Capstone Project

Gbadamassi, Luqman

Intro to Data Science

Professor Wallisch

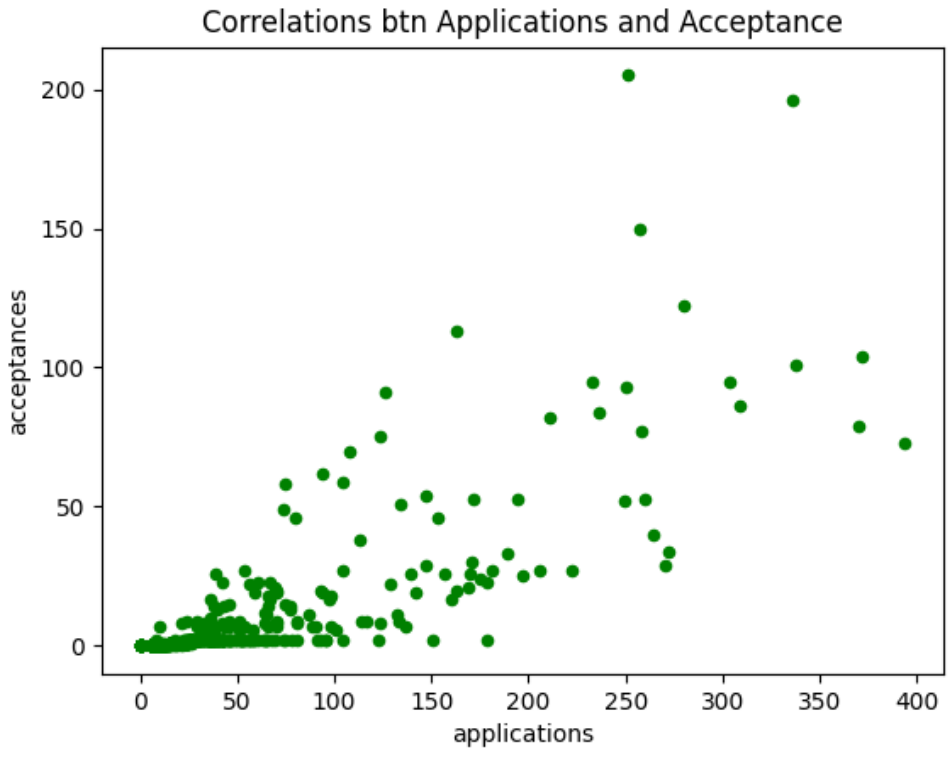
08/22/2022

In this work I used pandas to read and manipulate the csv dataset. I dropped null rows since they raised some errors in the programs

For plotting the charts and graphs I used matplotlib and seaborn modules

I also used numpy, sklearn and scipy for principal component analysis for dimensionality reduction and statsmodel package

1. What is the correlations between the number applications and admissions to HSPHS?
   I selected the two columns for applications and acceptance from the Pandas data frame to plot the scatter plot using matplotlib

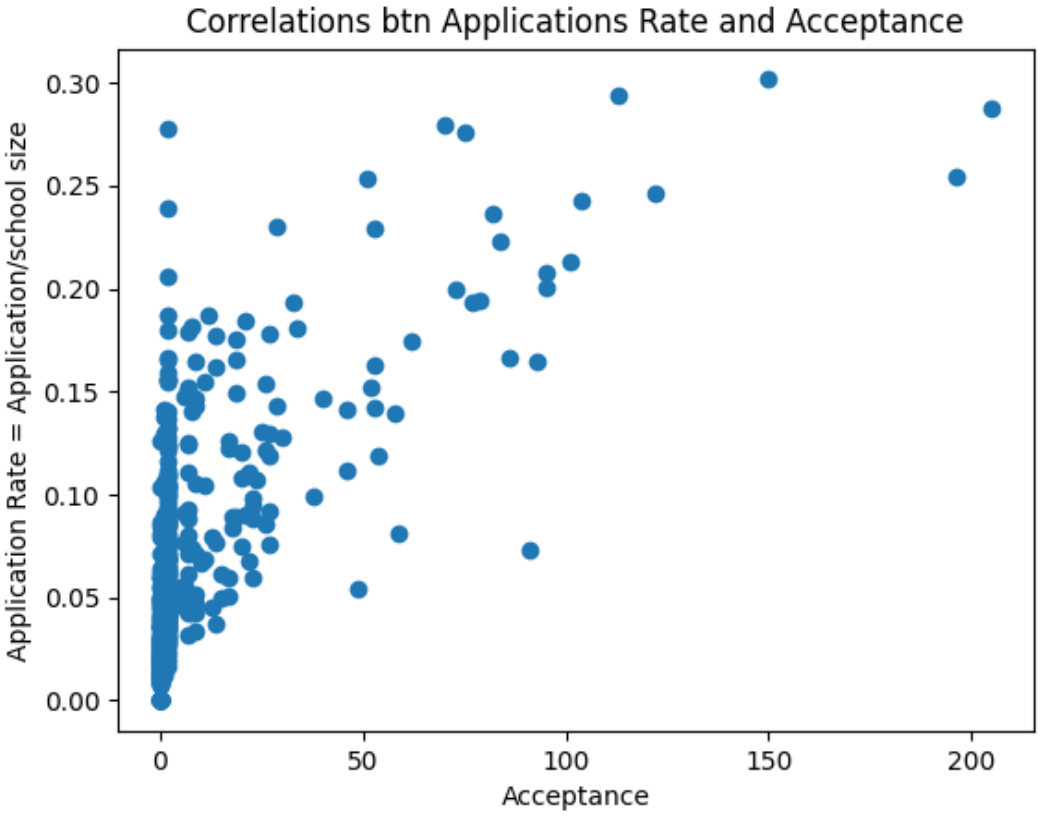


   I used the correlation function to obtain the correlation factor between applications and acceptance = 0.802 rounded to 3 decimal places.
2. What is a better predictor of admission to HSPHS? Raw number of applications or application "rate"?

