1. LIBRARIES & DEPENDENCIES

In [294…
```
pip install lightgbm
```

```
Requirement already satisfied: lightgbm in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (4.1.0)
Requirement already satisfied: numpy in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from lightgbm) (1.2
4.2)
Requirement already satisfied: scipy in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from lightgbm) (1.1
0.1)
Note: you may need to restart the kernel to use updated packages.
```

In [295…
```
pip install catboost
```

```
Requirement already satisfied: catboost in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (1.2.2)
Requirement already satisfied: graphviz in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from catboost)
(0.20.1)
Requirement already satisfied: matplotlib in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from catboost)
(3.7.0)
Requirement already satisfied: numpy>=1.16.0 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from catboo
st) (1.24.2)
Requirement already satisfied: pandas>=0.24 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from catboos
t) (1.5.3)
Requirement already satisfied: scipy in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from catboost) (1.1
0.1)
Requirement already satisfied: plotly in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from catboost) (5.
18.0)
Requirement already satisfied: six in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from catboost) (1.16.
0)
Requirement already satisfied: python-dateutil>=2.8.1 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (fr
om pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from pandas>
=0.24->catboost) (2022.7.1)
Requirement already satisfied: contourpy>=1.0.1 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from mat
plotlib->catboost) (1.0.7)
Requirement already satisfied: cycler>=0.10 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from matplot
lib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from ma
tplotlib->catboost) (4.38.0)
Requirement already satisfied: kiwisolver>=1.0.1 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from ma
tplotlib->catboost) (1.4.4)
Requirement already satisfied: packaging>=20.0 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from matp
lotlib->catboost) (23.0)
Requirement already satisfied: pillow>=6.2.0 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from matplo
tlib->catboost) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from mat
plotlib->catboost) (3.0.9)
Requirement already satisfied: tenacity>=6.2.0 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from plot
ly->catboost) (8.2.3)
Note: you may need to restart the kernel to use updated packages.
```

In [296…
```
!pip install pandas
```

```
WARNING: pip is being invoked by an old script wrapper. This will fail in a future version of pip.
Please see https://github.com/pypa/pip/issues/5599 for advice on fixing the underlying issue.
To avoid this problem you can invoke Python with '-m pip' instead of running pip directly.
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7
is no longer maintained. pip 21.0 will drop support for Python 2.7 in January 2021. More details about Python 2 sup
port in pip can be found at https://pip.pypa.io/en/latest/development/release-process/#python-2-support pip 21.0 wi
ll remove support for this functionality.
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in ./.local/lib/python2.7/site-packages (0.24.2)
Requirement already satisfied: numpy>=1.12.0 in ./.local/lib/python2.7/site-packages (from pandas) (1.16.6)
Requirement already satisfied: python-dateutil>=2.5.0 in ./.local/lib/python2.7/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2011k in ./.local/lib/python2.7/site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in ./.local/lib/python2.7/site-packages (from python-dateutil>=2.5.0->panda
s) (1.16.0)
```

In [297…
```
pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from imbala
nced-learn) (1.24.2)
Requirement already satisfied: scipy>=1.5.0 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from imbalan
ced-learn) (1.10.1)
Requirement already satisfied: scikit-learn>=1.0.2 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from
imbalanced-learn) (1.2.1)
Requirement already satisfied: joblib>=1.1.1 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from imbala
nced-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in ./.conda/envs/sklearn-env/lib/python3.11/site-packages (from
imbalanced-learn) (3.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [298…

```
# basic libraries
import pandas as pd
import numpy as np

# toolkits
from sklearn.impute import  SimpleImputer
from sklearn.pipeline import Pipeline # for pipeline
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import BorderlineSMOTE, ADASYN, RandomOverSampler
from sklearn.preprocessing import FunctionTransformer

# models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

# evaluation
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score

# visualization
import matplotlib.pyplot as plt
import seaborn as sns

# misc
import warnings
warnings.filterwarnings('ignore')

# set_config("display_diagram")
import os # may use it
RANDOM_SEED = 42
```

2. LOAD DATA

Get all 4 files

In [302…

```
## 1: Get Data - Dataset
def get_data(filename):
    '''
    input = "info", "class", "term", "transfer", string
    ouput = dataframe from larc files
    '''

    files = ! ls /data0/larc/*.csv
    list_file = []
    for file in files:
        list_file.append(file)

    if filename == "info":
        df = pd.read_csv(list_file[0])
    elif filename == "class":
```

```
        df = pd.read_csv(list_file[1])
    elif filename == "term":
        df = pd.read_csv(list_file[2])
    elif filename == "transfer":
        df = pd.read_csv(list_file[3])

    return df
```

In [303…

```
# setup major dataset variable
df_term = get_data("term")
df_info = get_data("info")
df_transfer = get_data('transfer')
```

Observations:

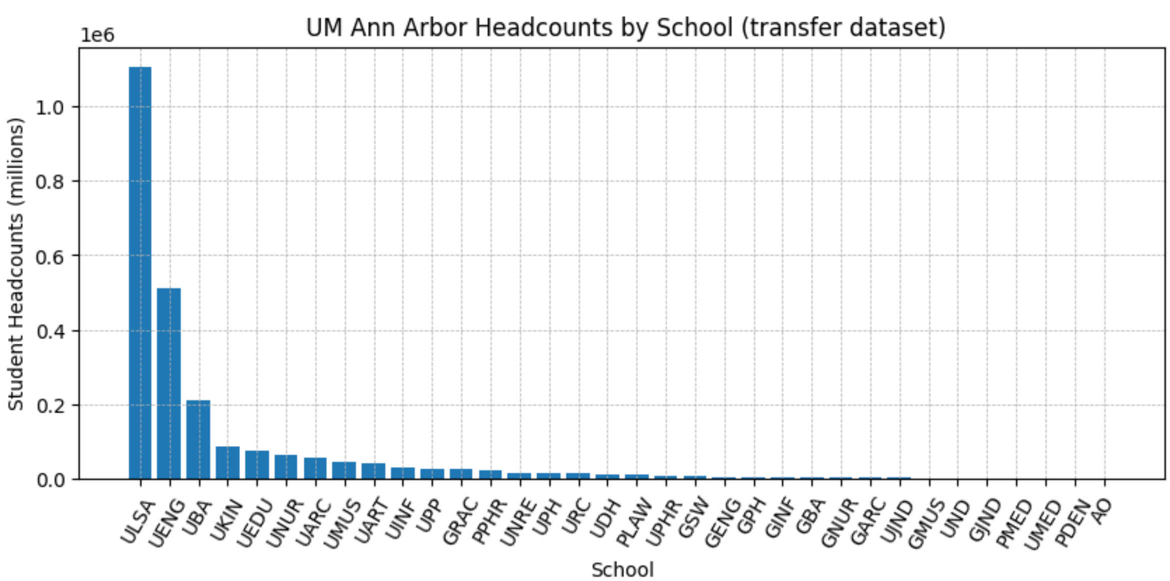
1. df_info - basic info from demographic, high school academic to current placement
2. df_term - term related, GPA, registration status
3. df_class - class related
4. df_transfer - transfer-needed info

3 EXPLORATORY DATA ANALYSIS

3.1 DATA ANAYLYSIS

In [304…

```
# check the school
data_school = df_transfer['ACAD_CRER_CD'].value_counts()
plt.figure(figsize=(10, 4))
plt.bar(data_school.index, data_school.values)
plt.xlabel('School')
plt.ylabel('Student Headcounts (millions)')
plt.title(' UM Ann Arbor Headcounts by School (transfer dataset)')
plt.xticks(rotation=60)
plt.yticks()
plt.grid(True, which='both', axis='both', linestyle='--', linewidth=0.5)
# Show chart
plt.show()
```



We will pick the top 6 schools: ULSA, UENG, UBA, KIN, UEDU and UNUR for transfer learning.

The size varies heavily - UENG is only less than 50% of the ULSA. Same us UBA vs UENG, then UKIN vs UBA. The final 2 UEDU and UNUR would be closer size with UKIN.

3.1 DATA MANIPULATION

In [305…

```
# get School Data - School data with tareget valuable
def get_uschool_data(school, condition):
    """
```

```
    input = school: under school name like "ULSA", "UEGN", string
    input = target: "both", "with", "cancel", string
    output = dataframe ready to use before datapreprocessing
    """
    # convert 'FIRST_TERM_ATTND_BEGIN_YR_MO' to datetime and extract the year
    df_info['FIRST_TERM_ATTND_BEGIN_YR_MO'] = pd.to_datetime(df_info['FIRST_TERM_ATTND_BEGIN_YR_MO'], errors='coerc
    df_info['first_term_year'] = df_info['FIRST_TERM_ATTND_BEGIN_YR_MO'].dt.year

    # ensure 'STDNT_BIRTH_YR' is in the correct format
    df_info['STDNT_BIRTH_YR'] = pd.to_numeric(df_info['STDNT_BIRTH_YR'], errors='coerce')

    # calculate 'age' using vectorized operations
    df_info['age'] = df_info['first_term_year'] - df_info['STDNT_BIRTH_YR']

    # round age to integer where it's not NaN, and handle unrealistic values
    df_info['age'] = df_info['age'].apply(lambda x: round(x) if 12 <= x <= 120 else np.nan)

    # calculate the mean age, ignoring NaN values, and round to the nearest integer
    mean_age = df_info['age'].mean().round()

    # impute NaN values in 'age' with the mean age
    df_info['age'].fillna(mean_age, inplace=True)

    # convert 'age' to an integer
    df_info['age'] = df_info['age'].astype(int)


    # define school from transfer dataset
    df_school = df_transfer[df_transfer['ACAD_CRER_CD'] == school]

    # get unique school ID list
    lst_school = df_school['STDNT_ID'].unique().tolist()

    # define under df from term dataset
    df_U = df_term[(df_term['CRER_LVL_CD'] == 'U')]

    # df for under in that school
    df_uschool = df_U[df_U['STDNT_ID'].isin(lst_school)]

    lst_uschool = df_uschool['STDNT_ID'].unique().tolist()

    if condition == 'both':
        target = ['WITH', 'CNCL']
    elif condition == 'cancel':
        target = ['CNCL']
    elif condition == 'withdraw':
        target = ['WITH']

    def check_first_year(group):
        # Extract the first three entries of 'REG_STAT_CD' for the group
        first_year = group['REG_STAT_CD'].head(4)

        return any(first_year.isin(target))

    # group by 'STDNT_ID' and apply the check_first_year function
    results = df_term.groupby('STDNT_ID').apply(check_first_year)
    lst_dropout = results[results].index.tolist()

    df_labelled = df_info[df_info['STDNT_ID'].isin(lst_dropout)]
    df_nonlabelled = df_info[~df_info['STDNT_ID'].isin(lst_dropout)]


    df_labelled['label'] = 1
    df_nonlabelled['label'] = 0

    df_final = pd.concat([df_labelled, df_nonlabelled])

    df = df_final[df_final['UM_DGR_1_ACAD_CRER_CD'] == school]

    df = df[['PRNT_DEP_NBR_CD',
             'EST_GROSS_FAM_INC_CD',
             'MAX_ACT_COMP_PCTL',
             'MAX_SATI_TOTAL_CALC_SCR',
             'HS_GPA',
             'age',
             'STDNT_SEX_CD',
             'STDNT_ETHNC_GRP_CD',
             'STDNT_CTZN_CNTRY_1_DES',
             'STDNT_CTZN_STAT_SHORT_DES',
             'SNGL_PRNT_IND',
```

