sampling-and-cv

October 16, 2023

[1]:

```python
# Loading Libraries
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np
```

[2]:

```python
# Dataframe
df = pd.read_csv('Data.csv')
df.head()
```

[2]:

```
   Unnamed: 0  gender  SeniorCitizen  Partner  Dependents  tenure  \
0        6607       1              0        0           1       1   
1        2598       0              0        0           0       7   
2        2345       0              0        0           1       4   
3        4093       0              0        0           0      29   
4         693       0              0        0           0       3   

   PhoneService  MultipleLines  InternetService  OnlineSecurity  ...  \
0             0              1                0               0  ...   
1             1              0                1               0  ...   
2             1              0                2               1  ...   
3             1              2                1               0  ...   
4             1              2                1               0  ...   

   DeviceProtection  TechSupport  StreamingTV  StreamingMovies  Contract  \
0                 0            0            0                0         0   
1                 2            0            0                0         0   
2                 1            1            1                1         0   
3                 0            0            0                0         0   
4                 0            0            0                0         0   

   PaperlessBilling  PaymentMethod  MonthlyCharges  TotalCharges  Churn  
0                 1              2           25.30          2153      1  
1                 1              2           75.15          4396      0  
2                 1              0           20.05          6211      0  
3                 1              1           76.00          1850      0  
4                 1              1           75.10          2350      1  
```

```
[5 rows x 21 columns]
```

[3]:

```
# Drop unnecessary columns/features
df.drop('Unnamed: 0',axis=1, inplace=True)
df.head()
```

[3]:

```
   gender  SeniorCitizen  Partner  Dependents  tenure  PhoneService  \
0       1              0        0           1       1             0
1       0              0        0           0       7             1
2       0              0        0           1       4             1
3       0              0        0           0      29             1
4       0              0        0           0       3             1

   MultipleLines  InternetService  OnlineSecurity  OnlineBackup  \
0              1                0               0             0
1              0                1               0             0
2              0                2               1             1
3              2                1               0             0
4              2                1               0             0

   DeviceProtection  TechSupport  StreamingTV  StreamingMovies  Contract  \
0                 0            0            0                0         0
1                 2            0            0                0         0
2                 1            1            1                1         0
3                 0            0            0                0         0
4                 0            0            0                0         0

   PaperlessBilling  PaymentMethod  MonthlyCharges  TotalCharges  Churn
0                 1              2           25.30          2153      1
1                 1              2           75.15          4396      0
2                 1              0           20.05          6211      0
3                 1              1           76.00          1850      0
4                 1              1           75.10          2350      1
```

[4]:

```
df.shape
```

[4]:

```
(5282, 20)
```

[5]:

```
# Count of Unique Values
df.Churn.value_counts()
```

[5]:

```
0    3892
1    1390
Name: Churn, dtype: int64
```

[6]:
```
# No value percentage
1390 / (3892+1390)
```

[6]:
```
0.2631578947368421
```

[7]:
```
# Summary of Dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5282 entries, 0 to 5281
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            5282 non-null   int64
 1   SeniorCitizen     5282 non-null   int64
 2   Partner           5282 non-null   int64
 3   Dependents        5282 non-null   int64
 4   tenure            5282 non-null   int64
 5   PhoneService      5282 non-null   int64
 6   MultipleLines     5282 non-null   int64
 7   InternetService   5282 non-null   int64
 8   OnlineSecurity    5282 non-null   int64
 9   OnlineBackup      5282 non-null   int64
 10  DeviceProtection  5282 non-null   int64
 11  TechSupport       5282 non-null   int64
 12  StreamingTV       5282 non-null   int64
 13  StreamingMovies   5282 non-null   int64
 14  Contract          5282 non-null   int64
 15  PaperlessBilling  5282 non-null   int64
 16  PaymentMethod     5282 non-null   int64
 17  MonthlyCharges    5282 non-null   float64
 18  TotalCharges      5282 non-null   int64
 19  Churn             5282 non-null   int64
dtypes: float64(1), int64(19)
memory usage: 825.4 KB
```

0.1 Analysis the dataset using seaborn / matplotlib / scatter

[8]:
```
# Correlation between features
df.corr()
```

[8]:
```
                 gender  SeniorCitizen   Partner  Dependents    tenure  \
gender         1.000000      -0.005691 -0.010143    0.011058  0.001303
SeniorCitizen -0.005691       1.000000  0.016648   -0.211271  0.006176
Partner       -0.010143       0.016648  1.000000    0.447629  0.382432
Dependents     0.011058      -0.211271  0.447629    1.000000  0.162933
tenure         0.001303       0.006176  0.382432    0.162933  1.000000
```

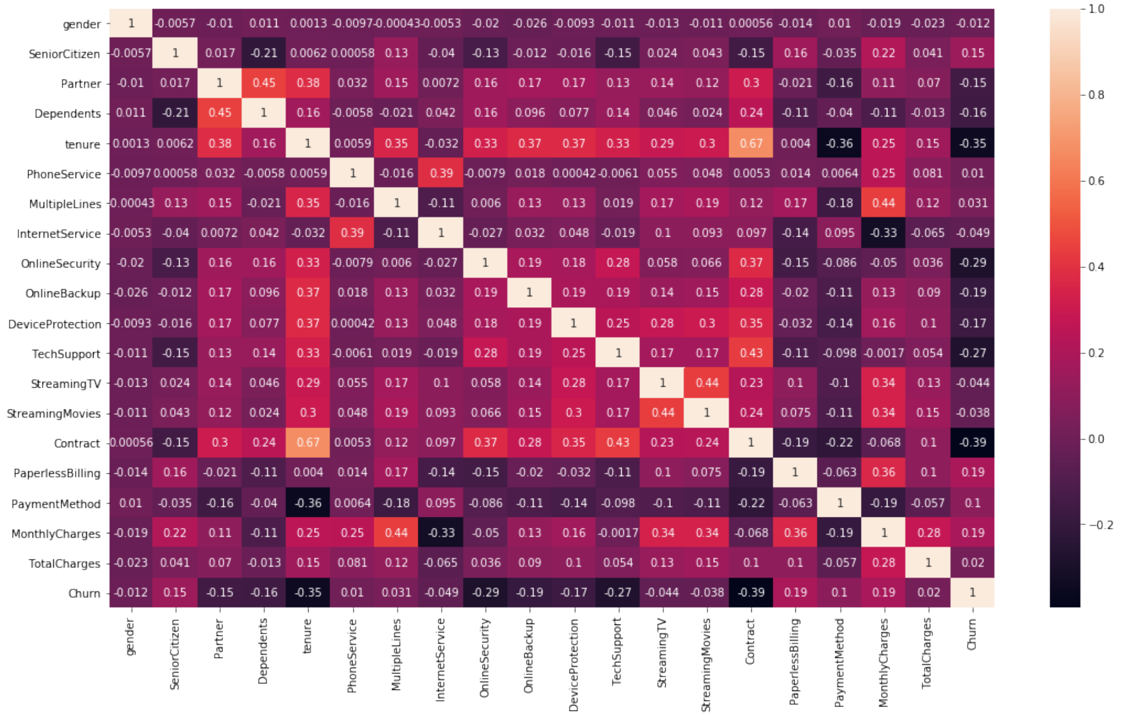
```
PhoneService     -0.009706       0.000581  0.031578   -0.005830  0.005912
MultipleLines    -0.000430       0.129040  0.151068   -0.020722  0.354790
InternetService  -0.005294      -0.040186  0.007205    0.041972 -0.032037
OnlineSecurity   -0.020189      -0.129719  0.159928    0.157262  0.325451
OnlineBackup     -0.025773      -0.012166  0.166939    0.095640  0.371342
DeviceProtection -0.009305      -0.015922  0.168815    0.076783  0.369331
TechSupport      -0.010619      -0.151078  0.132582    0.137274  0.326993
StreamingTV      -0.012994       0.024338  0.136715    0.045948  0.288135
StreamingMovies  -0.011420       0.043124  0.122310    0.023907  0.301600
Contract          0.000555      -0.151939  0.303243    0.243080  0.671184
PaperlessBilling -0.014090       0.156417 -0.020634   -0.109935  0.004043
PaymentMethod     0.010188      -0.035050 -0.160535   -0.040414 -0.360323
MonthlyCharges   -0.018822       0.219945  0.105603   -0.114920  0.253605
TotalCharges     -0.022718       0.040956  0.069859   -0.013196  0.152843
Churn            -0.011997       0.146549 -0.150053   -0.164490 -0.345544

                  PhoneService  MultipleLines  InternetService  \
gender               -0.009706      -0.000430        -0.005294
SeniorCitizen         0.000581       0.129040        -0.040186
Partner               0.031578       0.151068         0.007205
Dependents           -0.005830      -0.020722         0.041972
tenure                0.005912       0.354790        -0.032037
PhoneService          1.000000      -0.016345         0.385682
MultipleLines        -0.016345       1.000000        -0.105796
InternetService       0.385682      -0.105796         1.000000
OnlineSecurity       -0.007874       0.006028        -0.027201
OnlineBackup          0.017500       0.134460         0.032058
DeviceProtection      0.000422       0.132798         0.048320
TechSupport          -0.006139       0.019266        -0.018510
StreamingTV           0.055390       0.172380         0.101060
StreamingMovies       0.048362       0.188860         0.092672
Contract              0.005342       0.120023         0.097158
PaperlessBilling      0.014489       0.172369        -0.141856
PaymentMethod         0.006362      -0.176313         0.095054
MonthlyCharges        0.247419       0.436398        -0.325588
TotalCharges          0.081045       0.118773        -0.064748
Churn                 0.010122       0.031270        -0.048820

                  OnlineSecurity  OnlineBackup  DeviceProtection  TechSupport  \
gender                 -0.020189     -0.025773         -0.009305    -0.010619
SeniorCitizen          -0.129719     -0.012166         -0.015922    -0.151078
Partner                 0.159928      0.166939          0.168815     0.132582
Dependents              0.157262      0.095640          0.076783     0.137274
tenure                  0.325451      0.371342          0.369331     0.326993
PhoneService           -0.007874      0.017500          0.000422    -0.006139
MultipleLines           0.006028      0.134460          0.132798     0.019266
InternetService        -0.027201      0.032058          0.048320    -0.018510
```

```
OnlineSecurity          1.000000      0.186626          0.182355     0.276510
OnlineBackup            0.186626      1.000000          0.191348     0.189892
DeviceProtection        0.182355      0.191348          1.000000     0.247866
TechSupport             0.276510      0.189892          0.247866     1.000000
StreamingTV             0.057760      0.142882          0.278896     0.174178
StreamingMovies         0.065996      0.151145          0.301894     0.172420
Contract                0.371159      0.282646          0.352138     0.428750
PaperlessBilling       -0.146473     -0.019611         -0.032091    -0.113245
PaymentMethod          -0.085892     -0.111508         -0.135513    -0.097672
MonthlyCharges         -0.049605      0.125340          0.162808    -0.001710
TotalCharges            0.036493      0.090223          0.103120     0.053532
Churn                  -0.288926     -0.193152         -0.173138    -0.274718

                  StreamingTV  StreamingMovies  Contract  PaperlessBilling  \
gender              -0.012994        -0.011420  0.000555         -0.014090
SeniorCitizen        0.024338         0.043124 -0.151939          0.156417
Partner              0.136715         0.122310  0.303243         -0.020634
Dependents           0.045948         0.023907  0.243080         -0.109935
tenure               0.288135         0.301600  0.671184          0.004043
PhoneService         0.055390         0.048362  0.005342          0.014489
MultipleLines        0.172380         0.188860  0.120023          0.172369
InternetService      0.101060         0.092672  0.097158         -0.141856
OnlineSecurity       0.057760         0.065996  0.371159         -0.146473
OnlineBackup         0.142882         0.151145  0.282646         -0.019611
DeviceProtection     0.278896         0.301894  0.352138         -0.032091
TechSupport          0.174178         0.172420  0.428750         -0.113245
StreamingTV          1.000000         0.437809  0.231143          0.101389
StreamingMovies      0.437809         1.000000  0.236128          0.075255
Contract             0.231143         0.236128  1.000000         -0.185507
PaperlessBilling     0.101389         0.075255 -0.185507          1.000000
PaymentMethod       -0.100597        -0.114956 -0.218531         -0.063408
MonthlyCharges       0.338557         0.339162 -0.067540          0.359566
TotalCharges         0.134112         0.150553  0.104879          0.101619
Churn               -0.043920        -0.038240 -0.394490          0.188793

                  PaymentMethod  MonthlyCharges  TotalCharges     Churn
gender                 0.010188       -0.018822     -0.022718 -0.011997
SeniorCitizen         -0.035050        0.219945      0.040956  0.146549
Partner               -0.160535        0.105603      0.069859 -0.150053
Dependents            -0.040414       -0.114920     -0.013196 -0.164490
tenure                -0.360323        0.253605      0.152843 -0.345544
PhoneService           0.006362        0.247419      0.081045  0.010122
MultipleLines         -0.176313        0.436398      0.118773  0.031270
InternetService        0.095054       -0.325588     -0.064748 -0.048820
OnlineSecurity        -0.085892       -0.049605      0.036493 -0.288926
OnlineBackup          -0.111508        0.125340      0.090223 -0.193152
DeviceProtection      -0.135513        0.162808      0.103120 -0.173138
```

```
TechSupport        -0.097672      -0.001710      0.053532 -0.274718
StreamingTV        -0.100597       0.338557      0.134112 -0.043920
StreamingMovies    -0.114956       0.339162      0.150553 -0.038240
Contract           -0.218531      -0.067540      0.104879 -0.394490
PaperlessBilling   -0.063408       0.359566      0.101619  0.188793
PaymentMethod       1.000000      -0.194857     -0.056972  0.100015
MonthlyCharges     -0.194857       1.000000      0.279822  0.186615
TotalCharges       -0.056972       0.279822      1.000000  0.020294
Churn               0.100015       0.186615      0.020294  1.000000
```

