Wizard School

Machine Learning I Final Project

Group05

Students:

Dinis Fernandes - 20221848
Inês Santos - 20221916
Sara Ferrer - 20221947
Vidhi Rajanikante - 20221982
Yehor Malakhov - 20221691

Index

Packages Imports

In [1]:

```
#Pandas
import pandas as pd

#Numpy for calculations
import numpy as np

#Plots
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.gridspec as gridspec #to help in grid plotting

# Imputer for missing values
from sklearn.impute import KNNImputer

# Data Partition
from sklearn.model_selection import train_test_split, KFold

# Scaler
from sklearn.preprocessing import MinMaxScaler,StandardScaler,RobustSca

# Encoding
from sklearn.preprocessing import OneHotEncoder

# Feature Selection
from sklearn.linear_model import LogisticRegression, LassoCV
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

# Models
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import BaggingClassifier,AdaBoostClassifier,Stack
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier

#Evaluating
from sklearn.metrics import accuracy_score, classification_report, conf

# Warnings
import warnings
warnings.filterwarnings("ignore")
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\__init__.py:155: User
Warning: A NumPy version >=1.18.5 and <1.25.0 is required for this ver
sion of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversio
n}"
```

Data Preparation and Pre-Processing

Importing

In [2]:
```
initial_data = pd.read_csv("Project_train_dataset.csv")
initial_data.head()
```

Out[2]:

	Student ID	Program	Student Gender	Experience Level	Student Siblings	Student Family	Financial Background	School Dormitory
0	1	Sorcery School	male	22.0	1	0	7.2500	NaN
1	2	Magi Academy	female	38.0	1	0	71.2833	Cottage Chamber
2	3	Sorcery School	female	26.0	0	0	7.9250	NaN
3	5	Sorcery School	male	35.0	0	0	8.0500	NaN
4	6	Sorcery School	male	NaN	0	0	8.4583	NaN

Converting to better datatypes

In [3]:
```
# Checking the real memory usage
initial_data.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 713 entries, 0 to 712
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Student ID               713 non-null    int64
 1   Program                  713 non-null    object
 2   Student Gender           713 non-null    object
 3   Experience Level         567 non-null    float64
 4   Student Siblings         713 non-null    int64
 5   Student Family           713 non-null    int64
 6   Financial Background     713 non-null    float64
 7   School Dormitory         153 non-null    object
 8   School of Origin         713 non-null    object
 9   Student Social Influence  713 non-null    int64
 10  Favourite Study Element  713 non-null    object
 11  Admitted in School       713 non-null    int64
dtypes: float64(2), int64(5), object(5)
memory usage: 252.9 KB
```

In [4]:
```
# Getting the descriptive statstics to know what datatype changes to ma
initial_data.describe().T
```

Out[4]:

	count	mean	std	min	25%	50%	75%	max
Student ID	713.0	443.402525	257.180421	1.00	219.000	446.0	666.0	889.0000
Experience Level	567.0	29.890952	14.599272	0.42	20.750	28.0	39.0	80.0000
Student Siblings	713.0	0.521739	1.057287	0.00	0.000	0.0	1.0	8.0000
Student Family	713.0	0.354839	0.770985	0.00	0.000	0.0	0.0	6.0000
Financial Background	713.0	31.327238	50.903034	0.00	7.925	14.4	30.0	512.3292
Student Social Influence	713.0	12.719495	6.949648	1.00	7.000	13.0	19.0	24.0000
Admitted in School	713.0	0.353436	0.478372	0.00	0.000	0.0	1.0	1.0000

In [5]:
```
# Check the maxs of types of int
print('int8:[' + str(np.iinfo(np.int8).min) + ',' + str(np.iinfo(np.int
print('int16:[' + str(np.iinfo(np.int16).min) + ',' + str(np.iinfo(np.i
print('int32:[' + str(np.iinfo(np.int32).min) + ',' + str(np.iinfo(np.i
print('int64:[' + str(np.iinfo(np.int64).min) + ',' + str(np.iinfo(np.i
```

```
int8:[-128,127]
int16:[-32768,32767]
int32:[-2147483648,2147483647]
int64:[-9223372036854775808,9223372036854775807]
```

In [6]:
```
# Check the maxs of types of float
print('float16:[' + str(np.finfo(np.float16).min) + ',' + str(np.finfo(
print('float32:[' + str(np.finfo(np.float32).min) + ',' + str(np.finfo(
print('float64:[' + str(np.finfo(np.float64).min) + ',' + str(np.finfo(
```

```
float16:[-65500.0,65500.0]
float32:[-3.4028235e+38,3.4028235e+38]
float64:[-1.7976931348623157e+308,1.7976931348623157e+308]
```

After checking the descriptive statistics, the memory usage but also a few rows of data we decided to change the variables to the following datatypes, making the memory usage lower.

In [7]:
```
initial_data = initial_data.astype({'Student ID' : 'int16',
                                    'Program' : 'category',
                                    'Student Gender' : 'category',
                                    'Experience Level' : 'float16',
                                    'Student Siblings' : 'int8',
                                    'Student Family' : 'int8',
                                    'Financial Background' : 'float32',
                                    'School Dormitory' : 'category',
                                    'School of Origin' : 'category',
                                    'Student Social Influence' : 'int8'
                                    'Favourite Study Element' : 'categc
                                    'Admitted in School' : 'bool'})
```

In [8]:
```
# Checking the final memory usage of the data after all the datatype ch
initial_data.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 713 entries, 0 to 712
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Student ID               713 non-null    int16
 1   Program                  713 non-null    category
 2   Student Gender           713 non-null    category
 3   Experience Level         567 non-null    float16
 4   Student Siblings         713 non-null    int8
 5   Student Family           713 non-null    int8
 6   Financial Background     713 non-null    float32
 7   School Dormitory         153 non-null    category
 8   School of Origin         713 non-null    category
 9   Student Social Influence  713 non-null    int8
 10  Favourite Study Element  713 non-null    category
 11  Admitted in School       713 non-null    bool
dtypes: bool(1), category(5), float16(1), float32(1), int16(1), int8
(3)
memory usage: 13.8 KB
```

Thus, we can see that the memory usage has now been decreased to 13.1KB, from 252.9KB, after all the datatype changes made within the variables of the training dataset.

Understanding the data

In the following section we will try to understand our data in order to make mor informative decisions regarding our pre-processing, feature selection and even regarding our models or problems we may encounter.

In [9]:
```
initial_data.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
Student ID	713.0	443.402525	257.180421	1.000000	219.000	446.0	666.0	889.000000
Experience Level	567.0	29.890625	14.601562	0.419922	20.750	28.0	39.0	80.000000
Student Siblings	713.0	0.521739	1.057287	0.000000	0.000	0.0	1.0	8.000000
Student Family	713.0	0.354839	0.770985	0.000000	0.000	0.0	0.0	6.000000
Financial Background	713.0	31.327238	50.903030	0.000000	7.925	14.4	30.0	512.329224
Student Social Influence	713.0	12.719495	6.949648	1.000000	7.000	13.0	19.0	24.000000

In [10]:
```
initial_data.describe(include = "category").T
```

Out[10]:

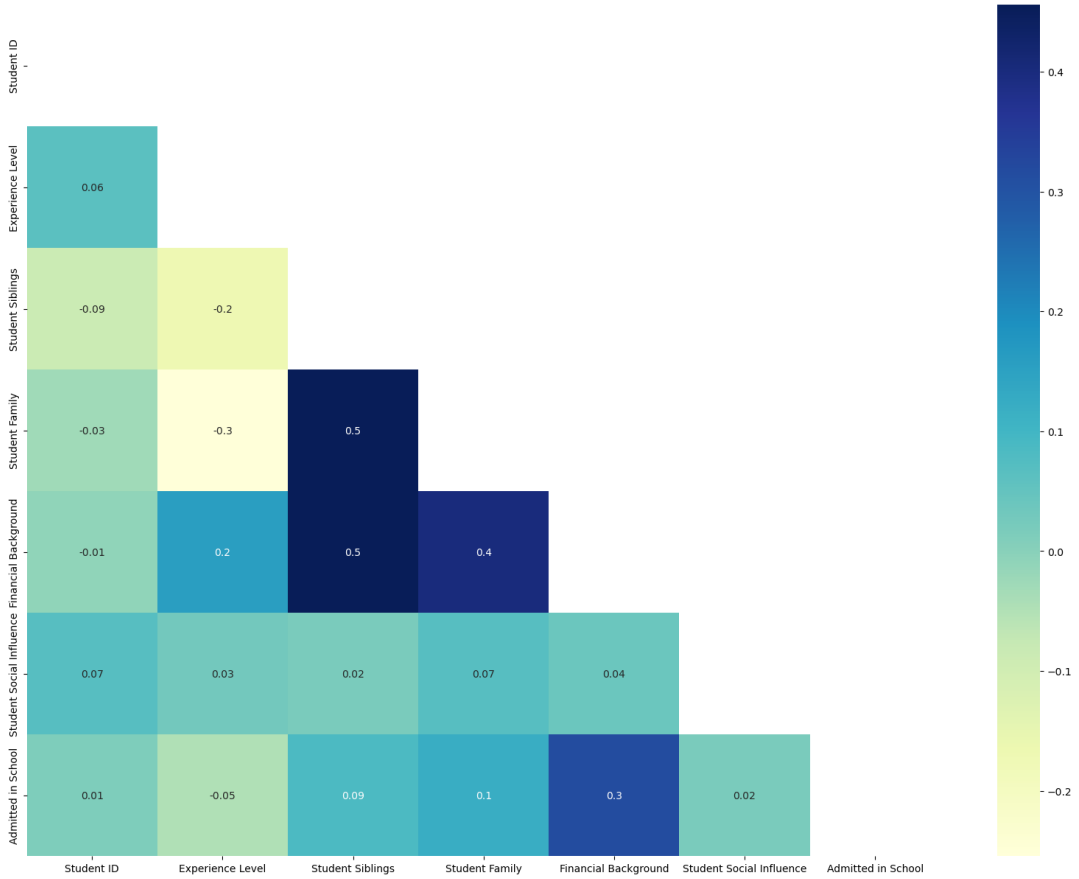
	count	unique	top	freq
Program	713	3	Sorcery School	391
Student Gender	713	2	male	469
School Dormitory	153	6	Mystical Chamber	51
School of Origin	713	3	Mystic Academy	524
Favourite Study Element	713	4	Earth	184

Visualizations

Numerical variables

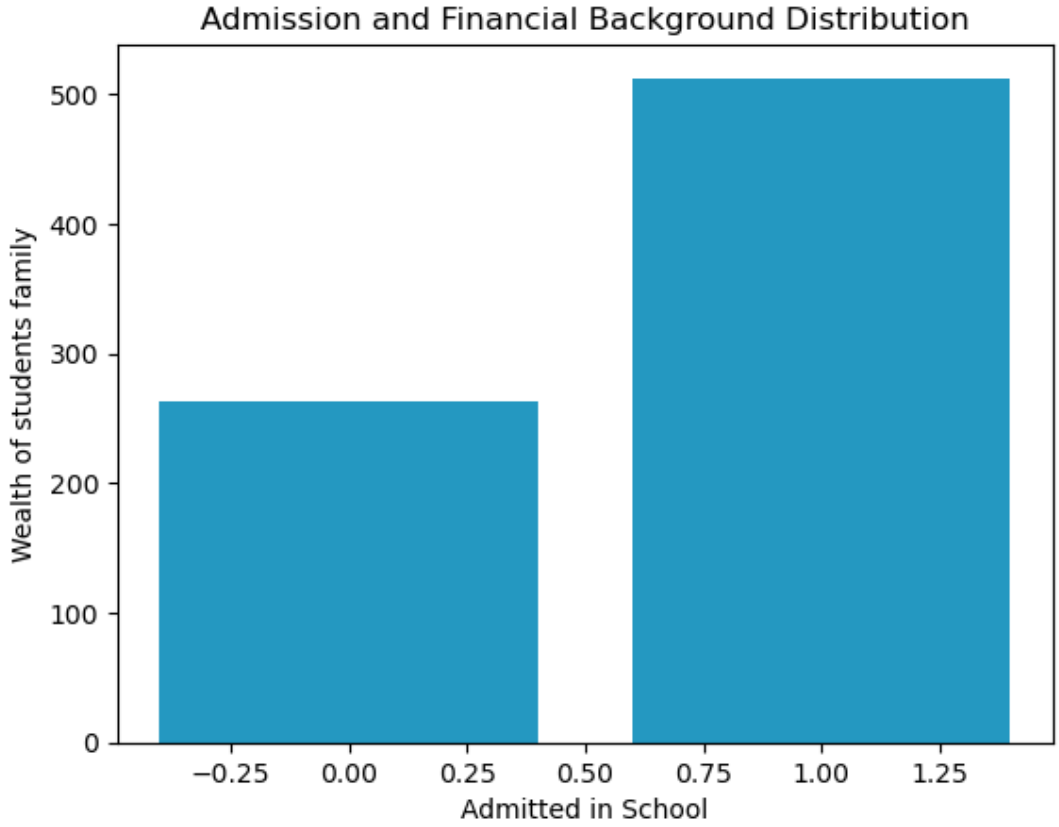
In the heatmap we can conclude that we don't have any numerical variable highly correlated to the target,however the highest one is 0.3 regarding the Financial Background. We would also like to point out that our highest correlation is 0.5 between the variables `Student Family` and `Student Siblings` .

In [11]:
```
"""corr = initial_data.select_dtypes(include=[np.number, bool]).corr(me

def cor_heatmap(cor):
    mask = np.triu(np.ones_like(cor))
    plt.figure(figsize=(20, 15))
    sns.heatmap(data = cor, annot = True, mask = mask, cmap = "YlGnBu",
    #vmin=-1, vmax=1
    )
    plt.show()

cor_heatmap(corr)"""
```

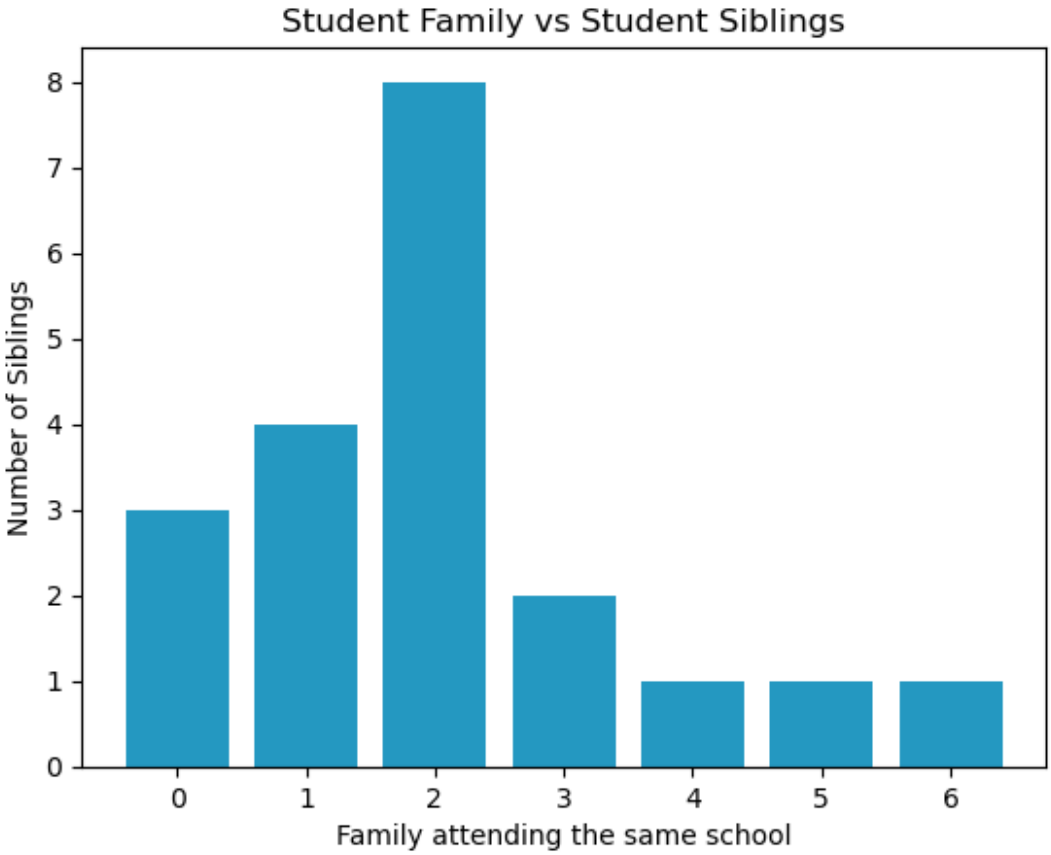


Here we can conclude that students that are wealthy have a higher chance to enter the Wizard School.

In [12]:
```
"""rgb_color = (0.14196078431372547, 0.5976470588235294, 0.756078431372
fig, ax = plt.subplots()
ax.bar(initial_data['Admitted in School'], initial_data['Financial Back
ax.set_xlabel('Admitted in School')
ax.set_ylabel('Wealth of students family')
ax.set_title('Admission and Financial Background Distribution')

plt.show()
"""
```



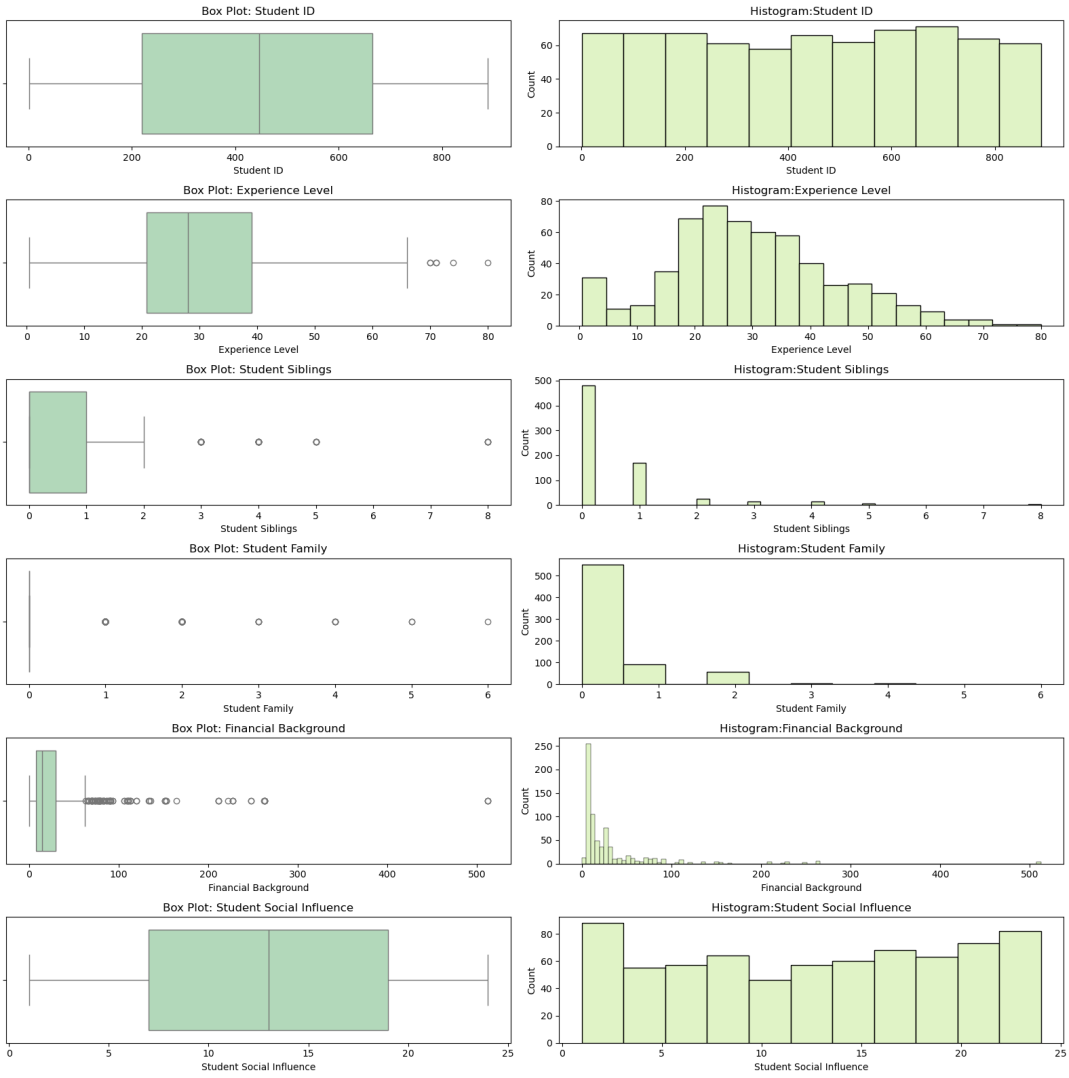
In [13]:
```
"""# Create a figure and axis
fig, ax = plt.subplots()

# Create the bar chart
ax.bar(initial_data['Student Family'], initial_data['Student Siblings']

# Set labels and title
ax.set_xlabel('Family attending the same school')
ax.set_ylabel('Number of Siblings')
ax.set_title('Student Family vs Student Siblings')

# Show the plot
plt.show()"""
```



After plotting different bar plots between the variables with higher correlation we decided to plot several boxplots to check for outliers.

In [14]:

```
"""#colors
color_box = (0.6718800461361015, 0.8715263360246059, 0.7164321414840447
color_hist = (0.84, 0.9372549019607843, 0.7011764705882353, 1.0)


rows = len(initial_data.select_dtypes(include= np.number).columns) # th
fig = plt.figure(figsize=(16, 16)) # creating the figure
gs = gridspec.GridSpec(rows, 2) #gridspec is used to give us the "index

#Plotting
for i, variable in enumerate(initial_data.select_dtypes(include= np.num

    #Box Plot
    ax = fig.add_subplot(gs[i, 0]) # do only in column 0 # ax is where
    sns.boxplot(data=initial_data, x=variable, ax=ax, color = color_box
    plt.title(f'Box Plot: {variable}')


    #Histogram
    ax = fig.add_subplot(gs[i, 1]) # do only on column 1
    sns.histplot(data=initial_data, x=variable, color=color_hist, ax=ax
    plt.title(f"Histogram:{variable}")

plt.tight_layout()
plt.show()

