
 36  StreamingMovies       10000 non-null  object 
 37  PaperlessBilling      10000 non-null  object 
 38  PaymentMethod         10000 non-null  object 
 39  Tenure                10000 non-null  float64
 40  MonthlyCharge         10000 non-null  float64
 41  Bandwidth_GB_Year     10000 non-null  float64
 42  Item1                 10000 non-null  int64  
 43  Item2                 10000 non-null  int64  
 44  Item3                 10000 non-null  int64  
 45  Item4                 10000 non-null  int64  
 46  Item5                 10000 non-null  int64  
 47  Item6                 10000 non-null  int64  
 48  Item7                 10000 non-null  int64  
 49  Item8                 10000 non-null  int64  
dtypes: float64(7), int64(16), object(27)
memory usage: 3.8+ MB
```

In [6]:
```
df2 = df.drop(['CaseOrder','Customer_id','Interaction', 'UID','Lat','Lng','Ti
```

Take a closer look at state and zip code to see if they wi be included in the regression. Looking at the number of unique values will reveal if using these variables will make the regression equation unwieldy.

In [7]:
```
df2.nunique(axis = 0)
```

Out[7]:
```
State                      52
Zip                      8583
Population               5933
Area                        3
Children                   11
Age                        72
Income                   9993
Marital                     5
Gender                      3
Churn                       2
Outage_sec_perweek       9986
Email                      23
Contacts                    8
Yearly_equip_failure        6
Techie                      2
Contract                    3
Port_modem                  2
Tablet                      2
InternetService             3
Phone                       2
Multiple                    2
OnlineSecurity              2
OnlineBackup                2
DeviceProtection            2
TechSupport                 2
StreamingTV                 2
StreamingMovies             2
PaperlessBilling            2
PaymentMethod               4
Tenure                   9996
MonthlyCharge             750
Bandwidth_GB_Year       10000
Item1                       7
Item2                       7
Item3                       8
Item4                       7
Item5                       7
Item6                       8
Item7                       7
Item8                       8
dtype: int64
```

There are 52 states and 8583 unique zip codes in the data set. For this overall look at churn this many variables would not be helpful. Exploring individual states or zip codes at a later date might be beneficial. State and zip will be dropped from the data set.

In [8]:
```
df3 = df2.drop(['State', 'Zip'], axis = 1)
```

Look for missing values, duplicates, and outliers

In [9]:
```
df3.isnull().count()
```

Out[9]:
```
Population              10000
Area                    10000
Children                10000