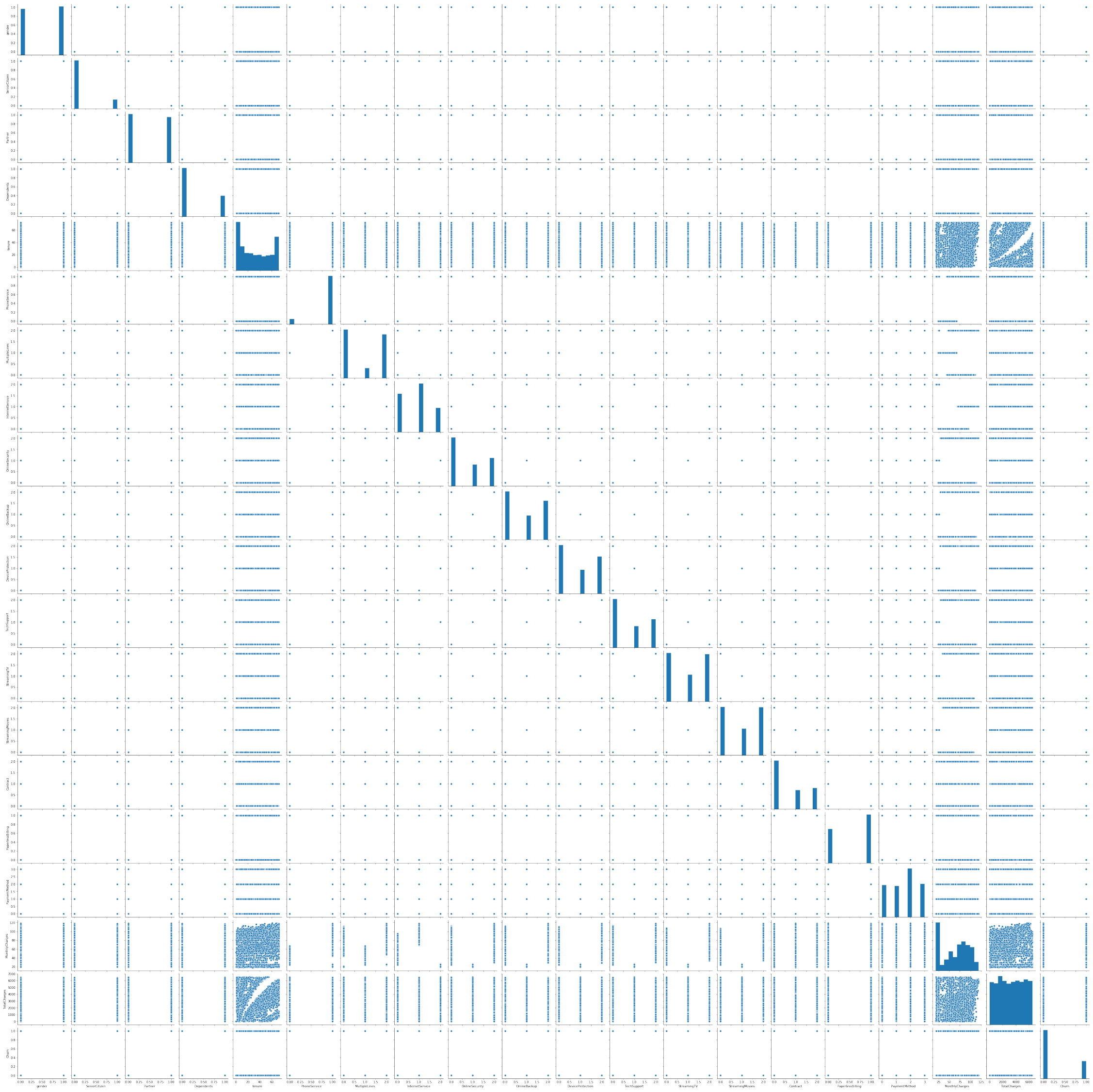
[9]:
```
plt.figure(figsize=(18,10))
sns.heatmap(df.corr(),annot=True)
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26417db08c8>
```



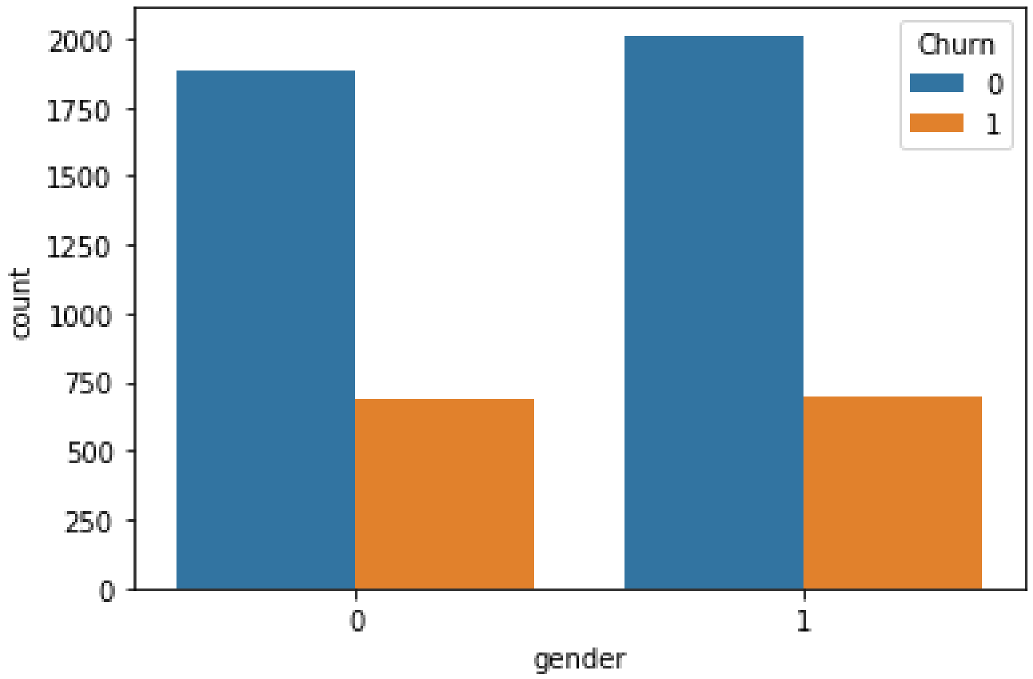
[10]:
```
sns.pairplot(df)
```

[10]:
```
<seaborn.axisgrid.PairGrid at 0x264186384c8>
```



```
[11]: sns.countplot(df.gender,hue='Churn',data=df)
```

```
[11]: <matplotlib.axes._subplots.AxesSubplot at 0x2642b64fb48>
```