From the hints provided, I calculated the application rate by dividing applications by school size

$$application\ rate\ = \frac{applications}{school\ size}$$

Using Numpy I created a new column for the application rate and used the correlation function to obtain a correlation factor of 0.6588 for the application rate and the acceptance columns According to the obtained correlation for the new value and the previous one, I would argue that raw number of applications is a better predictor of acceptances since it has a higher correlation coefficient of 0.802



`

Figure 2: Figure illustrating the correlation between the rate of application and acceptances

3. Which school has the best "per student" odds of sending someone to HSPHS?

For this question requiring the school with best odds for sending someone to HSPHS, I first calculated the acceptance rate using the formula $Acceptance\ Rate\ = \frac{acceptances}{School\ size}$ **and saved the results to an array**

I then calculated for "per student" odds using the formula

$$per\ student\ odds\ = \frac{acceptance\ rate}{1-acceptance\ rate}$$

I then sorted the arrays using Numpy Sorting algorithm (argmax) which gave "THE CHRISTA MCAULIFFE SCHOOL\I.S. 187" with the best odd of 0.306886 at index 304

4. Is there a relationship between how students perceive their school (as reported in columns L-Q) and how the school performs on objective measures of achievement (as noted in columns V-X)?

Here I first plotted a correlation scatter plot to establish if the features have some sort of relationship

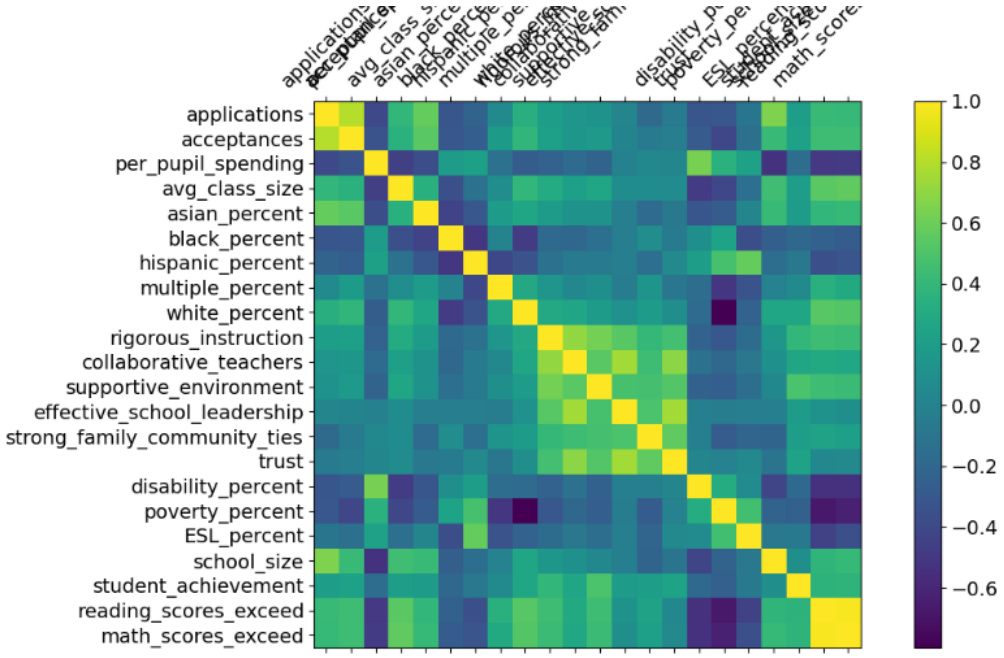
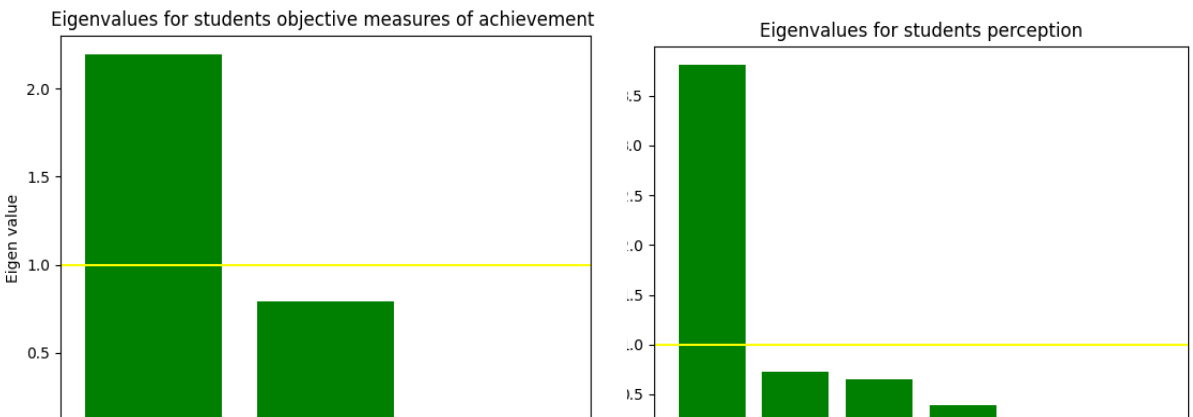


Figure 3: Correlation table for various features

Having established some relationship and multiple correlations, I divided the features to two (from columns L- Q and Columns V-X) to carry out two PCAs