```
                'STDNT_INTL_IND',
                'PRNT_MAX_ED_LVL_DES',
                'EST_GROSS_FAM_INC_DES',
                'label'
                ]]

    return df
```

In [306…

```
# get all withdraw dataset ready
df_ULSA_w = get_uschool_data('ULSA', 'withdraw')
df_UENG_w = get_uschool_data('UENG', 'withdraw')
df_UBA_w  = get_uschool_data('UBA' , 'withdraw')
df_UKIN_w = get_uschool_data('UKIN', 'withdraw')
df_UEDU_w = get_uschool_data('UEDU', 'withdraw')
df_UNUR_w = get_uschool_data('UNUR', 'withdraw')
```

In [307…

```
# get all cancel dataset ready
df_ULSA_c = get_uschool_data('ULSA', 'cancel')
df_UENG_c = get_uschool_data('UENG', 'cancel')
df_UBA_c  = get_uschool_data('UBA' , 'cancel')
df_UKIN_c = get_uschool_data('UKIN', 'cancel')
df_UEDU_c = get_uschool_data('UEDU', 'cancel')
df_UNUR_c = get_uschool_data('UNUR', 'cancel')
```

4 PIPELINE: PREPROCESSING & MODELLING

4.1 PART I: FIND OUT THE BEST MODEL AND MEASUREMENT

In [308…

```
df = df_UEDU_w # we could try any of the 6 df above. The smaller dataset, faster to get result

X = df.drop('label', axis=1)  # Replace 'label' with the actual name of label column
y = df['label']  # The label column

# identify numerical and categorical columns
num_cols = ['PRNT_DEP_NBR_CD',
                'EST_GROSS_FAM_INC_CD',
                'MAX_ACT_COMP_PCTL',
                'MAX_SATI_TOTAL_CALC_SCR',
                'HS_GPA',
                'age'
                ]
cat_cols = ['STDNT_SEX_CD',
                'STDNT_ETHNC_GRP_CD',
                'STDNT_CTZN_CNTRY_1_DES',
                'STDNT_CTZN_STAT_SHORT_DES',
                'SNGL_PRNT_IND',
                'STDNT_INTL_IND',
                'PRNT_MAX_ED_LVL_DES',
                'EST_GROSS_FAM_INC_DES'
                ]

# numerical transformer with SimpleImputer and StandardScaler
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # mean strategy for imputation
    ('scaler', StandardScaler())
])

# column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, num_cols),
]
)

# split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = RANDOM_SEED)
# list of (name, model) tuples
# we could feel free to pick the combinations of the models to run,
# some faster and better, some slower and performing not well
models = [
    ('LogisticRegression', LogisticRegression()),
    ('XGBoost', XGBClassifier(use_label_encoder=False, eval_metric='logloss')),
    ('DecisionTree', DecisionTreeClassifier(random_state = RANDOM_SEED)),
    ('RandomForest', RandomForestClassifier(random_state = RANDOM_SEED)),
    ('GradientBoosting', GradientBoostingClassifier(random_state = RANDOM_SEED)), # excellent
    ('SVM', SVC(probability=True, random_state = RANDOM_SEED)), # not that good result
```

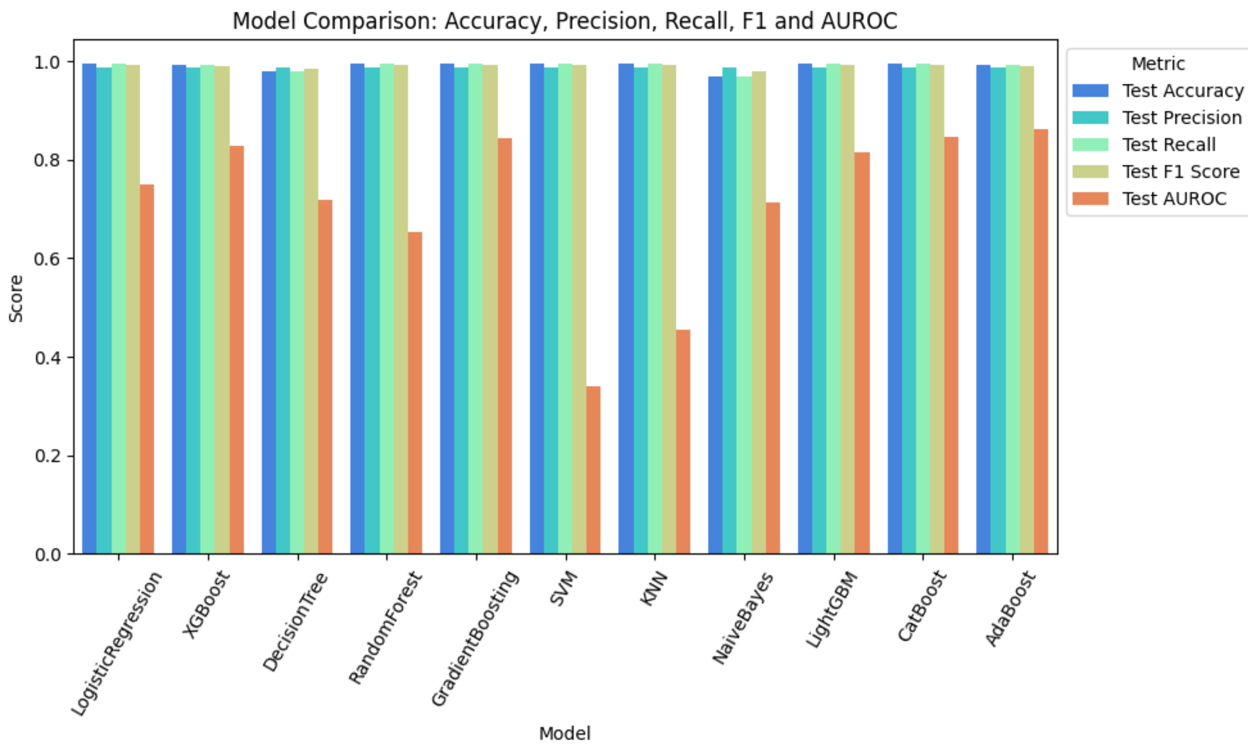
```python
    ('KNN', KNeighborsClassifier()),
    ('NaiveBayes', GaussianNB()), # excellent
    ('LightGBM', LGBMClassifier(silent=True, learning_rate=0.1, random_state = RANDOM_SEED, force_col_wise = True, v
    ('CatBoost', CatBoostClassifier(random_state = RANDOM_SEED,  verbose=False)), # excellent
    ('AdaBoost', AdaBoostClassifier(random_state = RANDOM_SEED)), # excellent
    # ('NeuralNetwork', MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000, activation='relu', solver='adam', ra
] # NN taking too long but not superior performance

# dictionary to hold model scores
model_scores = {}

# iterate over the models list
for name, model in models:

    model_pipeline = ImbPipeline(steps=[

        ('preprocessor', preprocessor),
        ('classifier', model)
    ])

    # cross-validation (make sure to use stratified folds for imbalanced datasets)
    stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state = RANDOM_SEED)
    scores = cross_val_score(model_pipeline, X_train, y_train, cv=stratified_kfold, scoring='roc_auc')

    # training the pipeline
    model_pipeline.fit(X_train, y_train)
    # model_pipeline.fit(X_resampled, y_resampled)

    # predictions
    y_pred = model_pipeline.predict(X_test)

    y_pred_probs = model_pipeline.predict_proba(X_test)[:, 1]

    # evaluation
    report = classification_report(y_test, y_pred, output_dict=True)

    # calculate AUROC
    auroc = roc_auc_score(y_test, y_pred_probs)

        # Store results in model_scores
    model_scores[name] = {
        'Test Accuracy': report['accuracy'],
        'Test Precision': report['weighted avg']['precision'],
        'Test Recall': report['weighted avg']['recall'],
        'Test F1 Score': report['weighted avg']['f1-score'],
        'Test AUROC': auroc
    }

# display results
for model_name, scores in model_scores.items():
    print(f"Model: {model_name}")
    for score_name, score_value in scores.items():
        print(f"{score_name}: {score_value:.4f}")
    print("_"*30)
```

```
Model: LogisticRegression
Test Accuracy: 0.9942
Test Precision: 0.9884
Test Recall: 0.9942
Test F1 Score: 0.9913
Test AUROC: 0.7502
______________________________
Model: XGBoost
Test Accuracy: 0.9922
Test Precision: 0.9884
Test Recall: 0.9922
Test F1 Score: 0.9903
Test AUROC: 0.8291
______________________________
Model: DecisionTree
Test Accuracy: 0.9806
Test Precision: 0.9883
Test Recall: 0.9806
Test F1 Score: 0.9845
Test AUROC: 0.7177
______________________________
Model: RandomForest
Test Accuracy: 0.9942
Test Precision: 0.9884
Test Recall: 0.9942
Test F1 Score: 0.9913
Test AUROC: 0.6527
______________________________
Model: GradientBoosting
Test Accuracy: 0.9942
Test Precision: 0.9884
Test Recall: 0.9942
Test F1 Score: 0.9913
Test AUROC: 0.8441
______________________________
Model: SVM
Test Accuracy: 0.9942
Test Precision: 0.9884
Test Recall: 0.9942
Test F1 Score: 0.9913
Test AUROC: 0.3408
______________________________
Model: KNN
Test Accuracy: 0.9942
Test Precision: 0.9884
Test Recall: 0.9942
Test F1 Score: 0.9913
Test AUROC: 0.4552
______________________________
Model: NaiveBayes
Test Accuracy: 0.9690
Test Precision: 0.9883
Test Recall: 0.9690
Test F1 Score: 0.9785
Test AUROC: 0.7138
______________________________
Model: LightGBM
Test Accuracy: 0.9942
Test Precision: 0.9884
Test Recall: 0.9942
Test F1 Score: 0.9913
Test AUROC: 0.8158
______________________________
Model: CatBoost
Test Accuracy: 0.9942
Test Precision: 0.9884
Test Recall: 0.9942
Test F1 Score: 0.9913
Test AUROC: 0.8470
______________________________
Model: AdaBoost
Test Accuracy: 0.9922
Test Precision: 0.9884
Test Recall: 0.9922
Test F1 Score: 0.9903
Test AUROC: 0.8626
______________________________
```

In [309…

```
# Convert model_scores to a DataFrame
data = []
```

```python
for model_name, scores in model_scores.items():
    for score_name, score_value in scores.items():
        data.append({'Model': model_name, 'Metric': score_name, 'Value': score_value})
df_scores = pd.DataFrame(data)

