Import Package

In [ ]:
```
import numpy as np
import re
import pandas as pd
```

Read Data

In [ ]:
```
# read first tab data

df1 = pd.read_excel('Data Scientist Project Data Prediction Model.xlsx', 0)
```

In [ ]:
```
# read second tab data

df2 = pd.read_excel('Data Scientist Project Data Prediction Model.xlsx', 1)
```

Data Pre-processing

Data Pre-processing for first tab data

In [ ]:
```
df1.head()
```

Out[ ]:

	Starting Term	Low Income	School	Race/Ethnicity	Gender	HS GPA	FIRST_GENERATION_DESCR	Retention to Next Fall
0	Fall 2017	No	Dietrich Sch Arts and Sciences	Non-White	F	4.100	First Generation	Retained
1	Fall 2017	No	Dietrich Sch Arts and Sciences	Non-White	M	3.450	Not First Generation	Retained
2	Fall 2017	No	Dietrich Sch Arts and Sciences	Non-White	M	3.767	Not First Generation	Retained
3	Fall 2017	No	Dietrich Sch Arts and Sciences	Non-White	F	4.280	First Generation	Retained
4	Fall 2017	No	Dietrich Sch Arts and Sciences	Non-White	F	4.470	First Generation	Retained

In [ ]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16359 entries, 0 to 16358
Data columns (total 8 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Starting Term           16359 non-null  object 
 1   Low Income              16359 non-null  object 
 2   School                  16359 non-null  object 
 3   Race/Ethnicity          16359 non-null  object 
 4   Gender                  16241 non-null  object 
 5   HS GPA                  16332 non-null  float64
 6   FIRST_GENERATION_DESCR  16359 non-null  object 
 7   Retention to Next Fall  16359 non-null  object 
dtypes: float64(1), object(7)
memory usage: 1022.6+ KB
```

Drop Duplicate Rows

In [ ]:
```
df1 = df1.drop_duplicates(keep='first')
df1 = df1.reset_index(drop=True)
```

Starting Term

In [ ]:
```
# check NA

df1['Starting Term'].isna().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# check value counts

df1['Starting Term'].value_counts()
```

Out[ ]:
```
Starting Term
Fall 2020    3065
Fall 2019    3002
Fall 2018    2901
Fall 2017    2879
Name: count, dtype: int64
```

In [ ]:
```
# create a new feature variable 'year_retained' which is the difference between 2020 and the year when the student started at Pitts
# the reason to choose 2020 is that the target variable is whether or not student will be retained in fall 2021

new_feature = []

for i in df1.index:
    s = df1['Starting Term'][i]
    temp = 2020 - int(re.findall('\d+', s)[0])
    new_feature.append(temp)

df1.insert(0, 'year_retained', new_feature)
```

In [ ]:
```
# drop Starting Term

df1 = df1.drop('Starting Term', axis=1)
```

In [ ]:
```
df1.head()
```

Out[ ]:

	year_retained	Low Income	School	Race/Ethnicity	Gender	HS GPA	FIRST_GENERATION_DESCR	Retention to Next Fall
0	3	No	Dietrich Sch Arts and Sciences	Non-White	F	4.100	First Generation	Retained
1	3	No	Dietrich Sch Arts and Sciences	Non-White	M	3.450	Not First Generation	Retained
2	3	No	Dietrich Sch Arts and Sciences	Non-White	M	3.767	Not First Generation	Retained
3	3	No	Dietrich Sch Arts and Sciences	Non-White	F	4.280	First Generation	Retained
4	3	No	Dietrich Sch Arts and Sciences	Non-White	F	4.470	First Generation	Retained

Low Income

In [ ]:
```
# check NA

df1['Low Income'].isna().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# check value counts

df1['Low Income'].value_counts()
```

Out[ ]:
```
Low Income
No     9601
Yes    2246
Name: count, dtype: int64
```

In [ ]:
```
# encoding

df1['Low Income'] = df1['Low Income'] == 'Yes'
df1['Low Income'] = df1['Low Income'].astype('int64')
```

In [ ]:
```
# check value counts again

df1['Low Income'].value_counts()
```

Out[ ]:
```
Low Income
0    9601
1    2246
Name: count, dtype: int64
```

School

In [ ]:
```
# check NA
```

```
df1['School'].isna().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# check value counts

df1['School'].value_counts()
```

Out[ ]:
```
School
Dietrich Sch Arts and Sciences    7986
Swanson School of Engineering     1683
College of Business Admin         1206
School of Nursing                  599
Sch Computing and Information      372
Sch of Hlth & Rehab Scs              1
Name: count, dtype: int64
```

In [ ]:
```
# it is better to drop the value 'Sch of Hlth & Rehab Scs', since it has only 1 row

df1 = df1[df1['School'] != 'Sch of Hlth & Rehab Scs']
df1 = df1.reset_index(drop=True)
```

In [ ]:
```
# check value counts again

df1['School'].value_counts()
```

Out[ ]:
```
School
Dietrich Sch Arts and Sciences    7986
Swanson School of Engineering     1683
College of Business Admin         1206
School of Nursing                  599
Sch Computing and Information      372
Name: count, dtype: int64
```

Race/Ethnicity

In [ ]:
```
# check NA

df1['Race/Ethnicity'].isna().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# check value counts

df1['Race/Ethnicity'].value_counts()
```

Out[ ]:
```
Race/Ethnicity
White            7208
Non-White        3889
International     580
Unknown           169
Name: count, dtype: int64
```

In [ ]:
```
# we have Unkown value there with 173 rows
# I will treat "Unknown" as a separate category
```

Gender

In [ ]:
```
# check NA

df1['Gender'].isna().sum()
```

Out[ ]:
```
117
```

In [ ]:
```
# fill na with others

df1['Gender'] = df1['Gender'].fillna('other')
```

In [ ]:
```
# check value counts

df1['Gender'].value_counts()
```

Out[ ]:
```
Gender
F        6070
M        5659
other     117
Name: count, dtype: int64
```

HS GPA

In [ ]:
```
# check NA

df1['HS GPA'].isna().sum()
```

Out[ ]:
```
26
```

In [ ]:
```
# drop NA

df1 = df1[df1['HS GPA'].notna()]
df1 = df1.reset_index(drop=True)
```

In [ ]:
```
# check values distribution

df1['HS GPA'].describe()
```

Out[ ]:
```
count    11820.000000
mean         3.989855
std          0.454335
min          0.000000
25%          3.713000
50%          3.992000
75%          4.320000
max          7.774000
Name: HS GPA, dtype: float64
```

In [ ]:
```
# perform outlier detection with  IQR (interquartile range) method