"""
```

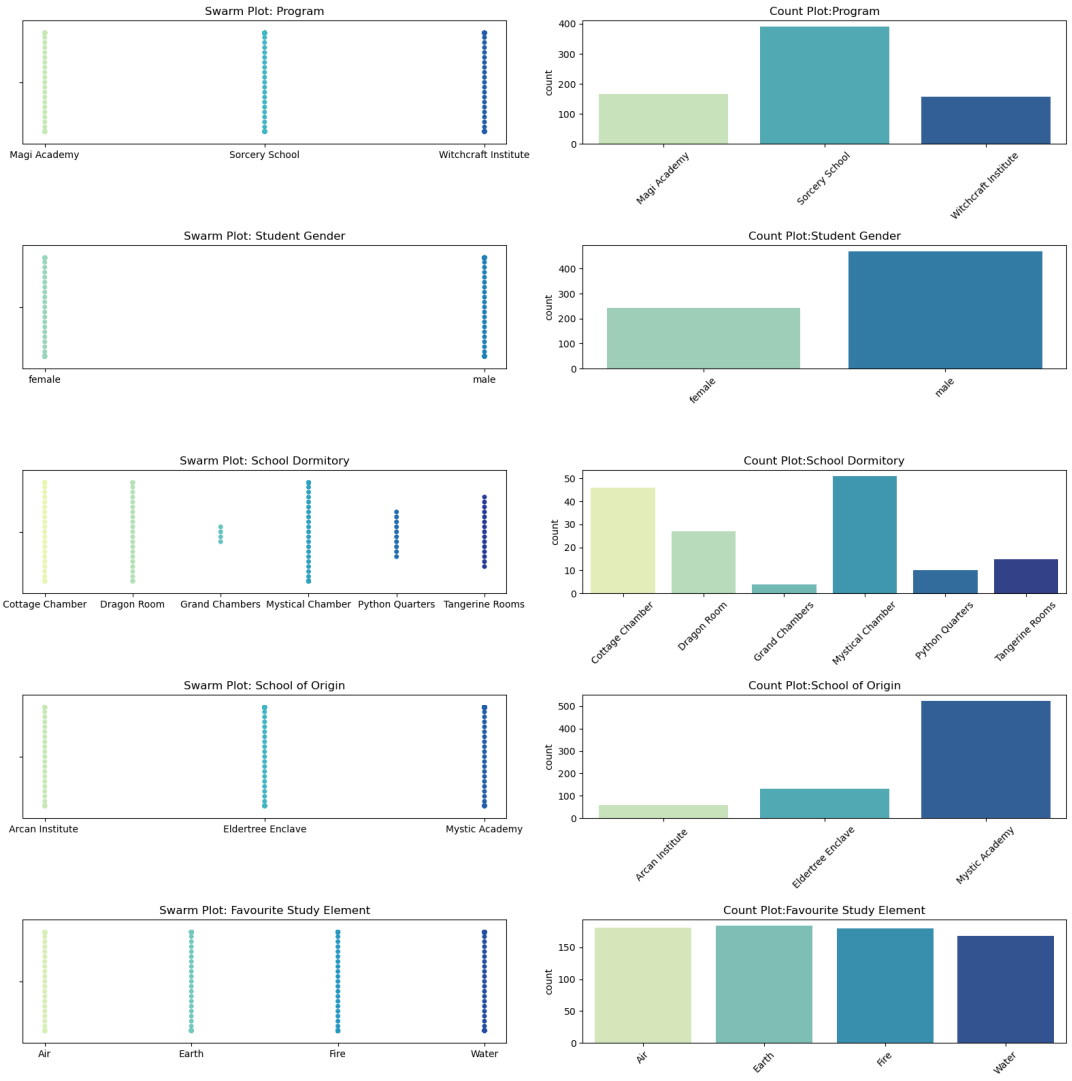


With this Boxplots and Histograms we can truly confirm that the variables `Experience Level` , `Student Siblings` , `Student Family` and `Financial Background` present outliers.

Categorical variables

In [15]:

```
"""rows = len(initial_data.select_dtypes(include="category").columns) #
fig = plt.figure(figsize = (16,16)) # creating the figure
gs = gridspec.GridSpec(rows, 2)

#Plotting
for i, variable in enumerate(initial_data.select_dtypes(include="catego

# Hue is just to make the graphs more beautiful and easier to understan

    #Swarm Plot (equivalent of boxplot but for categorical data, we can
    ax = fig.add_subplot(gs[i, 0]) # do only in column 0
    sns.swarmplot(data=initial_data, x=variable, ax=ax, palette = "YlGn
    ax.set_xlabel("")
    plt.title(f'Swarm Plot: {variable}')

    #Count Plot
    ax1 = fig.add_subplot(gs[i, 1]) # do only on column 1
    sns.countplot(data=initial_data, x=variable, palette = "YlGnBu", hu
    ax1.set_xticklabels(ax.get_xticklabels(), rotation=45)
    ax1.set_xlabel("")
    plt.title(f"Count Plot:{variable}")

plt.tight_layout()
plt.show()"""
```

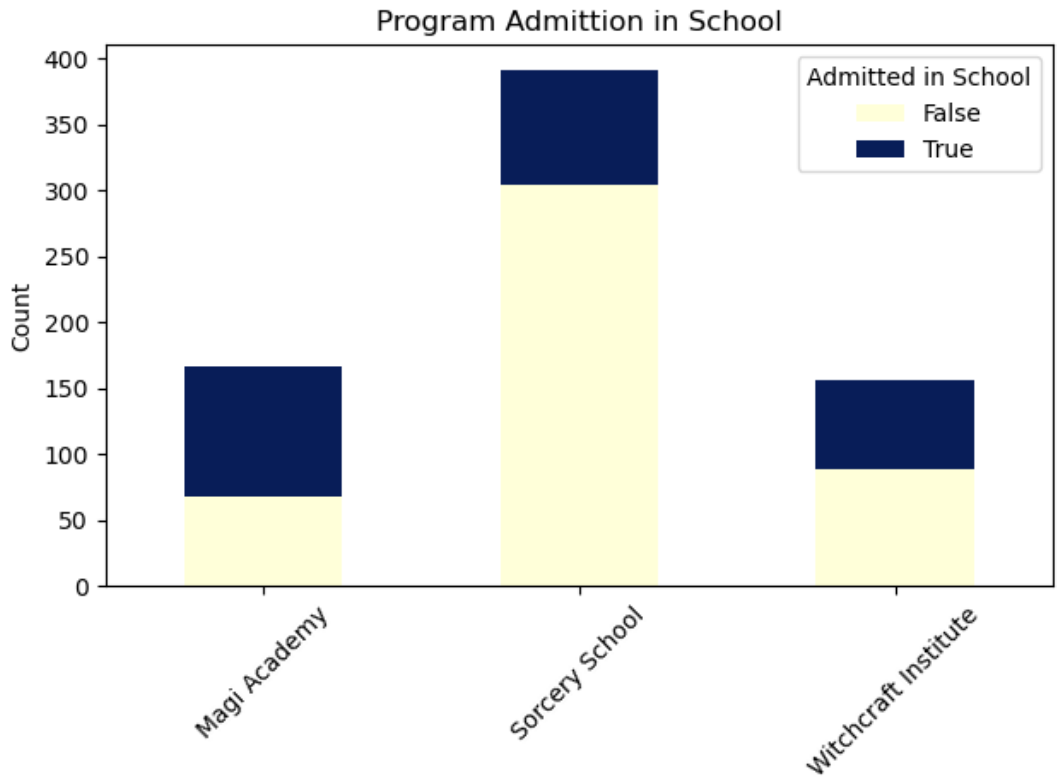


With these Swarm Plots and Count Plots we are sure that in the categorical variables we don't have outliers.

Next we decided to plot our categorical variables with our target to check their importance but also to analyze if the dataset is inbalanced.

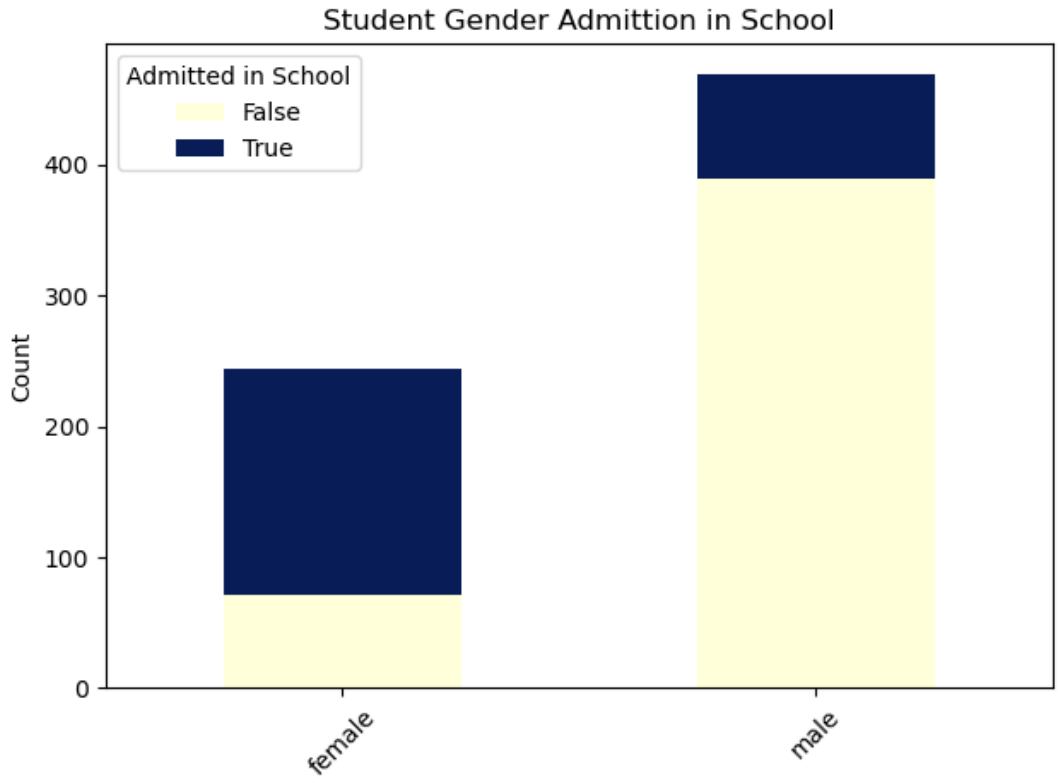
In [16]:
```
"""# Creating a crosstab of the categorical variables and being admitte
for i, var in enumerate(initial_data.select_dtypes(include = ["category
    crosstab = pd.crosstab(initial_data[str(var)], initial_data["Admitt


# Create a stacked bar plot
    plt.figure(figsize=(7, 7))
    crosstab.plot(kind='bar', stacked=True, colormap='YlGnBu')
    plt.title(f"{var} Admittion in School")
    plt.xlabel("")
    plt.ylabel("Count")
    plt.legend(title="Admitted in School")
    plt.xticks(rotation=45)


    print(f"{var}\n")
    for category in crosstab.index:
        true_percent = (crosstab.loc[category, True] / (crosstab.loc[ca
        false_percent = (crosstab.loc[category, False] / (crosstab.loc[
        print(f"{category}: True={true_percent:.2f}%, False={false_perc
    plt.tight_layout()
    plt.show()"""
```

```
Program

Magi Academy: True=59.04%, False=40.96%
Sorcery School: True=22.25%, False=77.75%
Witchcraft Institute: True=42.95%, False=57.05%

<Figure size 700x700 with 0 Axes>
```

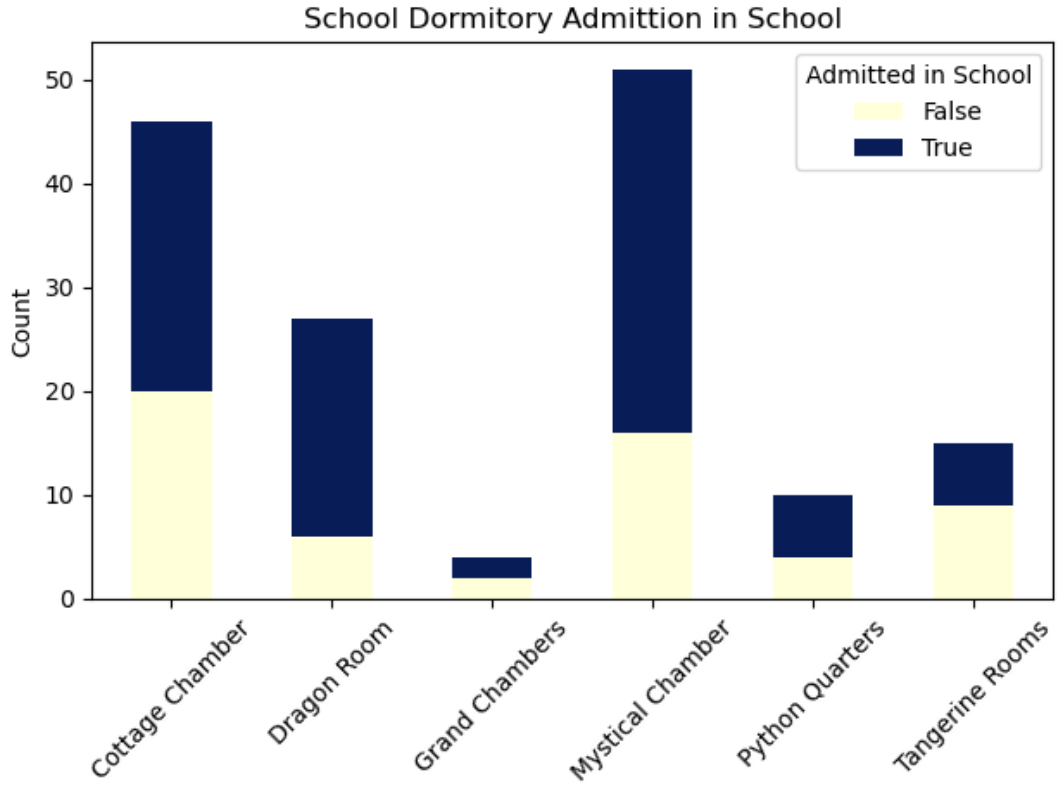



```
Student Gender

female: True=70.90%, False=29.10%
male: True=16.84%, False=83.16%
```

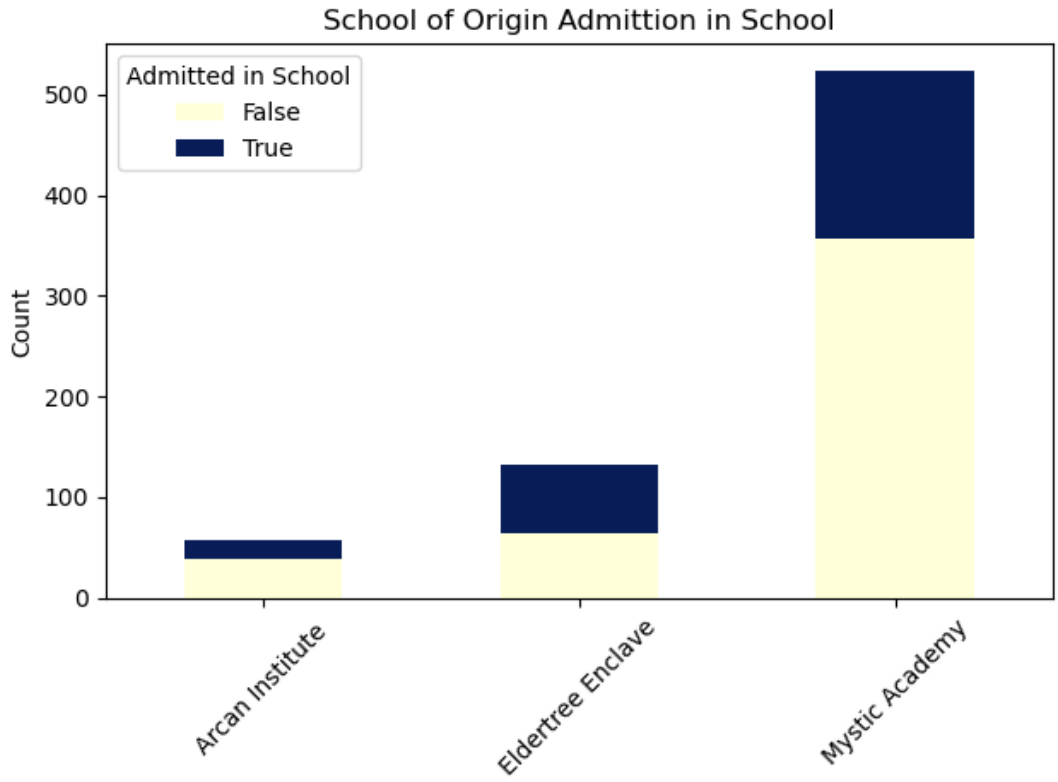
```
<Figure size 700x700 with 0 Axes>
```



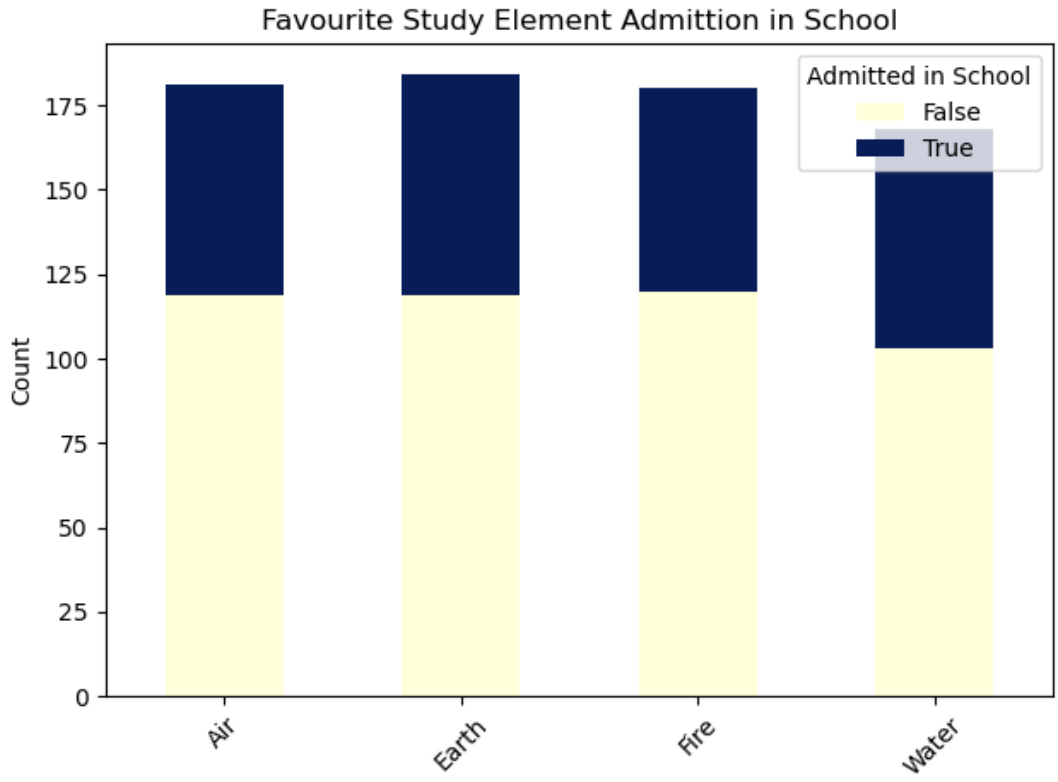

```
School Dormitory

Cottage Chamber: True=56.52%, False=43.48%
Dragon Room: True=77.78%, False=22.22%
Grand Chambers: True=50.00%, False=50.00%
Mystical Chamber: True=68.63%, False=31.37%
Python Quarters: True=60.00%, False=40.00%
Tangerine Rooms: True=40.00%, False=60.00%

<Figure size 700x700 with 0 Axes>
```



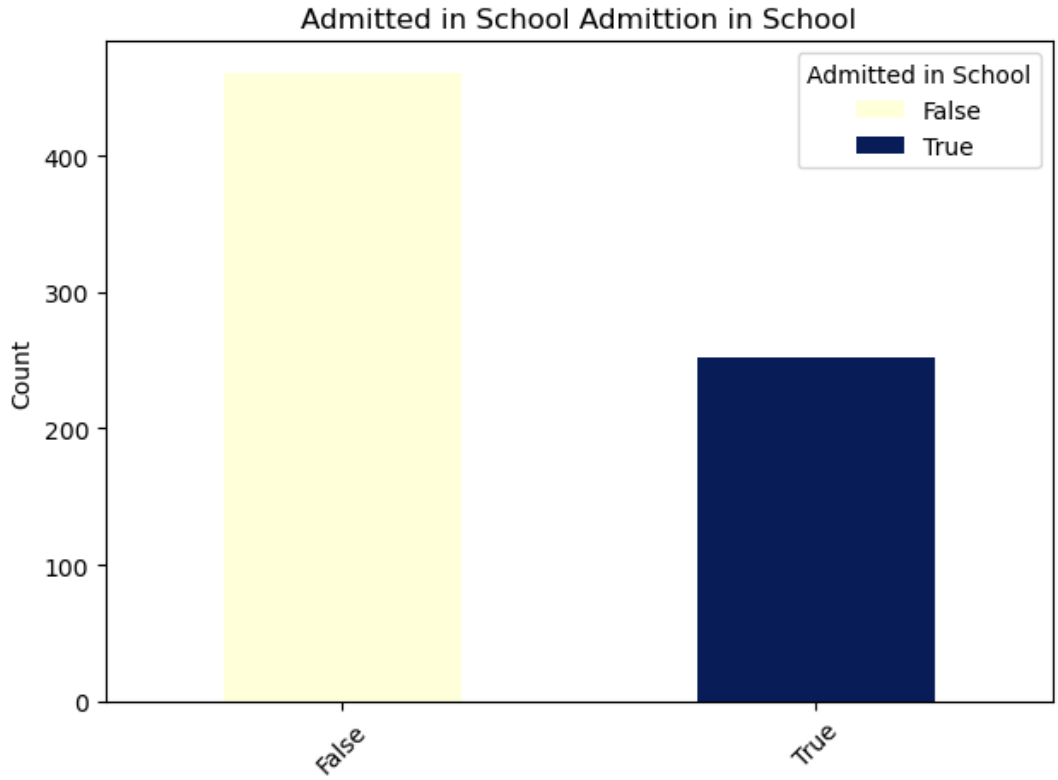
```
School of Origin

Arcan Institute : True=31.58%, False=68.42%
Eldertree Enclave: True=50.76%, False=49.24%
Mystic Academy: True=31.87%, False=68.13%

<Figure size 700x700 with 0 Axes>
```




```
Favourite Study Element

Air: True=34.25%, False=65.75%
Earth: True=35.33%, False=64.67%
Fire: True=33.33%, False=66.67%
Water: True=38.69%, False=61.31%

<Figure size 700x700 with 0 Axes>
```




```
Admitted in School

False: True=0.00%, False=100.00%
True: True=100.00%, False=0.00%

<Figure size 700x700 with 0 Axes>
```



We would like to point that our dataset is in fact inbalanced, we have more student who were not admitted in school rather than the ones that were.

Dividing initial data

In [17]:
```
X = initial_data.drop(["Admitted in School"], axis = 1)
y = initial_data["Admitted in School"]
```

Train Test Split

In [18]:
```
X_train, X_val , y_train, y_val = train_test_split(X,y,test_size = 0.3
```

Outliers

In order to improve the results, we tried different methods to treat outliers. Firstly, we created a dictionary with specific conditions to delete some observations. Our first step was to delete the observations with the most extreme values since other techniques would have been biased. We removed four observations in total. We then applied the remaining conditions to the first part of our data. The negation of these conditions resulted in only the observations that didn't meet the criteria. Next, we applied the *winsorize* method to the variables where outliers were detected, this method involves creating limits based on the percentiles of each variable. The percentile values were calculated manually instead of using the package of winsorize to have more control over the clipping values. However, we decided not to apply this method to the `Financial Background` variable due to it's impact on the outcome, cause the higher the "Financial Background", the higher the probability of entry into the school. Therefore, we applied a limit to the number itself, rather than a percentile.
We also decided to create a function to be easier to apply the same treatment to our datasets.

In [19]:
```
def outlier_treatment(array):

    #Conditions for each variable
    conditions = {'Experience Level': lambda x: x >= 75,
            'Student Siblings': lambda x: x >= 7,
            'Student Family': lambda x: x >= 6,
            'Financial Background': lambda x: x >= 490}

# Filter out rows based on conditions
    for variable, condition in conditions.items():
        array = array[~condition(array[variable])]


# Limits (percentiles) for winsorization
    lower_limit = 0 # As we didn't have any outliers at the left we don
    upper_limit = 90 # We decided to choose a big percentile so we don'

    outliers = ["Experience Level", "Student Siblings", "Student Family

    # Calculation of the quantiles
    percentiles_before_winsorize = array[outliers].quantile([lower_limi


#Clipping the arrays to the values defined
    for variable in outliers:
        array[variable] = array[variable].clip(
            lower=percentiles_before_winsorize.loc[lower_limit / 100, v
            upper=percentiles_before_winsorize.loc[upper_limit / 100, v

