Problem 1A:

Salary is hypothesized to depend on educational qualification and occupation. To understand the dependency, the salaries of 40 individuals [SalaryData.csv] are collected and each person's educational qualification and occupation are noted. Educational qualification is at three levels, High school graduate, Bachelor, and Doctorate. Occupation is at four levels, Administrative and clerical, Sales, Professional or specialty, and Executive or managerial. A different number of observations are in each level of education – occupation combination.

[Assume that the data follows a normal distribution. In reality, the normality assumption may not always hold if the sample size is small.]

First, Importing all the necessary libraries :

```
In [1]: import os
        import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        %matplotlib inline
        from statsmodels.formula.api import ols
        from statsmodels.stats.anova import _get_covariance, anova_lm
```

```
In [2]: #checking the current working directory:

        os.getcwd()

Out[2]: 'C:\\Users\\user'
```

```
In [3]: #changing the current working directory where the file is placed:

        os.chdir('D:\\SHUBHANK !\\GL\\4. TOPIC 3 - Advanced Statistics')
```

```
In [4]: os.getcwd()

Out[4]: 'D:\\SHUBHANK !\\GL\\4. TOPIC 3 - Advanced Statistics'
```

Loading the 'SalaryData.csv' file:

```
In [5]: df = pd.read_csv('SalaryData.csv')
```

In [6]:
```
# Checking the top records from the file:

df.head()
```

Out[6]:

	Education	Occupation	Salary
0	Doctorate	Adm-clerical	153197
1	Doctorate	Adm-clerical	115945
2	Doctorate	Adm-clerical	175935
3	Doctorate	Adm-clerical	220754
4	Doctorate	Sales	170769

In [7]:
```
# Checking the bottom records from the file:

df.tail()
```

Out[7]:

	Education	Occupation	Salary
35	Bachelors	Exec-managerial	173935
36	Bachelors	Exec-managerial	212448
37	Bachelors	Exec-managerial	173664
38	Bachelors	Exec-managerial	212760
39	Doctorate	Exec-managerial	212781

In [8]:
```
# Checking the five number summary and the other important description of the
 data:

df.describe()
```

Out[8]:

	Salary
count	40.000000
mean	162186.875000
std	64860.407506
min	50103.000000
25%	99897.500000
50%	169100.000000
75%	214440.750000
max	260151.000000

In [9]:
```
# Checking the shape of the data:

df.shape
```

Out[9]:
```
(40, 3)
```

In [10]:
```
# Checking the info:

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Education   40 non-null     object
 1   Occupation  40 non-null     object
 2   Salary      40 non-null     int64 
dtypes: int64(1), object(2)
memory usage: 1.1+ KB
```

In [11]:
```
# Checking the distinct counts of different eductaion and occupation:

df.Education.value_counts()
```

Out[11]:
```
 Doctorate    16
 Bachelors    15
 HS-grad       9
Name: Education, dtype: int64
```

We can see above that there are 3 kind of education: Doctorate, Bachelors and HS-grad.

In [12]:
```
df.Occupation.value_counts()
```

Out[12]:
```
 Prof-specialty     13
 Sales              12
 Adm-clerical       10
 Exec-managerial     5
Name: Occupation, dtype: int64
```

We can see that there are 4 kind of occupation: Prof-speciality, Sales, Adm-clerical, Exec-managerial.

Q1. State the null and the alternate hypothesis for conducting one-way ANOVA for both Education and Occupation individually.

Answer:

Null and Alternate hypothesis for Education:

Null Hypothesis(H0): The mean salary of all the 40 individuals is equal at all education level.

Alternate Hypothesis(Ha): The mean salary of all the 40 individuals are different for atleast one kind of education.

Null and Alternate hypothesis for Occupation:

Null Hypothesis(H0): The mean salary of all the 40 individuals is equal for all kind of Occupation.

Alternate Hypothesis(Ha): The mean salary of all the 40 individuals are different for atleast one kind of Occupation.

Q2. Perform a one-way ANOVA on Salary with respect to Education. State whether the null hypothesis is accepted or rejected based on the ANOVA results.

In [13]:
```
# One-way Anova on Salary with respect to Education:

formula = 'Salary ~ C(Education)'
model = ols(formula, df).fit()
aov_table = anova_lm(model)
print(aov_table)
```

```
                df        sum_sq       mean_sq          F        PR(>F)
C(Education)   2.0  1.026955e+11  5.134773e+10  30.95628  1.257709e-08
Residual      37.0  6.137256e+10  1.658718e+09       NaN           NaN
```

There are different kind of education which is influencing the salary of every individuals.

Variance in the salary caused by different education level.

Difference in salary of few individuals is because of the difference in their education.

So now, we can see that the p-value is less than the significance level(0.05), hence we can reject the null hypothesis and conclude that the mean salary is different for atleast one of the individual.

Q3. Perform a one-way ANOVA on Salary with respect to Occupation. State whether the null hypothesis is accepted or rejected based on the ANOVA results.

In [14]:
```
# One-way Anova on Salary with respect to Occupation:

formula = 'Salary ~ C(Occupation)'
model = ols(formula, df).fit()
aov_table = anova_lm(model)
print(aov_table)
```

```
                 df        sum_sq       mean_sq         F    PR(>F)
C(Occupation)   3.0  1.125878e+10  3.752928e+09  0.884144  0.458508
Residual       36.0  1.528092e+11  4.244701e+09       NaN       NaN
```

So now, we can see that the p-value(0.45 as above) is greater than the significance level(0.05), hence, in this case we fail to reject the null hypothesis.

Problem 1B:

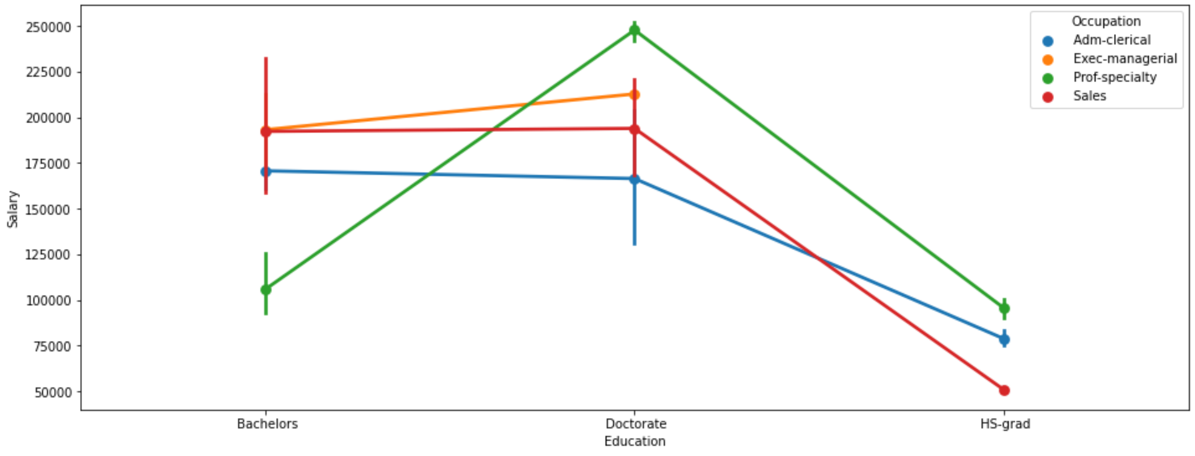
Q1. What is the interaction between two treatments? Analyze the effects of one variable on the other (Education and Occupation) with the help of an interaction plot.[hint: use the ‘pointplot’ function from the ‘seaborn’ function]

In [15]:
```
df['Education'] = pd.Categorical(df['Education'])
df['Occupation'] = pd.Categorical(df['Occupation'])

formula = 'Salary ~ C(Education) + C(Occupation)'
model = ols(formula, df).fit()
aov_table = anova_lm(model)
print(aov_table)

#pd.set_option('display.float_format', '{:.2f}'.format)
```

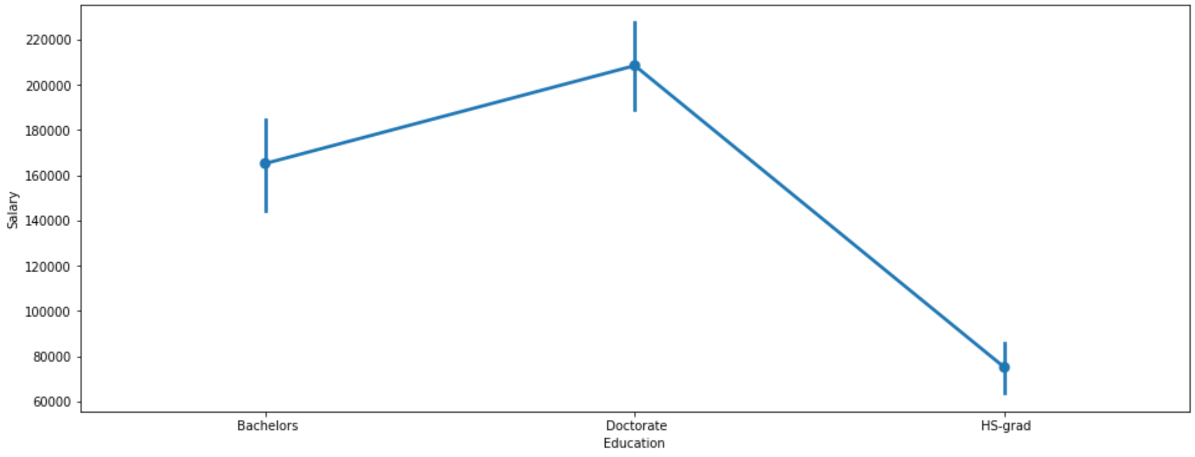
```
                 df        sum_sq       mean_sq          F        PR(>F)
C(Education)    2.0  1.026955e+11  5.134773e+10  31.257677  1.981539e-08
C(Occupation)   3.0  5.519946e+09  1.839982e+09   1.120080  3.545825e-01
Residual       34.0  5.585261e+10  1.642724e+09        NaN           NaN
```

We can see that the p-value in one of the treatments is greater than alpha(0.05).

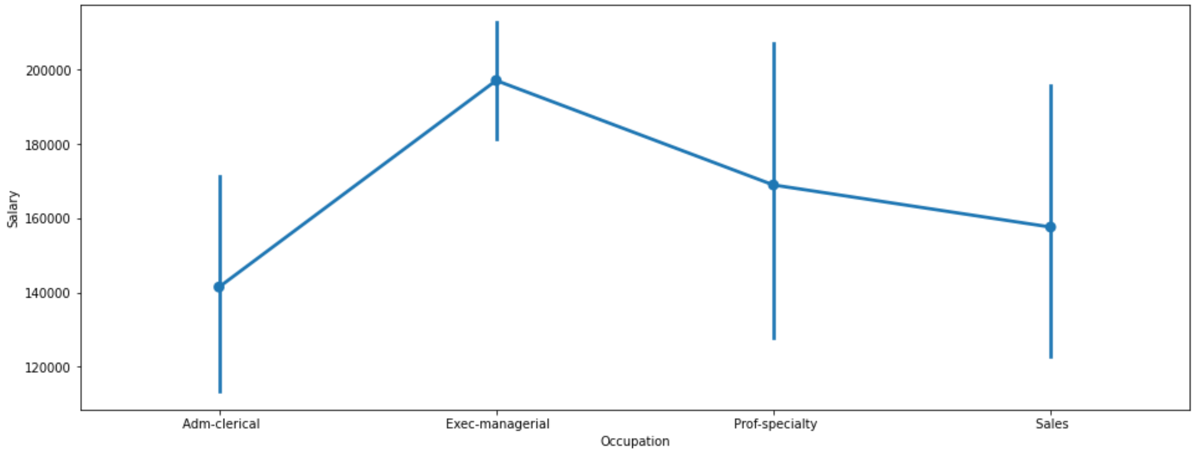
In [16]:
```
# Analyzing the effects of one variable on the other (Education and Occupatio
n) with the help of an interaction plot.


# first, analyzing the effect of Occupation on Education and how it is impacti
ng the salary:
plt.figure(figsize=(16, 6))
sns.pointplot(x='Education', y='Salary', hue='Occupation', data=df,)
plt.show()
```