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Value', hue='Metric', data=df_scores,  palette= 'rainbow')
plt.title('Model Comparison: Accuracy, Precision, Recall, F1 and AUROC')
plt.ylabel('Score')
plt.xlabel('Model')
plt.legend(title='Metric', loc='upper left', bbox_to_anchor=(1, 1))
plt.xticks(rotation=60)
plt.tight_layout()

# Show plot
plt.show()
```



Findings

1). AUROC is the best indicator to measure the performance due to imblanace nature, while Accuracy, Precision, Recall and F1 looks high due to super imblance nature of our dataset for our target variable "Withdraw" or "Cancel" - only 1~2% of the whole dataset.

2). Tree-based XGBoost, Gradient Boosting, LightGBM, Catboost and AdaBoost are the best models across the board

3). Naive Bayes performs well when larger dataset like ULSA and UENG.

4). Our dataset is non-linear by nature

5). Here we use UEDU as an example due to the more significant difference in AUROC and the smallest datset among the top 6 larger ones (faster)

4.2 PART II: FIND OUT THE BEST OVERSAMPLE STRATEGY WITHIN SCHOOL

In [310…

```python
def get_strategy_model_score(df_target):
    df = df_target
    X = df.drop('label', axis=1)  # Replace 'label' with the actual name of label column
    y = df['label']  # The label column
```

```python
# identify numerical and categorical columns
num_cols = ['PRNT_DEP_NBR_CD',
                'EST_GROSS_FAM_INC_CD',
                'MAX_ACT_COMP_PCTL',
                'MAX_SATI_TOTAL_CALC_SCR',
                'HS_GPA',
                'age'
                ]
cat_cols = ['STDNT_SEX_CD',
                'STDNT_ETHNC_GRP_CD',
                'STDNT_CTZN_CNTRY_1_DES',
                'STDNT_CTZN_STAT_SHORT_DES',
                'SNGL_PRNT_IND',
                'STDNT_INTL_IND',
                'PRNT_MAX_ED_LVL_DES',
                'EST_GROSS_FAM_INC_DES'
                ]

# numerical transformer with SimpleImputer and StandardScaler
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # mean strategy for imputation
    ('scaler', StandardScaler())
])

# column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, num_cols)
    ]
)

# split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = RANDOM_SEED)

models = [
    # ('LogisticRegression', LogisticRegression()),
    # ('XGBoost', XGBClassifier(use_label_encoder=False, eval_metric='logloss')),
    # ('DecisionTree', DecisionTreeClassifier(random_state = RANDOM_SEED)),
    # ('RandomForest', RandomForestClassifier(random_state = RANDOM_SEED)),
    ('GradientBoosting', GradientBoostingClassifier(random_state = RANDOM_SEED)), # excellent
    # ('SVM', SVC(probability=True, random_state = RANDOM_SEED)), # not that good result
    # ('KNN', KNeighborsClassifier()),
    # ('NaiveBayes', GaussianNB()), # excellent
    # ('LightGBM', LGBMClassifier(silent=True, learning_rate=0.1, random_state = RANDOM_SEED, force_col_wise =
    # ('CatBoost', CatBoostClassifier(random_state = RANDOM_SEED,  verbose=False)), # excellent
    # ('AdaBoost', AdaBoostClassifier(random_state = RANDOM_SEED)), # excellent
    # ('NeuralNetwork', MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000, activation='relu', solver='adam
] # NN taking too long but not superior performance

strategies = [
    ('adasyn', ADASYN(random_state=RANDOM_SEED)),
    ('smote', SMOTE(sampling_strategy='all', random_state=RANDOM_SEED)),
    ('ros', RandomOverSampler(random_state=RANDOM_SEED)),
    ('borderline_smote', BorderlineSMOTE(random_state=RANDOM_SEED, kind='borderline-1')),
    ('none', FunctionTransformer())  # No oversampling
]

# dictionary to hold model scores
model_scores = {}

# iterate over the models list
for model_name, model in models:
    for strategy_name, strategy in strategies:
        model_pipeline = ImbPipeline(steps=[
            ('preprocessor', preprocessor),
            ('strategy_name', strategy),  # Use the strategy here
            ('classifier', model)
        ])

        # cross-validation (make sure to use stratified folds for imbalanced datasets)
        stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state = RANDOM_SEED)
        scores = cross_val_score(model_pipeline, X_train, y_train, cv=stratified_kfold, scoring='roc_auc')

        # training the pipeline
        model_pipeline.fit(X_train, y_train)
        # model_pipeline.fit(X_resampled, y_resampled)

        # tredictions
        y_pred = model_pipeline.predict(X_test)
```

```python
            y_pred_probs = model_pipeline.predict_proba(X_test)[:, 1]

            # evaluation
            report = classification_report(y_test, y_pred, output_dict=True)

            # calculate AUROC
            auroc = roc_auc_score(y_test, y_pred_probs)

            # store results with both model name and strategy name as a key
            model_scores[(model_name, strategy_name)] = {
                'Test Accuracy': report['accuracy'],
                'Test Precision': report['weighted avg']['precision'],
                'Test Recall': report['weighted avg']['recall'],
                'Test F1 Score': report['weighted avg']['f1-score'],
                'Test AUROC': auroc
            }

    return model_scores
```

In [311…

```python
# display results
def print_model_scores(all_model_scores):
    model_scores = all_model_scores
    for (model_name, strategy_name), scores in model_scores.items():
        print(f"Model: {model_name}, Strategy: {strategy_name}")
        for score_name, score_value in scores.items():
            print(f"{score_name}: {score_value:.4f}")
        print("_"*30)
    return
```

In [312…

```python
score_UEDU = print_model_scores(all_model_scores = get_strategy_model_score(df_target = df_UEDU_w))
score_UEDU
```

```
Model: GradientBoosting, Strategy: adasyn
Test Accuracy: 0.8527
Test Precision: 0.9921
Test Recall: 0.8527
Test F1 Score: 0.9151
Test AUROC: 0.8996
______________________________
Model: GradientBoosting, Strategy: smote
Test Accuracy: 0.8508
Test Precision: 0.9921
Test Recall: 0.8508
Test F1 Score: 0.9140
Test AUROC: 0.8694
______________________________
Model: GradientBoosting, Strategy: ros
Test Accuracy: 0.8430
Test Precision: 0.9920
Test Recall: 0.8430
Test F1 Score: 0.9094
Test AUROC: 0.8739
______________________________
Model: GradientBoosting, Strategy: borderline_smote
Test Accuracy: 0.9070
Test Precision: 0.9879
Test Recall: 0.9070
Test F1 Score: 0.9457
Test AUROC: 0.7755
______________________________
Model: GradientBoosting, Strategy: none
Test Accuracy: 0.9942
Test Precision: 0.9884
Test Recall: 0.9942
Test F1 Score: 0.9913
Test AUROC: 0.8441
______________________________
```

In [313…

```python
score_ULSA = print_model_scores(all_model_scores = get_strategy_model_score(df_target = df_ULSA_w))
score_ULSA
```

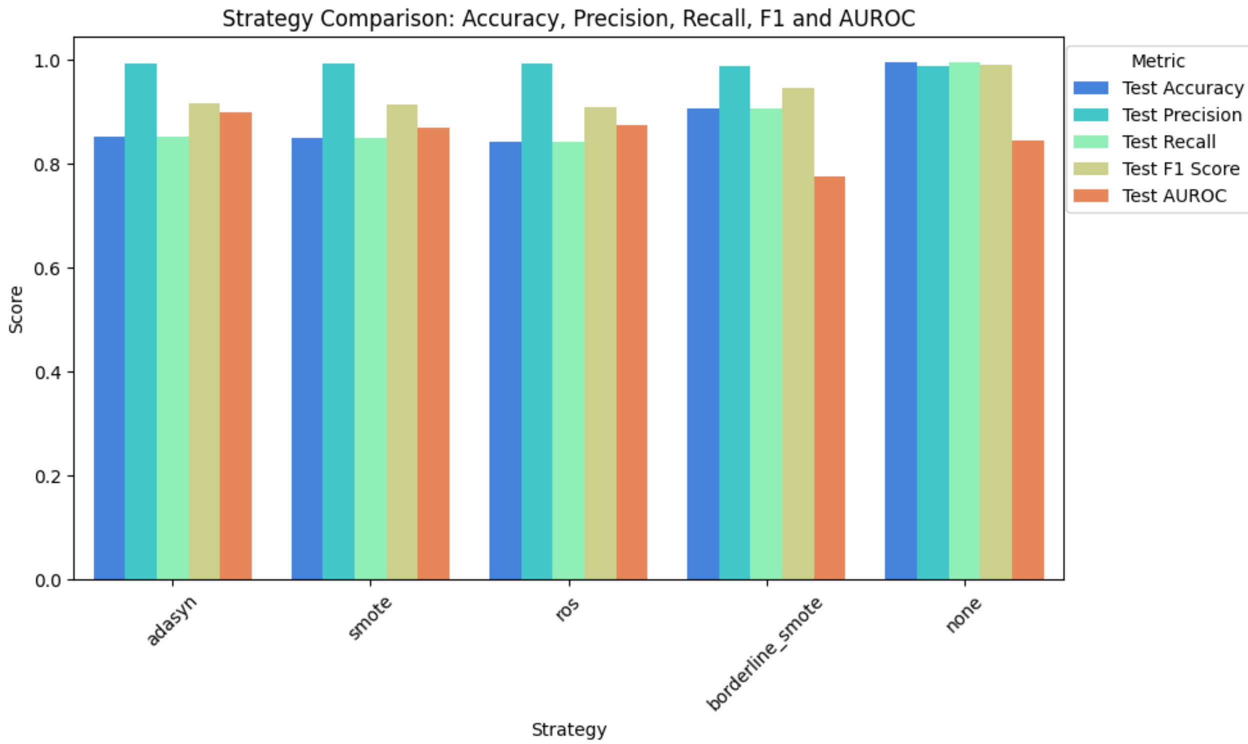
```
Model: GradientBoosting, Strategy: adasyn
Test Accuracy: 0.8194
Test Precision: 0.9522
Test Recall: 0.8194
Test F1 Score: 0.8767
Test AUROC: 0.6544
______________________________
Model: GradientBoosting, Strategy: smote
Test Accuracy: 0.8146
Test Precision: 0.9525
Test Recall: 0.8146
Test F1 Score: 0.8738
Test AUROC: 0.6555
______________________________
Model: GradientBoosting, Strategy: ros
Test Accuracy: 0.7724
Test Precision: 0.9533
Test Recall: 0.7724
Test F1 Score: 0.8475
Test AUROC: 0.6625
______________________________
Model: GradientBoosting, Strategy: borderline_smote
Test Accuracy: 0.8530
Test Precision: 0.9530
Test Recall: 0.8530
Test F1 Score: 0.8968
Test AUROC: 0.6571
______________________________
Model: GradientBoosting, Strategy: none
Test Accuracy: 0.9712
Test Precision: 0.9720
Test Recall: 0.9712
Test F1 Score: 0.9570
Test AUROC: 0.6617
______________________________
```

In [314…

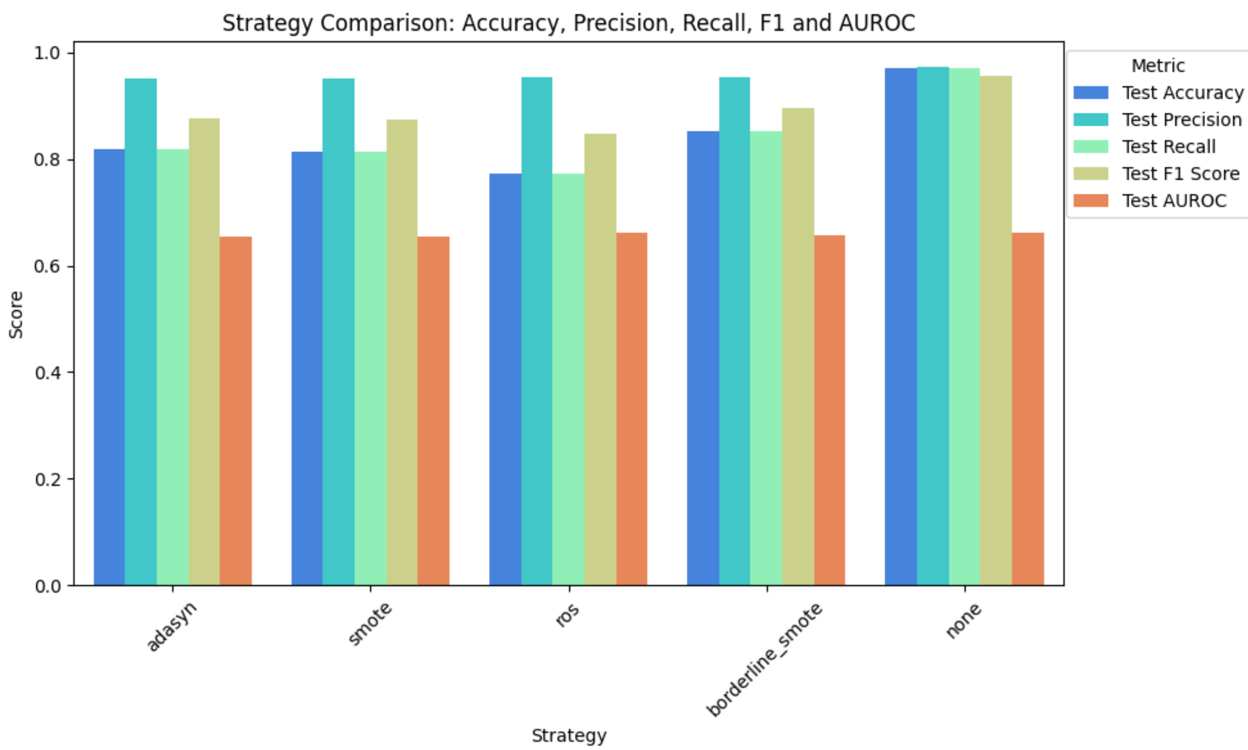
```
def chart_model_comparison(all_model_scores):
    model_scores = all_model_scores
    data = []
    for (model_name, strategy_name), scores in model_scores.items():
        for score_name, score_value in scores.items():
            data.append({'Strategy': strategy_name, 'Metric': score_name, 'Value': score_value})
    df_scores = pd.DataFrame(data)