Q1 = df1['HS GPA'].quantile(0.25)
Q3 = df1['HS GPA'].quantile(0.75)
IQR = Q3 - Q1

# Define bounds for the outliers
lower_bound = Q1 - 2 * IQR
upper_bound = Q3 + 2 * IQR

# Identify the outliers
outliers = df1[(df1['HS GPA'] < lower_bound) | (df1['HS GPA'] > upper_bound)]

print(len(outliers))
```

```
62
```

In [ ]:
```
# drop these outliers

df1 = df1.drop(outliers.index, axis=0)
df1 = df1.reset_index(drop=True)
```

In [ ]:
```
# check values distribution again

df1['HS GPA'].describe()
```

Out[ ]:
```
count    11758.000000
mean         4.000039
std          0.426192
min          2.500000
25%          3.719000
50%          3.997500
75%          4.320000
max          5.490000
Name: HS GPA, dtype: float64
```

FIRST_GENERATION_DESCR

In [ ]:
```
# check NA

df1['FIRST_GENERATION_DESCR'].isna().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# check value counts

df1['FIRST_GENERATION_DESCR'].value_counts()
```

Out[ ]:
```
FIRST_GENERATION_DESCR
Not First Generation    7814
Unknown                 2351
First Generation        1593
Name: count, dtype: int64
```

In [ ]:
```
# for the Unknown values in FIRST_GENERATION_DESCR column, it has 2351 rows, drop these is not good idea
# So, instead, I will treat "Unknown" as a separate category
```

Retention to Next Fall (target variable)

In [ ]:
```
# check NA

df1['Retention to Next Fall'].isna().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# check value counts

df1['Retention to Next Fall'].value_counts()
```

Out[ ]:
```
Retention to Next Fall
Retained        10696
Not Retained     1062
Name: count, dtype: int64
```

In [ ]:
```
# encoding

df1['Retention to Next Fall'] = df1['Retention to Next Fall'] == 'Retained'
df1['Retention to Next Fall'] = df1['Retention to Next Fall'].astype('int64')
```

In [ ]:
```
# check value counts

df1['Retention to Next Fall'].value_counts()
```

Out[ ]:
```
Retention to Next Fall
1    10696
0     1062
Name: count, dtype: int64
```

Data Pre-processing for second tab data

In [ ]:
```
df2.head()
```

Out[ ]:

	Starting Term	Low Income	School	Race/Ethnicity	Gender	HS GPA	FIRST_GENERATION_DESCR
0	Fall 2021	No	Dietrich Sch Arts and Sciences	Non-White	M	3.675	Not First Generation
1	Fall 2021	No	Sch Computing and Information	Non-White	M	3.460	Unknown
2	Fall 2021	No	College of Business Admin	Non-White	F	4.020	Not First Generation
3	Fall 2021	No	Dietrich Sch Arts and Sciences	Non-White	F	3.740	Not First Generation
4	Fall 2021	No	Dietrich Sch Arts and Sciences	Non-White	M	3.719	First Generation

In [ ]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4869 entries, 0 to 4868
Data columns (total 7 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Starting Term          4869 non-null   object 
 1   Low Income             4869 non-null   object 
 2   School                 4869 non-null   object 
 3   Race/Ethnicity         4869 non-null   object 
 4   Gender                 4767 non-null   object 
 5   HS GPA                 4859 non-null   float64
 6   FIRST_GENERATION_DESCR  4869 non-null   object 
dtypes: float64(1), object(6)
memory usage: 266.4+ KB
```

Check NA

In [ ]:
```
for i in df2.columns:
    print(i, df2[i].isna().sum())
    print('------------------------------')
```

```
Starting Term 0
------------------------------
Low Income 0
------------------------------
School 0
------------------------------
Race/Ethnicity 0
------------------------------
Gender 102
------------------------------
HS GPA 10
------------------------------
FIRST_GENERATION_DESCR 0
------------------------------
```

Fill NA for Gender

In [ ]:
```
df2['Gender'].isna().sum()
```

Out[ ]:
```
102
```

In [ ]:
```
df2['Gender'] = df2['Gender'].fillna('other')
```

In [ ]:
```
df2['Gender'].isna().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
df2['Gender'].value_counts()
```

Out[ ]:
```
Gender
F        2848
M        1919
other     102
Name: count, dtype: int64
```

Missing Value Imputation for HS GPA

In [ ]:
```
df2['HS GPA'].isna().sum()
```

Out[ ]:
```
10
```

In [ ]:
```
# impute the missing values in HS GPA with the mean value

mean_value = df2['HS GPA'].mean()
df2['HS GPA'].fillna(mean_value, inplace=True)
```

In [ ]:
```
df2['HS GPA'].isna().sum()
```

Out[ ]:
```
0
```

Starting Term

In [ ]:
```
# create a new feature variable 'year_retained' which is the difference between 2021 and the year when the student started at Pitts
# the reason to choose 2021 is that the required target variable is the likelihood that student will be retained in fall 2022

new_feature = []

for i in df2.index:
    s = df2['Starting Term'][i]
    temp = 2021 - int(re.findall('\d+', s)[0])
    new_feature.append(temp)

df2.insert(0, 'year_retained', new_feature)
```