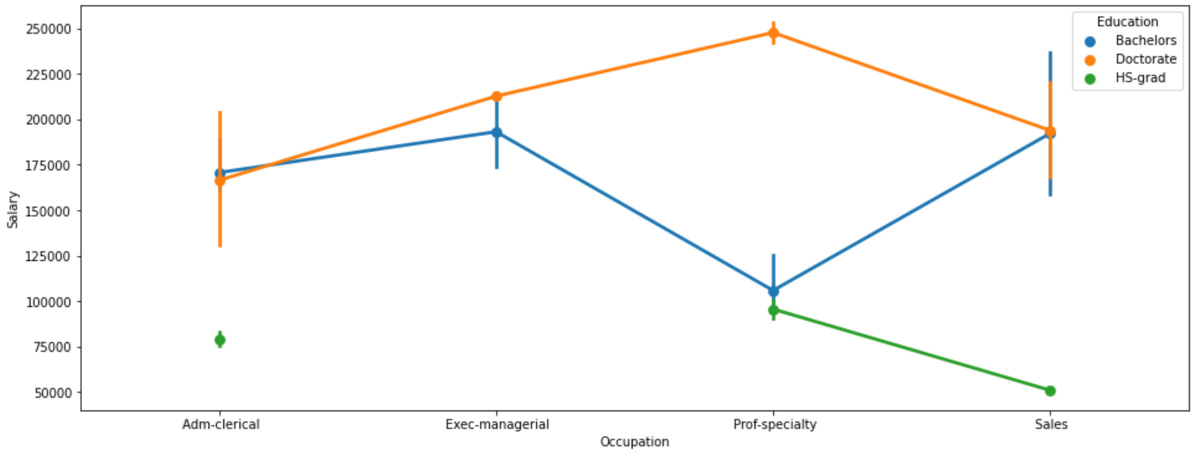
In [134]:
```
#we can also see the effect of education level on salary:

plt.figure(figsize=(16, 6))
sns.pointplot(x='Education', y='Salary', data=df,)
plt.show()
```



In [135]:
```
#we can also see the effect of occupation on salary:

plt.figure(figsize=(16, 6))
sns.pointplot(x='Occupation', y='Salary', data=df,)
plt.show()
```



In [17]:
```
# Now, analyzing the effect of Education on Occupation and how it is impacting
the salary:

plt.figure(figsize=(16, 6))
sns.pointplot(x='Occupation', y='Salary', hue='Education', data=df,)
plt.show()
```



Q2. Perform a two-way ANOVA based on Salary with respect to both Education and Occupation (along with their interaction Education*Occupation). State the null and alternative hypotheses and state your results. How will you interpret this result?

In [18]:
```
formula = 'Salary ~ C(Education) + C(Occupation) + C(Education):C(Occupation)'
model = ols(formula, df).fit()
aov_table = anova_lm(model)

aov_table
```

Out[18]:

	df	sum_sq	mean_sq	F	PR(>F)
C(Education)	2.0	1.026955e+11	5.134773e+10	72.211958	5.466264e-12
C(Occupation)	3.0	5.519946e+09	1.839982e+09	2.587626	7.211580e-02
C(Education):C(Occupation)	6.0	3.634909e+10	6.058182e+09	8.519815	2.232500e-05
Residual	29.0	2.062102e+10	7.110697e+08	NaN	NaN

We can see the little change in p-value of "occupation" without the interaction effect.

Here p-value is less than significance value(0.05) for Education, that means we can reject the null hypothesis.

There is very minor change in p-value of Occupation that is greater than significance value(0.05), so we fail to reject the null hypothesis.

The impact on dependent variable 'Salary' is much due to the 'Education' and the joint interaction effect of 'education' and 'occupation' together.

Q3. Explain the business implications of performing ANOVA for this particular case study.

Answer:

The impact on dependent variable 'Salary' is much due to the 'Education' and the joint interaction effect of 'education' and 'occupation' together.

The combined effect of education and occupation together makes a great impact on individuals salary.

Problem 2:

The dataset Education - Post 12th Standard.csv contains information on various colleges. You are expected to do a Principal Component Analysis for this case study according to the instructions given. The data dictionary of the 'Education - Post 12th Standard.csv' can be found in the following file: Data Dictionary.xlsx.

In [71]:
```
# Loading the dataset:

df1 = pd.read_csv('Education+-+Post+12th+Standard.csv')
```

Q. Perform Exploratory Data Analysis [both univariate and multivariate analysis to be performed]. What insight do you draw from the EDA?

Performing the basic EDA steps as below:

In [72]:
```
# Head of the dataset

df1.head()
```

Out[72]:

	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

In [73]:
```
# Shape of the dataset:

df1.shape
```

Out[73]:
```
(777, 18)
```

As we can see that dataset has 777 rows and 18 columns.

In [74]:
```
# Info of the dataset:

df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 777 entries, 0 to 776
Data columns (total 18 columns):
 #   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  
dtypes: float64(1), int64(16), object(1)
memory usage: 109.4+ KB
```

As we can see that there are 777 non-null rows, 1 column of object data type, 1 column of float data type and 16 columns of integer data type in the dataset.

In [136]:
```
# Summary of the dataset:

df1.describe().T
```

Out[136]:

	count	mean	std	min	25%	50%	75%	max
Apps	777.0	3001.638353	3870.201484	81.0	776.0	1558.0	3624.0	48094.0
Accept	777.0	2018.804376	2451.113971	72.0	604.0	1110.0	2424.0	26330.0
Enroll	777.0	779.972973	929.176190	35.0	242.0	434.0	902.0	6392.0
Top10perc	777.0	27.558559	17.640364	1.0	15.0	23.0	35.0	96.0
Top25perc	777.0	55.796654	19.804778	9.0	41.0	54.0	69.0	100.0
F.Undergrad	777.0	3699.907336	4850.420531	139.0	992.0	1707.0	4005.0	31643.0
P.Undergrad	777.0	855.298584	1522.431887	1.0	95.0	353.0	967.0	21836.0
Outstate	777.0	10440.669241	4023.016484	2340.0	7320.0	9990.0	12925.0	21700.0
Room.Board	777.0	4357.526384	1096.696416	1780.0	3597.0	4200.0	5050.0	8124.0
Books	777.0	549.380952	165.105360	96.0	470.0	500.0	600.0	2340.0
Personal	777.0	1340.642214	677.071454	250.0	850.0	1200.0	1700.0	6800.0
PhD	777.0	72.660232	16.328155	8.0	62.0	75.0	85.0	103.0
Terminal	777.0	79.702703	14.722359	24.0	71.0	82.0	92.0	100.0
S.F.Ratio	777.0	14.089704	3.958349	2.5	11.5	13.6	16.5	39.8
perc.alumni	777.0	22.797941	12.338089	1.0	13.0	21.0	31.0	64.0
Expend	777.0	9660.171171	5221.768440	3186.0	6751.0	8377.0	10830.0	56233.0
Grad.Rate	777.0	65.463320	17.177710	10.0	53.0	65.0	78.0	118.0

Observation:

1. Minimum No. of applications received are 81 and the maximum are 48094

2. Minimum No. of applications accepted are 72 and the maximum are 26330

3. Mean cost for room and board comes out to be 4357 rupees.

4. Maximum cost of the books for a student is 2340 rupees.

In [76]:
```
# Checking for duplicate records:

dup_rec = df1.duplicated()

print('--------------------')

print('Number of duplicate records = %d' %(dup_rec.sum()))
```

```
--------------------
Number of duplicate records = 0
```

In [77]:
```
# Checking for missing values:

df1.isnull().sum()
```

Out[77]:
```
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
```

As we can see that there are no missing values in the dataset.

In [78]:
```
(df1 == 0).sum()
```

Out[78]:
```
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    2
Expend         0
Grad.Rate      0
dtype: int64
```

As we can see that, only 'perc.alumni' column has 2 zeros.

In [80]:
```
# Replacing the 2 zeros in 'perc.alumni' field with it's median value:

df1['perc.alumni'].replace(to_replace=0, value=df1['perc.alumni'].median(),inp
lace=True)
```

In [81]:
```
# again checking zeros

(df1 == 0).sum()
```

Out[81]:
```
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
```

Now no zeros in the dataset.

Univariate Analysis:

In [83]:
```
# for doing univariate analysis for various numeric variables, we will define
 a function:
# Here we will use this function to show the histogram to display the distribu
tion
# and the box plot to show the five number summary and the outliers present in
the dataset.

def UniAnalysis(column,nbins):
    plt.figure()
    print('Below is the distribution of :'+column)
    sns.distplot(df1[column], kde=False, color='blue');
    plt.show()

    print('**********************************')

    plt.figure()
    print('Below is the box plot of :'+ column)
    ax = sns.boxplot(x=df1[column])
    plt.show()
```

In [86]:
```
df1_numerical = df1.select_dtypes(include = ['float64', 'int64'])


df1_numerical
```

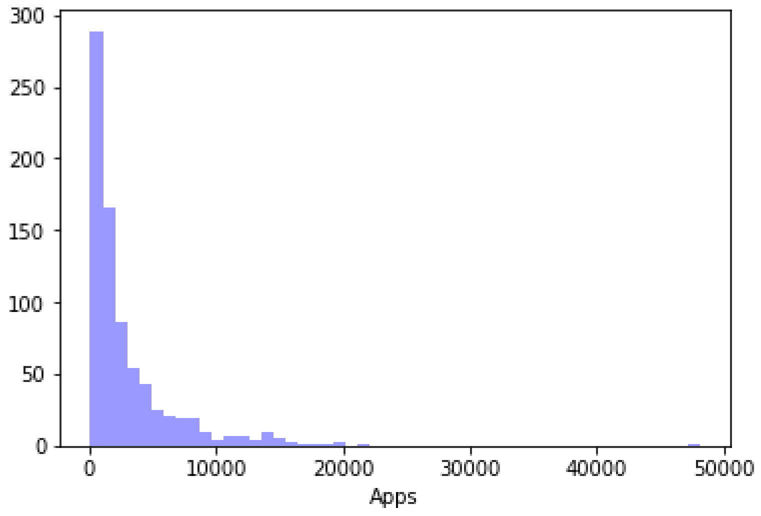
Out[86]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.B
0	1660	1232	721	23	52	2885	537	7440	3
1	2186	1924	512	16	29	2683	1227	12280	6
2	1428	1097	336	22	50	1036	99	11250	3
3	417	349	137	60	89	510	63	12960	5
4	193	146	55	16	44	249	869	7560	4
...	...	...	...	...	...	...	...	...	
772	2197	1515	543	4	26	3089	2029	6797	3
773	1959	1805	695	24	47	2849	1107	11520	4
774	2097	1915	695	34	61	2793	166	6900	4
775	10705	2453	1317	95	99	5217	83	19840	6
776	2989	1855	691	28	63	2988	1726	4990	3

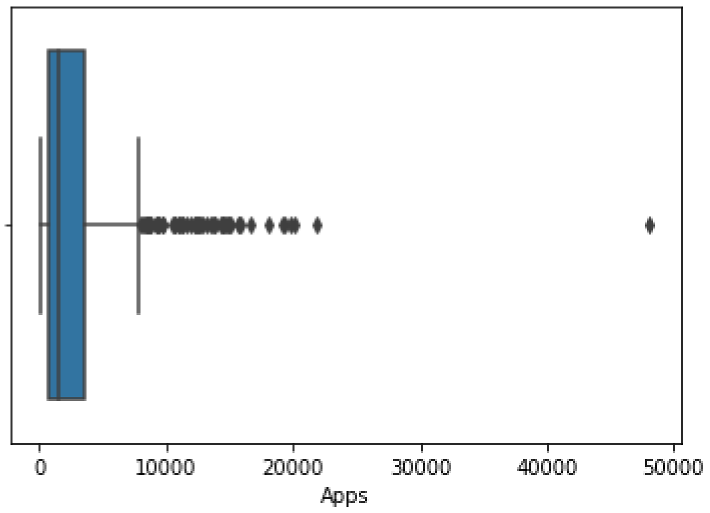
777 rows × 17 columns

In [87]:
```
list_of_numerical_columns = list(df1_numerical.columns.values)

for x in list_of_numerical_columns:
    UniAnalysis(x,20)
```