# Creating by hand the limit of Financial Background
    array['Financial Background'] = array['Financial Background'].apply
    return array
```

In [20]:
```
X_train_o = outlier_treatment(X_train)
X_val_o = outlier_treatment(X_val)
```

In [21]:
```
# storing the indeces of the outliers in a variable so we have the same
removed_indices = list(set(X_train.index) - set(X_train_o.index))
removed_indices_val = list(set(X_val.index) - set(X_val_o.index))

# Remove the corresponding rows from y_train and y_val
y_train = y_train.drop(index=removed_indices)
y_val = y_val.drop(index=removed_indices_val)
```

Encoding

In the following cell, we are going to drop the only categorical variable with missing values. We decided to drop `School Dormitory` because it has 79% of missing values which means it's imputation wouldn't make sense and we didn't find a great conection to the target, as per our previous analysis. Therefore we don't even need to make the encoding of this variable

In [22]:
```
X_train_o.drop(["School Dormitory"], axis = 1,inplace =True)
X_val_o.drop(["School Dormitory"], axis = 1,inplace =True)
```

In [23]:
```
X_train_o.reset_index(inplace= True, drop = True)
X_val_o.reset_index(inplace = True, drop = True)
```

In [24]:
```
encoder = OneHotEncoder()
df_encoded_train = pd.DataFrame()
df_encoded_val = pd.DataFrame()

for col in X_train_o.select_dtypes(include='category').columns:# Iterat
    encoder.fit(X_train_o[[col]]) # Fit the encoder to the columns of X
    encoded_columns = encoder.transform(X_train_o[[col]]).toarray() # T
    column_names = [f"{col}_{(value)}" for value in encoder.categories_
    encoded_df = pd.DataFrame(encoded_columns, columns=column_names) #

    encoded_columns_val = encoder.transform(X_val_o[[col]]).toarray() #
    column_names = [f"{col}_{(value)}" for value in encoder.categories_
    encoded_df_ = pd.DataFrame(encoded_columns_val, columns=column_name

    # Concatenate the encoded columns to the empty dataframe
    df_encoded_val = pd.concat([df_encoded_val, encoded_df_], axis=1)

    # Concatenating the encoded columns to empty dataframe
    df_encoded_train = pd.concat([df_encoded_train, encoded_df], axis=1

# Concatenating the encoded dataframe to the train data
new_X_train = pd.concat([X_train_o, df_encoded_train], axis=1)

# Concatenating the encoded dataframe to the val data
new_X_val = pd.concat([X_val_o, df_encoded_val], axis=1)
```

Here we drop the old categorical columns

In [25]:
```
new_X_train.drop(["Program" , "Student Gender", "School of Origin", "Fa
```

In [26]:
```
new_X_val.drop(["Program" , "Student Gender", "School of Origin", "Favc
```

Missing Values

We decided to impute the missing values using the KNN imputer, this will segment our data and acording to the n_neighbors will decide the best value for the missing value in that row.

In [27]:
```
imputer = KNNImputer(n_neighbors=7)

# Fit and transform on X_train
new_X_train = pd.DataFrame(imputer.fit_transform(new_X_train.select_dty
                                                 columns=new_X_train.select_
                                                 index=new_X_train.select_dt

# Transform on X_val
new_X_val = pd.DataFrame(imputer.transform(new_X_val.select_dtypes(incl
                                           columns=new_X_val.select_dtyp
                                           index=new_X_val.select_dtypes
```

Scaling Data

Min Max Scaler

X_train

In [28]:
```
# Extracting the Student IDs
student_ids = new_X_train['Student ID']

# Dropping the student ID column before scaling
df_t = new_X_train.drop(columns=['Student ID'])

#Scaling
mms = MinMaxScaler().fit(df_t)
scaled_data = mms.transform(df_t)


# New DataFrame with the scaled data
X_train_s = pd.DataFrame(scaled_data, columns=df_t.columns)

# Add the Student IDs back
X_train_s['Student ID'] = student_ids
```

X_val

In [29]:
```
# Extracting the Student IDs
student_ids_v = new_X_val['Student ID']

# Dropping the student ID column before scaling
df_v = new_X_val.drop(columns=['Student ID'])

#Scaling
scaled_data_v = mms.transform(df_v)


# New DataFrame with the scaled data
X_val_s = pd.DataFrame(scaled_data_v, columns=df_v.columns)

# Add the Student IDs back
X_val_s['Student ID'] = student_ids_v
```

Standard Scaler

X_train

In [30]:
```
# Extracting the Student IDs
student_ids = new_X_train['Student ID']

# Dropping the student ID column before scaling
df_standard = new_X_train.drop(columns=['Student ID'])

#Scaling
std = StandardScaler().fit(df_standard)
scaled_data_std = std.transform(df_standard)


# New DataFrame with the scaled data
X_train_std = pd.DataFrame(scaled_data_std, columns=df_standard.columns

# Add the Student IDs back
X_train_std['Student ID'] = student_ids
```

X_val

In [31]:
```
# Extracting the Student IDs
student_ids = new_X_val['Student ID']

# Dropping the student ID column before scaling
df_stadard_val = new_X_val.drop(columns=['Student ID'])

#Scaling
scaled_data_std_val = std.transform(df_stadard_val)


# New DataFrame with the scaled data
X_val_std = pd.DataFrame(scaled_data_std_val, columns=df_stadard_val.cc

# Add the Student IDs back
X_val_std['Student ID'] = student_ids
```

Robust Scaler

X_train

In [32]:
```
# Extracting the Student IDs
student_ids = new_X_train['Student ID']

# Dropping the student ID column before scaling
df_robust = new_X_train.drop(columns=['Student ID'])

#Scaling
rb = RobustScaler().fit(df_robust)
scaled_data_rb = rb.transform(df_robust)


# New DataFrame with the scaled data
X_train_rb = pd.DataFrame(scaled_data_rb, columns=df_robust.columns)

# Add the Student IDs back
X_train_rb['Student ID'] = student_ids
```

X_val

In [33]:
```
# Extracting the Student IDs
student_ids = new_X_val['Student ID']

# Dropping the student ID column before scaling
df_robust_val = new_X_val.drop(columns=['Student ID'])

#Scaling
scaled_data_rb_val = rb.transform(df_robust_val)


# New DataFrame with the scaled data
X_val_rb = pd.DataFrame(scaled_data_rb_val, columns=df_robust_val.colum

# Add the Student IDs back
X_val_rb['Student ID'] = student_ids
```

Feature Selection

Variance

There are no univariate variables for us to drop in `X_train_s` .

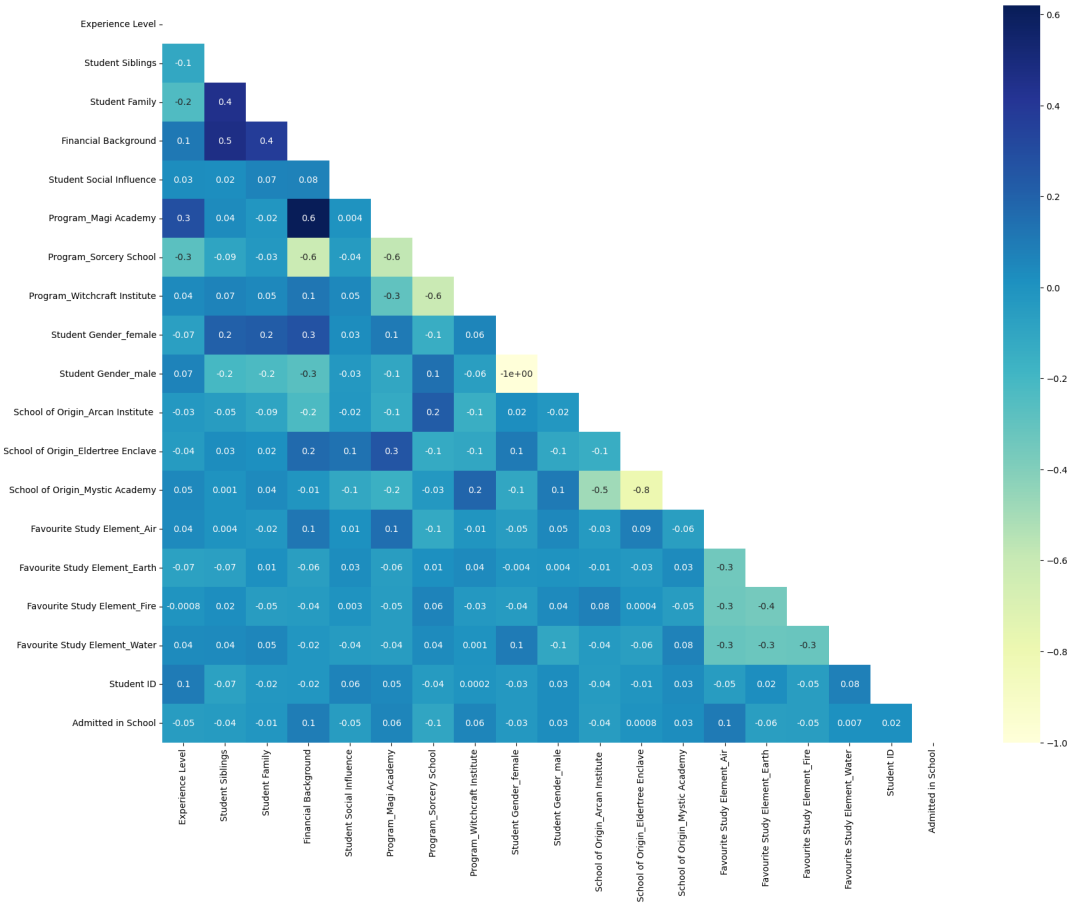
In [34]:
```
X_train_s.var() # checking if any of the numerical variables are univar
```

Out[34]:
```
Experience Level                        0.058885
Student Siblings                        0.219677
Student Family                          0.175171
Financial Background                    0.076965
Student Social Influence                0.090698
Program_Magi Academy                    0.171811
Program_Sorcery School                  0.248709
Program_Witchcraft Institute            0.181672
Student Gender_female                   0.223782
Student Gender_male                     0.223782
School of Origin_Arcan Institute        0.074291
School of Origin_Eldertree Enclave      0.156403
School of Origin_Mystic Academy         0.199413
Favourite Study Element_Air             0.185842
Favourite Study Element_Earth           0.195699
Favourite Study Element_Fire            0.192827
Favourite Study Element_Water           0.176275
Student ID                          65416.458456
dtype: float64
```

Correlation

In [35]:
```
"""all_train_n = X_train_s.join(y_train)"""
```

In [36]:
```
"""cor_spearman_train = all_train_n.corr(method ='spearman')
cor_heatmap(cor_spearman_train)"""
```



From the correlation map above, we can conclude that:

- No independent variable is highly correlated with the target, *'Admitted in School'*, thus we need to use other technique for further insights.
- The following pair of variables are highly correlated, fairly, namely:
  - *'Program_Magi Academy'* vs *'Financial Background'* (0.6)
  - *'Program_Sorcery School'* vs *'Financial Background'* (0.6)
  - *'Program_Sorcery School'* vs *'Program_Magi Academy'* (0.6)
  - *'School of Origin_Eldettree Enclave'* vs _ 'School of Origin_Mystic Academy'_ (0.8)

RFE

As mentioned above, we used RFE to check other insights. Logistic Regression was used as the base model to our Feature Selection as it is one of the simplest models.

In [37]:
```
model = LogisticRegression()
```

In [38]:
```
rfe = RFE(estimator = model, n_features_to_select = 10)
```

In [39]:
```
X_rfe = rfe.fit_transform(X = X_train_s, y = y_train)
```

In [40]:
```
selected_features = pd.Series(rfe.support_, index = X_train_s.columns)
selected_features
```

Out[40]:
```
Experience Level                        True
Student Siblings                        True
Student Family                         False
Financial Background                    True
Student Social Influence                True
Program_Magi Academy                    True
Program_Sorcery School                  True
Program_Witchcraft Institute            True
Student Gender_female                   True
Student Gender_male                     True
School of Origin_Arcan Institute       False
School of Origin_Eldertree Enclave     False
School of Origin_Mystic Academy         True
Favourite Study Element_Air            False
Favourite Study Element_Earth          False
Favourite Study Element_Fire           False
Favourite Study Element_Water          False
Student ID                             False
dtype: bool
```

In [41]:
```
print("RFE picked " + str(sum(selected_features==True)) + " variables a
```

```
RFE picked 10 variables and eliminated the other 8 variables.
```

Lasso

In [42]:
```
reg = LassoCV()
```

In [43]:
```
reg.fit(X_train_s, y_train)
```

Out[43]:
```
LassoCV()
```

In [44]:
```
coef = pd.Series(reg.coef_, index = X_train_s.columns)
coef.sort_values()
```

Out[44]:
```
Experience Level                     -0.261448
Program_Sorcery School               -0.185104
Student Siblings                     -0.050550
Favourite Study Element_Earth        -0.023523
School of Origin_Mystic Academy      -0.022698
Favourite Study Element_Air          -0.004513
Student Family                       -0.000000
Program_Witchcraft Institute         -0.000000
School of Origin_Arcan Institute      0.000000
Student Gender_male                  -0.000000
Student ID                            0.000051
Favourite Study Element_Water         0.002148
Favourite Study Element_Fire          0.011481
School of Origin_Eldertree Enclave    0.016261
Student Social Influence              0.022664
Financial Background                  0.065119
Program_Magi Academy                  0.080567
Student Gender_female                 0.532936
dtype: float64
```

In [45]:
```
coef != 0
```

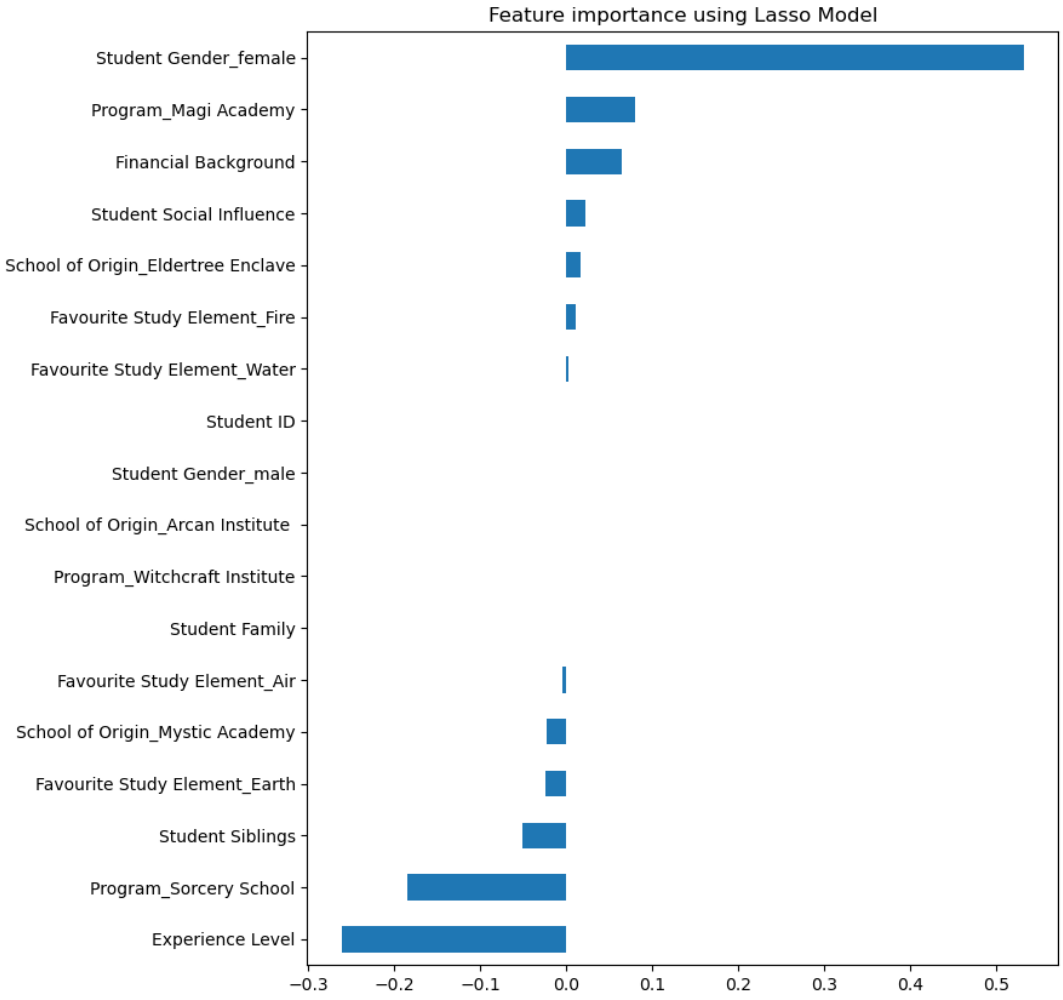
Out[45]:
```
Experience Level                      True
Student Siblings                      True
Student Family                       False
Financial Background                  True
Student Social Influence              True
Program_Magi Academy                  True
Program_Sorcery School                True
Program_Witchcraft Institute         False
Student Gender_female                 True
Student Gender_male                  False
School of Origin_Arcan Institute     False
School of Origin_Eldertree Enclave    True
School of Origin_Mystic Academy       True
Favourite Study Element_Air           True
Favourite Study Element_Earth         True
Favourite Study Element_Fire          True
Favourite Study Element_Water         True
Student ID                            True
dtype: bool
```

In [46]:
```
print("Lasso picked " + str(sum(coef != 0)) + " variables and eliminate
```

```
Lasso picked 14 variables and eliminated the other 4 variables
```

In [47]:
```
"""def plot_importance(coef,name):
    imp_coef = coef.sort_values()
    plt.figure(figsize=(8,10))
    imp_coef.plot(kind = "barh")
    plt.title("Feature importance using " + name + " Model")
    plt.show()"""
```

In [48]:
```
"""plot_importance(coef,'Lasso')"""
```



Random Forest

As we had some ties in the Feature selection we decided to also use a Random Forest to untie our results. We decided to use a Random Forest because is a robust model that severly depends on the features used to achieve good results.

In [49]:
```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=1

rf_classifier.fit(X_train_s, y_train)

feature_importances = rf_classifier.feature_importances_

feature = pd.Series(feature_importances, index = X_train_s.columns)
feature.sort_values(ascending = False)
```

Out[49]:
```
Student Gender_female                0.157602
Student Gender_male                  0.155122
Experience Level                     0.138177
Financial Background                 0.126505
Student ID                           0.116126
Student Social Influence             0.097152
Program_Sorcery School               0.039974
Student Siblings                     0.026549
Program_Magi Academy                 0.023965
Program_Witchcraft Institute         0.018152
Student Family                       0.017194
Favourite Study Element_Fire         0.014361
School of Origin_Mystic Academy      0.013510
Favourite Study Element_Air          0.013173
Favourite Study Element_Water        0.013052
Favourite Study Element_Earth        0.012252
School of Origin_Eldertree Enclave   0.011665
School of Origin_Arcan Institute     0.005469
dtype: float64
```

Final Insights

Predictor	RFE	Lasso	What to do? (One possible way to "solve")
Experience Level	Keep	Keep	Include in the model
Program_Sorcery School	Keep	Keep	Include in the model
Student Siblings	Keep	Keep	Include in the model
School of Origin_Mystic Academy	Keep	Keep	Include in the model
Favourite Study Element_Earth	Discard	Keep	Discard
Favourite Study Element_Air	Discard	Keep	Discard
Student Family	Discard	Discard	Discard
Program_Witchcraft Institute	Keep	Discard	Discard
School of Origin_Arcan Institute	Discard	Discard	Discard
Student Gender_male	Keep	Discard	Include in the model
Favourite Study Element_Water	Discard	Keep	Discard
School of Origin_Eldertree Enclave	Discard	Keep	Discard
Favourite Study Element_Fire	Discard	Keep	Discard
Student Social Influence	Keep	Keep	Include in the model
Financial Background	Keep	Keep	Include in the model
Program_Magi Academy	Keep	Keep	Include in the model

Predictor	RFE	Lasso	What to do? (One possible way to "solve")

In [50]:
```
X_train_r = X_train_s.drop(columns = ["Favourite Study Element_Earth",
                                      "Program_Witchcraft Institute", "
                                      "School of Origin_Eldertree Encla
                                     "Student ID"], axis = 1)
```

In [51]:
```
X_val_r =  X_val_s.drop(columns = ["Favourite Study Element_Earth", "Fa
                                   "Program_Witchcraft Institute", "
                                   "School of Origin_Eldertree Encla
                                  "Student ID"], axis = 1)
```

In [52]:
```
X_train_std = X_train_std.drop(columns = ["Favourite Study Element_Eart
                                          "Program_Witchcraft Institute", "
                                          "School of Origin_Eldertree Encla
                                         "Student ID"], axis = 1)
```

In [53]:
```
X_val_std = X_val_std.drop(columns = ["Favourite Study Element_Earth",
                                      "Program_Witchcraft Institute", "
                                      "School of Origin_Eldertree Encla
                                     "Student ID"], axis = 1)
```

In [54]:
```
X_train_rb = X_train_rb.drop(columns = ["Favourite Study Element_Earth"
                                        "Program_Witchcraft Institute", "
                                        "School of Origin_Eldertree Encla
                                       "Student ID"], axis = 1)
```

In [55]:
```
X_val_rb =  X_val_rb.drop(columns = ["Favourite Study Element_Earth", "
                                     "Program_Witchcraft Institute", "
                                     "School of Origin_Eldertree Encla
                                    "Student ID"], axis = 1)
```

Modeling with Min Max Scaler

Here we decided to do a GridSearch of all our models with Min Max. So we have a better idea of which are the best parameters for each model with this scaler.

Decision Tree Classifier

In [56]:
```
"""base_classifier_dtc1 = DecisionTreeClassifier(random_state=15)

parameters = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10,20,30,50,70,80,100],
    'min_samples_leaf': [1, 2, 4,6,8,10,12,14,16],
    'max_features': [None, 'sqrt', 'log2']}

grid_search_dtc1 = GridSearchCV(base_classifier_dtc1, parameters,verbos
grid_search_dtc1.fit(X_train_r, y_train)
"""
```

```
Fitting 5 folds for each of 5832 candidates, totalling 29160 fits
```

Out[56]:
```
GridSearchCV(estimator=DecisionTreeClassifier(random_state=15), n_jobs
=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [None, 10, 20, 30, 40, 50],
                         'max_features': [None, 'sqrt', 'log2'],
                         'min_samples_leaf': [1, 2, 4, 6, 8, 10, 12, 1
4, 16],
                         'min_samples_split': [2, 5, 10, 20, 30, 50, 7
0, 80,
                                               100],
                         'splitter': ['best', 'random']},
             scoring='f1', verbose=2)
```

In [57]:
```
"""means = grid_search_dtc1.cv_results_['mean_test_score']
stds = grid_search_dtc1.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_dtc1.best
                                      grid_search_dtc1.cv_results_['std_t
```

```
BEST RESULTS: 0.76081 (+/-0.04566) for {'criterion': 'gini', 'max_dept
h': None, 'max_features': None, 'min_samples_leaf': 12, 'min_samples_s
plit': 50, 'splitter': 'best'}
```

Naive Bayes Classifier

In [58]:
```
"""base_classifier_gnb = GaussianNB()

parameters = {'var_smoothing': np.logspace(0, -9, num=100)}

grid_search_gnb = GridSearchCV(base_classifier_gnb, parameters,n_jobs =
grid_search_gnb.fit(X_train_r, y_train)
"""
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

Out[58]:
```
GridSearchCV(estimator=GaussianNB(), n_jobs=-1,
             param_grid={'var_smoothing': array([1.00000000e+00, 8.111
30831e-01, 6.57933225e-01, 5.33669923e-01,
       4.32876128e-01, 3.51119173e-01, 2.84803587e-01, 2.31012970e-01,
       1.87381742e-01, 1.51991108e-01, 1.23284674e-01, 1.00000000e-01,
       8.11130831e-02, 6.57933225e-02, 5.33669923e-02, 4.32876128e-02,
       3.51119173e-02, 2.84803587e-02...
       1.23284674e-07, 1.00000000e-07, 8.11130831e-08, 6.57933225e-08,
       5.33669923e-08, 4.32876128e-08, 3.51119173e-08, 2.84803587e-08,
       2.31012970e-08, 1.87381742e-08, 1.51991108e-08, 1.23284674e-08,
       1.00000000e-08, 8.11130831e-09, 6.57933225e-09, 5.33669923e-09,
       4.32876128e-09, 3.51119173e-09, 2.84803587e-09, 2.31012970e-09,
       1.87381742e-09, 1.51991108e-09, 1.23284674e-09, 1.00000000e-0
9])},
             scoring='f1', verbose=2)
```

In [59]:
```
"""means = grid_search_gnb.cv_results_['mean_test_score']
stds = grid_search_gnb.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_gnb.best_
                                    grid_search_gnb.cv_results_['std_te
```

```
BEST RESULTS: 0.72796 (+/-0.05139) for {'var_smoothing': 0.15199110829
529336}
```

`var_smoothing` is a smoothing parameter that is added to the variances of the features. This parameter helps address situations where the variance of a particular feature is zero in the training data, which would cause issues when calculating probabilities in the Naive Bayes formula.