    plt.figure(figsize=(10, 6))
    sns.barplot(x='Strategy', y='Value', hue='Metric', data=df_scores, palette='rainbow')
    plt.title('Strategy Comparison: Accuracy, Precision, Recall, F1 and AUROC')
    plt.ylabel('Score')
    plt.xlabel('Strategy')
    plt.legend(title='Metric', loc='upper right', bbox_to_anchor=(1.2, 1))
    plt.xticks(rotation=45)

    plt.tight_layout()

    # Show plot
    return plt.show()
```

In [315…

```
chart_UEDU = chart_model_comparison(all_model_scores = get_strategy_model_score(df_target = df_UEDU_w))
chart_UEDU
```




```
In [316… chart_ULSA = chart_model_comparison(all_model_scores = get_strategy_model_score(df_target = df_ULSA_w))
chart_ULSA
```



Findings

1). When the dataset is in smaller size, the oversampling strategies mostly improve the AUROC. e.g. The smaller size dataset School of Education (UEDU) works with AUROC improvement from Adasyn(Adaptive Synthetic Sampling)(0.90), SMOTE(Synthetic Minority Over-sampling Technique)(0.87) and Random Over Sampler (0.87) better than nothing (0.84).

2). When the dataset is in larger size, the oversampling strategies does not significantly improve the AUROC

3). Among the oversampling strategies, Adasyn works the best. SMOTE and Random Over Sampler also good. ADASYN adapts to the density distribution of the minority class, SMOTE creates synthetic samples rather than just duplicating existing ones.Random Over Sampler randomly duplicates examples from the minority class. Our Borderline SMOTE does not perform well to beat no oversampling. Likely the dataset borderline is not easy to tell.

4.3 PART III - TRANSFER LEARNING BETWEEN SCHOOLS W/O OVERSAMPLE STRATEGY

In [317…

```
def data_for_model(df, test_size):

    X = df.drop('label', axis=1)  # Replace 'label' with the actual name of label column
    y = df['label']  # The label column

    # identify numerical and categorical columns
    num_cols = ['PRNT_DEP_NBR_CD',
                'EST_GROSS_FAM_INC_CD',
                'MAX_ACT_COMP_PCTL',
                'MAX_SATI_TOTAL_CALC_SCR',
                'HS_GPA',
                'age'
                ]
    cat_cols = ['STDNT_SEX_CD',
                'STDNT_ETHNC_GRP_CD',
                'STDNT_CTZN_CNTRY_1_DES',
                'STDNT_CTZN_STAT_SHORT_DES',
                'SNGL_PRNT_IND',
                'STDNT_INTL_IND',
                'PRNT_MAX_ED_LVL_DES',
                'EST_GROSS_FAM_INC_DES'
                ]

    # numerical transformer with SimpleImputer and StandardScaler
    numeric_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='mean')),  # mean strategy for imputation
        ('scaler', StandardScaler())
    ])

    # column transformer
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, num_cols),
        ])

    # apply preprocessing to features
    X_preprocessed = preprocessor.fit_transform(X)

    # split the preprocessed data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_size= test_size, random_state=RANDOI

    return (X_train, X_test, y_train, y_test)
```

In [318…

```
def data_for_model(df, test_size):

    X = df.drop('label', axis=1)  # Replace 'label' with the actual name of label column
    y = df['label']  # The label column

    # identify numerical and categorical columns
    num_cols = ['PRNT_DEP_NBR_CD',
                'EST_GROSS_FAM_INC_CD',
                'MAX_ACT_COMP_PCTL',
                'MAX_SATI_TOTAL_CALC_SCR',
                'HS_GPA',
                'age'
                ]
    cat_cols = ['STDNT_SEX_CD',
                'STDNT_ETHNC_GRP_CD',
                'STDNT_CTZN_CNTRY_1_DES',
                'STDNT_CTZN_STAT_SHORT_DES',
                'SNGL_PRNT_IND',
                'STDNT_INTL_IND',
                'PRNT_MAX_ED_LVL_DES',
                'EST_GROSS_FAM_INC_DES'
                ]

    # numerical transformer with SimpleImputer and StandardScaler
```

```
    numeric_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='mean')),  # mean strategy for imputation
        ('scaler', StandardScaler())
    ])

    # column transformer
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, num_cols),
        ])

    # apply preprocessing to features
    X_preprocessed = preprocessor.fit_transform(X)

    # split the preprocessed data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_size= test_size, random_state=RANDOI

    return (X_train, X_test, y_train, y_test)
```

In [319…
```
def transfer_model(df_A, df_B, size_A, size_B):
    # load and preprocess Dataset A
    X_train_A, X_test_A, y_train_A, y_test_A = data_for_model(df = df_A, test_size = size_A)

    # train model on Dataset A
    model = GradientBoostingClassifier(random_state=RANDOM_SEED)
    model.fit(X_train_A, y_train_A)

    # load and preprocess Dataset B
    X_train_B, X_test_B, y_train_B, y_test_B = data_for_model(df = df_B, test_size = size_B)

    y_pred_B = model.predict(X_test_B)

    y_pred_probs_B = model.predict_proba(X_test_B)[:, 1]

    stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state = RANDOM_SEED)
    scores = cross_val_score(model, X_train_B, y_train_B, cv=stratified_kfold, scoring='roc_auc')

    # calculate AUROC
    auroc_B = roc_auc_score(y_test_B, y_pred_probs_B)

    return auroc_B
```

In [320…
```
def table_transfer(list_school_df, size):
    num_schools = len(list_school_df)
    # initialize an empty DataFrame with proper dimensions and labels
    df_results = pd.DataFrame(index=[f'School_{i+1}' for i in range(num_schools)],
                              columns=[f'School_{j+1}' for j in range(num_schools)])

    for i, df_A in enumerate(list_school_df):
        for j, df_B in enumerate(list_school_df):