Age                     10000
Income                  10000
Marital                 10000
Gender                  10000
Churn                   10000
Outage_sec_perweek      10000
Email                   10000
Contacts                10000
Yearly_equip_failure    10000
Techie                  10000
Contract                10000
Port_modem              10000
Tablet                  10000
InternetService         10000
Phone                   10000
Multiple                10000
OnlineSecurity          10000
OnlineBackup            10000
DeviceProtection        10000
TechSupport             10000
StreamingTV             10000
StreamingMovies         10000
PaperlessBilling        10000
PaymentMethod           10000
Tenure                  10000
MonthlyCharge           10000
Bandwidth_GB_Year       10000
Item1                   10000
Item2                   10000
Item3                   10000
Item4                   10000
Item5                   10000
Item6                   10000
Item7                   10000
Item8                   10000
dtype: int64
```

In [10]:
```
df3.duplicated()
```

Out[10]:
```
0       False
1       False
2       False
3       False
4       False
        ...
9995    False
9996    False
9997    False
9998    False
9999    False
Length: 10000, dtype: bool
```

In [11]:
```
df3.describe()
```

Out[11]:

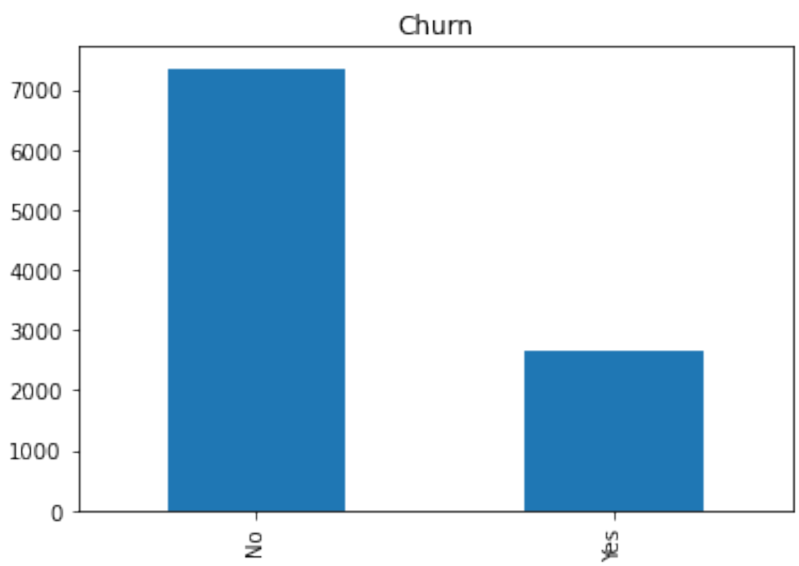
	Population	Children	Age	Income	Outage_sec_perweek	
count	10000.000000	10000.0000	10000.000000	10000.000000	10000.000000	10000.
mean	9756.562400	2.0877	53.078400	39806.926771	10.001848	12
std	14432.698671	2.1472	20.698882	28199.916702	2.976019	3.
min	0.000000	0.0000	18.000000	348.670000	0.099747	1.
25%	738.000000	0.0000	35.000000	19224.717500	8.018214	10.
50%	2910.500000	1.0000	53.000000	33170.605000	10.018560	12.
75%	13168.000000	3.0000	71.000000	53246.170000	11.969485	14.
max	111850.000000	10.0000	89.000000	258900.700000	21.207230	23.

Univariate Visualizations

Target variable Churn

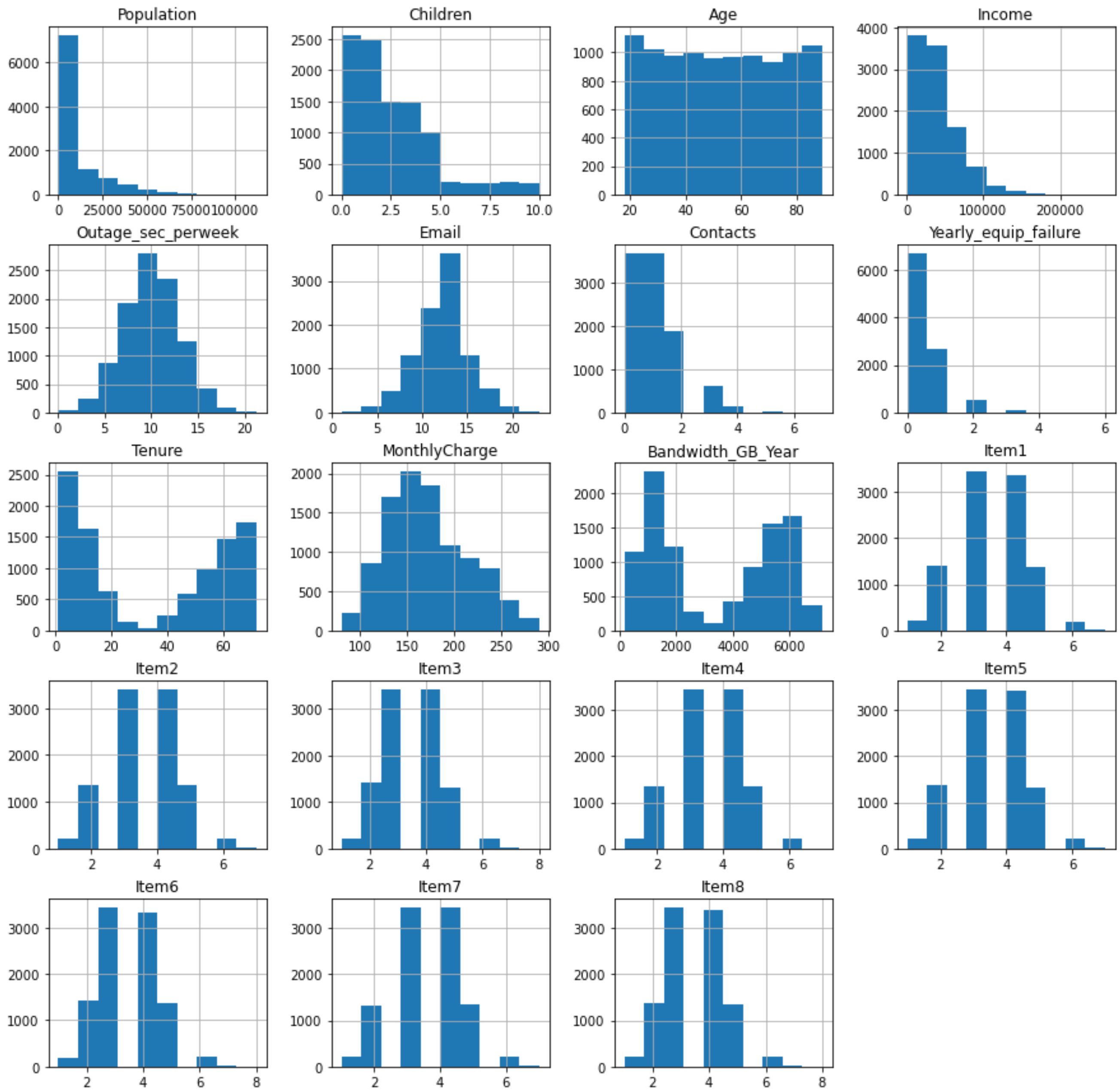
In [12]:
```
df3['Churn'].value_counts().plot.bar(title = 'Churn')
```

Out[12]: <AxesSubplot:title={'center':'Churn'}>

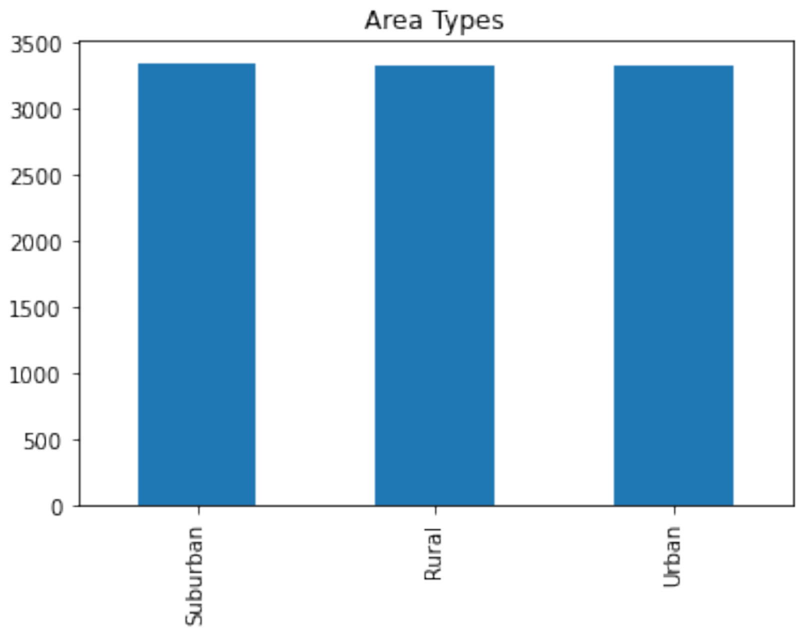


Predictor Variables

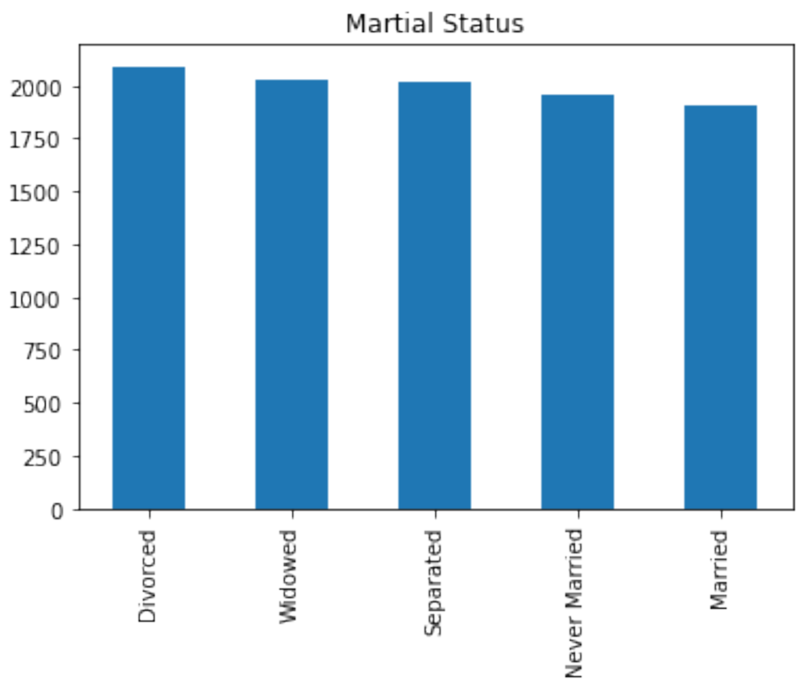
In [13]:
```
df3.hist(figsize = (15,15));
```



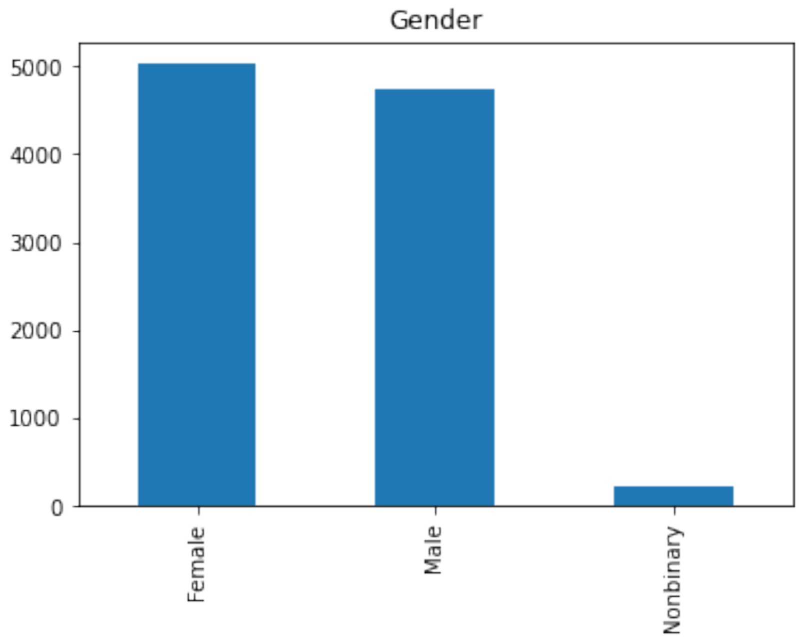
In [14]:
```
df3['Area'].value_counts().plot.bar(title = 'Area Types');
```



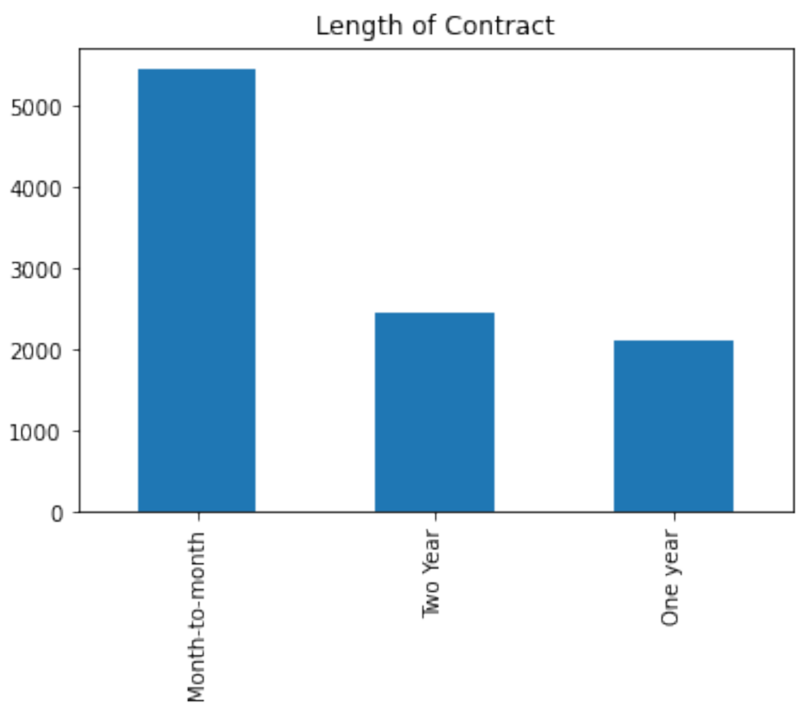
In [15]:
```
df3['Marital'].value_counts().plot.bar(title = 'Martial Status');
```



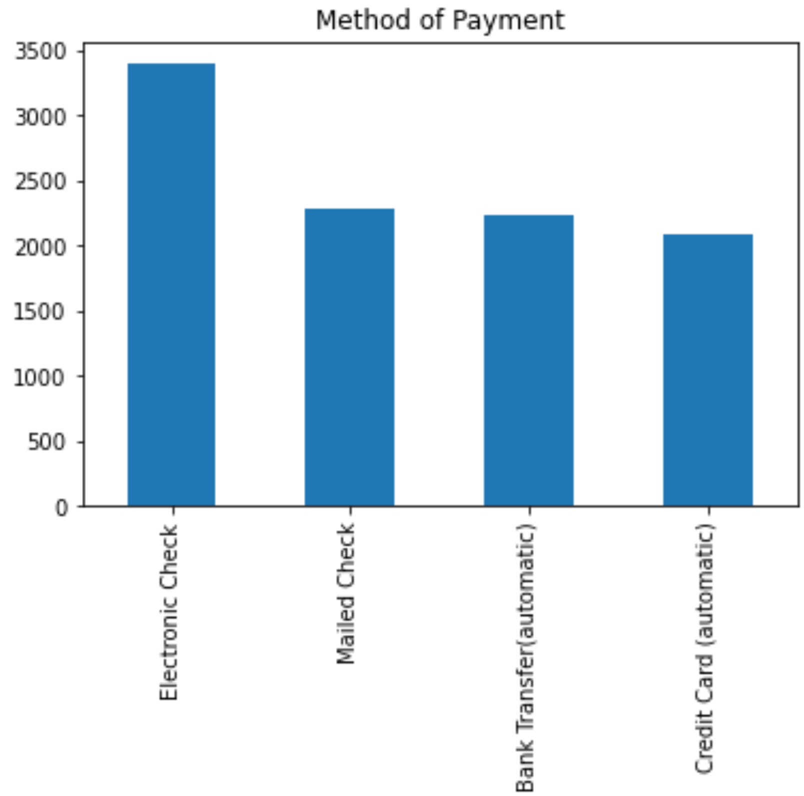
In [16]:
```
df3['Gender'].value_counts().plot.bar(title = 'Gender');
```



In [17]:
```
df3['Contract'].value_counts().plot.bar(title = 'Length of Contract');
```

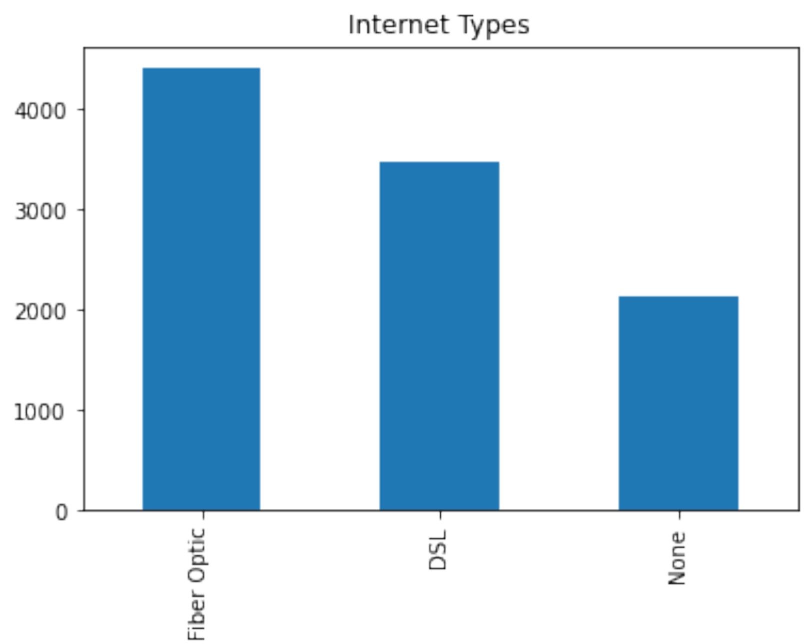


In [18]:
```
df3['PaymentMethod'].value_counts().plot.bar(title = 'Method of Payment');
```



In [19]:

```
df3['InternetService'].value_counts().plot.bar(title = 'Internet Types');
```

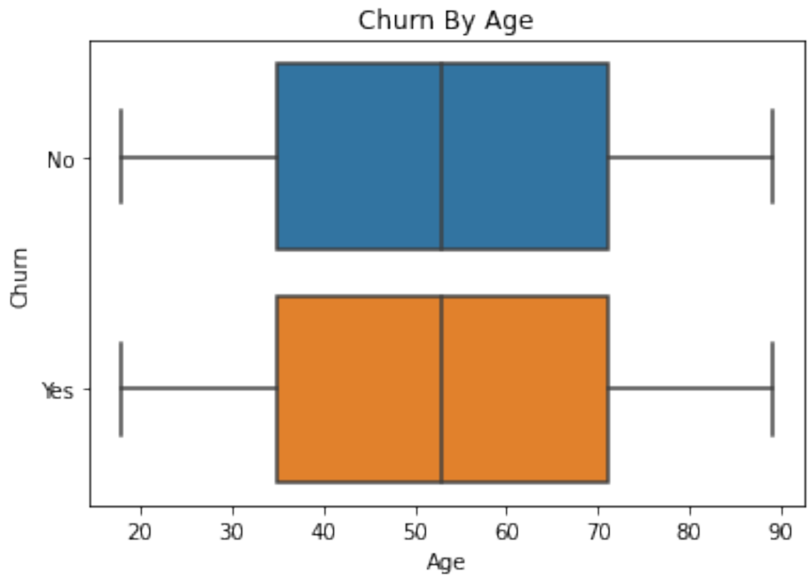


Bivariate Visualizations

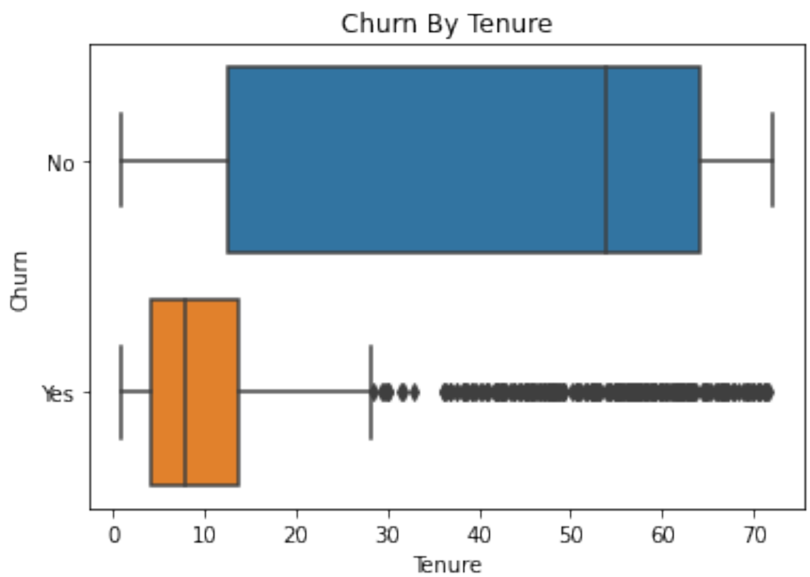
https://seaborn.pydata.org/tutorial/categorical.html

In [20]:

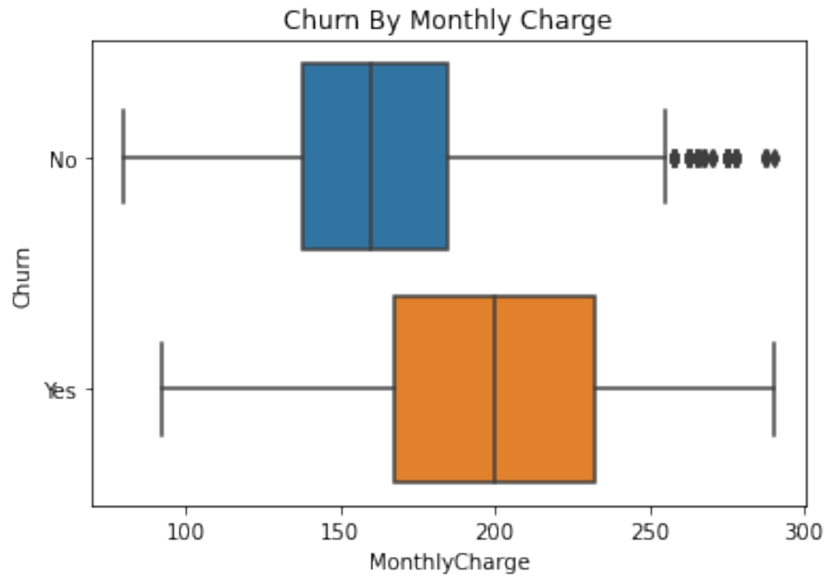
```
sn.boxplot(df3['Age'], df3['Churn']).set_title('Churn By Age');
```



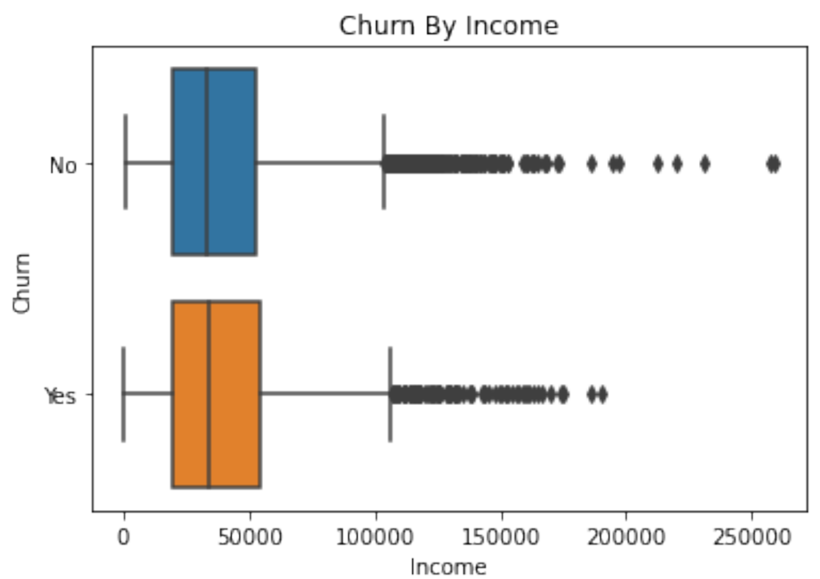
In [21]:
```
sn.boxplot(df3['Tenure'], df3['Churn']).set_title('Churn By Tenure');
```



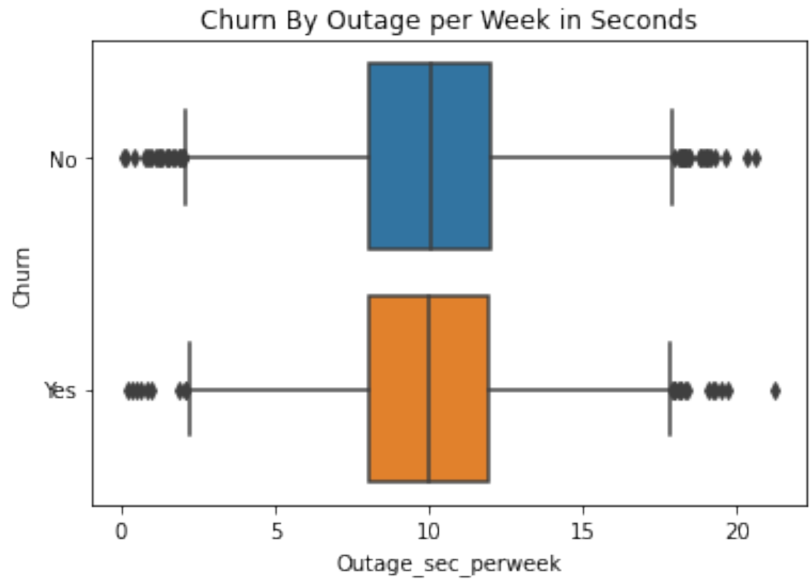
In [22]:
```
sn.boxplot(df3['MonthlyCharge'], df3['Churn']).set_title('Churn By Monthly Ch
```



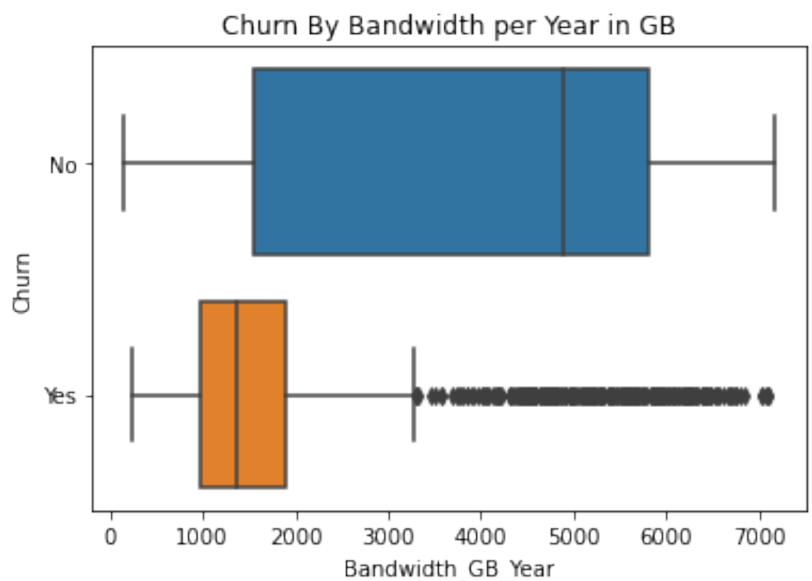
In [23]:
```
sn.boxplot(df3['Income'], df3['Churn']).set_title('Churn By Income');
```



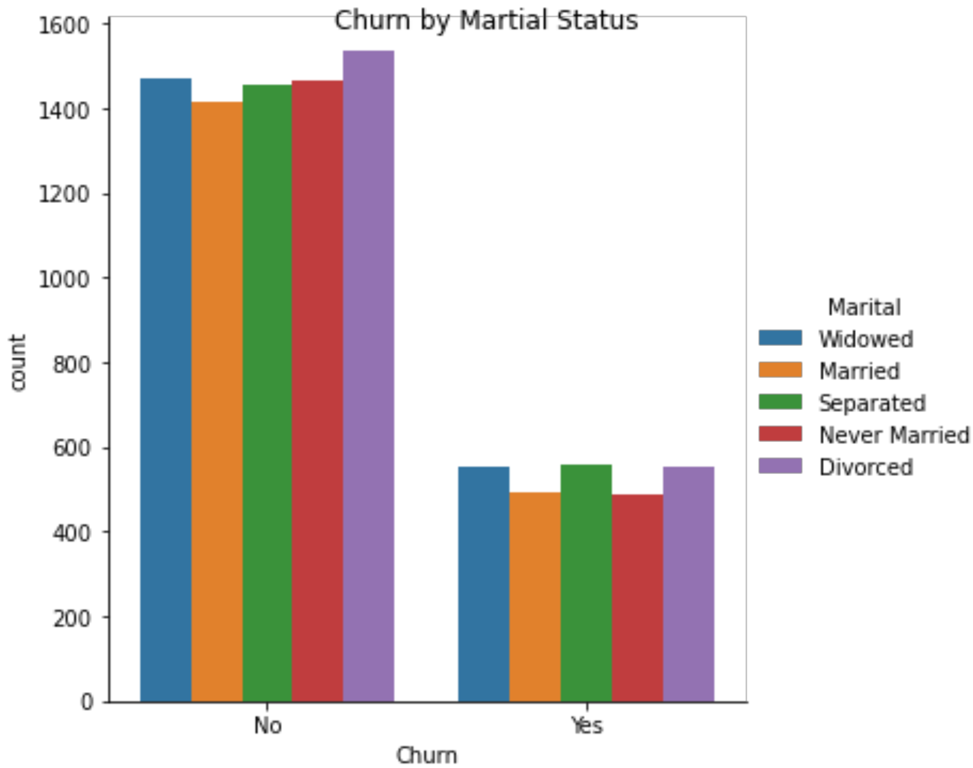
In [24]:
```
sn.boxplot(df3['Outage_sec_perweek'], df3['Churn']).set_title('Churn By Outag
```



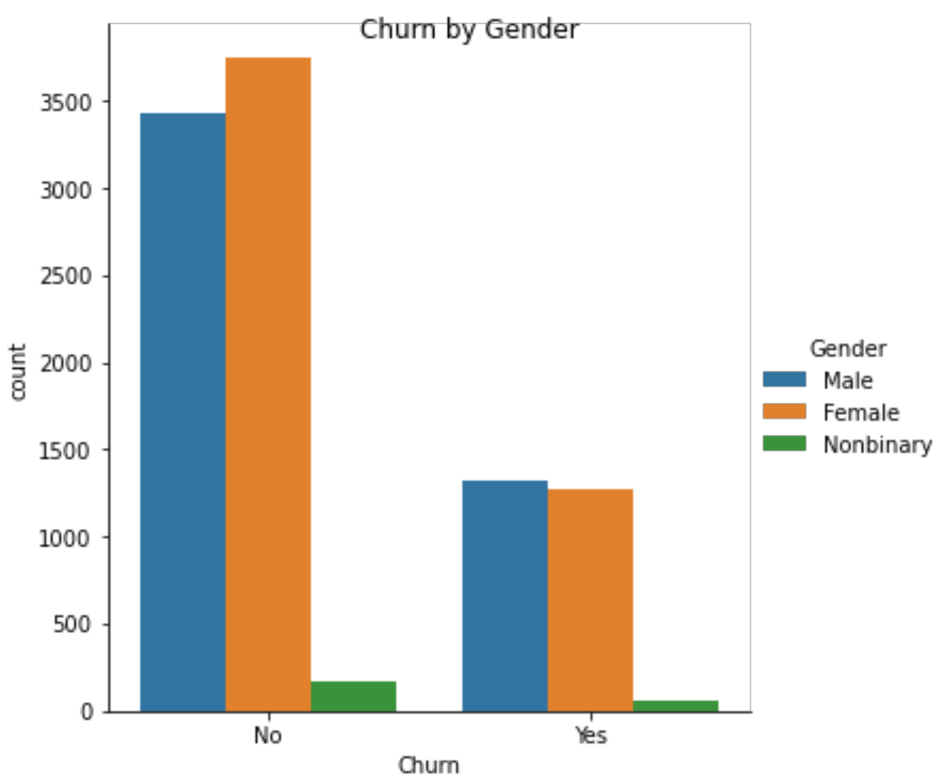
In [25]:
```