Logistic Regression

In [60]:
```
"""logistic_regression = LogisticRegression(random_state = 15)

params = {"penalty" : ["l1", "l2", "elasticnet", None],
          "fit_intercept" :  [True, False],
          "solver" : ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
         "max_iter": [30,50, 100, 200,300,400, 500]}

grid_search_lgr = GridSearchCV(logistic_regression, params,n_jobs = -1,
grid_search_lgr.fit(X_train_r, y_train)"""
```

```
Fitting 5 folds for each of 280 candidates, totalling 1400 fits
```

Out[60]:
```
GridSearchCV(estimator=LogisticRegression(random_state=15), n_jobs=-1,
             param_grid={'fit_intercept': [True, False],
                         'max_iter': [30, 50, 100, 200, 300, 400, 50
0],
                         'penalty': ['l1', 'l2', 'elasticnet', None],
                         'solver': ['newton-cg', 'lbfgs', 'liblinear',
'sag',
                                    'saga']},
             scoring='f1', verbose=2)
```

In [61]:
```
"""means = grid_search_lgr.cv_results_['mean_test_score']
stds = grid_search_lgr.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_lgr.best_
                                    grid_search_lgr.cv_results_['std_te
```

```
BEST RESULTS: 0.73030 (+/-0.04289) for {'fit_intercept': True, 'max_it
er': 30, 'penalty': 'l2', 'solver': 'newton-cg'}
```

K Nearest Neighbors

In [62]:
```
"""Knn = KNeighborsClassifier()
parames = {'n_neighbors': [3, 5, 7, 9, 11,13,15,17,19,21],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [10, 20, 30, 40, 50,60,70,80,90,100],
    'metric': ['euclidean', 'manhattan', 'minkowski']}

grid_knn = GridSearchCV(Knn, parames,verbose= 2, n_jobs = -1, scoring='
grid_knn.fit(X_train_r, y_train)"""
```

```
Fitting 5 folds for each of 2400 candidates, totalling 12000 fits
```

Out[62]:
```
GridSearchCV(estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree',
'brute'],
                         'leaf_size': [10, 20, 30, 40, 50, 60, 70, 80,
90, 100],
                         'metric': ['euclidean', 'manhattan', 'minkows
ki'],
                         'n_neighbors': [3, 5, 7, 9, 11, 13, 15, 17, 1
9, 21],
                         'weights': ['uniform', 'distance']},
             scoring='f1', verbose=2)
```

In [63]:
```
"""means = grid_knn.cv_results_['mean_test_score']
stds = grid_knn.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_knn.best_score_,
                                    grid_knn.cv_results_['std_test_scor
```

```
BEST RESULTS: 0.74344 (+/-0.06455) for {'algorithm': 'auto', 'leaf_siz
e': 10, 'metric': 'euclidean', 'n_neighbors': 21, 'weights': 'unifor
m'}
```

Bagging Ensembling

In Ensemble Classifiers we used weak models without any parameters tuning because Ensemble classifiers have a higher performance that way.

Decision Tree

In [64]:
```
"""base_classifier_dtc = DecisionTreeClassifier(random_state=15)

bagging_classifier_dtc = BaggingClassifier(base_classifier_dtc, random_

parameters = {
    'n_estimators': [10,30, 50,80, 100,200,300,500],
    'max_samples': [0.1, 0.2, 0.4, 0.6,0.8,1],
    'max_features': [0.1, 0.2, 0.4, 0.6,0.8,1],
    "bootstrap" : [True, False],
    "bootstrap_features" : [True,False]}


grid_search_dtc = GridSearchCV(bagging_classifier_dtc, parameters, n_jo

grid_search_dtc.fit(X_train_r, y_train)"""
```

```
Fitting 5 folds for each of 1152 candidates, totalling 5760 fits
```

Out[64]:
```
GridSearchCV(estimator=BaggingClassifier(base_estimator=DecisionTreeCl
assifier(random_state=15),
                                         random_state=15),
             n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'bootstrap_features': [True, False],
                         'max_features': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'max_samples': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'n_estimators': [10, 30, 50, 80, 100, 200, 30
0, 500]},
             scoring='f1', verbose=2)
```

In [65]:
```
"""means = grid_search_dtc.cv_results_['mean_test_score']
stds = grid_search_dtc.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_dtc.best_
                                    grid_search_dtc.cv_results_['std_te
```

```
BEST RESULTS: 0.76540 (+/-0.06318) for {'bootstrap': True, 'bootstrap_
features': True, 'max_features': 0.8, 'max_samples': 0.2, 'n_estimator
s': 30}
```

K Neighbors Classifier

In [66]:
```
"""base_classifier_knn = KNeighborsClassifier()

bagging_classifier_knn = BaggingClassifier(base_classifier_knn, random_

parameters = {
    'n_estimators': [10,30, 50,80, 100,150,200],
    'max_samples': [0.1, 0.2, 0.4, 0.6,0.8,1],
    'max_features': [0.1, 0.2, 0.4, 0.6,0.8,1],
    "bootstrap" : [True, False],
    "bootstrap_features" : [True,False]}


grid_search_knn = GridSearchCV(bagging_classifier_knn, parameters,n_job

grid_search_knn.fit(X_train_r, y_train)"""
```

```
Fitting 5 folds for each of 1008 candidates, totalling 5040 fits
```

Out[66]:
```
GridSearchCV(estimator=BaggingClassifier(base_estimator=KNeighborsClas
sifier(),
                                         random_state=15),
             n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'bootstrap_features': [True, False],
                         'max_features': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'max_samples': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'n_estimators': [10, 30, 50, 80, 100, 150, 20
0]},
             scoring='f1', verbose=2)
```

In [67]:
```
"""means = grid_search_knn.cv_results_['mean_test_score']
stds = grid_search_knn.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_knn.best_
                                    grid_search_knn.cv_results_['std_te
```

```
BEST RESULTS: 0.75823 (+/-0.06966) for {'bootstrap': True, 'bootstrap_
features': False, 'max_features': 0.8, 'max_samples': 0.6, 'n_estimato
rs': 30}
```

Logistic Regression

In [68]:
```
"""base_classifier_lg = LogisticRegression(random_state=15)

bagging_classifier_lg = BaggingClassifier(base_classifier_lg, random_st

parameters = {
    'n_estimators': [10,30, 50,80, 100,150,200],
    'max_samples': [0.1, 0.2, 0.4, 0.6,0.8,1],
    'max_features': [0.1, 0.2, 0.4, 0.6,0.8,1],
    "bootstrap" : [True, False],
    "bootstrap_features" : [True,False]}


grid_search_lg = GridSearchCV(bagging_classifier_lg, parameters,n_jobs

grid_search_lg.fit(X_train_r, y_train)"""
```

```
Fitting 5 folds for each of 1008 candidates, totalling 5040 fits
```

Out[68]:
```
GridSearchCV(estimator=BaggingClassifier(base_estimator=LogisticRegres
sion(random_state=15),
                                         random_state=15),
             n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'bootstrap_features': [True, False],
                         'max_features': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'max_samples': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'n_estimators': [10, 30, 50, 80, 100, 150, 20
0]},
             scoring='f1', verbose=2)
```

In [69]:
```
"""means = grid_search_lg.cv_results_['mean_test_score']
stds = grid_search_lg.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_lg.best_s
                                    grid_search_lg.cv_results_['std_tes
```

```
BEST RESULTS: 0.73091 (+/-0.08390) for {'bootstrap': True, 'bootstrap_
features': True, 'max_features': 0.4, 'max_samples': 0.1, 'n_estimator
s': 10}
```

Boosting Ensembling

Decision Trees

In [70]:
```
"""base_model_dtc_boost = DecisionTreeClassifier(max_depth=1, random_st

adaboost_model_dtc = AdaBoostClassifier(base_model_dtc_boost, random_st

parameters = {'n_estimators': [2,5,10,20,30,50,100,150],
              'learning_rate': [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0],
              'algorithm':["SAMME","SAMME.R"]}

grid_search_boost_dtc = GridSearchCV(adaboost_model_dtc, parameters,n_j

grid_search_boost_dtc.fit(X_train_r, y_train)"""
```

```
Fitting 5 folds for each of 160 candidates, totalling 800 fits
```

Out[70]:
```
GridSearchCV(estimator=AdaBoostClassifier(base_estimator=DecisionTreeC
lassifier(max_depth=1,

random_state=15),
                                          random_state=15),
             n_jobs=-1,
             param_grid={'algorithm': ['SAMME', 'SAMME.R'],
                         'learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5, 0.
6, 0.7,
                                           0.8, 0.9, 1.0],
                         'n_estimators': [2, 5, 10, 20, 30, 50, 100, 1
50]},
             scoring='f1', verbose=2)
```

In [71]:
```
"""means = grid_search_boost_dtc.cv_results_['mean_test_score']
stds = grid_search_boost_dtc.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_boost_dtc
                                    grid_search_boost_dtc.cv_results_['
```

```
BEST RESULTS: 0.76701 (+/-0.06701) for {'algorithm': 'SAMME', 'learnin
g_rate': 1.0, 'n_estimators': 30}
```

Logistic Regression

In [72]:
```
"""base_model_lg_boost = LogisticRegression(random_state=15)

adaboost_model_lg = AdaBoostClassifier(base_model_lg_boost, random_stat

parameters = {'n_estimators': [2,5,10,20,30,50,100,150],
              'learning_rate': [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0],
              'algorithm':["SAMME","SAMME.R"]}

grid_search_boost_lg = GridSearchCV(adaboost_model_lg, parameters, n_jo

grid_search_boost_lg.fit(X_train_r, y_train)"""
```

```
Fitting 5 folds for each of 160 candidates, totalling 800 fits
```

Out[72]:
```
GridSearchCV(estimator=AdaBoostClassifier(base_estimator=LogisticRegre
ssion(random_state=15),
                                          random_state=15),
             n_jobs=-1,
             param_grid={'algorithm': ['SAMME', 'SAMME.R'],
                         'learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5, 0.
6, 0.7,
                                           0.8, 0.9, 1.0],
                         'n_estimators': [2, 5, 10, 20, 30, 50, 100, 1
50]},
             scoring='f1', verbose=2)
```

In [73]:
```
"""means = grid_search_boost_lg.cv_results_['mean_test_score']
stds = grid_search_boost_lg.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_boost_lg.
                                    grid_search_boost_lg.cv_results_['s
```

```
BEST RESULTS: 0.73283 (+/-0.04483) for {'algorithm': 'SAMME', 'learnin
g_rate': 0.4, 'n_estimators': 50}
```

Random Forest

In [74]:
```
"""rf_classifier = RandomForestClassifier(random_state = 15)

parameters = {
    'n_estimators': [30,50,80, 100, 200,250],
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10, 20, 30,40,50],
    'min_samples_split': [5,10,20,30,50,70],
    'min_samples_leaf':[4,6,8,10,12,14,16],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False]}


grid_search_rf = GridSearchCV(estimator=rf_classifier, verbose = 2,n_jo
grid_search_rf.fit(X_train_r, y_train)"""
```

```
Fitting 2 folds for each of 18144 candidates, totalling 36288 fits
```

Out[74]:
```
GridSearchCV(cv=2, estimator=RandomForestClassifier(random_state=15),
n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'criterion': ['gini', 'entropy'],
                         'max_depth': [5, 10, 20, 30, 40, 50],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_leaf': [4, 6, 8, 10, 12, 14, 1
6],
                         'min_samples_split': [5, 10, 20, 30, 50, 70],
                         'n_estimators': [30, 50, 80, 100, 200, 250]},
             scoring='f1', verbose=2)
```

In [75]:
```
"""means = grid_search_rf.cv_results_['mean_test_score']
stds = grid_search_rf.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_rf.best_s
                                    grid_search_rf.cv_results_['std_tes
```

```
BEST RESULTS: 0.74385 (+/-0.04186) for {'bootstrap': False, 'criterio
n': 'gini', 'max_depth': 20, 'max_features': 'auto', 'min_samples_lea
f': 4, 'min_samples_split': 20, 'n_estimators': 30}
```

Stacking Ensembling

To do our stacking we decided to use all our previous models with the parameters defined by the gridsearch. We also decided to use a boosting ensemble classifier as a meta classifier so that the agregattion of our results is the best possible.

In [76]:
```
"""base_classifier_dtc = DecisionTreeClassifier(random_state=15)
base_model_dtc_boost = DecisionTreeClassifier(max_depth=1, random_state

est = [('decision_tree', DecisionTreeClassifier(criterion='gini', max_d
       ('gaussian_nb', GaussianNB(var_smoothing=0.15199110829529336)),
       ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=10, met
       ('logistic_regression', LogisticRegression(fit_intercept=True, m
      ('bagging', BaggingClassifier(base_classifier_dtc,bootstrap= True
      ('boosting',AdaBoostClassifier(base_model_dtc_boost,algorithm='SA
      ('random_forest', RandomForestClassifier(random_state = 15, boots

meta_classifier = AdaBoostClassifier(n_estimators=50, random_state = 15

stacking_clf = StackingClassifier(estimators=est, final_estimator=meta_
stacking_clf.fit(X_train_r, y_train)

y_pred = stacking_clf.predict(X_val_r)
f1 = f1_score(y_val, y_pred, average='weighted')
print(f"F1 Score: {f1}")"""
```

```
F1 Score: 0.7672846495145174
```

We then repeated everything for Standard And Robust Scalers

Modeling with Standard Scaler

Decision Trees

In [77]:
```
"""base_classifier_dtc1 = DecisionTreeClassifier(random_state=15)

parameters = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10,20,30,50,70,80],
    'min_samples_leaf': [1, 2, 4,6,8,10,12,14,16],
    'max_features': [None, 'sqrt', 'log2']}

grid_search_dtc1 = GridSearchCV(base_classifier_dtc1, parameters,n_jobs
grid_search_dtc1.fit(X_train_std, y_train)"""
```

```
Fitting 5 folds for each of 5184 candidates, totalling 25920 fits
```

Out[77]:
```
GridSearchCV(estimator=DecisionTreeClassifier(random_state=15), n_jobs
=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [None, 10, 20, 30, 40, 50],
                         'max_features': [None, 'sqrt', 'log2'],
                         'min_samples_leaf': [1, 2, 4, 6, 8, 10, 12, 1
4, 16],
                         'min_samples_split': [2, 5, 10, 20, 30, 50, 7
0, 80],
                         'splitter': ['best', 'random']},
             scoring='f1', verbose=2)
```

In [78]:
```
"""means = grid_search_dtc1.cv_results_['mean_test_score']
stds = grid_search_dtc1.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_dtc1.best
                                    grid_search_dtc1.cv_results_['std_t
```

```
BEST RESULTS: 0.76081 (+/-0.04566) for {'criterion': 'gini', 'max_dept
h': None, 'max_features': None, 'min_samples_leaf': 12, 'min_samples_s
plit': 50, 'splitter': 'best'}
```

Naive Bayes Classifier

In [79]:
```
"""base_classifier_gnb = GaussianNB()

parameters = {'var_smoothing': np.logspace(0, -9, num=100)}

grid_search_gnb = GridSearchCV(base_classifier_gnb, parameters,n_jobs=
grid_search_gnb.fit(X_train_std, y_train)
"""
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

Out[79]:
```
GridSearchCV(estimator=GaussianNB(), n_jobs=-1,
             param_grid={'var_smoothing': array([1.00000000e+00, 8.111
30831e-01, 6.57933225e-01, 5.33669923e-01,
       4.32876128e-01, 3.51119173e-01, 2.84803587e-01, 2.31012970e-01,
       1.87381742e-01, 1.51991108e-01, 1.23284674e-01, 1.00000000e-01,
       8.11130831e-02, 6.57933225e-02, 5.33669923e-02, 4.32876128e-02,
       3.51119173e-02, 2.84803587e-02...
       1.23284674e-07, 1.00000000e-07, 8.11130831e-08, 6.57933225e-08,
       5.33669923e-08, 4.32876128e-08, 3.51119173e-08, 2.84803587e-08,
       2.31012970e-08, 1.87381742e-08, 1.51991108e-08, 1.23284674e-08,
       1.00000000e-08, 8.11130831e-09, 6.57933225e-09, 5.33669923e-09,
       4.32876128e-09, 3.51119173e-09, 2.84803587e-09, 2.31012970e-09,
       1.87381742e-09, 1.51991108e-09, 1.23284674e-09, 1.00000000e-0
9])},
             scoring='f1', verbose=2)
```

In [80]:
```
"""means = grid_search_gnb.cv_results_['mean_test_score']
stds = grid_search_gnb.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_gnb.best_
                                    grid_search_gnb.cv_results_['std_te
```

```
BEST RESULTS: 0.72598 (+/-0.05177) for {'var_smoothing': 0.43287612810
830584}
```

Logistic Regression

In [81]:
```
"""logistic_regression = LogisticRegression(random_state = 15)

params = {"penalty" : ["l1", "l2", "elasticnet", None],
          "fit_intercept" :  [True, False],
          "solver" : ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
         "max_iter": [30,50, 100, 200,300]}

grid_search_lgr = GridSearchCV(logistic_regression, params,verbose = 2,
grid_search_lgr.fit(X_train_std, y_train)"""
```

```
Fitting 5 folds for each of 200 candidates, totalling 1000 fits
```

Out[81]:
```
GridSearchCV(estimator=LogisticRegression(random_state=15), n_jobs=-1,
             param_grid={'fit_intercept': [True, False],
                         'max_iter': [30, 50, 100, 200, 300],
                         'penalty': ['l1', 'l2', 'elasticnet', None],
                         'solver': ['newton-cg', 'lbfgs', 'liblinear',
'sag',
                                    'saga']},
             scoring='f1', verbose=2)
```

In [82]:
```
"""means = grid_search_lgr.cv_results_['mean_test_score']
stds = grid_search_lgr.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_lgr.best_
                                    grid_search_lgr.cv_results_['std_te
```

```
BEST RESULTS: 0.73239 (+/-0.04513) for {'fit_intercept': True, 'max_it
er': 30, 'penalty': 'l1', 'solver': 'liblinear'}
```

K Nearest Neighbors

In [83]:
```
"""Knn = KNeighborsClassifier()
parames = {'n_neighbors': [3, 5, 7, 9, 11,13,15,17],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [10, 20, 30, 40, 50,60,70,80,90,100],
    'metric': ['euclidean', 'manhattan', 'minkowski']}

grid_knn = GridSearchCV(Knn, parames,verbose = 2,n_jobs = -1 ,scoring='
grid_knn.fit(X_train_std, y_train)"""
```

```
Fitting 5 folds for each of 1920 candidates, totalling 9600 fits
```

Out[83]:
```
GridSearchCV(estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree',
'brute'],
                         'leaf_size': [10, 20, 30, 40, 50, 60, 70, 80,
90, 100],
                         'metric': ['euclidean', 'manhattan', 'minkows
ki'],
                         'n_neighbors': [3, 5, 7, 9, 11, 13, 15, 17],
                         'weights': ['uniform', 'distance']},
             scoring='f1', verbose=2)
```

In [84]:
```
"""means = grid_knn.cv_results_['mean_test_score']
stds = grid_knn.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_knn.best_score_,
                                      grid_knn.cv_results_['std_test_scor
```

```
BEST RESULTS: 0.75520 (+/-0.05483) for {'algorithm': 'auto', 'leaf_siz
e': 10, 'metric': 'manhattan', 'n_neighbors': 7, 'weights': 'uniform'}
```

Bagging Ensembling

Decision Tree

In [85]:
```
"""base_classifier_dtc = DecisionTreeClassifier(random_state=15)

bagging_classifier_dtc = BaggingClassifier(base_classifier_dtc, random_

parameters = {
    'n_estimators': [10,30, 50,80, 100,200],
    'max_samples': [0.1, 0.2, 0.4, 0.6,0.8,1],
    'max_features': [0.1, 0.2, 0.4, 0.6,0.8,1],
    "bootstrap" : [True, False],
    "bootstrap_features" : [True,False]}


grid_search_dtc = GridSearchCV(bagging_classifier_dtc,verbose = 2,n_job

grid_search_dtc.fit(X_train_std, y_train)"""
```

```
Fitting 5 folds for each of 864 candidates, totalling 4320 fits
```

Out[85]:
```
GridSearchCV(estimator=BaggingClassifier(base_estimator=DecisionTreeCl
assifier(random_state=15),
                                         random_state=15),
             n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'bootstrap_features': [True, False],
                         'max_features': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'max_samples': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'n_estimators': [10, 30, 50, 80, 100, 200]},
             scoring='f1', verbose=2)
```

In [86]:
```
"""means = grid_search_dtc.cv_results_['mean_test_score']
stds = grid_search_dtc.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_dtc.best_
                                    grid_search_dtc.cv_results_['std_te
```

```
BEST RESULTS: 0.76311 (+/-0.06567) for {'bootstrap': True, 'bootstrap_
features': True, 'max_features': 0.8, 'max_samples': 0.2, 'n_estimator
s': 30}
```

K Nearest Neighbors

In [87]:
```
"""base_classifier_knn = KNeighborsClassifier()

bagging_classifier_knn = BaggingClassifier(base_classifier_knn, random_

parameters = {
    'n_estimators': [10,30, 50,80, 100,150,200,300],
    'max_samples': [0.1, 0.2, 0.4, 0.6,0.8,1],
    'max_features': [0.1, 0.2, 0.4, 0.6,0.8,1],
    "bootstrap" : [True, False],
    "bootstrap_features" : [True,False]}


grid_search_knn = GridSearchCV(bagging_classifier_knn,verbose = 2,n_job

grid_search_knn.fit(X_train_std, y_train)"""
```

```
Fitting 5 folds for each of 1152 candidates, totalling 5760 fits
```

Out[87]:
```
GridSearchCV(estimator=BaggingClassifier(base_estimator=KNeighborsClas
sifier(),
                                         random_state=15),
             n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'bootstrap_features': [True, False],
                         'max_features': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'max_samples': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'n_estimators': [10, 30, 50, 80, 100, 150, 20
0, 300]},
             scoring='f1', verbose=2)
```

In [88]:
```
"""means = grid_search_knn.cv_results_['mean_test_score']
stds = grid_search_knn.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_knn.best_
                                    grid_search_knn.cv_results_['std_te
```