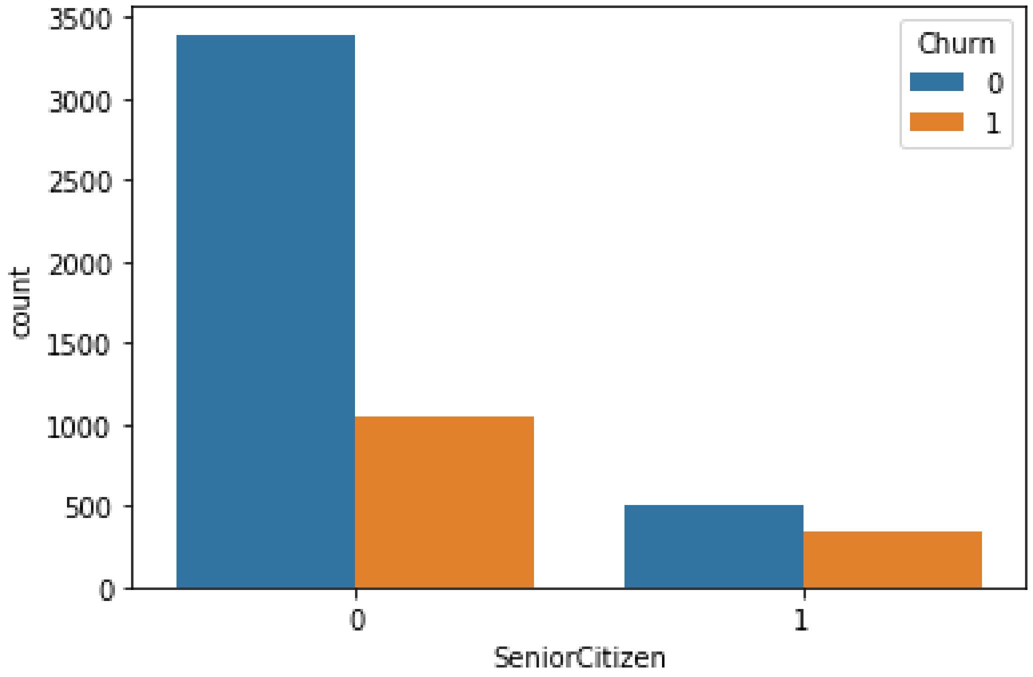


[12]:

```python
sns.countplot(df['SeniorCitizen'],hue=df.Churn)
```

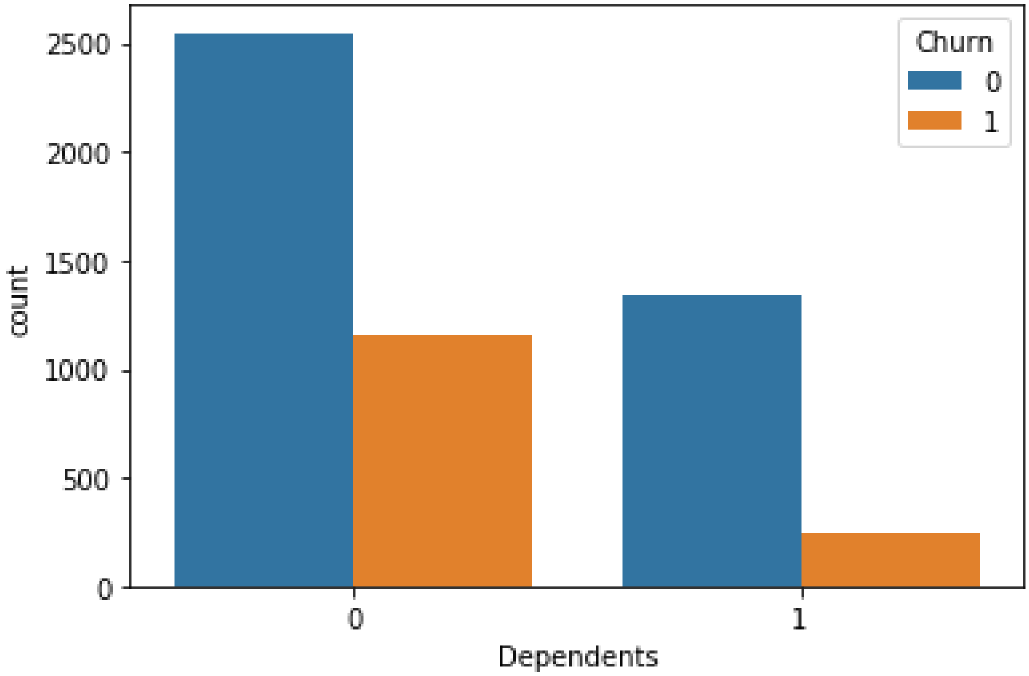
[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2642b509d08>
```



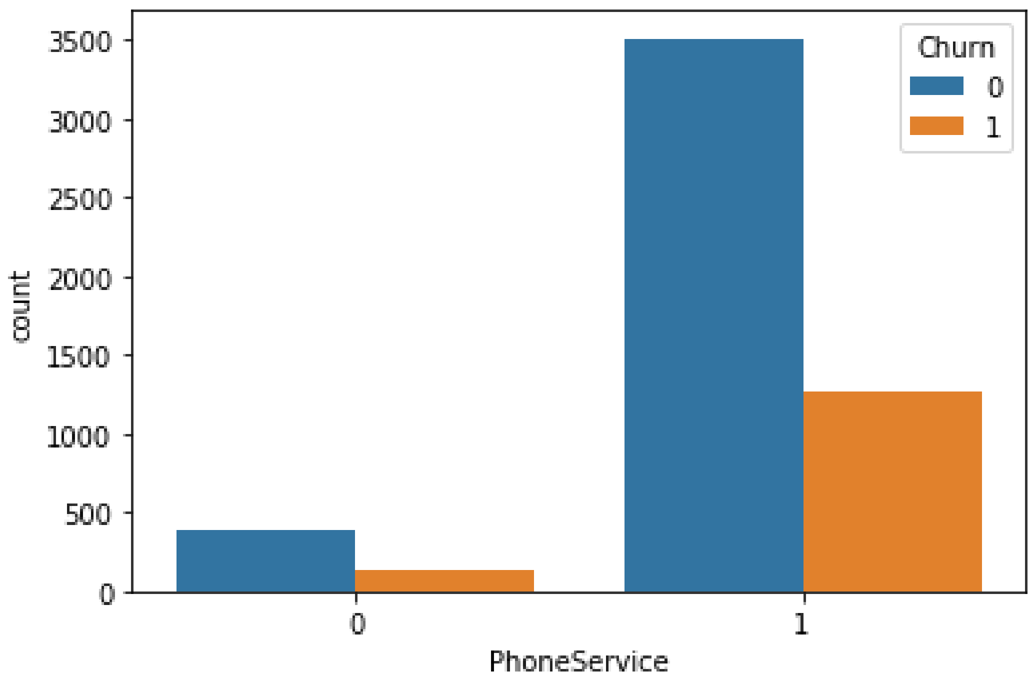
```
[13]: sns.countplot(df.Dependents,hue=df.Churn)
```

```
[13]: <matplotlib.axes._subplots.AxesSubplot at 0x2642bdc2748>
```



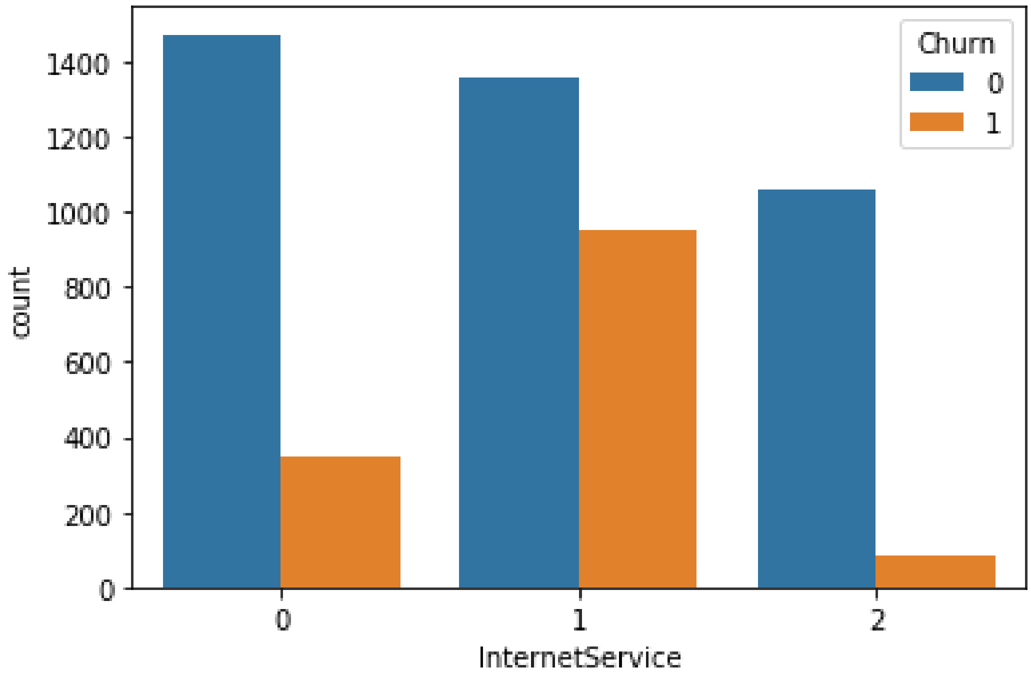
```
[14]: sns.countplot(df.PhoneService,hue=df.Churn)
```