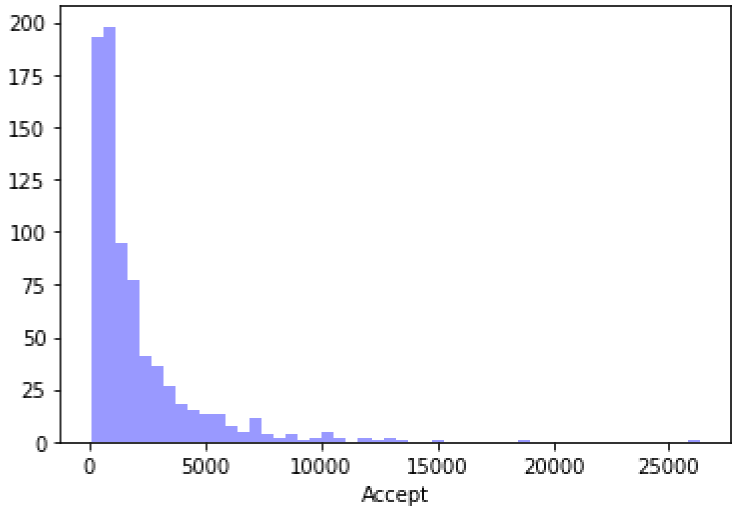
```
Below is the distribution of :Apps
```



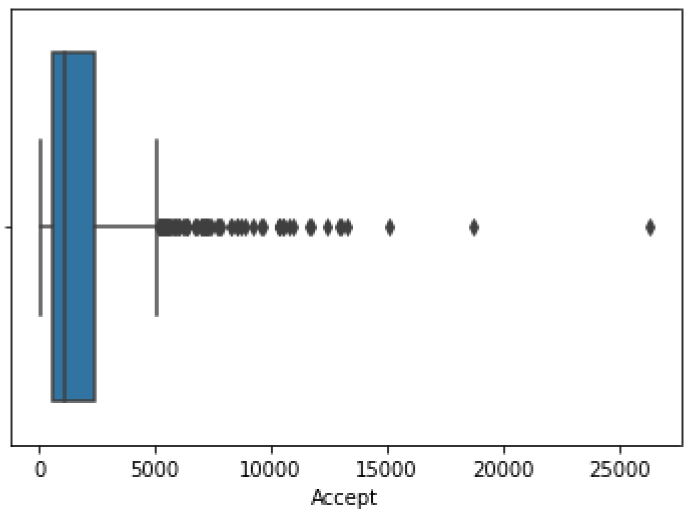
```
*********************************
Below is the box plot of :Apps
```



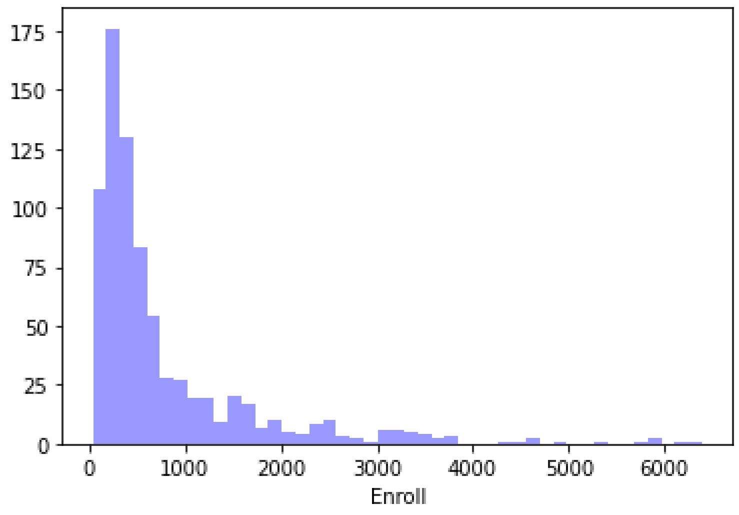
```
Below is the distribution of :Accept
```



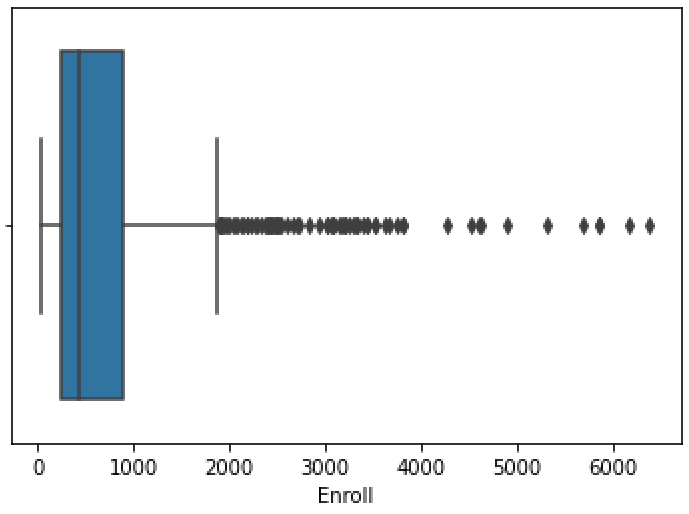
```
*********************************
Below is the box plot of :Accept
```



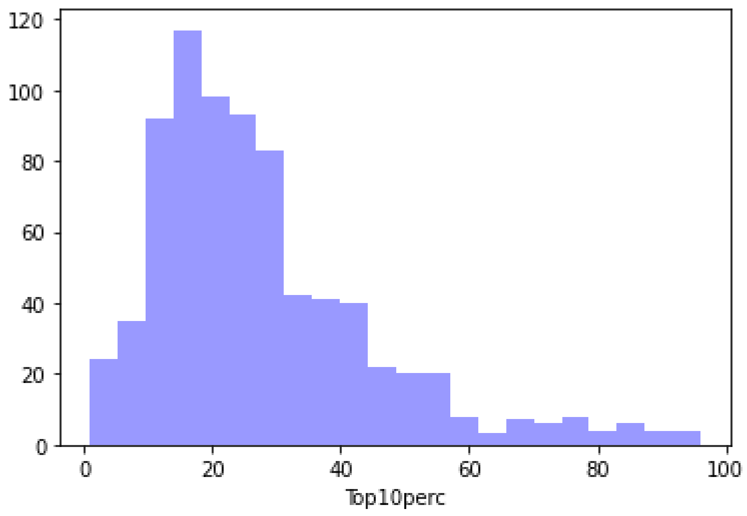
```
Below is the distribution of :Enroll
```



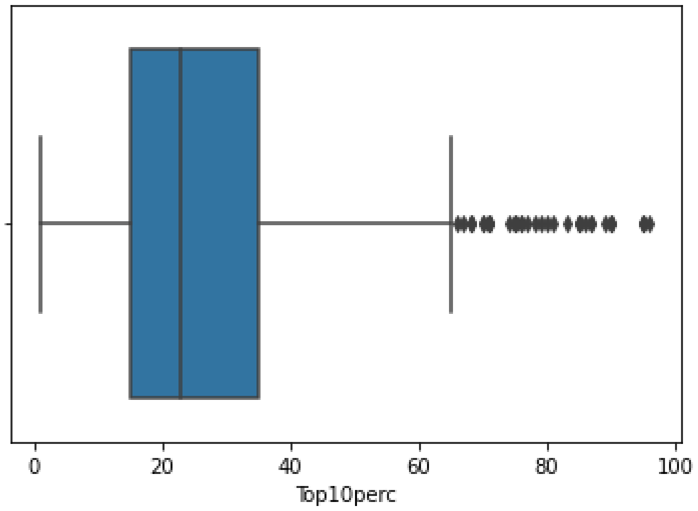
```
*********************************
Below is the box plot of :Enroll
```



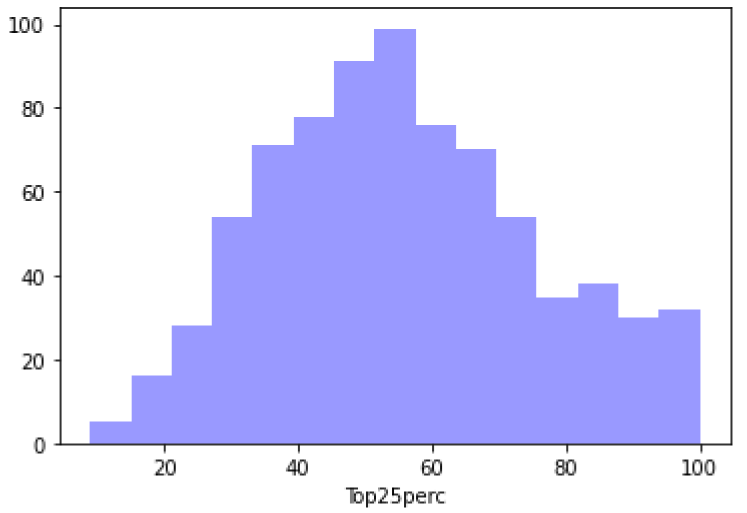
```
Below is the distribution of :Top10perc
```



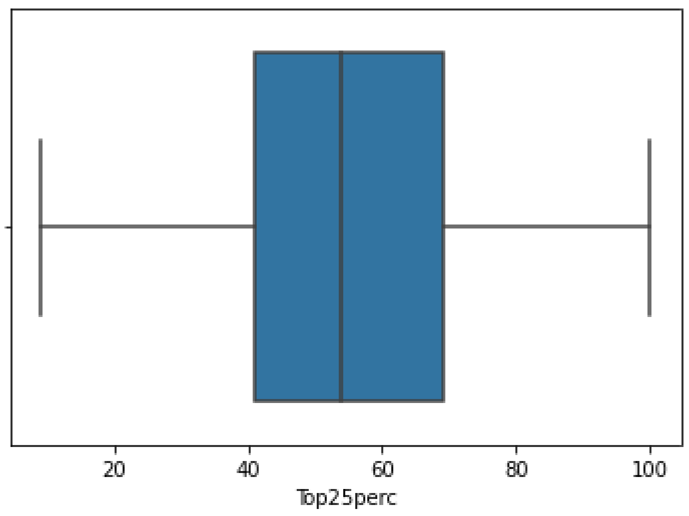
```
**********************************
Below is the box plot of :Top10perc
```



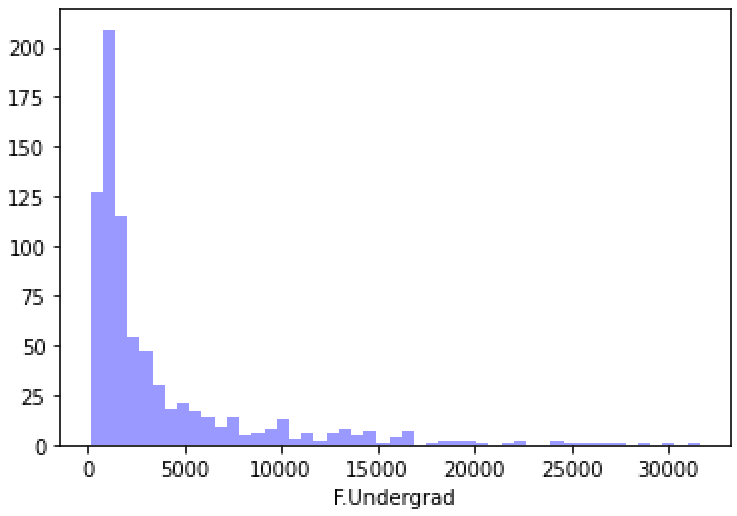
```
Below is the distribution of :Top25perc
```