In [ ]:
```
df2 = df2.drop('Starting Term', axis=1)
```

In [ ]:
```
df2.head()
```

Out[ ]:

	year_retained	Low Income	School	Race/Ethnicity	Gender	HS GPA	FIRST_GENERATION_DESCR
0	0	No	Dietrich Sch Arts and Sciences	Non-White	M	3.675	Not First Generation
1	0	No	Sch Computing and Information	Non-White	M	3.460	Unknown
2	0	No	College of Business Admin	Non-White	F	4.020	Not First Generation
3	0	No	Dietrich Sch Arts and Sciences	Non-White	F	3.740	Not First Generation
4	0	No	Dietrich Sch Arts and Sciences	Non-White	M	3.719	First Generation

Low Income

In [ ]:
```
# check value counts

df2['Low Income'].value_counts()
```

Out[ ]:
```
Low Income
No     3995
Yes     874
Name: count, dtype: int64
```

In [ ]:
```
# encoding

df2['Low Income'] = df2['Low Income'] == 'Yes'
df2['Low Income'] = df2['Low Income'].astype('int64')
```

In [ ]:
```
df2['Low Income'].value_counts()
```

Out[ ]:
```
Low Income
0    3995
1     874
Name: count, dtype: int64
```

School

In [ ]:
```
# check value counts

df2['School'].value_counts()
```

Out[ ]:
```
School
Dietrich Sch Arts and Sciences    3386
Swanson School of Engineering      656
College of Business Admin          400
Sch Computing and Information      248
School of Nursing                  179
Name: count, dtype: int64
```

Race/Ethnicity

In [ ]:
```
# check value counts

df2['Race/Ethnicity'].value_counts()
```

Out[ ]:
```
Race/Ethnicity
White            2965
Non-White        1660
International     136
Unknown           108
Name: count, dtype: int64
```

Gender

In [ ]:
```
# check value counts

df2['Gender'].value_counts()
```

Out[ ]:
```
Gender
F        2848
M        1919
other     102
Name: count, dtype: int64
```

HS GPA

In [ ]:
```
# check NA

df2['HS GPA'].describe()
```

Out[ ]:
```
count    4869.000000
mean        4.101894
std         0.427540
min         0.000000
25%         3.842000
50%         4.140000
75%         4.410000
max         6.656000
Name: HS GPA, dtype: float64
```

FIRST_GENERATION_DESCR

In [ ]:
```
# check value counts

df2['FIRST_GENERATION_DESCR'].value_counts()
```

Out[ ]:
```
FIRST_GENERATION_DESCR
Not First Generation    3501
First Generation         711
Unknown                  657
Name: count, dtype: int64
```

Statistical Testing for Feature Selection

Categorical Variable vs Target Variable

In [ ]:
```
# Chi-square test
# This test is used to determine if there is a significant association between two categorical variables

from scipy.stats import chi2_contingency

selected_feature = []

for i in ['Low Income', 'School', 'Race/Ethnicity', 'Gender', 'FIRST_GENERATION_DESCR']:
```

```
    contigency_table = pd.crosstab(df1[i], df1['Retention to Next Fall'])
    chi2, p, dof, expected = chi2_contingency(contigency_table)
    print('p_value of', i, ':', p)
    if p < 0.05:
        selected_feature.append(i)
```

```
p_value of Low Income : 0.004559202414943518
p_value of School : 5.277903431012622e-17
p_value of Race/Ethnicity : 0.38437875595235743
p_value of Gender : 0.02553446807073885
p_value of FIRST_GENERATION_DESCR : 0.056669451561581125
```

In [ ]:
```
selected_feature
```

Out[ ]:
```
['Low Income', 'School', 'Gender']
```

Numerical variable vs binary target

In [ ]:
```
# Point-Biserial Correlation Test

from scipy import stats

r, p = stats.pointbiserialr(df1['Retention to Next Fall'], df1['HS GPA'])

print(p)
```

```
1.1743553679207567e-08
```

In [ ]:
```
from scipy import stats

r, p = stats.pointbiserialr(
    df1['Retention to Next Fall'], df1['year_retained'])

print(p)
```

```
0.3511196800687448
```

In [ ]:
```
selected_feature.append('HS GPA')
```

In [ ]:
```
selected_feature
```

Out[ ]:
```
['Low Income', 'School', 'Gender', 'HS GPA']
```

In [ ]:
```
selected_feature = ['HS GPA', 'Low Income', 'School', 'Gender']
```

New Dataset based on selected features

In [ ]:
```
df1 = df1[selected_feature+['Retention to Next Fall']]
df2 = df2[selected_feature]
```

In [ ]:
```
df1.head()
```

Out[ ]:

	HS GPA	Low Income	School	Gender	Retention to Next Fall
0	4.100	0	Dietrich Sch Arts and Sciences	F	1
1	3.450	0	Dietrich Sch Arts and Sciences	M	1
2	3.767	0	Dietrich Sch Arts and Sciences	M	1
3	4.280	0	Dietrich Sch Arts and Sciences	F	1
4	4.470	0	Dietrich Sch Arts and Sciences	F	1

In [ ]:
```
df2.head()
```

Out[ ]:

	HS GPA	Low Income	School	Gender
0	3.675	0	Dietrich Sch Arts and Sciences	M
1	3.460	0	Sch Computing and Information	M
2	4.020	0	College of Business Admin	F
3	3.740	0	Dietrich Sch Arts and Sciences	F
4	3.719	0	Dietrich Sch Arts and Sciences	M