```
[14]: <matplotlib.axes._subplots.AxesSubplot at 0x2642f3d3348>
```



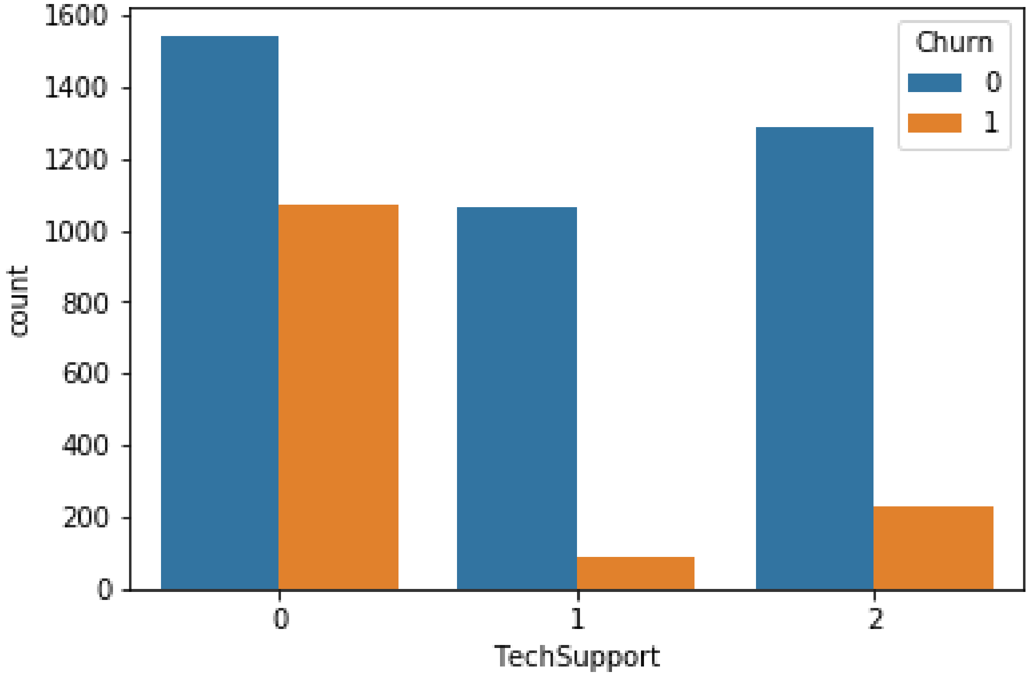
```
[15]: sns.countplot(df['InternetService'],hue=df.Churn)
```

```
[15]: <matplotlib.axes._subplots.AxesSubplot at 0x2642f3d3588>
```



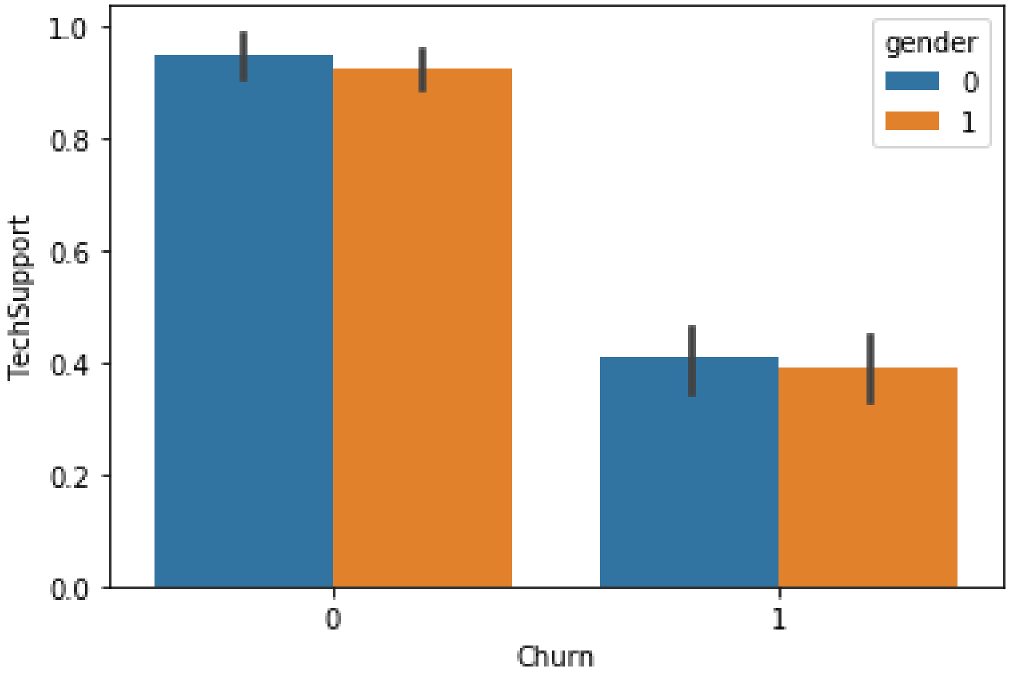
[16]:
```
sns.countplot(df.TechSupport,hue="Churn" , data=df)
```

[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26430429508>
```



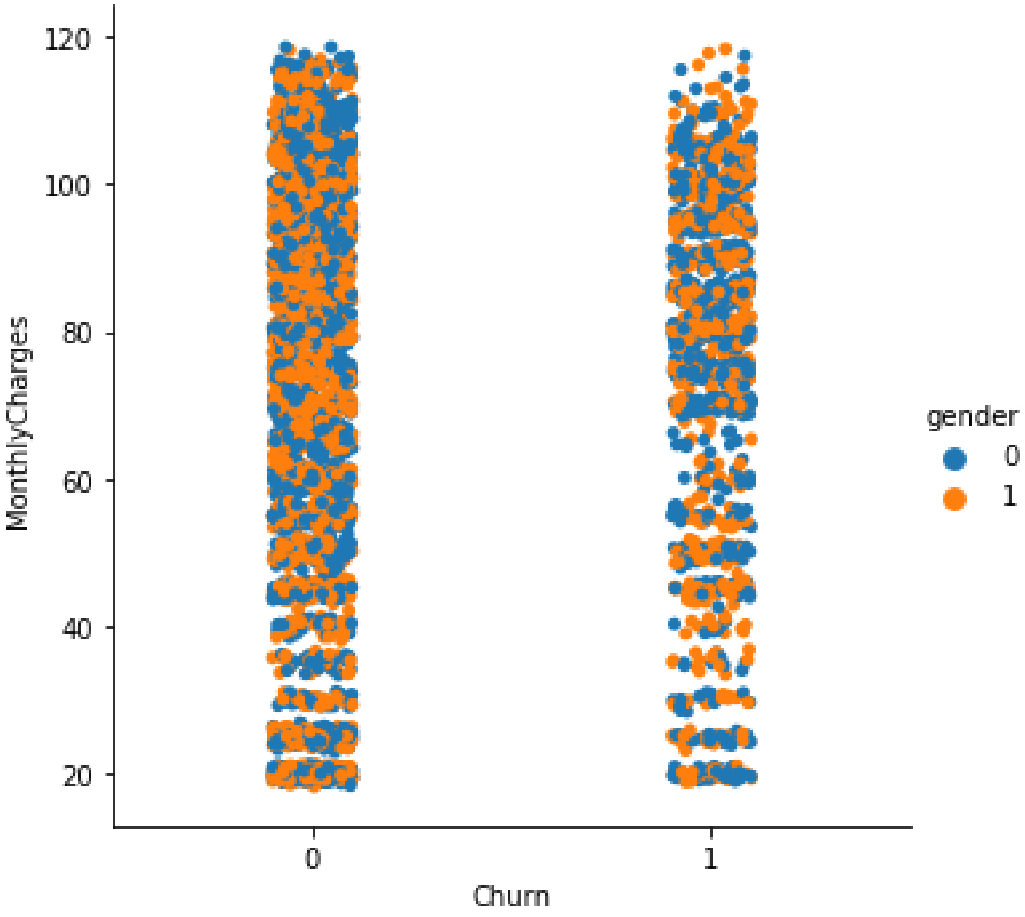
[17]:
```
sns.barplot(df.Churn,df.TechSupport,hue='gender',data=df)
```

[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2643045dc08>
```



[18]:
```
sns.catplot(x="Churn", y="MonthlyCharges",hue="gender", data=df)
```

[18]:
```
<seaborn.axisgrid.FacetGrid at 0x2643042cc48>
```



```
[19]: x = df.drop('Churn',axis=1)
      y = df.Churn
```

```
[20]: x.shape
```

```
[20]: (5282, 19)
```

```
[21]: y.value_counts()
```

```
[21]: 0    3892
      1    1390
      Name: Churn, dtype: int64
```

1 ML model and evaluating model by cross validation before sampling

1.1 Hold out Cross Validation

[22]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest=train_test_split(x,y, test_size=0.3,␣
  ↪random_state=42)
```