sn.boxplot(df3['Bandwidth_GB_Year'], df3['Churn']).set_title('Churn By Bandwi
```



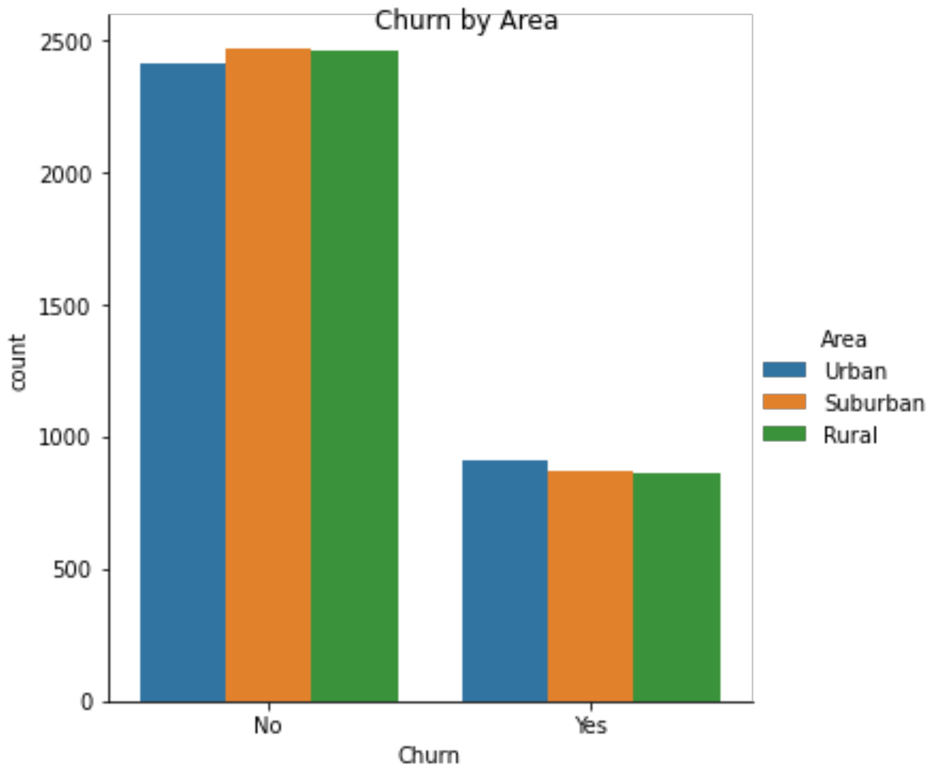
In [26]:
```
g = sn.catplot(x='Churn', hue="Marital", kind="count",data=df3);
g.fig.suptitle('Churn by Martial Status');
```



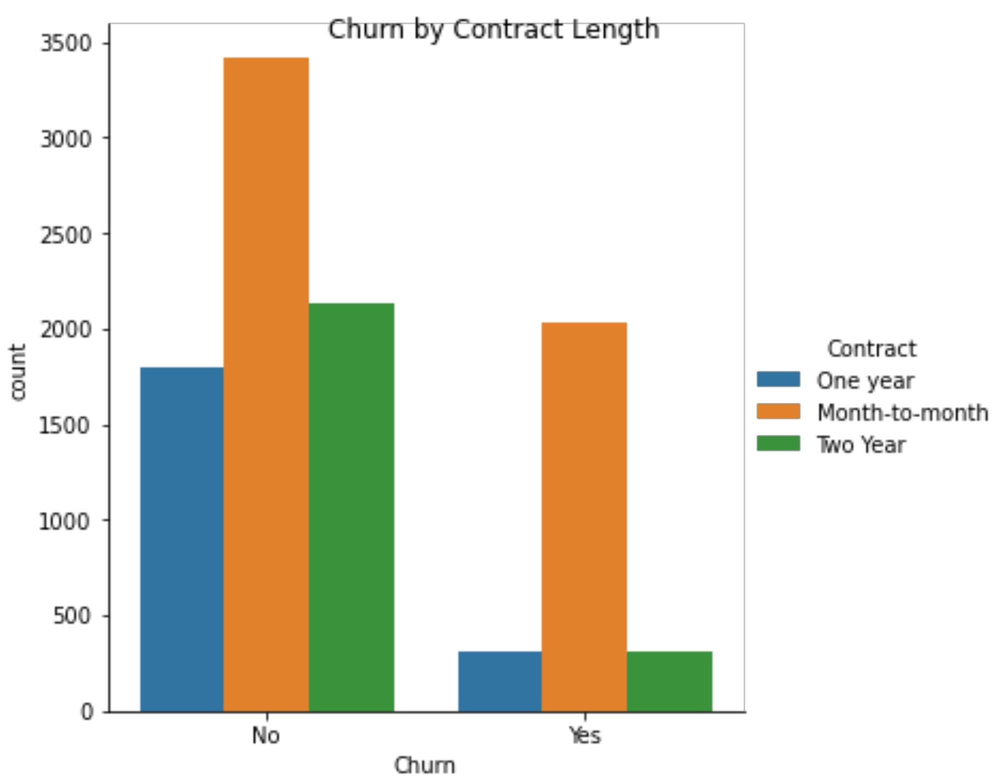
In [27]:
```
g = sn.catplot(x='Churn', hue="Gender", kind="count",data=df3);
g.fig.suptitle('Churn by Gender');
```



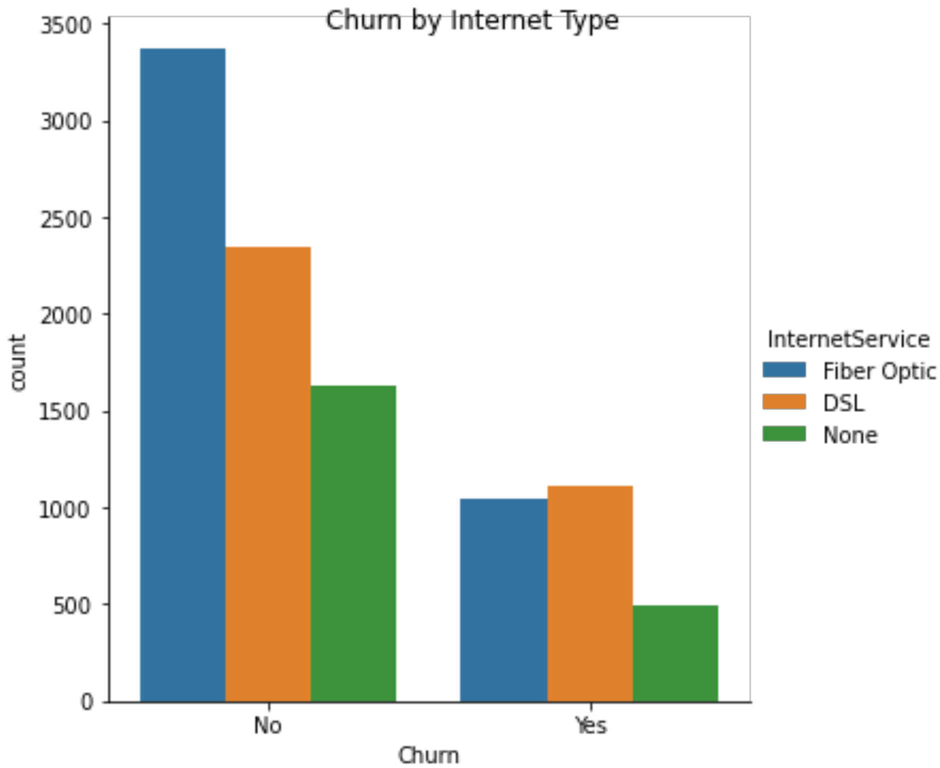
In [28]:
```
g = sn.catplot(x='Churn', hue="Area", kind="count",data=df3);
g.fig.suptitle('Churn by Area');
```



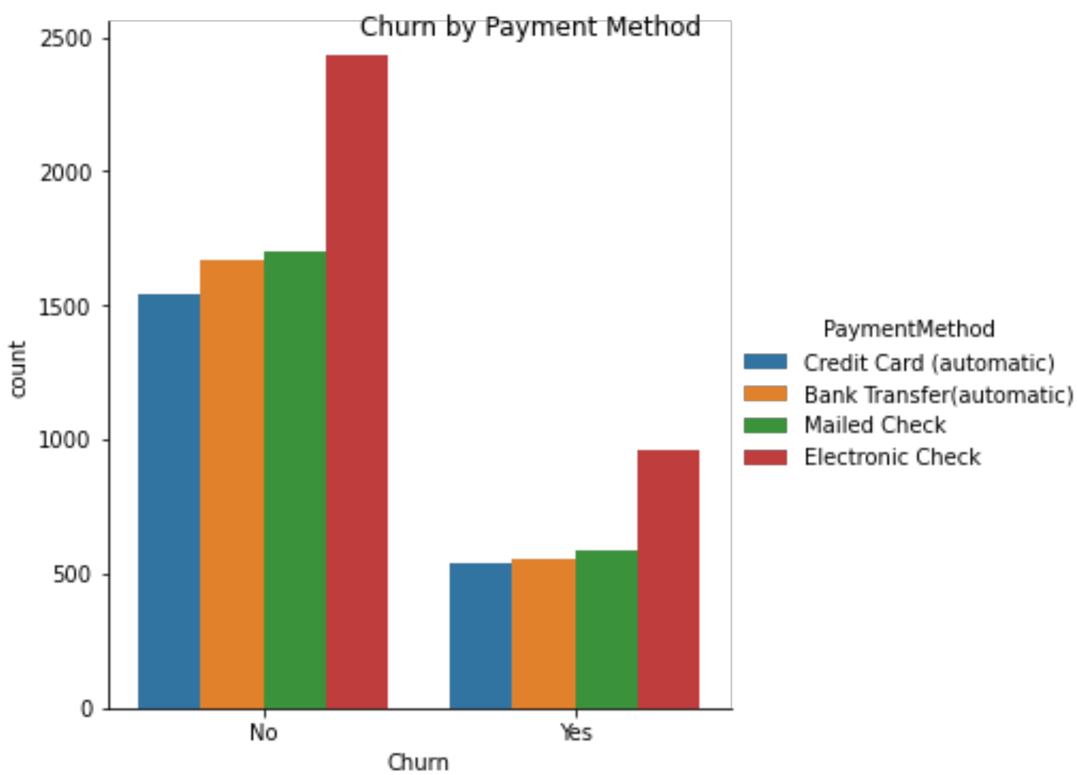
In [29]:
```
g = sn.catplot(x='Churn', hue="Contract", kind="count",data=df3);
g.fig.suptitle('Churn by Contract Length');
```



In [30]:
```
g = sn.catplot(x='Churn', hue="InternetService", kind="count",data=df3);
g.fig.suptitle('Churn by Internet Type');
```



In [31]:
```
g = sn.catplot(x='Churn', hue="PaymentMethod", kind="count",data=df3);
g.fig.suptitle('Churn by Payment Method');
```



Create Dummy variables for all categorical columns and drop unneeded columns. (code used from: https://towardsdatascience.com/the-dummys-guide-to-creating-dummy-variables-f21faddb1d40)

In [32]:
```
df3 = pd.get_dummies(df3)
df3 = df3.drop(columns = ['Churn_No', 'Techie_No', 'Port_modem_No','Tablet_No
```

In [33]:
```
df3.head()
```

Out[33]:

	Population	Children	Age	Income	Outage_sec_perweek	Email	Contacts	Yearly_equip_fa
0	38	0	68	28561.99	7.978323	10	0	
1	10446	1	27	21704.77	11.699080	12	0	
2	3735	4	50	9609.57	10.752800	9	0	
3	13863	1	48	18925.23	14.913540	15	2	
4	11352	0	83	40074.19	8.147417	16	2	

5 rows × 53 columns

Save Copy of Clean Data

In [34]:
```
df3.to_csv('logistic_prepared_churn.csv')
```

Construct Initial Logistic Regression Model

https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a

In [35]:
```
y = df3['Churn_Yes']
X = df3.loc[:, df3.columns != 'Churn_Yes']
```

In [36]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.80, r
```