One-hot Encoding

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [ ]:
```
encoder = OneHotEncoder(sparse = False, dtype=int)

categorical_features = ['School', 'Gender']

one_hot_encoded = encoder.fit_transform(df1[categorical_features])

feature_names = encoder.get_feature_names_out(categorical_features)

df1_encoded = pd.DataFrame(one_hot_encoded, columns=feature_names)

df1 = df1.drop(categorical_features, axis=1)

df1 = pd.concat([df1, df1_encoded], axis=1)
```

```
c:\Users\yang\Desktop\pitt_project\pitt_venv\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

In [ ]:
```
df1.head()
```

Out[ ]:

	HS GPA	Low Income	Retention to Next Fall	School_College of Business Admin	School_Dietrich Sch Arts and Sciences	School_Sch Computing and Information	School_School of Nursing	School_Swanson School of Engineering	Gender_F	Gender_M	Gender_other
0	4.100	0	1	0	1	0	0	0	1	0	0
1	3.450	0	1	0	1	0	0	0	0	1	0
2	3.767	0	1	0	1	0	0	0	0	1	0
3	4.280	0	1	0	1	0	0	0	1	0	0
4	4.470	0	1	0	1	0	0	0	1	0	0

In [ ]:
```
one_hot_encoded_new = encoder.transform(df2[categorical_features])
df2_encoded = pd.DataFrame(one_hot_encoded_new, columns=feature_names)
df2 = df2.drop(categorical_features, axis=1)
df2 = pd.concat([df2, df2_encoded], axis=1)
```

In [ ]:
```
df2.head()
```

Out[ ]:

	HS GPA	Low Income	School_College of Business Admin	School_Dietrich Sch Arts and Sciences	School_Sch Computing and Information	School_School of Nursing	School_Swanson School of Engineering	Gender_F	Gender_M	Gender_other
0	3.675	0	0	1	0	0	0	0	1	0
1	3.460	0	0	0	1	0	0	0	1	0
2	4.020	0	1	0	0	0	0	1	0	0
3	3.740	0	0	1	0	0	0	1	0	0
4	3.719	0	0	1	0	0	0	0	1	0

Scaling

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

In [ ]:
```
scaler = StandardScaler()

df1['HS GPA'] = scaler.fit_transform(df1[['HS GPA']])

df2['HS GPA'] = scaler.transform(df2[['HS GPA']])
```

In [ ]:
```
df1.head()
```

Out[ ]:

	HS GPA	Low Income	Retention to Next Fall	School_College of Business Admin	School_Dietrich Sch Arts and Sciences	School_Sch Computing and Information	School_School of Nursing	School_Swanson School of Engineering	Gender_F	Gender_M	Gender_other
0	0.234554	0	1	0	1	0	0	0	1	0	0
1	-1.290645	0	1	0	1	0	0	0	0	1	0
2	-0.546817	0	1	0	1	0	0	0	0	1	0
3	0.656917	0	1	0	1	0	0	0	1	0	0
4	1.102745	0	1	0	1	0	0	0	1	0	0

In [ ]:
```
df2.head()
```

Out[ ]:

	HS GPA	Low Income	School_College of Business Admin	School_Dietrich Sch Arts and Sciences	School_Sch Computing and Information	School_School of Nursing	School_Swanson School of Engineering	Gender_F	Gender_M	Gender_other
0	-0.762692	0	0	1	0	0	0	0	1	0
1	-1.267181	0	0	0	1	0	0	0	1	0
2	0.046837	0	1	0	0	0	0	1	0	0
3	-0.610172	0	0	1	0	0	0	1	0	0
4	-0.659447	0	0	1	0	0	0	0	1	0

Set up Metric Evaluation

Find the Best cut-point

In [ ]:
```
threshold = []
count = 0

for i in range(1001):
    threshold.append(count)
    count += 0.001
    count = round(count, 3)

def metric_evaluation(threshold, probabilities, y_test):

    result = []

    for k in threshold:

        y_pred = np.where(probabilities[:, 1] > k, 1, 0)

        pred_list = list(y_pred)
        test_list = list(y_test)

        count_false_negative = 0
```

```
        count_false_positive = 0
        count_true_positive = 0
        count_true_negative = 0

        for i, j in list(zip(pred_list, test_list)):
            if i == 0 and j == 1:
                count_false_negative += 1
            elif i == 1 and j == 0:
                count_false_positive += 1
            elif i == 1 and j == 1:
                count_true_positive += 1
            elif i == 0 and j == 0:
                count_true_negative += 1

        accuracy = round((count_true_negative+count_true_positive)/(count_true_negative +
                                                                    count_true_positive+count_false_negative+count_false_positive), 2)

        try:
            precision = round(count_true_positive /
                              (count_true_positive+count_false_positive), 2)
        except:
            precision = None

        try:
            precision2 = round(count_true_negative /
                               (count_true_negative+count_false_negative), 2)
        except:
            precision2 = None

        temp = {'threshold': k, 'TP_num': count_true_positive, 'FP_num': count_false_positive,
                'FN_num': count_false_negative, 'TN_num': count_true_negative,
                'acc': accuracy, 'ppv': precision, 'npv': precision2}

        result.append(temp)

    return result

def best_threshold(result):

    temp = []

    for i in result:
        threshold = i['threshold']
        acc = i['acc']
        npv = i['npv']
        num_FN = i['FN_num']
        num_TN = i['TN_num']
        if npv is not None:
            if acc >= 0.8 and npv >= 0.1 and num_FN+num_TN >= 40 and num_FN+num_TN <=400 :
                temp.append(threshold)

    if temp == []:
        return 'No such threshold exist'
    else:
        return temp
```

In [ ]:
```
best_cut_point = {}
```

Model Selection Metric

In [ ]:
```
# AUC

metric_AUC = {}
```

Model Training - Logistic Regression

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
X = df1.drop('Retention to Next Fall', axis=1)
y = df1['Retention to Next Fall']
```

In [ ]:
```
X.head()
```

Out[ ]:

	HS GPA	Low Income	School_College of Business Admin	School_Dietrich Sch Arts and Sciences	School_Sch Computing and Information	School_School of Nursing	School_Swanson School of Engineering	Gender_F	Gender_M	Gender_other
0	0.234554	0	0	1	0	0	0	1	0	0
1	-1.290645	0	0	1	0	0	0	0	1	0
2	-0.546817	0	0	1	0	0	0	0	1	0
3	0.656917	0	0	1	0	0	0	1	0	0
4	1.102745	0	0	1	0	0	0	1	0	0