[23]:
```
xtrain.shape
```

[23]:
```
(3697, 19)
```

[24]:
```
ytrain.shape
```

[24]:
```
(3697,)
```

1.1.1 Decision Tree Classifier

[25]:
```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
```

[26]:
```
dtc.fit(xtrain,ytrain)
```

[26]:
```
DecisionTreeClassifier()
```

[27]:
```
score_dtc = dtc.score(xtest,ytest)
```

[28]:
```
score_dtc
```

[28]:
```
0.7129337539432177
```

1.1.2 Random Forest Classifier

[29]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
```

[30]:
```
rfc.fit(xtrain,ytrain)
```

[30]:
```
RandomForestClassifier()
```

[31]:
```
score_rfc = rfc.score(xtest,ytest)
```

[32]:
```
score_rfc
```

[32]:
```
0.7899053627760252
```

1.1.3 Xtream Gradient boosting (XGBoost) Classifier

```
[33]: pip install xgboost
```

```
Requirement already satisfied: xgboost in d:\anaconda3\lib\site-packages (1.6.1)
Requirement already satisfied: numpy in d:\anaconda3\lib\site-packages (from
xgboost) (1.18.1)
Requirement already satisfied: scipy in d:\anaconda3\lib\site-packages (from
xgboost) (1.4.1)
Note: you may need to restart the kernel to use updated packages.
```

```
[34]: from xgboost import XGBClassifier
      xgbc = XGBClassifier()
```

```
[35]: xgbc.fit(xtrain,ytrain)
```

```
[35]: XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
                    colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
                    early_stopping_rounds=None, enable_categorical=False,
                    eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
                    importance_type=None, interaction_constraints='',
                    learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
                    max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
                    missing=nan, monotone_constraints='()', n_estimators=100,
                    n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
                    reg_alpha=0, reg_lambda=1, …)
```

```
[36]: score_xgbc = xgbc.score(xtest,ytest)
```

```
[37]: score_xgbc
```

```
[37]: 0.7728706624605678
```

1.1.4 AdaBoost Classifier

```
[38]: from sklearn.ensemble import AdaBoostClassifier
      abc = AdaBoostClassifier()
```

```
[39]: abc.fit(xtrain,ytrain)
```

```
[39]: AdaBoostClassifier()
```

```
[40]: score_abc = abc.score(xtest,ytest)
```

```
[41]: score_abc
```

```
[41]: 0.7867507886435331
```

1.1.5 K-Nearest Neighbour (KNN) Classifier

```
[42]: from sklearn.neighbors import KNeighborsClassifier
      knnc = KNeighborsClassifier()
```

```
[43]: knnc.fit(xtrain,ytrain)
```

```
[43]: KNeighborsClassifier()
```

```
[44]: score_knnc = knnc.score(xtest,ytest)
```

```
[45]: score_knnc
```

```
[45]: 0.7488958990536277
```

1.1.6 Logistic Regression

```
[46]: from sklearn.linear_model import LogisticRegression
      lrc = LogisticRegression()
```

```
[47]: lrc.fit(xtrain,ytrain)
```

```
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
[47]: LogisticRegression()
```

```
[48]: score_lrc = lrc.score(xtest,ytest)
```

```
[49]: score_lrc
```

```
[49]: 0.7886435331230284
```

1.2 Comparison

```
[50]: cdf = pd.DataFrame([['Decision Tree',score_dtc],['Random␣
       ↪Forest',score_rfc],['XGBoost',score_xgbc],['Ada␣
       ↪Boost',score_abc],['K-Nearest Neighbor',score_knnc],['Logistic␣
       ↪Regression',score_lrc]],columns=['Classifier', 'Accuracy'])
```

```
cdf
```

[50]:

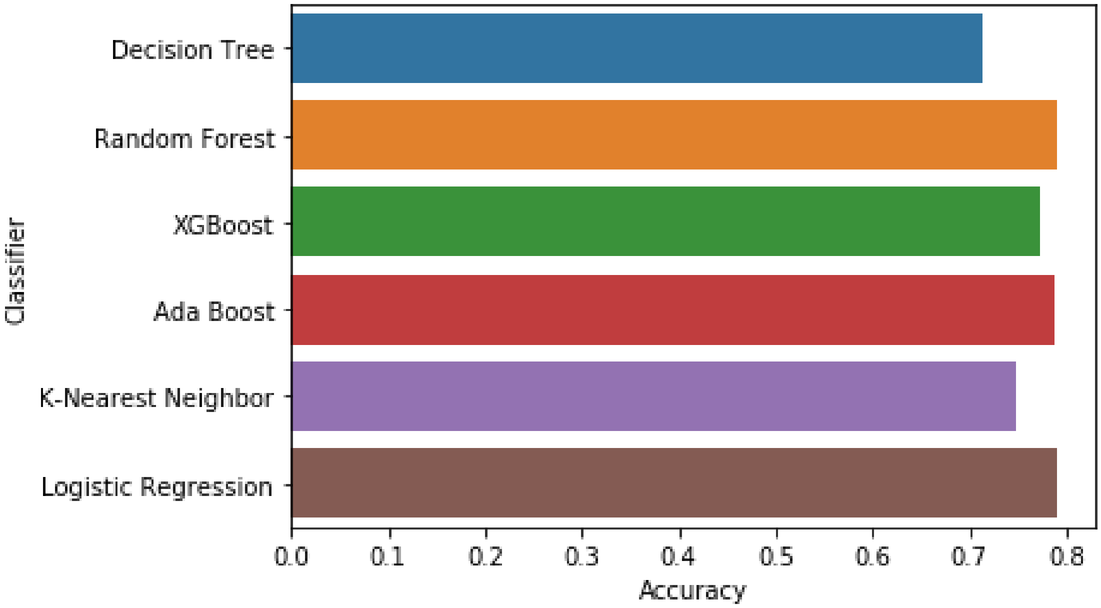
```
             Classifier  Accuracy
0         Decision Tree  0.712934
1         Random Forest  0.789905
2               XGBoost  0.772871
3             Ada Boost  0.786751
4    K-Nearest Neighbor  0.748896
5   Logistic Regression  0.788644
```

[51]:

```
sns.barplot(cdf.Accuracy,cdf.Classifier)
```

[51]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x26431f2b048>
```



1.2.1 K-Fold Cross Validation

[52]:

```
from sklearn.model_selection import KFold, cross_val_score
kf = KFold(n_splits=5) #each fold will contain 20% data
```

[53]:

```
result_kf = cross_val_score(lrc, x, y,cv=kf)
```

```
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

[54]:

```
result_kf
```

[54]:

```
array([0.81740776, 0.78807947, 0.7907197 , 0.77556818, 0.80681818])
```

[55]:

```
result_kf.min()
```

[55]:

```
0.7755681818181818
```

[56]:

```
result_kf.max()
```

[56]:

```
0.8174077578051088
```

1.2.2 Stratified k-fold Cross Validation

[57]:

```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold()
```

[58]:

```
result_skf = cross_val_score(lrc, x, y,cv=skf)
```

```
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
[59]: result_skf
```

```
[59]: array([0.81078524, 0.79564806, 0.79166667, 0.77840909, 0.79734848])
```

```
[60]: result_skf.max()
```

```
[60]: 0.8107852412488175
```

```
[61]: result_skf.min()
```

```
[61]: 0.7784090909090909
```

1.2.3 Leave One Out Cross Validation(LOOC)

```
[62]: #from sklearn.model_selection import LeaveOneOut
      #loo = LeaveOneOut()
      #result = cross_val_score(lrc,X,y,cv=loo)
      #result
      #result.mean()
```

2 sampling on given dataset and Create ML model and evaluating model by cross validation again (after sampling)

[63]:

```
y.value_counts()
```

[63]:

```
0    3892
1    1390
Name: Churn, dtype: int64
```

[64]:

```
x.shape
```

[64]:

```
(5282, 19)
```

2.1 Sampling

2.2 S1. CROSS Validation with Synthetic Minority Oversampling Technique (SMOTETomek)

[65]:

```
pip install imblearn
```

```
Collecting imblearn
  Using cached imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Collecting imbalanced-learn
  Using cached imbalanced_learn-0.9.1-py3-none-any.whl (199 kB)
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\anaconda3\lib\site-
packages (from imbalanced-learn->imblearn) (3.1.0)
  Using cached imbalanced_learn-0.9.0-py3-none-any.whl (199 kB)
Requirement already satisfied: scikit-learn>=1.0.1 in d:\anaconda3\lib\site-
packages (from imbalanced-learn->imblearn) (1.0.2)
Requirement already satisfied: scipy>=1.1.0 in d:\anaconda3\lib\site-packages
(from imbalanced-learn->imblearn) (1.4.1)
Requirement already satisfied: numpy>=1.14.6 in d:\anaconda3\lib\site-packages
(from imbalanced-learn->imblearn) (1.18.1)
Requirement already satisfied: joblib>=0.11 in d:\anaconda3\lib\site-packages
(from imbalanced-learn->imblearn) (0.14.1)
Installing collected packages: imbalanced-learn, imblearn
Successfully installed imbalanced-learn-0.9.0 imblearn-0.0
Note: you may need to restart the kernel to use updated packages.
```

[66]:

```
from imblearn.combine import SMOTETomek
smt = SMOTETomek()
```

[70]:

```
x_smt, y_smt = smt.fit_resample(x,y)
```

[71]:

```
x_smt.shape
```

[71]:

```
(7190, 19)
```

[73]:
```
y_smt.value_counts()
```

[73]:
```
1    3595
0    3595
Name: Churn, dtype: int64
```

2.3 Hold out cross validation

[74]:
```
x_smt_train,x_smt_test,y_smt_train,y_smt_test =␣
 ↪train_test_split(x_smt,y_smt,train_size=.7,random_state=42)
```

[76]:
```
x_smt_train.shape
```

[76]:
```
(5033, 19)
```

[77]:
```
x_smt_test.shape
```

[77]:
```
(2157, 19)
```

2.3.1 Decision Tree Classifier

[78]:
```
dtc_smt = DecisionTreeClassifier()
dtc_smt.fit(x_smt_train,y_smt_train)
```

[78]:
```
DecisionTreeClassifier()
```

[79]:
```
score_dtc_smt = dtc_smt.score(x_smt_test,y_smt_test)
```

[80]:
```
score_dtc_smt
```

[80]:
```
0.8191933240611962
```

2.3.2 Random Forest Classifier

[81]:
```
rfc_smt = RandomForestClassifier()
rfc_smt.fit(x_smt_train,y_smt_train)
```

[81]:
```
RandomForestClassifier()
```

[82]:
```
score_rfc_smt = rfc_smt.score(x_smt_test,y_smt_test)
score_rfc_smt
```

[82]:
```
0.8687992582290218
```

2.3.3 XGBoost Classifier

[83]:
```
xgbc_smt = XGBClassifier()
xgbc_smt.fit(x_smt_train,y_smt_train)
```

[83]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, …)
```