            # call the transfer_model function with df_A and df_B
            result = transfer_model(df_A, df_B, size, size)

            # store the result in the DataFrame
            df_results.iloc[i, j] = result

    return df_results
```

In [321…
```
list_school_df_w = [df_UEDU_w, df_UNUR_w, df_UKIN_w, df_UBA_w, df_UENG_w, df_ULSA_w]
df_result_w_30 = table_transfer(list_school_df_w, 0.3)
df_result_w_30
```

Out[321]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.792328	0.745355	0.589556	0.533835	0.655616	0.584723
School_2	0.775553	0.706672	0.628318	0.562803	0.593763	0.554865
School_3	0.888557	0.685154	0.621841	0.616561	0.658479	0.596718
School_4	0.786476	0.600473	0.552357	0.608837	0.621271	0.612151
School_5	0.852926	0.683004	0.620828	0.587599	0.686984	0.635783
School_6	0.883225	0.674152	0.68049	0.613919	0.699243	0.677151

In [322…
```
list_school_df_c = [df_UEDU_c, df_UNUR_c, df_UKIN_c, df_UBA_c, df_UENG_c, df_ULSA_c]
df_result_c_30 = table_transfer(list_school_df_c, 0.3)
df_result_c_30
```

Out[322]:

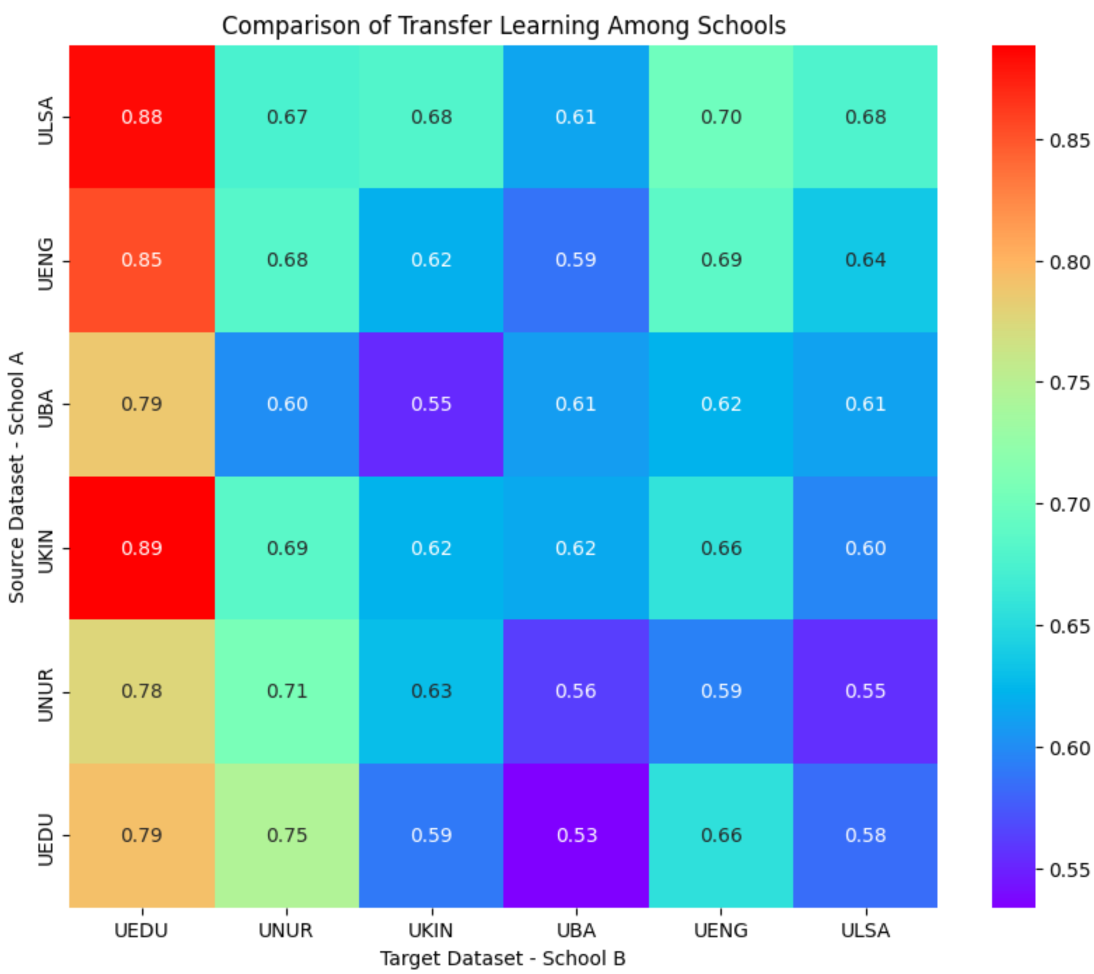
	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.545223	0.540386	0.465665	0.538545	0.544223	0.537009
School_2	0.519045	0.585949	0.481703	0.469685	0.53792	0.522597
School_3	0.65373	0.597061	0.639121	0.472782	0.573884	0.577626
School_4	0.650654	0.603304	0.563424	0.644072	0.583235	0.599683
School_5	0.722186	0.615367	0.564978	0.544646	0.614239	0.610825
School_6	0.760798	0.613629	0.616673	0.551483	0.596383	0.649786

In [323…

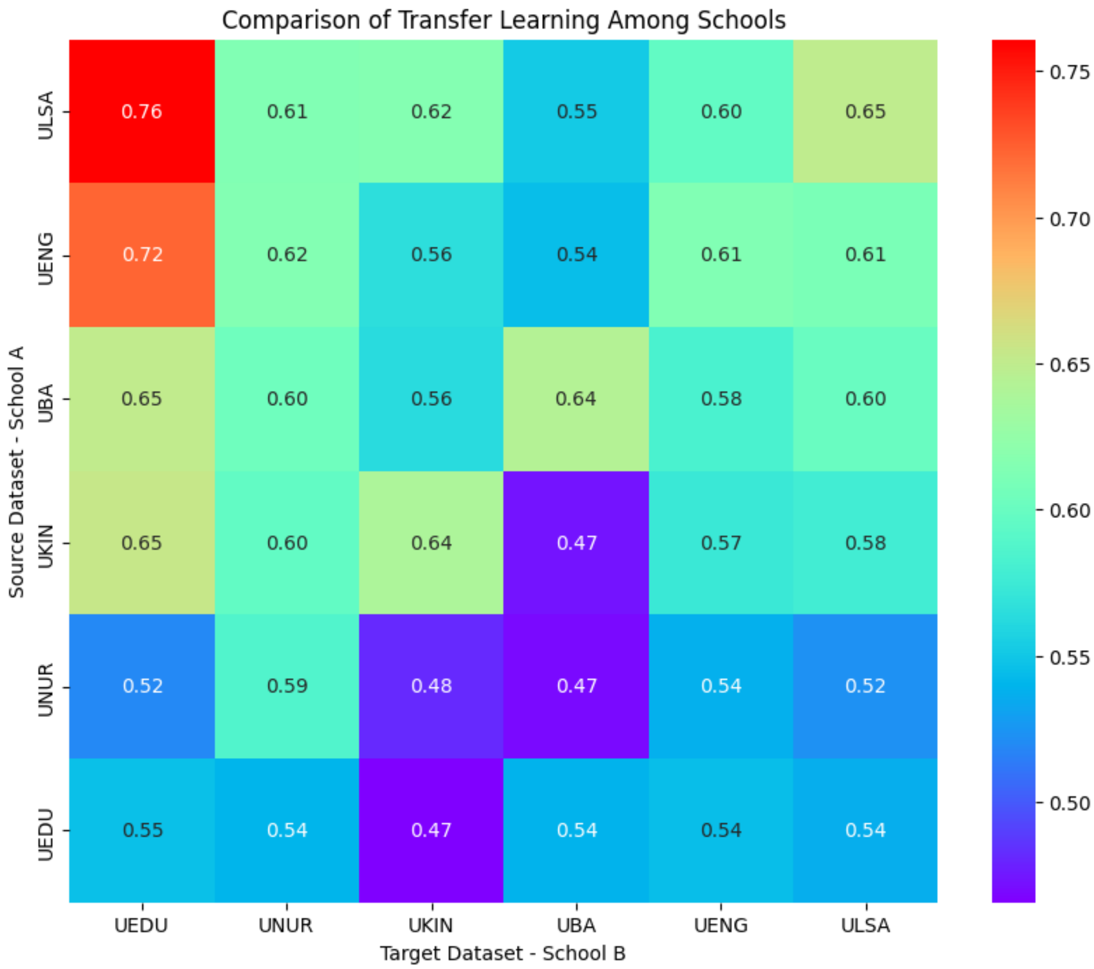
```
def chart_transfer_learning(df, list_schools):
    schools = list_schools
    # convert df_result to a numpy array and then reshape it to a 6x6 matrix
    matrix = df.to_numpy().reshape(len(schools), len(schools))

    # create a new DataFrame with the reshaped matrix
    df_matrix = pd.DataFrame(matrix, index=schools, columns=schools)

    for col in df_matrix.columns:
        df_matrix[col] = pd.to_numeric(df_matrix[col], errors='coerce')

    # create a heatmap
    plt.figure(figsize=(10, 8))
    ax = sns.heatmap(df_matrix, annot=True, cmap='rainbow', fmt=".2f")
    plt.title('Comparison of Transfer Learning Among Schools')
    plt.xlabel('Target Dataset - School B')
    plt.ylabel('Source Dataset - School A')
    # Invert the y-axis
    ax.invert_yaxis()
    return plt.show()
```

In [324…

```
list_schools = ['UEDU', 'UNUR', 'UKIN', 'UBA', 'UENG', 'ULSA']
df = df_result_w_30
chart_w_30 = chart_transfer_learning(df, list_schools)
chart_w_30
```



In [325…
```
df = df_result_c_30
chart_c_30 = chart_transfer_learning(df, list_schools)
chart_c_30
```



In [350…
```
df_result_w_30_new = df_result_w_30.subtract(df_result_w_30.values.diagonal(), axis=0)
df_result_w_30_new
```

Out[350]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.0	-0.046972	-0.202771	-0.258493	-0.136712	-0.207604
School_2	0.068881	0.0	-0.078354	-0.143868	-0.112909	-0.151807
School_3	0.266715	0.063313	0.0	-0.005281	0.036638	-0.025123
School_4	0.177639	-0.008364	-0.056479	0.0	0.012434	0.003315
School_5	0.165942	-0.003981	-0.066156	-0.099386	0.0	-0.051201
School_6	0.206074	-0.002999	0.003339	-0.063233	0.022092	0.0

In [351…
```
df_result_c_30_new = df_result_c_30.subtract(df_result_c_30.values.diagonal(), axis=0)
df_result_c_30_new
```

Out[351]:

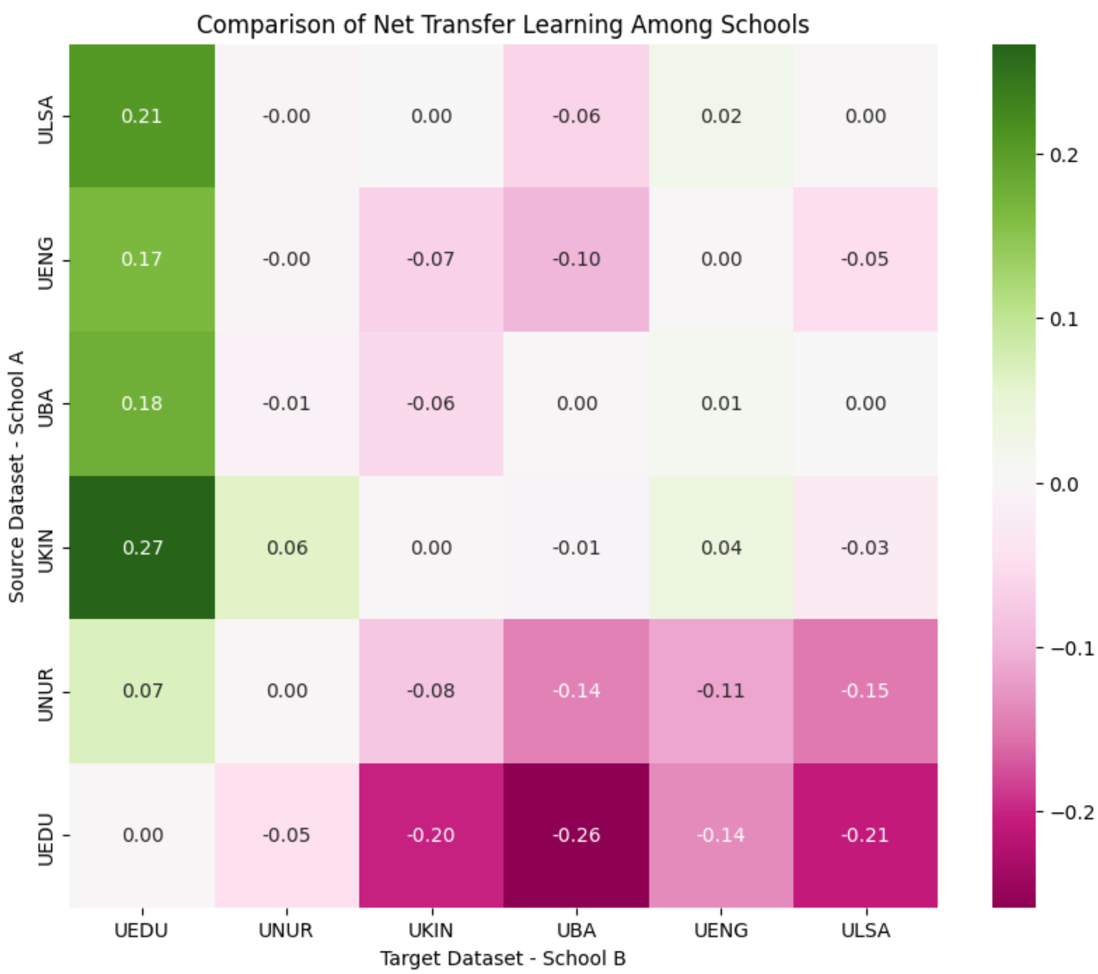
	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.0	-0.004836	-0.079558	-0.006678	-0.001	-0.008213
School_2	-0.066904	0.0	-0.104246	-0.116264	-0.048028	-0.063352
School_3	0.01461	-0.04206	0.0	-0.166338	-0.065236	-0.061495
School_4	0.006582	-0.040768	-0.080648	0.0	-0.060837	-0.044389
School_5	0.107947	0.001128	-0.049261	-0.069593	0.0	-0.003414
School_6	0.111012	-0.036158	-0.033113	-0.098303	-0.053403	0.0

In [367…
```
def chart_net_transfer_learning(df, list_schools, colormap):
    # ensure the DataFrame has schools as both rows and columns
    if df.shape[0] != len(list_schools) or df.shape[1] != len(list_schools):
```

