

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.xlsx**.

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

*Step 1 - import all the necessary libraries that we think we will requiring to perform the EDA.**

Importing Libraries

```
In [18]: import pandas as pd  
import seaborn as sns  
import numpy as np  
from sklearn import datasets  
import matplotlib.pyplot as plt  
import seaborn as sns  
%matplotlib inline
```

Loading the data set

Excel file using pandas and `read_excel` file.

```
In [336]: df=pd.read_excel('Education - Post 12th Standard-1.xlsx')
```

Basic Data Exploration

In this step, we will perform the below operations to check what the data set comprises of. We will check the below things:

- head of the dataset
- shape of the dataset
- info of the dataset
- summary of the dataset

In [337]: df.head()

Out[337]:

	Names	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstat
0	Abilene Christian University	1660	1232	721	23	52	2885	537	7441
1	Adelphi University	2186	1924	512	16	29	2683	1227	12281
2	Adrian College	1428	1097	336	22	50	1036	99	11251
3	Agnes Scott College	417	349	137	60	89	510	63	12961
4	Alaska Pacific University	193	146	55	16	44	249	869	7561

head function will tell you the top records in the data set. By default python shows you only top 5 records.

In [338]: df.shape

Out[338]: (777, 18)

Shape attribute tells us number of observations and variables we have in the data set. It is used to check the dimension of data. The cars data set has 777 observations and 18 variables in the data set.

In [339]: df.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
```

info() is used to check the Information about the data and the datatypes of each respective attributes.

In [340]: df.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
```

The describe method will help to see how data has been spread for the numerical values. We can clearly see the minimum value, mean values, different percentile values and maximum values.

In [25]: df.describe()

Out[25]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Un
count	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	777
mean	3001.638353	2018.804376	779.972973	27.558559	55.796654	3699.907336	855
std	3870.201484	2451.113971	929.176190	17.640364	19.804778	4850.420531	1522
min	81.000000	72.000000	35.000000	1.000000	9.000000	139.000000	1
25%	776.000000	604.000000	242.000000	15.000000	41.000000	992.000000	95
50%	1558.000000	1110.000000	434.000000	23.000000	54.000000	1707.000000	353
75%	3624.000000	2424.000000	902.000000	35.000000	69.000000	4005.000000	967
max	48094.000000	26330.000000	6392.000000	96.000000	100.000000	31643.000000	21836

Check for Duplicate records

In [26]: # Check for duplicate data
dups = df.duplicated()
dups.sum()

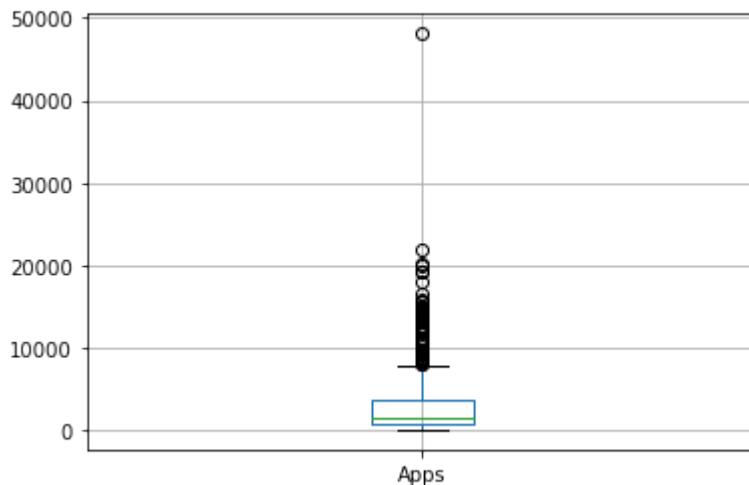
Out[26]: 0

Since we have 0 duplicate records in the data, no need to remove the duplicates.

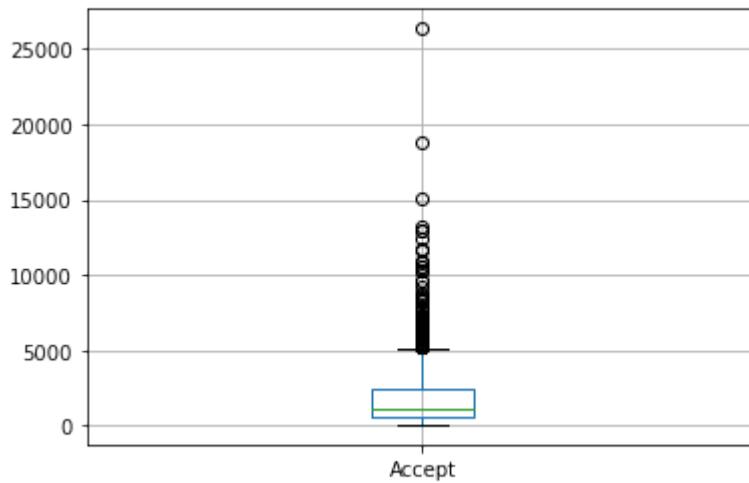
Outlier Treatment

To check for outliers, we will be plotting the box plots.