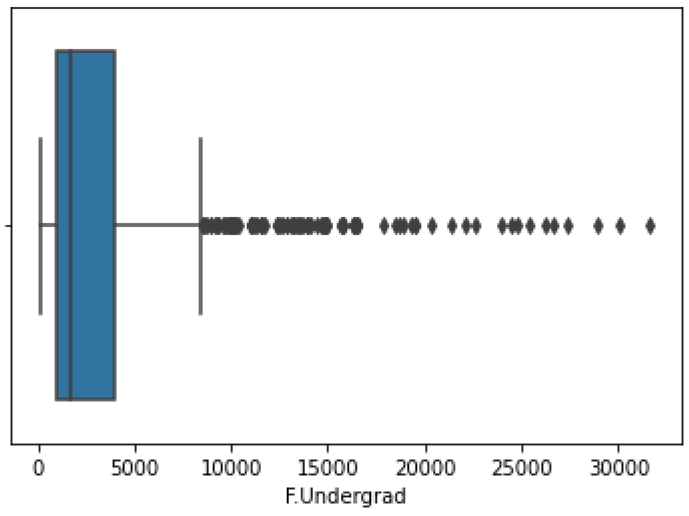
```
**********************************
Below is the box plot of :Top25perc
```



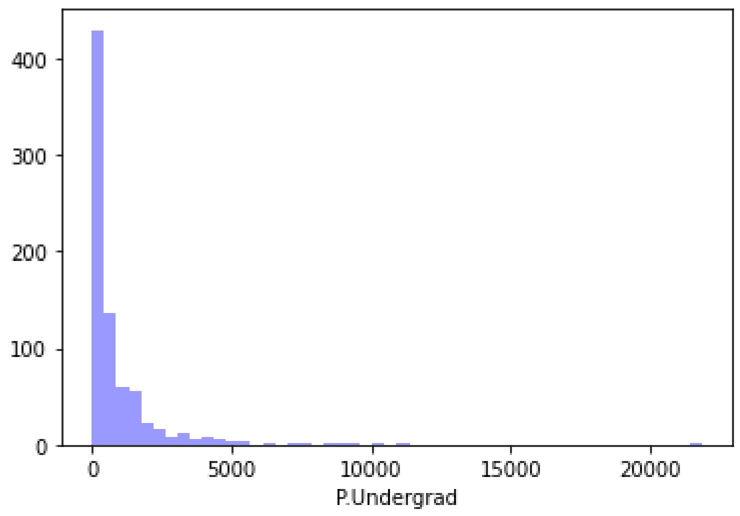
```
Below is the distribution of :F.Undergrad
```



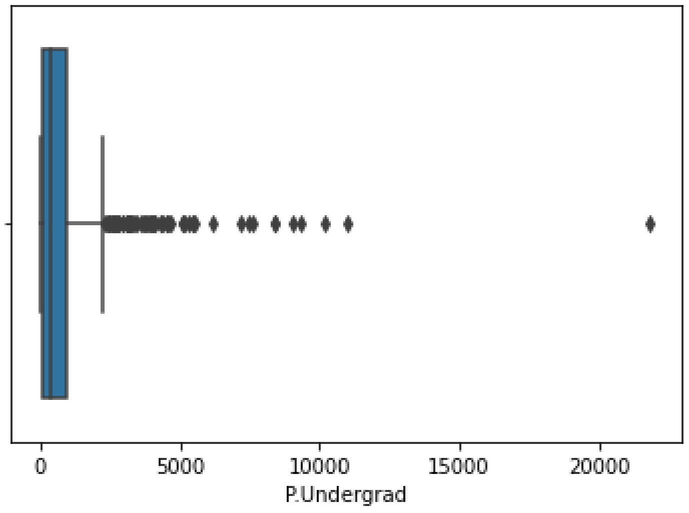
```
**********************************
Below is the box plot of :F.Undergrad
```



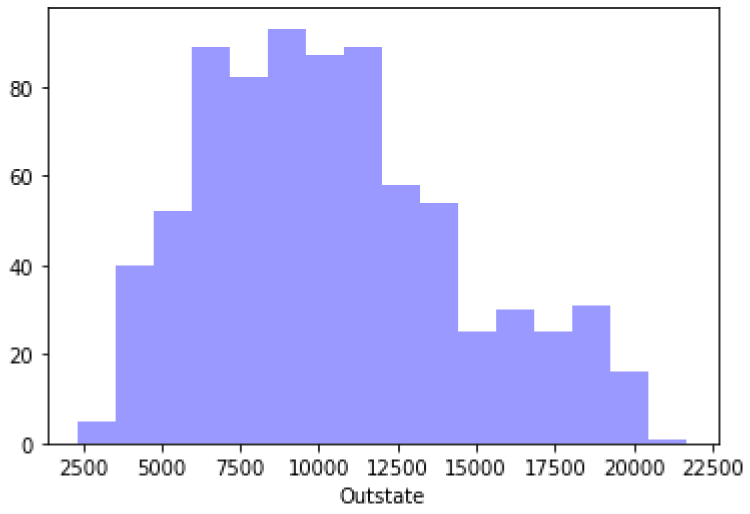
```
Below is the distribution of :P.Undergrad
```



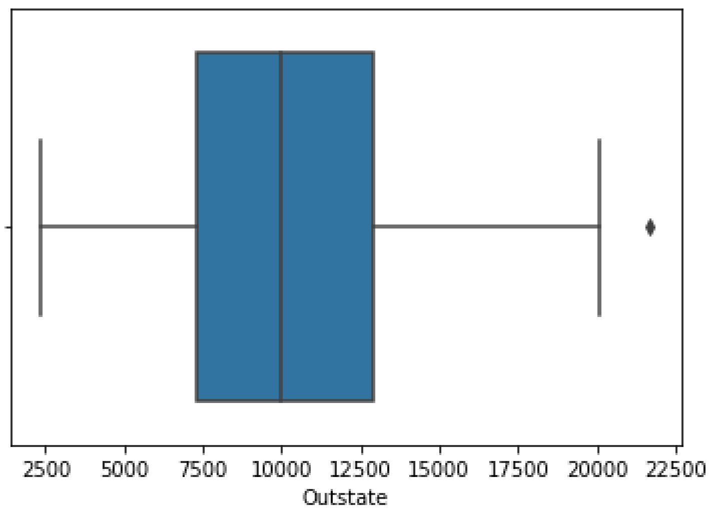
```
*********************************
Below is the box plot of :P.Undergrad
```



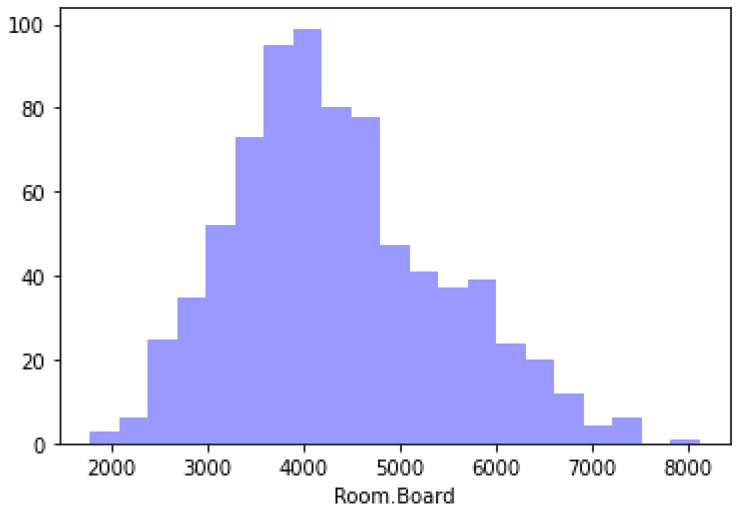
```
Below is the distribution of :Outstate
```



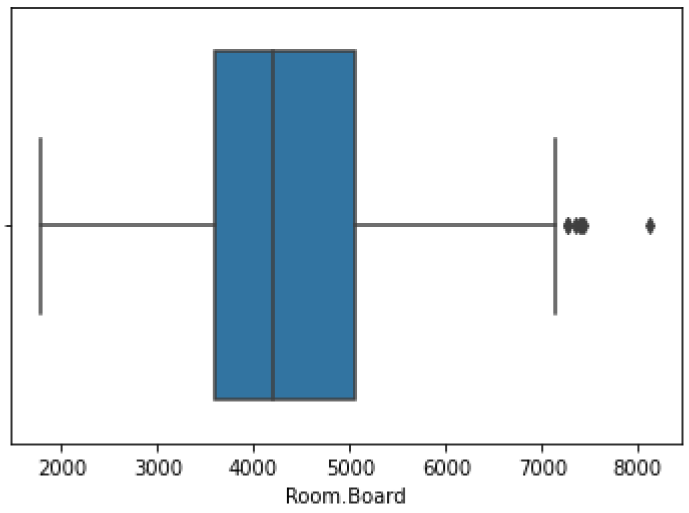
```
*********************************
Below is the box plot of :Outstate
```



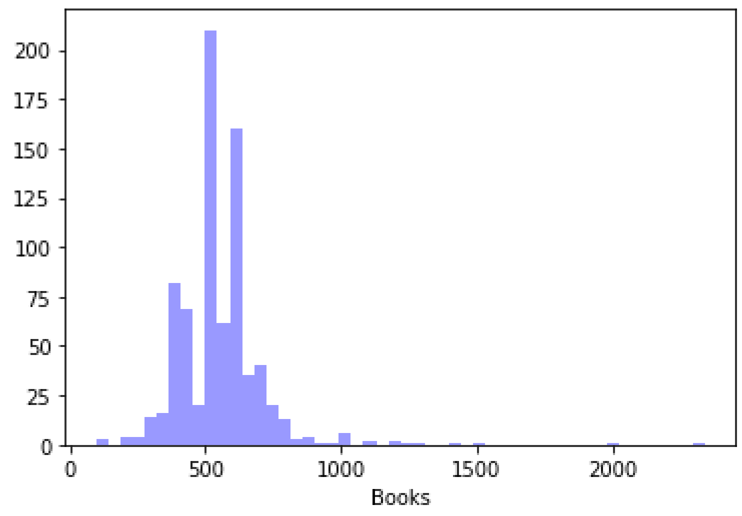
```
Below is the distribution of :Room.Board
```



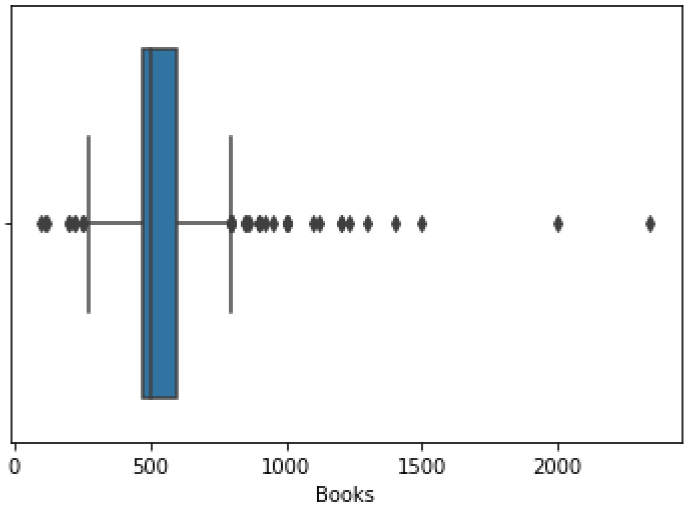
```
**********************************
Below is the box plot of :Room.Board
```



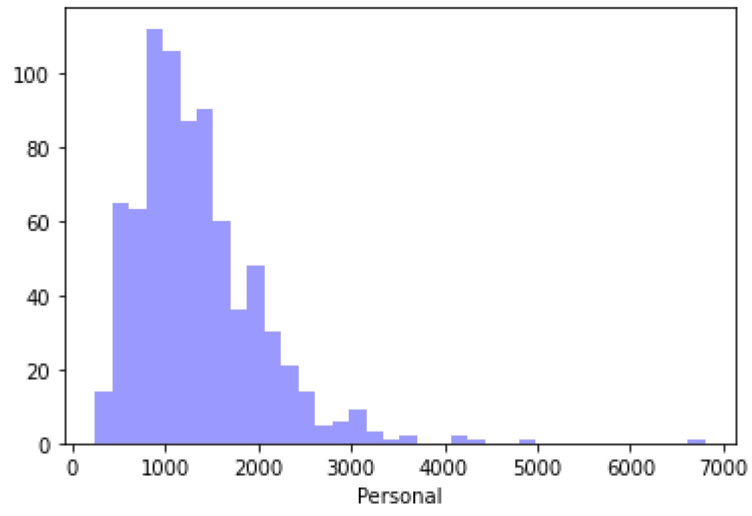
```
Below is the distribution of :Books
```