[84]:
```
score_xgbc_smt = xgbc_smt.score(x_smt_test,y_smt_test)
score_xgbc_smt
```

[84]:
```
0.866481223922114
```

2.3.4 Ada Boost classifier

[85]:
```
abc_smt = AdaBoostClassifier()
abc_smt.fit(x_smt_train,y_smt_train)
```

[85]:
```
AdaBoostClassifier()
```

[86]:
```
score_abc_smt = abc_smt.score(x_smt_test,y_smt_test)
score_abc_smt
```

[86]:
```
0.8437644877144181
```

2.3.5 KNN Classifier

[87]:
```
knnc_smt = KNeighborsClassifier()
knnc_smt.fit(x_smt_train,y_smt_train)
```

[87]:
```
KNeighborsClassifier()
```

[88]:
```
score_knnc_smt = knnc_smt.score(x_smt_test,y_smt_test)
score_knnc_smt
```

[88]:
```
0.799721835883171
```

2.3.6 Logistic Regression

[89]:
```
lrc_smt = LogisticRegression()
lrc_smt.fit(x_smt_train,y_smt_train)
```

```
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

[89]:
```
LogisticRegression()
```

[90]:
```
score_lrc_smt = lrc_smt.score(x_smt_test,y_smt_test)
score_lrc_smt
```

[90]:
```
0.8293926750115902
```

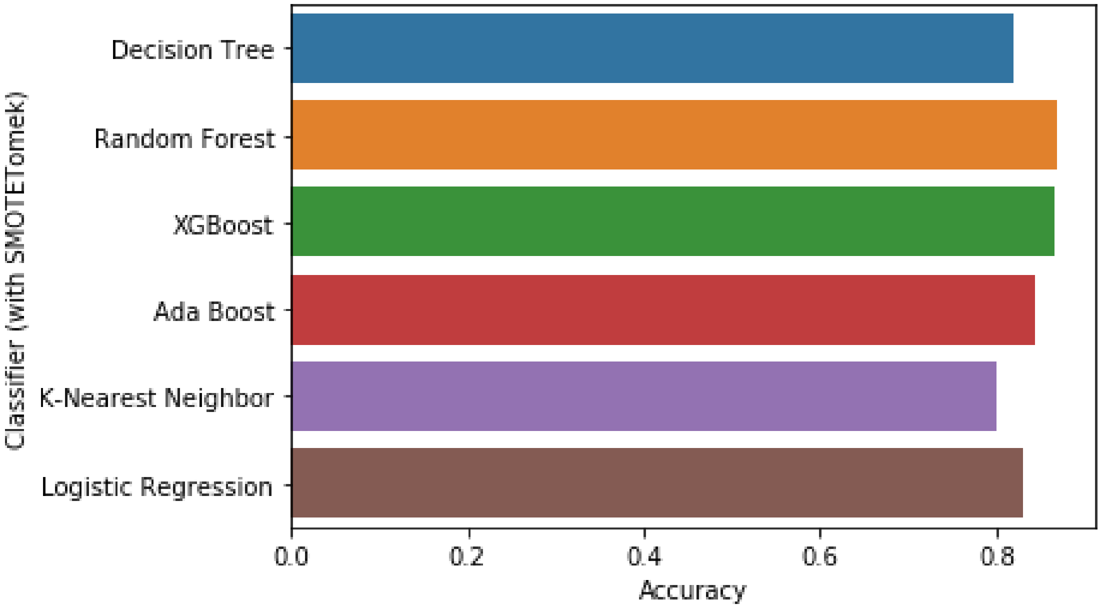
2.3.7 Comparison

[182]:
```
df_smt = pd.DataFrame([['Decision Tree (SMOTETomek)',score_dtc_smt],['Random␣
 ↪Forest (SMOTETomek)',score_rfc_smt],['XGBoost␣
 ↪(SMOTETomek)',score_xgbc_smt],['Ada Boost␣
 ↪(SMOTETomek)',score_abc_smt],['K-Nearest Neighbor␣
 ↪(SMOTETomek)',score_knnc_smt],['Logistic Regression␣
 ↪(SMOTETomek)',score_lrc_smt]],columns=['Classifier', 'Accuracy'])
df_smt
```

[182]:
```
                        Classifier  Accuracy
0        Decision Tree (SMOTETomek)  0.819193
1        Random Forest (SMOTETomek)  0.868799
2              XGBoost (SMOTETomek)  0.866481
3            Ada Boost (SMOTETomek)  0.843764
4   K-Nearest Neighbor (SMOTETomek)  0.799722
5  Logistic Regression (SMOTETomek)  0.829393
```

[94]:
```
sns.barplot(df_smt.Accuracy,df_smt.Classifier])
```

[94]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26430558a48>
```



2.3.8 K-Fold Cross Validation

[95]:

```
kf_smt = KFold(n_splits=5)
result_kf_smt = cross_val_score(rfc_smt, x_smt, y_smt,cv=kf_smt)
result_kf_smt
```

[95]:

```
array([0.80667594, 0.81641168, 0.79763561, 0.91655076, 0.95201669])
```

[96]:

```
result_kf.min()
```

[96]:

```
0.7755681818181818
```

[97]:

```
result_kf.max()
```

[97]:

```
0.8174077578051088
```

2.3.9 Stratified k-fold Cross Validation

[98]:

```
skf_smt = StratifiedKFold()
result_skf_smt = cross_val_score(rfc_smt, x_smt, y_smt,cv=skf_smt)
result_skf_smt
```

[98]:

```
array([0.73504868, 0.80945758, 0.91655076, 0.89916551, 0.90403338])
```

[99]:

```
result_skf_smt.max()
```

[99]:
```
0.9165507649513213
```

[100]:
```
result_skf_smt.min()
```

[100]:
```
0.7350486787204451
```

2.3.10 Leave-One-Out-Cross-Validation(LOOC)

[102]:
```
#loo_smt = LeaveOneOut()
#result_smt = cross_val_score(rfc_smt,x_smt,y_smt,cv=loo_smt)
#result_smt
#result_smt.mean()
```

3 S2. Cross Validation with Near Miss

[104]:
```
from imblearn.under_sampling import NearMiss
nm = NearMiss()
```

[106]:
```
x_nm,y_nm = nm.fit_resample(x,y)
```

[109]:
```
y.value_counts()
```

[109]:
```
0    3892
1    1390
Name: Churn, dtype: int64
```

[108]:
```
y_nm.value_counts()
```

[108]:
```
1    1390
0    1390
Name: Churn, dtype: int64
```

3.1 Hold out Cross Validation

[110]:
```
x_nm_train,x_nm_test,y_nm_train,y_nm_test =␣
  ↪train_test_split(x_nm,y_nm,train_size=.7,random_state=42)
```

[111]:
```
x_nm_train.shape
```

[111]:
```
(1945, 19)
```

[112]:
```
x_nm_test.shape
```

[112]:
```
(835, 19)
```

3.1.1 Decision Tree Classifier

[113]:
```
dtc_nm = DecisionTreeClassifier()
dtc_nm.fit(x_nm_train,y_nm_train)
```

[113]:
```
DecisionTreeClassifier()
```

[114]:
```
score_dtc_nm = dtc_nm.score(x_nm_test,y_nm_test)
```

[115]:
```
score_dtc_nm
```

[115]:
```
0.6299401197604791
```

3.1.2 Random Forest Classifier

[116]:
```
rfc_nm = RandomForestClassifier()
rfc_nm.fit(x_nm_train,y_nm_train)
```

[116]:
```
RandomForestClassifier()
```

[117]:
```
score_rfc_nm = rfc_nm.score(x_nm_test,y_nm_test)
score_rfc_nm
```

[117]:
```
0.6574850299401198
```

3.1.3 XGBoost Classifier

[118]:
```
xgbc_nm = XGBClassifier()
xgbc_nm.fit(x_nm_train,y_nm_train)
```

[118]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, …)
```

[119]:
```
score_xgbc_nm = xgbc_nm.score(x_nm_test,y_nm_test)
score_xgbc_nm
```

[119]:
```
0.6730538922155689
```

3.1.4 Ada Boost classifier

```
[120]: abc_nm = AdaBoostClassifier()
       abc_nm.fit(x_nm_train,y_nm_train)
```

```
[120]: AdaBoostClassifier()
```

```
[121]: score_abc_nm = abc_nm.score(x_nm_test,y_nm_test)
       score_abc_nm
```

```
[121]: 0.6778443113772455
```

3.1.5 KNN Classifier

```
[122]: knnc_nm = KNeighborsClassifier()
       knnc_nm.fit(x_nm_train,y_nm_train)
```

```
[122]: KNeighborsClassifier()
```

```
[123]: score_knnc_nm = knnc_nm.score(x_nm_test,y_nm_test)
       score_knnc_nm
```

```
[123]: 0.578443113772455
```

3.1.6 Logistic Regression

```
[124]: lrc_nm = LogisticRegression()
       lrc_nm.fit(x_nm_train,y_nm_train)
```

```
[124]: LogisticRegression()
```

```
[125]: score_lrc_nm = lrc_nm.score(x_nm_test,y_nm_test)
       score_lrc_nm
```