In [ ]:
```
y.head()
```

Out[ ]:
```
0    1
1    1
2    1
3    1
4    1
Name: Retention to Next Fall, dtype: int64
```

In [ ]:
```
# Split the data into a training set and a test set
# Here, we take into account imbalanced dataset with "stratify=y"

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

In [ ]:
```python
# Define the logistic regression model

logreg = LogisticRegression()
```

In [ ]:
```python
# Define the hyperparameters

hyperparameters = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga']
}
```

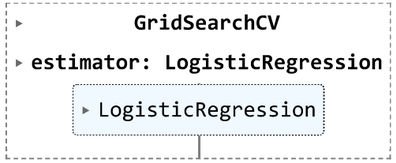
In [ ]:
```python
# Initialize the GridSearchCV model

LR_grid_search = GridSearchCV(logreg, hyperparameters, cv=5, scoring='roc_auc')
```

In [ ]:
```python
# Fit the model

LR_grid_search.fit(X_train, y_train)
```

Out[ ]:



In [ ]:
```python
# Get the best parameters

best_params = LR_grid_search.best_params_
print(f"Best parameters: {best_params}")
```

```
Best parameters: {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
```

In [ ]:
```python
# Get the best score

best_score = LR_grid_search.best_score_
print(f"Best score: {best_score}")
```

```
Best score: 0.5939373139042428
```

In [ ]:
```python
# Predict the labels of the test set

y_pred = LR_grid_search.predict(X_test)
```

In [ ]:
```python
# Calculate the accuracy, precision, recall, F1 score, and ROC AUC score
# Note: Accuracy, precision, recall and F1 score are calculated based on cut-point 0.5, they can not be used for model selection
# For model selection, we can use ROC AUC score since this metric is calculated baded on different cut-point
# For the best cut-point of the selected model,
# we can use the self-defined function written above which takes various factors into consideration,
# such as accuracy, precision, number of predicted not retained stuents

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
```

```
f1 = f1_score(y_test, y_pred)

roc_auc = roc_auc_score(y_test, LR_grid_search.predict_proba(X_test)[:, 1])

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"ROC AUC Score: {roc_auc}")
```

```
Accuracy: 0.9098639455782312
Precision: 0.9098639455782312
Recall: 1.0
F1 Score: 0.9528049866429207
ROC AUC Score: 0.5592818726855934
```

In [ ]:
```
metric_AUC['AUC of Logistic Regression'] = round(roc_auc,4)
metric_AUC
```

Out[ ]:
```
{'AUC of Logistic Regression': 0.5593}
```

In [ ]:
```
LR_probability = LR_grid_search.predict_proba(X_test)
```

In [ ]:
```
# for each list, the first element is the predicted probability of being not retained
# the second element is the predicted probability of being retained

LR_probability
```

Out[ ]:
```
array([[0.05065003, 0.94934997],
       [0.07310281, 0.92689719],
       [0.05003643, 0.94996357],
       ...,
       [0.06456379, 0.93543621],
       [0.11445885, 0.88554115],
       [0.05408489, 0.94591511]])
```

In [ ]:
```
LR_threshold_evaluation = metric_evaluation(threshold, LR_probability, y_test)
```

In [ ]:
```
best_LR_threshold = best_threshold(LR_threshold_evaluation)
```

In [ ]:
```
best_LR_threshold = list(set([round(i,2) for i in best_LR_threshold]))
best_LR_threshold
```

Out[ ]:
```
[0.86, 0.87, 0.88]
```

In [ ]:
```
for i in best_LR_threshold:
    print(LR_threshold_evaluation[int(i*1000)])
    print('-------------------------------------------------------------------------------------------------------------------')
```

```
{'threshold': 0.86, 'TP_num': 2093, 'FP_num': 205, 'FN_num': 47, 'TN_num': 7, 'acc': 0.89, 'ppv': 0.91, 'npv': 0.13}
-------------------------------------------------------------------------------------------------------------------
{'threshold': 0.87, 'TP_num': 1986, 'FP_num': 191, 'FN_num': 154, 'TN_num': 21, 'acc': 0.85, 'ppv': 0.91, 'npv': 0.12}
-------------------------------------------------------------------------------------------------------------------
{'threshold': 0.88, 'TP_num': 1810, 'FP_num': 169, 'FN_num': 330, 'TN_num': 43, 'acc': 0.79, 'ppv': 0.91, 'npv': 0.12}
-------------------------------------------------------------------------------------------------------------------
```

In [ ]:
```
# threshold is the cut-point
# TP_num is the number of retained students who are predicted as retained by the model
# FP_num is the number of not retained studetns who are predicted as retained by the model
# FN_num is the number of retained students who are predicted as not retained by the model
# TN_num is the number of not retained students who are predicted as not retained by the model
# acc is the accuracy calculated on this threshold
# ppv - the probability of correct prediction if the student is predicted as retained by the model
# npv - the probability of correct prediction if the student is predicted as not retained by the model

# For Logistic Regression, I choose 0.87 as the best cut-point from the consideration of accuracy, ppv, npv
# and also consider about the capability of the university in providing the additional support and recources

best_cut_point['Best Cut Point of Logistic Regression'] = 0.87
```

Model Training - Decision Tree

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:
```
# Define the Decision Tree model

dtree = DecisionTreeClassifier(random_state=42)
```

In [ ]:
```
# Define the hyperparameters

hyperparameters = {
    'max_depth': range(1, 21),
    'min_samples_split': range(2, 21)
}
```

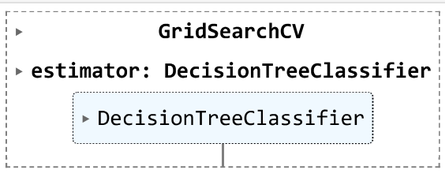
In [ ]:
```
# Initialize the GridSearchCV model

DT_grid_search = GridSearchCV(dtree, hyperparameters, cv=5, scoring='roc_auc')
```

In [ ]:
```
# Fit the model