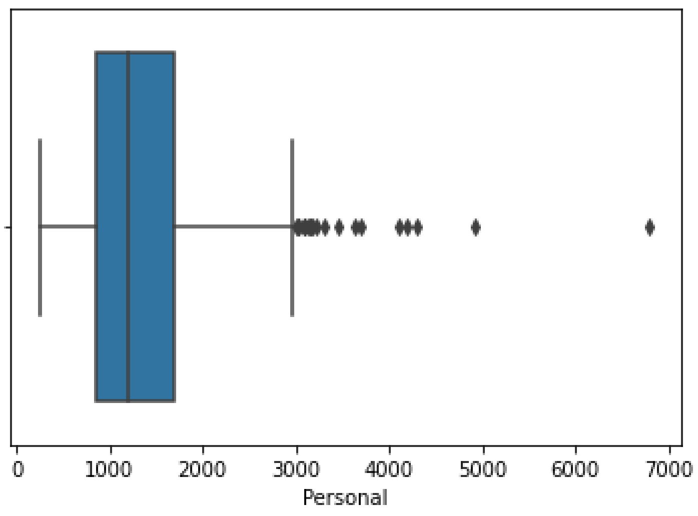
```
*********************************
Below is the box plot of :Books
```



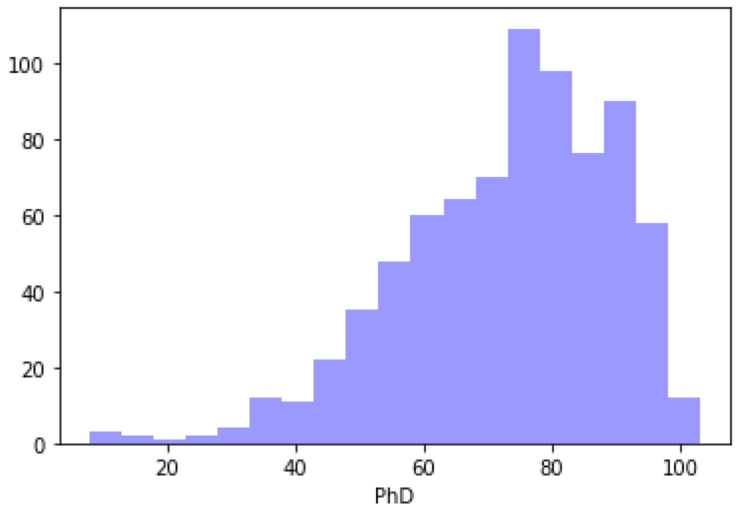
```
Below is the distribution of :Personal
```



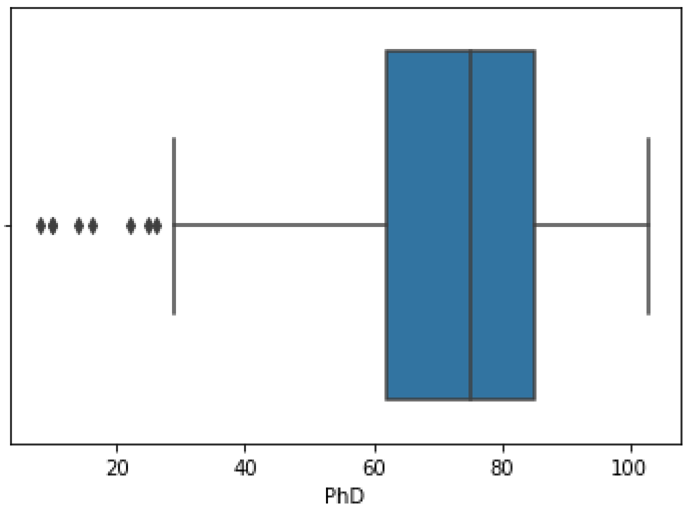
```
*********************************
Below is the box plot of :Personal
```



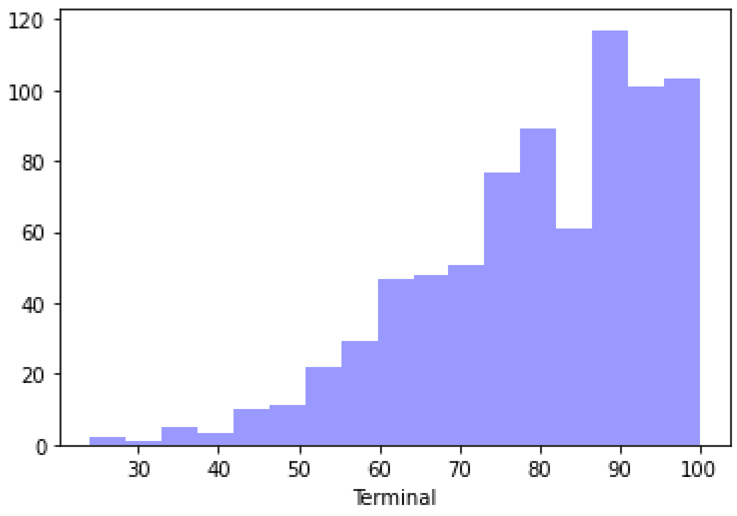
```
Below is the distribution of :PhD
```



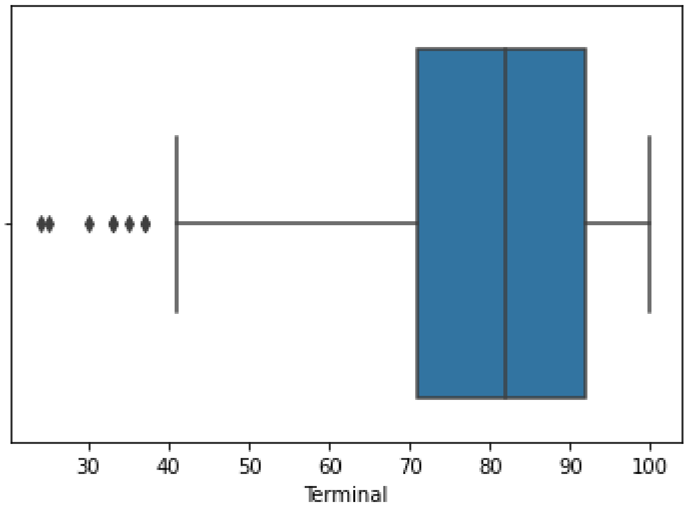
```
**********************************
Below is the box plot of :PhD
```



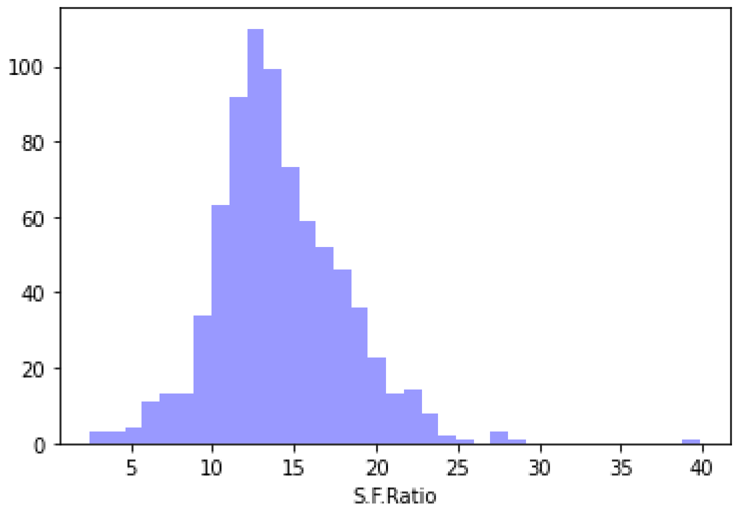
```
Below is the distribution of :Terminal
```



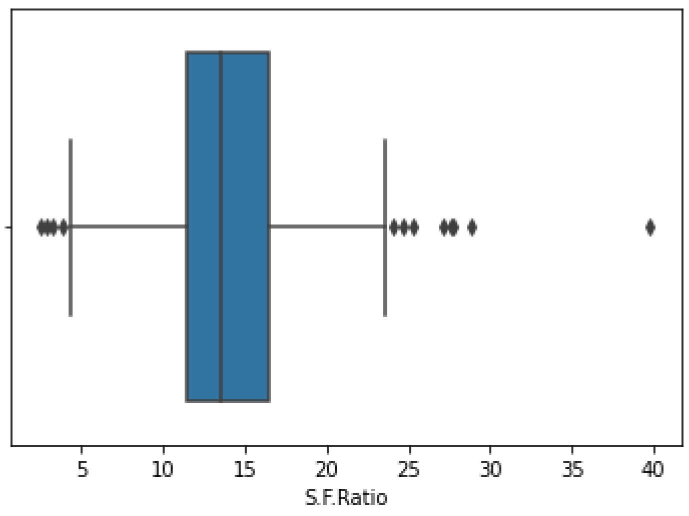
```
*********************************
Below is the box plot of :Terminal
```



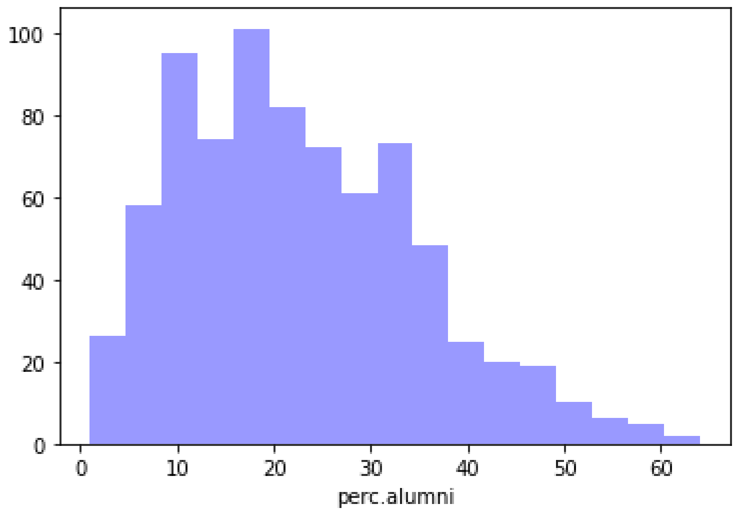
```
Below is the distribution of :S.F.Ratio
```



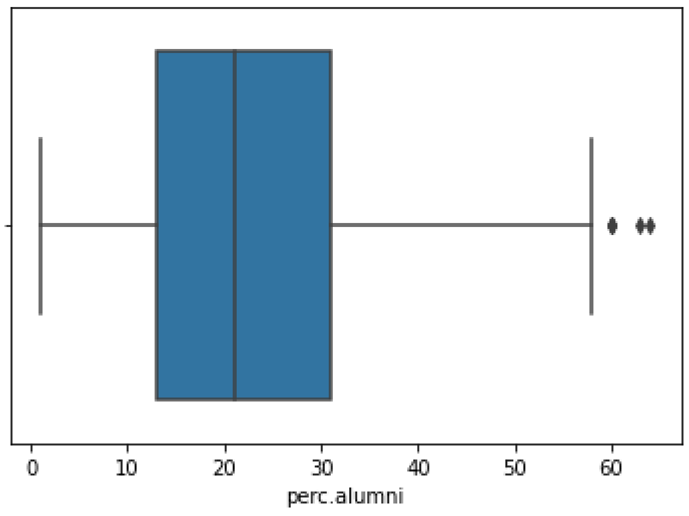
```
*********************************
Below is the box plot of :S.F.Ratio
```



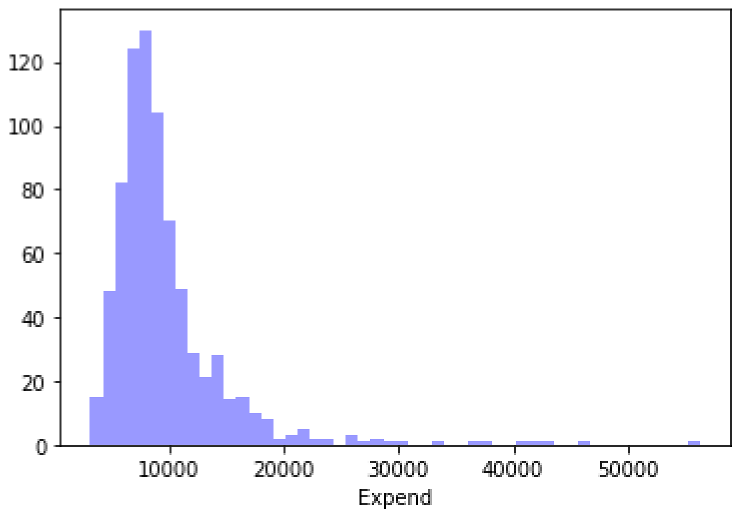
```
Below is the distribution of :perc.alumni
```



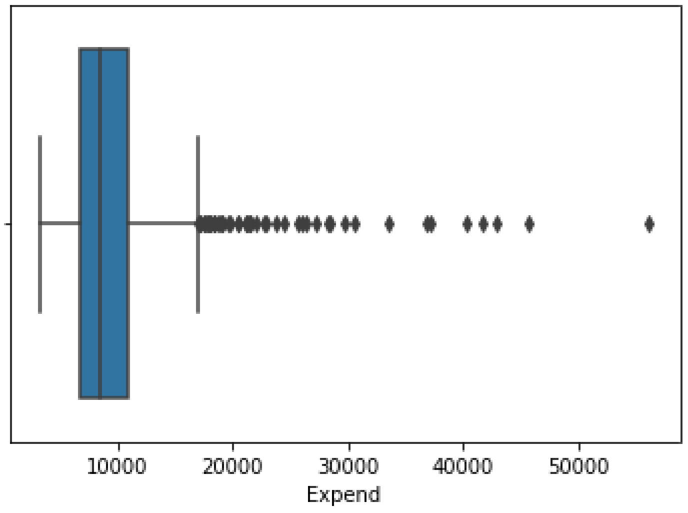
```
**********************************
Below is the box plot of :perc.alumni
```



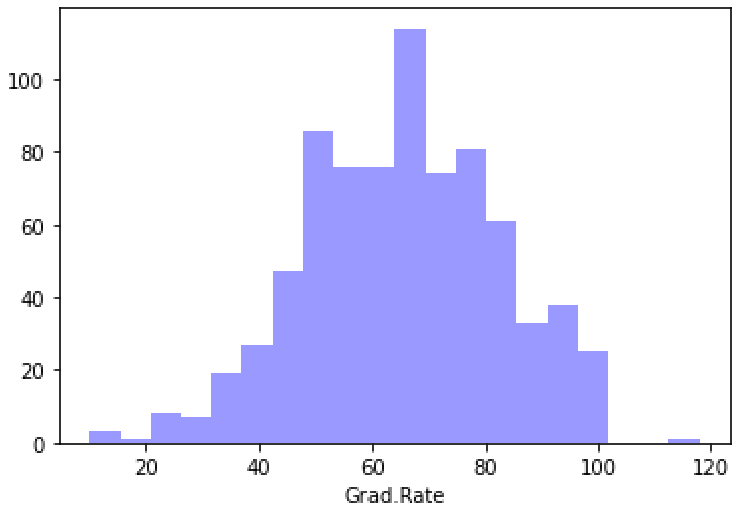
```
Below is the distribution of :Expend
```



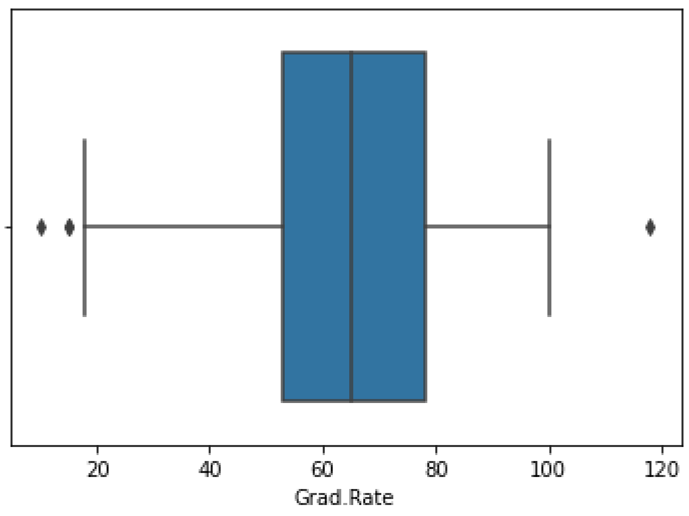
```
*********************************
Below is the box plot of :Expend
```



```
Below is the distribution of :Grad.Rate
```

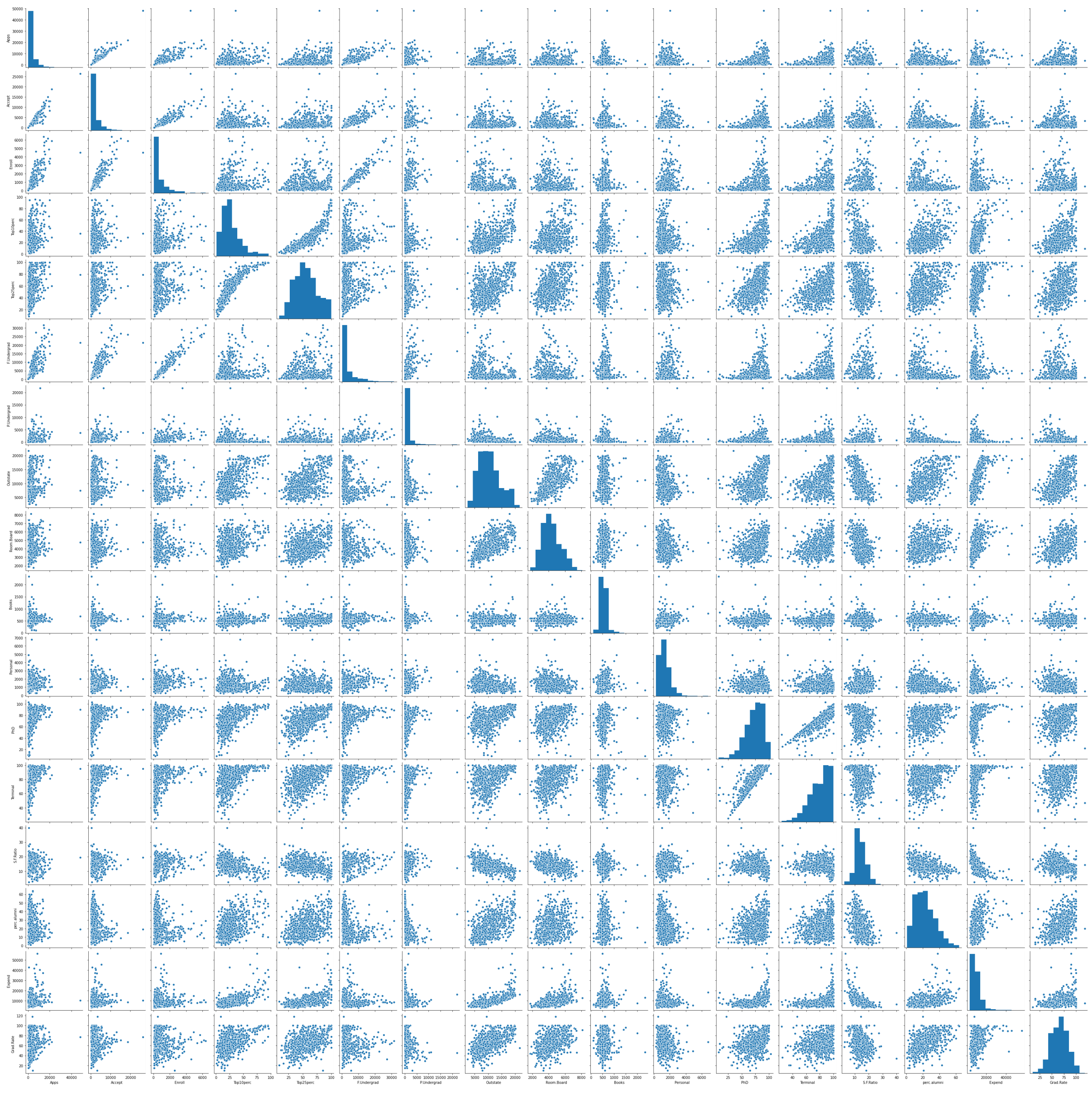


```
*********************************
Below is the box plot of :Grad.Rate
```



Bivariate Ananlysis:

In [88]:
```
sns.pairplot(df1_numerical)
plt.show()
```



In [89]:
```
#correlation matrix

corr_data = df1_numerical.corr()

corr_data
```

Out[89]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	
Apps	1.000000	0.943451	0.846822	0.338834	0.351640	0.814491	0.398264	
Accept	0.943451	1.000000	0.911637	0.192447	0.247476	0.874223	0.441271	-
Enroll	0.846822	0.911637	1.000000	0.181294	0.226745	0.964640	0.513069	-
Top10perc	0.338834	0.192447	0.181294	1.000000	0.891995	0.141289	-0.105356	
Top25perc	0.351640	0.247476	0.226745	0.891995	1.000000	0.199445	-0.053577	
F.Undergrad	0.814491	0.874223	0.964640	0.141289	0.199445	1.000000	0.570512	-
P.Undergrad	0.398264	0.441271	0.513069	-0.105356	-0.053577	0.570512	1.000000	-
Outstate	0.050159	-0.025755	-0.155477	0.562331	0.489394	-0.215742	-0.253512	
Room.Board	0.164939	0.090899	-0.040232	0.371480	0.331490	-0.068890	-0.061326	
Books	0.132559	0.113525	0.112711	0.118858	0.115527	0.115550	0.081200	
Personal	0.178731	0.200989	0.280929	-0.093316	-0.080810	0.317200	0.319882	-
PhD	0.390697	0.355758	0.331469	0.531828	0.545862	0.318337	0.149114	
Terminal	0.369491	0.337583	0.308274	0.491135	0.524749	0.300019	0.141904	
S.F.Ratio	0.095633	0.176229	0.237271	-0.384875	-0.294629	0.279703	0.232531	-
perc.alumni	-0.091649	-0.161391	-0.181458	0.452853	0.418289	-0.229185	-0.282213	
Expend	0.259592	0.124717	0.064169	0.660913	0.527447	0.018652	-0.083568	
Grad.Rate	0.146755	0.067313	-0.022341	0.494989	0.477281	-0.078773	-0.257001	

Keeping the original dataset unchanged before proceeding further with the dataset:

In [31]:
```
# to keep the original dataset unchanged , we will first make a copy of that f
or further use.
# for just in case if we need our original dataset anytime later

df2 = df1.copy()    #copying it to new datframe df2

df2
```

Out[31]:

	Names	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outs
0	Abilene Christian University	1660	1232	721	23	52	2885	537	7
1	Adelphi University	2186	1924	512	16	29	2683	1227	12
2	Adrian College	1428	1097	336	22	50	1036	99	1
3	Agnes Scott College	417	349	137	60	89	510	63	12
4	Alaska Pacific University	193	146	55	16	44	249	869	7
...	...	...	...	...	...	...	...	...	
772	Worcester State College	2197	1515	543	4	26	3089	2029	6
773	Xavier University	1959	1805	695	24	47	2849	1107	1
774	Xavier University of Louisiana	2097	1915	695	34	61	2793	166	6
775	Yale University	10705	2453	1317	95	99	5217	83	19
776	York College of Pennsylvania	2989	1855	691	28	63	2988	1726	4

777 rows × 18 columns

Q. Is scaling necessary for PCA in this case? Give justification and perform scaling.

Justification:

Most of the time, the variables present in the data are of different scales, for example one variable having 4 digits of numeric values and other having single digit. So, it becomes difficult to compare these variables. That's why we use feature scaling to standardize the range of features of data. It is very important step.

In this method, we convert different scales of measurement into single scale.

We will use zscore to normalize the data(only for numerical data).

In [97]:
```
# Scaling of the data.

from scipy.stats import zscore
df_num_scaled=df1_numerical.apply(zscore)
```

In [98]:
```
# we will see that now all the numeric values of the variable has normalized a
nd scaled to one scale.

df_num_scaled.head()
```

Out[98]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate
0	-0.346882	-0.321205	-0.063509	-0.258583	-0.191827	-0.168116	-0.209207	-0.746356
1	-0.210884	-0.038703	-0.288584	-0.655656	-1.353911	-0.209788	0.244307	0.457496
2	-0.406866	-0.376318	-0.478121	-0.315307	-0.292878	-0.549565	-0.497090	0.201305
3	-0.668261	-0.681682	-0.692427	1.840231	1.677612	-0.658079	-0.520752	0.626633
4	-0.726176	-0.764555	-0.780735	-0.655656	-0.596031	-0.711924	0.009005	-0.716508

Q. Comment on the comparison between the covariance and the correlation matrices from this data [on scaled data].

Correlation is a measure used to represent how strongly two random variables are related to each other. It is basically the scaled form of covariance.

Covariance is nothing but a measure of correlation. Covariance indicates the direction of the linear relationship between variables.

In our below correlation matrices we can see the correlation between all the 17 variables in the dataset but in covariance it is linear relationship between variables.

```
In [105]: #Covariance matrices:

          covariance_matrix = np.cov(df_num_scaled.T)
          print('Covariance Matrix \n%s', covariance_matrix )
```

```
Covariance Matrix
%s [[ 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]]
```

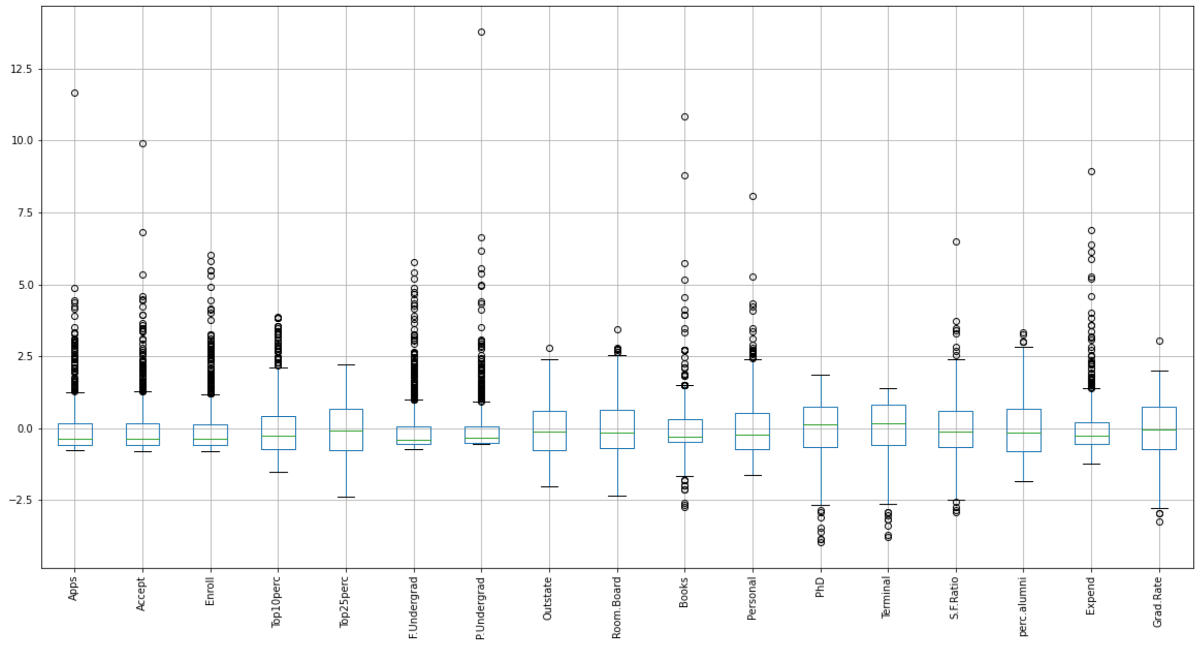
In [106]:
```
#correlation matrices:

corr = df_num_scaled.corr(method='pearson')
corr
```

Out[106]:

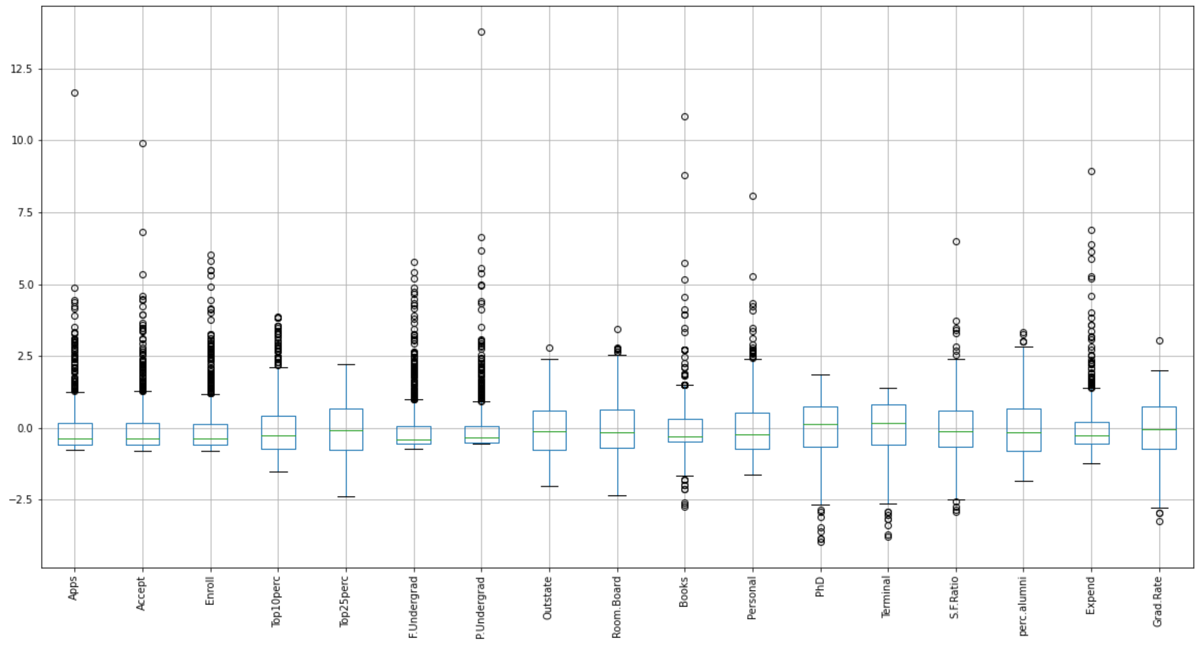
	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	
Apps	1.000000	0.943451	0.846822	0.338834	0.351640	0.814491	0.398264	
Accept	0.943451	1.000000	0.911637	0.192447	0.247476	0.874223	0.441271	-
Enroll	0.846822	0.911637	1.000000	0.181294	0.226745	0.964640	0.513069	-
Top10perc	0.338834	0.192447	0.181294	1.000000	0.891995	0.141289	-0.105356	
Top25perc	0.351640	0.247476	0.226745	0.891995	1.000000	0.199445	-0.053577	
F.Undergrad	0.814491	0.874223	0.964640	0.141289	0.199445	1.000000	0.570512	-
P.Undergrad	0.398264	0.441271	0.513069	-0.105356	-0.053577	0.570512	1.000000	-
Outstate	0.050159	-0.025755	-0.155477	0.562331	0.489394	-0.215742	-0.253512	
Room.Board	0.164939	0.090899	-0.040232	0.371480	0.331490	-0.068890	-0.061326	
Books	0.132559	0.113525	0.112711	0.118858	0.115527	0.115550	0.081200	
Personal	0.178731	0.200989	0.280929	-0.093316	-0.080810	0.317200	0.319882	-
PhD	0.390697	0.355758	0.331469	0.531828	0.545862	0.318337	0.149114	
Terminal	0.369491	0.337583	0.308274	0.491135	0.524749	0.300019	0.141904	
S.F.Ratio	0.095633	0.176229	0.237271	-0.384875	-0.294629	0.279703	0.232531	-
perc.alumni	-0.090226	-0.159990	-0.180794	0.455485	0.417864	-0.229462	-0.280792	
Expend	0.259592	0.124717	0.064169	0.660913	0.527447	0.018652	-0.083568	
Grad.Rate	0.146755	0.067313	-0.022341	0.494989	0.477281	-0.078773	-0.257001	

Q. Check the dataset for outliers before and after scaling. What insight do you derive here? [Please do not treat Outliers unless specifically asked to do so]

In [107]:
```
# box plot to show outliers in the dataset before scaling


df1_numerical.boxplot(figsize=(20,10))
plt.xticks(rotation=90)
plt.show()
```



In [108]:
```
# boxplot to show outliers in the dataset after scaling

df_num_scaled.boxplot(figsize=(20,10))
plt.xticks(rotation=90)
plt.show()
```



Outliers are present in the data. We can see clearly the difference after the scaling of the data. Now every variable is showing on single scale.

Q. Extract the eigenvalues and eigenvectors.[print both]

In [109]:
```
# Getting the eigen values and eigen vector:

eigen_value, eigen_vectors = np.linalg.eig(covariance_matrix)
print('Eigen Values \n %s', eigen_value )

print('--------')

print('Eigen Vectors \n %s', eigen_vectors )
```

```
Eigen Values
 %s [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
 %s [[-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]]
```

Q. Perform PCA and export the data of the Principal Component (eigenvectors) into a data frame with the original features.

We will perform PCA, but before that we will do few statistical tests to see whether we should consider performing PCA or not.

Bartlett's Test of Sphericity:

Bartlett's test of sphericity tests the hypothesis that the variables are uncorrelated in the population.

H0: All variables in the data are uncorrelated.

Ha: At least one pair of variables in the data are correlated.

If the null hypothesis cannot be rejected, then PCA is not advisable.

If the p-value is small, then we can reject the null hypothesis and agree that there is atleast one pair of variables in the data which are correlated hence PCA is recommended.

In [110]:
```
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity

chi_square_value,p_value = calculate_bartlett_sphericity(df_num_scaled)

p_value
```

Out[110]:
```
0.0
```

Here, we can see that the p-value is small(0.0), so now we can reject the null hypothesis and agree that there is atleast one pair of variables in the data which are correlated hence PCA is recommended.

Performing KMO Test :

Measure of sampling adequacy (MSA) is an index used to examine how appropriate PCA is.

If MSA is less than 0.5, PCA is not recommended, since no reduction is expected.

But if the MSA > 0.7 then it is expected to provide a considerable reduction in the dimension and extraction of meaningful components.

In [111]:
```
from factor_analyzer.factor_analyzer import calculate_kmo

kmo_all,kmo_model=calculate_kmo(df_num_scaled)

kmo_model
```

Out[111]:
```
0.8131251200373524
```

Here we can see that, MSA is 0.81 and that is greater than 0.7, so it is expected to provide a considerable reduction in the dimension and extraction of meaningful components.

Performing PCA:

In [112]:
```
#If we want to see how much variance is being explained by our all principal c
omponenets and also want to see
#the cumulative variance explained then first we have to do the sum of all eig
en values.

total = sum(eigen_value)

total
```

Out[112]:
```
17.02190721649484
```

In [113]:
```
# we can also see how much percentage variance being explained by particular e
igen values by below calculation.
# we have to divide eigen value of that particular principal componenet by sum
of all the total eigen values.

perc_var_exp = [(i/total)*100 for i in sorted(eigen_value, reverse=True)]

print('Percentage variance being explained by particular eigen values :\n\n',p
erc_var_exp)
```

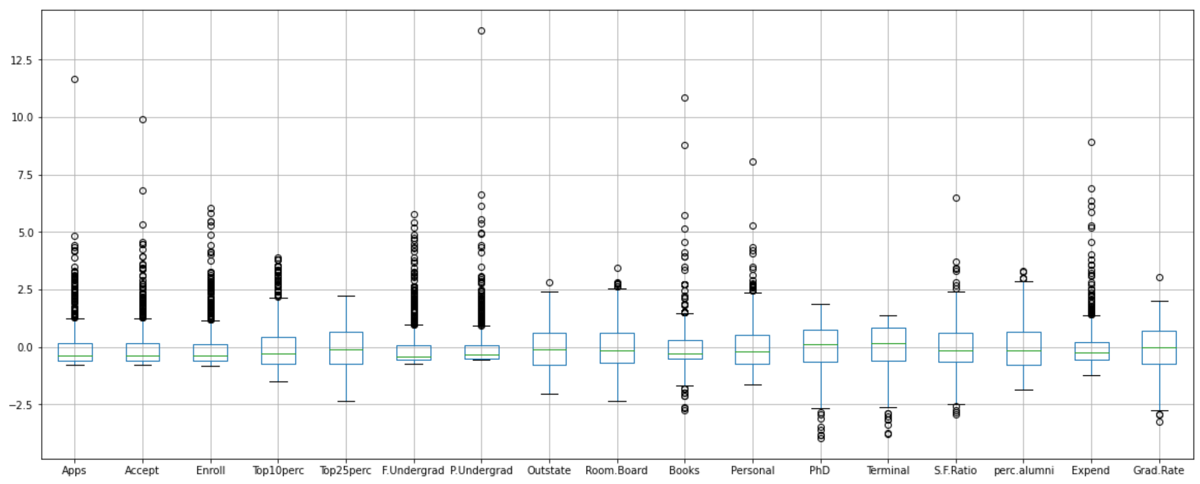
```
Percentage variance being explained by particular eigen values :

 [32.02062819886913, 26.340214436112465, 6.900916554222499, 5.92298922292629
3, 5.488405110358485, 4.984700954557448, 3.558871491746655, 3.453621336999266
3, 3.1172336798217164, 2.3751915258938, 1.8414263209386879, 1.296041400123535
9, 0.9857541228001174, 0.8458423350830034, 0.5171255833731926, 0.215754010072
75546, 0.13528371610095133]
```

In [114]:
```
# and cumulative variance explained by eigen values is calculated as :

cumulative_values_of_eigen_values = np.cumsum(perc_var_exp)

print('Cumulative values explained by eigen values :\n\n',cumulative_values_of
_eigen_values)
```

```
Cumulative values explained by eigen values :

 [ 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.        ]
```

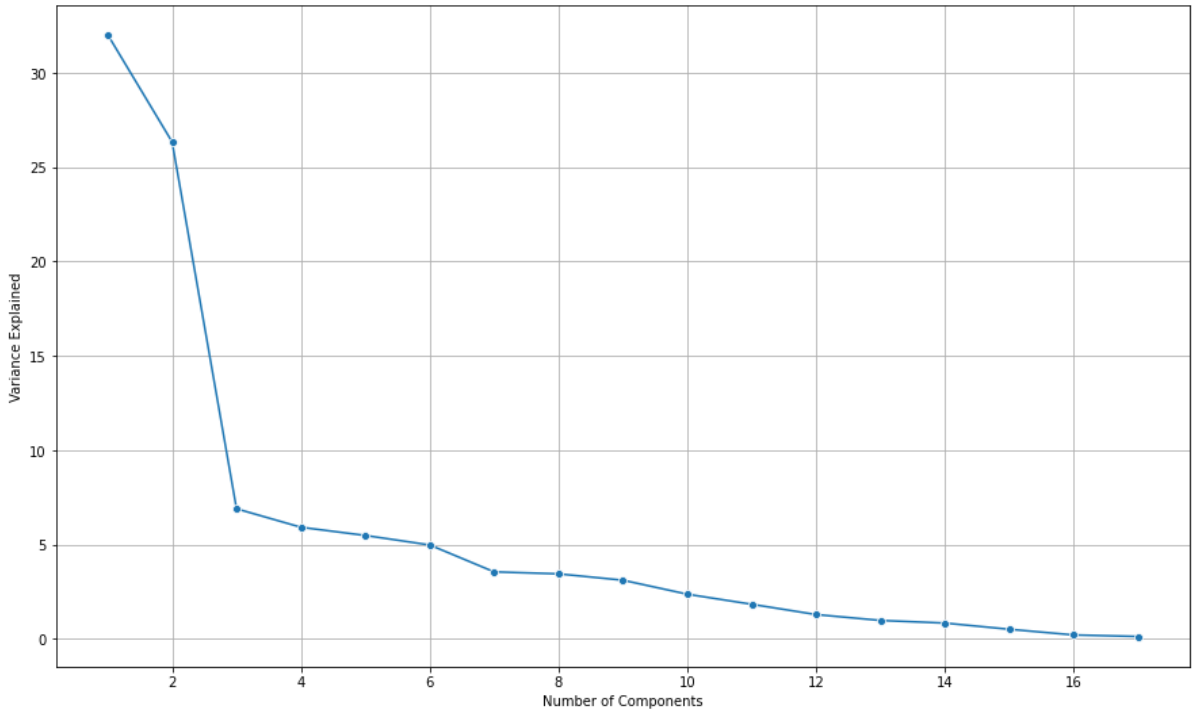
```
In [116]: df_num_scaled.boxplot(figsize=(20,8))
```

```
Out[116]: <AxesSubplot:>
```



Below is the Scree Plot to get the number of components to be built:

In [117]:
```
# Scree Plot:

plt.figure(figsize=(15,9))

# we will make the lineplot:

sns.lineplot(y=perc_var_exp, x=range(1,len(perc_var_exp)+1),marker='o')

plt.xlabel('Number of Components',fontsize=10)

plt.ylabel('Variance Explained',fontsize=10)

plt.grid()

plt.show()
```



We will take 6 PCA dimensions out of 17, because after that point, there is a continuous decline in the variance as displaying in the above scree plot.

In [121]:
```
# Now we will apply PCA for the 6 :
# Out of 17 columns, we have decided for only 6 PCA dimensions, so the dimensi
onality reduced from 17 to 6.
# importing PCA from sklearn:

from sklearn.decomposition import PCA

pca = PCA(n_components=6)

df_pca = pca.fit_transform(df_num_scaled)
```

Out[121]:
```
array([[-1.59285540e+00,  7.67333510e-01, -1.01073452e-01,
        -9.21749413e-01, -7.43975433e-01, -2.98306010e-01],
       [-2.19240180e+00, -5.78829984e-01,  2.27879802e+00,
         3.58891825e+00,  1.05999665e+00, -1.77137392e-01],
       [-1.43096371e+00, -1.09281889e+00, -4.38092808e-01,
         6.77240527e-01, -3.69613276e-01, -9.60591686e-01],
       ...,
       [-7.32560596e-01, -7.72352397e-02, -4.05644759e-04,
         5.43162812e-02, -5.16021117e-01,  4.68014245e-01],
       [ 7.91932735e+00, -2.06832886e+00,  2.07356382e+00,
         8.52053973e-01, -9.47754802e-01, -2.06993727e+00],
       [-4.69508066e-01,  3.66660943e-01, -1.32891512e+00,
        -1.08022562e-01, -1.13217595e+00,  8.39893111e-01]])
```

In [122]:
```
# transpose of the component

df_pca.transpose()
```

Out[122]:
```
array([[-1.59285540e+00, -2.19240180e+00, -1.43096371e+00, ...,
        -7.32560596e-01,  7.91932735e+00, -4.69508066e-01],
       [ 7.67333510e-01, -5.78829984e-01, -1.09281889e+00, ...,
        -7.72352397e-02, -2.06832886e+00,  3.66660943e-01],
       [-1.01073452e-01,  2.27879802e+00, -4.38092808e-01, ...,
        -4.05644759e-04,  2.07356382e+00, -1.32891512e+00],
       [-9.21749413e-01,  3.58891825e+00,  6.77240527e-01, ...,
         5.43162812e-02,  8.52053973e-01, -1.08022562e-01],
       [-7.43975433e-01,  1.05999665e+00, -3.69613276e-01, ...,
        -5.16021117e-01, -9.47754802e-01, -1.13217595e+00],
       [-2.98306010e-01, -1.77137392e-01, -9.60591686e-01, ...,
         4.68014245e-01, -2.06993727e+00,  8.39893111e-01]])
```

In [123]:
```
# Loading of each feature on the components

pca.components_
```

Out[123]:
```
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],
       [-0.0630921 , -0.10124906, -0.08298556,  0.03505553, -0.02414794,
        -0.06139299,  0.13968172,  0.04659887,  0.14896739,  0.67741165,
         0.49972112, -0.12702837, -0.06603755, -0.2898484 , -0.14698927,
         0.22674398, -0.20806465],
       [ 0.28131053,  0.26781735,  0.16182677, -0.05154725, -0.10976654,
         0.10041234, -0.15855849,  0.13129136,  0.18499599,  0.08708922,
        -0.23071057, -0.53472483, -0.51944302, -0.16118949,  0.01731422,
         0.07927349,  0.26912907],
       [ 0.00574141,  0.05578609, -0.05569364, -0.39543434, -0.42653359,
        -0.04345436,  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,
        -0.04345425, -0.19119858, -0.03000039,  0.16275545,  0.64105495,
        -0.331398  ,  0.09125552,  0.15492765,  0.48704588, -0.04734001,
        -0.29811862,  0.21616331]])
```

In [124]:
```
# Quantom of variance explained:

pca.explained_variance_ratio_
```

Out[124]:
```
array([0.32020628, 0.26340214, 0.06900917, 0.05922989, 0.05488405,
       0.04984701])
```

In [125]:
```
# Now we will create the dataframe of loading againts each field:

new_df_loading_pca = pd.DataFrame(pca.components_,columns=list(df_num_scaled))

new_df_loading_pca
```

Out[125]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate
0	0.248766	0.207602	0.176304	0.354274	0.344001	0.154641	0.026443	0.294736
1	0.331598	0.372117	0.403724	-0.082412	-0.044779	0.417674	0.315088	-0.249644
2	-0.063092	-0.101249	-0.082986	0.035056	-0.024148	-0.061393	0.139682	0.046599
3	0.281311	0.267817	0.161827	-0.051547	-0.109767	0.100412	-0.158558	0.131291
4	0.005741	0.055786	-0.055694	-0.395434	-0.426534	-0.043454	0.302385	0.222532
5	-0.016237	0.007535	-0.042558	-0.052693	0.033092	-0.043454	-0.191199	-0.030000

In [126]:
```
#getting the shape and identifying the pattern:

new_df_loading_pca.shape
```

Out[126]:
```
(6, 17)
```

In [127]:
```
#Checking the head of this dataframe:

new_df_loading_pca.head(6)
```

Out[127]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate
0	0.248766	0.207602	0.176304	0.354274	0.344001	0.154641	0.026443	0.294736
1	0.331598	0.372117	0.403724	-0.082412	-0.044779	0.417674	0.315088	-0.249644
2	-0.063092	-0.101249	-0.082986	0.035056	-0.024148	-0.061393	0.139682	0.046599
3	0.281311	0.267817	0.161827	-0.051547	-0.109767	0.100412	-0.158558	0.131291
4	0.005741	0.055786	-0.055694	-0.395434	-0.426534	-0.043454	0.302385	0.222532
5	-0.016237	0.007535	-0.042558	-0.052693	0.033092	-0.043454	-0.191199	-0.030000

Q. Write down the explicit form of the first PC (in terms of the eigenvectors. Use values with two places of decimals only).

In [140]:
```
np.round(eigen_vectors[0],2)
```

Out[140]:
```
array([-0.25,  0.33,  0.06, -0.28,  0.01,  0.02,  0.04,  0.1 ,  0.09,
       -0.05,  0.36, -0.46,  0.04, -0.13,  0.08, -0.6 ,  0.02])
```

Q. Consider 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?

In [132]:
```
cumulative_values_of_eigen_values = np.cumsum(perc_var_exp)

print('Cumulative values explained by eigen values :',cumulative_values_of_eig
en_values)
```

```
Cumulative values explained by eigen values : [ 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.        ]
```

Eigen vectors are our principal components. Eigen values helps us to understand the quantom of variance which is being explained by our principal components.

```
In [ ]:
```