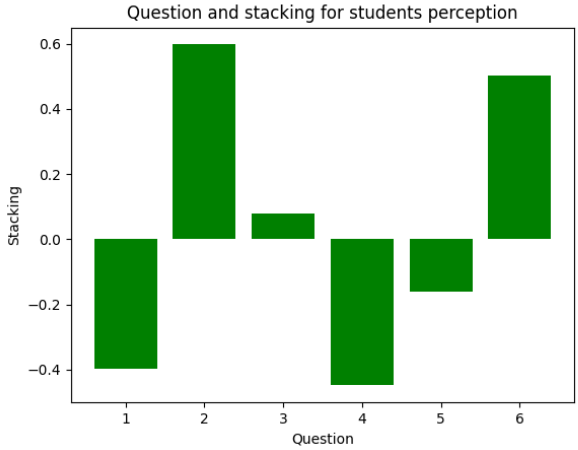
I used StandardScaler to standardize the dataset's features onto unit scale (mean = 0 and variance = 1) which is a requirement for the optimal performance of the machine learning algorithms

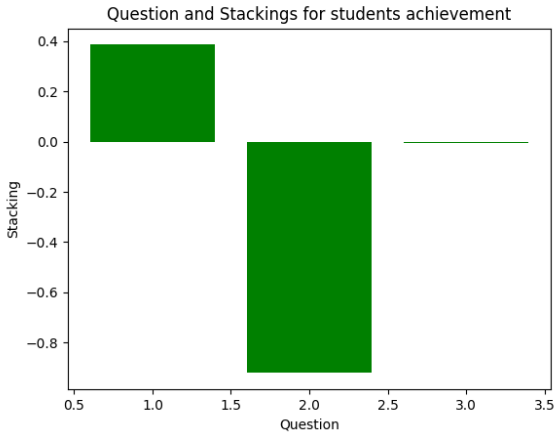
The original data had 6 columns ('rigorous_instruction', 'collaborative_teachers', 'supportive_environment','effective_school_leadership','strong_family_community_ties','trust'



) which the code projected the original data which is 6 and 3(for measures of achievement) dimensional into 2 dimensions

I used the Kaiser rule (yellow line) to select one important feature for each case then compared the scores of each stacking





For the perception chart on the left I found two strong features and two weak features. For the stackings of student achievements, I found the stacking of Reading scores as the vital feature

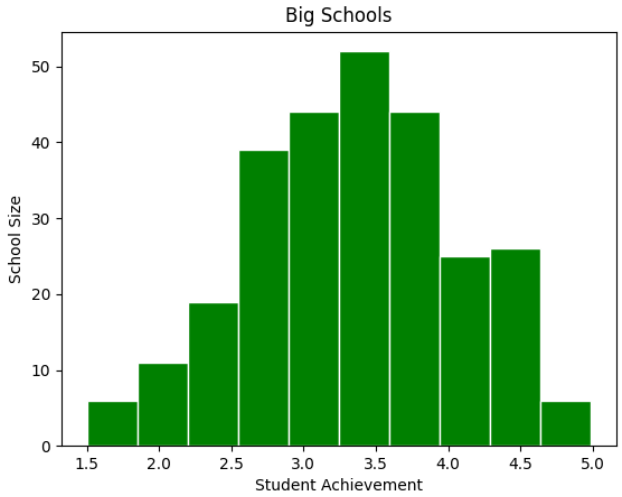
I used ANOVA for each of the first principal component axes to get a quantitative estimate of the significance of the clusters

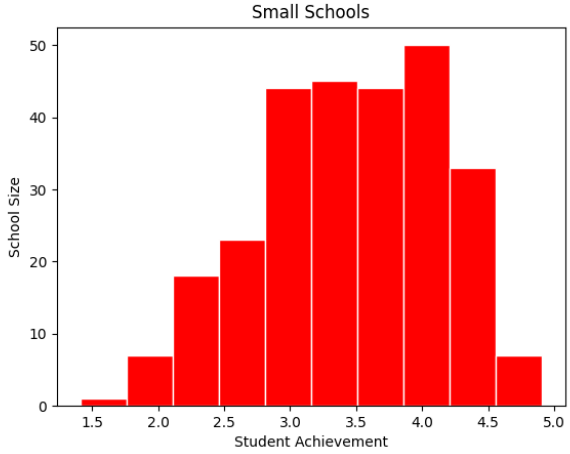
I observed that trust, reading scores exceed, and rigorous instruction all gives significant p-values with good F-values and sums of squares and the 'reading scores exceed' was deterministic as well therefore the way students perceive their school has a positive influence on their objective measures of achievement

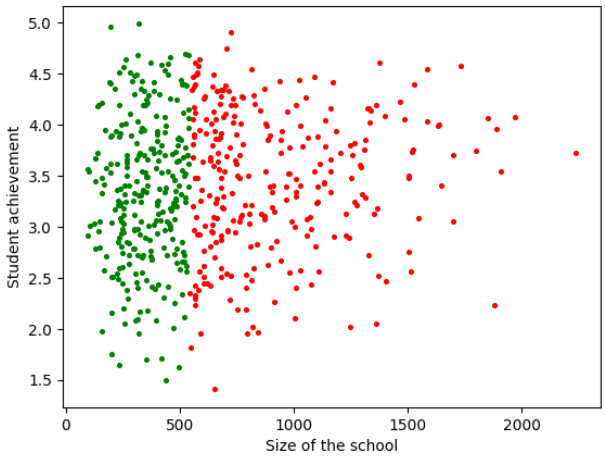
5. Test a hypothesis of your choice as to which kind of school (e.g. small schools vs large schools or charter schools vs. not (or any other classification, such as rich vs. poor school)) performs differently than another kind either on some dependent measure, e.g. objective measures of achievement or admission to HSPHS (pick one).
My hypothesis is big schools performs better than small schools' in respect to objective measures of achievement

I first plotted a scatter plot of both sizes of schools against the achievement of students and the way the plots are distributed intuitively informs that there is no relationship therefore my hypothesis is null (school size does not affect the student achievement)

I will plot two independent normal distributions to compare each performance to justify this