In [37]:
```
allmodel = LogisticRegression(solver = 'liblinear', random_state = 0)
```

In [38]:
```
allmodel.fit(X_train, y_train)
```

Out[38]:
```
LogisticRegression(random_state=0, solver='liblinear')
```

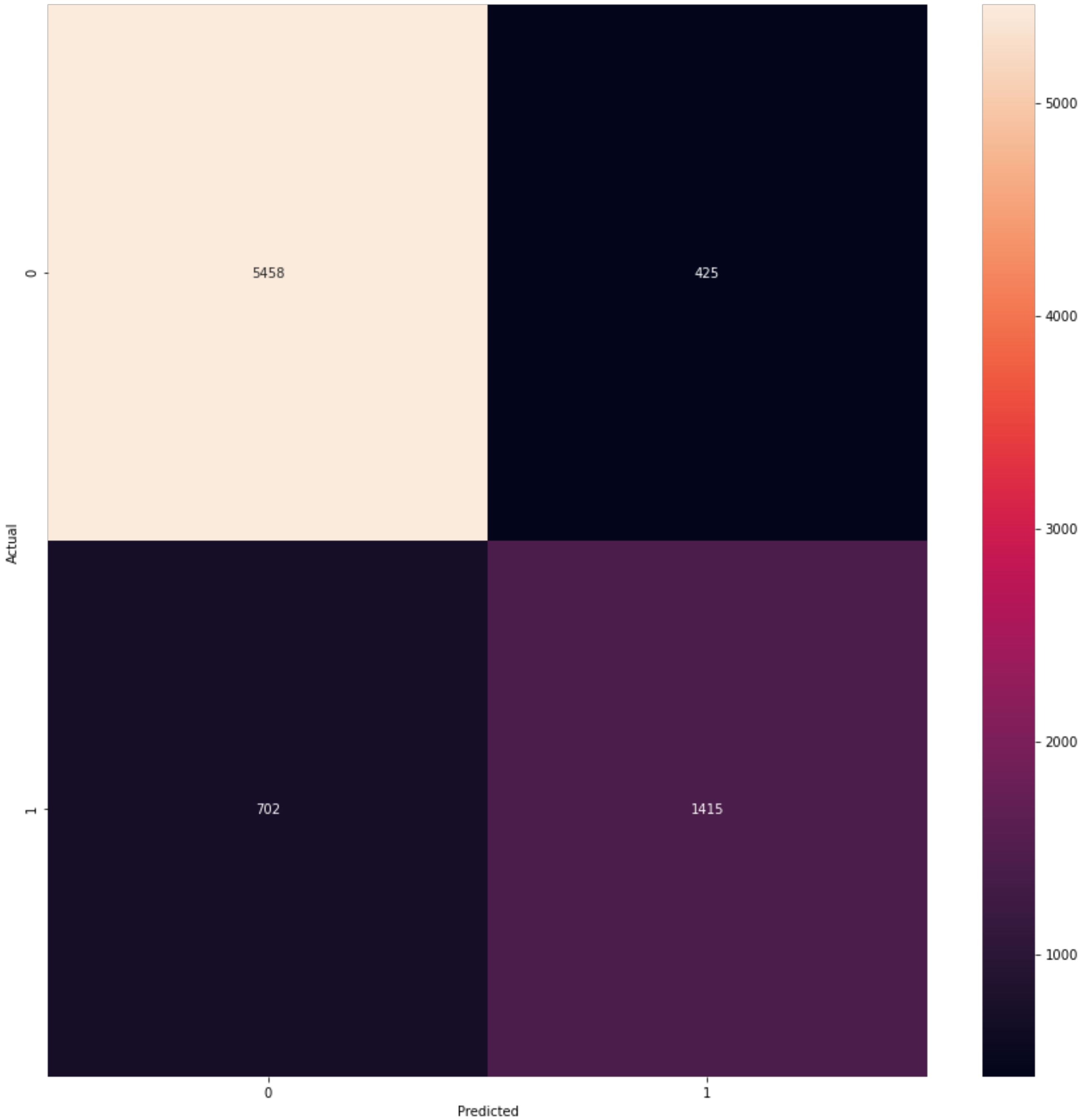
In [39]:
```
allpredictions = allmodel.predict(X_test)
print(allpredictions)
```

```
[0 0 1 ... 0 1 0]
```