```
BEST RESULTS: 0.75809 (+/-0.06674) for {'bootstrap': True, 'bootstrap_
features': False, 'max_features': 0.8, 'max_samples': 0.8, 'n_estimato
rs': 200}
```

Logistic Regression

In [89]:
```
"""base_classifier_lg = LogisticRegression(random_state=15)

bagging_classifier_lg = BaggingClassifier(base_classifier_lg, random_st

parameters = {
    'n_estimators': [10,30, 50,80, 100,150,200,300],
    'max_samples': [0.1, 0.2, 0.4, 0.6,0.8,1],
    'max_features': [0.1, 0.2, 0.4, 0.6,0.8,1],
    "bootstrap" : [True, False],
    "bootstrap_features" : [True,False]}


grid_search_lg = GridSearchCV(bagging_classifier_lg,verbose = 2,n_jobs

grid_search_lg.fit(X_train_std, y_train)"""
```

```
Fitting 5 folds for each of 1152 candidates, totalling 5760 fits
```

Out[89]:
```
GridSearchCV(estimator=BaggingClassifier(base_estimator=LogisticRegres
sion(random_state=15),
                                         random_state=15),
             n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'bootstrap_features': [True, False],
                         'max_features': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'max_samples': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'n_estimators': [10, 30, 50, 80, 100, 150, 20
0, 300]},
             scoring='f1', verbose=2)
```

In [90]:
```
"""means = grid_search_lg.cv_results_['mean_test_score']
stds = grid_search_lg.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_lg.best_s
                                    grid_search_lg.cv_results_['std_tes
```

```
BEST RESULTS: 0.73382 (+/-0.04044) for {'bootstrap': True, 'bootstrap_
features': False, 'max_features': 0.8, 'max_samples': 0.8, 'n_estimato
rs': 80}
```

Boosting Ensembling

Decision Tree

In [91]:
```
"""base_model_dtc_boost = DecisionTreeClassifier(max_depth=1, random_st

adaboost_model_dtc = AdaBoostClassifier(base_model_dtc_boost, random_st

parameters = {'n_estimators': [2,5,10,20,30,50,100,150],
              'learning_rate': [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0],
              'algorithm':["SAMME","SAMME.R"]}

grid_search_boost_dtc = GridSearchCV(adaboost_model_dtc,verbose = 2,n_j

grid_search_boost_dtc.fit(X_train_std, y_train)"""
```

```
Fitting 5 folds for each of 160 candidates, totalling 800 fits
```

Out[91]:
```
GridSearchCV(estimator=AdaBoostClassifier(base_estimator=DecisionTreeC
lassifier(max_depth=1,

random_state=15),
                                          random_state=15),
             n_jobs=-1,
             param_grid={'algorithm': ['SAMME', 'SAMME.R'],
                         'learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5, 0.
6, 0.7,
                                           0.8, 0.9, 1.0],
                         'n_estimators': [2, 5, 10, 20, 30, 50, 100, 1
50]},
             scoring='f1', verbose=2)
```

In [92]:
```
"""means = grid_search_boost_dtc.cv_results_['mean_test_score']
stds = grid_search_boost_dtc.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_boost_dtc
                                    grid_search_boost_dtc.cv_results_['
```

```
BEST RESULTS: 0.76701 (+/-0.06701) for {'algorithm': 'SAMME', 'learnin
g_rate': 1.0, 'n_estimators': 30}
```

Logistic Regression

In [93]:
```
"""base_model_lg_boost = LogisticRegression(random_state=15)

adaboost_model_lg = AdaBoostClassifier(base_model_lg_boost, random_stat

parameters = {'n_estimators': [2,5,10,20,30,50,100,150],
              'learning_rate': [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8],
              'algorithm':["SAMME","SAMME.R"]}

grid_search_boost_lg = GridSearchCV(adaboost_model_lg, verbose = 2,n_jo

grid_search_boost_lg.fit(X_train_std, y_train)"""
```

```
Fitting 5 folds for each of 128 candidates, totalling 640 fits
```

Out[93]:
```
GridSearchCV(estimator=AdaBoostClassifier(base_estimator=LogisticRegre
ssion(random_state=15),
                                          random_state=15),
             n_jobs=-1,
             param_grid={'algorithm': ['SAMME', 'SAMME.R'],
                         'learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5, 0.
6, 0.7,
                                           0.8],
                         'n_estimators': [2, 5, 10, 20, 30, 50, 100, 1
50]},
             scoring='f1', verbose=2)
```

In [94]:
```
"""means = grid_search_boost_lg.cv_results_['mean_test_score']
stds = grid_search_boost_lg.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_boost_lg.
                                    grid_search_boost_lg.cv_results_['s
```

```
BEST RESULTS: 0.73348 (+/-0.06500) for {'algorithm': 'SAMME', 'learnin
g_rate': 0.7, 'n_estimators': 20}
```

Random Forest

In [95]:
```
"""rf_classifier = RandomForestClassifier(random_state = 15,verbose = 2

parameters = {
    'n_estimators': [30,50,80, 100, 200,250],
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10, 20, 30,40,50],
    'min_samples_split': [5,10,20,30,50,70],
    'min_samples_leaf':[4,6,8,10,12,14,16],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False]}


grid_search_rf = GridSearchCV(estimator=rf_classifier, verbose = 2,n_jo
grid_search_rf.fit(X_train_std, y_train)"""
```

```
Fitting 2 folds for each of 18144 candidates, totalling 36288 fits
building tree 1 of 30
building tree 2 of 30
building tree 3 of 30
building tree 4 of 30
building tree 5 of 30
building tree 6 of 30
building tree 7 of 30
building tree 8 of 30
building tree 9 of 30
building tree 10 of 30
building tree 11 of 30
building tree 12 of 30
building tree 13 of 30
building tree 14 of 30
building tree 15 of 30
building tree 16 of 30
building tree 17 of 30
building tree 18 of 30
```

In [96]:
```
"""means = grid_search_rf.cv_results_['mean_test_score']
stds = grid_search_rf.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_rf.best_s
                                    grid_search_rf.cv_results_['std_tes
```

```
BEST RESULTS: 0.74385 (+/-0.04186) for {'bootstrap': False, 'criterio
n': 'gini', 'max_depth': 20, 'max_features': 'auto', 'min_samples_lea
f': 4, 'min_samples_split': 20, 'n_estimators': 30}
```

Stacking Ensembling

In [97]:
```
"""base_classifier_dtc = DecisionTreeClassifier(random_state=15)
base_model_dtc_boost = DecisionTreeClassifier(max_depth=1, random_state

est = [('decision_tree', DecisionTreeClassifier(criterion='gini', max_d
       ('gaussian_nb', GaussianNB(var_smoothing=0.43287612810830584)),
       ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=10, met
       ('logistic_regression', LogisticRegression(fit_intercept=True, m
      ('bagging', BaggingClassifier(base_classifier_dtc,bootstrap= True
      ('boosting',AdaBoostClassifier(base_model_dtc_boost,algorithm='SA
      ('random_forest', RandomForestClassifier(bootstrap=False, criteri

meta_classifier = AdaBoostClassifier(n_estimators=50, random_state = 15

stacking_clf = StackingClassifier(estimators=est, final_estimator=meta_
stacking_clf.fit(X_train_std, y_train)

y_pred = stacking_clf.predict(X_val_std)
f1 = f1_score(y_val, y_pred, average='weighted')
print(f"F1 Score: {f1}")"""
```

```
F1 Score: 0.7530249733330076
```

Modeling with Robust Scaler

Decision Trees

In [98]:
```
"""base_classifier_dtc1 = DecisionTreeClassifier(random_state=15)

parameters = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10,20,30,50,70,80],
    'min_samples_leaf': [1, 2, 4,6,8,10,12,14,16],
    'max_features': [None, 'sqrt', 'log2']}

grid_search_dtc1 = GridSearchCV(base_classifier_dtc1,verbose = 2,n_jobs
grid_search_dtc1.fit(X_train_rb, y_train)"""
```

```
Fitting 5 folds for each of 5184 candidates, totalling 25920 fits
```

Out[98]:
```
GridSearchCV(estimator=DecisionTreeClassifier(random_state=15), n_jobs
=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [None, 10, 20, 30, 40, 50],
                         'max_features': [None, 'sqrt', 'log2'],
                         'min_samples_leaf': [1, 2, 4, 6, 8, 10, 12, 1
4, 16],
                         'min_samples_split': [2, 5, 10, 20, 30, 50, 7
0, 80],
                         'splitter': ['best', 'random']},
             scoring='f1', verbose=2)
```

In [99]:
```
"""means = grid_search_dtc1.cv_results_['mean_test_score']
stds = grid_search_dtc1.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_dtc1.best
                                    grid_search_dtc1.cv_results_['std_t
```

```
BEST RESULTS: 0.76081 (+/-0.04566) for {'criterion': 'gini', 'max_dept
h': None, 'max_features': None, 'min_samples_leaf': 12, 'min_samples_s
plit': 50, 'splitter': 'best'}
```

Naive Bayes Classifier

In [100]:
```
"""base_classifier_gnb = GaussianNB()

parameters = {'var_smoothing': np.logspace(0, -9, num=100)}

grid_search_gnb = GridSearchCV(base_classifier_gnb, verbose = 2,n_jobs
grid_search_gnb.fit(X_train_rb, y_train)
"""
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

Out[100]:
```
GridSearchCV(estimator=GaussianNB(), n_jobs=-1,
             param_grid={'var_smoothing': array([1.00000000e+00, 8.111
30831e-01, 6.57933225e-01, 5.33669923e-01,
       4.32876128e-01, 3.51119173e-01, 2.84803587e-01, 2.31012970e-01,
       1.87381742e-01, 1.51991108e-01, 1.23284674e-01, 1.00000000e-01,
       8.11130831e-02, 6.57933225e-02, 5.33669923e-02, 4.32876128e-02,
       3.51119173e-02, 2.84803587e-02...
       1.23284674e-07, 1.00000000e-07, 8.11130831e-08, 6.57933225e-08,
       5.33669923e-08, 4.32876128e-08, 3.51119173e-08, 2.84803587e-08,
       2.31012970e-08, 1.87381742e-08, 1.51991108e-08, 1.23284674e-08,
       1.00000000e-08, 8.11130831e-09, 6.57933225e-09, 5.33669923e-09,
       4.32876128e-09, 3.51119173e-09, 2.84803587e-09, 2.31012970e-09,
       1.87381742e-09, 1.51991108e-09, 1.23284674e-09, 1.00000000e-0
9])},
             scoring='f1', verbose=2)
```

In [101]:
```
"""means = grid_search_gnb.cv_results_['mean_test_score']
stds = grid_search_gnb.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_gnb.best_
                                    grid_search_gnb.cv_results_['std_te
```

```
BEST RESULTS: 0.71951 (+/-0.04594) for {'var_smoothing': 0.01519911082
952933}
```

Logistic Regression

In [102]:
```
"""logistic_regression = LogisticRegression(random_state = 15)

params = {"penalty" : ["l1", "l2", "elasticnet", None],
          "fit_intercept" :  [True, False],
          "solver" : ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
         "max_iter": [30,50, 100, 200,300]}

grid_search_lgr = GridSearchCV(logistic_regression, verbose = 2,n_jobs
grid_search_lgr.fit(X_train_rb, y_train)"""
```

```
Fitting 5 folds for each of 200 candidates, totalling 1000 fits
```

Out[102]:
```
GridSearchCV(estimator=LogisticRegression(random_state=15), n_jobs=-1,
             param_grid={'fit_intercept': [True, False],
                         'max_iter': [30, 50, 100, 200, 300],
                         'penalty': ['l1', 'l2', 'elasticnet', None],
                         'solver': ['newton-cg', 'lbfgs', 'liblinear',
'sag',
                                    'saga']},
             scoring='f1', verbose=2)
```

In [103]:
```
"""means = grid_search_lgr.cv_results_['mean_test_score']
stds = grid_search_lgr.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_lgr.best_
                                    grid_search_lgr.cv_results_['std_te
```

```
BEST RESULTS: 0.73239 (+/-0.04513) for {'fit_intercept': True, 'max_it
er': 30, 'penalty': 'l1', 'solver': 'liblinear'}
```

K Nearest Neighbors

In [104]:
```
"""Knn = KNeighborsClassifier()
parames = {'n_neighbors': [3, 5, 7, 9, 11,13,15,17],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [10, 20, 30, 40, 50,60,70,80,90,100],
    'metric': ['euclidean', 'manhattan', 'minkowski']}

grid_knn = GridSearchCV(Knn,param_grid= parames, verbose = 2,n_jobs = -
grid_knn.fit(X_train_rb, y_train)"""
```

```
Fitting 5 folds for each of 1920 candidates, totalling 9600 fits
```

Out[104]:
```
GridSearchCV(estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree',
'brute'],
                         'leaf_size': [10, 20, 30, 40, 50, 60, 70, 80,
90, 100],
                         'metric': ['euclidean', 'manhattan', 'minkows
ki'],
                         'n_neighbors': [3, 5, 7, 9, 11, 13, 15, 17],
                         'weights': ['uniform', 'distance']},
             scoring='f1', verbose=2)
```

In [105]:
```
"""means = grid_knn.cv_results_['mean_test_score']
stds = grid_knn.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_knn.best_score_,
                                      grid_knn.cv_results_['std_test_scor
```

```
BEST RESULTS: 0.75539 (+/-0.06401) for {'algorithm': 'auto', 'leaf_siz
e': 10, 'metric': 'euclidean', 'n_neighbors': 15, 'weights': 'distanc
e'}
```

Bagging Ensembling

Decision Tree

In [106]:
```
"""base_classifier_dtc = DecisionTreeClassifier(random_state=15)

bagging_classifier_dtc = BaggingClassifier(base_classifier_dtc, random_

parameters = {
    'n_estimators': [10,30, 50,80, 100,200],
    'max_samples': [0.1, 0.2, 0.4, 0.6,0.8,1],
    'max_features': [0.1, 0.2, 0.4, 0.6,0.8,1],
    "bootstrap" : [True, False],
    "bootstrap_features" : [True,False]}


grid_search_dtc = GridSearchCV(bagging_classifier_dtc,verbose = 2,n_job

grid_search_dtc.fit(X_train_rb, y_train)"""
```

```
Fitting 5 folds for each of 864 candidates, totalling 4320 fits
```

Out[106]:
```
GridSearchCV(estimator=BaggingClassifier(base_estimator=DecisionTreeCl
assifier(random_state=15),
                                         random_state=15),
             n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'bootstrap_features': [True, False],
                         'max_features': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'max_samples': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'n_estimators': [10, 30, 50, 80, 100, 200]},
             scoring='f1', verbose=2)
```

In [107]:
```
"""means = grid_search_dtc.cv_results_['mean_test_score']
stds = grid_search_dtc.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_dtc.best_
                                    grid_search_dtc.cv_results_['std_te
```

```
BEST RESULTS: 0.76555 (+/-0.06743) for {'bootstrap': True, 'bootstrap_
features': True, 'max_features': 0.8, 'max_samples': 0.2, 'n_estimator
s': 30}
```

K Nearest Neighbors

In [108]:
```
"""base_classifier_knn = KNeighborsClassifier()

bagging_classifier_knn = BaggingClassifier(base_classifier_knn, random_

parameters = {
    'n_estimators': [10,30, 50,80, 100,150,200,300],
    'max_samples': [0.1, 0.2, 0.4, 0.6,0.8,1],
    'max_features': [0.1, 0.2, 0.4, 0.6,0.8,1],
    "bootstrap" : [True, False],
    "bootstrap_features" : [True,False]}


grid_search_knn = GridSearchCV(bagging_classifier_knn,verbose = 2,n_job

grid_search_knn.fit(X_train_rb, y_train)"""
```

```
Fitting 5 folds for each of 1152 candidates, totalling 5760 fits
```

Out[108]:
```
GridSearchCV(estimator=BaggingClassifier(base_estimator=KNeighborsClas
sifier(),
                                         random_state=15),
             n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'bootstrap_features': [True, False],
                         'max_features': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'max_samples': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'n_estimators': [10, 30, 50, 80, 100, 150, 20
0, 300]},
             scoring='f1', verbose=2)
```

In [109]:
```
"""means = grid_search_knn.cv_results_['mean_test_score']
stds = grid_search_knn.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_knn.best_
                                    grid_search_knn.cv_results_['std_te
```

```
BEST RESULTS: 0.76455 (+/-0.08524) for {'bootstrap': False, 'bootstrap
_features': True, 'max_features': 0.6, 'max_samples': 0.8, 'n_estimato
rs': 10}
```

Logistic Regression

In [110]:
```
"""base_classifier_lg = LogisticRegression(random_state=15)

bagging_classifier_lg = BaggingClassifier(base_classifier_lg, random_st

parameters = {
    'n_estimators': [10,30, 50,80, 100,150,200,300],
    'max_samples': [0.1, 0.2, 0.4, 0.6,0.8,1],
    'max_features': [0.1, 0.2, 0.4, 0.6,0.8,1],
    "bootstrap" : [True, False],
    "bootstrap_features" : [True,False]}


grid_search_lg = GridSearchCV(bagging_classifier_lg,verbose = 2,n_jobs

grid_search_lg.fit(X_train_rb, y_train)"""
```

```
Fitting 5 folds for each of 1152 candidates, totalling 5760 fits
```

Out[110]:
```
GridSearchCV(estimator=BaggingClassifier(base_estimator=LogisticRegres
sion(random_state=15),
                                         random_state=15),
             n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'bootstrap_features': [True, False],
                         'max_features': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'max_samples': [0.1, 0.2, 0.4, 0.6, 0.8, 1],
                         'n_estimators': [10, 30, 50, 80, 100, 150, 20
0, 300]},
             scoring='f1', verbose=2)
```

In [111]:
```
"""means = grid_search_lg.cv_results_['mean_test_score']
stds = grid_search_lg.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_lg.best_s
                                    grid_search_lg.cv_results_['std_tes
```

```
BEST RESULTS: 0.74031 (+/-0.04783) for {'bootstrap': True, 'bootstrap_
features': False, 'max_features': 0.8, 'max_samples': 0.4, 'n_estimato
rs': 150}
```

Boosting Ensembling

Decision Tree

In [112]:
```
"""base_model_dtc_boost = DecisionTreeClassifier(max_depth=1, random_st

adaboost_model_dtc = AdaBoostClassifier(base_model_dtc_boost, random_st

parameters = {'n_estimators': [2,5,10,20,30,50,100,150],
              'learning_rate': [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8],
              'algorithm':["SAMME","SAMME.R"]}

grid_search_boost_dtc = GridSearchCV(adaboost_model_dtc,verbose = 2,n_j

grid_search_boost_dtc.fit(X_train_rb, y_train)"""
```

```
Fitting 5 folds for each of 128 candidates, totalling 640 fits
```

Out[112]:
```
GridSearchCV(estimator=AdaBoostClassifier(base_estimator=DecisionTreeC
lassifier(max_depth=1,

random_state=15),
                                          random_state=15),
             n_jobs=-1,
             param_grid={'algorithm': ['SAMME', 'SAMME.R'],
                         'learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5, 0.
6, 0.7,
                                           0.8],
                         'n_estimators': [2, 5, 10, 20, 30, 50, 100, 1
50]},
             scoring='f1', verbose=2)
```

In [113]:
```
"""means = grid_search_boost_dtc.cv_results_['mean_test_score']
stds = grid_search_boost_dtc.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_boost_dtc
                                    grid_search_boost_dtc.cv_results_['
```

```
BEST RESULTS: 0.75685 (+/-0.06790) for {'algorithm': 'SAMME', 'learnin
g_rate': 0.6, 'n_estimators': 150}
```

Logistic Regression

In [114]:
```
"""base_model_lg_boost = LogisticRegression(random_state=15)

adaboost_model_lg = AdaBoostClassifier(base_model_lg_boost, random_stat

parameters = {'n_estimators': [2,5,10,20,30,50,100,150],
              'learning_rate': [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8],
              'algorithm':["SAMME","SAMME.R"]}

grid_search_boost_lg = GridSearchCV(adaboost_model_lg,verbose = 2,n_job

grid_search_boost_lg.fit(X_train_rb, y_train)"""
```

```
Fitting 5 folds for each of 128 candidates, totalling 640 fits
```

Out[114]:
```
GridSearchCV(estimator=AdaBoostClassifier(base_estimator=LogisticRegre
ssion(random_state=15),
                                          random_state=15),
             n_jobs=-1,
             param_grid={'algorithm': ['SAMME', 'SAMME.R'],
                         'learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5, 0.
6, 0.7,
                                           0.8],
                         'n_estimators': [2, 5, 10, 20, 30, 50, 100, 1
50]},
             scoring='f1', verbose=2)
```

In [115]:
```
m"""eans = grid_search_boost_lg.cv_results_['mean_test_score']
stds = grid_search_boost_lg.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_boost_lg.
                                    grid_search_boost_lg.cv_results_['s
```

```
BEST RESULTS: 0.73305 (+/-0.05282) for {'algorithm': 'SAMME.R', 'learn
ing_rate': 0.2, 'n_estimators': 100}
```

Random Forest

In [116]:
```
"""rf_classifier = RandomForestClassifier(random_state = 15)

parameters = {
    'n_estimators': [30,50,80, 100, 200,250],
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10, 20, 30,40,50],
    'min_samples_split': [5,10,20,30,50,70],
    'min_samples_leaf':[4,6,8,10,12,14,16],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False]}

grid_search_rf = GridSearchCV(estimator=rf_classifier, verbose = 2,n_jo
grid_search_rf.fit(X_train_rb, y_train)"""
```

```
Fitting 2 folds for each of 18144 candidates, totalling 36288 fits
```

Out[116]:
```
GridSearchCV(cv=2, estimator=RandomForestClassifier(random_state=15),
n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'criterion': ['gini', 'entropy'],
                         'max_depth': [5, 10, 20, 30, 40, 50],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_leaf': [4, 6, 8, 10, 12, 14, 1
6],
                         'min_samples_split': [5, 10, 20, 30, 50, 70],
                         'n_estimators': [30, 50, 80, 100, 200, 250]},
             scoring='f1', verbose=2)
```

In [117]:
```
"""means = grid_search_rf.cv_results_['mean_test_score']
stds = grid_search_rf.cv_results_['std_test_score']

print("BEST RESULTS: %0.5f (+/-%0.05f) for %r" % (grid_search_rf.best_s
                                    grid_search_rf.cv_results_['std_tes
```

```
BEST RESULTS: 0.74385 (+/-0.04186) for {'bootstrap': False, 'criterio
n': 'gini', 'max_depth': 20, 'max_features': 'auto', 'min_samples_lea
f': 4, 'min_samples_split': 20, 'n_estimators': 30}
```

Stacking Ensembling

In [118]:
```
"""base_classifier_knn = KNeighborsClassifier()

base_model_dtc_boost = DecisionTreeClassifier(max_depth=1, random_state

est = [('decision_tree', DecisionTreeClassifier(criterion='gini', max_d
       ('gaussian_nb', GaussianNB(var_smoothing= 0.01519911082952933)),
       ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=10, met
       ('logistic_regression', LogisticRegression(fit_intercept=True, m
      ('bagging', BaggingClassifier(base_classifier_knn,bootstrap= Fals
      ('boosting',AdaBoostClassifier(base_model_dtc_boost,algorithm='SA
      ("RandomForest",  RandomForestClassifier(bootstrap = False, crite


meta_classifier = AdaBoostClassifier(n_estimators=50, random_state = 15

stacking_clf = StackingClassifier(estimators=est, final_estimator=meta_
stacking_clf.fit(X_train_rb, y_train)

y_pred = stacking_clf.predict(X_val_rb)
f1 = f1_score(y_val, y_pred, average='weighted')
print(f"F1 Score: {f1}")"""
```

```
F1 Score: 0.7672846495145174
```

Stacking Ensembling All Time Best

Here we decided to colect all the best classifiers from all scallers and put it together in a final Stacking Ensemble Classifier

In [119]:
```
"""base_classifier_knn = DecisionTreeClassifier(random_state=15)
base_model_dtc_boost = DecisionTreeClassifier(max_depth=1, random_state

est = [('decision_tree', DecisionTreeClassifier(criterion='gini', max_d
      ('Naive_Bayes', GaussianNB(var_smoothing=0.15199110829529336)),
      ('logistic_regression', LogisticRegression(fit_intercept=True, ma
      ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=10, metr
      ('bagging', BaggingClassifier(base_classifier_knn,bootstrap= Fals
      ('boosting',AdaBoostClassifier(base_model_dtc_boost,algorithm='SA
      ("RandomForest",  RandomForestClassifier(bootstrap = False, crite

meta_classifier = AdaBoostClassifier(n_estimators=100, random_state = 1

stacking_clf = StackingClassifier(estimators=est, final_estimator=meta_
stacking_clf.fit(X_train_rb, y_train)

y_pred = stacking_clf.predict(X_val_rb)
f1 = f1_score(y_val, y_pred, average='weighted')
print(f"F1 Score: {f1}")"""
```

```
F1 Score: 0.7474790877710722
```

Evaluating

Here we evaluated our best models to the validation data. We would like to also point out that we choose Robust Scaler as our final, because generally the models fitted with data scaled with it had better results.