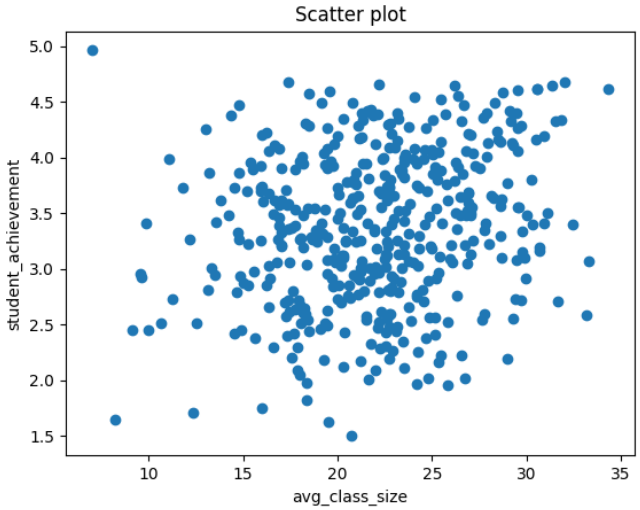




I used the statsmodel module to compute for p- value which was 0.06516976 which opposes my hypothesis

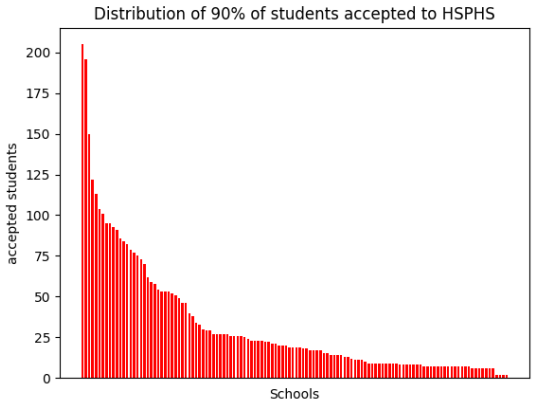
Therefore, the school size has no effect on the student achievement

6. Is there any evidence that the availability of material resources (e.g. per student spending or class size) impacts objective measures of achievement or admission to HSPHS?



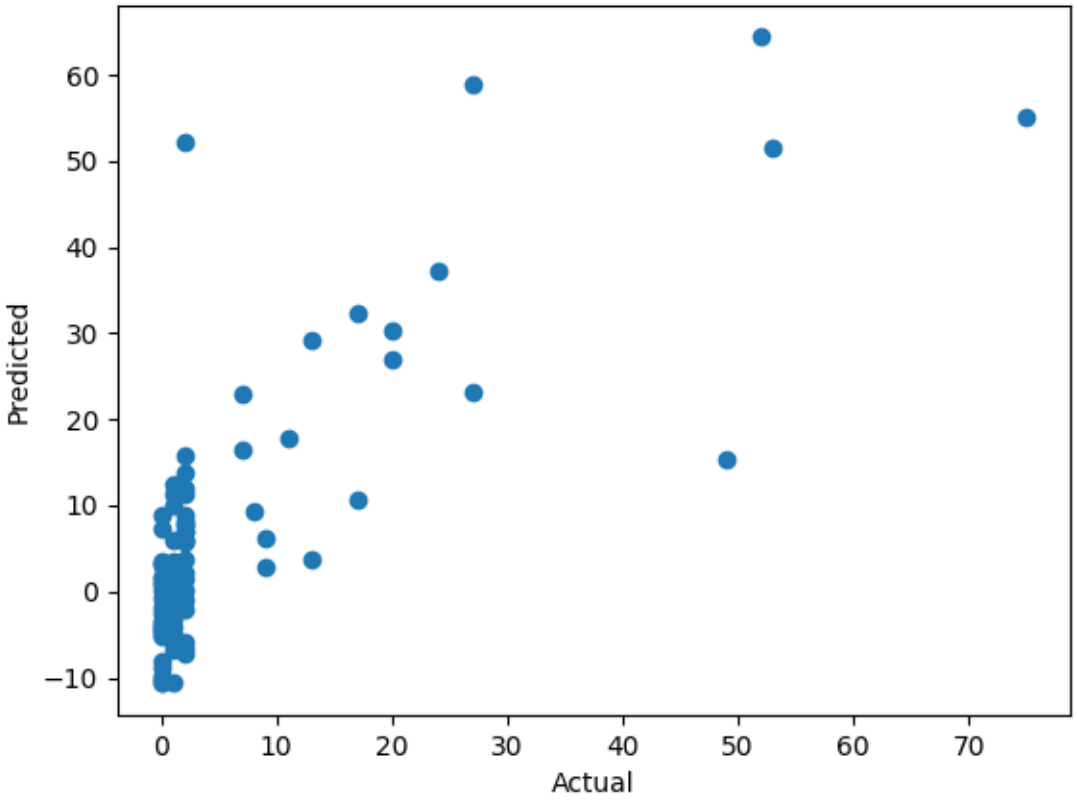
For this question I tested class size against the student achievement levels. After removing nan variables from the array, I calculated the median of the average class size. I used the median to slice the data into two categories, either greater than or less than the median. Since the data was normally distributed, I did a t-test which yielded a p-value of 0.002 and was less than 0.05 of my alpha level which clues that the class size has no effect on objective measures of achievement.