```python
        raise ValueError("DataFrame must have the same number of rows and columns as the list of schools")

    # assign the list of schools as index and columns of the DataFrame
    df.index = list_schools
    df.columns = list_schools

    # convert all columns to numeric values
    for col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')

    # create a heatmap
    plt.figure(figsize=(10, 8))
    ax = sns.heatmap(df, annot=True, cmap=colormap, fmt=".2f")
    plt.title('Comparison of Net Transfer Learning Among Schools')
    plt.xlabel('Target Dataset - School B')
    plt.ylabel('Source Dataset - School A')
    ax.invert_yaxis()
    plt.show()
```

In [368…

```python
list_schools = ['UEDU', 'UNUR', 'UKIN', 'UBA', 'UENG', 'ULSA']
df = df_result_w_30_new
colormap = 'PiYG'
chart_result_w_30_new = chart_net_transfer_learning(df , list_schools, colormap)
```



In [369…

```python
df = df_result_c_30_new
colormap = 'PiYG'
chart_result_c_30_new = chart_net_transfer_learning(df , list_schools, colormap)
```

1). Pick one of the best models: GradientBoosting as our model to transfer the learning, it did shows some successful transfer learning UKIN to UEDU for target variable "Withdraw" 0.89 in AUROC. Also from ULSA to UEDU 0.88 and UENG to UEDUE 0.85. In the "Cancel" cases, the best transfer learning is from ULSA to UEDU 0.79 and from UENG to UEDU 0.72.

2). Dataset size UEDU < UNUR < UKIN < UBA < UENG < ULSA. We could see a trend that the larger the original dataset is in comparison to the target dataset, the performance is better than opposite direction. However, some specific case did still show against the trend (From UBSA to UKIN in "withdraw" case).

3). Overall, Smaller dataset shows the best performance in general to be transferred from learger dataset. The smallest datset UEDU successfully got AUROC boostup if transfer learning from the school with larger dataset.

4). School-specific factor like UBSA still shows resistance of success from transfer learning

4.4 PART IV - TRANSFER LEARNING BETWEEN SCHOOLS WITH DIFFERENT SPLIT SIZES

In [326…

```
def chart_compare_transfer_learning(df, list_schools, colormap):
    schools = list_schools
    # convert df_result to a numpy array and then reshape it to a 6x6 matrix
    matrix = df.to_numpy().reshape(len(schools), len(schools))

    # create a new DataFrame with the reshaped matrix
    df_matrix = pd.DataFrame(matrix, index=schools, columns=schools)

    for col in df_matrix.columns:
        df_matrix[col] = pd.to_numeric(df_matrix[col], errors='coerce')

    # create a heatmap
    plt.figure(figsize=(10, 8))
```

```python
    ax = sns.heatmap(df_matrix, annot=True, cmap=colormap, fmt=".2f")
    plt.title('Comparison of Transfer Learning Among Schools')
    plt.xlabel('Target Dataset - School B')
    plt.ylabel('Source Dataset - School A')
    # invert the y-axis
    ax.invert_yaxis()
    return plt.show()
```

In [327…

```python
list_school_df_w = [df_UEDU_w, df_UNUR_w, df_UKIN_w, df_UBA_w, df_UENG_w, df_ULSA_w]
df_result_w_20 = table_transfer(list_school_df_w, 0.2)
df_result_w_20
```

Out[327]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.844055	0.684657	0.663451	0.567224	0.667596	0.584776
School_2	0.758609	0.700299	0.715295	0.619647	0.616617	0.547174
School_3	0.847303	0.615125	0.630168	0.595699	0.662313	0.59616
School_4	0.69883	0.579026	0.502192	0.58831	0.571826	0.586594
School_5	0.78655	0.669695	0.665365	0.604123	0.69162	0.631851
School_6	0.824237	0.615098	0.710645	0.618641	0.707644	0.661655

In [328…

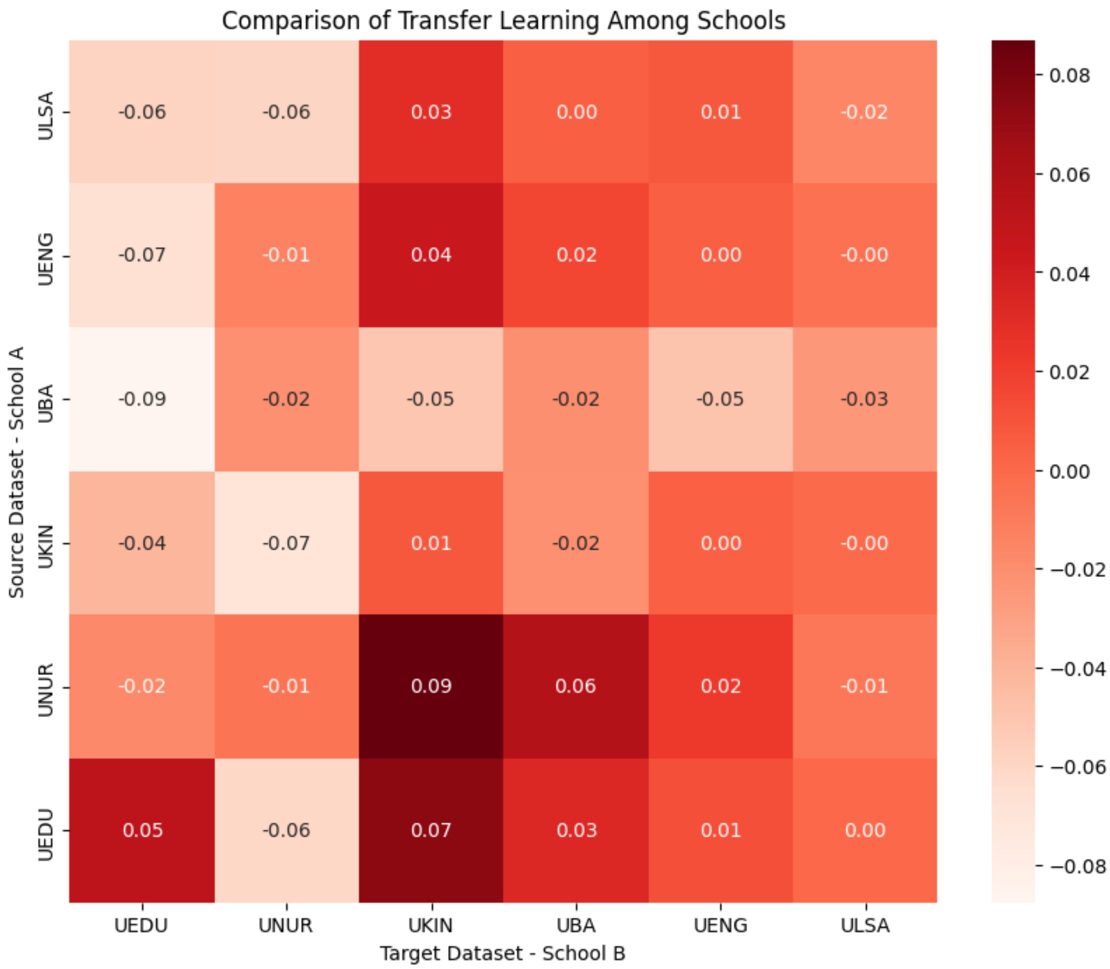
```python
df_compare_w_20vs30 = df_result_w_20 - df_result_w_30
df_compare_w_20vs30
```

Out[328]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.051727	-0.060698	0.073895	0.033389	0.01198	0.000053
School_2	-0.016943	-0.006372	0.086978	0.056843	0.022854	-0.00769
School_3	-0.041253	-0.070029	0.008327	-0.020862	0.003834	-0.000559
School_4	-0.087646	-0.021446	-0.050165	-0.020527	-0.049444	-0.025557
School_5	-0.066376	-0.013308	0.044536	0.016525	0.004636	-0.003932
School_6	-0.058988	-0.059054	0.030155	0.004723	0.008401	-0.015496

In [329…

```python
df = df_compare_w_20vs30
colormap = 'Reds'
chart_w_20vs30 = chart_compare_transfer_learning(df, list_schools, colormap)
```



In [330…