```
[125]: 0.6634730538922156
```

3.1.7 Comparison

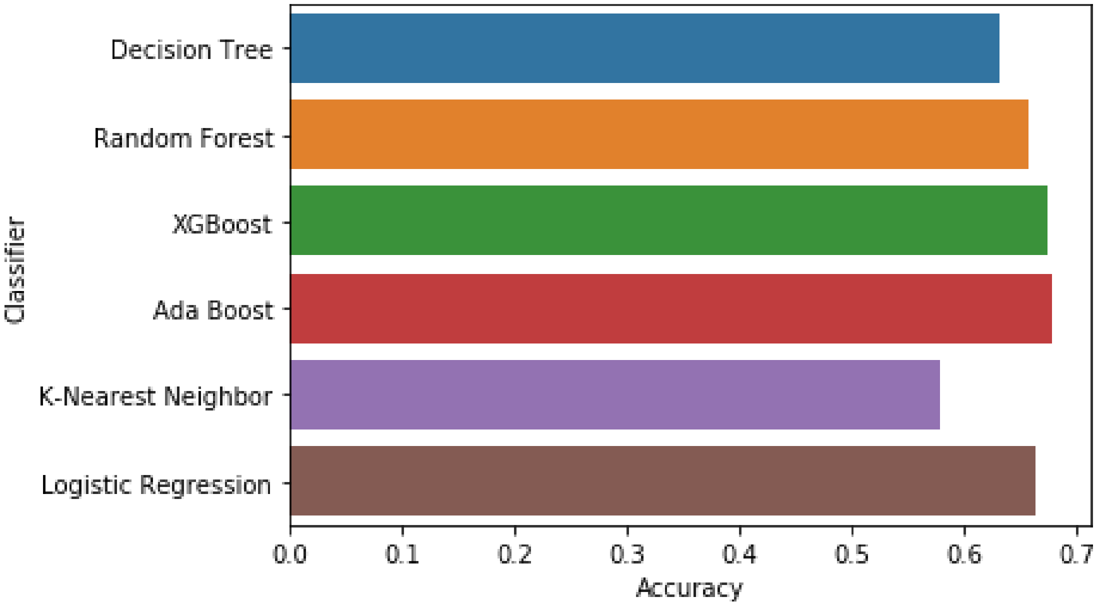
```
[185]: df_nm = pd.DataFrame([['Decision Tree (Near Miss)',score_dtc_nm],['Random␣
         ↪Forest (Near Miss)',score_rfc_nm],['XGBoost (Near␣
         ↪Miss)',score_xgbc_nm],['Ada Boost (Near Miss)',score_abc_nm],['K-Nearest␣
         ↪Neighbor (Near Miss)',score_knnc_nm],['Logistic Regression (Near␣
         ↪Miss)',score_lrc_nm]],columns=['Classifier', 'Accuracy'])
       df_nm
```

```
[185]:                        Classifier  Accuracy
       0        Decision Tree (Near Miss)  0.629940
```

```
1        Random Forest (Near Miss)  0.657485
2              XGBoost (Near Miss)  0.673054
3            Ada Boost (Near Miss)  0.677844
4   K-Nearest Neighbor (Near Miss)  0.578443
5  Logistic Regression (Near Miss)  0.663473
```

[174]:
```
sns.barplot(df_nm.Accuracy,df_nm.Classifier)
```

[174]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2643474cb08>
```



3.1.8 K-Fold Cross Validation

[130]:
```
kf_nm = KFold(n_splits=5)
result_kf_nm = cross_val_score(abc_nm, x_nm, y_nm,cv=kf_nm)
result_kf_nm
```

[130]:
```
array([0.26978417, 0.4676259 , 0.72841727, 0.52697842, 0.47661871])
```

[131]:
```
result_kf_nm.min()
```

[131]:
```
0.2697841726618705
```

[132]:
```
result_kf_nm.max()
```

[132]:
```
0.7284172661870504
```

3.1.9 Stratified k-fold Cross Validation

[133]:
```
skf_nm = StratifiedKFold()
result_skf_nm = cross_val_score(abc_nm, x_nm, y_nm,cv=skf_nm)
result_skf_nm
```

[133]:
```
array([0.50179856, 0.64568345, 0.67625899, 0.69064748, 0.69244604])
```

[134]:
```
result_skf_nm.min()
```

[134]:
```
0.5017985611510791
```

[135]:
```
result_skf_nm.max()
```

[135]:
```
0.6924460431654677
```

3.1.10 Leave-One-Out-Cross-Validation(LOOC)

[136]:
```
#loo_nm = LeaveOneOut()
#result_nm = cross_val_score(abc_nm,x_nm,y_nm,cv=loo_nm)
#result_nm
#result_nm.mean()
```

4 S3. Cross Validation with Random Over Sampler

[138]:
```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler()
```

[139]:
```
x_ros,y_ros = ros.fit_resample(x,y)
```

[140]:
```
y.value_counts()
```

[140]:
```
0    3892
1    1390
Name: Churn, dtype: int64
```

[141]:
```
y_ros.value_counts()
```

[141]:
```
1    3892
0    3892
Name: Churn, dtype: int64
```

4.0.1 Hold Out Cross Validation

[142]:
```
x_ros_train,x_ros_test,y_ros_train,y_ros_test =␣
 ↪train_test_split(x_ros,y_ros,train_size=.7,random_state=42)
```

[145]:
```
x_ros_train.shape
```

[145]:
```
(5448, 19)
```

[146]:
```

```

[146]:
```
(2336, 19)
```

4.0.2 Decision Tree Classifier

[147]:
```
dtc_ros = DecisionTreeClassifier()
dtc_ros.fit(x_ros_train,y_ros_train)
```

[147]:
```
DecisionTreeClassifier()
```

[149]:
```
score_dtc_ros = dtc_ros.score(x_ros_test,y_ros_test)
score_dtc_ros
```

[149]:
```
0.8608732876712328
```

4.0.3 Random Forest Classifier

[150]:
```
rfc_ros = RandomForestClassifier()
rfc_ros.fit(x_ros_train,y_ros_train)
```

[150]:
```
RandomForestClassifier()
```

[151]:
```
score_rfc_ros = rfc_ros.score(x_ros_test,y_ros_test)
score_rfc_ros
```

[151]:
```
0.875
```

4.0.4 XGBoost Classifier

[152]:
```
xgbc_ros = XGBClassifier()
xgbc_ros.fit(x_ros_train,y_ros_train)
```

[152]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, …)
```

[153]:
```
score_xgbc_ros = xgbc_ros.score(x_ros_test,y_ros_test)
score_xgbc_ros
```

[153]:
```
0.8441780821917808
```

4.0.5 Ada Boost classifier

[154]:
```
abc_ros = AdaBoostClassifier()
abc_ros.fit(x_ros_train,y_ros_train)
```

[154]:
```
AdaBoostClassifier()
```

[155]:
```
score_abc_ros = abc_ros.score(x_ros_test,y_ros_test)
score_abc_ros
```

[155]:
```
0.7551369863013698
```

4.0.6 KNN Classifier

[156]:
```
knnc_ros = KNeighborsClassifier()
knnc_ros.fit(x_ros_train,y_ros_train)
```

[156]:
```
KNeighborsClassifier()
```

[157]:
```
score_knnc_ros = knnc_ros.score(x_ros_test,y_ros_test)
score_knnc_ros
```

[157]:
```
0.7277397260273972
```

4.0.7 Logistic Regression

[158]:
```
lrc_ros = LogisticRegression()
lrc_ros.fit(x_ros_train,y_ros_train)
```

```
D:\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

[158]:
```
LogisticRegression()
```

[159]:
```
score_lrc_ros = lrc_ros.score(x_ros_test,y_ros_test)
score_lrc_ros
```

[159]:
```
0.7478595890410958
```

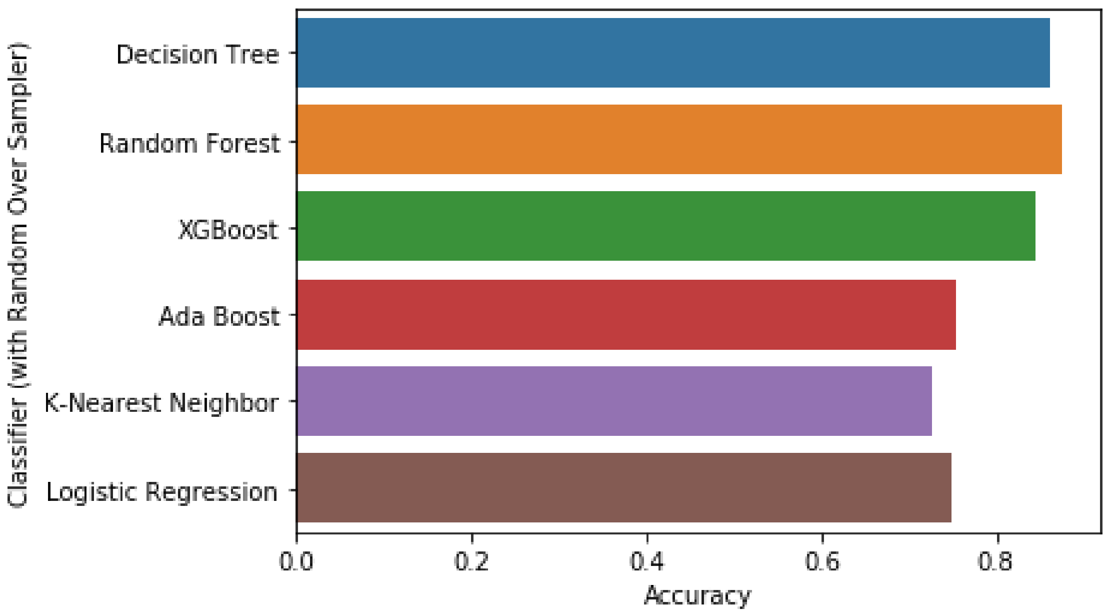
4.0.8 Comparison

[184]:
```
df_ros = pd.DataFrame([['Decision Tree (Random Over␣
 ↪Sampler)',score_dtc_ros],['Random Forest (Random Over␣
 ↪Sampler)',score_rfc_ros],['XGBoost (Random Over␣
 ↪Sampler)',score_xgbc_ros],['Ada Boost (Random Over␣
 ↪Sampler)',score_abc_ros],['K-Nearest Neighbor (Random Over␣
 ↪Sampler)',score_knnc_ros],['Logistic Regression (Random Over␣
 ↪Sampler)',score_lrc_ros]],columns=['Classifier', 'Accuracy'])
df_ros
```

