File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
```

Load the dirty churn data [...]

Load the clean churn rate data [...]

Data Preprocessing

In [5]:
```python
df = df.drop(columns="userid")
```

One Hot Encoder (before tts)

In [6]:
```python
df = pd.get_dummies(data=df, drop_first=True)
df
```

Out[6]:

	churn	age	trivia_played	trivia_shared_results	trivia_view_unlocked	trivia_view_results	cards_share	cards_viewed	cards_helpful	cards_not_helpfu
0	1	22.0	0	0	0	0	0	48	0	
1	1	25.0	0	0	5	0	0	5	0	
2	1	32.0	0	0	0	0	0	49	0	
3	1	26.0	0	0	169	0	0	184	0	
4	1	28.0	0	0	11	0	0	65	0	
...	...	...	...	...	...	...	...	...	...	.
48074	0	42.0	0	0	1	0	0	6	0	
48075	0	38.0	0	0	0	0	0	207	0	
48076	0	20.0	5	0	51	5	0	132	0	
48077	0	27.0	0	0	77	0	0	143	0	
48078	0	33.0	3	2	18	3	0	20	0	

48079 rows × 42 columns

In [7]:
```python
# Don't know how to work OneHotEncoder

# from sklearn.compose import ColumnTransformer
# from sklearn.preprocessing import OneHotEncoder

# encoder = OneHotEncoder(handle_unknown="ignore",)
# df2 = encoder.fit_transform(df)
```

In [8]:
```python
X = df.drop(labels="churn", axis=1).to_numpy()
X
```

Out[8]:
```
array([[22.,  0.,  0., ...,  1.,  0.,  0.],
       [25.,  0.,  0., ...,  0.,  0.,  0.],
       [32.,  0.,  0., ...,  0.,  1.,  0.],
       ...,
       [20.,  5.,  0., ...,  0.,  0.,  0.],
       [27.,  0.,  0., ...,  0.,  1.,  0.],
       [33.,  3.,  2., ...,  0.,  0.,  0.]])
```

In [9]:
```python
y = df["churn"].values
y
```

Out[9]:
```
array([1, 1, 1, ..., 0, 0, 0], dtype=int64)
```

tts

In [10]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Feature Scaling

In [11]:
```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Model Training

In [12]:

```python
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=0)
clf.fit(X_train, y_train)
```

Out[12]:

```
LogisticRegression(random_state=0)
```

In [13]:

```python
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

predictions = clf.predict(X_test)
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.67      0.77      0.72      5620
           1       0.60      0.48      0.53      3996

    accuracy                           0.65      9616
   macro avg       0.64      0.62      0.62      9616
weighted avg       0.64      0.65      0.64      9616

0.6491264559068219
[[4343 1277]
 [2097 1899]]
```

cv

In [14]:

```python
from sklearn.model_selection import cross_val_score

accuracies = cross_val_score(estimator=clf, X=X_train, y=y_train, cv=10, verbose=1) # k = 10
print(accuracies)
print(f"Mean: {accuracies.mean() * 100} %")
print(f"Std: {accuracies.std() * 100} %")
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[0.64803743 0.64231869 0.65115675 0.63520541 0.6474259  0.6450858
 0.65444618 0.66016641 0.65106604 0.63806552]
Mean: 64.72974125619561 %
Std: 0.7123905317268312 %
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    1.2s finished
```

GridSearchCV

In [15]:

```python
from sklearn.model_selection import GridSearchCV

parameters = {"C": [0.001, 0.01, 0.1, 1, 10, 100, 1000], "penalty": ['l1', 'l2']}

grid_search = GridSearchCV(estimator=clf, param_grid=parameters,
                           scoring="accuracy",
                           cv=10, # cv = k = 10
                           n_jobs=-1, # all cpu
                           verbose=1 # print
                          )
grid_search.fit(X_train, y_train)
print(grid_search.best_estimator_)
print(grid_search.best_params_)
print(grid_search.best_score_) # precision
print(grid_search.best_index_)
```

```
Fitting 10 folds for each of 14 candidates, totalling 140 fits
```

```
C:\Users\Perry\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:922: UserWarning: One or more of the test scor
es are non-finite: [       nan 0.6434494         nan 0.64690737        nan 0.64706342
        nan 0.64729741        nan 0.64721942        nan 0.64727143
        nan 0.64721942]
  warnings.warn(
```

```
LogisticRegression(C=1, random_state=0)
{'C': 1, 'penalty': 'l2'}
0.647297412561956
7
```