```
list_school_df_w = [df_UEDU_w, df_UNUR_w, df_UKIN_w, df_UBA_w, df_UENG_w, df_ULSA_w]
df_result_w_15 = table_transfer(list_school_df_w, 0.15)
df_result_w_15
```

Out[330]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.818182	0.684373	0.67948	0.517185	0.6775	0.583381
School_2	0.872078	0.698113	0.698252	0.526079	0.59843	0.511324
School_3	0.838312	0.60687	0.614991	0.561526	0.666334	0.593586
School_4	0.683766	0.640397	0.544449	0.550101	0.59769	0.605035
School_5	0.818182	0.677165	0.671088	0.533046	0.663198	0.645275
School_6	0.856494	0.600581	0.691696	0.556085	0.698575	0.670547

In [331…

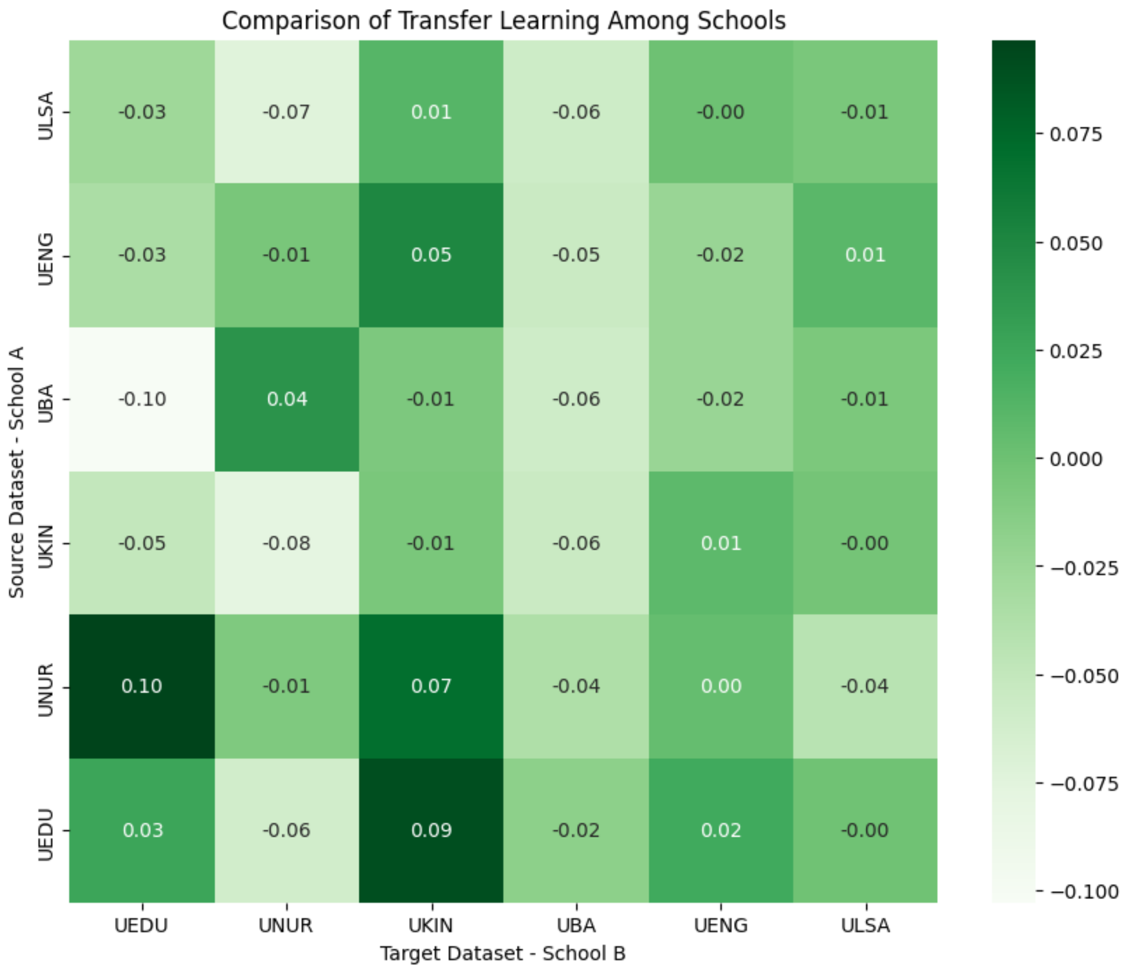
```
df_compare_w_15vs30 = df_result_w_15 - df_result_w_30
df_compare_w_15vs30
```

Out[331]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.025854	-0.060982	0.089924	-0.01665	0.021884	-0.001343
School_2	0.096525	-0.008558	0.069934	-0.036724	0.004667	-0.043541
School_3	-0.050245	-0.078284	-0.00685	-0.055035	0.007855	-0.003132
School_4	-0.10271	0.039924	-0.007908	-0.058736	-0.02358	-0.007117
School_5	-0.034744	-0.005839	0.05026	-0.054553	-0.023786	0.009492
School_6	-0.026731	-0.073571	0.011206	-0.057833	-0.000668	-0.006604

In [332…

```
df = df_compare_w_15vs30
colormap = 'Greens'
chart_w_15vs30 = chart_compare_transfer_learning(df, list_schools, colormap)
```



In [333…

```
list_school_df_c = [df_UEDU_c, df_UNUR_c, df_UKIN_c, df_UBA_c, df_UENG_c, df_ULSA_c]
df_result_c_20 = table_transfer(list_school_df_c, 0.2)
df_result_c_20
```

Out[333]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.761934	0.613751	0.462754	0.533516	0.521817	0.511438
School_2	0.73499	0.5585	0.48873	0.495072	0.544637	0.538443
School_3	0.832185	0.560409	0.627026	0.498052	0.588889	0.572314
School_4	0.653174	0.599232	0.521722	0.608392	0.584597	0.58806
School_5	0.810901	0.588532	0.517491	0.543024	0.620631	0.610854
School_6	0.842643	0.595151	0.590183	0.554102	0.603087	0.647281

In [334…

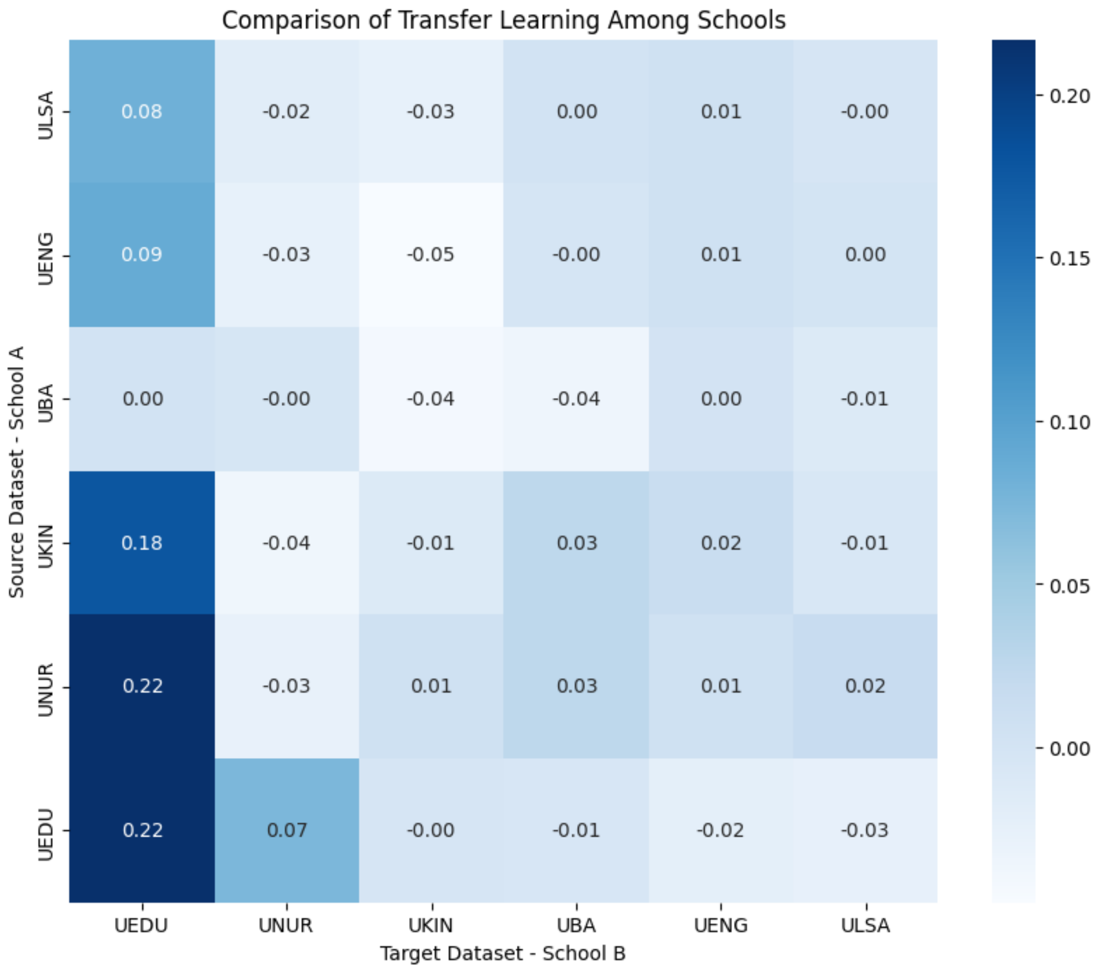
```
df_compare_c_20vs30 = df_result_c_20 - df_result_c_30
df_compare_c_20vs30
```

Out[334]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.216712	0.073365	-0.002911	-0.005029	-0.022406	-0.025571
School_2	0.215946	-0.027449	0.007027	0.025387	0.006716	0.015847
School_3	0.178455	-0.036653	-0.012094	0.02527	0.015005	-0.005312
School_4	0.00252	-0.004071	-0.041702	-0.03568	0.001362	-0.011624
School_5	0.088715	-0.026835	-0.047487	-0.001622	0.006392	0.00003
School_6	0.081844	-0.018477	-0.02649	0.002619	0.006704	-0.002505

In [335…

```
df = df_compare_c_20vs30
colormap = 'Blues'
chart_c_20vs30 = chart_compare_transfer_learning(df, list_schools, colormap)
```



In [336…