DT_grid_search.fit(X_train, y_train)
```

Out[ ]:



In [ ]:
```
# Get the best parameters

best_params = DT_grid_search.best_params_
print(f"Best parameters: {best_params}")
```

```
Best parameters: {'max_depth': 4, 'min_samples_split': 2}
```

In [ ]:
```
# Get the best score

best_score = DT_grid_search.best_score_
print(f"Best score: {best_score}")
```

```
Best score: 0.5941607914734688
```

In [ ]:
```
# Predict the labels of the test set

y_pred = DT_grid_search.predict(X_test)
```

In [ ]:
```
# Calculate the accuracy, precision, recall, F1 score, and ROC AUC score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, DT_grid_search.predict_proba(X_test)[:, 1])

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"ROC AUC Score: {roc_auc}")
```

```
Accuracy: 0.9090136054421769
Precision: 0.9097872340425532
Recall: 0.9990654205607477
F1 Score: 0.9523385300668151
ROC AUC Score: 0.5642831952036678
```

In [ ]:
```
metric_AUC['AUC of Decision Tree'] = round(roc_auc,4)
metric_AUC
```

Out[ ]:
```
{'AUC of Logistic Regression': 0.5593, 'AUC of Decision Tree': 0.5643}
```

In [ ]:
```
DT_probability = DT_grid_search.predict_proba(X_test)
DT_threshold_evaluation = metric_evaluation(threshold, DT_probability, y_test)
best_DT_threshold = best_threshold(DT_threshold_evaluation)
```

In [ ]:
```
best_DT_threshold = list(set([round(i,2) for i in best_DT_threshold]))
best_DT_threshold
```

Out[ ]:
```
[0.86, 0.87, 0.88, 0.89, 0.9]
```

In [ ]:
```
for i in best_DT_threshold:
    print(DT_threshold_evaluation[int(i*1000)])
    print('--------------------------------------------------------------------------------------------------------------------------')
```

```
{'threshold': 0.86, 'TP_num': 2107, 'FP_num': 208, 'FN_num': 33, 'TN_num': 4, 'acc': 0.9, 'ppv': 0.91, 'npv': 0.11}
--------------------------------------------------------------------------------------------------------------
{'threshold': 0.87, 'TP_num': 1856, 'FP_num': 170, 'FN_num': 284, 'TN_num': 42, 'acc': 0.81, 'ppv': 0.92, 'npv': 0.13}
--------------------------------------------------------------------------------------------------------------
{'threshold': 0.88, 'TP_num': 1851, 'FP_num': 170, 'FN_num': 289, 'TN_num': 42, 'acc': 0.8, 'ppv': 0.92, 'npv': 0.13}
--------------------------------------------------------------------------------------------------------------
{'threshold': 0.89, 'TP_num': 1851, 'FP_num': 170, 'FN_num': 289, 'TN_num': 42, 'acc': 0.8, 'ppv': 0.92, 'npv': 0.13}
--------------------------------------------------------------------------------------------------------------
{'threshold': 0.9, 'TP_num': 915, 'FP_num': 68, 'FN_num': 1225, 'TN_num': 144, 'acc': 0.45, 'ppv': 0.93, 'npv': 0.11}
--------------------------------------------------------------------------------------------------------------
```

In [ ]:
```python
# threshold is the cut-point
# TP_num is the number of retained students who are predicted as retained by the model
# FP_num is the number of not retained studetns who are predicted as retained by the model
# FN_num is the number of retained students who are predicted as not retained by the model
# TN_num is the number of not retained students who are predicted as not retained by the model
# acc is the accuracy calculated on this threshold
# ppv - the probability of correct prediction if the student is predicted as retained by the model
# npv - the probability of correct prediction if the student is predicted as not retained by the model

# For Decision Tree, I choose 0.87 as the best cut-point from the consideration of accuracy, ppv, npv
# and also consider about the capability of the university in providing the additional support and recources

best_cut_point['Best Cut Point of Decision Tree'] = 0.87
```

Model Training - Random Forest

In [ ]:
```python
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```python
# Define the Random Forest model

rf = RandomForestClassifier(random_state=42)
```

In [ ]:
```python
# Define the hyperparameters

hyperparameters = {
   'n_estimators': [100, 200, 300, 400, 500],
   'max_depth': range(1, 21)
}
```

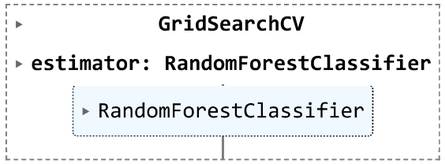
In [ ]:
```python
# Initialize the GridSearchCV model

RF_grid_search = GridSearchCV(rf, hyperparameters, cv=5, scoring='roc_auc')
```

In [ ]:
```python
# Fit the model

RF_grid_search.fit(X_train, y_train)
```

Out[ ]:



In [ ]:
```
# Get the best parameters

best_params = RF_grid_search.best_params_
print(f"Best parameters: {best_params}")
```

```
Best parameters: {'max_depth': 5, 'n_estimators': 100}
```

In [ ]:
```
# Get the best score

best_score = RF_grid_search.best_score_
print(f"Best score: {best_score}")
```

```
Best score: 0.6105682585662284
```

In [ ]:
```
# Predict the labels of the test set

y_pred = RF_grid_search.predict(X_test)
```

In [ ]:
```
# Calculate the accuracy, precision, recall, F1 score, and ROC AUC score
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, RF_grid_search.predict_proba(X_test)[:, 1])

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"ROC AUC Score: {roc_auc}")
```