https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python

In [40]:

```
cm = metrics.confusion_matrix(y_test, allpredictions)
plt.figure(figsize = (15,15))
sn.heatmap(cm, annot = True, fmt = 'g');
plt.ylabel('Actual');
plt.xlabel('Predicted');
```

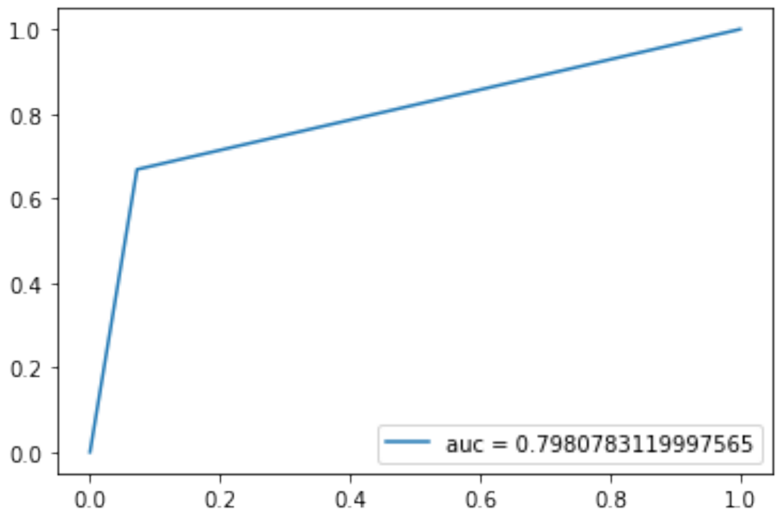


In [41]:

```
accuracy = print('Accuracy:', metrics.accuracy_score(y_test, allpredictions))
precision = print('Precision:',metrics.precision_score(y_test, allpredictions
recall = print('Recall:', metrics.recall_score(y_test, allpredictions))
```

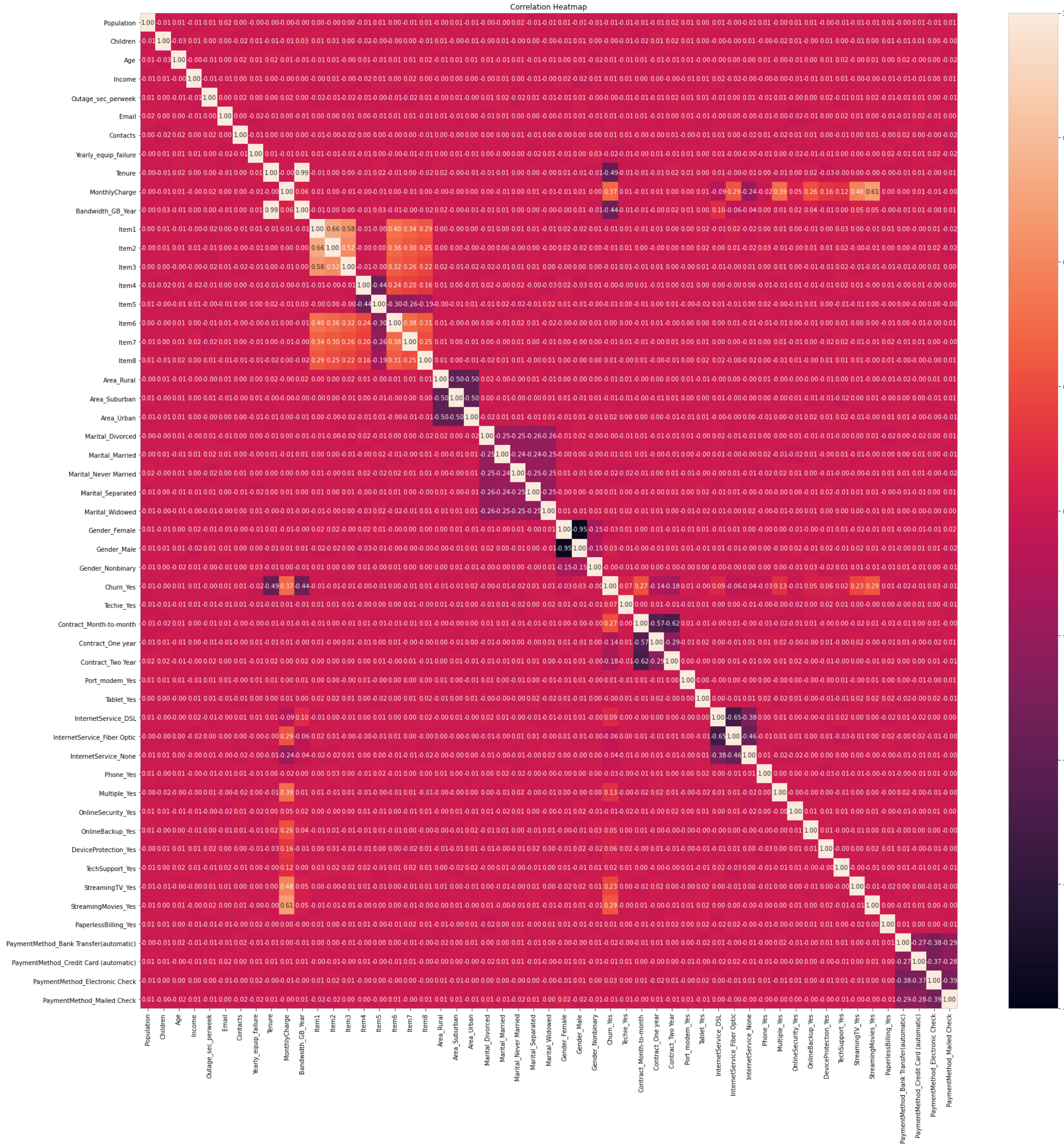
```
Accuracy: 0.859125
Precision: 0.7690217391304348
Recall: 0.6683986773736419
```

In [42]:
```
pred_prob = allmodel.predict(X_test)
fpr, tpr, _  = metrics.roc_curve(y_test, pred_prob)
auc = metrics.roc_auc_score(y_test, pred_prob)
plt.plot(fpr, tpr, label = 'auc = '+str(auc))
plt.legend(loc = 4)
plt.show()
```



Model Reduction

In [43]:
```
plt.figure(figsize = (30,30))
heat_map = sn.heatmap(df3.corr(),vmin = -1, vmax = 1, annot = True, fmt = '.2
heat_map.set_title('Correlation Heatmap');
```



https://datascience.stackexchange.com/questions/39137/how-can-i-check-the-correlation-between-features-and-target-variable

In [44]:
```
pd.options.display.max_rows = 100
df3[df3.columns[1:]].corr()['Churn_Yes'][:]
```