```
list_school_df_c = [df_UEDU_c, df_UNUR_c, df_UKIN_c, df_UBA_c, df_UENG_c, df_ULSA_c]
df_result_c_15 = table_transfer(list_school_df_c, 0.15)
df_result_c_15
```

Out[336]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.6	0.523588	0.487615	0.574878	0.527159	0.541089
School_2	0.645288	0.525624	0.434357	0.465882	0.550054	0.519991
School_3	0.939267	0.509066	0.585578	0.522936	0.576422	0.565962
School_4	0.58377	0.639954	0.535092	0.605287	0.577453	0.586146
School_5	0.831414	0.495544	0.579464	0.558575	0.612886	0.617512
School_6	0.727225	0.570841	0.607518	0.571921	0.589479	0.652037

In [337…

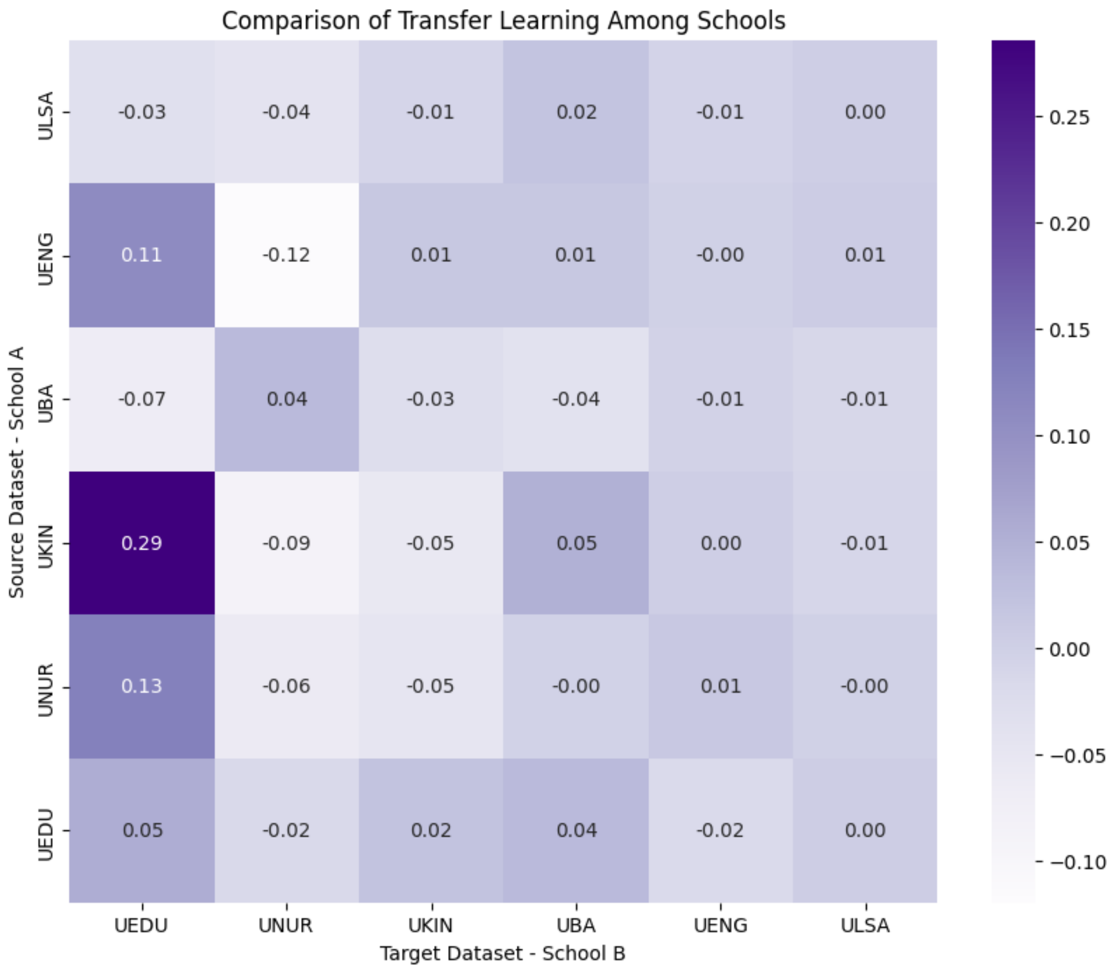
```
df_compare_c_15vs30 = df_result_c_15 - df_result_c_30
df_compare_c_15vs30
```

Out[337]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.054777	-0.016798	0.02195	0.036333	-0.017064	0.00408
School_2	0.126243	-0.060325	-0.047346	-0.003803	0.012134	-0.002606
School_3	0.285537	-0.087995	-0.053543	0.050154	0.002537	-0.011664
School_4	-0.066885	0.03665	-0.028332	-0.038785	-0.005783	-0.013537
School_5	0.109228	-0.119823	0.014486	0.013929	-0.001352	0.006688
School_6	-0.033573	-0.042787	-0.009155	0.020438	-0.006904	0.002251

In [338…

```
df = df_compare_c_15vs30
colormap = 'Purples'
chart_c_15vs30 = chart_compare_transfer_learning(df, list_schools, colormap)
```



1). We comapre the split size from original setting 0.3 with the 0.2 or 0.15. We found no matter it's in the "Withdraw" or "Cancel" Target Varaible, 0.2 all performs the best in comparison to 0.3 or 0.15

4.5 PART V: TRANSFER LEARNING BETWEEN SCHOOLS WITH OVERSAMPLE STRATEGY

In [339…

```
def transfer_model_oversample(df_A, df_B, size_A, size_B, strategy_A):
    # load and preprocess Dataset A
    X_train_A, X_test_A, y_train_A, y_test_A = data_for_model(df = df_A, test_size = size_A)

    if strategy_A == True:
        # apply SMOTE to Dataset A training set
        smote = SMOTE(random_state=RANDOM_SEED)
        # ada = ADASYN(random_state=RANDOM_SEED)
        X_train_A_smote, y_train_A_smote = smote.fit_resample(X_train_A, y_train_A)
        # X_train_A_ada, y_train_A_ada = ada.fit_resample(X_train_A, y_train_A)

        # train model on Dataset A after applying SMOTE
        model = GradientBoostingClassifier(random_state=RANDOM_SEED)
        model.fit(X_train_A_smote, y_train_A_smote)
        # model.fit(X_train_A_ada, y_train_A_ada)

    elif strategy_A == False:
        # train model on Dataset A
        model = GradientBoostingClassifier(random_state=RANDOM_SEED)
        model.fit(X_train_A, y_train_A)

    # load and preprocess Dataset B
    X_train_B, X_test_B, y_train_B, y_test_B = data_for_model(df = df_B, test_size = size_B)

    y_pred_B = model.predict(X_test_B)
    y_pred_probs_B = model.predict_proba(X_test_B)[:, 1]
    stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state = RANDOM_SEED)
```

```
    scores = cross_val_score(model, X_train_B, y_train_B, cv=stratified_kfold, scoring='roc_auc')
    auroc_B = roc_auc_score(y_test_B, y_pred_probs_B)

    # calculate AUROC
    auroc_B = roc_auc_score(y_test_B, y_pred_probs_B)

    return auroc_B
```

In [340…

```
def table_transfer_oversample(list_school_df, size):
    num_schools = len(list_school_df)
    # initialize an empty DataFrame with proper dimensions and labels
    df_results = pd.DataFrame(index=[f'School_{i+1}' for i in range(num_schools)],
                              columns=[f'School_{j+1}' for j in range(num_schools)])

    for i, df_A in enumerate(list_school_df):
        for j, df_B in enumerate(list_school_df):

            # call the transfer_model function with df_A and df_B
            result = transfer_model_oversample(df_A, df_B, size, size, strategy_A)

            # store the result in the DataFrame
            df_results.iloc[i, j] = result

    return df_results
```

In [341…

```
strategy_A = True
list_school_df_w = [df_UEDU_w, df_UNUR_w, df_UKIN_w, df_UBA_w, df_UENG_w, df_ULSA_w]
df_result_w_smote_t = table_transfer_oversample(list_school_df_w, 0.2)
df_result_w_smote_t
```

Out[341]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.869071	0.677394	0.615447	0.423685	0.566256	0.40034
School_2	0.548083	0.722334	0.52547	0.460498	0.455893	0.488164
School_3	0.475309	0.626795	0.622781	0.4986	0.556894	0.426787
School_4	0.701105	0.610392	0.485797	0.536073	0.480124	0.460621
School_5	0.380442	0.594668	0.36922	0.48225	0.672212	0.45967
School_6	0.582846	0.605686	0.420586	0.565281	0.456595	0.656418

In [342…

```
strategy_A = False
df_result_w_f = table_transfer_oversample(list_school_df_w, 0.2)
df_result_w_f
```

Out[342]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.844055	0.684657	0.663451	0.567224	0.667596	0.584776
School_2	0.758609	0.700299	0.715295	0.619647	0.616617	0.547174
School_3	0.847303	0.615125	0.630168	0.595699	0.662313	0.59616
School_4	0.69883	0.579026	0.502192	0.58831	0.571826	0.586594
School_5	0.78655	0.669695	0.665365	0.604123	0.69162	0.631851
School_6	0.824237	0.615098	0.710645	0.618641	0.707644	0.661655

In [343…

```
df_w_smote = df_result_w_smote_t - df_result_w_f
df_w_smote
```

Out[343]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.025016	-0.007263	-0.048004	-0.143539	-0.10134	-0.184436
School_2	-0.210526	0.022035	-0.189825	-0.159149	-0.160724	-0.05901
School_3	-0.371995	0.01167	-0.007387	-0.097099	-0.105418	-0.169373
School_4	0.002274	0.031366	-0.016396	-0.052238	-0.091702	-0.125973
School_5	-0.406108	-0.075027	-0.296144	-0.121873	-0.019408	-0.17218
School_6	-0.241391	-0.009412	-0.290059	-0.05336	-0.251049	-0.005237

In [370…

```
df = df_w_smote
colormap = 'bwr'
chart_w_smote = chart_net_transfer_learning(df, list_schools, colormap)
```

In [345…

```
strategy_A = True
list_school_df_c = [df_UEDU_c, df_UNUR_c, df_UKIN_c, df_UBA_c, df_UENG_c, df_ULSA_c]
df_result_c_smote_t = table_transfer_oversample(list_school_df_c, 0.2)
df_result_c_smote_t
```

Out[345]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.649606	0.496812	0.465894	0.471687	0.501718	0.472052
School_2	0.71752	0.614847	0.401486	0.521364	0.499117	0.534126
School_3	0.461614	0.470638	0.559443	0.467383	0.469527	0.432283
School_4	0.516609	0.495919	0.512372	0.555181	0.446214	0.479481
School_5	0.365404	0.440343	0.495141	0.486188	0.605539	0.437216
School_6	0.617372	0.582399	0.432546	0.534819	0.452987	0.620652

In [346…

```
strategy_A = False
df_result_c_f = table_transfer_oversample(list_school_df_c, 0.2)
df_result_c_f
```

Out[346]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	0.761934	0.613751	0.462754	0.533516	0.521817	0.511438
School_2	0.73499	0.5585	0.48873	0.495072	0.544637	0.538443
School_3	0.832185	0.560409	0.627026	0.498052	0.588889	0.572314
School_4	0.653174	0.599232	0.521722	0.608392	0.584597	0.58806
School_5	0.810901	0.588532	0.517491	0.543024	0.620631	0.610854
School_6	0.842643	0.595151	0.590183	0.554102	0.603087	0.647281

In [347…

```
df_c_smote = df_result_c_smote_t - df_result_c_f
df_c_smote
```

Out[347]:

	School_1	School_2	School_3	School_4	School_5	School_6
School_1	-0.112328	-0.116939	0.00314	-0.061829	-0.020099	-0.039386
School_2	-0.01747	0.056347	-0.087244	0.026292	-0.04552	-0.004317
School_3	-0.370571	-0.08977	-0.067584	-0.030669	-0.119362	-0.140032
School_4	-0.136565	-0.103314	-0.00935	-0.053211	-0.138383	-0.108579
School_5	-0.445497	-0.148189	-0.02235	-0.056837	-0.015092	-0.173638
School_6	-0.225271	-0.012752	-0.157637	-0.019282	-0.1501	-0.026629

In [371…

```
df = df_c_smote
colormap = 'bwr'
chart_c_smote = chart_net_transfer_learning(df, list_schools, colormap)
```

1). We comapre when we use the SMOTE to dataset A to see how it performs after tarnsfer learning, we found there's no very significant better after SMOTE applied.

2). It's specifically getting worse when the UEDU, the smallest school got applied. Likely the smallest to-be-transferred dataset is sensitive from the source if applied by SMOTE.