[184]:
```
                                  Classifier  Accuracy
0        Decision Tree (Random Over Sampler)  0.860873
1        Random Forest (Random Over Sampler)  0.875000
2              XGBoost (Random Over Sampler)  0.844178
3            Ada Boost (Random Over Sampler)  0.755137
4   K-Nearest Neighbor (Random Over Sampler)  0.727740
5  Logistic Regression (Random Over Sampler)  0.747860
```

[162]:
```
sns.barplot(df_ros.Accuracy,df_ros.Classifier)
```

[162]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x264348341c8>
```



4.0.9 K-Fold Cross Validation

```
[163]: kf_ros = KFold(n_splits=5)
       result_kf_ros = cross_val_score(abc_ros, x_ros, y_ros,cv=kf_ros)
       result_kf_ros
```

```
[163]: array([0.72254335, 0.72575466, 0.71612075, 0.77007065, 0.68187661])
```

```
[164]: result_kf_ros.min()
```

```
[164]: 0.6818766066838047
```

```
[165]: result_kf_ros.max()
```

```
[165]: 0.7700706486833655
```

4.0.10 Stratified k-fold Cross Validation

```
[166]: skf_ros = StratifiedKFold()
       result_skf_ros = cross_val_score(abc_ros, x_ros, y_ros,cv=skf_ros)
       result_skf_ros
```

```
[166]: array([0.77520873, 0.77135517, 0.77777778, 0.7495183 , 0.77763496])
```

```
[167]: result_skf_ros.min()
```

```
[167]: 0.7495183044315993
```

```
[168]: result_skf_ros.max()
```

```
[168]: 0.7777777777777778
```

4.0.11 Leave-One-Out-Cross-Validation(LOOC)

```
[169]: #loo_ros = LeaveOneOut()
       #result_ros = cross_val_score(abc_ros,x_ros,y_ros,cv=loo_ros)
       #result_ros
       #result_ros.mean()
```

5 Evaluate all results

```
[200]: df_compare = pd.concat([cdf,df_smt,df_nm,df_ros])
       df_compare
```

```
[200]:                             Classifier  Accuracy
       0                         Decision Tree  0.712934
       1                         Random Forest  0.789905
       2                               XGBoost  0.772871
       3                             Ada Boost  0.786751
       4                    K-Nearest Neighbor  0.748896
       5                   Logistic Regression  0.788644
       0            Decision Tree (SMOTETomek)  0.819193
       1            Random Forest (SMOTETomek)  0.868799
       2                  XGBoost (SMOTETomek)  0.866481
       3                Ada Boost (SMOTETomek)  0.843764
       4       K-Nearest Neighbor (SMOTETomek)  0.799722
       5      Logistic Regression (SMOTETomek)  0.829393
       0             Decision Tree (Near Miss)  0.629940
       1             Random Forest (Near Miss)  0.657485
       2                   XGBoost (Near Miss)  0.673054
       3                 Ada Boost (Near Miss)  0.677844
       4        K-Nearest Neighbor (Near Miss)  0.578443
       5       Logistic Regression (Near Miss)  0.663473
       0     Decision Tree (Random Over Sampler)  0.860873
       1     Random Forest (Random Over Sampler)  0.875000
       2           XGBoost (Random Over Sampler)  0.844178
       3         Ada Boost (Random Over Sampler)  0.755137
       4  K-Nearest Neighbor (Random Over Sampler)  0.727740
       5 Logistic Regression (Random Over Sampler)  0.747860
```

```
[201]: df_compare.reset_index(drop=True,inplace=True)
```

[202]:

```
df_compare
```

[202]:

```
                                 Classifier  Accuracy
0                             Decision Tree  0.712934
1                             Random Forest  0.789905
2                                   XGBoost  0.772871
3                                 Ada Boost  0.786751
4                        K-Nearest Neighbor  0.748896
5                       Logistic Regression  0.788644
6                Decision Tree (SMOTETomek)  0.819193
7                Random Forest (SMOTETomek)  0.868799
8                      XGBoost (SMOTETomek)  0.866481
9                    Ada Boost (SMOTETomek)  0.843764
10          K-Nearest Neighbor (SMOTETomek)  0.799722
11         Logistic Regression (SMOTETomek)  0.829393
12                Decision Tree (Near Miss)  0.629940
13                Random Forest (Near Miss)  0.657485
14                      XGBoost (Near Miss)  0.673054
15                    Ada Boost (Near Miss)  0.677844
16           K-Nearest Neighbor (Near Miss)  0.578443
17          Logistic Regression (Near Miss)  0.663473
18      Decision Tree (Random Over Sampler)  0.860873
19      Random Forest (Random Over Sampler)  0.875000
20            XGBoost (Random Over Sampler)  0.844178
21          Ada Boost (Random Over Sampler)  0.755137
22   K-Nearest Neighbor (Random Over Sampler)  0.727740
23  Logistic Regression (Random Over Sampler)  0.747860
```

[195]:

```
df_compare.Accuracy.max()
```

[195]:

```
0.875
```

[206]:

```
# Locating Maximum row values
df_compare.loc[df_compare['Accuracy'].idxmax()]
```

[206]:

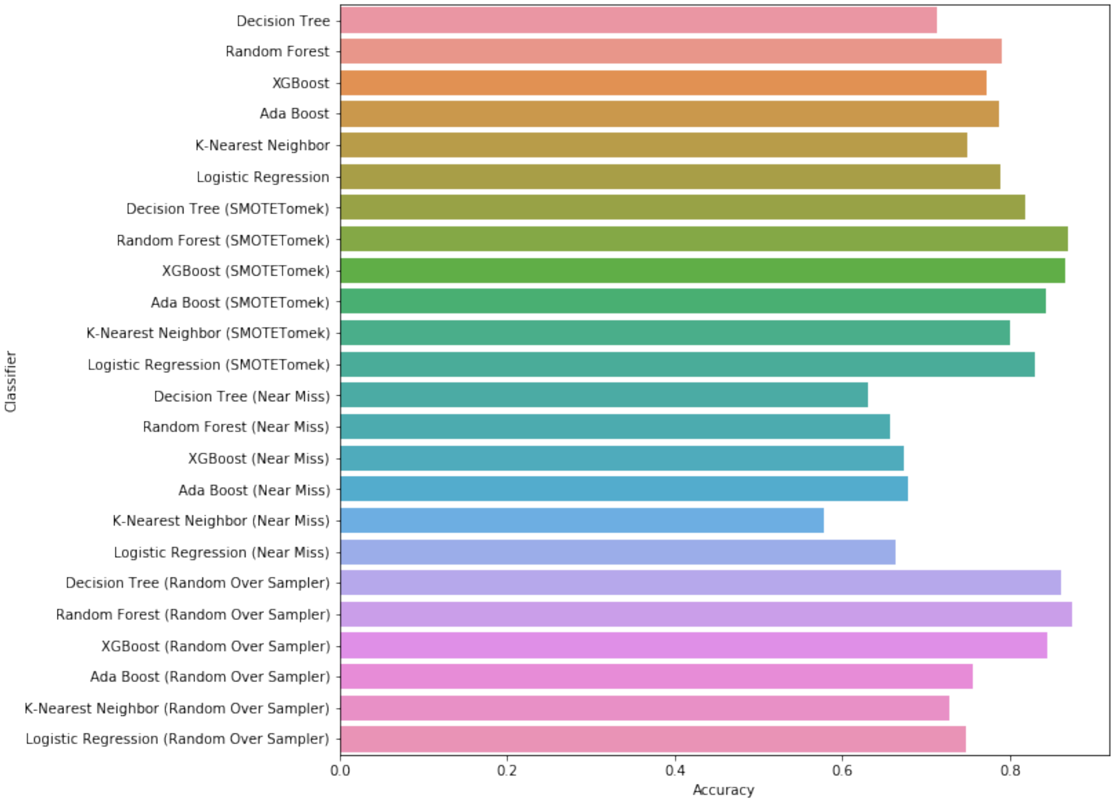
```
Classifier    Random Forest (Random Over Sampler)
Accuracy                                    0.875
Name: 19, dtype: object
```

[193]:

```
plt.figure(figsize=(10,10))
sns.barplot(x='Accuracy',y='Classifier',data=df_compare)
```

[193]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x26436e3c948>
```


Decision Tree
Random Forest
XGBoost
Ada Boost
K-Nearest Neighbor
Logistic Regression
Decision Tree (SMOTETomek)
Random Forest (SMOTETomek)
XGBoost (SMOTETomek)
Ada Boost (SMOTETomek)
K-Nearest Neighbor (SMOTETomek)
Logistic Regression (SMOTETomek)
Decision Tree (Near Miss)
Random Forest (Near Miss)
XGBoost (Near Miss)
Ada Boost (Near Miss)
K-Nearest Neighbor (Near Miss)
Logistic Regression (Near Miss)
Decision Tree (Random Over Sampler)
Random Forest (Random Over Sampler)
XGBoost (Random Over Sampler)
Ada Boost (Random Over Sampler)
K-Nearest Neighbor (Random Over Sampler)
Logistic Regression (Random Over Sampler)
Classifier
0.0
0.2
0.4
0.6
0.8
Accuracy