Out[44]:
```
Children                                    -0.004264
Age                                          0.005630
Income                                       0.005937
Outage_sec_perweek                          -0.000156
Email                                        0.012326
Contacts                                     0.008567
Yearly_equip_failure                        -0.015927
Tenure                                      -0.485475
MonthlyCharge                                0.372938
Bandwidth_GB_Year                           -0.441669
Item1                                       -0.007341
Item2                                       -0.013253
Item3                                       -0.011143
Item4                                       -0.003396
Item5                                       -0.013971
Item6                                        0.001130
Item7                                       -0.008851
Item8                                        0.005653
Area_Rural                                  -0.008971
Area_Suburban                               -0.006574
Area_Urban                                   0.015554
Marital_Divorced                            -0.000769
Marital_Married                             -0.007731
Marital_Never Married                       -0.017331
Marital_Separated                            0.014854
Marital_Widowed                              0.010622
Gender_Female                               -0.027021
Gender_Male                                  0.028061
Gender_Nonbinary                            -0.003341
Churn_Yes                                    1.000000
Techie_Yes                                   0.066722
Contract_Month-to-month                      0.267653
Contract_One year                           -0.139043
Contract_Two Year                           -0.178337
Port_modem_Yes                               0.008157
Tablet_Yes                                  -0.002779
InternetService_DSL                          0.093487
InternetService_Fiber Optic                 -0.058472
InternetService_None                        -0.037742
Phone_Yes                                   -0.026297
Multiple_Yes                                 0.131771
OnlineSecurity_Yes                          -0.013540
OnlineBackup_Yes                             0.050508
DeviceProtection_Yes                         0.056489
TechSupport_Yes                              0.018838
StreamingTV_Yes                              0.230151
StreamingMovies_Yes                          0.289262
PaperlessBilling_Yes                         0.007030
PaymentMethod_Bank Transfer(automatic)      -0.017795
PaymentMethod_Credit Card (automatic)       -0.006693
PaymentMethod_Electronic Check               0.029914
PaymentMethod_Mailed Check                  -0.009626
Name: Churn_Yes, dtype: float64
```

Reduced Model

In [45]:
```
df4 = df3[['Churn_Yes', 'Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year', 'Cont
```

In [46]:
```
y = df4['Churn_Yes']
X = df4.loc[:, df4.columns != 'Churn_Yes']
```

In [47]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.80, r
```

In [48]:
```
redmodel = LogisticRegression(solver = 'liblinear', random_state = 0)
```

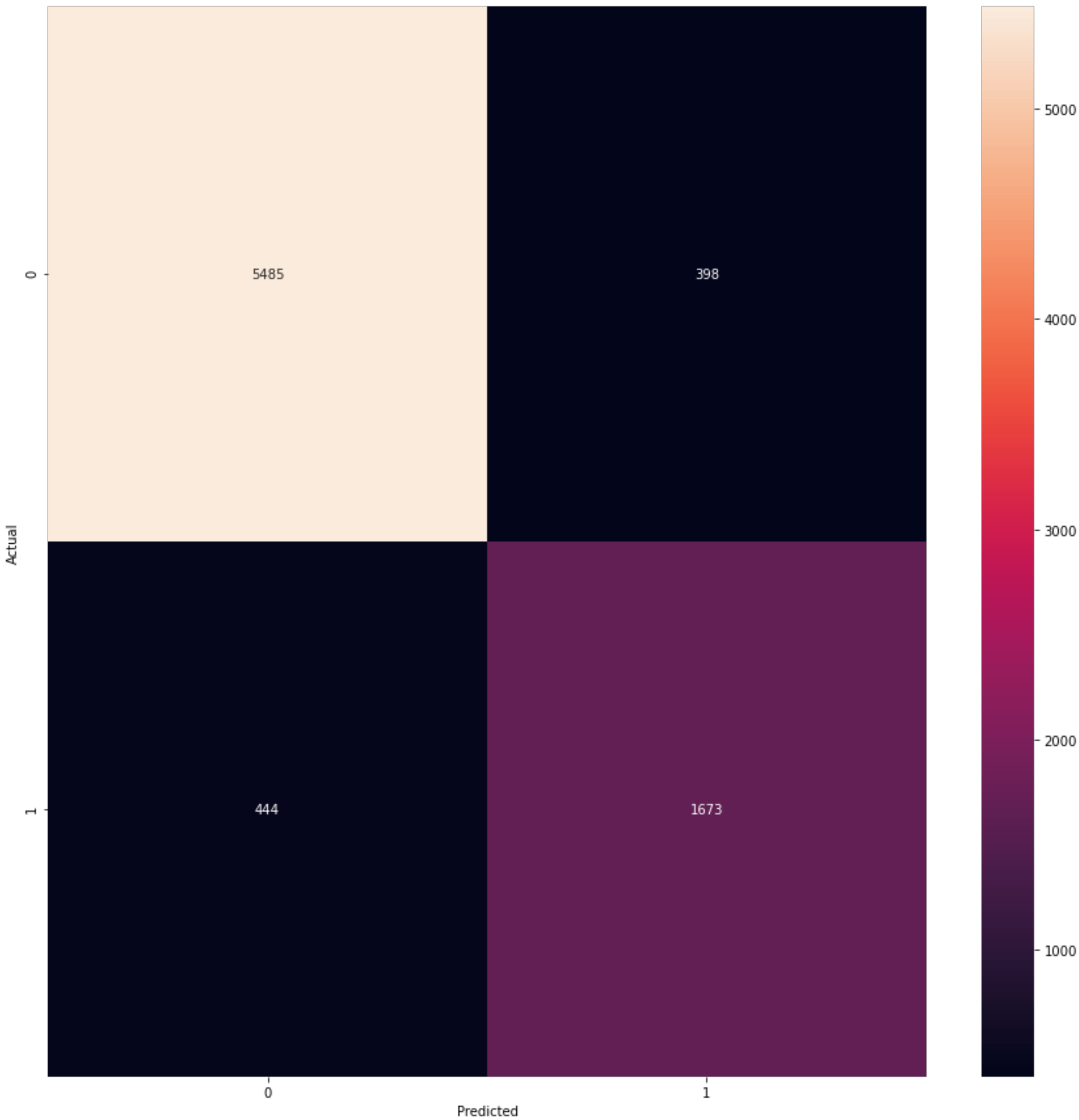
In [49]:
```
redmodel.fit(X_train, y_train)
```

Out[49]:
```
LogisticRegression(random_state=0, solver='liblinear')
```

In [50]:
```
redpredictions = redmodel.predict(X_test)
print(redpredictions)
```

```
[0 0 1 ... 0 1 0]
```

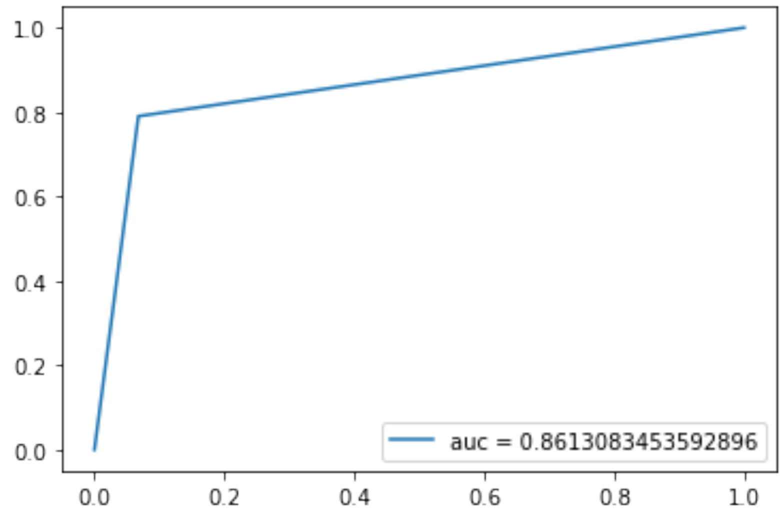
In [51]:
```
cm = metrics.confusion_matrix(y_test, redpredictions)
plt.figure(figsize = (15,15))
sn.heatmap(cm, annot = True, fmt = 'g');
plt.ylabel('Actual');
plt.xlabel('Predicted');
```



In [52]:
```
accuracy = print('Accuracy:', metrics.accuracy_score(y_test, redpredictions))
precision = print('Precision:',metrics.precision_score(y_test, redpredictions
recall = print('Recall:', metrics.recall_score(y_test, redpredictions))
```

```
Accuracy: 0.89475
Precision: 0.8078223080637373
Recall: 0.7902692489371752
```

In [53]:
```
pred_prob = redmodel.predict(X_test)
fpr, tpr, _  = metrics.roc_curve(y_test, pred_prob)
auc = metrics.roc_auc_score(y_test, pred_prob)
plt.plot(fpr, tpr, label = 'auc = '+str(auc))
plt.legend(loc = 4)
plt.show()
```



In [55]:
```
redmodel.coef_
```

Out[55]:
```
array([[-2.71883489e-01,  7.76387217e-03,  2.24675648e-03,
         2.37715719e+00,  1.66342876e+00,  1.45445120e+00]])
```

In [ ]: