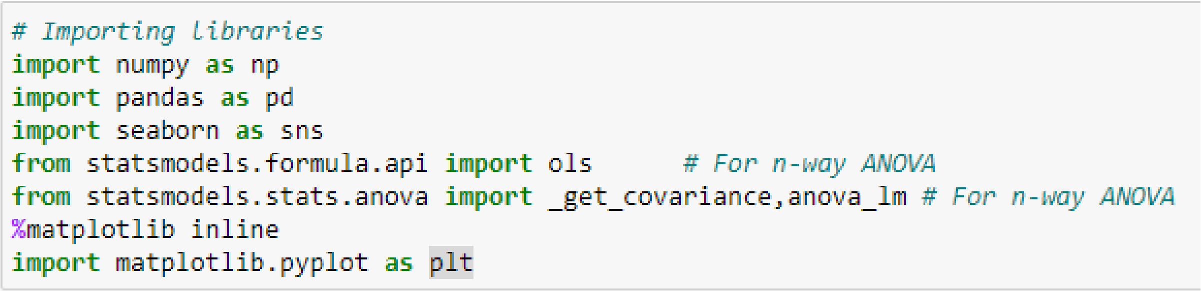
Problem 1:

A research laboratory was developing a new compound for the relief of severe cases of hay fever. In an experiment with 36 volunteers, the amounts of the two active ingredients (A & B) in the compound were varied at three levels each. Randomization was used in assigning four volunteers to each of the nine treatments. The data on hours of relief can be found in the following .csv file: **Fever-2.csv.**

[Assume all of the ANOVA assumptions are satisfied]

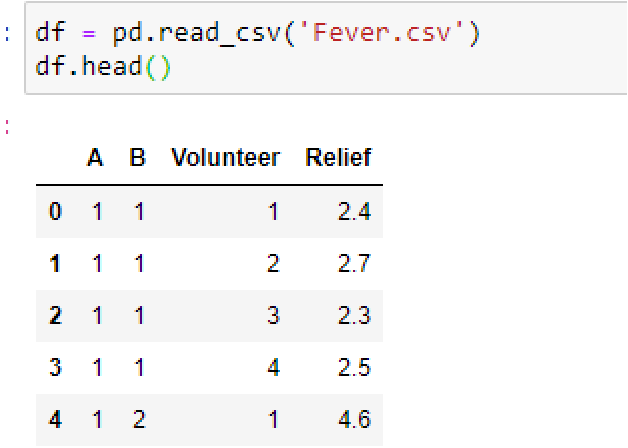
Solution: -

Loading the important libraries: -



```
# Importing libraries
import numpy as np
import pandas as pd
import seaborn as sns
from statsmodels.formula.api import ols      # For n-way ANOVA
from statsmodels.stats.anova import _get_covariance,anova_lm # For n-way ANOVA
%matplotlib inline
import matplotlib.pyplot as plt
```

Loading the dataset using df.head(): -

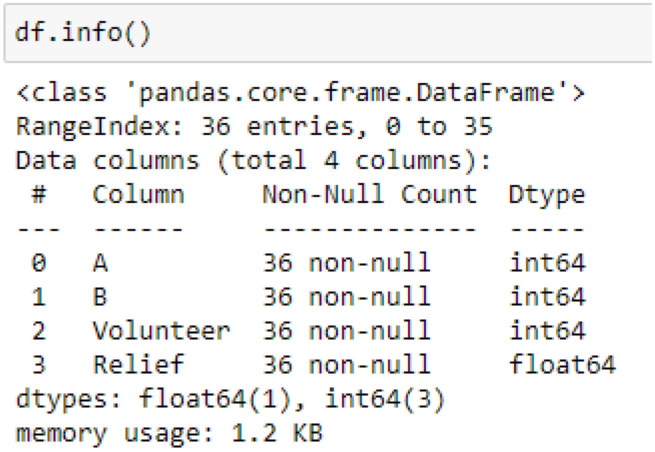


```
df = pd.read_csv('Fever.csv')
df.head()
```

	A	B	Volunteer	Relief
0	1	1	1	2.4
1	1	1	2	2.7
2	1	1	3	2.3
3	1	1	4	2.5
4	1	2	1	4.6

Observing the head, we can see there are 4 columns. A and B are the levels of ingredients, Relief describes the time take taken to get relieved, after consuming the medicine. Volunteer, although seems like a number, but actually is a categorical tag.

Checking basic info: df.info()



```
df.info()
```

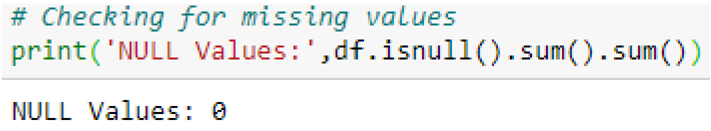
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   A          36 non-null     int64  
 1   B          36 non-null     int64  
 2   Volunteer  36 non-null     int64  
 3   Relief     36 non-null     float64
dtypes: float64(1), int64(3)
memory usage: 1.2 KB
```

As described in the info, all the variables are of integer or float datatype.

There are 36 Non-Null values in each column. The two columns A and B contain levels of ingredients used in the medicine. Relief variable is the target variable here.

Treatments performed on the relief variable to see how it varies, and what is the best possible combination available.

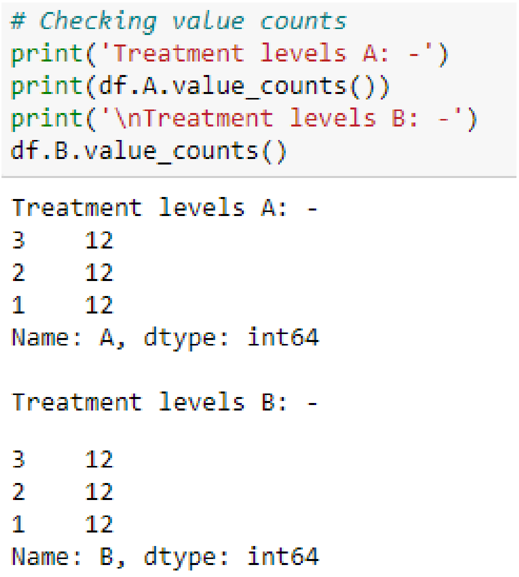
Confirming there are no null values: df.isnull().sum().sum()



```
# Checking for missing values
print('NULL Values:',df.isnull().sum().sum())
```

```
NULL Values: 0
```

Checking Sublevels in the two treatment variables A and B using value_counts()



```
# Checking value counts
print('Treatment levels A: -')
print(df.A.value_counts())
print('\nTreatment levels B: -')
df.B.value_counts()
```

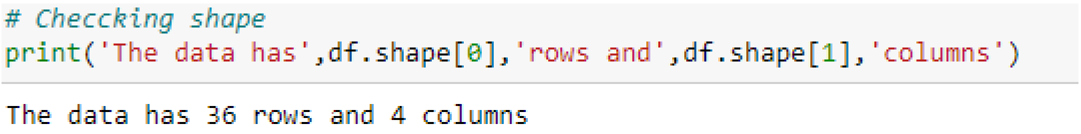
```
Treatment levels A: -
3    12
2    12
1    12
Name: A, dtype: int64

Treatment levels B: -

3    12
2    12
1    12
Name: B, dtype: int64
```

There are 3 sublevels in both A and B.

Checking shape using df.shape: -



```
# Checcking shape
print('The data has',df.shape[0],'rows and',df.shape[1],'columns')
```

```
The data has 36 rows and 4 columns
```

1.1 State the Null and Alternate Hypothesis for conducting one-way ANOVA for both the variables ‘A’ and ‘B’ individually. [both statement and statistical form like Ho=mu, Ha>mu]

Hypothesis for A: -

There are 3 treatment levels in A.

NULL Hypothesis: The average relief time due to all the three levels in A is same. Mathematically it can be stated as: **μ1=μ2=μ3, where μ stands for average relief time.**

ALTERNATIVE Hypothesis: The average relief time due to the three treatment levels is not same i.e. at least one pair of means is unequal. Mathematically it can be stated as**: μ1 ≠μ2≠ μ3, or μ1 =μ2≠ μ3 or μ1≠μ2=μ3**

Hypothesis for B: -

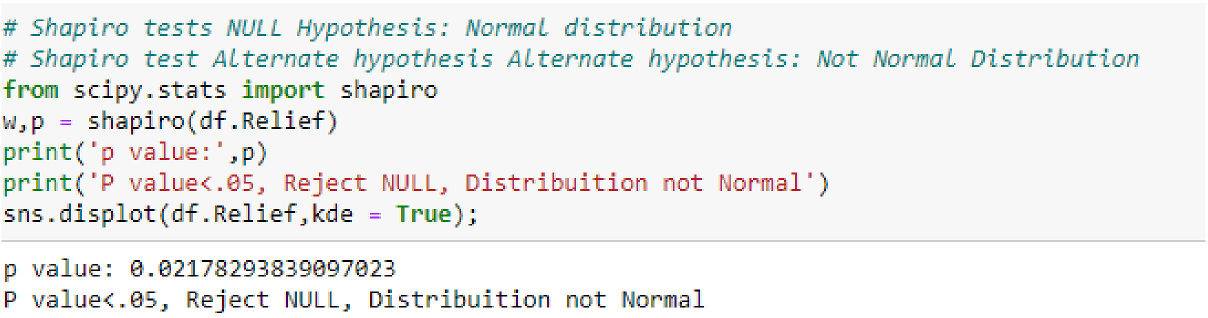
There are 3 treatment levels in B as well.

NULL Hypothesis: The average relief time due to all the three levels in B is same. Mathematically it can be stated as: **μ1=μ2=μ3**

ALTERNATIVE Hypothesis: The average relief time due to the three treatment levels is not same i.e. at least one pair of means is unequal. Mathematically it can be stated as**: μ1 ≠μ2≠ μ3, or μ1 =μ2≠ μ3 or μ1≠μ2=μ3**

1.2) Perform one-way ANOVA for variable ‘A’ with respect to the variable ‘Relief’. State whether the Null Hypothesis is accepted or rejected based on the ANOVA results.

Before moving to ANOVA, we need to ensure if the Dependent variable is Normally distributed. Although it was mentioned in the question to assume all the assumptions of ANOVA are met. The Normality can be confirmed using Shapiro test.



```
# Shapiro tests NULL Hypothesis: Normal distribution
# Shapiro test Alternate hypothesis Alternate hypothesis: Not Normal Distribution
from scipy.stats import shapiro
w,p = shapiro(df.Relief)
print('p value:',p)
print('P value<.05, Reject NULL, Distribuition not Normal')
sns.displot(df.Relief,kde = True);
```

```
p value: 0.02178293839097023
P value<.05, Reject NULL, Distribuition not Normal
```

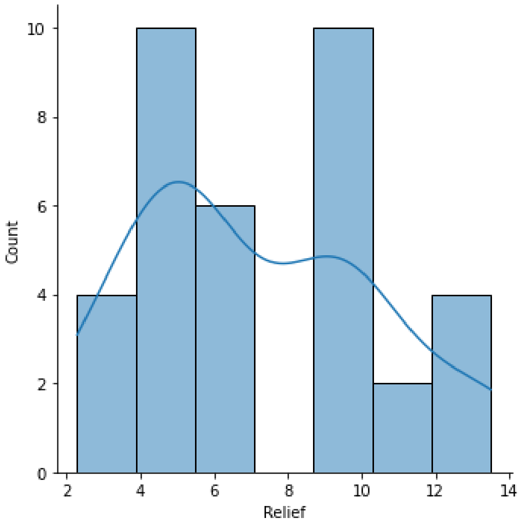
According to Shapiro test: -

The NULL states that distribution is Normal, the alternative states the opposite.

The P value is less than .05, hence we have to reject NULL and thus the distribution is not normal. But as mentioned in the problem statement, we can still go ahead.

Plotting a histogram with kde = True, will make it visually clear.

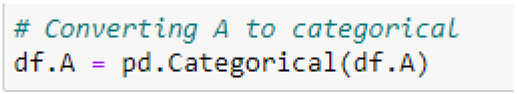
Using sns.displot(x=df.Relief, kde = True)



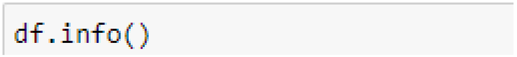
The histogram says it all. To improve the results of ANOVA, adding more data points, would eventually lead to the distribution becoming Normal. (Central limit theorem)

Moving ahead with one-way ANOVA (variable A and Relief)

The variable A is interpreted as int type, but we know, it's a level and is categorical in nature, hence we need to explicitly convert it into categorical. Using pd.Categorical() function.



```
# Converting A to categorical
df.A = pd.Categorical(df.A)
```



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   A          36 non-null     category
 1   B          36 non-null     int64
 2   Volunteer  36 non-null     int64
 3   Relief     36 non-null     float64
```

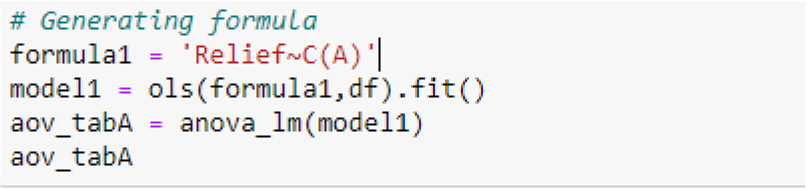
The column A is successfully converted to categorical now. We can move ahead with ANOVA.

The hypothesis assumed was: -

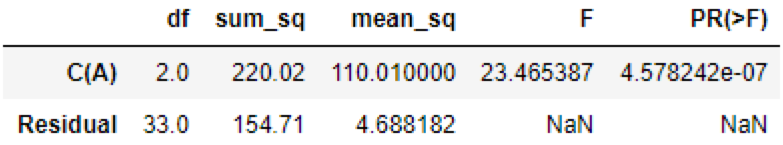
NULL: **μ1=μ2=μ3**

ALTERNATE: **μ1 ≠μ2≠ μ3, or μ1 =μ2≠ μ3 or μ1≠μ2=μ3 (at least one pair unequal)**

Generating formula: -



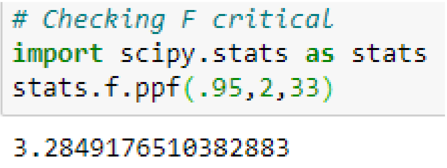
```
# Generating formula
formula1 = 'Relief~C(A)'
model1 = ols(formula1,df).fit()
aov_tabA = anova_lm(model1)
aov_tabA
```



	df	sum_sq	mean_sq	F	PR(>F)
C(A)	2.0	220.02	110.010000	23.465387	4.578242e-07
Residual	33.0	154.71	4.688182	NaN	NaN

The formula takes into account the effect of ingredient A over the relief variable. The ANOVA table obtained presents to us an F value and a P value.

Calculating F critical:



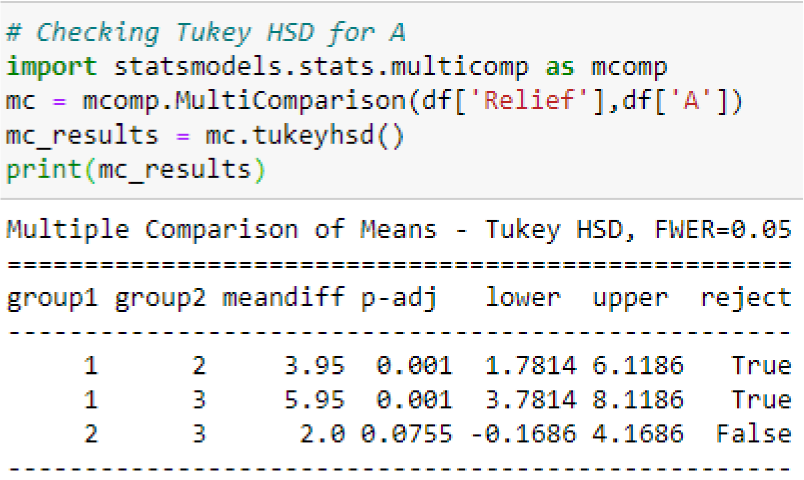
```
# Checking F critical
import scipy.stats as stats
stats.f.ppf(.95,2,33)
```

```
3.2849176510382883
```

Observing the table, F value obtained is much more than the F critical and the P value is less than .05. The P values almost tends to zero. Thus, we have clear evidence that ingredient A plays a significant role in the relief variable.

So far, we know that the relief time due to the three levels in A is not same for at least one pair, but we do not know yet which pair is unequal or of all the three are unequal.

To get an insight about the same, we need to perform Tukey HSD.



```
# Checking Tukey HSD for A
import statsmodels.stats.multicomp as mcomp
mc = mcomp.MultiComparison(df['Relief'],df['A'])
mc_results = mc.tukeyhsd()
print(mc_results)
```

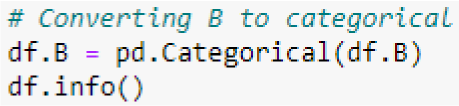
```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
group1 group2 meandiff p-adj   lower  upper  reject
----------------------------------------------------
     1      2     3.95  0.001  1.7814 6.1186   True
     1      3     5.95  0.001  3.7814 8.1186   True
     2      3      2.0 0.0755 -0.1686 4.1686  False
----------------------------------------------------
```

Looking at the table above we notice group 2 and group 3 has reject = False, whereas group 1,2 and group 1,3 have reject = True. This implies ingredient level 1,2 and 1,3 have unequal means for the target variable, thus rejecting NULL, while level 2 and 3 have equal means, hence failing to reject NULL.

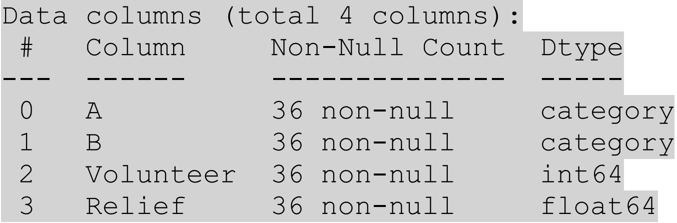
Mathematically it can be stated as: μ2=μ3, μ1≠μ2, μ1≠μ3, where μ is the mean relief time due to treatment level A.

1.3) Perform one-way ANOVA for variable ‘B’ with respect to the variable ‘Relief’. State whether the Null Hypothesis is accepted or rejected based on the ANOVA results.

Converting the column B to categorical: -

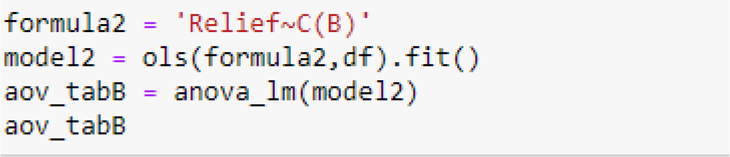


```
# Converting B to categorical
df.B = pd.Categorical(df.B)
df.info()
```

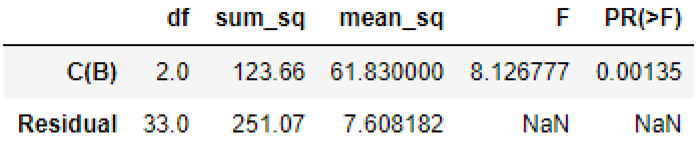


```
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   A          36 non-null     category
 1   B          36 non-null     category
 2   Volunteer  36 non-null     int64
 3   Relief     36 non-null     float64
```

The column B is categorical now.
Generating formula: -



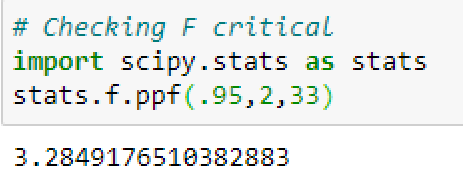
```
formula2 = 'Relief~C(B)'
model2 = ols(formula2,df).fit()
aov_tabB = anova_lm(model2)
aov_tabB
```



	df	sum_sq	mean_sq	F	PR(>F)
C(B)	2.0	123.66	61.830000	8.126777	0.00135
Residual	33.0	251.07	7.608182	NaN	NaN

The formula takes into account the effect of ingredient B over the relief variable. The ANOVA table obtained presents to us an F value and a P value.

Calculating F critical:



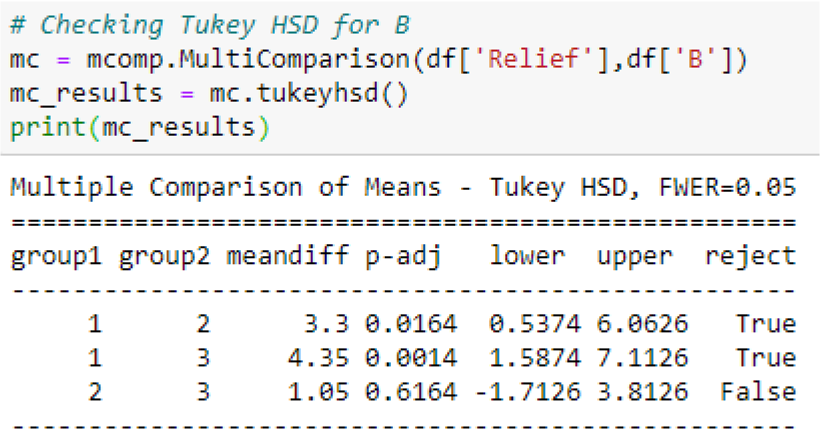
```
# Checking F critical
import scipy.stats as stats
stats.f.ppf(.95,2,33)
```

```
3.2849176510382883
```

Observing the table, F value obtained is more than the F critical and the P value is less than .05. The P values almost tends to zero. Thus, we have clear evidence that ingredient B also plays a significant role in the relief variable.

So far, we know that the relief time due to the three levels in B is not same for at least one pair, but we do not know yet which pair is unequal or of all the three are unequal.

To get an insight about the same, we need to perform Tukey HSD.



```
# Checking Tukey HSD for B
mc = mcomp.MultiComparison(df['Relief'],df['B'])
mc_results = mc.tukeyhsd()
print(mc_results)
```

```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================
group1 group2 meandiff p-adj   lower  upper  reject
---------------------------------------------------
     1      2      3.3 0.0164  0.5374 6.0626   True
     1      3     4.35 0.0014  1.5874 7.1126   True
     2      3     1.05 0.6164 -1.7126 3.8126  False
---------------------------------------------------
```

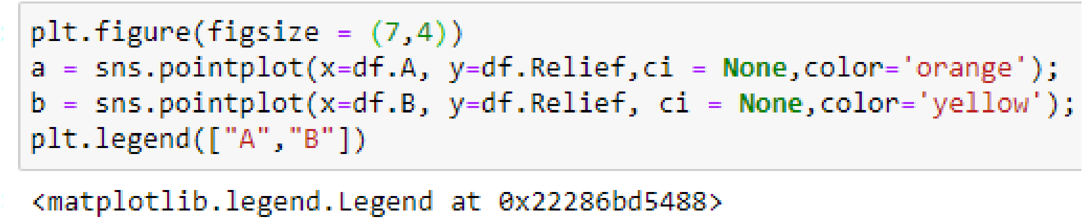
Looking at the table above we notice group 2 and group 3 has reject = False, whereas group 1,2 and group 1,3 have reject = True. This implies ingredient level 1,2 and 1,3 have unequal means for the target variable, thus rejecting NULL, while level 2 and 3 have equal means, hence failing to reject NULL.

Mathematically it can be stated as: μ2=μ3, μ1≠μ2, μ1≠μ3, where μ is the mean relief time due to treatment level B.

1.4) Analyse the effects of one variable on another with the help of an interaction plot. What is the interaction between the two treatments?

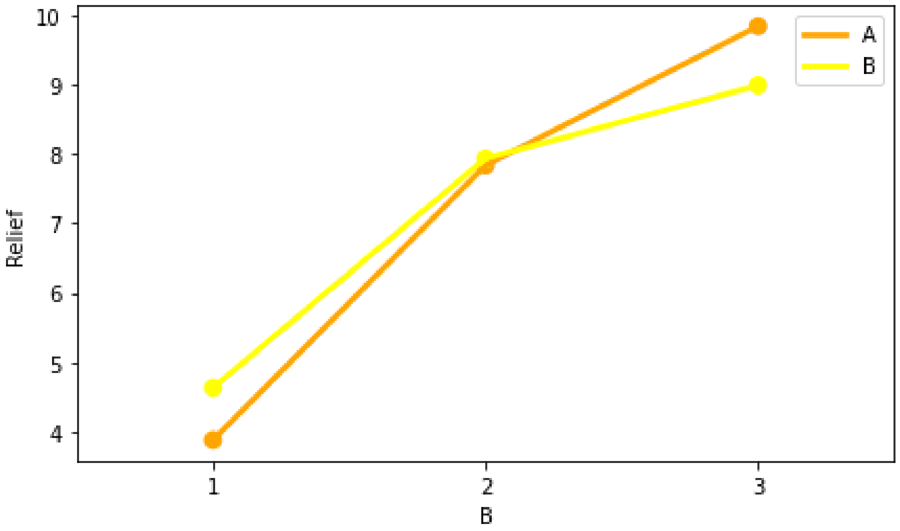
The interaction between the two can be found using point plots. The rule of the thumb being that, if at all there is interaction between the variables, the point plots would cross/intersect each other.

Plotting the same using seaborn library: sns.pointplot()



```
plt.figure(figsize = (7,4))
a = sns.pointplot(x=df.A, y=df.Relief,ci = None,color='orange');
b = sns.pointplot(x=df.B, y=df.Relief, ci = None,color='yellow');
plt.legend(["A","B"])
```

```
<matplotlib.legend.Legend at 0x22286bd5488>
```

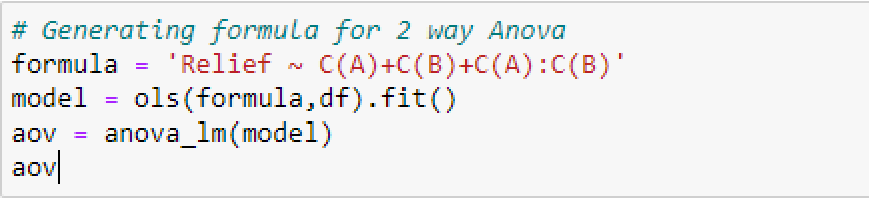


The plot says it all, the two-point plots intersect and hence we can say that there is significant interaction between the variables.

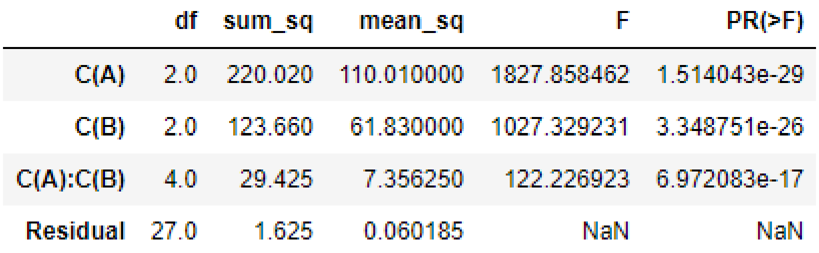
The interaction effect means that the dependent variable is affected by both the treatment levels and we need to incorporate this effect in our ANOVA model. Only after including this in the ANOVA model, we can determine the confidence level for this effect.

1.5) Perform a two-way ANOVA based on the different ingredients (variable ‘A’ & ‘B’ along with their interaction 'A*B') with the variable 'Relief' and state your results.

Generating the model formula two-way ANOVA: -



```
# Generating formula for 2 way Anova
formula = 'Relief ~ C(A)+C(B)+C(A):C(B)'
model = ols(formula,df).fit()
aov = anova_lm(model)
aov
```

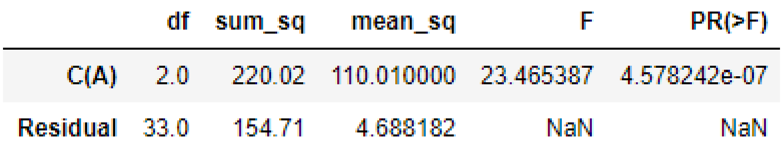


	df	sum_sq	mean_sq	F	PR(>F)
C(A)	2.0	220.020	110.010000	1827.858462	1.514043e-29
C(B)	2.0	123.660	61.830000	1027.329231	3.348751e-26
C(A):C(B)	4.0	29.425	7.356250	122.226923	6.972083e-17
Residual	27.0	1.625	0.060185	NaN	NaN

The above formula takes into account the effect of all the three factors on the relief variable i.e. A, B and their interaction.

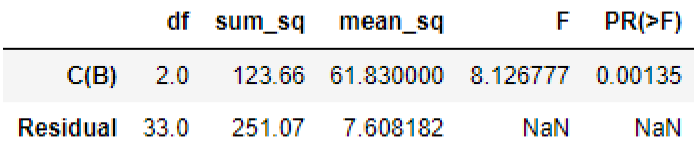
Let’s look how the P value has changed when all the three are taken together.

P value for one-way ANOVA with A: -



	df	sum_sq	mean_sq	F	PR(>F)
C(A)	2.0	220.02	110.010000	23.465387	4.578242e-07
Residual	33.0	154.71	4.688182	NaN	NaN

P value for one-way ANOVA with B: -



	df	sum_sq	mean_sq	F	PR(>F)
C(B)	2.0	123.66	61.830000	8.126777	0.00135
Residual	33.0	251.07	7.608182	NaN	NaN

It can be clearly noticed that how much the significance effect due to A and B have increased when we introduced interaction effect into the picture.

Now we can say with 99.99999……..% confidence level that both A, B and their interaction has a significance effect on the target variable relief.

1.6) Mention the business implications of performing ANOVA for this particular case study.

This case study involved experimentation for a particular drug to cure fever especially in terms of relief time. Lesser the relief time, better the drug. The experiment involves two ingredients A and B varied at 3 levels, 9 treatments in total. 36 volunteers were selected, namely 1,2,3,4. The volunteers were randomly assigned the drug treatment.

Looking at the business perspective we need to select the right combination of treatment level and decide which of the two treatment levels are really significant for relief time. For performing this analysis, we have ANOVA for the rescue.

Through ANOVA we can come to a conclusion are the ingredients and their sublevels really significant, also if they are, what happens when they interact. Specifically, we can also find, which sublevels in the two treatment levels affect the relief variable.

No doubt, we can find the same results as discussed above through experimentation, but experimentation is a longer and time taking process. If by accident a wrong combination is consumed by a patient, it may lead to disastrous results. Also, there is a limit to no of experiments that can be performed. Of course, the experiments need to be performed on a large scale and various stages, but only after the proper significant treatment combinations are known. The significant treatments are revealed through ANOVA.

To begin with, one-way ANOVA was performed on w.r.t. variable A. Looking at the P value (as it was less than .05) treatment level A was concluded to be a significant factor. Thus, we came to know for at-least one sublevel pair in A the mean value for relief value varied, but we did not know which sublevel was that at this point of time. To find the same Tukey HSD test was performed, and it was found that $\mu 2=\mu 3$, $\mu 1\neq\mu 2$, $\mu 1\neq\mu 3$. Thus, we know sublevels 1,2 and 1,3 are a significant combination.

Secondly, the same process was performed on B w.r.t. relief variable. The P value found was less than .05, hence NULL was rejected, and to know the significance of the sublevels, Tukey HSD was performed which yielded the following result: $\mu 2=\mu 3$, $\mu 1\neq\mu 2$, $\mu 1\neq\mu 3$. Thus, now we know sublevels 1,2 and 1,2 are significant combinations.

Continuing further, two-way ANOVA was performed w.r.t. relief variable, also considering the interaction affect. The inclusion of interaction effect, increased the confidence level of A and B even more, almost tending to 100. The interaction also proved out be significant.

The process is absolutely worth investing time and money as it is now known which are the significant treatment levels and sublevels. The main advantage is that now the company has a better idea about the ingredients and it can better design the cure. Also, now instead of random experimentation it can focus on particular sublevels.

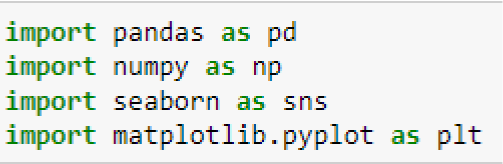
Thus, in my opinion, performing ANOVA in such circumstances should be the first priority to yield better informed results within less time and less money.

Problem 2:

The dataset **Education+-+Post+12th+Standard.csv** is a dataset that contains the names of various colleges. This particular case study is based on various parameters of various institutions. You are expected to do Principal Component Analysis for this case study according to the instructions given in the following rubric. The data dictionary of the 'Education - Post 12th Standard.csv' can be found in the following file: **Data Dictionary-3.xlsx**

Solution

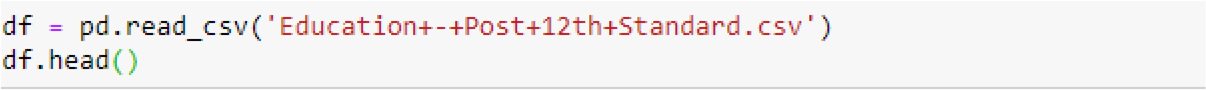
Loading the important libraries: -



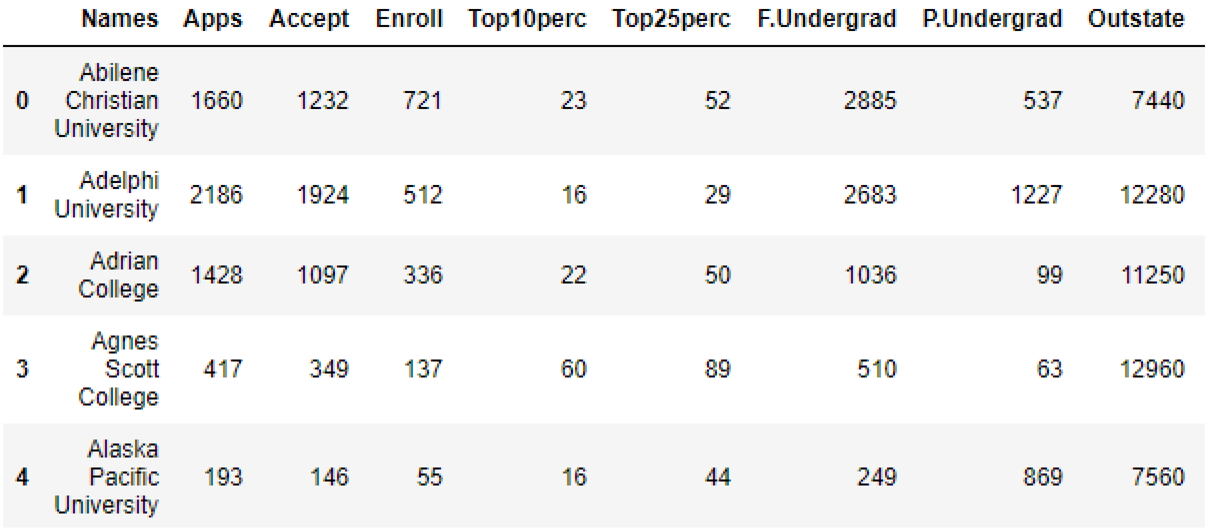
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Performing base EDA: -

Checking the head of the data: df.head()

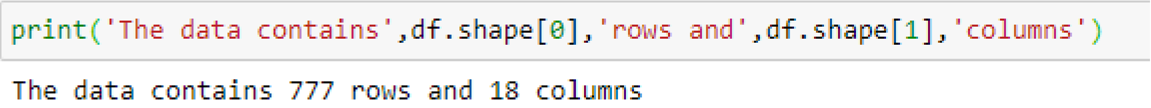


```
df = pd.read_csv('Education+-+Post+12th+Standard.csv')
df.head()
```



	Names	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate
0	Abilene Christian University	1660	1232	721	23	52	2885	537	7440
1	Adelphi University	2186	1924	512	16	29	2683	1227	12280
2	Adrian College	1428	1097	336	22	50	1036	99	11250
3	Agnes Scott College	417	349	137	60	89	510	63	12960
4	Alaska Pacific University	193	146	55	16	44	249	869	7560

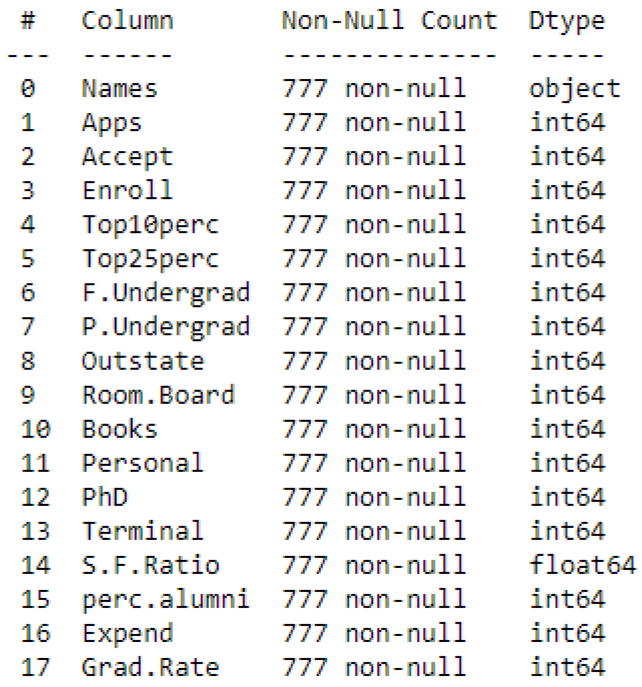
Checking the shape: df.shape



```
print('The data contains',df.shape[0],'rows and',df.shape[1],'columns')
```

```
The data contains 777 rows and 18 columns
```

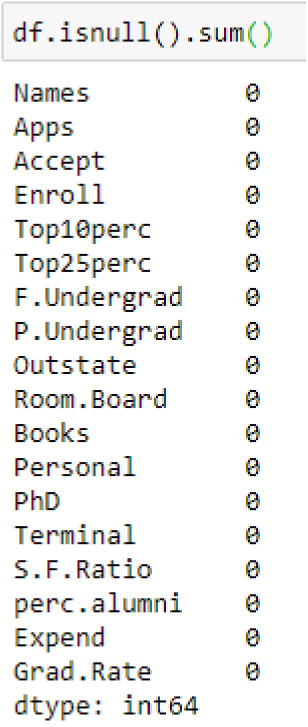
Checking basic information: df.info()



```
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Names        777 non-null    object 
 1   Apps         777 non-null    int64  
 2   Accept       777 non-null    int64  
 3   Enroll       777 non-null    int64  
 4   Top10perc    777 non-null    int64  
 5   Top25perc    777 non-null    int64  
 6   F.Undergrad  777 non-null    int64  
 7   P.Undergrad  777 non-null    int64  
 8   Outstate     777 non-null    int64  
 9   Room.Board   777 non-null    int64  
 10  Books        777 non-null    int64  
 11  Personal     777 non-null    int64  
 12  PhD          777 non-null    int64  
 13  Terminal     777 non-null    int64  
 14  S.F.Ratio    777 non-null    float64
 15  perc.alumni  777 non-null    int64  
 16  Expend       777 non-null    int64  
 17  Grad.Rate    777 non-null    int64  
```

As can be seen there are 18 columns, with 777 Non null entries each. The names column is of object type, rest are numeric (int64 or float64).

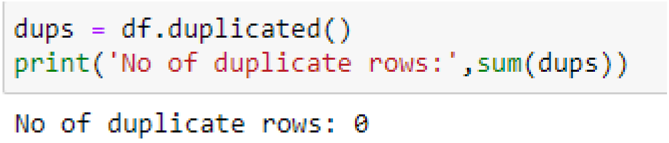
Checking for NULL values and duplicates: df.isnull().sum(), sum(df.duplicated())



```
df.isnull().sum()
```

```
Names          0
Apps           0
Accept         0
Enroll         0
Top10perc      0
Top25perc      0
F.Undergrad    0
P.Undergrad    0
Outstate       0
Room.Board     0
Books          0
Personal       0
PhD            0
Terminal       0
S.F.Ratio      0
perc.alumni    0
Expend         0
Grad.Rate      0
dtype: int64
```

The isnull() function confirms there are no outliers.

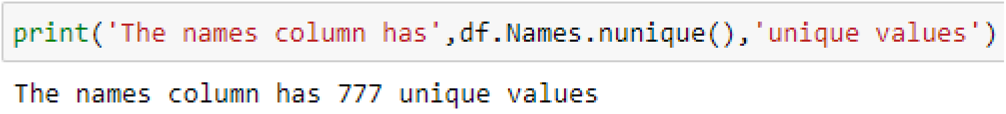


```
dups = df.duplicated()
print('No of duplicate rows:',sum(dups))

No of duplicate rows: 0
```

No of duplicate rows = 0.

No of unique values in the column Names: -

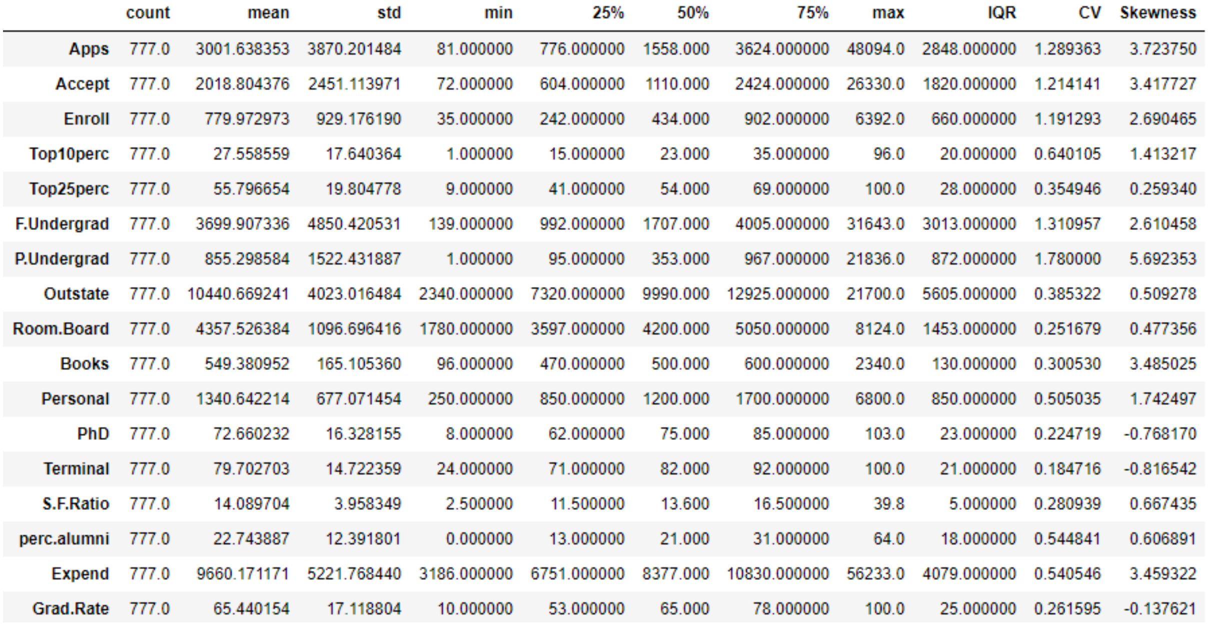


```
print('The names column has',df.Names.nunique(),'unique values')

The names column has 777 unique values
```

There are 777 unique values in the column names, i.e. there are 777 universities listed in names.

Checking for basic description: df.describe(). Additional columns IQR, CV and Skewness have been added.

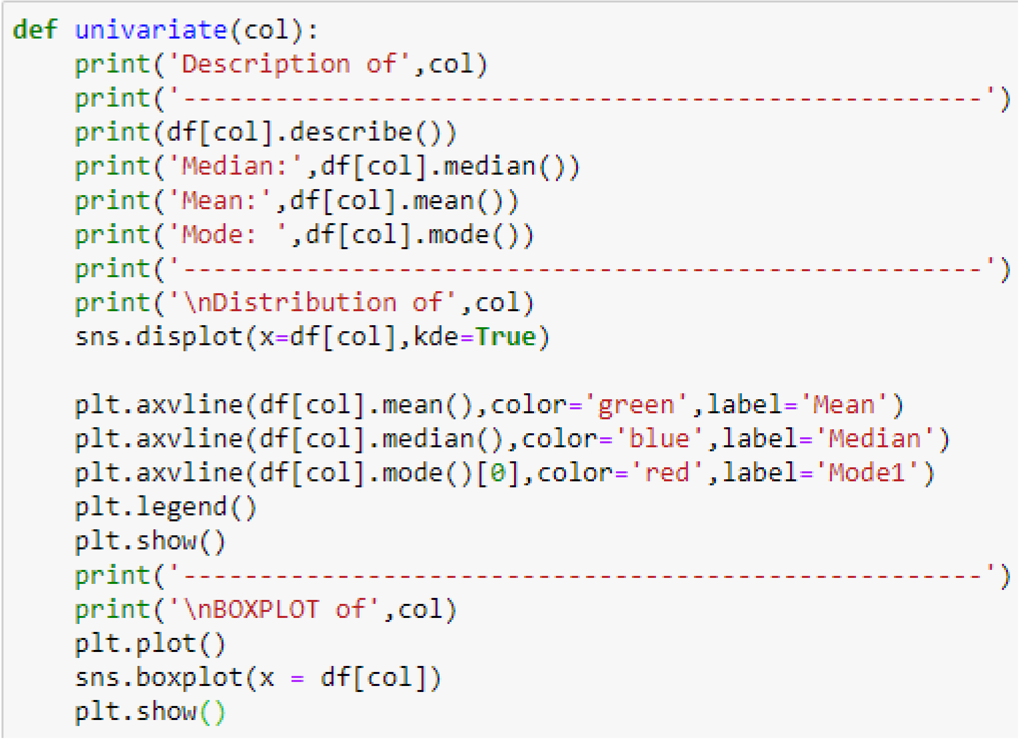


	count	mean	std	min	25%	50%	75%	max	IQR	CV	Skewness
Apps	777.0	3001.638353	3870.201484	81.000000	776.000000	1558.000	3624.000000	48094.0	2848.000000	1.289363	3.723750
Accept	777.0	2018.804376	2451.113971	72.000000	604.000000	1110.000	2424.000000	26330.0	1820.000000	1.214141	3.417727
Enroll	777.0	779.972973	929.176190	35.000000	242.000000	434.000	902.000000	6392.0	660.000000	1.191293	2.690465
Top10perc	777.0	27.558559	17.640364	1.000000	15.000000	23.000	35.000000	96.0	20.000000	0.640105	1.413217
Top25perc	777.0	55.796654	19.804778	9.000000	41.000000	54.000	69.000000	100.0	28.000000	0.354946	0.259340
F.Undergrad	777.0	3699.907336	4850.420531	139.000000	992.000000	1707.000	4005.000000	31643.0	3013.000000	1.310957	2.610458
P.Undergrad	777.0	855.298584	1522.431887	1.000000	95.000000	353.000	967.000000	21836.0	872.000000	1.780000	5.692353
Outstate	777.0	10440.669241	4023.016484	2340.000000	7320.000000	9990.000	12925.000000	21700.0	5605.000000	0.385322	0.509278
Room.Board	777.0	4357.526384	1096.696416	1780.000000	3597.000000	4200.000	5050.000000	8124.0	1453.000000	0.251679	0.477356
Books	777.0	549.380952	165.105360	96.000000	470.000000	500.000	600.000000	2340.0	130.000000	0.300530	3.485025
Personal	777.0	1340.642214	677.071454	250.000000	850.000000	1200.000	1700.000000	6800.0	850.000000	0.505035	1.742497
PhD	777.0	72.660232	16.328155	8.000000	62.000000	75.000	85.000000	103.0	23.000000	0.224719	-0.768170
Terminal	777.0	79.702703	14.722359	24.000000	71.000000	82.000	92.000000	100.0	21.000000	0.184716	-0.816542
S.F.Ratio	777.0	14.089704	3.958349	2.500000	11.500000	13.600	16.500000	39.8	5.000000	0.280939	0.667435
perc.alumni	777.0	22.743887	12.391801	0.000000	13.000000	21.000	31.000000	64.0	18.000000	0.544841	0.606891
Expend	777.0	9660.171171	5221.768440	3186.000000	6751.000000	8377.000	10830.000000	56233.0	4079.000000	0.540546	3.459322
Grad.Rate	777.0	65.440154	17.118804	10.000000	53.000000	65.000	78.000000	100.0	25.000000	0.261595	-0.137621

Observing the dataset, there seem to be some problem with Grad Rate and PhD. Both columns are in percentage, and it can’t be greater than 100%. The max Grad Rare is 118, and max PhD is 103, which is practically impossible.

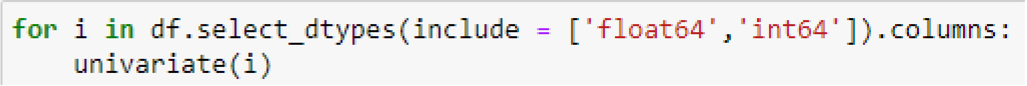
2.1) Perform Exploratory Data Analysis [both univariate and multivariate analysis to be performed]. The inferences drawn from this should be properly documented.

The dataset contains 18 columns, to perform univariate analysis for every variable is not recommended. Instead a function can be used and all the variables can be passed through a loop, generating results in one go.



```
def univariate(col):
    print('Description of',col)
    print('----------------------------------------------------')
    print(df[col].describe())
    print('Median:',df[col].median())
    print('Mean:',df[col].mean())
    print('Mode: ',df[col].mode())
    print('----------------------------------------------------')
    print('\nDistribution of',col)
    sns.displot(x=df[col],kde=True)

    plt.axvline(df[col].mean(),color='green',label='Mean')
    plt.axvline(df[col].median(),color='blue',label='Median')
    plt.axvline(df[col].mode()[0],color='red',label='Mode1')
    plt.legend()
    plt.show()
    print('----------------------------------------------------')
    print('\nBOXPLOT of',col)
    plt.plot()
    sns.boxplot(x = df[col])
    plt.show()
```



```
for i in df.select_dtypes(include = ['float64','int64']).columns:
    univariate(i)
```

The function inputs a specific column and displays the description, histogram and boxplot for the same, with mean median and 1st mode marked on the histogram. The float and int columns are passed into the function through a loop.

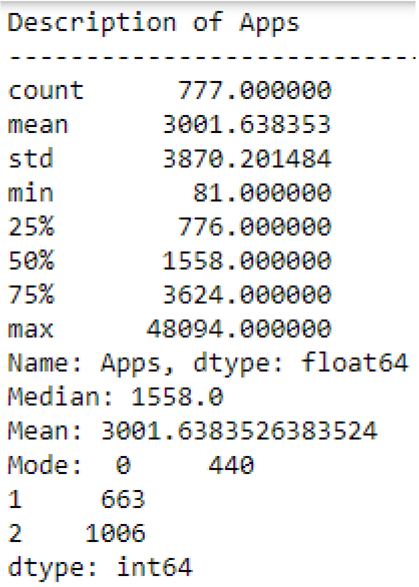
The univariate analysis of each variable can be found in the python file.

The significant variables with univariate analysis are listed below and errors if any are pointed out.

UNIVARIATE ANALYSIS: -

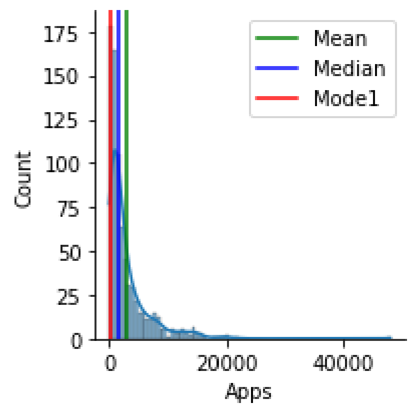
Major Insights about the univariate analysis: -

1. Apps: No of applications received.

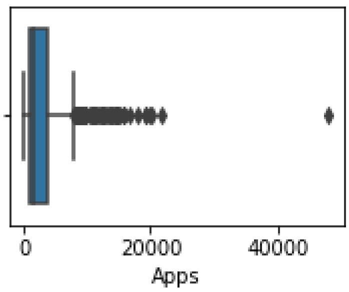


```
Description of Apps
----------------------------
count       777.000000
mean       3001.638353
std        3870.201484
min          81.000000
25%         776.000000
50%        1558.000000
75%        3624.000000
max       48094.000000
Name: Apps, dtype: float64
Median: 1558.0
Mean: 3001.6383526383524
Mode:  0     440
1     663
2    1006
dtype: int64
```

Distribution of Apps

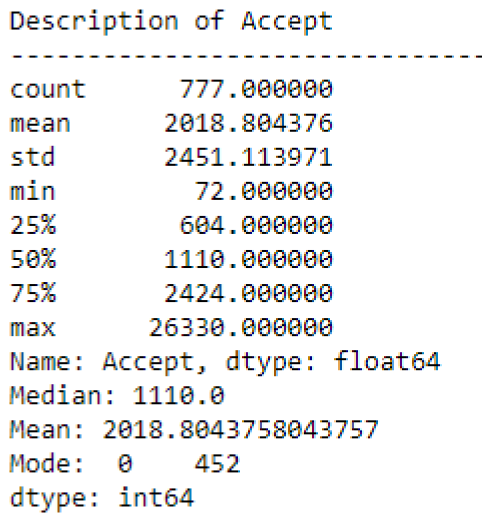


BOXPLOT of Apps

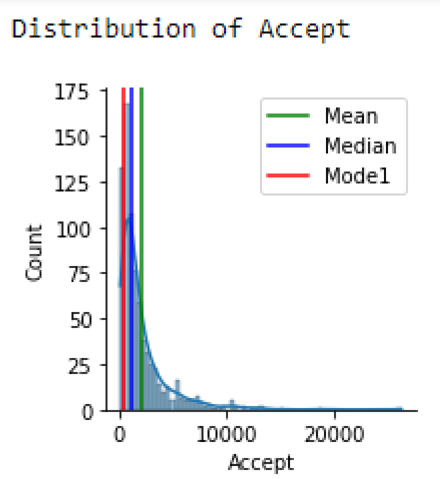


The distribution is Right skewed with a high skewness = 3.73. This implies Mean>Median>Mode. Standard deviation>Mean leading to high CV (1.28). This suggests a high inconsistency. The variable is multimodal. Also, significant gap between min and max.

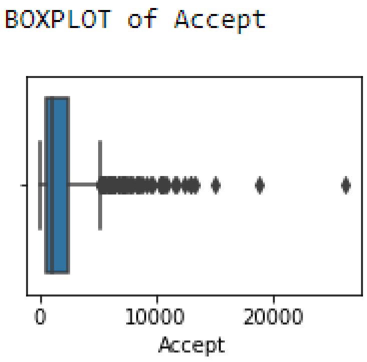
2. Accept: No of applications received.



```
Description of Accept
-------------------------------
count      777.000000
mean      2018.804376
std       2451.113971
min         72.000000
25%        604.000000
50%       1110.000000
75%       2424.000000
max      26330.000000
Name: Accept, dtype: float64
Median: 1110.0
Mean: 2018.8043758043757
Mode:  0    452
dtype: int64
```



Distribution of Accept

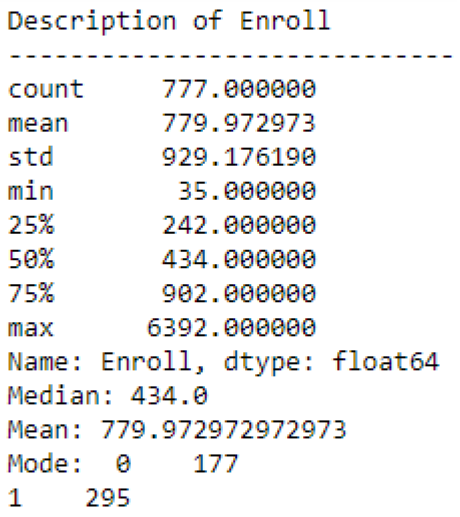


BOXPLOT of Accept

The distribution is right skewed with Skewness = 3.41. Mean>Median>Mode. There are outliers present. The std>mean implying high CV. CV = 1.21,implying high inconsistency. The min and max are way apart.

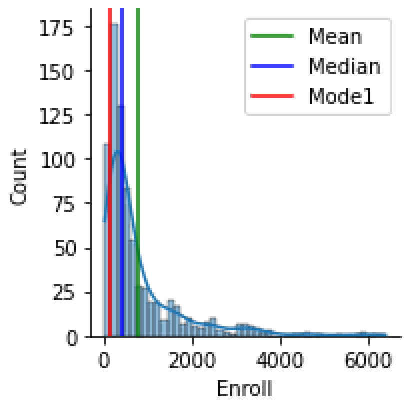
The overall acceptance seems pretty good. With median applications of 1558, the median accepted are1110, thus accepting a majority of students.

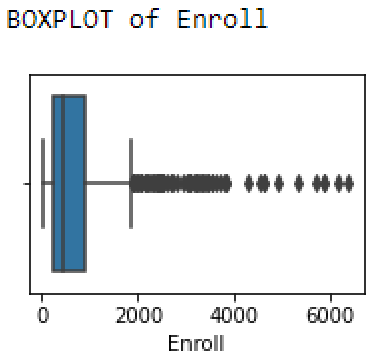
3. Enrol: No of new students enrolled



```
Description of Enroll
-----------------------------
count     777.000000
mean      779.972973
std       929.176190
min        35.000000
25%       242.000000
50%       434.000000
75%       902.000000
max      6392.000000
Name: Enroll, dtype: float64
Median: 434.0
Mean: 779.972972972973
Mode:  0    177
1    295
```

Distribution of Enroll

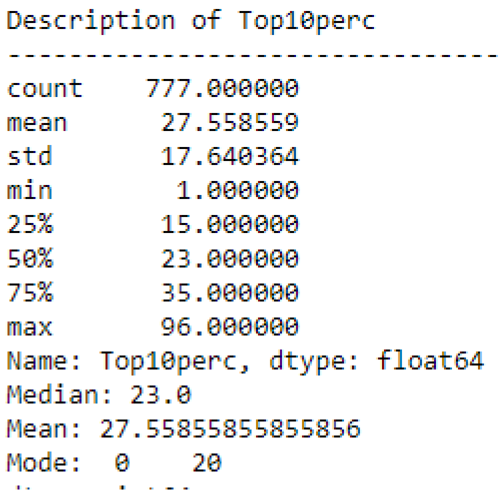




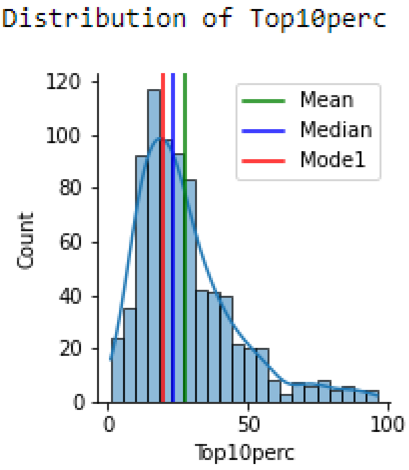
Right skewed distribution. Skewness = 2.69. Mean>Median>Mode. Std dev > Mean leading to high CV (1.19). Inconsistent distribution. Data is Multimodal. Enrollment ranges from 35 to 6392 with median 434. The two modes are 177 and 295. Boxplots confirms outliers.

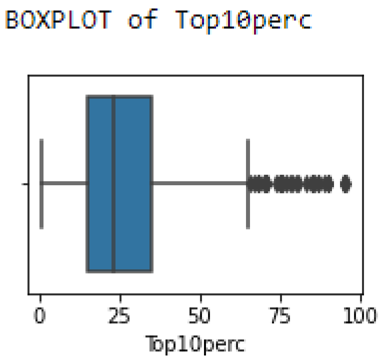
The median enrolment = 434 which is quite less compared to the median accepted.

4. Top 10 perc: Percentage of students from top 10 percentage of higher sec. class



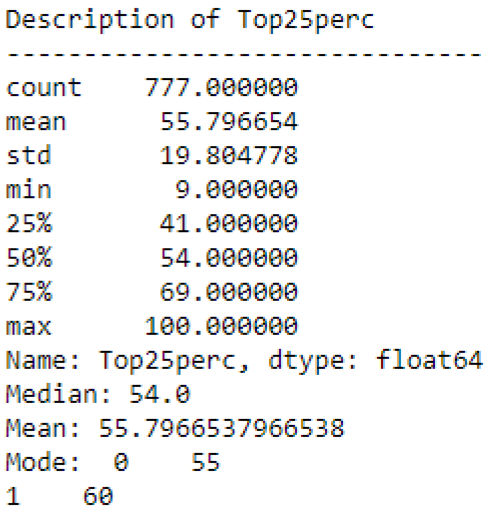
```
Description of Top10perc
--------------------------------
count    777.000000
mean      27.558559
std       17.640364
min        1.000000
25%       15.000000
50%       23.000000
75%       35.000000
max       96.000000
Name: Top10perc, dtype: float64
Median: 23.0
Mean: 27.55855855855856
Mode:   0    20
```



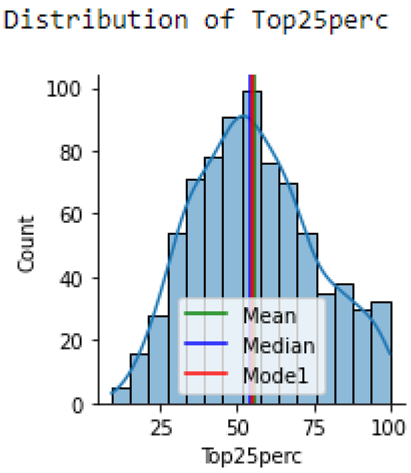


The distribution is right skewed, with skewness = 1.41. Altough, the data is right skewed, the mean and median are close apart. Std dev<Mean, hence CV<1. CV = .64 pointing towards a better consistency. There are many outliers present in the data.

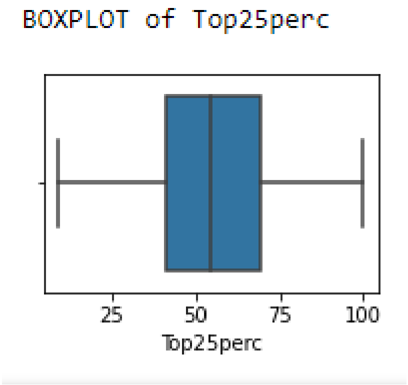
5. Top 25 Perc: Percentage of students from the top 25 percent of highher sec. class



```
Description of Top25perc
-------------------------------
count    777.000000
mean      55.796654
std       19.804778
min        9.000000
25%       41.000000
50%       54.000000
75%       69.000000
max      100.000000
Name: Top25perc, dtype: float64
Median: 54.0
Mean: 55.7966537966538
Mode:  0    55
1    60
```



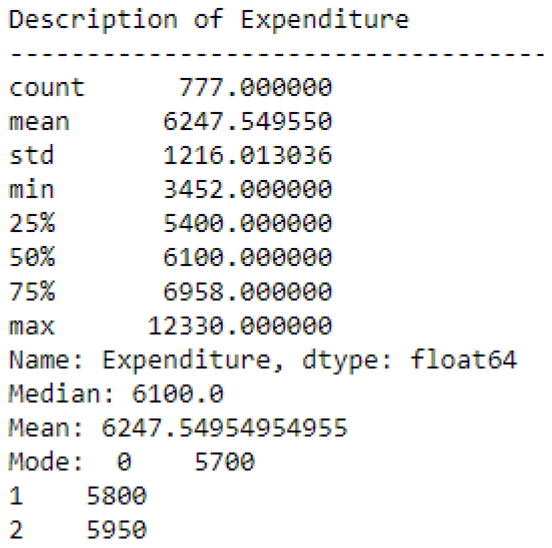
Distribution of Top25perc



BOXPLOT of Top25perc

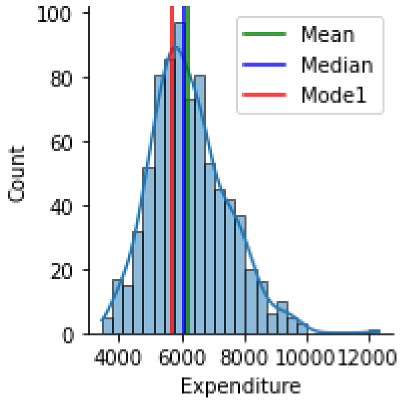
The distribution appears to be Normal. Skewness = .25, stating slightly positive skew. The mean,median and mode almost overlap. There are two modes. The COV .35, stating consistency, and there are no outliers, possibly the only column with no outliers.

6. Expenditure: This columns has been explicitly added which includes Room, Books and Personal expenses.

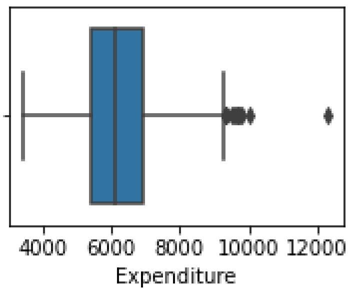


```
Description of Expenditure
-----------------------------------
count       777.000000
mean       6247.549550
std        1216.013036
min        3452.000000
25%        5400.000000
50%        6100.000000
75%        6958.000000
max       12330.000000
Name: Expenditure, dtype: float64
Median: 6100.0
Mean: 6247.54954954955
Mode:  0    5700
1    5800
2    5950
```

Distribution of Expenditure



BOXPLOT of Expenditure

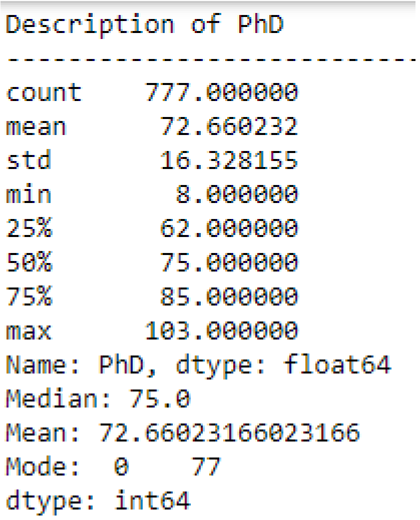


The data seems to be distributed Normally. Skewness = .56, a little bit right skewed. The mean and median are close apart. The min and max are not very far away. The min being 3452, and the max being 12330.

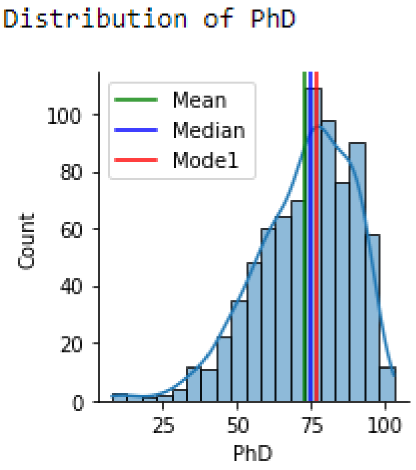
The std dev is less than mean, hence we get a less CV of .19, pointing a significant consistency.

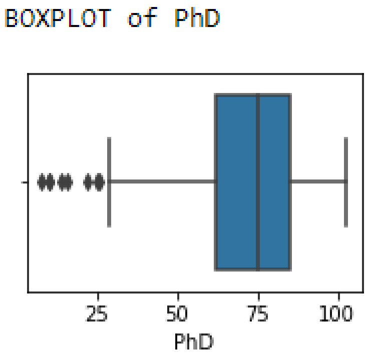
There are a few outliers present in the data.

7. PhD: Percentage of Faculties with PhD.



```
Description of PhD
---------------------------
count     777.000000
mean       72.660232
std        16.328155
min         8.000000
25%        62.000000
50%        75.000000
75%        85.000000
max       103.000000
Name: PhD, dtype: float64
Median: 75.0
Mean: 72.66023166023166
Mode:  0    77
dtype: int64
```

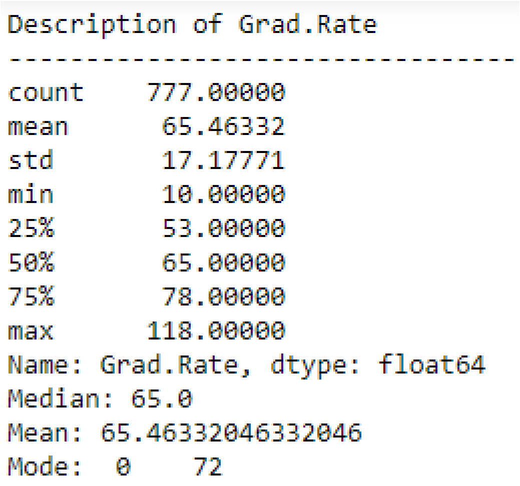




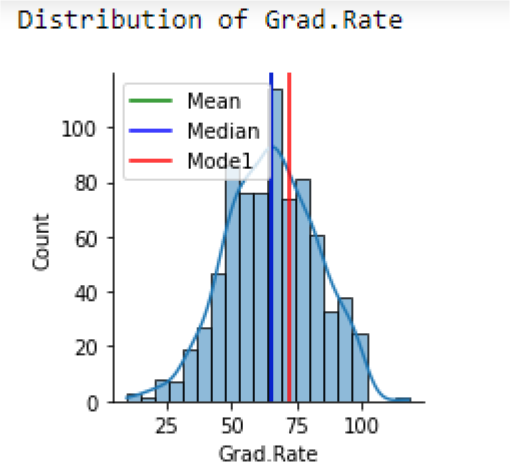
The distribution of PhD is left skewed. The skewness value = -.76. The std<mean and the CV = .22, indicating significant consistency. There are indeed outliers in the data.

Looking at the data carefully, max value is 103, which is anamolous. Percentage can't be greater than 100 in this case. This needs to be treated.

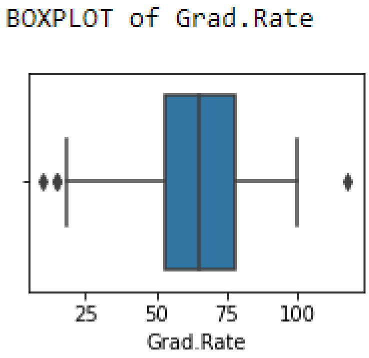
8. Grad Rate: Graduation Rate



```
Description of Grad.Rate
---------------------------------
count     777.00000
mean       65.46332
std        17.17771
min        10.00000
25%        53.00000
50%        65.00000
75%        78.00000
max       118.00000
Name: Grad.Rate, dtype: float64
Median: 65.0
Mean: 65.46332046332046
Mode:  0    72
```



Distribution of Grad.Rate

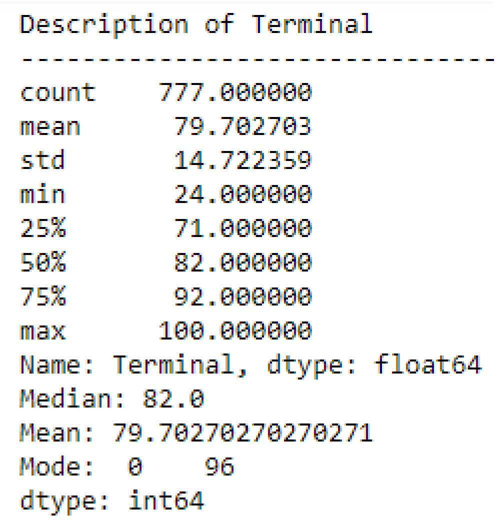


BOXPLOT of Grad.Rate

The distribution is left skewed, skewness = -.13. Mean and Median are same, indicating an almost normal distrubution. However, the mode>mean and median. The std deviation is 17.11 which is quite less compared to the mean, hence CV = .26, implying well consistent distrubution.

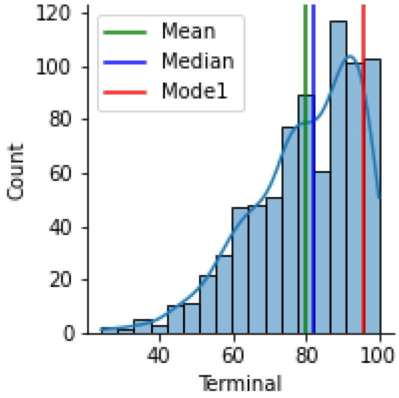
One error we can see here is the max value. The max value is 118, which is not practically possible, as no of graduates passing can't be more than the ones enrolled.

9. Terminal Degree: Percentage of faculties with students with terminal degrees

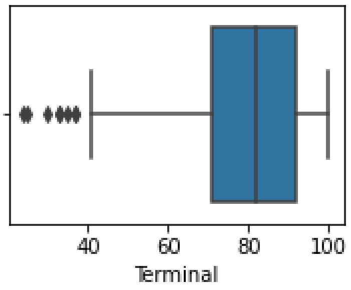


```
Description of Terminal
-------------------------------
count    777.000000
mean      79.702703
std       14.722359
min       24.000000
25%       71.000000
50%       82.000000
75%       92.000000
max      100.000000
Name: Terminal, dtype: float64
Median: 82.0
Mean: 79.70270270270271
Mode:  0    96
dtype: int64
```

Distribution of Terminal



BOXPLOT of Terminal

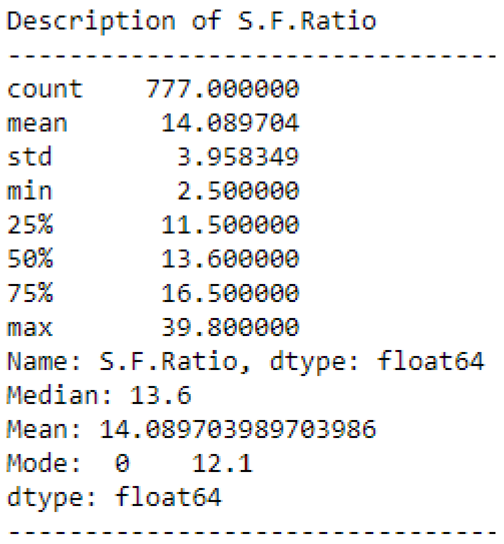


The distribution is skewed to the left, with a skewness level of -.81. Mean and Median are close apart, but Median>Mean.

The mode is 96% signifying that in max universities 96% of students reach terminal degree, which is indeed a positive sign.

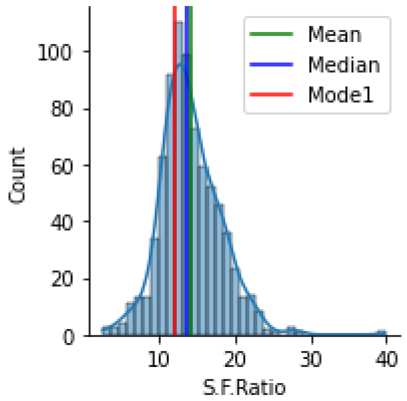
The CV=.18, indicating consistency.

10. S.F. Ratio: Student Faculty Ration

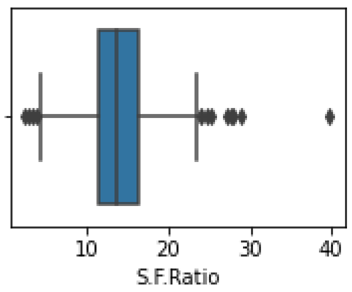


```
Description of S.F.Ratio
--------------------------------
count    777.000000
mean      14.089704
std        3.958349
min        2.500000
25%       11.500000
50%       13.600000
75%       16.500000
max       39.800000
Name: S.F.Ratio, dtype: float64
Median: 13.6
Mean: 14.089703989703986
Mode:  0    12.1
dtype: float64
--------------------------------
```

Distribution of S.F.Ratio



BOXPLOT of S.F.Ratio



The distribution seems normal, with mean and median very close apart. However the skewness = .26, indicating a bit right skewness.

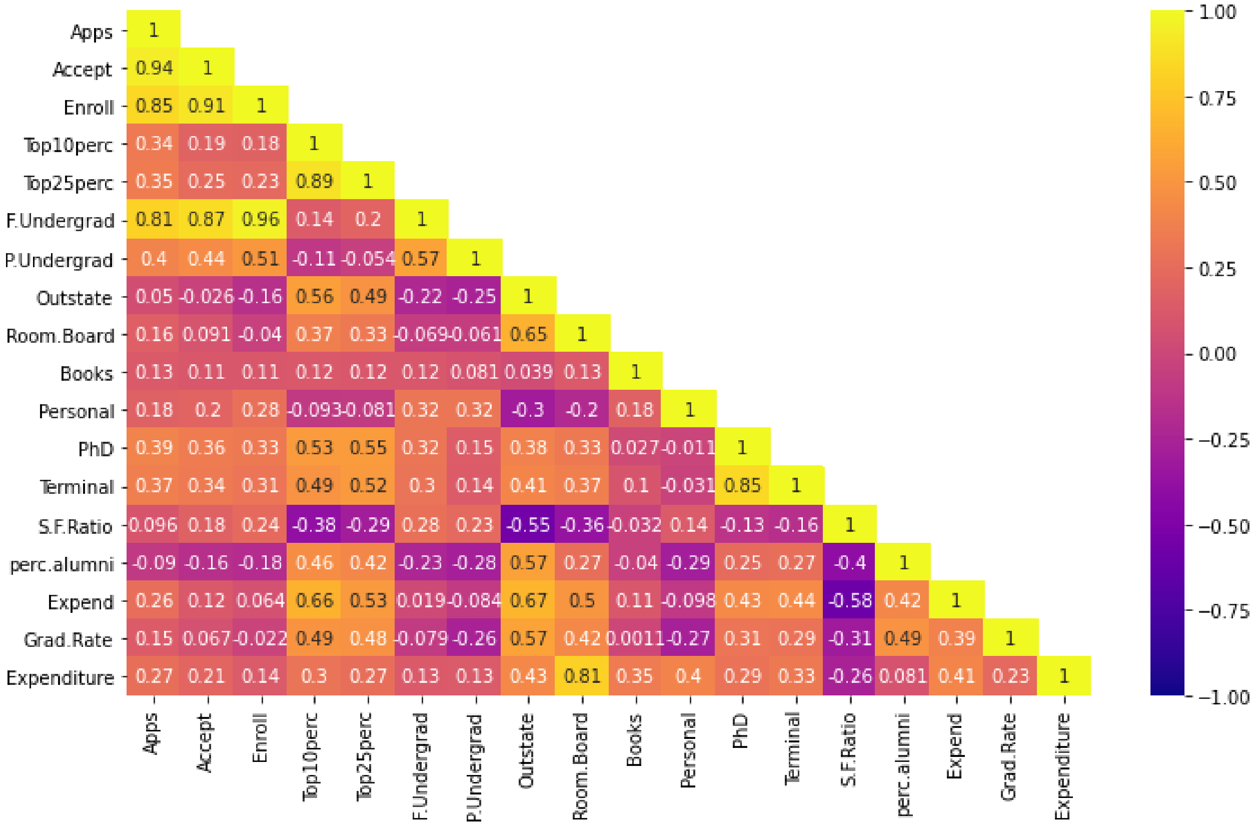
The std deviation is very less, .and thus the CV, CV = .28. Consistency can be very well scene.

Boxplot suggests presence of outliers.

MULTIVARIATE ANALYSIS

To get an idea about the multivariate analysis, a pair plot or a heat map can be a good option. But keeping in mind the no of variables, the heatmap could help us better identify the strengths between the variables.

Using sns.heatmap()



The heatmap suggests strong relationships between the following variables: -

1. Accept and Apps : .94

2. Enroll and Apps : .85

3. Enroll and Accept: .91

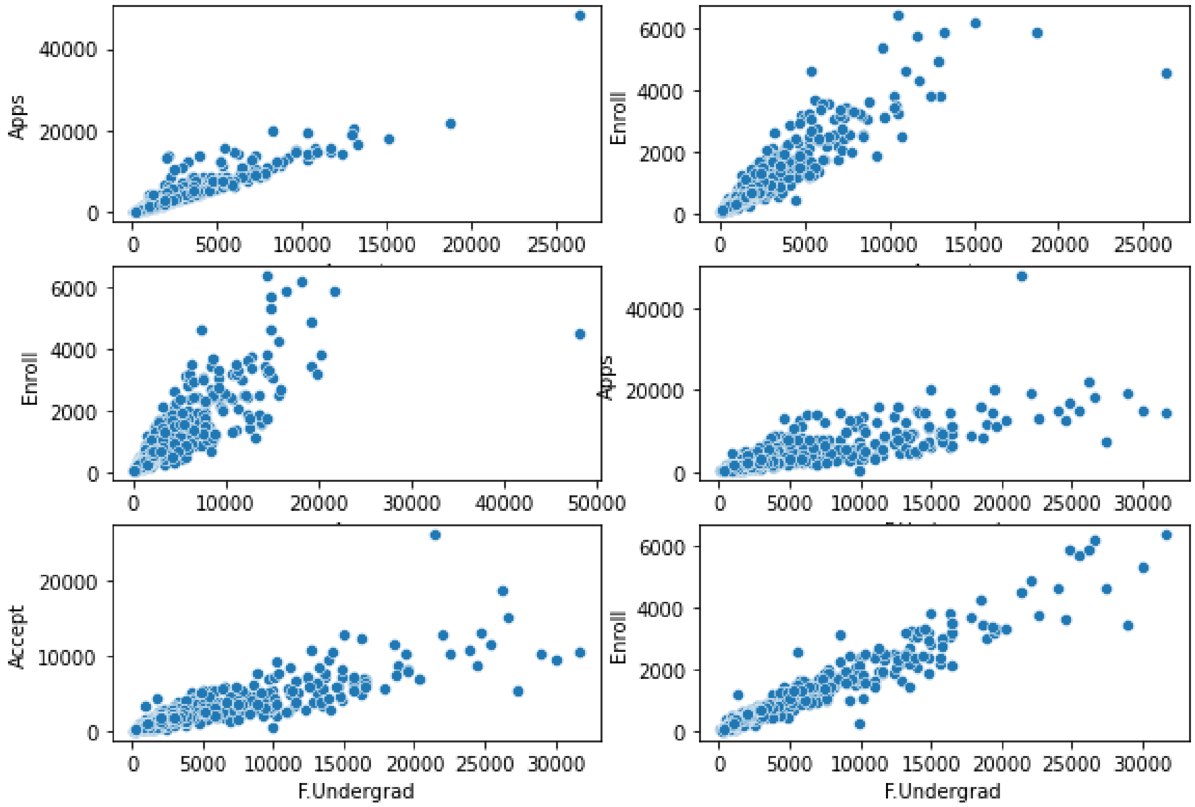
4. F.Undergrad and Apps: .81

5. F.Undergrad and Accept: .87

6. F.Undergrad and Enroll: .96

We need to keep in mind these are nor cause-effect relationships, rather just how the strongly the variables evolve w.r.t. each other. Further, we can visualize them using scatterplot.

Using sns.scatterplot()



The scatterplots reveal how strongly they move with each other, in terms of direction and strength.

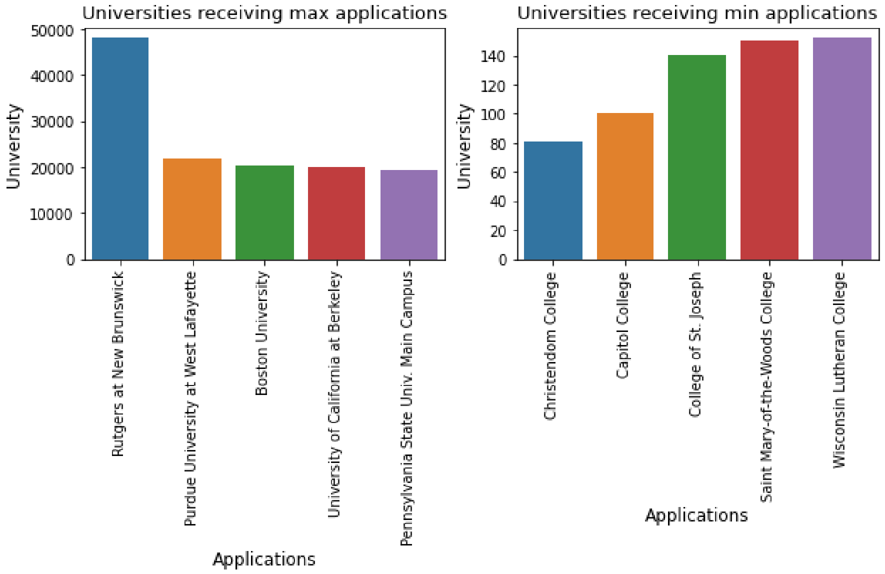
BIVARIATE ANALYSIS

Digging further, we can extract more information w.r.t. the univaersities and the variables listed, that would give some meaningful insights.

We can find the following information: -

1. Application trend among universities
2. Acceptance rate
3. Observing the students from top 10 percent of the high. Sec. class
4. Expenditure trend
5. Universities with max PhD faculties
6. Graduation rate
7. Student Faculty ratio
8. Terminal degree
9. Instructional Expenditure

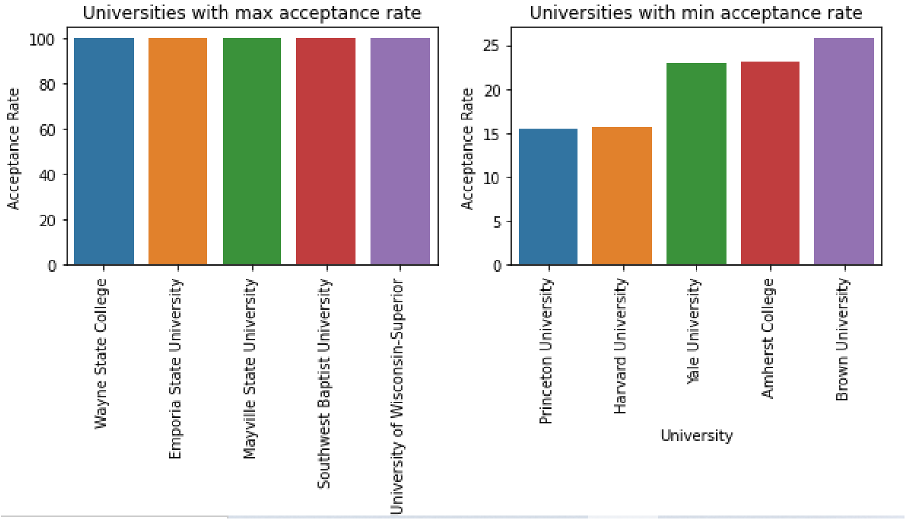
1. Application trend



University of Rutgers receives the max applications, while Christendom receives the least

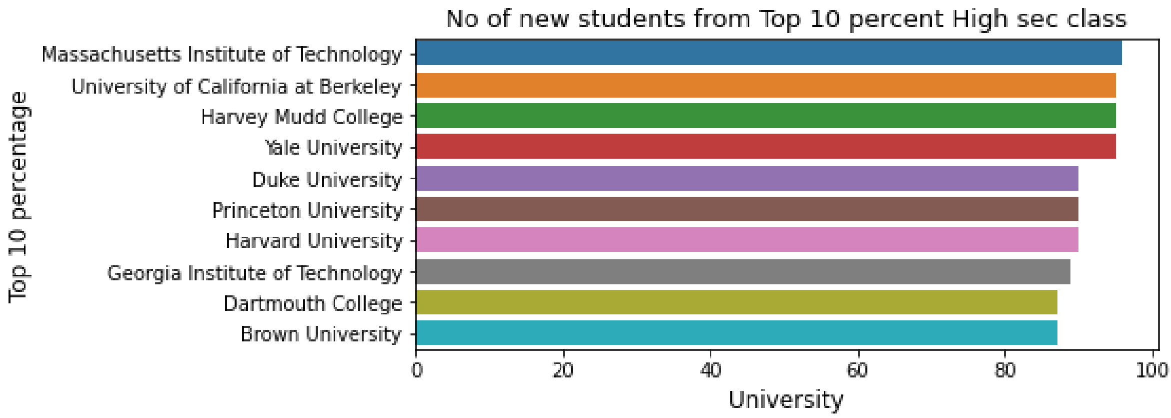
2. Acceptance Rate

Although not mentioned in the data set it can be calculated as: Accepted/Applied.



There are several universities with 100% acceptance rate, however the once with least acceptance rate are ofcourse the Ivy league institutions.

3. Observing the students from top 10 percent of the high. Sec. class

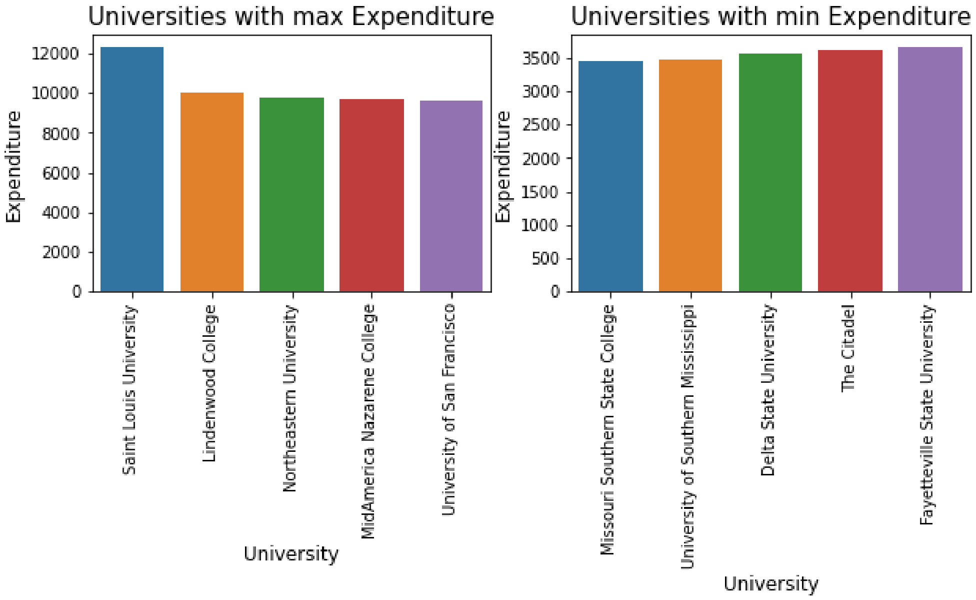


The results are of no surprise. Absolutely max no of new students belonging from the top 10 percent of the institutes ought to go to MIT, UCL, Harvard, Duje, Yale, Princeton and Brown.

These students comprise of the creamy layer.

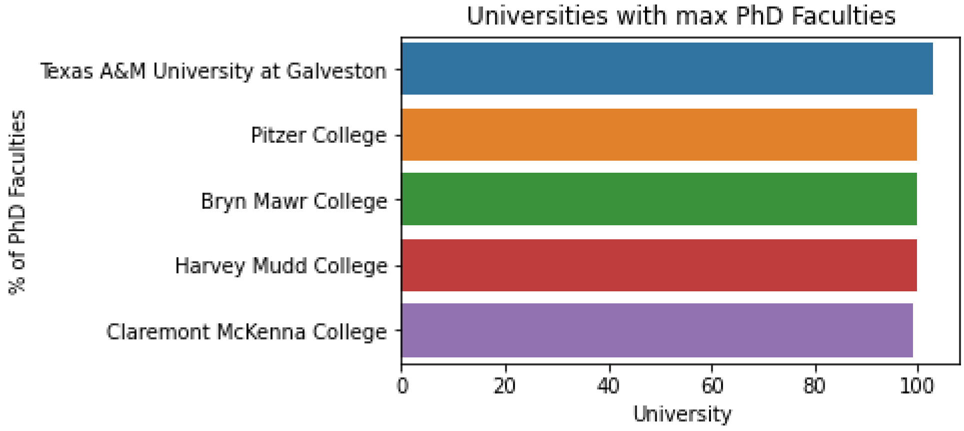
4. Expenditure trend

The total expenditure includes the cost of room, books and personal expenditure.



There are a wide range of universities available in terms of cost. Ranging from a min expenditure of less than 3500 to a whooping expenditure of 12000.

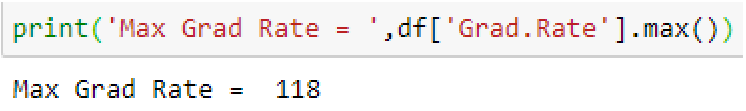
5. PhD Faculties



There are many universities with more than 90% of faculties with PhD, but Texas A&M tops the list with all the faculties as PhD.

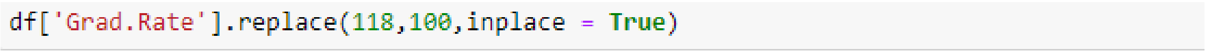
6. Graduation Rate

As discussed earlier, Grad Rate was greater than 100. Treating the same: -

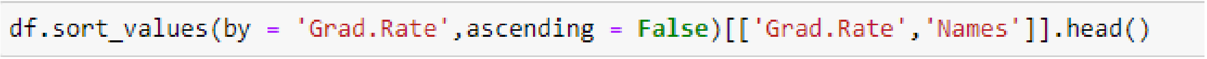


```
print('Max Grad Rate = ',df['Grad.Rate'].max())
```

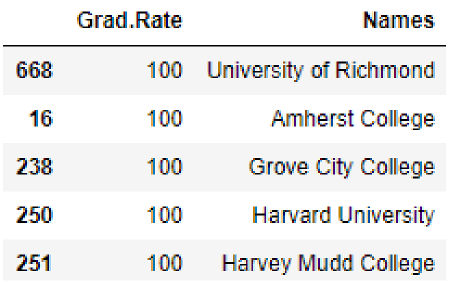
```
Max Grad Rate =  118
```



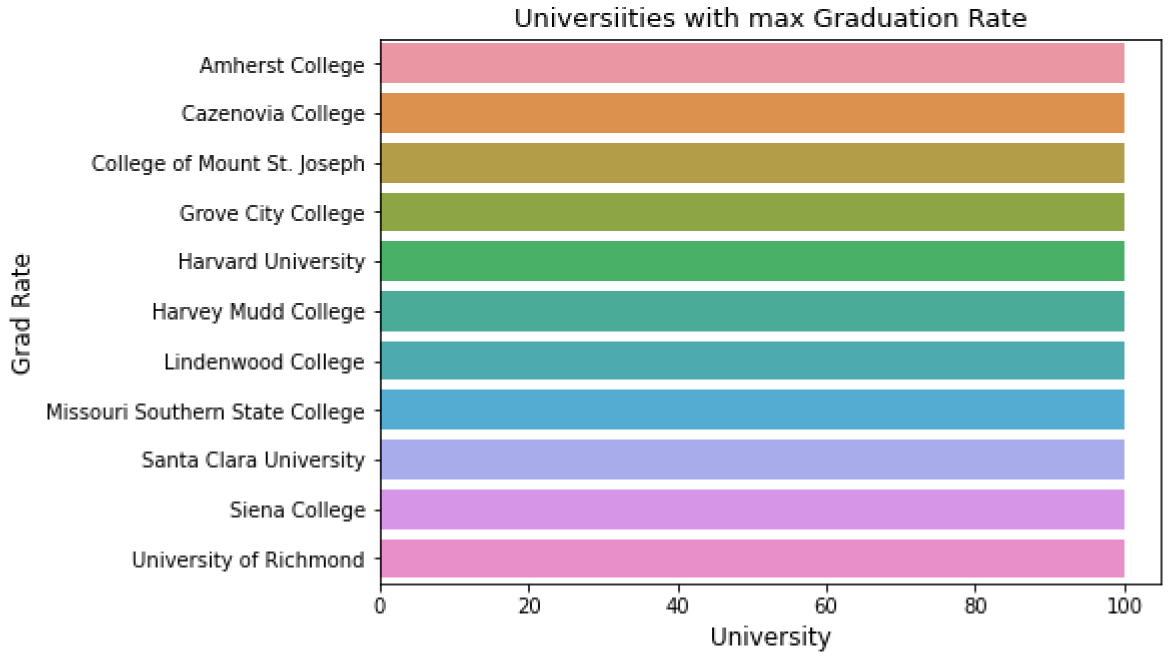
```
df['Grad.Rate'].replace(118,100,inplace = True)
```



```
df.sort_values(by = 'Grad.Rate',ascending = False)[['Grad.Rate','Names']].head()
```

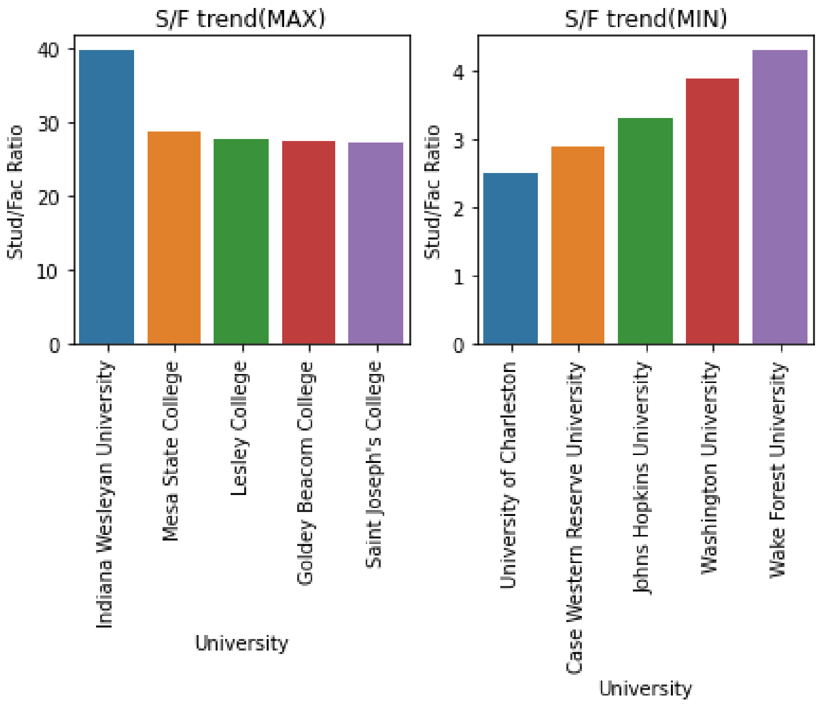


	Grad.Rate	Names
668	100	University of Richmond
16	100	Amherst College
238	100	Grove City College
250	100	Harvard University
251	100	Harvey Mudd College



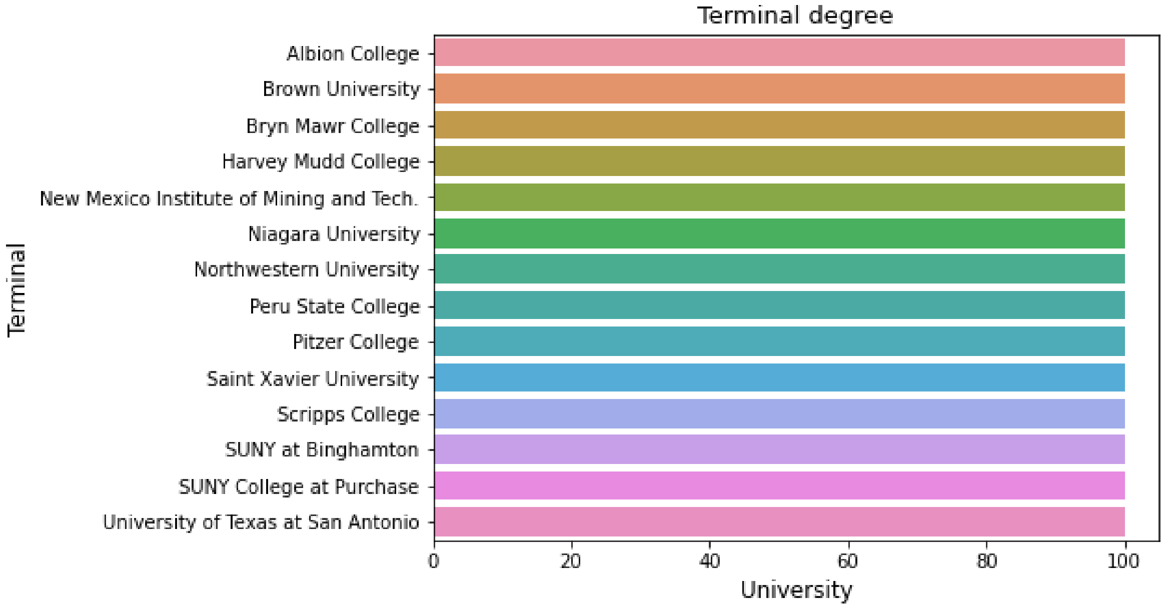
The above graph shows the top universities with 100% graduation rate.

7. Student Faculty Ratio



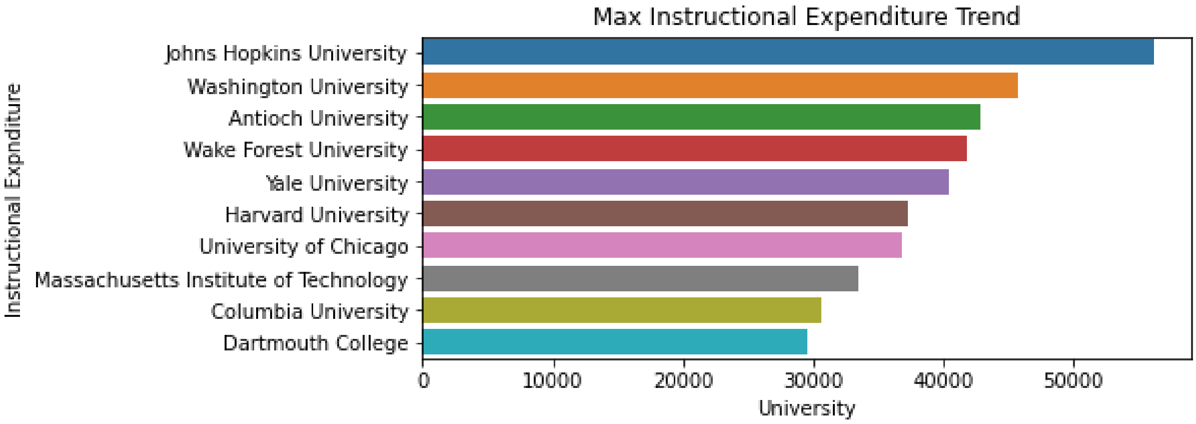
Lesser the student faculty ration, more are the teachers available for a particular student. Suppose University of charleston has S/F ratio of 2.5, which means, for every faculty there are 2.5 students, although 2.5 is not practically possible, but this what it means mathematically.On the other hand Indiana Wesleyan has a ration of 40 suggesting, a faculty has to handle 40 students.

8. Terminal Degree



The above universities have 100% faculties have terminal degree.

9. Instructional Expenditure: Amount spent by University per student



As can be seen, JHU tops the list with over 50k Expedn.

2.2) Scale the variables and write the inference for using the type of scaling function for this case study.

Scaling the data is one of the most important steps. In the given data set there are 18 columns, with numerical values having different units. It is not possible to make comparisons between two different units. For e.g. we can't compare height in metres to weight in kgs. Both units will have different magnitude. The one with greater magnitude and variability is going to influence the model more.

Hence to give a sense of equality, we need to perform scaling, so that it's an apple to apple comparison and not apple to oranges.

There are two types of scaling that can be used here: Min max scaling and Standard scaler.

The Standard scaler or Z scaler works on the following formula: -

$$\frac{X - X(bar)}{\sigma}$$

The formula changes the mean to zero and standard deviation to ±1. The ditribution of data shifts to an almost normal distribution.

It can be used in scenarios when there are outliers and we want them to scale them in a small interval.

The min max scaleer uses the following formula: -

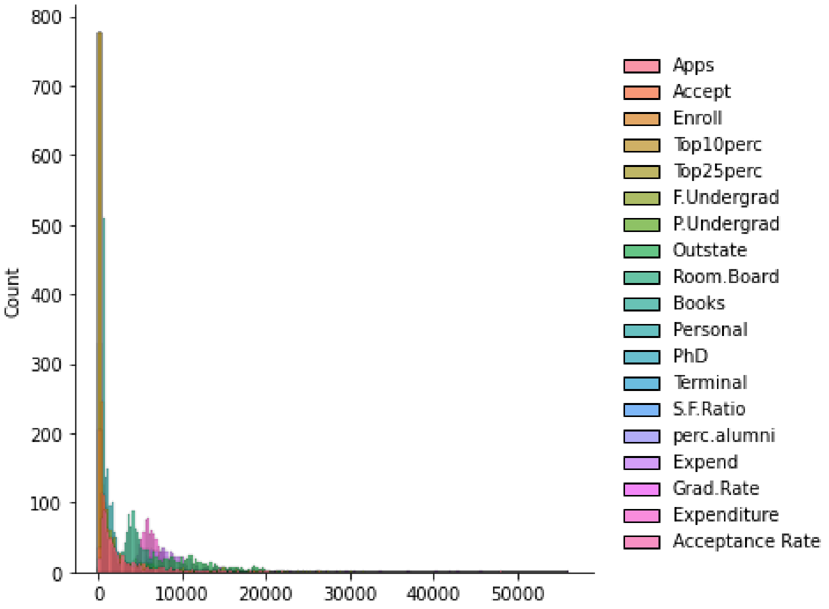
$$\frac{X - X(min)}{X(max) - X(\mathrm{min})}$$

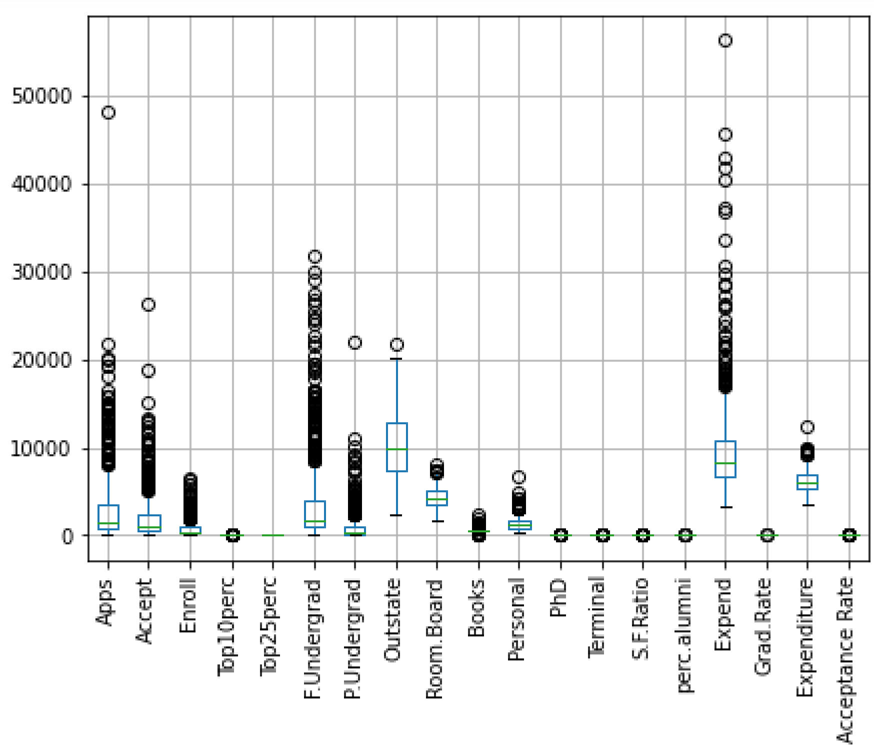
The formula scales the data in an interval of 0 to 1.It is least disruptive to outliers.

For the data set given it would be wise to move ahead with the standard scaler, as it transforms the mean to zero and std deviation to 1. The outliers seem to be in a small interval. Also, the distribution tends to normal, and for a statician, Normal distribution is always an added advantage.

To analye the effect of scaling, lets look at the distribution of data before and after applying scaling: -

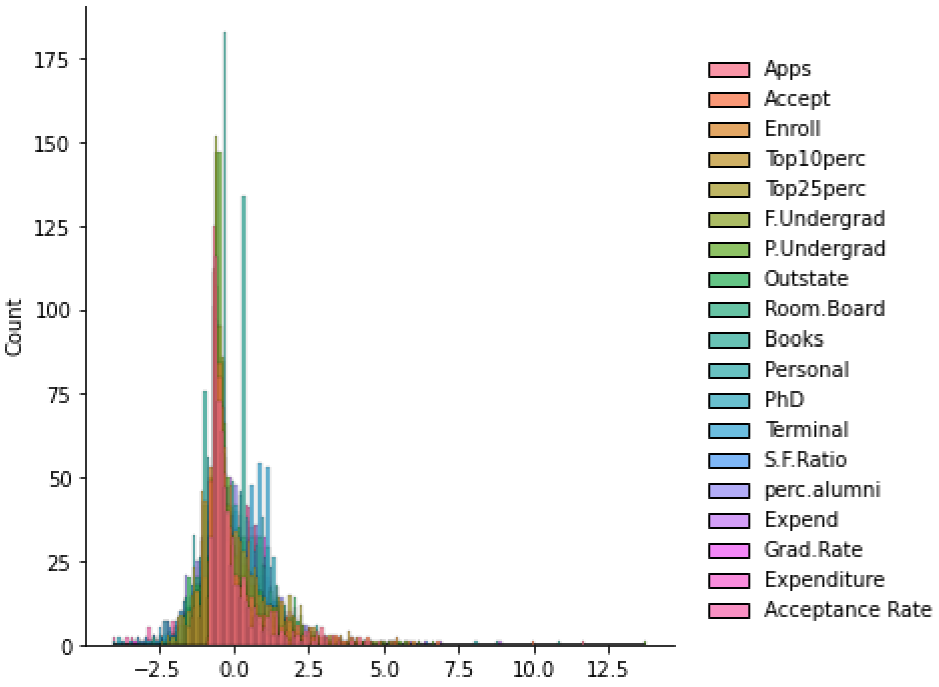
Distribution of data before scaling

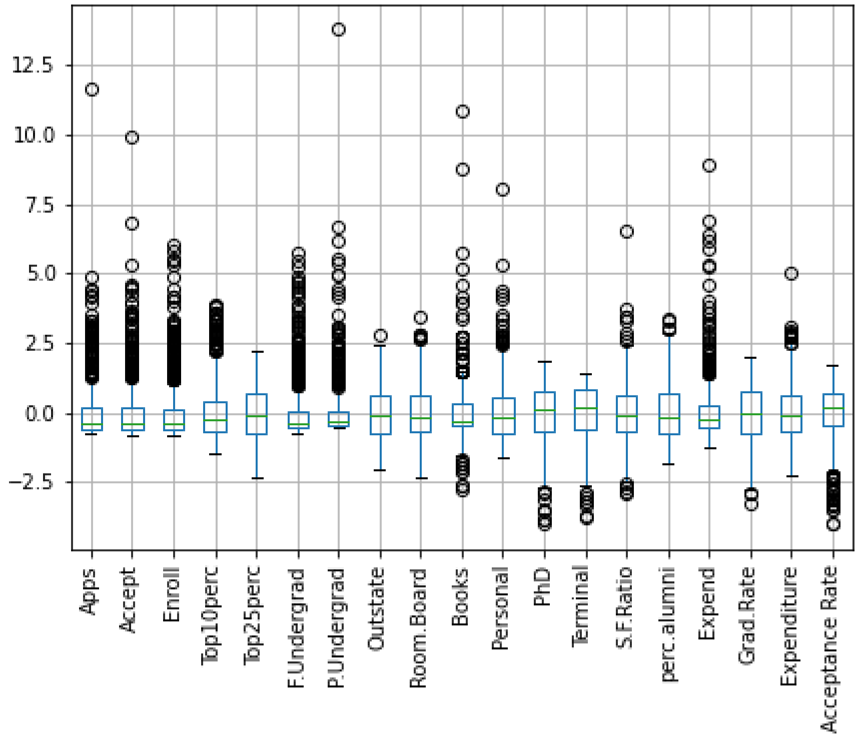




The data is right skewed and there are many outliers present in the data.

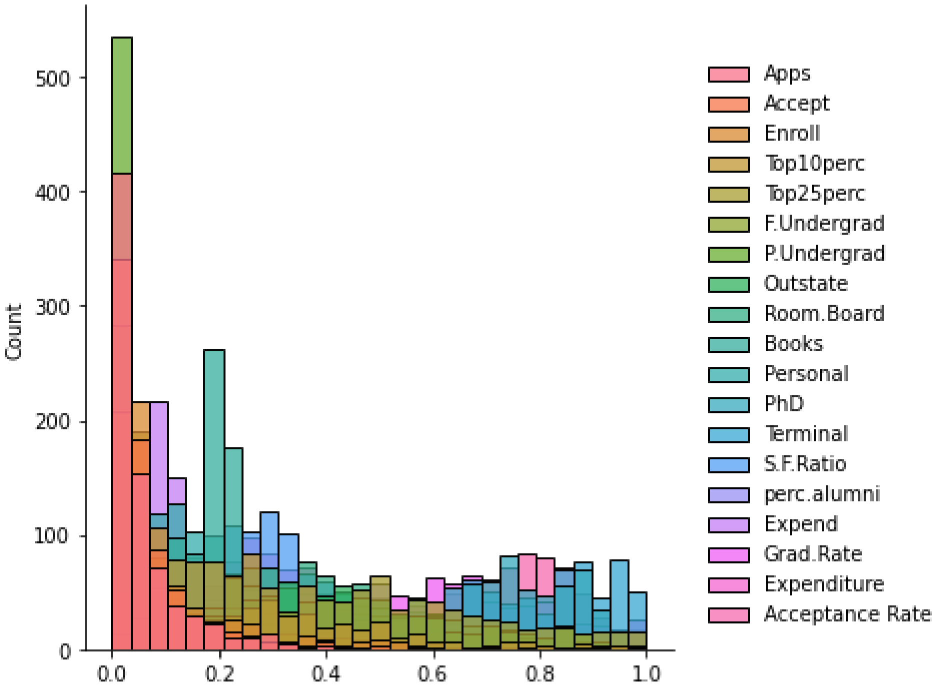
Distribution after applying Standard scaler

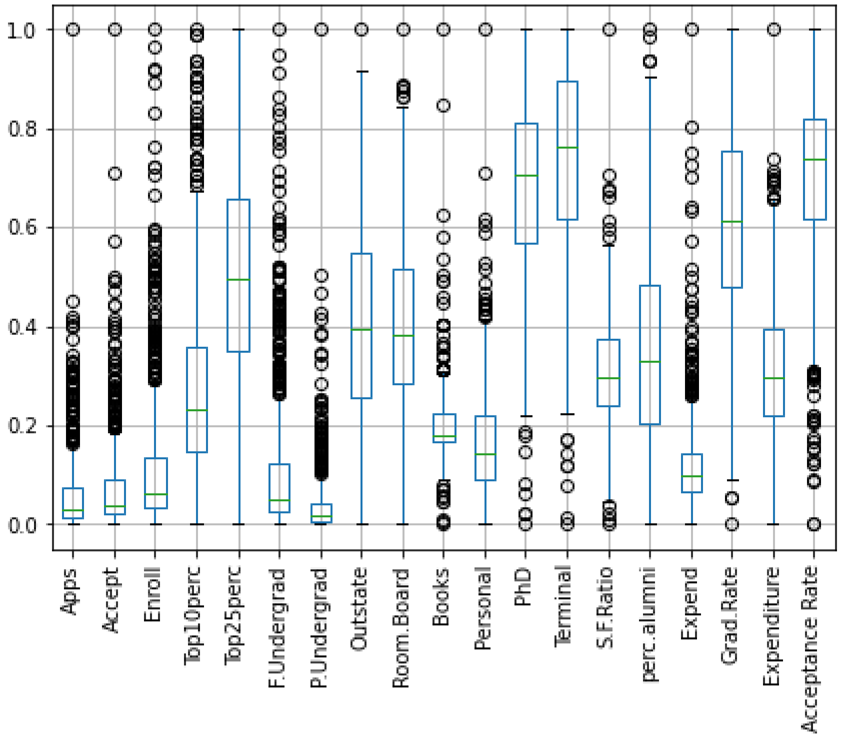




The distribution is tending to normal now. Also the outliers are scaled to a smaller interval.

Distribution after Min max scaler





Shape of the original data is retained, and the data is scaled between 0 and 1.

2.3) Comment on the comparison between covariance and the correlation matrix after scaling.

Let’s mathematically see, what is meant by covariation and corelation.

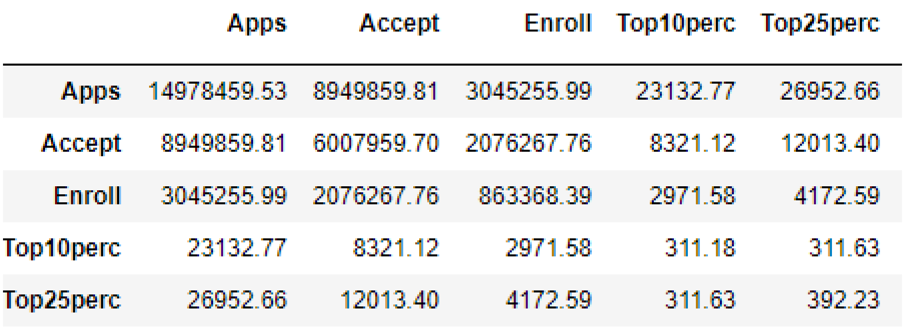
Covariance (X, Y) or cov (X, Y) = $\frac{\sum(Xi-Xbar)\times(Yi-Ybar)}{n}$

Corelation is defined as Corr (X, Y) = $\frac{COV\ (X,Y)}{\sigma x \sigma y}$

Çovariance tells us the direction of relationship between X and Y. A positive relationship means, both variables progress in the same direction

Corelation also tells the strength of the relationship. The corelation value is between -1 to +1. The value close to 1 or -1 indicates more strength, while close to zero indicates weak strength.

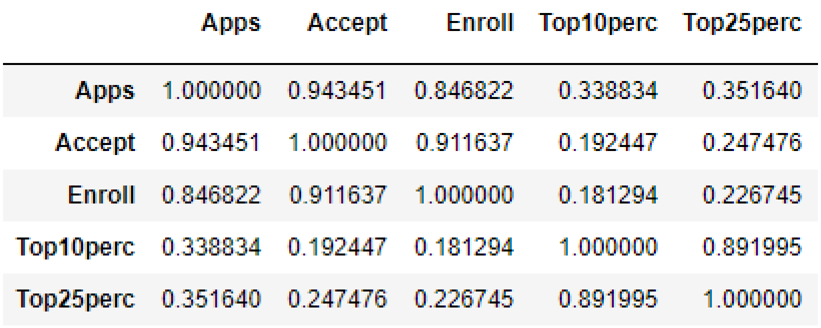
Covariance and Corelation for original data



	Apps	Accept	Enroll	Top10perc	Top25perc
Apps	14978459.53	8949859.81	3045255.99	23132.77	26952.66
Accept	8949859.81	6007959.70	2076267.76	8321.12	12013.40
Enroll	3045255.99	2076267.76	863368.39	2971.58	4172.59
Top10perc	23132.77	8321.12	2971.58	311.18	311.63
Top25perc	26952.66	12013.40	4172.59	311.63	392.23

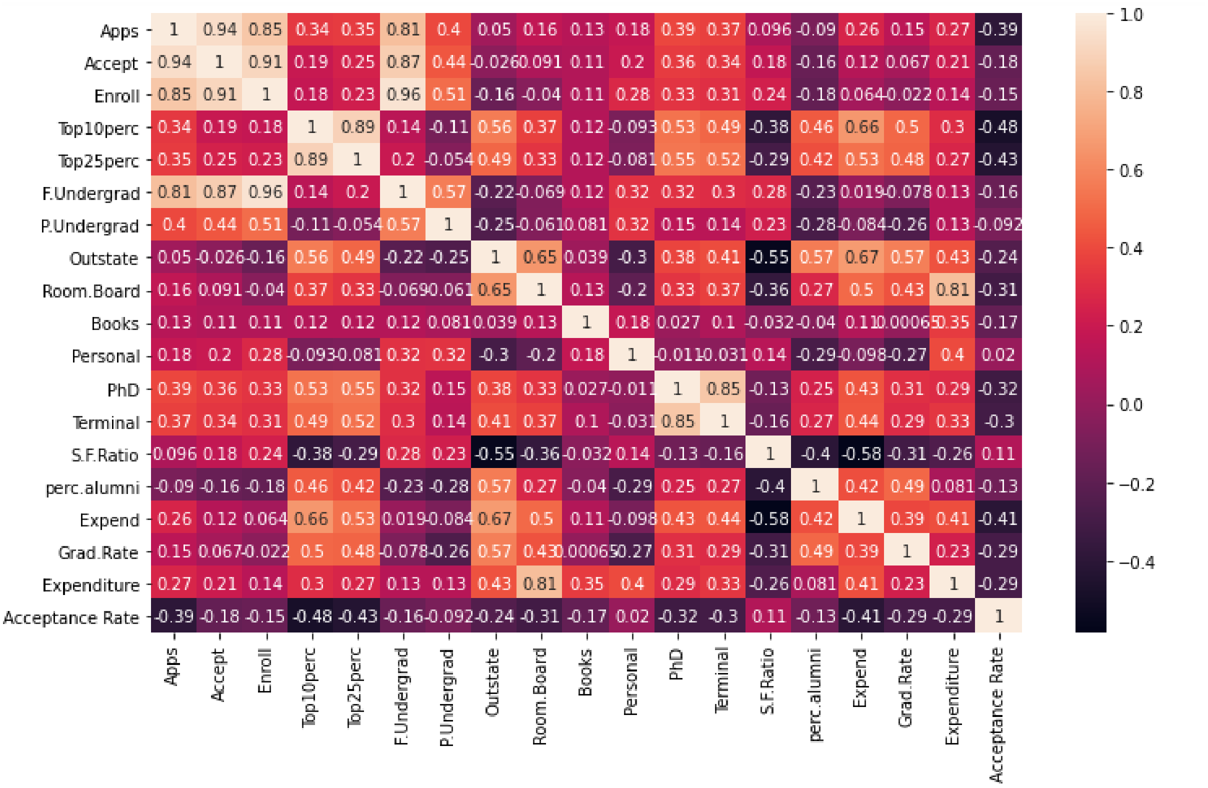
Here is a glmpse of the covariance matrix for original data. We can clearly see, the magnitude ranges from as low as 311 to as high as 1.4 e7. The covariance matrix is not of much use, as the values are not scaled.

To get a better idea about the strenght we can have a look at the corelation matrix.



	Apps	Accept	Enroll	Top10perc	Top25perc
Apps	1.000000	0.943451	0.846822	0.338834	0.351640
Accept	0.943451	1.000000	0.911637	0.192447	0.247476
Enroll	0.846822	0.911637	1.000000	0.181294	0.226745
Top10perc	0.338834	0.192447	0.181294	1.000000	0.891995
Top25perc	0.351640	0.247476	0.226745	0.891995	1.000000

The glimpse of the corelation matrix tells us that the diagonal is 1, and the values are scaled now. A heatmap could better justify the same.



	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	PhD	Terminal	S.F.Ratio	perc.alumni	Expend	Grad.Rate	Expenditure	Acceptance Rate
Apps	1	0.94	0.85	0.34	0.35	0.81	0.4	0.05	0.16	0.13	0.18	0.39	0.37	0.096	-0.09	0.26	0.15	0.27	-0.39
Accept	0.94	1	0.91	0.19	0.25	0.87	0.44	-0.026	0.091	0.11	0.2	0.36	0.34	0.18	-0.16	0.12	0.067	0.21	-0.18
Enroll	0.85	0.91	1	0.18	0.23	0.96	0.51	-0.16	-0.04	0.11	0.28	0.33	0.31	0.24	-0.18	0.064	-0.022	0.14	-0.15
Top10perc	0.34	0.19	0.18	1	0.89	0.14	-0.11	0.56	0.37	0.12	-0.093	0.53	0.49	-0.38	0.46	0.66	0.5	0.3	-0.48
Top25perc	0.35	0.25	0.23	0.89	1	0.2	-0.054	0.49	0.33	0.12	-0.081	0.55	0.52	-0.29	0.42	0.53	0.48	0.27	-0.43
F.Undergrad	0.81	0.87	0.96	0.14	0.2	1	0.57	-0.22	-0.069	0.12	0.32	0.32	0.3	0.28	-0.23	0.019	-0.078	0.13	-0.16
P.Undergrad	0.4	0.44	0.51	-0.11	-0.054	0.57	1	-0.25	-0.061	0.081	0.32	0.15	0.14	0.23	-0.28	-0.084	-0.26	0.13	-0.092
Outstate	0.05	-0.026	-0.16	0.56	0.49	-0.22	-0.25	1	0.65	0.039	-0.3	0.38	0.41	-0.55	0.57	0.67	0.57	0.43	-0.24
Room.Board	0.16	0.091	-0.04	0.37	0.33	-0.069	-0.061	0.65	1	0.13	-0.2	0.33	0.37	-0.36	0.27	0.5	0.43	0.81	-0.31
Books	0.13	0.11	0.11	0.12	0.12	0.12	0.081	0.039	0.13	1	0.18	0.027	0.1	-0.032	-0.04	0.11	0.00065	0.35	-0.17
Personal	0.18	0.2	0.28	-0.093	-0.081	0.32	0.32	-0.3	-0.2	0.18	1	-0.011	-0.031	0.14	-0.29	-0.098	-0.27	0.4	0.02
PhD	0.39	0.36	0.33	0.53	0.55	0.32	0.15	0.38	0.33	0.027	-0.011	1	0.85	-0.13	0.25	0.43	0.31	0.29	-0.32
Terminal	0.37	0.34	0.31	0.49	0.52	0.3	0.14	0.41	0.37	0.1	-0.031	0.85	1	-0.16	0.27	0.44	0.29	0.33	-0.3
S.F.Ratio	0.096	0.18	0.24	-0.38	-0.29	0.28	0.23	-0.55	-0.36	-0.032	0.14	-0.13	-0.16	1	-0.4	-0.58	-0.31	-0.26	0.11
perc.alumni	-0.09	-0.16	-0.18	0.46	0.42	-0.23	-0.28	0.57	0.27	-0.04	-0.29	0.25	0.27	-0.4	1	0.42	0.49	0.081	-0.13
Expend	0.26	0.12	0.064	0.66	0.53	0.019	-0.084	0.67	0.5	0.11	-0.098	0.43	0.44	-0.58	0.42	1	0.39	0.41	-0.41
Grad.Rate	0.15	0.067	-0.022	0.5	0.48	-0.078	-0.26	0.57	0.43	0.00065	-0.27	0.31	0.29	-0.31	0.49	0.39	1	0.23	-0.29
Expenditure	0.27	0.21	0.14	0.3	0.27	0.13	0.13	0.43	0.81	0.35	0.4	0.29	0.33	-0.26	0.081	0.41	0.23	1	-0.29
Acceptance Rate	-0.39	-0.18	-0.15	-0.48	-0.43	-0.16	-0.092	-0.24	-0.31	-0.17	0.02	-0.32	-0.3	0.11	-0.13	-0.41	-0.29	-0.29	1

The heatmap depict all the strenghts in a scaled manner between -1 to +1.

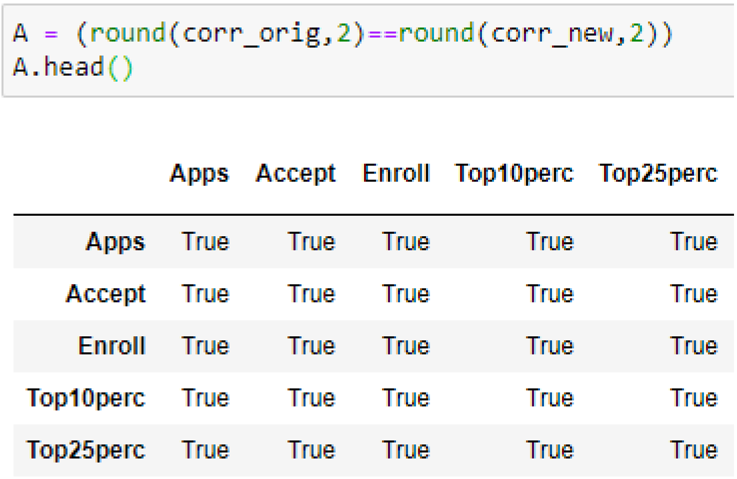
Covariance and Corelation for the new data

The new data is scaled as per the standard scaler. The new mean is zero and the std deviation is ±1.

Thus COV (X, Y) = $\frac{\sum(Xi-Xbar)\times(Yi-Ybar)}{n}$ and CORR (X,Y) = $\frac{COV\ (X,Y)}{\sigma x \sigma y}$ are going to be the same. The reason being that, during scaling the variables were already divided by the std deviation. Hence while calculating the COV, it is in a way giving the values for correlation. At the same time while computing the corelation there is no effect of std deviation as after scaling the std deviation has changed to 1. Thus overall, the corelation matrix for orignal data, the corelation matrix for new data and the covariance data for new data are the same.

Comparing them with each other would make it even more clear. Since the dataset consists of too many columns, a wiser way would be to just store there comparisons in a separate dataframe. This datafram would contain only two values either True or False. If at all the dataframe contains a single False, we would know the dataframes are not equal.

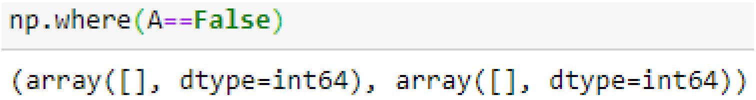
Comparing corelation original data and scaled data: -



```
A = (round(corr_orig,2)==round(corr_new,2))
A.head()
```

	Apps	Accept	Enroll	Top10perc	Top25perc
Apps	True	True	True	True	True
Accept	True	True	True	True	True
Enroll	True	True	True	True	True
Top10perc	True	True	True	True	True
Top25perc	True	True	True	True	True

Just to have an understanding, the comparison values are stored in a dataframe called A. The values depicted so far hold variable True indicating the values are equal. Using np we can find if there is a False value in the dataframe.

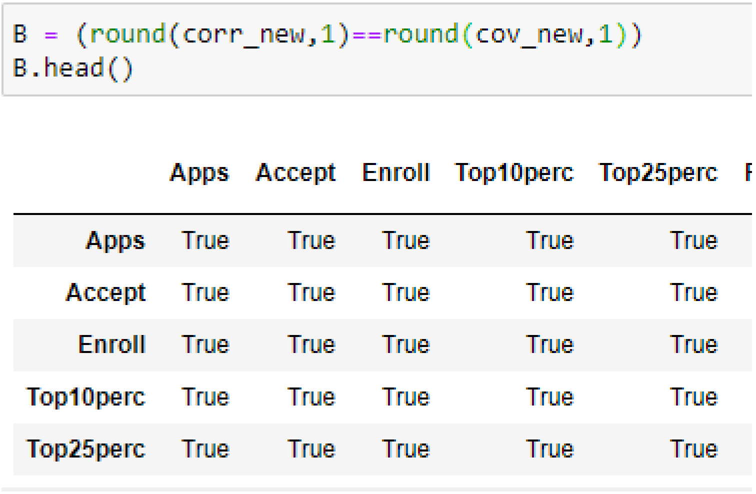


```
np.where(A==False)
```

```
(array([], dtype=int64), array([], dtype=int64))
```

No false values, hence we can say that the two datasets are equal

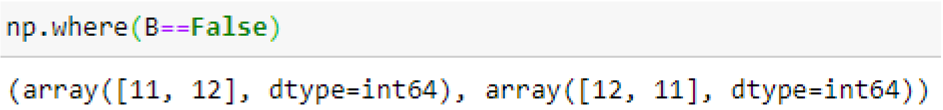
Comparing corelation original data and scaled data: -



```
B = (round(corr_new,1)==round(cov_new,1))
B.head()
```

	Apps	Accept	Enroll	Top10perc	Top25perc	F
Apps	True	True	True	True	True	
Accept	True	True	True	True	True	
Enroll	True	True	True	True	True	
Top10perc	True	True	True	True	True	
Top25perc	True	True	True	True	True	

Similarly, the boolean values for comparison of original corelation and covariance new are stored. Let’s find out if there is any False value in the dataframe.

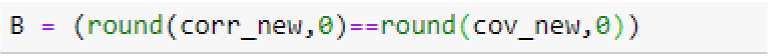


```
np.where(B==False)
```

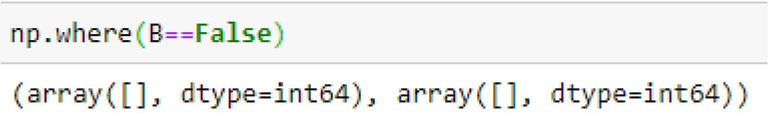
```
(array([11, 12], dtype=int64), array([12, 11], dtype=int64))
```

Yes, there is a False value. Row 11 and column 12 do not match.

But if we only consider the whole no value, ignoring the decimal, the values are entirely same.



```
B = (round(corr_new,0)==round(cov_new,0))
```



```
np.where(B==False)
```

```
(array([], dtype=int64), array([], dtype=int64))
```

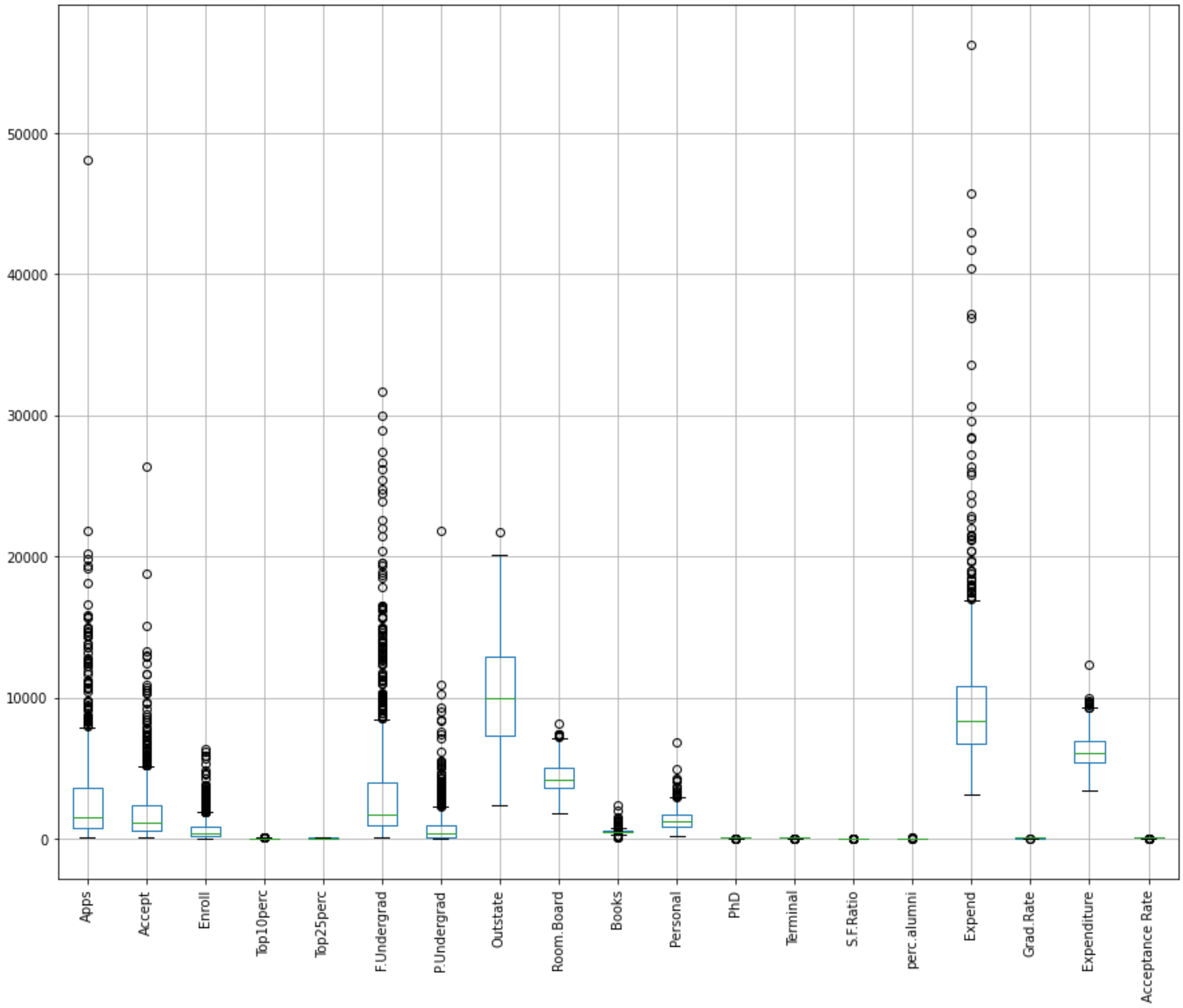
Yes the assumption holds good, and the values ignoring the decimals are same.

Final Conclusion

The corelation matrix for the original data set, covariance matrix for the scaled data and the corelation matrix for the scaled data are all the same, the reasons for this have already been listed above.

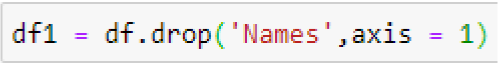
2.4) Check the dataset for outliers before and after scaling. Draw your inferences from this exercise.

Using df.boxplot() to have a look at the outliers.

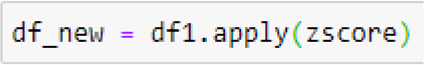


As we can observe, there are multiple outliers present in the data. Except the column Top25perc every column contains outliers.

Let's see what impact scaling has on outliers. In a previous question (2.2), the scaling and its effect was demonstrated. Using the same we have the following: -



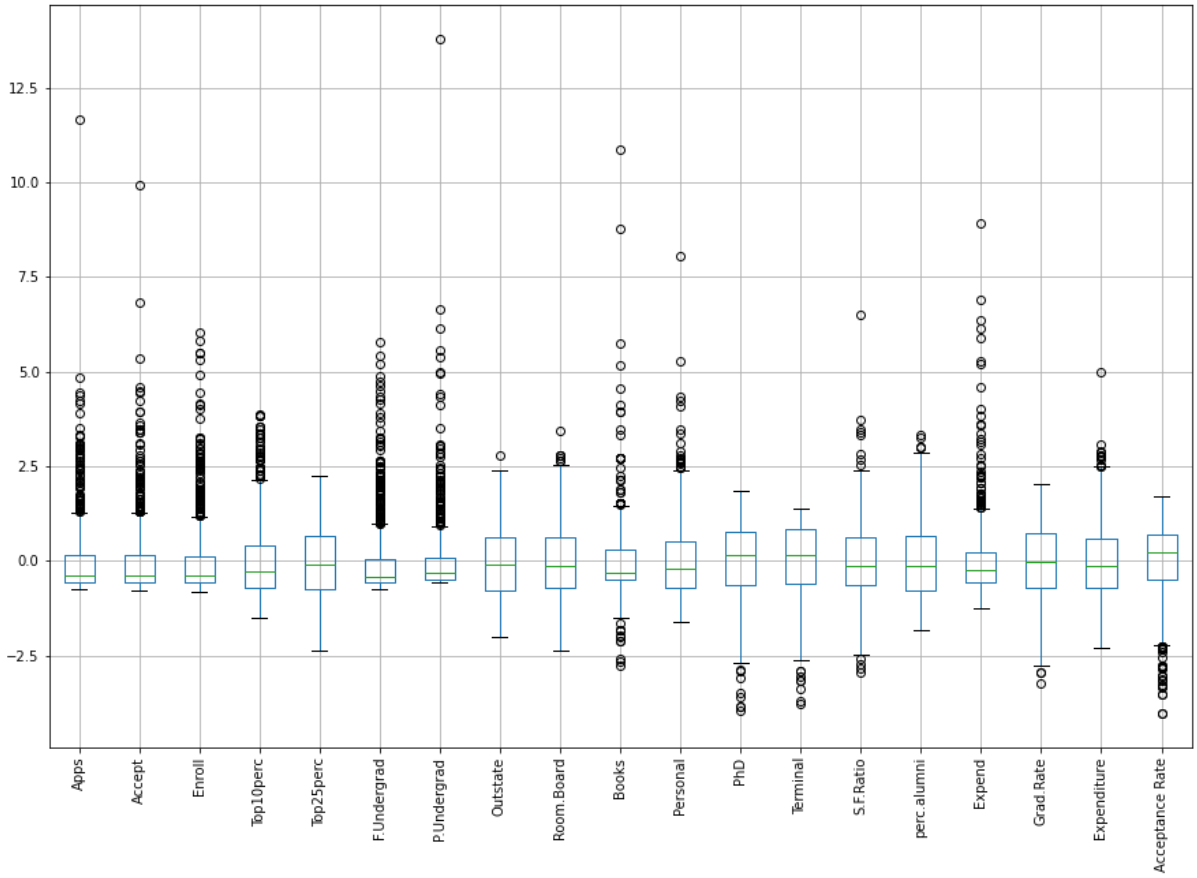
```
df1 = df.drop('Names',axis = 1)
```



```
df_new = df1.apply(zscore)
```

Previously for Q2.2 data frame df_new was created to store the scaled data(Using Z scaling).

The boxplot for the scaled data: -



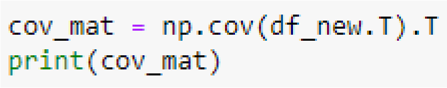
Scaling does not have any significant effect on the outliers, except for the fact that the outliers have a compact interval now. Also, all the values are on the same scale.

But the fact of the matter is that we can’t expect to get rid off outliers using scaling. We need to explicitly delete from the data. However, there is a significant loss of information on removing outliers. Although removing outliers is an essential step for data cleaning and preparation, but here we can see there are many outliers, and removing them would lead to a significant information loss.

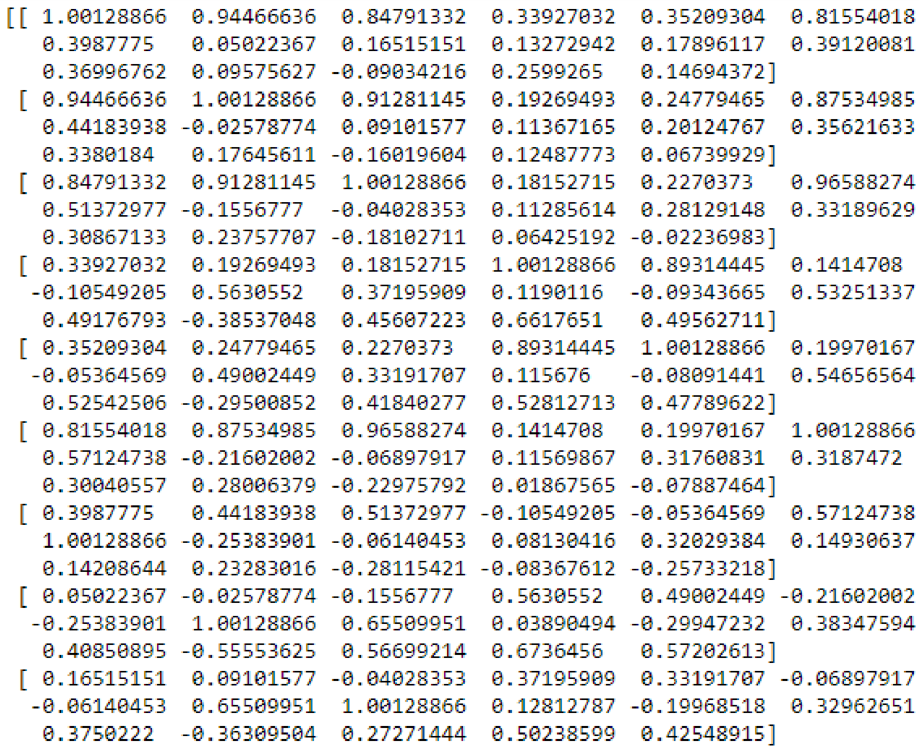
(As mentioned in the FAQ’s we can move ahead without removing outliers)

2.5) Build the covariance matrix and calculate the eigenvalues and the eigenvector.

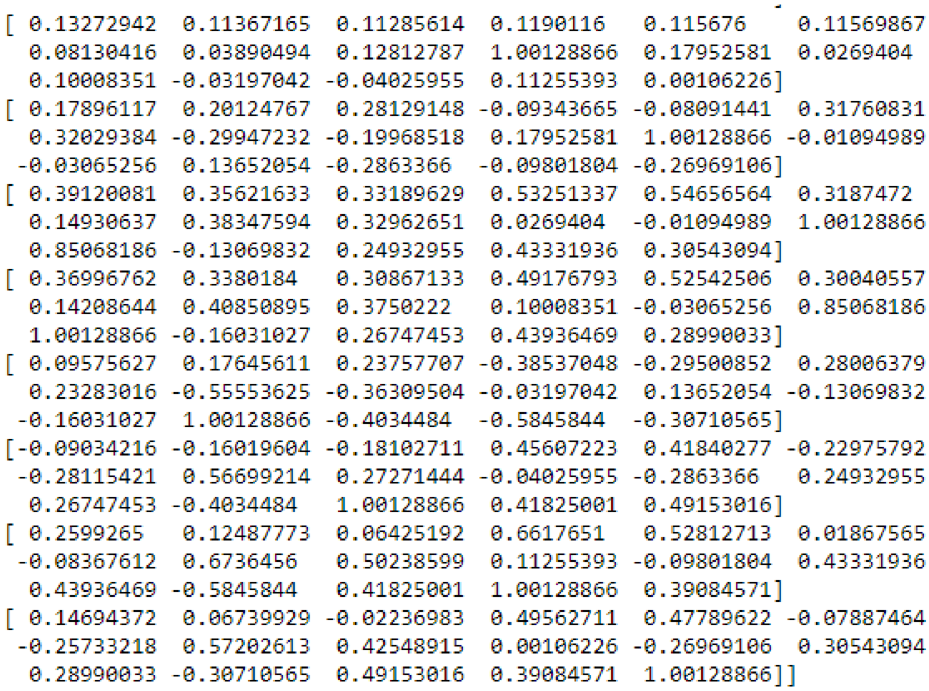
The covariance matrix is calculated as: -



```
cov_mat = np.cov(df_new.T).T
print(cov_mat)
```

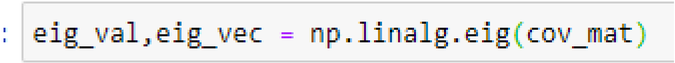


```
[[ 1.00128866  0.94466636  0.84791332  0.33927032  0.35209304  0.81554018
   0.3987775   0.05022367  0.16515151  0.13272942  0.17896117  0.39120081
   0.36996762  0.09575627 -0.09034216  0.2599265   0.14694372]
 [ 0.94466636  1.00128866  0.91281145  0.19269493  0.24779465  0.87534985
   0.44183938 -0.02578774  0.09101577  0.11367165  0.20124767  0.35621633
   0.3380184   0.17645611 -0.16019604  0.12487773  0.06739929]
 [ 0.84791332  0.91281145  1.00128866  0.18152715  0.2270373   0.96588274
   0.51372977 -0.1556777  -0.04028353  0.11285614  0.28129148  0.33189629
   0.30867133  0.23757707 -0.18102711  0.06425192 -0.02236983]
 [ 0.33927032  0.19269493  0.18152715  1.00128866  0.89314445  0.1414708
  -0.10549205  0.5630552   0.37195909  0.1190116  -0.09343665  0.53251337
   0.49176793 -0.38537048  0.45607223  0.6617651   0.49562711]
 [ 0.35209304  0.24779465  0.2270373   0.89314445  1.00128866  0.19970167
  -0.05364569  0.49002449  0.33191707  0.115676   -0.08091441  0.54656564
   0.52542506 -0.29500852  0.41840277  0.52812713  0.47789622]
 [ 0.81554018  0.87534985  0.96588274  0.1414708   0.19970167  1.00128866
   0.57124738 -0.21602002 -0.06897917  0.11569867  0.31760831  0.3187472
   0.30040557  0.28006379 -0.22975792  0.01867565 -0.07887464]
 [ 0.3987775   0.44183938  0.51372977 -0.10549205 -0.05364569  0.57124738
   1.00128866 -0.25383901 -0.06140453  0.08130416  0.32029384  0.14930637
   0.14208644  0.23283016 -0.28115421 -0.08367612 -0.25733218]
 [ 0.05022367 -0.02578774 -0.1556777   0.5630552   0.49002449 -0.21602002
  -0.25383901  1.00128866  0.65509951  0.03890494 -0.29947232  0.38347594
   0.40850895 -0.55553625  0.56699214  0.6736456   0.57202613]
 [ 0.16515151  0.09101577 -0.04028353  0.37195909  0.33191707 -0.06897917
  -0.06140453  0.65509951  1.00128866  0.12812787 -0.19968518  0.32962651
   0.3750222  -0.36309504  0.27271444  0.50238599  0.42548915]
```

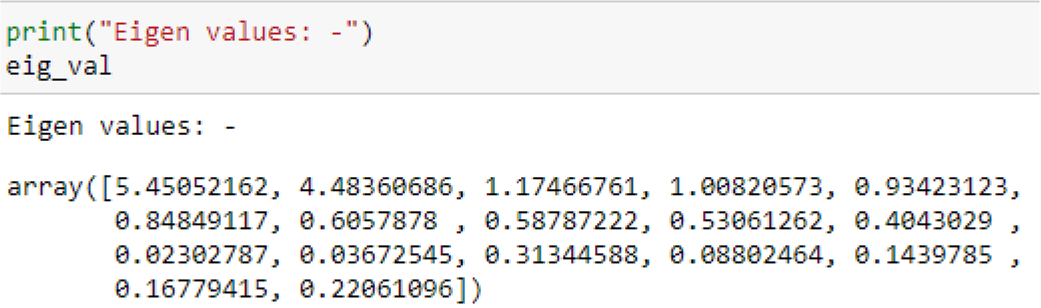


```
[ 0.13272942  0.11367165  0.11285614  0.1190116   0.115676    0.11569867
  0.08130416  0.03890494  0.12812787  1.00128866  0.17952581  0.0269404
  0.10008351 -0.03197042 -0.04025955  0.11255393  0.00106226]
[ 0.17896117  0.20124767  0.28129148 -0.09343665 -0.08091441  0.31760831
  0.32029384 -0.29947232 -0.19968518  0.17952581  1.00128866 -0.01094989
 -0.03065256  0.13652054 -0.2863366  -0.09801804 -0.26969106]
[ 0.39120081  0.35621633  0.33189629  0.53251337  0.54656564  0.3187472
  0.14930637  0.38347594  0.32962651  0.0269404  -0.01094989  1.00128866
  0.85068186 -0.13069832  0.24932955  0.43331936  0.30543094]
[ 0.36996762  0.3380184   0.30867133  0.49176793  0.52542506  0.30040557
  0.14208644  0.40850895  0.3750222   0.10008351 -0.03065256  0.85068186
  1.00128866 -0.16031027  0.26747453  0.43936469  0.28990033]
[ 0.09575627  0.17645611  0.23757707 -0.38537048 -0.29500852  0.28006379
  0.23283016 -0.55553625 -0.36309504 -0.03197042  0.13652054 -0.13069832
 -0.16031027  1.00128866 -0.4034484  -0.5845844  -0.30710565]
[-0.09034216 -0.16019604 -0.18102711  0.45607223  0.41840277 -0.22975792
 -0.28115421  0.56699214  0.27271444 -0.04025955 -0.2863366   0.24932955
  0.26747453 -0.4034484   1.00128866  0.41825001  0.49153016]
[ 0.2599265   0.12487773  0.06425192  0.6617651   0.52812713  0.01867565
 -0.08367612  0.6736456   0.50238599  0.11255393 -0.09801804  0.43331936
  0.43936469 -0.5845844   0.41825001  1.00128866  0.39084571]
[ 0.14694372  0.06739929 -0.02236983  0.49562711  0.47789622 -0.07887464
 -0.25733218  0.57202613  0.42548915  0.00106226 -0.26969106  0.30543094
  0.28990033 -0.30710565  0.49153016  0.39084571  1.00128866]]
```

The eigen values and eigen vectors are calculated as: -



```
: eig_val,eig_vec = np.linalg.eig(cov_mat)
```

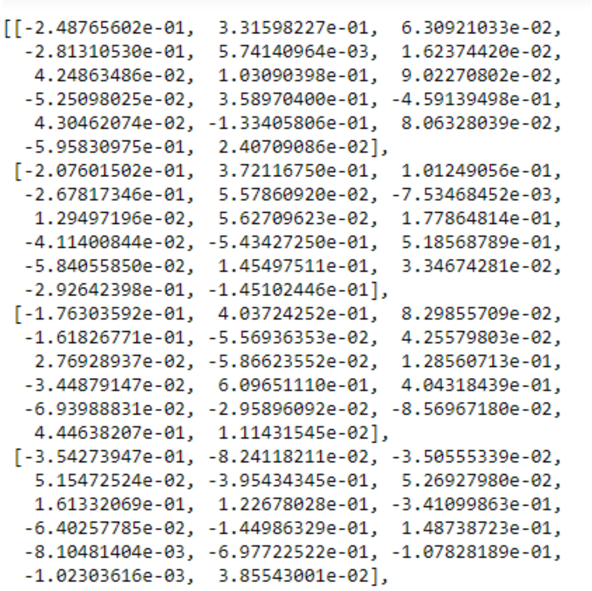


```
print("Eigen values: -")
eig_val
```

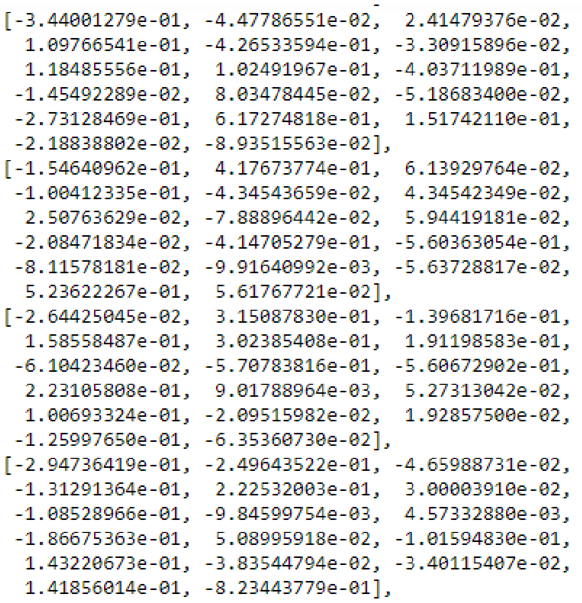
```
Eigen values: -

array([5.45052162, 4.48360686, 1.17466761, 1.00820573, 0.93423123,
       0.84849117, 0.6057878 , 0.58787222, 0.53061262, 0.4043029 ,
       0.02302787, 0.03672545, 0.31344588, 0.08802464, 0.1439785 ,
       0.16779415, 0.22061096])
```

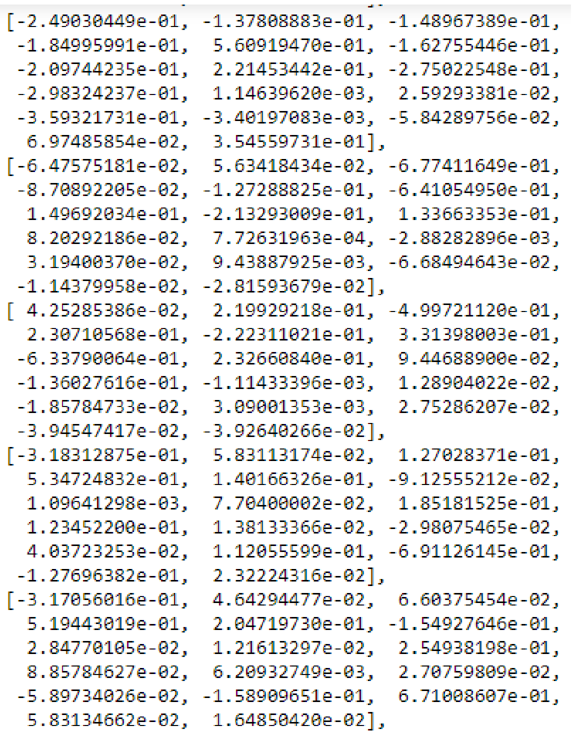
Eigen vectors: -



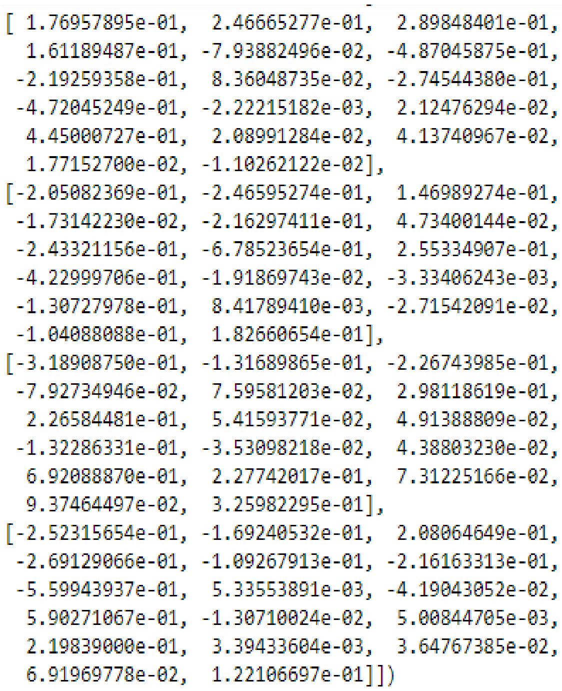
```
[[-2.48765602e-01,  3.31598227e-01,  6.30921033e-02,
  -2.81310530e-01,  5.74140964e-03,  1.62374420e-02,
   4.24863486e-02,  1.03090398e-01,  9.02270802e-02,
  -5.25098025e-02,  3.58970400e-01, -4.59139498e-01,
   4.30462074e-02, -1.33405806e-01,  8.06328039e-02,
  -5.95830975e-01,  2.40709086e-02],
 [-2.07601502e-01,  3.72116750e-01,  1.01249056e-01,
  -2.67817346e-01,  5.57860920e-02, -7.53468452e-03,
   1.29497196e-02,  5.62709623e-02,  1.77864814e-01,
  -4.11400844e-02, -5.43427250e-01,  5.18568789e-01,
  -5.84055850e-02,  1.45497511e-01,  3.34674281e-02,
  -2.92642398e-01, -1.45102446e-01],
 [-1.76303592e-01,  4.03724252e-01,  8.29855709e-02,
  -1.61826771e-01, -5.56936353e-02,  4.25579803e-02,
   2.76928937e-02, -5.86623552e-02,  1.28560713e-01,
  -3.44879147e-02,  6.09651110e-01,  4.04318439e-01,
  -6.93988831e-02, -2.95896092e-02, -8.56967180e-02,
   4.44638207e-01,  1.11431545e-02],
 [-3.54273947e-01, -8.24118211e-02, -3.50555339e-02,
   5.15472524e-02, -3.95434345e-01,  5.26927980e-02,
   1.61332069e-01,  1.22678028e-01, -3.41099863e-01,
  -6.40257785e-02, -1.44986329e-01,  1.48738723e-01,
  -8.10481404e-03, -6.97722522e-01, -1.07828189e-01,
  -1.02303616e-03,  3.85543001e-02],
```



```
[-3.44001279e-01, -4.47786551e-02,  2.41479376e-02,
  1.09766541e-01, -4.26533594e-01, -3.30915896e-02,
  1.18485556e-01,  1.02491967e-01, -4.03711989e-01,
 -1.45492289e-02,  8.03478445e-02, -5.18683400e-02,
 -2.73128469e-01,  6.17274818e-01,  1.51742110e-01,
 -2.18838802e-02, -8.93515563e-02],
[-1.54640962e-01,  4.17673774e-01,  6.13929764e-02,
 -1.00412335e-01, -4.34543659e-02,  4.34542349e-02,
  2.50763629e-02, -7.88896442e-02,  5.94419181e-02,
 -2.08471834e-02, -4.14705279e-01, -5.60363054e-01,
 -8.11578181e-02, -9.91640992e-03, -5.63728817e-02,
  5.23622267e-01,  5.61767721e-02],
[-2.64425045e-02,  3.15087830e-01, -1.39681716e-01,
  1.58558487e-01,  3.02385408e-01,  1.91198583e-01,
 -6.10423460e-02, -5.70783816e-01, -5.60672902e-01,
  2.23105808e-01,  9.01788964e-03,  5.27313042e-02,
  1.00693324e-01, -2.09515982e-02,  1.92857500e-02,
 -1.25997650e-01, -6.35360730e-02],
[-2.94736419e-01, -2.49643522e-01, -4.65988731e-02,
 -1.31291364e-01,  2.22532003e-01,  3.00003910e-02,
 -1.08528966e-01, -9.84599754e-03,  4.57332880e-03,
 -1.86675363e-01,  5.08995918e-02, -1.01594830e-01,
  1.43220673e-01, -3.83544794e-02, -3.40115407e-02,
  1.41856014e-01, -8.23443779e-01],
```



```
[-2.49030449e-01, -1.37808883e-01, -1.48967389e-01,
 -1.84995991e-01,  5.60919470e-01, -1.62755446e-01,
 -2.09744235e-01,  2.21453442e-01, -2.75022548e-01,
 -2.98324237e-01,  1.14639620e-03,  2.59293381e-02,
 -3.59321731e-01, -3.40197083e-03, -5.84289756e-02,
  6.97485854e-02,  3.54559731e-01],
[-6.47575181e-02,  5.63418434e-02, -6.77411649e-01,
 -8.70892205e-02, -1.27288825e-01, -6.41054950e-01,
  1.49692034e-01, -2.13293009e-01,  1.33663353e-01,
  8.20292186e-02,  7.72631963e-04, -2.88282896e-03,
  3.19400370e-02,  9.43887925e-03, -6.68494643e-02,
 -1.14379958e-02, -2.81593679e-02],
[ 4.25285386e-02,  2.19929218e-01, -4.99721120e-01,
  2.30710568e-01, -2.22311021e-01,  3.31398003e-01,
 -6.33790064e-01,  2.32660840e-01,  9.44688900e-02,
 -1.36027616e-01, -1.11433396e-03,  1.28904022e-02,
 -1.85784733e-02,  3.09001353e-03,  2.75286207e-02,
 -3.94547417e-02, -3.92640266e-02],
[-3.18312875e-01,  5.83113174e-02,  1.27028371e-01,
  5.34724832e-01,  1.40166326e-01, -9.12555212e-02,
  1.09641298e-03,  7.70400002e-02,  1.85181525e-01,
  1.23452200e-01,  1.38133366e-02, -2.98075465e-02,
  4.03723253e-02,  1.12055599e-01, -6.91126145e-01,
 -1.27696382e-01,  2.32224316e-02],
[-3.17056016e-01,  4.64294477e-02,  6.60375454e-02,
  5.19443019e-01,  2.04719730e-01, -1.54927646e-01,
  2.84770105e-02,  1.21613297e-02,  2.54938198e-01,
  8.85784627e-02,  6.20932749e-03,  2.70759809e-02,
 -5.89734026e-02, -1.58909651e-01,  6.71008607e-01,
  5.83134662e-02,  1.64850420e-02],
```



```
[ 1.76957895e-01,  2.46665277e-01,  2.89848401e-01,
  1.61189487e-01, -7.93882496e-02, -4.87045875e-01,
 -2.19259358e-01,  8.36048735e-02, -2.74544380e-01,
 -4.72045249e-01, -2.22215182e-03,  2.12476294e-02,
  4.45000727e-01,  2.08991284e-02,  4.13740967e-02,
  1.77152700e-02, -1.10262122e-02],
[-2.05082369e-01, -2.46595274e-01,  1.46989274e-01,
 -1.73142230e-02, -2.16297411e-01,  4.73400144e-02,
 -2.43321156e-01, -6.78523654e-01,  2.55334907e-01,
 -4.22999706e-01, -1.91869743e-02, -3.33406243e-03,
 -1.30727978e-01,  8.41789410e-03, -2.71542091e-02,
 -1.04088088e-01,  1.82660654e-01],
[-3.18908750e-01, -1.31689865e-01, -2.26743985e-01,
 -7.92734946e-02,  7.59581203e-02,  2.98118619e-01,
  2.26584481e-01,  5.41593771e-02,  4.91388809e-02,
 -1.32286331e-01, -3.53098218e-02,  4.38803230e-02,
  6.92088870e-01,  2.27742017e-01,  7.31225166e-02,
  9.37464497e-02,  3.25982295e-01],
[-2.52315654e-01, -1.69240532e-01,  2.08064649e-01,
 -2.69129066e-01, -1.09267913e-01, -2.16163313e-01,
 -5.59943937e-01,  5.33553891e-03, -4.19043052e-02,
  5.90271067e-01, -1.30710024e-02,  5.00844705e-03,
  2.19839000e-01,  3.39433604e-03,  3.64767385e-02,
  6.91969778e-02,  1.22106697e-01]])
```

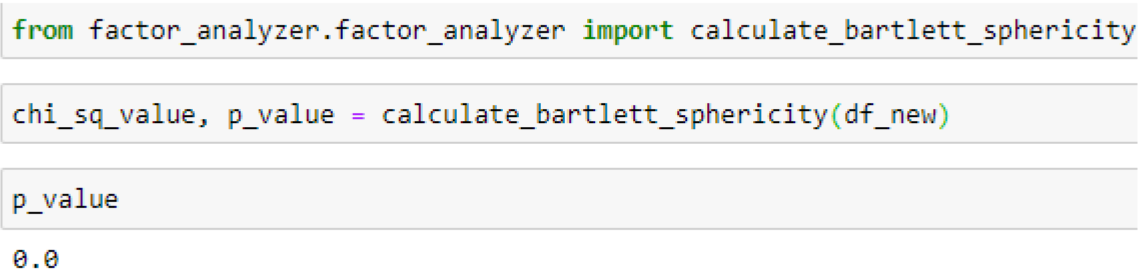
The eigen vectors contains 17 rows and 17 columns.

2.6) Write the explicit form of the first PC (in terms of Eigen Vectors).

Before moving ahead with PCA we need to perform Bartlett's sphericity test and KMO test to know if PCA can really do us some good.

Bartlett's test tells us if the data is corelated or not. The NULL hypothesis assumes the data to be uncorrelated, while the ALTERNATE says at least one pair is corelated.

Performing the Bartlett's sphericity test: -



```
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity
```

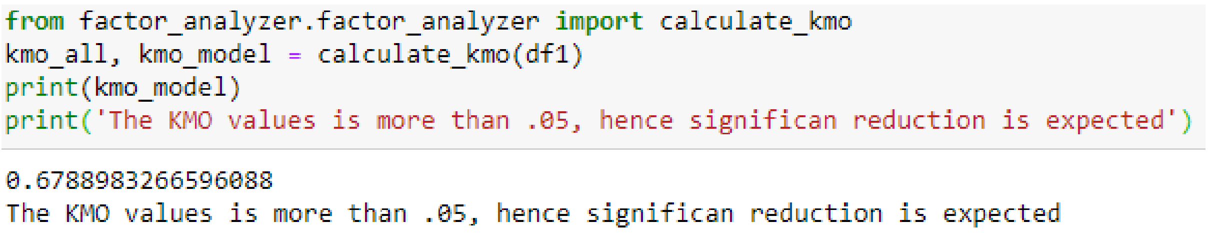
```
chi_sq_value, p_value = calculate_bartlett_sphericity(df_new)
```

```
p_value
```

```
0.0
```

P value is less than .05, hence NULL can be rejected, thus we can say at-least one pair is corelated, and hence performing PCA cane solve this collinearity.

Performing KMO test: KMO test tests for the data adequacy. It gives MSA (measure of sampling adequacy) which must be greater than .5. A value greater than .5 suggests significant reduction of data.



```
from factor_analyzer.factor_analyzer import calculate_kmo
kmo_all, kmo_model = calculate_kmo(df1)
print(kmo_model)
print('The KMO values is more than .05, hence significan reduction is expected')
```

```
0.6788983266596088
The KMO values is more than .05, hence significan reduction is expected
```

MSA = .67, significant reduction expected.

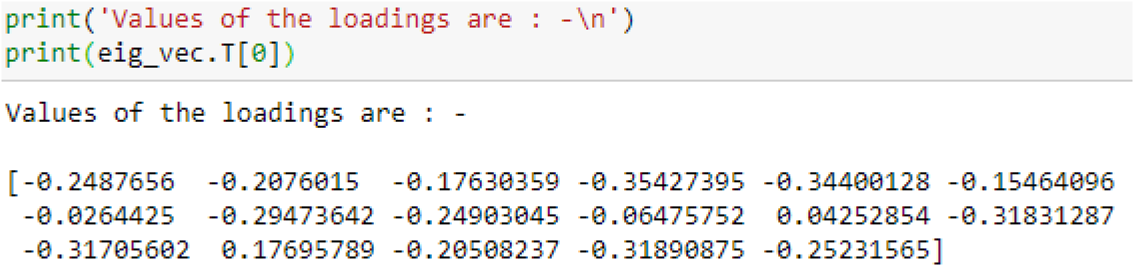
Now we can move ahead with PCA

The explicit form of PCA can be expressed as: -

a1x1 + a2x2 + a3x3 + a4x4 + a5x5.............a17x17

where a1, a2, a3......a17 are the loadings that we found from eigen vectors

and x1, x2, x3, ...x17are 1st values of the 17 variables each.



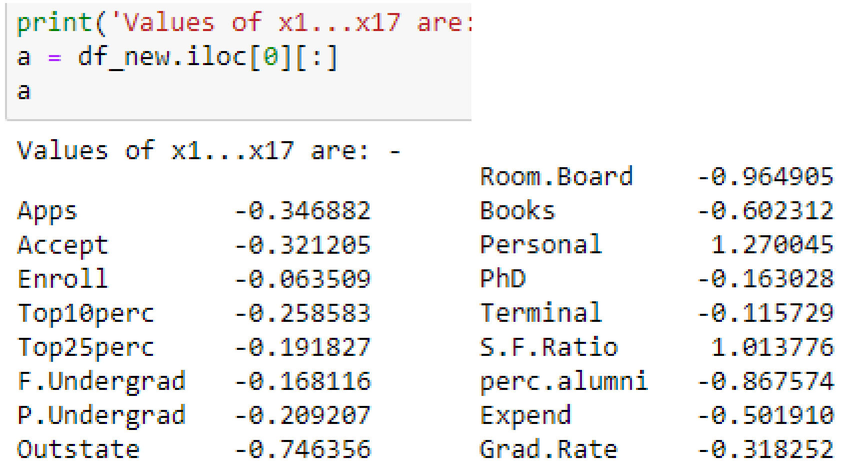
```
print('Values of the loadings are : -\n')
print(eig_vec.T[0])
```

```
Values of the loadings are : -

[-0.2487656  -0.2076015  -0.17630359 -0.35427395 -0.34400128 -0.15464096
 -0.0264425  -0.29473642 -0.24903045 -0.06475752  0.04252854 -0.31831287
 -0.31705602  0.17695789 -0.20508237 -0.31890875 -0.25231565]
```

The above values are the loadings or the coefficients (a1……...a17)

Similarly calculating values for x1……….x17, first values of each of the columns each.



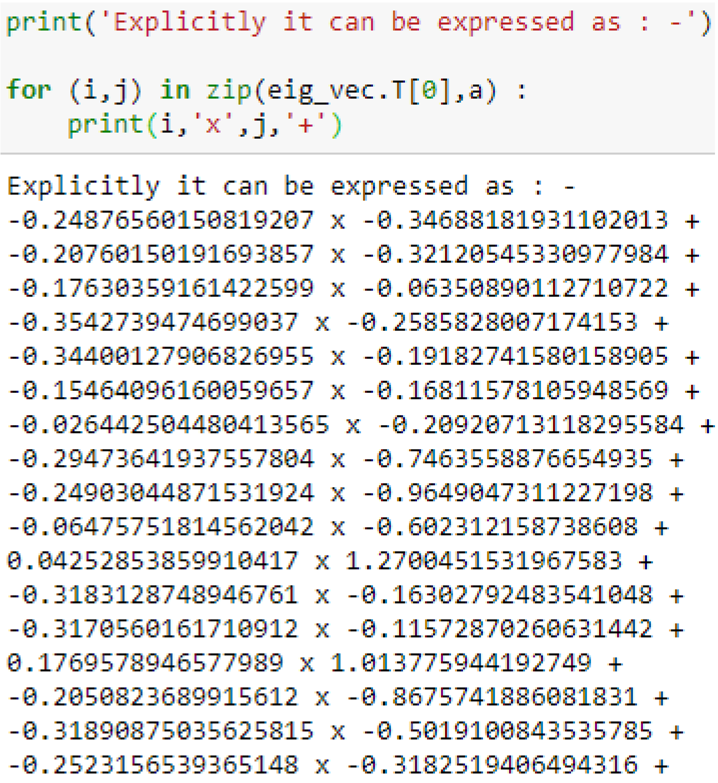
```
print('Values of x1...x17 are:
a = df_new.iloc[0][:]
a
```

```
Values of x1...x17 are: -
                                  Room.Board     -0.964905
Apps           -0.346882          Books          -0.602312
Accept         -0.321205          Personal        1.270045
Enroll         -0.063509          PhD            -0.163028
Top10perc      -0.258583          Terminal       -0.115729
Top25perc      -0.191827          S.F.Ratio       1.013776
F.Undergrad    -0.168116          perc.alumni    -0.867574
P.Undergrad    -0.209207          Expend         -0.501910
Outstate       -0.746356          Grad.Rate      -0.318252
```

The above values are the first values each for all the 17 columns.

To state the explicit form of the we can run two simultaneous for loops. One for the eigen value matrix containing loadings and one for the first values of each column each.

This turns out to be the following: -



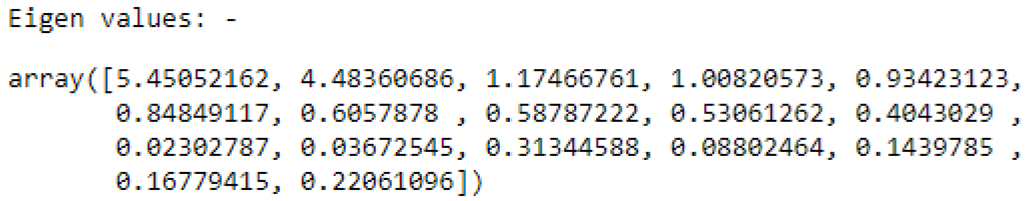
```
print('Explicitly it can be expressed as : -')

for (i,j) in zip(eig_vec.T[0],a) :
    print(i,'x',j,'+')
```

```
Explicitly it can be expressed as : -
-0.24876560150819207 x -0.34688181931102013 +
-0.20760150191693857 x -0.32120545330977984 +
-0.17630359161422599 x -0.06350890112710722 +
-0.3542739474699037 x -0.2585828007174153 +
-0.34400127906826955 x -0.19182741580158905 +
-0.15464096160059657 x -0.16811578105948569 +
-0.026442504480413565 x -0.20920713118295584 +
-0.29473641937557804 x -0.7463558876654935 +
-0.24903044871531924 x -0.9649047311227198 +
-0.06475751814562042 x -0.602312158738608 +
0.04252853859910417 x 1.2700451531967583 +
-0.3183128748946761 x -0.16302792483541048 +
-0.3170560161710912 x -0.11572870260631442 +
0.1769578946577989 x 1.013775944192749 +
-0.2050823689915612 x -0.8675741886081831 +
-0.31890875035625815 x -0.5019100843535785 +
-0.2523156539365148 x -0.3182519406494316 +
```

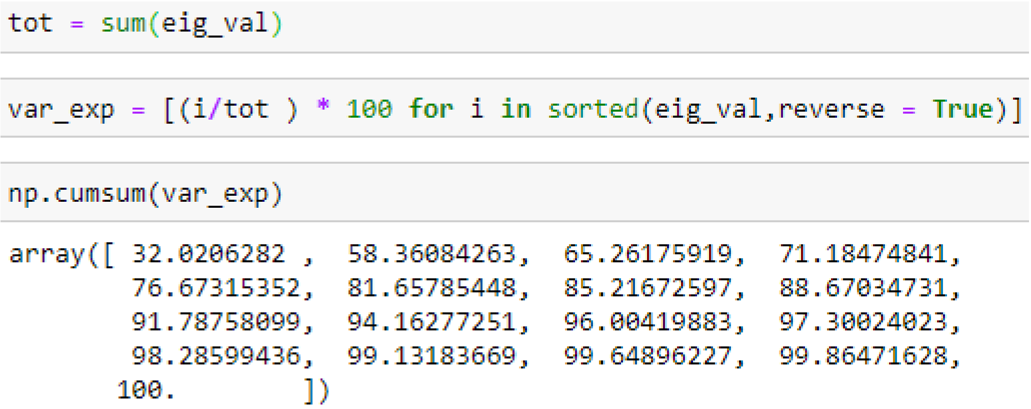
2.7) Discuss the cumulative values of the eigenvalues. How does it help you to decide on the optimum number of principal components? What do the eigenvectors indicate? Perform PCA and export the data of the Principal Component scores into a data frame.

The cumulative eigen values discuss the cumulative variance captured. We need to capture at-least 80 percent of variance in the data, less than that is not recommended. Eigen values help us decide how many principal components must be considered while data reduction. Following are the eigen values: -



```
Eigen values: -

array([5.45052162, 4.48360686, 1.17466761, 1.00820573, 0.93423123,
       0.84849117, 0.6057878 , 0.58787222, 0.53061262, 0.4043029 ,
       0.02302787, 0.03672545, 0.31344588, 0.08802464, 0.1439785 ,
       0.16779415, 0.22061096])
```



```
tot = sum(eig_val)
```

```
var_exp = [(i/tot ) * 100 for i in sorted(eig_val,reverse = True)]
```

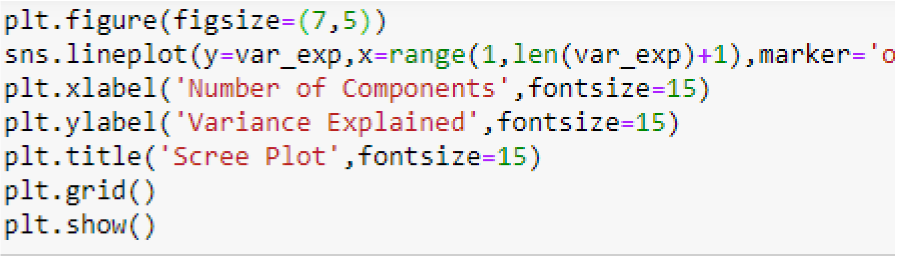
```
np.cumsum(var_exp)
```

```
array([ 32.0206282 ,  58.36084263,  65.26175919,  71.18474841,
        76.67315352,  81.65785448,  85.21672597,  88.67034731,
        91.78758099,  94.16277251,  96.00419883,  97.30024023,
        98.28599436,  99.13183669,  99.64896227,  99.86471628,
       100.        ])
```

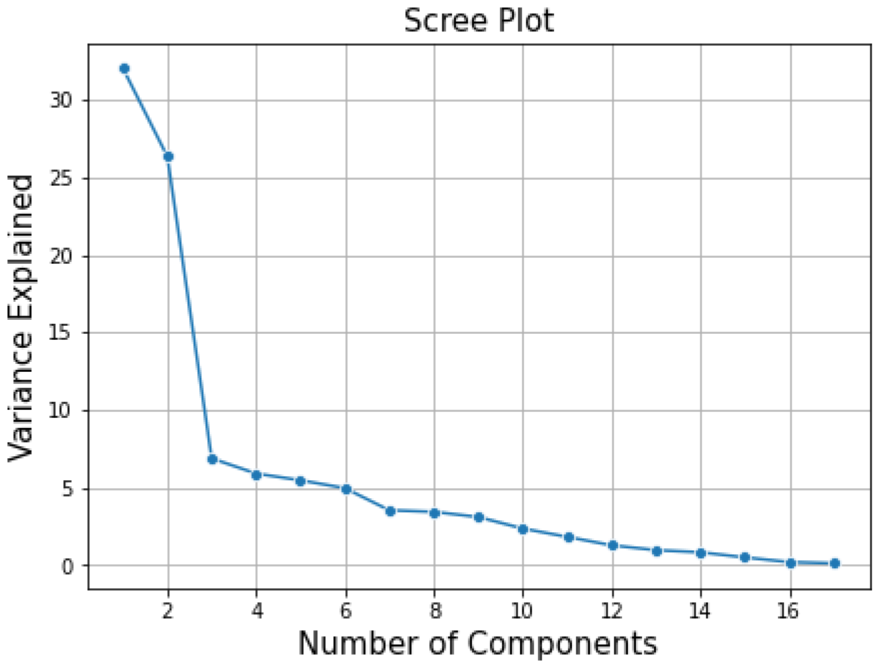
The cumulative variance captured by 6 components is more than 80, hence 6 variables are enough to contain the entire data set.

On the other hand eigen vectors consists of the loadings or coefficients for the PC equation which was explained just in the previous question.

Generating Scree Plot: -

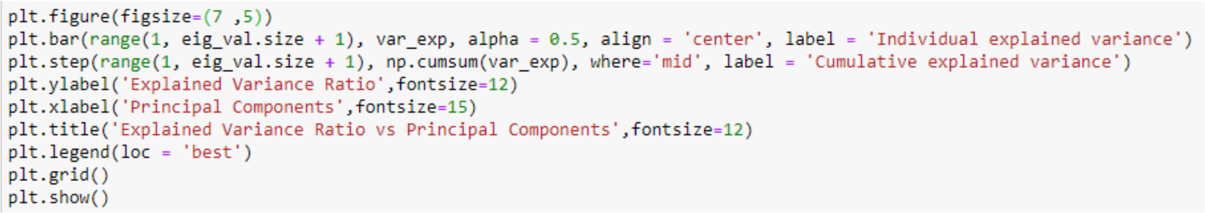


```
plt.figure(figsize=(7,5))
sns.lineplot(y=var_exp,x=range(1,len(var_exp)+1),marker='o
plt.xlabel('Number of Components',fontsize=15)
plt.ylabel('Variance Explained',fontsize=15)
plt.title('Scree Plot',fontsize=15)
plt.grid()
plt.show()
```

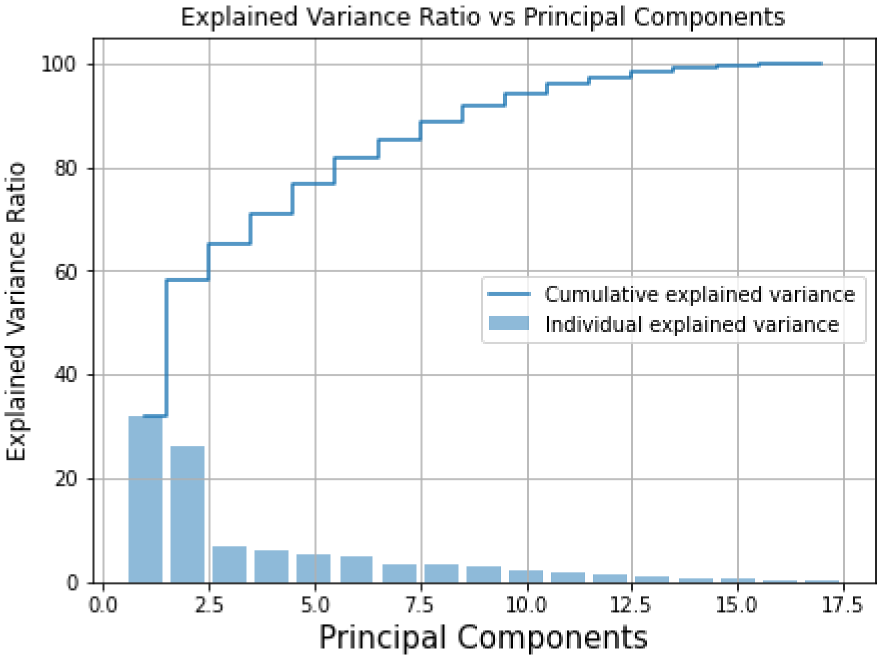


The sudden bend is also known as the elbow. Exactly at the 6th component, about 81.65% of variance is captured.

Generating the cumulative variance captured plot: -



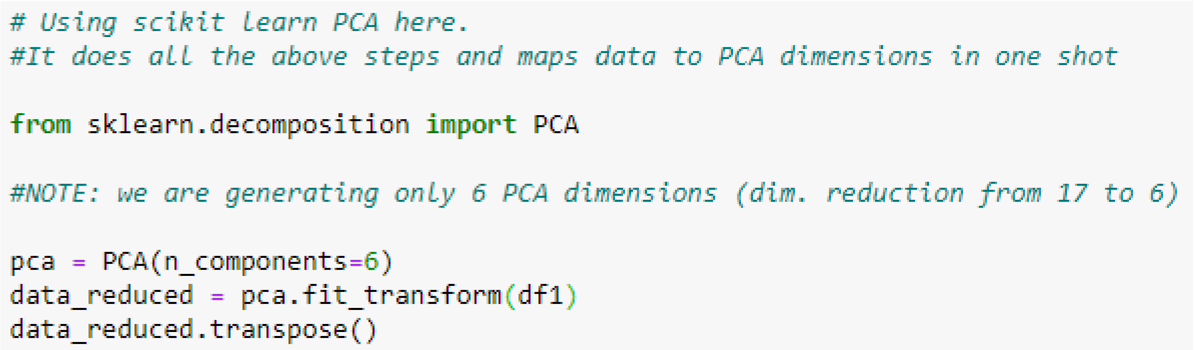
```
plt.figure(figsize=(7 ,5))
plt.bar(range(1, eig_val.size + 1), var_exp, alpha = 0.5, align = 'center', label = 'Individual explained variance')
plt.step(range(1, eig_val.size + 1), np.cumsum(var_exp), where='mid', label = 'Cumulative explained variance')
plt.ylabel('Explained Variance Ratio',fontsize=12)
plt.xlabel('Principal Components',fontsize=15)
plt.title('Explained Variance Ratio vs Principal Components',fontsize=12)
plt.legend(loc = 'best')
plt.grid()
plt.show()
```



About 80% of variance is captured by the 6th component.

PCA through SK Learn Library

PCA can be performed at a single go, through sk learn library.



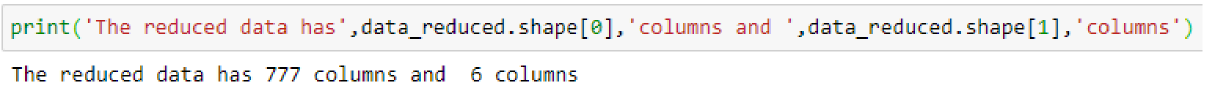
```
# Using scikit learn PCA here.
#It does all the above steps and maps data to PCA dimensions in one shot

from sklearn.decomposition import PCA

#NOTE: we are generating only 6 PCA dimensions (dim. reduction from 17 to 6)

pca = PCA(n_components=6)
data_reduced = pca.fit_transform(df1)
data_reduced.transpose()
```

The components sent as arguments = 6, which was decided based on the amount of variance captured, through the eigen value and eigen vector approach done previously.

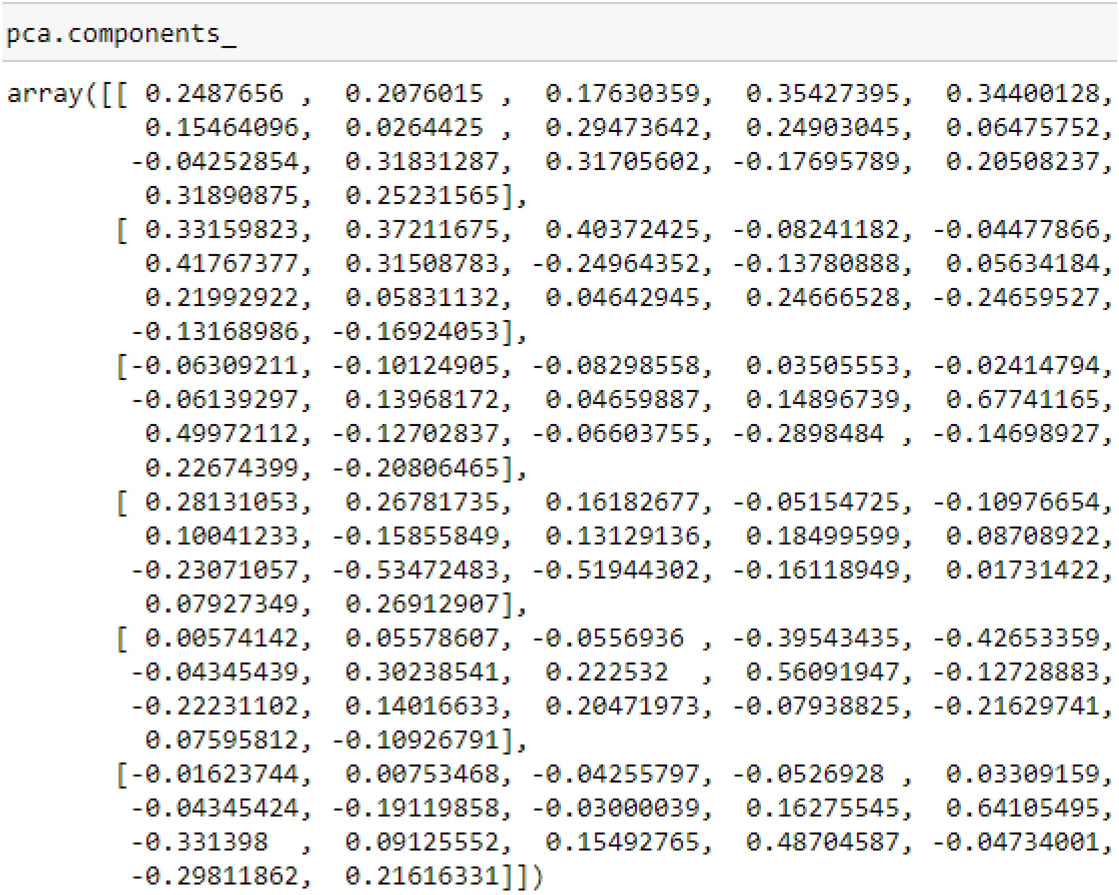


```
print('The reduced data has',data_reduced.shape[0],'columns and ',data_reduced.shape[1],'columns')
```

```
The reduced data has 777 columns and  6 columns
```

The data_reduced variable stores the Principal components for all the 6 columns and 777 rows.

Looking at the PC's: -

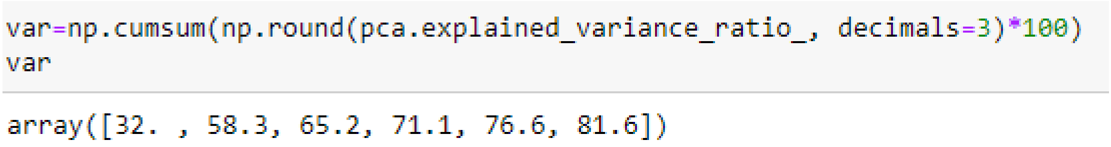


```
pca.components_
```

```
array([[ 0.2487656 ,  0.2076015 ,  0.17630359,  0.35427395,  0.34400128,
         0.15464096,  0.0264425 ,  0.29473642,  0.24903045,  0.06475752,
        -0.04252854,  0.31831287,  0.31705602, -0.17695789,  0.20508237,
         0.31890875,  0.25231565],
       [ 0.33159823,  0.37211675,  0.40372425, -0.08241182, -0.04477866,
         0.41767377,  0.31508783, -0.24964352, -0.13780888,  0.05634184,
         0.21992922,  0.05831132,  0.04642945,  0.24666528, -0.24659527,
        -0.13168986, -0.16924053],
       [-0.06309211, -0.10124905, -0.08298558,  0.03505553, -0.02414794,
        -0.06139297,  0.13968172,  0.04659887,  0.14896739,  0.67741165,
         0.49972112, -0.12702837, -0.06603755, -0.2898484 , -0.14698927,
         0.22674399, -0.20806465],
       [ 0.28131053,  0.26781735,  0.16182677, -0.05154725, -0.10976654,
         0.10041233, -0.15855849,  0.13129136,  0.18499599,  0.08708922,
        -0.23071057, -0.53472483, -0.51944302, -0.16118949,  0.01731422,
         0.07927349,  0.26912907],
       [ 0.00574142,  0.05578607, -0.0556936 , -0.39543435, -0.42653359,
        -0.04345439,  0.30238541,  0.222532  ,  0.56091947, -0.12728883,
        -0.22231102,  0.14016633,  0.20471973, -0.07938825, -0.21629741,
         0.07595812, -0.10926791],
       [-0.01623744,  0.00753468, -0.04255797, -0.0526928 ,  0.03309159,
        -0.04345424, -0.19119858, -0.03000039,  0.16275545,  0.64105495,
        -0.331398  ,  0.09125552,  0.15492765,  0.48704587, -0.04734001,
        -0.29811862,  0.21616331]])
```

Closely observing they are same as the eigen vectors, just with a difference of sign.

Checking the cumulative variance captured: -

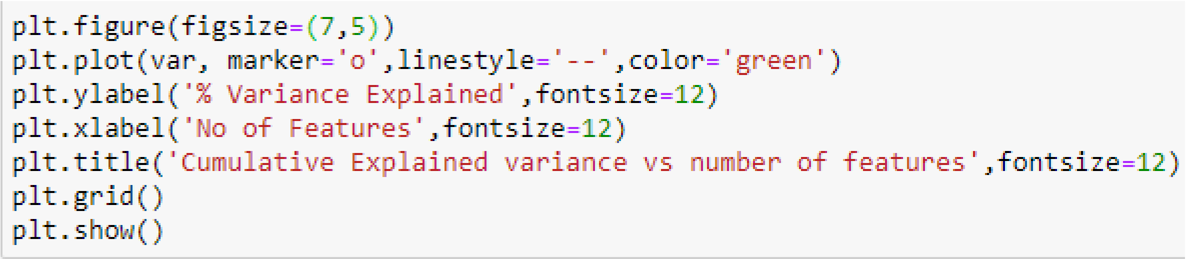


```
var=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=3)*100)
var
```

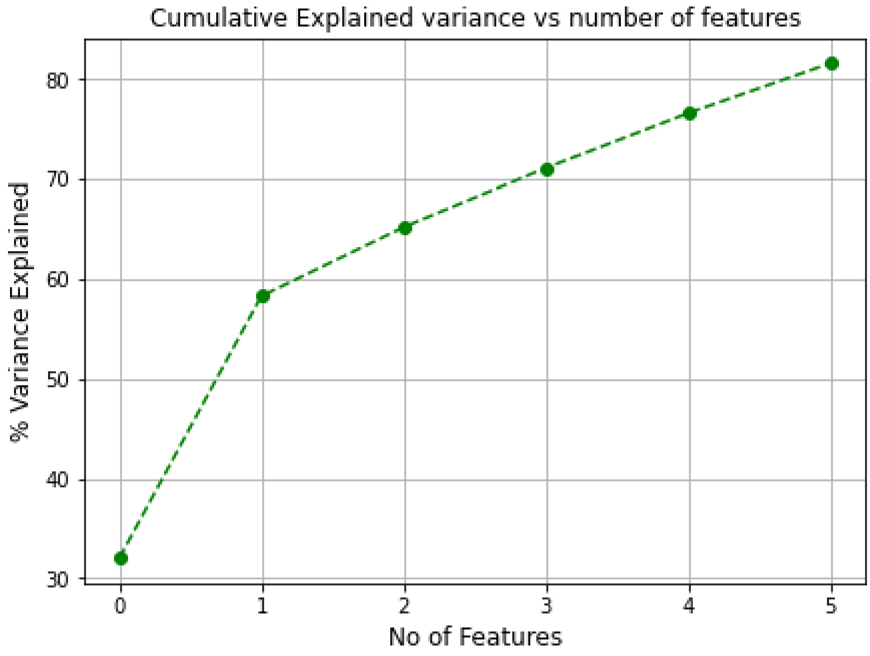
```
array([32. , 58.3, 65.2, 71.1, 76.6, 81.6])
```

This matches with our preliminary analysis as well. Capturing 81.6% of the data is good enough.

Generating the cum. Variance plot vs, no of components: -

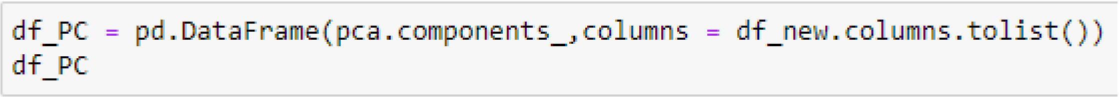


```
plt.figure(figsize=(7,5))
plt.plot(var, marker='o',linestyle='--',color='green')
plt.ylabel('% Variance Explained',fontsize=12)
plt.xlabel('No of Features',fontsize=12)
plt.title('Cumulative Explained variance vs number of features',fontsize=12)
plt.grid()
plt.show()
```

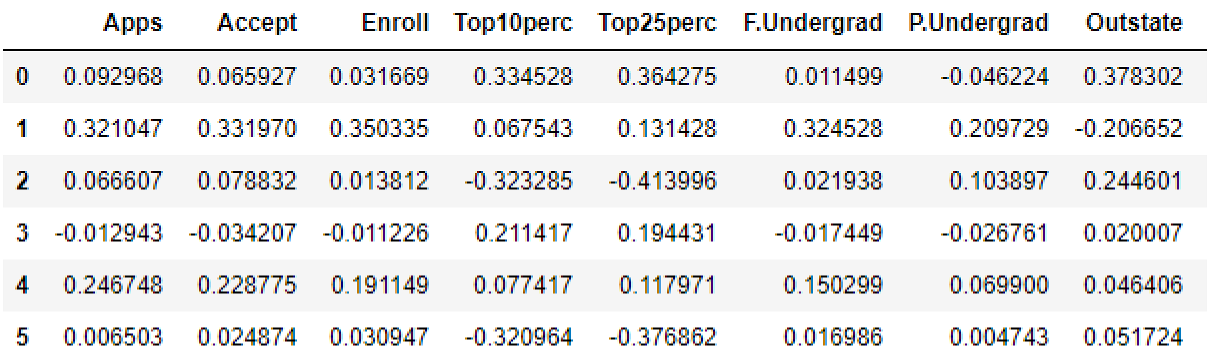


Exporting Data from PCA to a data frame: -

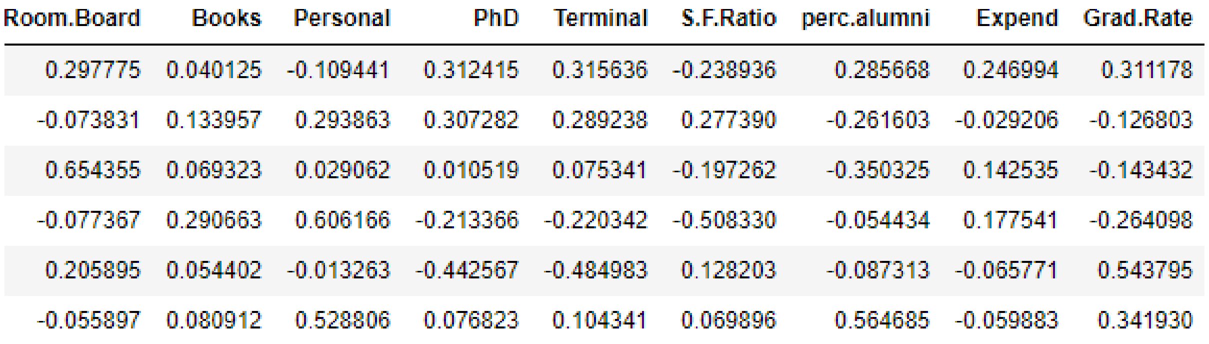
Following are the loadings of the variables captured in a data frame: -



```
df_PC = pd.DataFrame(pca.components_,columns = df_new.columns.tolist())
df_PC
```



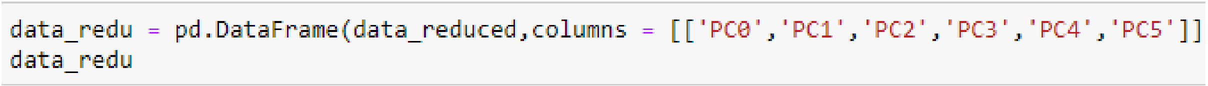
	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate
0	0.092968	0.065927	0.031669	0.334528	0.364275	0.011499	-0.046224	0.378302
1	0.321047	0.331970	0.350335	0.067543	0.131428	0.324528	0.209729	-0.206652
2	0.066607	0.078832	0.013812	-0.323285	-0.413996	0.021938	0.103897	0.244601
3	-0.012943	-0.034207	-0.011226	0.211417	0.194431	-0.017449	-0.026761	0.020007
4	0.246748	0.228775	0.191149	0.077417	0.117971	0.150299	0.069900	0.046406
5	0.006503	0.024874	0.030947	-0.320964	-0.376862	0.016986	0.004743	0.051724



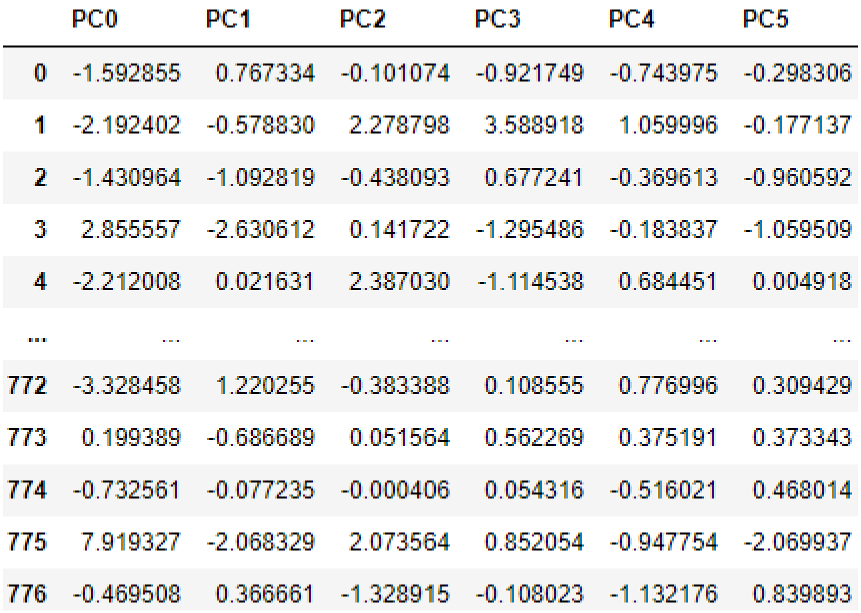
Room.Board	Books	Personal	PhD	Terminal	S.F.Ratio	perc.alumni	Expend	Grad.Rate
0.297775	0.040125	-0.109441	0.312415	0.315636	-0.238936	0.285668	0.246994	0.311178
-0.073831	0.133957	0.293863	0.307282	0.289238	0.277390	-0.261603	-0.029206	-0.126803
0.654355	0.069323	0.029062	0.010519	0.075341	-0.197262	-0.350325	0.142535	-0.143432
-0.077367	0.290663	0.606166	-0.213366	-0.220342	-0.508330	-0.054434	0.177541	-0.264098
0.205895	0.054402	-0.013263	-0.442567	-0.484983	0.128203	-0.087313	-0.065771	0.543795
-0.055897	0.080912	0.528806	0.076823	0.104341	0.069896	0.564685	-0.059883	0.341930

The above data frame shows the loadings or the coefficients of variables.

The values of Principal components were already stored in data_reduced, exporting them to a data frame: -



```
data_redu = pd.DataFrame(data_reduced,columns = [['PC0','PC1','PC2','PC3','PC4','PC5']]
data_redu
```



	PC0	PC1	PC2	PC3	PC4	PC5
0	-1.592855	0.767334	-0.101074	-0.921749	-0.743975	-0.298306
1	-2.192402	-0.578830	2.278798	3.588918	1.059996	-0.177137
2	-1.430964	-1.092819	-0.438093	0.677241	-0.369613	-0.960592
3	2.855557	-2.630612	0.141722	-1.295486	-0.183837	-1.059509
4	-2.212008	0.021631	2.387030	-1.114538	0.684451	0.004918
...	...	...	...	...	...	...
772	-3.328458	1.220255	-0.383388	0.108555	0.776996	0.309429
773	0.199389	-0.686689	0.051564	0.562269	0.375191	0.373343
774	-0.732561	-0.077235	-0.000406	0.054316	-0.516021	0.468014
775	7.919327	-2.068329	2.073564	0.852054	-0.947754	-2.069937
776	-0.469508	0.366661	-1.328915	-0.108023	-1.132176	0.839893

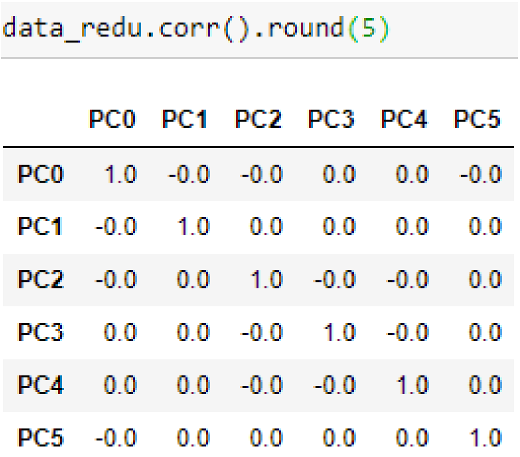
2.8) Mention the business implication of using the Principal Component Analysis for this case study. [Hint: Write Interpretations of the Principal Components Obtained]

PCA as we know is a dimensionality reduction technique. It compresses the data but at the same time retains max amount of information possible.

Performing Principal component analysis on the data we have been successfully able to reduce the number of numeric columns from 17 to 6, i.e. almost 1/3rd. At the same time, the variance captured is more than 80%, to be precise it is 81.65%.

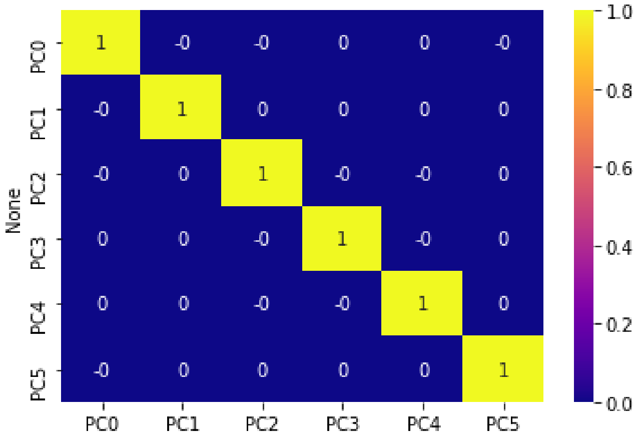
In a nutshell we have retained 81.65% of information, while compressing it to 1/3rd of its size which in itself is a feat of PCA.

Apart from that we were also able to get rid of the problem of multicollinearity. This can be seen from the correlation matrix, and looking at its heat map.



```
data_redu.corr().round(5)
```

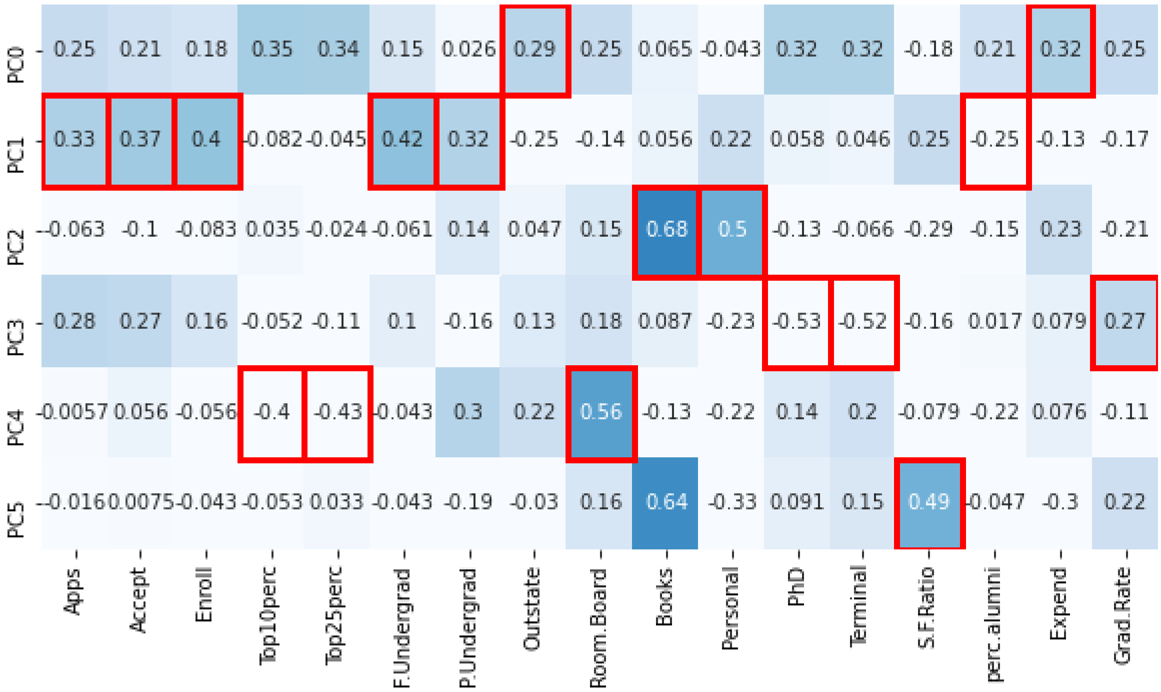
	PC0	PC1	PC2	PC3	PC4	PC5
PC0	1.0	-0.0	-0.0	0.0	0.0	-0.0
PC1	-0.0	1.0	0.0	0.0	0.0	0.0
PC2	-0.0	0.0	1.0	-0.0	-0.0	0.0
PC3	0.0	0.0	-0.0	1.0	-0.0	0.0
PC4	0.0	0.0	-0.0	-0.0	1.0	0.0
PC5	-0.0	0.0	0.0	0.0	0.0	1.0



Apart from the diagonal elements every other element is zero, indicating the elements are now independent of each other.

Moving ahead, lets try to interpret the principal components: -

The loadings are represented in a heatmap below, with the max loadings being highlighted. (The figure is highlighted by importing Rectangle from Matplotlib)



	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	PhD	Terminal	S.F.Ratio	perc.alumni	Expend	Grad.Rate
PC0	0.25	0.21	0.18	0.35	0.34	0.15	0.026	0.29	0.25	0.065	-0.043	0.32	0.32	-0.18	0.21	0.32	0.25
PC1	0.33	0.37	0.4	-0.082	-0.045	0.42	0.32	-0.25	-0.14	0.056	0.22	0.058	0.046	0.25	-0.25	-0.13	-0.17
PC2	-0.063	-0.1	-0.083	0.035	-0.024	-0.061	0.14	0.047	0.15	0.68	0.5	-0.13	-0.066	-0.29	-0.15	0.23	-0.21
PC3	0.28	0.27	0.16	-0.052	-0.11	0.1	-0.16	0.13	0.18	0.087	-0.23	-0.53	-0.52	-0.16	0.017	0.079	0.27
PC4	-0.0057	0.056	-0.056	-0.4	-0.43	-0.043	0.3	0.22	0.56	-0.13	-0.22	0.14	0.2	-0.079	-0.22	0.076	-0.11
PC5	-0.016	0.0075	-0.043	-0.053	0.033	-0.043	-0.19	-0.03	0.16	0.64	-0.33	0.091	0.15	0.49	-0.047	-0.3	0.22

Observing the colour indicators, max variance is captured by PC0, followed by the rest. Comparing the loadings, across all the variables, PC1 has the max no of highest loadings. If given a rule to label these components, we could probably identify a pattern and use unsupervised learning algorithms for prediction purposes. We fall short of knowledge to do the same at this point of time. However, we can make basic interpretations

PC0: Max loading for Outstate and Expend

PC1: Max loading for Apps, Accept, Enrol, F. Undergrad, P. Undergrad, Perc.Alumini.

PC2: Max loading for Books and Personal

PC3: Max loading for PhD, Terminal, Grad Rate

PC4: Top 10 perc, Top 25 perc, Room Board

PC5: SF Ratio

In general, PC1 can be related to students, PC2 up-to some extent to Expenditure, PC3 seems more of inclined towards brilliance in academia of Faculties, PC4 relates to the most brilliant student minds, PC5 relates to student faculty ratio.

However, there is no clear rule mentioned to tag these components.