```
Accuracy: 0.9098639455782312
Precision: 0.9098639455782312
Recall: 1.0
F1 Score: 0.9528049866429207
ROC AUC Score: 0.5761814494798095
```

In [ ]:
```
metric_AUC['AUC of Random Forest'] = round(roc_auc,4)
metric_AUC
```

Out[ ]:
```
{'AUC of Logistic Regression': 0.5593,
 'AUC of Decision Tree': 0.5643,
 'AUC of Random Forest': 0.5762}
```

In [ ]:
```
RF_probability = RF_grid_search.predict_proba(X_test)
RF_threshold_evaluation = metric_evaluation(threshold, RF_probability, y_test)
best_RF_threshold = best_threshold(RF_threshold_evaluation)
```

In [ ]:
```
best_RF_threshold = list(set([round(i,2) for i in best_RF_threshold]))
best_RF_threshold
```

Out[ ]:
```
[0.85, 0.86, 0.87, 0.89, 0.88]
```

In [ ]:
```
for i in best_RF_threshold:
    print(RF_threshold_evaluation[int(i*1000)])
    print('-----------------------------------------------------------------------------------------------------------------------')
```

```
{'threshold': 0.85, 'TP_num': 2106, 'FP_num': 209, 'FN_num': 34, 'TN_num': 3, 'acc': 0.9, 'ppv': 0.91, 'npv': 0.08}
-----------------------------------------------------------------------------------------------------------------------
{'threshold': 0.86, 'TP_num': 2102, 'FP_num': 208, 'FN_num': 38, 'TN_num': 4, 'acc': 0.9, 'ppv': 0.91, 'npv': 0.1}
-----------------------------------------------------------------------------------------------------------------------
{'threshold': 0.87, 'TP_num': 2067, 'FP_num': 202, 'FN_num': 73, 'TN_num': 10, 'acc': 0.88, 'ppv': 0.91, 'npv': 0.12}
-----------------------------------------------------------------------------------------------------------------------
{'threshold': 0.89, 'TP_num': 1671, 'FP_num': 145, 'FN_num': 469, 'TN_num': 67, 'acc': 0.74, 'ppv': 0.92, 'npv': 0.12}
-----------------------------------------------------------------------------------------------------------------------
{'threshold': 0.88, 'TP_num': 1999, 'FP_num': 195, 'FN_num': 141, 'TN_num': 17, 'acc': 0.86, 'ppv': 0.91, 'npv': 0.11}
-----------------------------------------------------------------------------------------------------------------------
```

In [ ]:
```
# threshold is the cut-point
# TP_num is the number of retained students who are predicted as retained by the model
# FP_num is the number of not retained studetns who are predicted as retained by the model
# FN_num is the number of retained students who are predicted as not retained by the model
# TN_num is the number of not retained students who are predicted as not retained by the model
# acc is the accuracy calculated on this threshold
# ppv - the probability of correct prediction if the student is predicted as retained by the model
# npv - the probability of correct prediction if the student is predicted as not retained by the model

# For Random Forest, I choose 0.88 as the best cut-point from the consideration of accuracy, ppv, npv
# and also consider about the capability of the university in providing the additional support and recources

best_cut_point['Best Cut Point of Randomo Forest'] = 0.88
```

Model Training - Catboost (gradient boost)

In [ ]:
```
from catboost import CatBoostClassifier
```

In [ ]:
```
# Define the CatBoost model

params = {'loss_function': 'Logloss',
          'eval_metric': 'AUC',
          'random_state': 42,
          'early_stopping_rounds': 200,
          'iterations': 300,
          'silent': True
          }

cb = CatBoostClassifier(**params)
```

In [ ]:
```
# Define the hyperparameters
hyperparameters = {
    'learning_rate': [0.01, 0.05, 0.1],
```

```
    'depth': [2,4,6,8,10],
    'scale_pos_weight': [0.1,0.4,0.8,1]
}
```

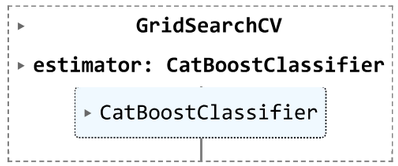
In [ ]:
```
# Initialize the GridSearchCV model

cat_grid_search = GridSearchCV(cb, hyperparameters, cv=5, scoring='roc_auc')
```

In [ ]:
```
# Fit the model

cat_grid_search.fit(X_train, y_train)
```

Out[ ]:



In [ ]:
```
# Get the best parameters

best_params = cat_grid_search.best_params_
print(f"Best parameters: {best_params}")
```

```
Best parameters: {'depth': 4, 'learning_rate': 0.01, 'scale_pos_weight': 0.8}
```

In [ ]:
```
# Get the best score

best_score = cat_grid_search.best_score_
print(f"Best score: {best_score}")
```

```
Best score: 0.6136154560006735
```

In [ ]:
```
# Predict the labels of the test set

y_pred = cat_grid_search.predict(X_test)
```

In [ ]:
```
# Calculate the accuracy, precision, recall, F1 score, and ROC AUC score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, cat_grid_search.predict_proba(X_test)[:, 1])

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"ROC AUC Score: {roc_auc}")
```

```
Accuracy: 0.9098639455782312
Precision: 0.9098639455782312
Recall: 1.0
F1 Score: 0.9528049866429207
ROC AUC Score: 0.5734063657203314
```

In [ ]:
```
metric_AUC['AUC of CatBoost'] = round(roc_auc, 4)
```

In [ ]:
```
metric_AUC
```

Out[ ]:
```
{'AUC of Logistic Regression': 0.5593,
 'AUC of Decision Tree': 0.5643,
 'AUC of Random Forest': 0.5762,
 'AUC of CatBoost': 0.5734}
```

In [ ]:
```
cat_probability = cat_grid_search.predict_proba(X_test)
cat_threshold_evaluation = metric_evaluation(threshold, cat_probability, y_test)
best_cat_threshold = best_threshold(cat_threshold_evaluation)
```

In [ ]:
```
best_cat_threshold = list(set([round(i,2) for i in best_cat_threshold]))
best_cat_threshold
```

Out[ ]:
```
[0.81, 0.82, 0.85, 0.83, 0.84]
```

In [ ]:
```
for i in best_cat_threshold:
    print(cat_threshold_evaluation[int(i*1000)])
    print('----------------------------------------------------------------------------------------------------------------------------')
```

```
{'threshold': 0.81, 'TP_num': 2115, 'FP_num': 210, 'FN_num': 25, 'TN_num': 2, 'acc': 0.9, 'ppv': 0.91, 'npv': 0.07}
----------------------------------------------------------------------------------------------------------------------------
{'threshold': 0.82, 'TP_num': 2094, 'FP_num': 207, 'FN_num': 46, 'TN_num': 5, 'acc': 0.89, 'ppv': 0.91, 'npv': 0.1}
----------------------------------------------------------------------------------------------------------------------------
{'threshold': 0.85, 'TP_num': 1835, 'FP_num': 172, 'FN_num': 305, 'TN_num': 40, 'acc': 0.8, 'ppv': 0.91, 'npv': 0.12}
----------------------------------------------------------------------------------------------------------------------------
{'threshold': 0.83, 'TP_num': 2042, 'FP_num': 198, 'FN_num': 98, 'TN_num': 14, 'acc': 0.87, 'ppv': 0.91, 'npv': 0.12}
----------------------------------------------------------------------------------------------------------------------------
{'threshold': 0.84, 'TP_num': 1991, 'FP_num': 189, 'FN_num': 149, 'TN_num': 23, 'acc': 0.86, 'ppv': 0.91, 'npv': 0.13}
----------------------------------------------------------------------------------------------------------------------------
```

In [ ]:
```
# threshold is the cut-point
# TP_num is the number of retained students who are predicted as retained by the model
# FP_num is the number of not retained studetns who are predicted as retained by the model
# FN_num is the number of retained students who are predicted as not retained by the model
# TN_num is the number of not retained students who are predicted as not retained by the model
# acc is the accuracy calculated on this threshold
# ppv - the probability of correct prediction if the student is predicted as retained by the model
# npv - the probability of correct prediction if the student is predicted as not retained by the model

# For catboost, I choose 0.85 as the best cut-point from the consideration of accuracy, ppv, npv
# and also consider about the capability of the university in providing the additional support and recources

best_cut_point['Best Cut Point of CatBoost'] = 0.85
```

Model Selection and Find Best Cut-point

In [ ]:
```
metric_AUC
```

Out[ ]:
```
{'AUC of Logistic Regression': 0.5593,
 'AUC of Decision Tree': 0.5643,
 'AUC of Random Forest': 0.5762,
 'AUC of CatBoost': 0.5734}
```

In [ ]:
```
best_cut_point
```

Out[ ]:
```
{'Best Cut Point of Logistic Regression': 0.87,
 'Best Cut Point of Decision Tree': 0.87,
 'Best Cut Point of Randomo Forest': 0.88,
 'Best Cut Point of CatBoost': 0.85}
```

In [ ]:
```
# Since Random Forest has the highest AUC score,
# we will select Random Forest as the best model and will use it to make the prediction for second tab dataset
# and the best cut-point of Random Forest is 0.88
```

Find the likelihood for second tab data

In [ ]:
```
df2.head()
```

Out[ ]:

	HS GPA	Low Income	School_College of Business Admin	School_Dietrich Sch Arts and Sciences	School_Sch Computing and Information	School_School of Nursing	School_Swanson School of Engineering	Gender_F	Gender_M	Gender_other
0	-0.762692	0	0	1	0	0	0	0	1	0
1	-1.267181	0	0	0	1	0	0	0	1	0
2	0.046837	0	1	0	0	0	0	1	0	0
3	-0.610172	0	0	1	0	0	0	1	0	0
4	-0.659447	0	0	1	0	0	0	0	1	0

In [ ]:
```
likelihood_result = RF_grid_search.predict_proba(df2)
```

In [ ]:
```
likelihood_result = likelihood_result[::, 1]
```

In [ ]:
```
likelihood_result = list(likelihood_result)
```

In [ ]:
```
likelihood_result = [round(i,3) for i in likelihood_result]
```

In [ ]:
```
temp_df = pd.read_excel('Data Scientist Project Data Prediction Model.xlsx', 1)
```

In [ ]:
```
temp_df.head()
```

Out[ ]:

	Starting Term	Low Income	School	Race/Ethnicity	Gender	HS GPA	FIRST_GENERATION_DESCR
0	Fall 2021	No	Dietrich Sch Arts and Sciences	Non-White	M	3.675	Not First Generation
1	Fall 2021	No	Sch Computing and Information	Non-White	M	3.460	Unknown
2	Fall 2021	No	College of Business Admin	Non-White	F	4.020	Not First Generation
3	Fall 2021	No	Dietrich Sch Arts and Sciences	Non-White	F	3.740	Not First Generation
4	Fall 2021	No	Dietrich Sch Arts and Sciences	Non-White	M	3.719	First Generation

In [ ]:
```
temp_df['Likelihood of being retained'] = likelihood_result
```

In [ ]:
```
temp_df.head()
```

Out[ ]:

	Starting Term	Low Income	School	Race/Ethnicity	Gender	HS GPA	FIRST_GENERATION_DESCR	Likelihood of being retained
0	Fall 2021	No	Dietrich Sch Arts and Sciences	Non-White	M	3.675	Not First Generation	0.892
1	Fall 2021	No	Sch Computing and Information	Non-White	M	3.460	Unknown	0.930
2	Fall 2021	No	College of Business Admin	Non-White	F	4.020	Not First Generation	0.950
3	Fall 2021	No	Dietrich Sch Arts and Sciences	Non-White	F	3.740	Not First Generation	0.902
4	Fall 2021	No	Dietrich Sch Arts and Sciences	Non-White	M	3.719	First Generation	0.892

In [ ]:
```
# with 0.88 cut-point, check how many students will be predicted as not retained
# so that advisors in the university will provide additional support for them

len(temp_df[temp_df['Likelihood of being retained'] <= 0.88])
```

Out[ ]:
```
184
```

In [ ]:
```
# there are 184 studetents be predicted as not retained
# this is a reasonable number,
# and I believe the university should have the capability to provide additional supports for these students
```

In [ ]:
```
temp_df.to_csv('updated_fall_2021_data.csv')
```