7. What proportion of schools accounts for 90% of all students accepted to HSPHS?

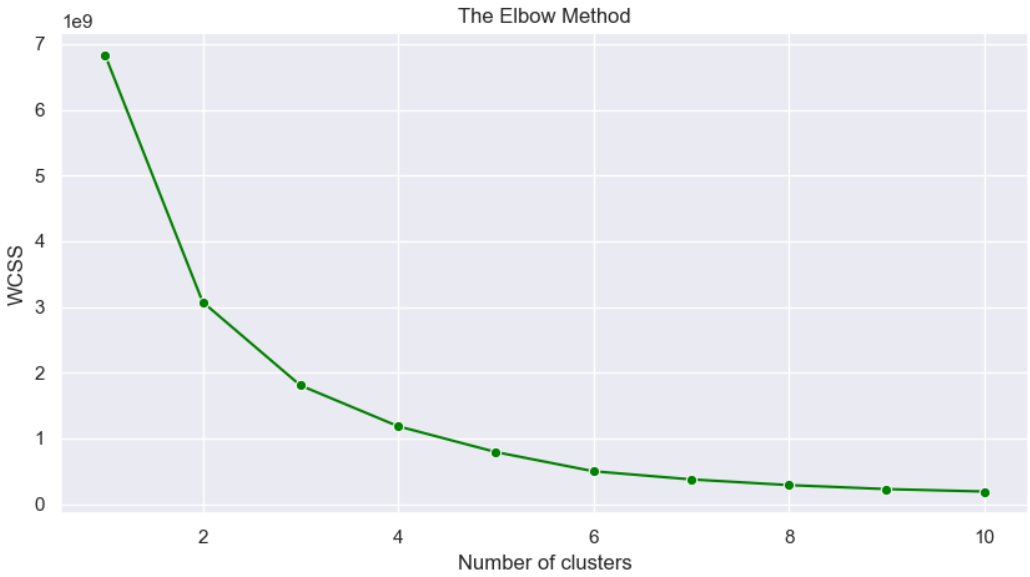


I sorted the number of acceptances from the most to the least then found out how many students equated to 90% of all accepted to HSPHS. I iterated over the sorted array as I added to subtotal until my result was equal or large to the value that corresponded to 90% which was about 20.7% of all schools

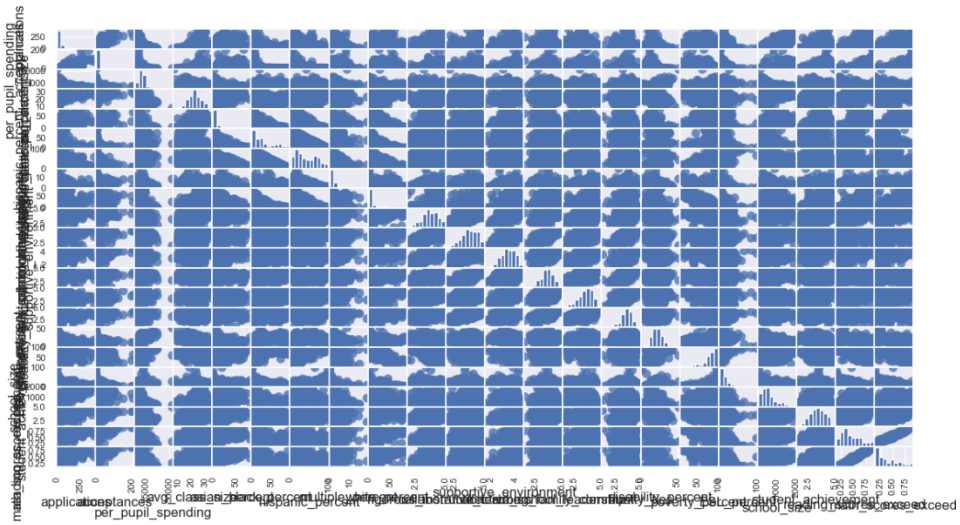
8. Build a model of your choice – clustering, classification or prediction – that includes all factors – as to what school characteristics are most important in terms of:
I first started with multiple regression but the model was not so accurate (accuracy of 37.6%) so I preferred to use clustering



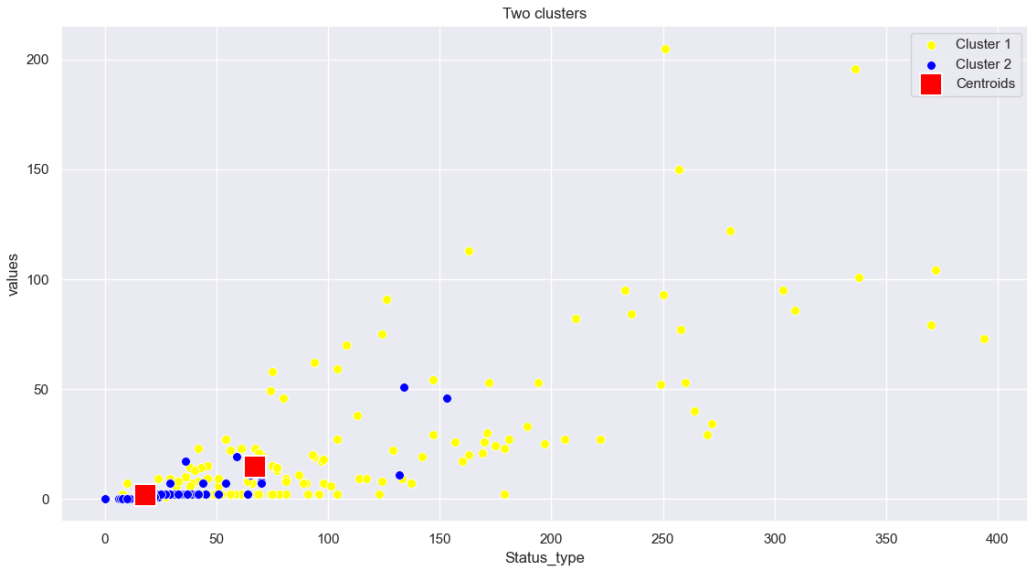
Before starting with the clustering, I explored the data by loading the dataset .csv into Python with pandas. I used the K-means clustering algorithm and Elbow method Clustering algorithms allowed me to discover homogeneous subgroups within the dataset so that data points in each cluster were comparable and similar to each other based on a similarity measure like euclidean-based distance and correlation-based distance. Instead of iterating over different values for K manually, I plotted the Elbow algorithm results to get the optimal value for K from the graph which was 2