Decision Tree Classifier

In [120]:
```
dtc =  DecisionTreeClassifier(criterion='gini', max_depth=None, max_fea

dtc.fit(X_train_rb, y_train)

y_pred = dtc.predict(X_val_rb)


precision = precision_score(y_val, y_pred)
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)
conf_matrix = confusion_matrix(y_val, y_pred)
recall = recall_score(y_val, y_pred)

print(f"Recall: {recall}")
print(f"Precision: {precision}")
print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Recall: 0.5405405405405406
Precision: 0.7547169811320755
Accuracy: 0.7740384615384616
F1 Score: 0.6299212598425198
Confusion Matrix:
 [[121  13]
 [ 34  40]]
```

Naive Bayes Classifier

In [121]:
```
nb = GaussianNB(var_smoothing= 0.15199110829529336)
nb.fit(X_train_rb, y_train)

y_pred = nb.predict(X_val_rb)

precision = precision_score(y_val, y_pred)
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)
conf_matrix = confusion_matrix(y_val, y_pred)
recall = recall_score(y_val, y_pred)

print(f"Recall: {recall}")
print(f"Precision: {precision}")
print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Recall: 0.6216216216216216
Precision: 0.647887323943662
Accuracy: 0.7451923076923077
F1 Score: 0.6344827586206897
Confusion Matrix:
 [[109  25]
 [ 28  46]]
```

Logistic Regression

In [122]:
```
lg = LogisticRegression(fit_intercept= True, max_iter =30 , penalty = '
lg.fit(X_train_rb, y_train)

y_pred = lg.predict(X_val_rb)

precision = precision_score(y_val, y_pred)
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)
conf_matrix = confusion_matrix(y_val, y_pred)
recall = recall_score(y_val, y_pred)

print(f"Recall: {recall}")
print(f"Precision: {precision}")
print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Recall: 0.5945945945945946
Precision: 0.6875
Accuracy: 0.7596153846153846
F1 Score: 0.6376811594202898
Confusion Matrix:
 [[114  20]
 [ 30  44]]
```

K Nearest Neighbors

In [160]:
```
knn =  KNeighborsClassifier(algorithm='auto', leaf_size=10, metric='euc
knn.fit(X_train_rb, y_train)

y_pred = knn.predict(X_val_rb)

precision = precision_score(y_val, y_pred)
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)
conf_matrix = confusion_matrix(y_val, y_pred)
recall = recall_score(y_val, y_pred)

print(f"Recall: {recall}")
print(f"Precision: {precision}")
print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Recall: 0.6081081081081081
Precision: 0.7258064516129032
Accuracy: 0.7788461538461539
F1 Score: 0.6617647058823529
Confusion Matrix:
 [[117  17]
 [ 29  45]]
```

Bagging Ensembling

In [124]:
```
bgg_knn = KNeighborsClassifier()

bgg = BaggingClassifier(bgg_knn,bootstrap= False, bootstrap_features=Tr
                        max_features= 0.6, max_samples= 0.8, n_estimato

bgg.fit(X_train_rb, y_train)

y_pred = bgg.predict(X_val_rb)

precision = precision_score(y_val, y_pred)
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)
conf_matrix = confusion_matrix(y_val, y_pred)
recall = recall_score(y_val, y_pred)

print(f"Recall: {recall}")
print(f"Precision: {precision}")
print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Recall: 0.581081081081081
Precision: 0.6935483870967742
Accuracy: 0.7596153846153846
F1 Score: 0.6323529411764707
Confusion Matrix:
 [[115  19]
 [ 31  43]]
```

Boosting Ensembling

In [125]:
```
dtc_boost = DecisionTreeClassifier(max_depth = 1, random_state=15)
boost = AdaBoostClassifier(dtc_boost,algorithm='SAMME', learning_rate=0

boost.fit(X_train_rb, y_train)

y_pred = boost.predict(X_val_rb)

precision = precision_score(y_val, y_pred)
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)
conf_matrix = confusion_matrix(y_val, y_pred)
recall = recall_score(y_val, y_pred)

print(f"Recall: {recall}")
print(f"Precision: {precision}")
print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Recall: 0.6891891891891891
Precision: 0.6986301369863014
Accuracy: 0.7836538461538461
F1 Score: 0.6938775510204082
Confusion Matrix:
 [[112  22]
 [ 23  51]]
```

Random Forest

In [126]:
```
randomf_classifier = RandomForestClassifier(bootstrap = False, criteric
                                             max_depth = 20, max_feature
                                             min_samples_split= 20,n_est

randomf_classifier.fit(X_train_rb,y_train)
y_pred = randomf_classifier.predict(X_val_rb)

precision = precision_score(y_val, y_pred)
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)
conf_matrix = confusion_matrix(y_val, y_pred)
recall = recall_score(y_val, y_pred)

print(f"Recall: {recall}")
print(f"Precision: {precision}")
print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Recall: 0.5540540540540541
Precision: 0.7735849056603774
Accuracy: 0.7836538461538461
F1 Score: 0.6456692913385828
Confusion Matrix:
 [[122  12]
 [ 33  41]]
```

Stacking Ensembling

In [127]:
```
base_classifier_dtc = DecisionTreeClassifier(random_state=15)
base_model_dtc_boost = DecisionTreeClassifier(max_depth=1, random_state

est = [('decision_tree', DecisionTreeClassifier(criterion='gini', max_d
       ('gaussian_nb', GaussianNB(var_smoothing=0.15199110829529336)),
       ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=10, met
       ('logistic_regression', LogisticRegression(fit_intercept=True, m
      ('bagging', BaggingClassifier(base_classifier_dtc,bootstrap= True
      ('boosting',AdaBoostClassifier(base_model_dtc_boost,algorithm='SA
      ('random_forest', RandomForestClassifier(random_state = 15, boots

meta_classifier = AdaBoostClassifier(n_estimators=100, random_state = 1

stacking_clf = StackingClassifier(estimators=est, final_estimator=meta_
stacking_clf.fit(X_train_r, y_train)

y_pred = stacking_clf.predict(X_val_rb)

precision = precision_score(y_val, y_pred)
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)
conf_matrix = confusion_matrix(y_val, y_pred)
recall = recall_score(y_val, y_pred)

print(f"Recall: {recall}")
print(f"Precision: {precision}")
print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

```
Recall: 1.0
Precision: 0.37373737373737376
Accuracy: 0.40384615384615385
F1 Score: 0.5441176470588236
Confusion Matrix:
 [[ 10 124]
 [  0  74]]
```

Evalutating with Cross Validation

After evaluating our results based on validation data, we decided to use KFold, to divide and truly assess our models. We choose Cross Validation because it reduces bias in the results and goes through all the data to assess it, therefore the results are more reliable.

Decision Tree Classifier

In [128]:

```
X_cv = initial_data.drop(["Admitted in School"], axis = 1)
y_cv = initial_data["Admitted in School"]
# KFold Instance
kf = KFold(shuffle = True, random_state = 15, n_splits= 7)

#Lists to store the results
score_train_precision = []
score_train_accuracy = []
score_train_f1 = []
score_train_conf_matrix = []
score_train_recall = []

score_val_precision = []
score_val_accuracy= []
score_val_f1 = []
score_val_conf_matrix= []
score_val_recall = []

# Splitting the data and getting X_train, y_train
for train_index, test_index in kf.split(X_cv):
    X_train_cv, X_val_cv = X_cv.iloc[train_index], X_cv.iloc[test_index
    y_train_cv, y_val_cv = y_cv.iloc[train_index], y_cv.iloc[test_index

#Treating the outliers
    X_train_o_cv= outlier_treatment(X_train_cv)
    X_val_o_cv = outlier_treatment(X_val_cv)

    # Removing the observations in y_train and y_val
    removed_indices_cv = list(set(X_train_cv.index) - set(X_train_o_cv.
    removed_indices_val_cv = list(set(X_val_cv.index) - set(X_val_o_cv.

    y_train_cv = y_train_cv.drop(index=removed_indices_cv)
    y_val_cv = y_val_cv.drop(index=removed_indices_val_cv)


# At this point we don't have outliers so we are going to encode
    X_train_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_val_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_train_o_cv.reset_index(inplace= True, drop = True)
    X_val_o_cv.reset_index(inplace = True, drop = True)

    # Encoding
    encoder_cv = OneHotEncoder()
    df_encoded_train_cv = pd.DataFrame()
    df_encoded_val_cv = pd.DataFrame()

    # Iterating through categorical columns and perform one-hot encodin
    for col in X_train_o_cv.select_dtypes(include='category').columns:
        encoder_cv.fit(X_train_o_cv[[col]])
        encoded_columns_cv = encoder_cv.transform(X_train_o_cv[[col]]).
        column_names_cv = [f"{col}_{(value)}" for value in encoder_cv.c
        encoded_df_cv = pd.DataFrame(encoded_columns_cv, columns=columr

        encoded_columns_val_cv = encoder_cv.transform(X_val_o_cv[[col]]
        column_names_val_cv = [f"{col}_{(value)}" for value in encoder_
        encoded_df_val_cv = pd.DataFrame(encoded_columns_val_cv, columr

        # Concatenate the encoded columns to the result DataFrame
        df_encoded_val_cv = pd.concat([df_encoded_val_cv, encoded_df_va

        # Concatenating the encoded columns to the result DataFrame
```

```
        df_encoded_train_cv = pd.concat([df_encoded_train_cv, encoded_d

    # Concatenating the encoded dataframe to the train
    X_train = pd.concat([X_train_o_cv, df_encoded_train_cv], axis=1)

    # Concatenating the encoded dataframe to the train
    X_val = pd.concat([X_val_o_cv, df_encoded_val_cv], axis=1)

    X_train.drop(["Program" , "Student Gender", "School of Origin", "Fa
    X_val.drop(["Program" , "Student Gender", "School of Origin", "Favo

# At this point we completed the encoding

# Imputting the Missing values
    imputer_cv = KNNImputer(n_neighbors=7)

    # Fit and transform on X_train
    X_train = pd.DataFrame(imputer_cv.fit_transform(X_train.select_dtyp
                                            columns=X_train.select_dtyp
                                            index=X_train.select_dtypes

    # Transform on X_val
    X_val = pd.DataFrame(imputer_cv.transform(X_val.select_dtypes(inclu
                                          columns=X_val.select_dtypes(i
                                          index=X_val.select_dtypes(inc



    # Extracting the Student IDs
    student_ids_cv = X_train['Student ID']

    # Dropping the student ID column before scaling
    df_robust_cv = X_train.drop(columns=['Student ID'])

# We finished imputation so we can start scaling

    #Scaling
    rb_cv = RobustScaler().fit(df_robust_cv)
    scaled_data_rb_cv = rb_cv.transform(df_robust_cv)


    # New DataFrame with the scaled data
    X_train_cv = pd.DataFrame(scaled_data_rb_cv, columns=df_robust_cv.c

    # Add the Student IDs back
    X_train_cv['Student ID'] = student_ids_cv

    # Extracting the Student IDs
    student_ids_val_cv = X_val['Student ID']

    # Dropping the student ID column before scaling
    df_robust_val_cv = X_val.drop(columns=['Student ID'])

    #Scaling
    scaled_data_rb_val_cv = rb_cv.transform(df_robust_val_cv)


    # New DataFrame with the scaled data
    X_val_cv = pd.DataFrame(scaled_data_rb_val_cv, columns=df_robust_va

    # Add the Student IDs back
```

```
    X_val_cv['Student ID'] = student_ids_val_cv

#Feature Selection
    X_train_cv = X_train_cv.drop(columns = ["Favourite Study Element_Ea
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

    X_val_cv = X_val_cv.drop(columns = ["Favourite Study Element_Earth"
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

# Here we have our dataset ready to evaluation
# Model
    dtc_cv =  DecisionTreeClassifier(criterion='gini', max_depth=None,
    dtc_cv.fit(X_train_cv, y_train_cv)

#Predicting
    y_pred_train = dtc_cv.predict(X_train_cv)
    y_pred_val =dtc_cv.predict(X_val_cv)

#Assessing
    precision_train = precision_score(y_train_cv, y_pred_train)
    precision_val = precision_score(y_val_cv, y_pred_val)

    accuracy_train = accuracy_score(y_train_cv, y_pred_train)
    accuracy_val = accuracy_score(y_val_cv,y_pred_val)

    f1_train = f1_score(y_train_cv, y_pred_train)
    f1_val = f1_score(y_val_cv,y_pred_val)

    conf_matrix_train = confusion_matrix(y_train_cv, y_pred_train)
    conf_matrix_val = confusion_matrix(y_val_cv, y_pred_val)

    recall_train = recall_score(y_train_cv, y_pred_train)
    recall_val = recall_score(y_val_cv, y_pred_val)

    score_train_precision.append(precision_train)
    score_val_precision.append(precision_val)

    score_train_accuracy.append(accuracy_train)
    score_val_accuracy.append(accuracy_val)

    score_train_f1.append(f1_train)
    score_val_f1.append(f1_val)

    score_train_recall.append(recall_train)
    score_val_recall.append(recall_val)

print(f"Recall Train: {np.mean(score_train_recall)}")
print(f"Recall Validation: {np.mean(score_val_recall)}")

print(f"Precision Train: {np.mean(score_train_precision)}")
print(f"Precision Validation: {np.mean(score_val_precision)}")

print(f"Accuracy Train: {np.mean(score_train_accuracy)}")
print(f"Accuracy Validation: {np.mean(score_val_accuracy)}")

print(f"F1 Score Train: {np.mean(score_train_f1)}")
print(f"F1 Score Validation: {np.mean(score_val_f1)}")
```

```
Recall Train: 0.6988190524299176
Recall Validation: 0.6384521362933772
Precision Train: 0.8410425858554439
Precision Validation: 0.7959548473373358
Accuracy Train: 0.8465974713476124
Accuracy Validation: 0.8098937792938958
F1 Score Train: 0.7619859228442959
F1 Score Validation: 0.7028097926613628
```

Naive Bayes Classifier

In [129]:

```
X_cv = initial_data.drop(["Admitted in School"], axis = 1)
y_cv = initial_data["Admitted in School"]
# KFold Instance
kf = KFold(shuffle = True, random_state = 15, n_splits= 7)

#Lists to store the results
score_train_precision = []
score_train_accuracy = []
score_train_f1 = []
score_train_conf_matrix = []
score_train_recall = []

score_val_precision = []
score_val_accuracy= []
score_val_f1 = []
score_val_conf_matrix= []
score_val_recall = []

# Splitting the data and getting X_train, y_train
for train_index, test_index in kf.split(X_cv):
    X_train_cv, X_val_cv = X_cv.iloc[train_index], X_cv.iloc[test_index
    y_train_cv, y_val_cv = y_cv.iloc[train_index], y_cv.iloc[test_index

#Treating the outliers
    X_train_o_cv= outlier_treatment(X_train_cv)
    X_val_o_cv = outlier_treatment(X_val_cv)

    # Removing the observations in y_train and y_val
    removed_indices_cv = list(set(X_train_cv.index) - set(X_train_o_cv.
    removed_indices_val_cv = list(set(X_val_cv.index) - set(X_val_o_cv.

    y_train_cv = y_train_cv.drop(index=removed_indices_cv)
    y_val_cv = y_val_cv.drop(index=removed_indices_val_cv)


# At this point we don't have outliers so we are going to encode
    X_train_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_val_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_train_o_cv.reset_index(inplace= True, drop = True)
    X_val_o_cv.reset_index(inplace = True, drop = True)

    # Encoding
    encoder_cv = OneHotEncoder()
    df_encoded_train_cv = pd.DataFrame()
    df_encoded_val_cv = pd.DataFrame()

    # Iterating through categorical columns and perform one-hot encodin
    for col in X_train_o_cv.select_dtypes(include='category').columns:
        encoder_cv.fit(X_train_o_cv[[col]])
        encoded_columns_cv = encoder_cv.transform(X_train_o_cv[[col]]).
        column_names_cv = [f"{col}_{(value)}" for value in encoder_cv.c
        encoded_df_cv = pd.DataFrame(encoded_columns_cv, columns=columr

        encoded_columns_val_cv = encoder_cv.transform(X_val_o_cv[[col]]
        column_names_val_cv = [f"{col}_{(value)}" for value in encoder_
        encoded_df_val_cv = pd.DataFrame(encoded_columns_val_cv, columr

        # Concatenate the encoded columns to the result DataFrame
        df_encoded_val_cv = pd.concat([df_encoded_val_cv, encoded_df_va

        # Concatenating the encoded columns to the result DataFrame
```

```
        df_encoded_train_cv = pd.concat([df_encoded_train_cv, encoded_d

    # Concatenating the encoded dataframe to the train
    X_train = pd.concat([X_train_o_cv, df_encoded_train_cv], axis=1)

    # Concatenating the encoded dataframe to the train
    X_val = pd.concat([X_val_o_cv, df_encoded_val_cv], axis=1)

    X_train.drop(["Program" , "Student Gender", "School of Origin", "Fa
    X_val.drop(["Program" , "Student Gender", "School of Origin", "Favo

# At this point we completed the encoding

# Imputting the Missing values
    imputer_cv = KNNImputer(n_neighbors=7)

    # Fit and transform on X_train
    X_train = pd.DataFrame(imputer_cv.fit_transform(X_train.select_dtyp
                                            columns=X_train.select_dtyp
                                            index=X_train.select_dtypes

    # Transform on X_val
    X_val = pd.DataFrame(imputer_cv.transform(X_val.select_dtypes(inclu
                                          columns=X_val.select_dtypes(i
                                          index=X_val.select_dtypes(inc


# We can start scaling

    # Extracting the Student IDs
    student_ids_cv = X_train['Student ID']

    # Dropping the student ID column before scaling
    df_robust_cv = X_train.drop(columns=['Student ID'])

    #Scaling
    rb_cv = RobustScaler().fit(df_robust_cv)
    scaled_data_rb_cv = rb_cv.transform(df_robust_cv)


    # New DataFrame with the scaled data
    X_train_cv = pd.DataFrame(scaled_data_rb_cv, columns=df_robust_cv.c

    # Add the Student IDs back
    X_train_cv['Student ID'] = student_ids_cv

    # Extracting the Student IDs
    student_ids_val_cv = X_val['Student ID']

    # Dropping the student ID column before scaling
    df_robust_val_cv = X_val.drop(columns=['Student ID'])

    #Scaling
    scaled_data_rb_val_cv = rb_cv.transform(df_robust_val_cv)


    # New DataFrame with the scaled data
    X_val_cv = pd.DataFrame(scaled_data_rb_val_cv, columns=df_robust_va

    # Add the Student IDs back
    X_val_cv['Student ID'] = student_ids_val_cv
```

```
# Feature Selection
    X_train_cv = X_train_cv.drop(columns = ["Favourite Study Element_Ea
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

    X_val_cv = X_val_cv.drop(columns = ["Favourite Study Element_Earth"
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

# Here we have our dataset ready to evaluation
# Modeling
    nb_cv = GaussianNB(var_smoothing= 0.15199110829529336)
    nb_cv.fit(X_train_cv, y_train_cv)

#Predicting
    y_pred_train = nb_cv.predict(X_train_cv)
    y_pred_val =nb_cv.predict(X_val_cv)

# Assessing
    precision_train = precision_score(y_train_cv, y_pred_train)
    precision_val = precision_score(y_val_cv, y_pred_val)

    accuracy_train = accuracy_score(y_train_cv, y_pred_train)
    accuracy_val = accuracy_score(y_val_cv,y_pred_val)

    f1_train = f1_score(y_train_cv, y_pred_train)
    f1_val = f1_score(y_val_cv,y_pred_val)

    conf_matrix_train = confusion_matrix(y_train_cv, y_pred_train)
    conf_matrix_val = confusion_matrix(y_val_cv, y_pred_val)

    recall_train = recall_score(y_train_cv, y_pred_train)
    recall_val = recall_score(y_val_cv, y_pred_val)

    score_train_precision.append(precision_train)
    score_val_precision.append(precision_val)

    score_train_accuracy.append(accuracy_train)
    score_val_accuracy.append(accuracy_val)

    score_train_f1.append(f1_train)
    score_val_f1.append(f1_val)

    score_train_recall.append(recall_train)
    score_val_recall.append(recall_val)

print(f"Recall Train: {np.mean(score_train_recall)}")
print(f"Recall Validation: {np.mean(score_val_recall)}")

print(f"Precision Train: {np.mean(score_train_precision)}")
print(f"Precision Validation: {np.mean(score_val_precision)}")

print(f"Accuracy Train: {np.mean(score_train_accuracy)}")
print(f"Accuracy Validation: {np.mean(score_val_accuracy)}")

print(f"F1 Score Train: {np.mean(score_train_f1)}")
print(f"F1 Score Validation: {np.mean(score_val_f1)}")
```

```
Recall Train: 0.6190495657028308
Recall Validation: 0.6241460049244978
Precision Train: 0.6974048407724253
Precision Validation: 0.7034611765704203
Accuracy Train: 0.7710746820862784
Accuracy Validation: 0.771449371827939
F1 Score Train: 0.6554821639197669
F1 Score Validation: 0.6576125992817536
```

Logistic Regression

In [130]:

```
X_cv = initial_data.drop(["Admitted in School"], axis = 1)
y_cv = initial_data["Admitted in School"]
# KFold Instance
kf = KFold(shuffle = True, random_state = 15, n_splits= 7)

#Lists to store the results
score_train_precision = []
score_train_accuracy = []
score_train_f1 = []
score_train_conf_matrix = []
score_train_recall = []

score_val_precision = []
score_val_accuracy= []
score_val_f1 = []
score_val_conf_matrix= []
score_val_recall = []

# Splitting the data and getting X_train, y_train
for train_index, test_index in kf.split(X_cv):
    X_train_cv, X_val_cv = X_cv.iloc[train_index], X_cv.iloc[test_index
    y_train_cv, y_val_cv = y_cv.iloc[train_index], y_cv.iloc[test_index

#Treating the outliers
    X_train_o_cv= outlier_treatment(X_train_cv)
    X_val_o_cv = outlier_treatment(X_val_cv)

    # Removing the observations in y_train and y_val
    removed_indices_cv = list(set(X_train_cv.index) - set(X_train_o_cv.
    removed_indices_val_cv = list(set(X_val_cv.index) - set(X_val_o_cv.

    y_train_cv = y_train_cv.drop(index=removed_indices_cv)
    y_val_cv = y_val_cv.drop(index=removed_indices_val_cv)