In [16]:

```python
grid_predictions = grid_search.predict(X_test)

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

print(classification_report(y_test, grid_predictions))
print(accuracy_score(y_test, grid_predictions))
print(confusion_matrix(y_test, grid_predictions))
```

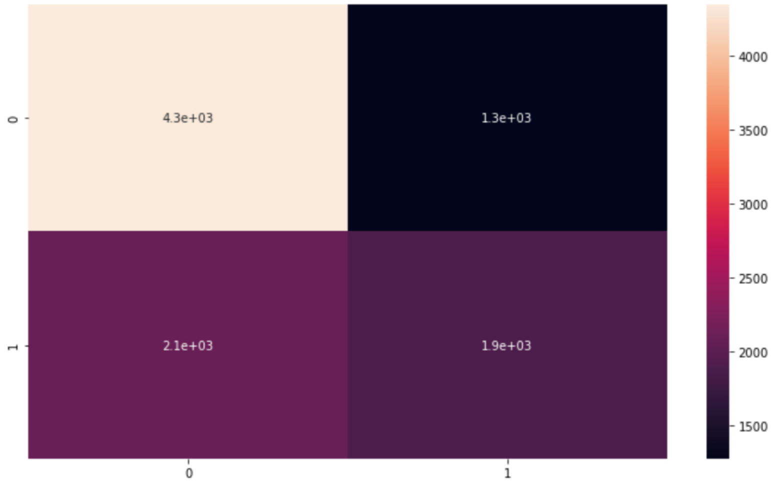
```
              precision    recall  f1-score   support

           0       0.67      0.77      0.72      5620
           1       0.60      0.48      0.53      3996

    accuracy                           0.65      9616
   macro avg       0.64      0.62      0.62      9616
weighted avg       0.64      0.65      0.64      9616

0.6491264559068219
[[4343 1277]
 [2097 1899]]
```

In [17]:
```python
plt.figure(figsize=(12, 7))
sns.heatmap(data=confusion_matrix(y_test, grid_predictions), annot=True);
```



Feature Selection 递归特征消除（Recursive feature elimination） and Re-do Model Training

In [18]:
```python
from sklearn.feature_selection import RFE # 递归特征消除 (Recursive feature elimination)

clf = LogisticRegression()
rfe = RFE(estimator=clf, n_features_to_select=10) # find 10 columns that are most valuable
rfe.fit(X_train, y_train)
```

Out[18]:
```
RFE(estimator=LogisticRegression(), n_features_to_select=10)
```

In [19]:
```python
rfe.support_
```

Out[19]:
```
array([ True,  True, False,  True,  True, False, False, False, False,
       False, False,  True, False, False,  True, False, False,  True,
        True, False, False,  True, False, False, False, False, False,
        True, False, False, False, False, False, False, False, False,
       False, False, False, False, False])
```

In [20]:
```python
rfe.ranking_
```

Out[20]:
```
array([ 1,  1, 20,  1,  1,  9,  2, 13, 28, 12, 14,  1,  3, 22,  1, 10,  4,
        1,  1,  7,  6,  1, 11,  5, 21, 19, 29,  1,  8, 30, 25, 32, 23, 31,
       15, 18, 26, 17, 27, 24, 16])
```

In [21]:
```python
X = df.drop(labels="churn", axis=1)
# using rfe.support_ as selected features
X = X[X.columns[rfe.support_]]
y = df["churn"]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

from sklearn.preprocessing import StandardScaler, MinMaxScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=0)
clf.fit(X_train, y_train)

predictions = clf.predict(X_test)

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

predictions = clf.predict(X_test)
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.67      0.77      0.72      5620
           1       0.59      0.46      0.52      3996

    accuracy                           0.64      9616
   macro avg       0.63      0.62      0.62      9616
weighted avg       0.64      0.64      0.63      9616

0.6424708818635607
[[4344 1276]
 [2162 1834]]
```

In [22]:
```python
clf.coef_
```

Out[22]:
```
array([[-0.15039154,  0.27296565, -0.71876625, -0.32931913,  0.15731766,
         0.1145698 ,  0.10112571,  0.11232106, -0.13146983,  0.08687853]])
```