Used pairplot for Visualizing the data correlation in the dataset



The clusters of the highly determining features from the pca



a) Sending students to HSPHS
I established that the raw applications have a huge impact on the acceptance

b) Achieving high scores on objective measures of achievement
A supportive environment is a key factor to better measures of achievement which goes from having a conducive learning environment, healthy community and supportive teachers

9. Write an overall summary of your findings – what school characteristics seem to be most relevant in determining acceptance of their students to HSPHS?
Generally, all the features on the data set provided can have a significant effect on determining the acceptance of students since each school has its own unique set of strengths and weaknesses. From the analysis I did above reading scores exceeded have a high impact on the achievement of the students therefore the students need incentives that would make their performances improve like a good class to teacher ratio and a positive surrounding and school infrastructure
10. Imagine you are working for the New York City Department of Education as a data scientist (like one of my former students). What actionable recommendations would you make on how to improve schools so that they

a) Send more students to HSPHS
From the models created and analysis done, only a small fraction of the schools is considered for most of the applications (about 20% of schools for 90% of admissions), however, student achievement has a high impact on the acceptance so I would recommend students to apply as much as possible to HSPHS and work on their achievements to boost their chances

b) Improve objective measures or achievement
Create a conducive and supportive learning environment through healthy relationships between students and good teacher-student relationship

Hire more teachers to have more one-to-one sessions to create good rapport between staff and students