# At this point we don't have outliers so we are going to encode
    X_train_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_val_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_train_o_cv.reset_index(inplace= True, drop = True)
    X_val_o_cv.reset_index(inplace = True, drop = True)

    # Encoding
    encoder_cv = OneHotEncoder()
    df_encoded_train_cv = pd.DataFrame()
    df_encoded_val_cv = pd.DataFrame()

    # Iterating through categorical columns and perform one-hot encodin
    for col in X_train_o_cv.select_dtypes(include='category').columns:
        encoder_cv.fit(X_train_o_cv[[col]])
        encoded_columns_cv = encoder_cv.transform(X_train_o_cv[[col]]).
        column_names_cv = [f"{col}_{(value)}" for value in encoder_cv.c
        encoded_df_cv = pd.DataFrame(encoded_columns_cv, columns=columr

        encoded_columns_val_cv = encoder_cv.transform(X_val_o_cv[[col]]
        column_names_val_cv = [f"{col}_{(value)}" for value in encoder_
        encoded_df_val_cv = pd.DataFrame(encoded_columns_val_cv, columr

        # Concatenate the encoded columns to the result DataFrame
        df_encoded_val_cv = pd.concat([df_encoded_val_cv, encoded_df_va

        # Concatenating the encoded columns to the result DataFrame
```

```
        df_encoded_train_cv = pd.concat([df_encoded_train_cv, encoded_d

    # Concatenating the encoded dataframe to the train
    X_train = pd.concat([X_train_o_cv, df_encoded_train_cv], axis=1)

    # Concatenating the encoded dataframe to the train
    X_val = pd.concat([X_val_o_cv, df_encoded_val_cv], axis=1)

    X_train.drop(["Program" , "Student Gender", "School of Origin", "Fa
    X_val.drop(["Program" , "Student Gender", "School of Origin", "Favo

# At this point we completed the encoding

# Imputting the Missing values
    imputer_cv = KNNImputer(n_neighbors=7)

    # Fit and transform on X_train
    X_train = pd.DataFrame(imputer_cv.fit_transform(X_train.select_dtyp
                                            columns=X_train.select_dtyp
                                            index=X_train.select_dtypes

    # Transform on X_val
    X_val = pd.DataFrame(imputer_cv.transform(X_val.select_dtypes(inclu
                                          columns=X_val.select_dtypes(i
                                          index=X_val.select_dtypes(inc


# We can start Scaling
    # Extracting the Student IDs
    student_ids_cv = X_train['Student ID']

    # Dropping the student ID column before scaling
    df_robust_cv = X_train.drop(columns=['Student ID'])

    #Scaling
    rb_cv = RobustScaler().fit(df_robust_cv)
    scaled_data_rb_cv = rb_cv.transform(df_robust_cv)


    # New DataFrame with the scaled data
    X_train_cv = pd.DataFrame(scaled_data_rb_cv, columns=df_robust_cv.c

    # Add the Student IDs back
    X_train_cv['Student ID'] = student_ids_cv

    # Extracting the Student IDs
    student_ids_val_cv = X_val['Student ID']

    # Dropping the student ID column before scaling
    df_robust_val_cv = X_val.drop(columns=['Student ID'])

    #Scaling
    scaled_data_rb_val_cv = rb_cv.transform(df_robust_val_cv)


    # New DataFrame with the scaled data
    X_val_cv = pd.DataFrame(scaled_data_rb_val_cv, columns=df_robust_va

    # Add the Student IDs back
    X_val_cv['Student ID'] = student_ids_val_cv
```

```
#Feature Selection
    X_train_cv = X_train_cv.drop(columns = ["Favourite Study Element_Ea
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

    X_val_cv = X_val_cv.drop(columns = ["Favourite Study Element_Earth"
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

# Here we have our dataset ready to evaluation
# Modeling
    lg_cv = LogisticRegression(fit_intercept= True, max_iter =30 , pena
    lg_cv.fit(X_train_cv, y_train_cv)

#Predicting
    y_pred_train = lg_cv.predict(X_train_cv)
    y_pred_val =lg_cv.predict(X_val_cv)

#Assessing
    precision_train = precision_score(y_train_cv, y_pred_train)
    precision_val = precision_score(y_val_cv, y_pred_val)

    accuracy_train = accuracy_score(y_train_cv, y_pred_train)
    accuracy_val = accuracy_score(y_val_cv,y_pred_val)

    f1_train = f1_score(y_train_cv, y_pred_train)
    f1_val = f1_score(y_val_cv,y_pred_val)

    conf_matrix_train = confusion_matrix(y_train_cv, y_pred_train)
    conf_matrix_val = confusion_matrix(y_val_cv, y_pred_val)

    recall_train = recall_score(y_train_cv, y_pred_train)
    recall_val = recall_score(y_val_cv, y_pred_val)

    score_train_precision.append(precision_train)
    score_val_precision.append(precision_val)

    score_train_accuracy.append(accuracy_train)
    score_val_accuracy.append(accuracy_val)

    score_train_f1.append(f1_train)
    score_val_f1.append(f1_val)

    score_train_recall.append(recall_train)
    score_val_recall.append(recall_val)

print(f"Recall Train: {np.mean(score_train_recall)}")
print(f"Recall Validation: {np.mean(score_val_recall)}")

print(f"Precision Train: {np.mean(score_train_precision)}")
print(f"Precision Validation: {np.mean(score_val_precision)}")

print(f"Accuracy Train: {np.mean(score_train_accuracy)}")
print(f"Accuracy Validation: {np.mean(score_val_accuracy)}")

print(f"F1 Score Train: {np.mean(score_train_f1)}")
print(f"F1 Score Validation: {np.mean(score_val_f1)}")
```

```
Recall Train: 0.6945510842352496
Recall Validation: 0.6958295446210493
Precision Train: 0.7503561725038839
Precision Validation: 0.752807794672121
Accuracy Train: 0.8108525768729459
Accuracy Validation: 0.8085065313254015
F1 Score Train: 0.7211973161352974
F1 Score Validation: 0.719241252837057
```

K Nearest Neighbors

In [161]:

```
X_cv = initial_data.drop(["Admitted in School"], axis = 1)
y_cv = initial_data["Admitted in School"]
# KFold Instance
kf = KFold(shuffle = True, random_state = 15, n_splits= 7)

#Lists to store the results
score_train_precision = []
score_train_accuracy = []
score_train_f1 = []
score_train_conf_matrix = []
score_train_recall = []

score_val_precision = []
score_val_accuracy= []
score_val_f1 = []
score_val_conf_matrix= []
score_val_recall = []

# Splitting the data and getting X_train, y_train
for train_index, test_index in kf.split(X_cv):
    X_train_cv, X_val_cv = X_cv.iloc[train_index], X_cv.iloc[test_index
    y_train_cv, y_val_cv = y_cv.iloc[train_index], y_cv.iloc[test_index

#Treating the outliers
    X_train_o_cv= outlier_treatment(X_train_cv)
    X_val_o_cv = outlier_treatment(X_val_cv)

    # Removing the observations in y_train and y_val
    removed_indices_cv = list(set(X_train_cv.index) - set(X_train_o_cv.
    removed_indices_val_cv = list(set(X_val_cv.index) - set(X_val_o_cv.

    y_train_cv = y_train_cv.drop(index=removed_indices_cv)
    y_val_cv = y_val_cv.drop(index=removed_indices_val_cv)


# At this point we don't have outliers so we are going to encode
    X_train_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_val_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_train_o_cv.reset_index(inplace= True, drop = True)
    X_val_o_cv.reset_index(inplace = True, drop = True)

    # Encoding
    encoder_cv = OneHotEncoder()
    df_encoded_train_cv = pd.DataFrame()
    df_encoded_val_cv = pd.DataFrame()

    # Iterating through categorical columns and perform one-hot encodin
    for col in X_train_o_cv.select_dtypes(include='category').columns:
        encoder_cv.fit(X_train_o_cv[[col]])
        encoded_columns_cv = encoder_cv.transform(X_train_o_cv[[col]]).
        column_names_cv = [f"{col}_{(value)}" for value in encoder_cv.c
        encoded_df_cv = pd.DataFrame(encoded_columns_cv, columns=columr

        encoded_columns_val_cv = encoder_cv.transform(X_val_o_cv[[col]]
        column_names_val_cv = [f"{col}_{(value)}" for value in encoder_
        encoded_df_val_cv = pd.DataFrame(encoded_columns_val_cv, columr

        # Concatenate the encoded columns to the result DataFrame
        df_encoded_val_cv = pd.concat([df_encoded_val_cv, encoded_df_va

        # Concatenating the encoded columns to the result DataFrame
```

```
        df_encoded_train_cv = pd.concat([df_encoded_train_cv, encoded_d

    # Concatenating the encoded dataframe to the train
    X_train = pd.concat([X_train_o_cv, df_encoded_train_cv], axis=1)

    # Concatenating the encoded dataframe to the train
    X_val = pd.concat([X_val_o_cv, df_encoded_val_cv], axis=1)

    X_train.drop(["Program" , "Student Gender", "School of Origin", "Fa
    X_val.drop(["Program" , "Student Gender", "School of Origin", "Favo

# At this point we completed the encoding

# Imputting the Missing values
    imputer_cv = KNNImputer(n_neighbors=7)

    # Fit and transform on X_train
    X_train = pd.DataFrame(imputer_cv.fit_transform(X_train.select_dtyp
                                            columns=X_train.select_dtyp
                                            index=X_train.select_dtypes

    # Transform on X_val
    X_val = pd.DataFrame(imputer_cv.transform(X_val.select_dtypes(inclu
                                          columns=X_val.select_dtypes(i
                                          index=X_val.select_dtypes(inc


# We can start scaling
    # Extracting the Student IDs
    student_ids_cv = X_train['Student ID']

    # Dropping the student ID column before scaling
    df_robust_cv = X_train.drop(columns=['Student ID'])

    #Scaling
    rb_cv = RobustScaler().fit(df_robust_cv)
    scaled_data_rb_cv = rb_cv.transform(df_robust_cv)


    # New DataFrame with the scaled data
    X_train_cv = pd.DataFrame(scaled_data_rb_cv, columns=df_robust_cv.c

    # Add the Student IDs back
    X_train_cv['Student ID'] = student_ids_cv

    # Extracting the Student IDs
    student_ids_val_cv = X_val['Student ID']

    # Dropping the student ID column before scaling
    df_robust_val_cv = X_val.drop(columns=['Student ID'])

    #Scaling
    scaled_data_rb_val_cv = rb_cv.transform(df_robust_val_cv)


    # New DataFrame with the scaled data
    X_val_cv = pd.DataFrame(scaled_data_rb_val_cv, columns=df_robust_va

    # Add the Student IDs back
    X_val_cv['Student ID'] = student_ids_val_cv
```

```
#Feature Selection
    X_train_cv = X_train_cv.drop(columns = ["Favourite Study Element_Ea
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

    X_val_cv = X_val_cv.drop(columns = ["Favourite Study Element_Earth"
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

# Here we have our dataset ready to evaluation
# Modeling
    knn_cv =  KNeighborsClassifier(algorithm='auto', leaf_size=10, metr
    knn_cv.fit(X_train_cv, y_train_cv)



#Predicting
    y_pred_train = knn_cv.predict(X_train_cv)
    y_pred_val = knn_cv.predict(X_val_cv)

#Assessing
    precision_train = precision_score(y_train_cv, y_pred_train)
    precision_val = precision_score(y_val_cv, y_pred_val)

    accuracy_train = accuracy_score(y_train_cv, y_pred_train)
    accuracy_val = accuracy_score(y_val_cv,y_pred_val)

    f1_train = f1_score(y_train_cv, y_pred_train)
    f1_val = f1_score(y_val_cv,y_pred_val)

    conf_matrix_train = confusion_matrix(y_train_cv, y_pred_train)
    conf_matrix_val = confusion_matrix(y_val_cv, y_pred_val)

    recall_train = recall_score(y_train_cv, y_pred_train)
    recall_val = recall_score(y_val_cv, y_pred_val)

    score_train_precision.append(precision_train)
    score_val_precision.append(precision_val)

    score_train_accuracy.append(accuracy_train)
    score_val_accuracy.append(accuracy_val)

    score_train_f1.append(f1_train)
    score_val_f1.append(f1_val)

    score_train_recall.append(recall_train)
    score_val_recall.append(recall_val)

print(f"Recall Train: {np.mean(score_train_recall)}")
print(f"Recall Validation: {np.mean(score_val_recall)}")

print(f"Precision Train: {np.mean(score_train_precision)}")
print(f"Precision Validation: {np.mean(score_val_precision)}")

print(f"Accuracy Train: {np.mean(score_train_accuracy)}")
print(f"Accuracy Validation: {np.mean(score_val_accuracy)}")

print(f"F1 Score Train: {np.mean(score_train_f1)}")
print(f"F1 Score Validation: {np.mean(score_val_f1)}")
```

```
Recall Train: 1.0
Recall Validation: 0.6477996274137536
Precision Train: 1.0
Precision Validation: 0.776288116189594
Accuracy Train: 1.0
Accuracy Validation: 0.8057201098261088
F1 Score Train: 1.0
F1 Score Validation: 0.7008754065157097
```

Bagging Ensembling

In [132]:

```
X_cv = initial_data.drop(["Admitted in School"], axis = 1)
y_cv = initial_data["Admitted in School"]
# KFold Instance
kf = KFold(shuffle = True, random_state = 15, n_splits= 7)

#Lists to store the results
score_train_precision = []
score_train_accuracy = []
score_train_f1 = []
score_train_conf_matrix = []
score_train_recall = []

score_val_precision = []
score_val_accuracy= []
score_val_f1 = []
score_val_conf_matrix= []
score_val_recall = []

# Splitting the data and getting X_train, y_train
for train_index, test_index in kf.split(X_cv):
    X_train_cv, X_val_cv = X_cv.iloc[train_index], X_cv.iloc[test_index
    y_train_cv, y_val_cv = y_cv.iloc[train_index], y_cv.iloc[test_index

#Treating the outliers
    X_train_o_cv= outlier_treatment(X_train_cv)
    X_val_o_cv = outlier_treatment(X_val_cv)

    # Removing the observations in y_train and y_val
    removed_indices_cv = list(set(X_train_cv.index) - set(X_train_o_cv.
    removed_indices_val_cv = list(set(X_val_cv.index) - set(X_val_o_cv.

    y_train_cv = y_train_cv.drop(index=removed_indices_cv)
    y_val_cv = y_val_cv.drop(index=removed_indices_val_cv)


# At this point we don't have outliers so we are going to encode
    X_train_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_val_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_train_o_cv.reset_index(inplace= True, drop = True)
    X_val_o_cv.reset_index(inplace = True, drop = True)

    # Encoding
    encoder_cv = OneHotEncoder()
    df_encoded_train_cv = pd.DataFrame()
    df_encoded_val_cv = pd.DataFrame()

    # Iterating through categorical columns and perform one-hot encodin
    for col in X_train_o_cv.select_dtypes(include='category').columns:
        encoder_cv.fit(X_train_o_cv[[col]])
        encoded_columns_cv = encoder_cv.transform(X_train_o_cv[[col]]).
        column_names_cv = [f"{col}_{(value)}" for value in encoder_cv.c
        encoded_df_cv = pd.DataFrame(encoded_columns_cv, columns=columr

        encoded_columns_val_cv = encoder_cv.transform(X_val_o_cv[[col]]
        column_names_val_cv = [f"{col}_{(value)}" for value in encoder_
        encoded_df_val_cv = pd.DataFrame(encoded_columns_val_cv, columr

        # Concatenate the encoded columns to the result DataFrame
        df_encoded_val_cv = pd.concat([df_encoded_val_cv, encoded_df_va

        # Concatenating the encoded columns to the result DataFrame
```

```
        df_encoded_train_cv = pd.concat([df_encoded_train_cv, encoded_d

    # Concatenating the encoded dataframe to the train
    X_train = pd.concat([X_train_o_cv, df_encoded_train_cv], axis=1)

    # Concatenating the encoded dataframe to the train
    X_val = pd.concat([X_val_o_cv, df_encoded_val_cv], axis=1)

    X_train.drop(["Program" , "Student Gender", "School of Origin", "Fa
    X_val.drop(["Program" , "Student Gender", "School of Origin", "Favo

# At this point we completed the encoding

# Imputting the Missing values
    imputer_cv = KNNImputer(n_neighbors=7)

    # Fit and transform on X_train
    X_train = pd.DataFrame(imputer_cv.fit_transform(X_train.select_dtyp
                                            columns=X_train.select_dtyp
                                            index=X_train.select_dtypes

    # Transform on X_val
    X_val = pd.DataFrame(imputer_cv.transform(X_val.select_dtypes(inclu
                                          columns=X_val.select_dtypes(i
                                          index=X_val.select_dtypes(inc


# We can Scale

    # Extracting the Student IDs
    student_ids_cv = X_train['Student ID']

    # Dropping the student ID column before scaling
    df_robust_cv = X_train.drop(columns=['Student ID'])

    #Scaling
    rb_cv = RobustScaler().fit(df_robust_cv)
    scaled_data_rb_cv = rb_cv.transform(df_robust_cv)


    # New DataFrame with the scaled data
    X_train_cv = pd.DataFrame(scaled_data_rb_cv, columns=df_robust_cv.c

    # Add the Student IDs back
    X_train_cv['Student ID'] = student_ids_cv

    # Extracting the Student IDs
    student_ids_val_cv = X_val['Student ID']

    # Dropping the student ID column before scaling
    df_robust_val_cv = X_val.drop(columns=['Student ID'])

    #Scaling
    scaled_data_rb_val_cv = rb_cv.transform(df_robust_val_cv)


    # New DataFrame with the scaled data
    X_val_cv = pd.DataFrame(scaled_data_rb_val_cv, columns=df_robust_va

    # Add the Student IDs back
    X_val_cv['Student ID'] = student_ids_val_cv
```

```
#Feature Selection
    X_train_cv = X_train_cv.drop(columns = ["Favourite Study Element_Ea
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

    X_val_cv = X_val_cv.drop(columns = ["Favourite Study Element_Earth"
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

# Here we have our dataset ready to evaluation
# Modeling
    bgg_knn_cv = KNeighborsClassifier()

    bgg_cv = BaggingClassifier(bgg_knn_cv,bootstrap= False, bootstrap_f
                          max_features= 0.6, max_samples= 0.8, n_estimato
    bgg_cv.fit(X_train_cv, y_train_cv)

#Predicting
    y_pred_train = bgg_cv.predict(X_train_cv)
    y_pred_val = bgg_cv.predict(X_val_cv)

#Assessing
    precision_train = precision_score(y_train_cv, y_pred_train)
    precision_val = precision_score(y_val_cv, y_pred_val)

    accuracy_train = accuracy_score(y_train_cv, y_pred_train)
    accuracy_val = accuracy_score(y_val_cv,y_pred_val)

    f1_train = f1_score(y_train_cv, y_pred_train)
    f1_val = f1_score(y_val_cv,y_pred_val)

    conf_matrix_train = confusion_matrix(y_train_cv, y_pred_train)
    conf_matrix_val = confusion_matrix(y_val_cv, y_pred_val)

    recall_train = recall_score(y_train_cv, y_pred_train)
    recall_val = recall_score(y_val_cv, y_pred_val)

    score_train_precision.append(precision_train)
    score_val_precision.append(precision_val)

    score_train_accuracy.append(accuracy_train)
    score_val_accuracy.append(accuracy_val)

    score_train_f1.append(f1_train)
    score_val_f1.append(f1_val)

    score_train_recall.append(recall_train)
    score_val_recall.append(recall_val)

print(f"Recall Train: {np.mean(score_train_recall)}")
print(f"Recall Validation: {np.mean(score_val_recall)}")

print(f"Precision Train: {np.mean(score_train_precision)}")
print(f"Precision Validation: {np.mean(score_val_precision)}")

print(f"Accuracy Train: {np.mean(score_train_accuracy)}")
print(f"Accuracy Validation: {np.mean(score_val_accuracy)}")
```

```
print(f"F1 Score Train: {np.mean(score_train_f1)}")
print(f"F1 Score Validation: {np.mean(score_val_f1)}")
```

```
Recall Train: 0.7114697410273358
Recall Validation: 0.6715402590143335
Precision Train: 0.8676422616370824
Precision Validation: 0.7952022772163732
Accuracy Train: 0.8598531039602564
Accuracy Validation: 0.8198513464791857
F1 Score Train: 0.7815033194359453
F1 Score Validation: 0.7235436018981847
```

Boosting Ensembling

In [133]:

```
X_cv = initial_data.drop(["Admitted in School"], axis = 1)
y_cv = initial_data["Admitted in School"]
# KFold Instance
kf = KFold(shuffle = True, random_state = 15, n_splits= 7)

#Lists to store the results
score_train_precision = []
score_train_accuracy = []
score_train_f1 = []
score_train_conf_matrix = []
score_train_recall = []

score_val_precision = []
score_val_accuracy= []
score_val_f1 = []
score_val_conf_matrix= []
score_val_recall = []

# Splitting the data and getting X_train, y_train
for train_index, test_index in kf.split(X_cv):
    X_train_cv, X_val_cv = X_cv.iloc[train_index], X_cv.iloc[test_index
    y_train_cv, y_val_cv = y_cv.iloc[train_index], y_cv.iloc[test_index

#Treating the outliers
    X_train_o_cv= outlier_treatment(X_train_cv)
    X_val_o_cv = outlier_treatment(X_val_cv)

    # Removing the observations in y_train and y_val
    removed_indices_cv = list(set(X_train_cv.index) - set(X_train_o_cv.
    removed_indices_val_cv = list(set(X_val_cv.index) - set(X_val_o_cv.

    y_train_cv = y_train_cv.drop(index=removed_indices_cv)
    y_val_cv = y_val_cv.drop(index=removed_indices_val_cv)


# At this point we don't have outliers so we are going to encode
    X_train_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_val_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_train_o_cv.reset_index(inplace= True, drop = True)
    X_val_o_cv.reset_index(inplace = True, drop = True)

    # Encoding
    encoder_cv = OneHotEncoder()
    df_encoded_train_cv = pd.DataFrame()
    df_encoded_val_cv = pd.DataFrame()

    # Iterating through categorical columns and perform one-hot encodin
    for col in X_train_o_cv.select_dtypes(include='category').columns:
        encoder_cv.fit(X_train_o_cv[[col]])
        encoded_columns_cv = encoder_cv.transform(X_train_o_cv[[col]]).
        column_names_cv = [f"{col}_{(value)}" for value in encoder_cv.c
        encoded_df_cv = pd.DataFrame(encoded_columns_cv, columns=columr

        encoded_columns_val_cv = encoder_cv.transform(X_val_o_cv[[col]]
        column_names_val_cv = [f"{col}_{(value)}" for value in encoder_
        encoded_df_val_cv = pd.DataFrame(encoded_columns_val_cv, columr

        # Concatenate the encoded columns to the result DataFrame
        df_encoded_val_cv = pd.concat([df_encoded_val_cv, encoded_df_va

        # Concatenating the encoded columns to the result DataFrame
```

```
        df_encoded_train_cv = pd.concat([df_encoded_train_cv, encoded_d

    # Concatenating the encoded dataframe to the train
    X_train = pd.concat([X_train_o_cv, df_encoded_train_cv], axis=1)

    # Concatenating the encoded dataframe to the train
    X_val = pd.concat([X_val_o_cv, df_encoded_val_cv], axis=1)

    X_train.drop(["Program" , "Student Gender", "School of Origin", "Fa
    X_val.drop(["Program" , "Student Gender", "School of Origin", "Favo

# At this point we completed the encoding

# Imputting the Missing values
    imputer_cv = KNNImputer(n_neighbors=7)

    # Fit and transform on X_train
    X_train = pd.DataFrame(imputer_cv.fit_transform(X_train.select_dtyp
                                            columns=X_train.select_dtyp
                                            index=X_train.select_dtypes

    # Transform on X_val
    X_val = pd.DataFrame(imputer_cv.transform(X_val.select_dtypes(inclu
                                          columns=X_val.select_dtypes(i
                                          index=X_val.select_dtypes(inc


# Start Scaling
    # Extracting the Student IDs
    student_ids_cv = X_train['Student ID']

    # Dropping the student ID column before scaling
    df_robust_cv = X_train.drop(columns=['Student ID'])

    #Scaling
    rb_cv = RobustScaler().fit(df_robust_cv)
    scaled_data_rb_cv = rb_cv.transform(df_robust_cv)


    # New DataFrame with the scaled data
    X_train_cv = pd.DataFrame(scaled_data_rb_cv, columns=df_robust_cv.c

    # Add the Student IDs back
    X_train_cv['Student ID'] = student_ids_cv

    # Extracting the Student IDs
    student_ids_val_cv = X_val['Student ID']

    # Dropping the student ID column before scaling
    df_robust_val_cv = X_val.drop(columns=['Student ID'])

    #Scaling
    scaled_data_rb_val_cv = rb_cv.transform(df_robust_val_cv)


    # New DataFrame with the scaled data
    X_val_cv = pd.DataFrame(scaled_data_rb_val_cv, columns=df_robust_va

    # Add the Student IDs back
    X_val_cv['Student ID'] = student_ids_val_cv
```

```
#Feature Selection
    X_train_cv = X_train_cv.drop(columns = ["Favourite Study Element_Ea
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

    X_val_cv = X_val_cv.drop(columns = ["Favourite Study Element_Earth"
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

# Here we have our dataset ready to evaluation
# Modeling
    dtc_boost_cv = DecisionTreeClassifier(max_depth = 1, random_state=1
    boost_cv = AdaBoostClassifier(dtc_boost_cv,algorithm='SAMME', learn

    boost_cv.fit(X_train_cv, y_train_cv)
#Predicting
    y_pred_train = boost_cv.predict(X_train_cv)
    y_pred_val = boost_cv.predict(X_val_cv)

# Assessing
    precision_train = precision_score(y_train_cv, y_pred_train)
    precision_val = precision_score(y_val_cv, y_pred_val)

    accuracy_train = accuracy_score(y_train_cv, y_pred_train)
    accuracy_val = accuracy_score(y_val_cv,y_pred_val)

    f1_train = f1_score(y_train_cv, y_pred_train)
    f1_val = f1_score(y_val_cv,y_pred_val)

    conf_matrix_train = confusion_matrix(y_train_cv, y_pred_train)
    conf_matrix_val = confusion_matrix(y_val_cv, y_pred_val)

    recall_train = recall_score(y_train_cv, y_pred_train)
    recall_val = recall_score(y_val_cv, y_pred_val)

    score_train_precision.append(precision_train)
    score_val_precision.append(precision_val)

    score_train_accuracy.append(accuracy_train)
    score_val_accuracy.append(accuracy_val)

    score_train_f1.append(f1_train)
    score_val_f1.append(f1_val)

    score_train_recall.append(recall_train)
    score_val_recall.append(recall_val)

print(f"Recall Train: {np.mean(score_train_recall)}")
print(f"Recall Validation: {np.mean(score_val_recall)}")

print(f"Precision Train: {np.mean(score_train_precision)}")
print(f"Precision Validation: {np.mean(score_val_precision)}")

print(f"Accuracy Train: {np.mean(score_train_accuracy)}")
print(f"Accuracy Validation: {np.mean(score_val_accuracy)}")

print(f"F1 Score Train: {np.mean(score_train_f1)}")
print(f"F1 Score Validation: {np.mean(score_val_f1)}")
```

```
Recall Train: 0.7225147684023153
Recall Validation: 0.726371289855123
Precision Train: 0.7447026487020334
Precision Validation: 0.734261405606588
Accuracy Train: 0.8148741723626535
Accuracy Validation: 0.8071201153728819
F1 Score Train: 0.7333594160619684
F1 Score Validation: 0.7268155584864877
```

Random Forest

In [134]:

```
X_cv = initial_data.drop(["Admitted in School"], axis = 1)
y_cv = initial_data["Admitted in School"]
# KFold Instance
kf = KFold(shuffle = True, random_state = 15, n_splits= 7)

#Lists to store the results
score_train_precision = []
score_train_accuracy = []
score_train_f1 = []
score_train_conf_matrix = []
score_train_recall = []

score_val_precision = []
score_val_accuracy= []
score_val_f1 = []
score_val_conf_matrix= []
score_val_recall = []

# Splitting the data and getting X_train, y_train
for train_index, test_index in kf.split(X_cv):
    X_train_cv, X_val_cv = X_cv.iloc[train_index], X_cv.iloc[test_index
    y_train_cv, y_val_cv = y_cv.iloc[train_index], y_cv.iloc[test_index

#Treating the outliers
    X_train_o_cv= outlier_treatment(X_train_cv)
    X_val_o_cv = outlier_treatment(X_val_cv)

    # Removing the observations in y_train and y_val
    removed_indices_cv = list(set(X_train_cv.index) - set(X_train_o_cv.
    removed_indices_val_cv = list(set(X_val_cv.index) - set(X_val_o_cv.

    y_train_cv = y_train_cv.drop(index=removed_indices_cv)
    y_val_cv = y_val_cv.drop(index=removed_indices_val_cv)


# At this point we don't have outliers so we are going to encode
    X_train_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_val_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_train_o_cv.reset_index(inplace= True, drop = True)
    X_val_o_cv.reset_index(inplace = True, drop = True)

    # Encoding
    encoder_cv = OneHotEncoder()
    df_encoded_train_cv = pd.DataFrame()
    df_encoded_val_cv = pd.DataFrame()

    # Iterating through categorical columns and perform one-hot encodin
    for col in X_train_o_cv.select_dtypes(include='category').columns:
        encoder_cv.fit(X_train_o_cv[[col]])
        encoded_columns_cv = encoder_cv.transform(X_train_o_cv[[col]]).
        column_names_cv = [f"{col}_{(value)}" for value in encoder_cv.c
        encoded_df_cv = pd.DataFrame(encoded_columns_cv, columns=columr

        encoded_columns_val_cv = encoder_cv.transform(X_val_o_cv[[col]]
        column_names_val_cv = [f"{col}_{(value)}" for value in encoder_
        encoded_df_val_cv = pd.DataFrame(encoded_columns_val_cv, columr

        # Concatenate the encoded columns to the result DataFrame
        df_encoded_val_cv = pd.concat([df_encoded_val_cv, encoded_df_va

        # Concatenating the encoded columns to the result DataFrame
```

```
        df_encoded_train_cv = pd.concat([df_encoded_train_cv, encoded_d

    # Concatenating the encoded dataframe to the train
    X_train = pd.concat([X_train_o_cv, df_encoded_train_cv], axis=1)

    # Concatenating the encoded dataframe to the train
    X_val = pd.concat([X_val_o_cv, df_encoded_val_cv], axis=1)

    X_train.drop(["Program" , "Student Gender", "School of Origin", "Fa
    X_val.drop(["Program" , "Student Gender", "School of Origin", "Favo

# At this point we completed the encoding

# Imputting the Missing values
    imputer_cv = KNNImputer(n_neighbors=7)

    # Fit and transform on X_train
    X_train = pd.DataFrame(imputer_cv.fit_transform(X_train.select_dtyp
                                            columns=X_train.select_dtyp
                                            index=X_train.select_dtypes

    # Transform on X_val
    X_val = pd.DataFrame(imputer_cv.transform(X_val.select_dtypes(inclu
                                          columns=X_val.select_dtypes(i
                                          index=X_val.select_dtypes(inc



    # Extracting the Student IDs
    student_ids_cv = X_train['Student ID']

    # Dropping the student ID column before scaling
    df_robust_cv = X_train.drop(columns=['Student ID'])

    #Scaling
    rb_cv = RobustScaler().fit(df_robust_cv)
    scaled_data_rb_cv = rb_cv.transform(df_robust_cv)


    # New DataFrame with the scaled data
    X_train_cv = pd.DataFrame(scaled_data_rb_cv, columns=df_robust_cv.c

    # Add the Student IDs back
    X_train_cv['Student ID'] = student_ids_cv

    # Extracting the Student IDs
    student_ids_val_cv = X_val['Student ID']

    # Dropping the student ID column before scaling
    df_robust_val_cv = X_val.drop(columns=['Student ID'])

    #Scaling
    scaled_data_rb_val_cv = rb_cv.transform(df_robust_val_cv)


    # New DataFrame with the scaled data
    X_val_cv = pd.DataFrame(scaled_data_rb_val_cv, columns=df_robust_va

    # Add the Student IDs back
    X_val_cv['Student ID'] = student_ids_val_cv
```

```
    #Feature Selection
    X_train_cv = X_train_cv.drop(columns = ["Favourite Study Element_Ea
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

    X_val_cv = X_val_cv.drop(columns = ["Favourite Study Element_Earth"
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

# Here we have our dataset ready to evaluation
    #Modeling
    randomf_classifier_cv = RandomForestClassifier(bootstrap = False, c
                                                   max_depth = 20, max_feature
                                                   min_samples_split= 20,n_est

    randomf_classifier_cv.fit(X_train_cv,y_train_cv)
    #Predicting
    y_pred_train = randomf_classifier_cv.predict(X_train_cv)
    y_pred_val =randomf_classifier_cv.predict(X_val_cv)
    #Evaluating
    precision_train = precision_score(y_train_cv, y_pred_train)
    precision_val = precision_score(y_val_cv, y_pred_val)

    accuracy_train = accuracy_score(y_train_cv, y_pred_train)
    accuracy_val = accuracy_score(y_val_cv,y_pred_val)

    f1_train = f1_score(y_train_cv, y_pred_train)
    f1_val = f1_score(y_val_cv,y_pred_val)

    conf_matrix_train = confusion_matrix(y_train_cv, y_pred_train)
    conf_matrix_val = confusion_matrix(y_val_cv, y_pred_val)

    recall_train = recall_score(y_train_cv, y_pred_train)
    recall_val = recall_score(y_val_cv, y_pred_val)

    score_train_precision.append(precision_train)
    score_val_precision.append(precision_val)

    score_train_accuracy.append(accuracy_train)
    score_val_accuracy.append(accuracy_val)

    score_train_f1.append(f1_train)
    score_val_f1.append(f1_val)

    score_train_recall.append(recall_train)
    score_val_recall.append(recall_val)

print(f"Recall Train: {np.mean(score_train_recall)}")
print(f"Recall Validation: {np.mean(score_val_recall)}")

print(f"Precision Train: {np.mean(score_train_precision)}")
print(f"Precision Validation: {np.mean(score_val_precision)}")

print(f"Accuracy Train: {np.mean(score_train_accuracy)}")
print(f"Accuracy Validation: {np.mean(score_val_accuracy)}")

print(f"F1 Score Train: {np.mean(score_train_f1)}")
print(f"F1 Score Validation: {np.mean(score_val_f1)}")
```

```
Recall Train: 0.7760125968778776
Recall Validation: 0.6754247899812652
Precision Train: 0.9009085763027358
Precision Validation: 0.7924325285669823
Accuracy Train: 0.8908660696429843
Accuracy Validation: 0.8184230523892726
F1 Score Train: 0.8335763243321334
F1 Score Validation: 0.7229773607020553
```

Stacking Ensembling

In [135]:

```
X_cv = initial_data.drop(["Admitted in School"], axis = 1)
y_cv = initial_data["Admitted in School"]
# KFold Instance
kf = KFold(shuffle = True, random_state = 15, n_splits= 7)

#Lists to store the results
score_train_precision = []
score_train_accuracy = []
score_train_f1 = []
score_train_conf_matrix = []
score_train_recall = []

score_val_precision = []
score_val_accuracy= []
score_val_f1 = []
score_val_conf_matrix= []
score_val_recall = []

# Splitting the data and getting X_train, y_train
for train_index, test_index in kf.split(X_cv):
    X_train_cv, X_val_cv = X_cv.iloc[train_index], X_cv.iloc[test_index
    y_train_cv, y_val_cv = y_cv.iloc[train_index], y_cv.iloc[test_index

#Treating the outliers
    X_train_o_cv= outlier_treatment(X_train_cv)
    X_val_o_cv = outlier_treatment(X_val_cv)

    # Removing the observations in y_train and y_val
    removed_indices_cv = list(set(X_train_cv.index) - set(X_train_o_cv.
    removed_indices_val_cv = list(set(X_val_cv.index) - set(X_val_o_cv.

    y_train_cv = y_train_cv.drop(index=removed_indices_cv)
    y_val_cv = y_val_cv.drop(index=removed_indices_val_cv)


# At this point we don't have outliers so we are going to encode
    X_train_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_val_o_cv.drop(["School Dormitory"], axis = 1,inplace =True)
    X_train_o_cv.reset_index(inplace= True, drop = True)
    X_val_o_cv.reset_index(inplace = True, drop = True)

    # Encoding
    encoder_cv = OneHotEncoder()
    df_encoded_train_cv = pd.DataFrame()
    df_encoded_val_cv = pd.DataFrame()

    # Iterating through categorical columns and perform one-hot encodin
    for col in X_train_o_cv.select_dtypes(include='category').columns:
        encoder_cv.fit(X_train_o_cv[[col]])
        encoded_columns_cv = encoder_cv.transform(X_train_o_cv[[col]]).
        column_names_cv = [f"{col}_{(value)}" for value in encoder_cv.c
        encoded_df_cv = pd.DataFrame(encoded_columns_cv, columns=columr

        encoded_columns_val_cv = encoder_cv.transform(X_val_o_cv[[col]]
        column_names_val_cv = [f"{col}_{(value)}" for value in encoder_
        encoded_df_val_cv = pd.DataFrame(encoded_columns_val_cv, columr

        # Concatenate the encoded columns to the result DataFrame
        df_encoded_val_cv = pd.concat([df_encoded_val_cv, encoded_df_va

        # Concatenating the encoded columns to the result DataFrame
```

```
        df_encoded_train_cv = pd.concat([df_encoded_train_cv, encoded_d

    # Concatenating the encoded dataframe to the train
    X_train = pd.concat([X_train_o_cv, df_encoded_train_cv], axis=1)

    # Concatenating the encoded dataframe to the train
    X_val = pd.concat([X_val_o_cv, df_encoded_val_cv], axis=1)

    X_train.drop(["Program" , "Student Gender", "School of Origin", "Fa
    X_val.drop(["Program" , "Student Gender", "School of Origin", "Favo

# At this point we completed the encoding

# Imputting the Missing values
    imputer_cv = KNNImputer(n_neighbors=7)

    # Fit and transform on X_train
    X_train = pd.DataFrame(imputer_cv.fit_transform(X_train.select_dtyp
                                            columns=X_train.select_dtyp
                                            index=X_train.select_dtypes

    # Transform on X_val
    X_val = pd.DataFrame(imputer_cv.transform(X_val.select_dtypes(inclu
                                          columns=X_val.select_dtypes(i
                                          index=X_val.select_dtypes(inc



    # Extracting the Student IDs
    student_ids_cv = X_train['Student ID']

    # Dropping the student ID column before scaling
    df_robust_cv = X_train.drop(columns=['Student ID'])

    #Scaling
    rb_cv = RobustScaler().fit(df_robust_cv)
    scaled_data_rb_cv = rb_cv.transform(df_robust_cv)


    # New DataFrame with the scaled data
    X_train_cv = pd.DataFrame(scaled_data_rb_cv, columns=df_robust_cv.c

    # Add the Student IDs back
    X_train_cv['Student ID'] = student_ids_cv

    # Extracting the Student IDs
    student_ids_val_cv = X_val['Student ID']

    # Dropping the student ID column before scaling
    df_robust_val_cv = X_val.drop(columns=['Student ID'])

    #Scaling
    scaled_data_rb_val_cv = rb_cv.transform(df_robust_val_cv)


    # New DataFrame with the scaled data
    X_val_cv = pd.DataFrame(scaled_data_rb_val_cv, columns=df_robust_va

    # Add the Student IDs back
    X_val_cv['Student ID'] = student_ids_val_cv
    #Feature Selection
```

```
    X_train_cv = X_train_cv.drop(columns = ["Favourite Study Element_Ea
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

    X_val_cv = X_val_cv.drop(columns = ["Favourite Study Element_Earth"
                                            "Program_Witchcraft Institute", "
                                            "School of Origin_Eldertree Encla
                                           "Student ID"], axis = 1)

# Here we have our dataset ready to evaluation
# Modeling
    base_classifier_dtc_cv = DecisionTreeClassifier(random_state=15)
    base_model_dtc_boost_cv = DecisionTreeClassifier(max_depth=1, randc

    est = [('decision_tree', DecisionTreeClassifier(criterion='gini', m
       ('gaussian_nb', GaussianNB(var_smoothing=0.15199110829529336)),
       ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=10, met
       ('logistic_regression', LogisticRegression(fit_intercept=True, m
      ('bagging', BaggingClassifier(base_classifier_dtc_cv,bootstrap= T
      ('boosting',AdaBoostClassifier(base_model_dtc_boost_cv,algorithm=
      ('random_forest', RandomForestClassifier(random_state = 15, boots

    meta_classifier_cv = AdaBoostClassifier(n_estimators=100, random_st

    stacking_clf_cv = StackingClassifier(estimators=est, final_estimato
    stacking_clf_cv.fit(X_train_cv, y_train_cv)
    #Predicting
    y_pred_train = stacking_clf_cv.predict(X_train_cv)
    y_pred_val = stacking_clf_cv.predict(X_val_cv)
    #Evaluating
    precision_train = precision_score(y_train_cv, y_pred_train)
    precision_val = precision_score(y_val_cv, y_pred_val)

    accuracy_train = accuracy_score(y_train_cv, y_pred_train)
    accuracy_val = accuracy_score(y_val_cv,y_pred_val)

    f1_train = f1_score(y_train_cv, y_pred_train)
    f1_val = f1_score(y_val_cv,y_pred_val)

    conf_matrix_train = confusion_matrix(y_train_cv, y_pred_train)
    conf_matrix_val = confusion_matrix(y_val_cv, y_pred_val)

    recall_train = recall_score(y_train_cv, y_pred_train)
    recall_val = recall_score(y_val_cv, y_pred_val)

    score_train_precision.append(precision_train)
    score_val_precision.append(precision_val)

    score_train_accuracy.append(accuracy_train)
    score_val_accuracy.append(accuracy_val)

    score_train_f1.append(f1_train)
    score_val_f1.append(f1_val)

    score_train_recall.append(recall_train)
    score_val_recall.append(recall_val)

print(f"Recall Train: {np.mean(score_train_recall)}")
print(f"Recall Validation: {np.mean(score_val_recall)}")
```

```
print(f"Precision Train: {np.mean(score_train_precision)}")
print(f"Precision Validation: {np.mean(score_val_precision)}")

print(f"Accuracy Train: {np.mean(score_train_accuracy)}")
print(f"Accuracy Validation: {np.mean(score_val_accuracy)}")

print(f"F1 Score Train: {np.mean(score_train_f1)}")
print(f"F1 Score Validation: {np.mean(score_val_f1)}")
```

```
Recall Train: 0.7233575536887625
Recall Validation: 0.664698384034271
Precision Train: 0.8300765003186392
Precision Validation: 0.7435044871084916
Accuracy Train: 0.8492054683037132
Accuracy Validation: 0.7970646476412346
F1 Score Train: 0.7720433345437947
F1 Score Validation: 0.6955766330958663
```

Model Deployement

Pre Processing

Import

In [136]:

```
data = pd.read_csv("Project_test_dataset.csv")
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 176 entries, 0 to 175
Data columns (total 11 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Student ID                176 non-null    int64  
 1   Program                   176 non-null    object 
 2   Student Gender            176 non-null    object 
 3   Experience Level          145 non-null    float64
 4   Student Siblings          176 non-null    int64  
 5   Student Family            176 non-null    int64  
 6   Financial Background      176 non-null    float64
 7   School Dormitory          49 non-null     object 
 8   School of Origin          176 non-null    object 
 9   Student Social Influence  176 non-null    int64  
 10  Favourite Study Element   176 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 15.2+ KB
```

Outlier Treatment modified

In [137]:
```
# Set the limits for winsorization
lower_limit = 0
upper_limit = 90
outliers = ["Experience Level", "Student Siblings", "Student Family"]

    # Manual calculation of percentiles
percentiles_before_winsorize = data[outliers].quantile([lower_limit/100

for variable in outliers:
    data[variable] = data[variable].clip(
        lower=percentiles_before_winsorize.loc[lower_limit / 100, varia
        upper=percentiles_before_winsorize.loc[upper_limit / 100, varia

data['Financial Background'] = data['Financial Background'].apply(lambd
```

Encoding

In [138]:
```
data.drop(["School Dormitory"], axis = 1,inplace =True)
```

In [139]:
```
df_encoded_dep = pd.DataFrame()

for col in data.select_dtypes(include='object').columns:
    encoder.fit(X_train_o[[col]])
    encoded_columns_d = encoder.transform(data[[col]]).toarray()
    column_names_d = [f"{col}_{(value)}" for value in encoder.categorie
    encoded_df_d = pd.DataFrame(encoded_columns_d, columns=column_names

    # Concatenating the encoded columns to the result DataFrame
    df_encoded_dep = pd.concat([df_encoded_dep, encoded_df_d], axis=1)

# Concatenating the encoded dataframe
data_ready = pd.concat([data, df_encoded_dep], axis=1)
```

In [140]:
```
data_ready.drop(["Program" , "Student Gender", "School of Origin", "Fav
```

Imput missing values

In [141]:
```
# Transform
data_ready = pd.DataFrame(imputer.transform(data_ready.select_dtypes(ir
                                            columns=data_ready.select_d
                                            index=data_ready.select_dty
```

Scaling

In [142]:
```
# Extracting the Student IDs
student_ids = data_ready['Student ID']

# Dropping the student ID column before scaling
df_d = data_ready.drop(columns=['Student ID'])

#Scaling
scaled_data_d = rb.transform(df_d)


# New DataFrame with the scaled data
data_dep = pd.DataFrame(scaled_data_d, columns=df_d.columns)

# Add the Student IDs back
data_dep['Student ID'] = student_ids
```

Feature Selection

In [143]:
```
data_dep = data_dep.drop(columns = ["Favourite Study Element_Earth", "F
                                    "Program_Witchcraft Institute", "
                                    "School of Origin_Eldertree Encla
                                   ], axis = 1)
```

Predicting

In [144]:
```
student_ids = data_dep['Student ID']
data_dep.drop(columns = ["Student ID"],inplace = True)
```

In [145]:
```
dtc_boost_cv = DecisionTreeClassifier(max_depth = 1, random_state=15)
boost_cv = AdaBoostClassifier(dtc_boost_cv,algorithm='SAMME', learning_

boost_cv.fit(X_train_rb, y_train)


# Make predictions
predictions = boost_cv.predict(data_dep)
predictions = predictions.astype(int)
```

Final csv

In [146]:
```
# Create the DataFrame with student IDs and predictions
result_df = pd.DataFrame({'Student ID': student_ids, 'Admitted in Schoc
result_df
```

Out[146]:

	Student ID	Admitted in School
0	836.0	1
1	323.0	1
2	117.0	0
3	444.0	1
4	619.0	1
...	...	...
171	21.0	0
172	366.0	0
173	890.0	0
174	749.0	0
175	653.0	0

176 rows × 2 columns

In [147]:
```
# Export the predictions to a CSV file
result_df.to_csv('predictionsboost.csv',index =False)
```

In [ ]:
```

```