CODE

```
#Packages imported

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

import statsmodels.api as sm

from statsmodels.formula.api import ols

from sklearn.cluster import KMeans

import seaborn as sn

from sklearn.cluster import KMeans, DBSCAN # clustering algorithms

import warnings

from sklearn.model_selection import train_test_split

from sklearn.decomposition import PCA # dimensionality reduction

from sklearn.metrics import r2_score

from sklearn.metrics import silhouette_score # used as a metric to evaluate the cohesion in a cluster

from sklearn.neighbors import NearestNeighbors # for selecting the optimal eps value when using DBSCAN

from sklearn.preprocessing import LabelEncoder

from scipy import stats

from matplotlib import colors

from matplotlib.ticker import PercentFormatter


#Read code

dataset = pd.read_csv("middleSchoolData.csv", encoding='utf-8')

df = pd.DataFrame(dataset)

print(df.columns)

#%%Question 1 ---> Correlations btn Apk and Acceptance
```

```
df.plot(kind = 'scatter', x = 'applications', y = 'acceptances', color = "g")
plt.title('Correlations btn Applications Rate and Acceptance')
plt.show()
corr_fact = (df['applications'].corr(df['acceptances']))
print(corr_fact.round(3))

#%%Question 2
apk_rate = df["applications"]/df["school_size"]
corr = apk_rate.corr(df['acceptances'])
print(corr.round(4))
plt.title('Correlations btn Applications Rate and Acceptance')
plt.scatter(df['acceptances'], apk_rate)
plt.xlabel("Acceptance")
plt.ylabel("Application Rate = Application/school size")
plt.show()

#%%%Question 3
acpt_rate = df["acceptances"]/df["school_size"]
odds = acpt_rate/(1-acpt_rate)
print(odds[304])
print(np.argmax(odds))
print(df["school_name"][np.argmax(odds)])

#%%Question 4
#plt.matshow(df.corr())
#plt.show()

f = plt.figure(figsize=(19, 15))
plt.matshow(df.corr(), fignum=f.number)
plt.xticks(range(df.select_dtypes(['number']).shape[1]), df.select_dtypes(['number']).columns, fontsize=14, rotation=45)
plt.yticks(range(df.select_dtypes(['number']).shape[1]), df.select_dtypes(['number']).columns, fontsize=14)
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
```

```
plt.title('Correlation Matrix', fontsize=16);

plt.show()


df = pd.read_csv("middleSchoolData.csv").dropna()

plt.scatter(df['avg_class_size'],df['student_achievement'])

plt.show()

#Clean the data using dropna

features = ['rigorous_instruction', 'collaborative_teachers', 'supportive_environment',
'effective_school_leadership','strong_family_community_ties','trust','student_achievement']

# Separating out the features

x = df.loc[:, features].values

# Separating out the target

y = df.loc[:,['student_achievement']].values

# Standardizing the features

x = StandardScaler().fit_transform(x)


pca = PCA(n_components=2)

principalComponents = pca.fit_transform(x)

principalDf = pd.DataFrame(data = principalComponents

        , columns = ['principal component 1', 'principal component 2'])

finalDf = pd.concat([principalDf, df[['student_achievement']]], axis = 1)


#alt 1

#load dataframe and drop null

df = pd.read_csv('middleSchoolData.csv').dropna()

features = ['rigorous_instruction', 'collaborative_teachers', 'supportive_environment',
'effective_school_leadership','strong_family_community_ties','trust','student_achievement']

# Separating out the features

X = df.loc[:,features].values

y = df.loc[:,['student_achievement']].values

X.shape, y.shape

X_std = StandardScaler().fit_transform(X)

cov_mat= np.cov(X_std, rowvar=False)
```

```
cov_mat = np.cov(X_std.T)

eig_vals, eig_vecs = np.linalg.eig(cov_mat)


pca = PCA(n_components=2) # Here we can also give the percentage as a paramter to the PCA function as pca = PCA(.95). .95 means that we want to include 95% of the variance. Hence PCA will return the no of components which describe 95% of the variance. However we know from above computation that 2 components are enough so we have passed the 2 components.

principalComponents = pca.fit_transform(X_std)

principalDf = pd.DataFrame(data = principalComponents

        , columns = ['principal component 1', 'principal component 2'])

#print(principalDf['principal component 1']) # prints the top 5 rows

plt.plot(pca.explained_variance_)

plt.xlabel('Principal Component')

plt.ylabel('Eigenvalue')

plt.axhline(y=1, linewidth=1, color='r')

plt.title('Scree Plot of PCA: Component Eigenvalues')

plt.show()


#alt2


#%%Question 5


sizeTotal = df['school_size']

achiev = df['student_achievement']

sizeAchievementSS =(pd.concat([sizeTotal,achieve],axis=1)).dropna() #drops na's no charters

medianSSize = sizeAchiev['school_size'].median() #gives us the media which will be used to sort big/small class

#545.0

sizeAchievementSize = (sizeAchievementSS.sort_values(by ='school_size') ).reset_index(drop=True)

achievementBSize = sizeAchievementSize.iloc[:272,:] #slicing

achievementSSize = sizeAchievementSize.iloc[273:,:]

u2,p2 = stats.mannwhitneyu(achievementSSize['student_achievement'],achievementBSize['student_achievement'])

#Significant => Difference between performance depending on the school size
```

```
plt.scatter(achievementSSize['school_size'],achievementSSize['student_achievement'],color='red',s=7)
plt.scatter(achievementBSize['school_size'],achievementBSize['student_achievement'],color='green',s=7)
plt.ylabel('Student achievement')
plt.xlabel('Size of the school')
plt.show()
plt.hist(achievementBSize['student_achievement'], color = 'green', edgecolor = 'white')
plt.title('Big Schools')
plt.ylabel('School Size')
plt.xlabel('Student Achievement')
plt.show()

plt.hist(achievementSSize['student_achievement'], color = 'red', edgecolor = 'white')
plt.title('Small Schools')
plt.ylabel('School Size')
plt.xlabel('Student Achievement')
sizeschool_achievement_corr = data['school_size'].corr(data['student_achievement'])
#low correlation
print(stats.ttest_rel(achievementBSize['student_achievement'],achievementSSize['student_achievement']))
#Shows that there's no significant difference between the bot



#%%Question 6
df = pd.read_csv("middleSchoolData.csv").dropna()
x = df['acceptances']
y = df['student_achievement']
# Creating distribution
legend = ['distribution']

# Creating histogram
fig, axs = plt.subplots(1, 1,
                        figsize =(10, 7),
                        tight_layout = True)
```

```
# Remove axes splines
for s in ['top', 'bottom', 'left', 'right']:
    axs.spines[s].set_visible(False)

# Remove x, y ticks
axs.xaxis.set_ticks_position('none')
axs.yaxis.set_ticks_position('none')

# Add padding between axes and labels
axs.xaxis.set_tick_params(pad = 5)
axs.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
axs.grid(b = True, color ='grey',
        linestyle ='-.', linewidth = 0.5,
        alpha = 0.6)



# Creating histogram
N, bins, patches = axs.hist(x)
# Setting color
fracs = ((x**(1 / 5)) / x.max())
norm = colors.Normalize(fracs.min(), fracs.max())

for thisfrac, thispatch in zip(fracs, patches):
    color = plt.cm.viridis(norm(thisfrac))
    thispatch.set_facecolor(color)
plt.hist(df['acceptances'],color = 'g')
plt.title('Distribution of accepted students')
plt.xlabel('schools')
plt.ylabel('No of accepted students')
plt.show()
```

```
#%%Question 7
df = pd.read_csv("middleSchoolData.csv")
sch = df['school_name']
accept = data['acceptances']
accept = raw_accept.to_numpy() #convert to np
accept = sorted(accept,reverse=True) #Sort in decending order
total_accept = sum(accept)
accept_90 = 0.9*total_accept #check how many student account for 90%
tempCounter = 0
loopCounter = 0
for i in accept: #simple for loop counting and checking if the total students are equivalent of the # of "90%"
    if tempCounter>=accept_90:
        break
    else:
      tempCounter+=i
      loopCounter+=1
      schools_no = len(accept)
      proportion = loopCounter/schools_no
      # Roughly 20% of schools make up for 90% of acceptances to HSPHS
      # 124 schools up until index 123
      accept = data['acceptances']
      acceptNames = pd.concat([sch,accept],axis=1)
      acceptNames = (acceptNames.sort_values(by = 'acceptances',ascending=False).reset_index(drop=True))
      acceptNames = acceptNames.iloc[:124,:]
      x_pos = np.arange(len(acceptNames))
      plt.bar(x_pos,acceptNames['acceptances'],color='r',width=
      0.7)
      plt.xticks(x_pos,acceptNames['school_name'])
      plt.xticks(())
      plt.title('Distribution of 90% of students accepted to HSPHS')
      plt.xlabel('Schools')
      plt.ylabel('accepted students')
```

```
    plt.show()
#%%Question 8
#Multiple Linear Regression
warnings.filterwarnings("ignore")
df = pd.read_csv('middleSchoolData.csv').dropna()
df.drop(['dbn', 'school_name'], axis=1, inplace=True)
x = df.drop(['acceptances'],axis=1)
y = df['acceptances']

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
linreg=LinearRegression()
linreg.fit(x_train,y_train)
y_pred=linreg.predict(x_test)
Accuracy=r2_score(y_test,y_pred)*100

print(" Accuracy of the model is %.2f" %Accuracy)
plt.scatter(y_test,y_pred);
plt.xlabel('Actual');
plt.ylabel('Predicted');
plt.show()

#alt 1
def silhouettePlot(range_, data):
  '''
  we will use this function to plot a silhouette plot that helps us to evaluate the cohesion in clusters (k-means only)
  '''
  half_length = int(len(range_)/2)
  range_list = list(range_)
  fig, ax = plt.subplots(half_length, 2, figsize=(15,8))
  for _ in range_:
    kmeans = KMeans(n_clusters=_, random_state=42)
    q, mod = divmod(_ - range_list[0], 2)
    sv = SilhouetteVisualizer(kmeans, colors="yellowbrick", ax=ax[q][mod])
```

```
        ax[q][mod].set_title("Silhouette Plot with n={} Cluster".format(_))
        sv.fit(data)
    fig.tight_layout()
    fig.show()
silhouettePlot(range_, data)
def elbowPlot(range_, data, figsize=(10,10)):
    '''
    the elbow plot function helps to figure out the right amount of clusters for a dataset
    '''
    inertia_list = []
    for n in range_:
        kmeans = KMeans(n_clusters=n, random_state=42)
        kmeans.fit(data)
        inertia_list.append(kmeans.inertia_)

    # plotting
    fig = plt.figure(figsize=figsize)
    ax = fig.add_subplot(111)
    sns.lineplot(y=inertia_list, x=range_, ax=ax)
    ax.set_xlabel("Cluster")
    ax.set_ylabel("Inertia")
    ax.set_xticks(list(range_))
    fig.show()
def findOptimalEps(n_neighbors, data):
    '''
    function to find optimal eps distance when using DBSCAN; based on this article:
https://towardsdatascience.com/machine-learning-clustering-dbscan-determine-the-optimal-value-for-epsilon-eps-python-example-3100091cfbc
    '''
    neigh = NearestNeighbors(n_neighbors=n_neighbors)
    nbrs = neigh.fit(data)
    distances, indices = nbrs.kneighbors(data)
    distances = np.sort(distances, axis=0)
```

```
    distances = distances[:,1]
    plt.plot(distances)
def progressiveFeatureSelection(df, n_clusters=3, max_features=4,):
    '''
    very basic implementation of an algorithm for feature selection (unsupervised clustering); inspired by this post:
https://datascience.stackexchange.com/questions/67040/how-to-do-feature-selection-for-clustering-and-implement-it-in-pytho
n
    '''
    feature_list = list(df.columns)
    selected_features = list()
    # select starting feature
    initial_feature = ""
    high_score = 0
    for feature in feature_list:
        kmeans = KMeans(n_clusters=n_clusters, random_state=42)
        data_ = df[feature]
        labels = kmeans.fit_predict(data_.to_frame())
        score_ = silhouette_score(data_.to_frame(), labels)
        print("Proposed new feature {} with score {}". format(feature, score_))
        if score_ >= high_score:
            initial_feature = feature
            high_score = score_
    print("The initial feature is {} with a silhouette score of {}.".format(initial_feature, high_score))
    feature_list.remove(initial_feature)
    selected_features.append(initial_feature)
    for _ in range(max_features-1):
        high_score = 0
        selected_feature = ""
        print("Starting selection {}...".format(_))
        for feature in feature_list:
            selection_ = selected_features.copy()
            selection_.append(feature)
            kmeans = KMeans(n_clusters=n_clusters, random_state=42)
```

```
            data_ = df[selection_]
            labels = kmeans.fit_predict(data_)
            score_ = silhouette_score(data_, labels)
            print("Proposed new feature {} with score {}". format(feature, score_))
            if score_ > high_score:
                selected_feature = feature
                high_score = score_
        selected_features.append(selected_feature)
        feature_list.remove(selected_feature)
        print("Selected new feature {} with score {}". format(selected_feature, high_score))
    return selected_features

#alt 2 --> Using KMEANS
from sklearn.cluster import KMeans
df = pd.read_csv('middleSchoolData.csv').dropna()
df.drop(['dbn', 'school_name'], axis=1, inplace=True)
X = df
y = df['acceptances']
kmeans = KMeans(n_clusters=2, random_state=0)
ninety_quantile = df["acceptances"].quantile(0.9)
#df = df[df["acceptances"] <= ninety_quantile]
#print(df.head())
# fitting the values
kmeans.fit(X)

wcss = []
# for loop
for i in range(1, 11):

    # k-mean cluster model for different k values
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
```

```
    # inertia method returns wcss for that model
    wcss.append(kmeans.inertia_)
plt.figure(figsize=(10,5))
sns.lineplot(range(1, 11), wcss,marker='o',color='green')

# labeling
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

# creating graph in one line
fig, ax = plt.subplots(1, 3, gridspec_kw={'wspace': 0.3}, figsize=(15,5))

# for loop
for i in range(3):
    km = KMeans(n_clusters = 3, init='random', n_init=1, random_state=i)
    km.fit(X)
    ax[i].scatter(x= X.iloc[:, 0], y=X.iloc[:, 1], c= km.labels_)

#     printing the interia with graphs
    ax[i].set_title(f"Inertia = {round(km.inertia_, 2)}");
    plt.show()

g = pd.plotting.scatter_matrix(df, figsize=(10,10), marker = 'o', hist_kwds = {'bins': 10}, s = 60, alpha = 0.8)
plt.show()

kmeans = KMeans(n_clusters = 2, random_state = 42)
y_kmeans = kmeans.fit_predict(X)
X = np.array(X)

# size of image
plt.figure(figsize=(15,7))
```

```
# visualizing the clusters
sns.scatterplot(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], color = 'yellow', label = 'Cluster 1',s=50)
sns.scatterplot(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], color = 'blue', label = 'Cluster 2',s=50)
sns.scatterplot(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], color = 'green', label = 'Cluster 3',s=50)
sns.scatterplot(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color = 'red',
        label = 'Centroids',s=300,marker=',')

# labeling
# plt.grid(False)
plt.title('Two clusters')
plt.xlabel('Status_type')
plt.ylabel('values')
plt.legend()
plt.show()

#====================================END====================================
```