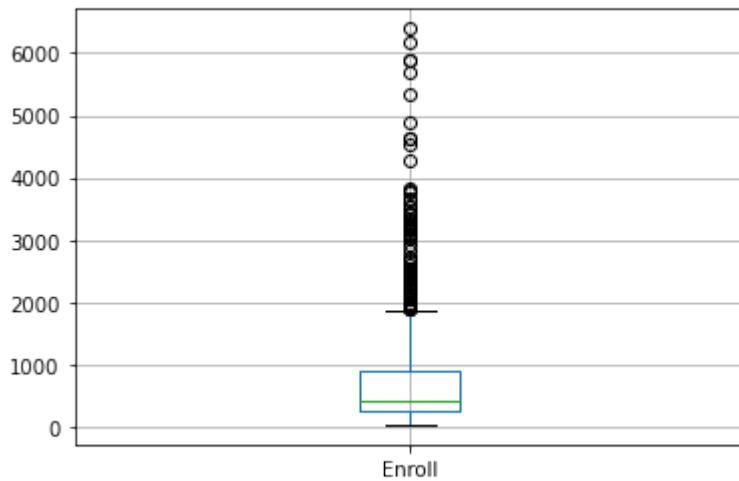
```
In [27]: df.boxplot(column=['Apps'])
plt.show()
```



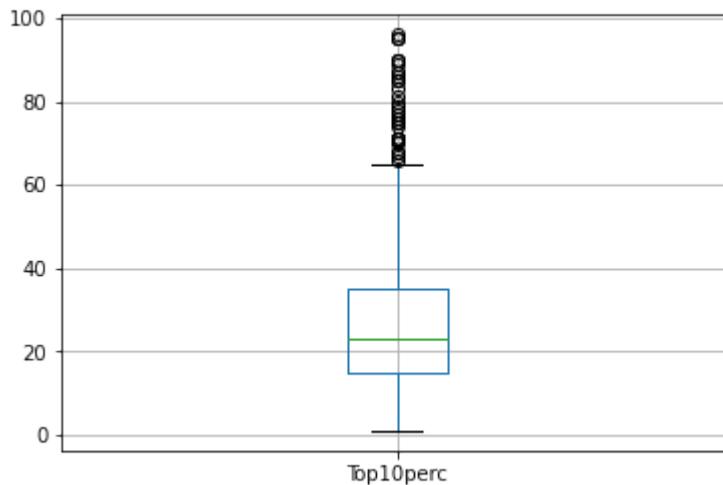
```
In [28]: df.boxplot(column=['Accept'])
plt.show()
```



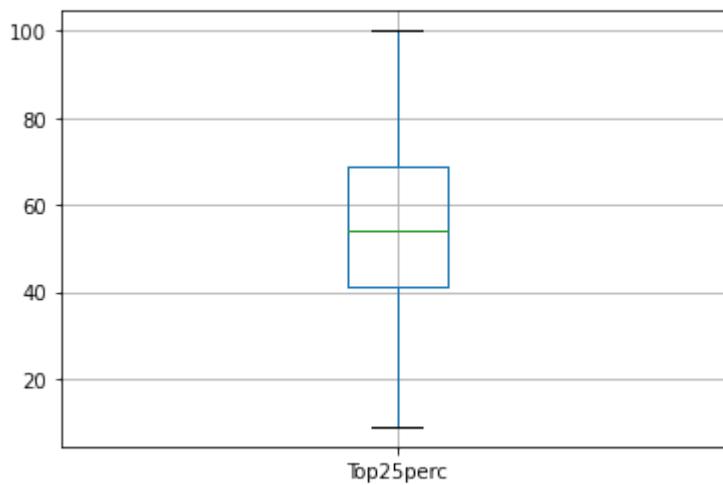
```
In [29]: df.boxplot(column=['Enroll'])
plt.show()
```



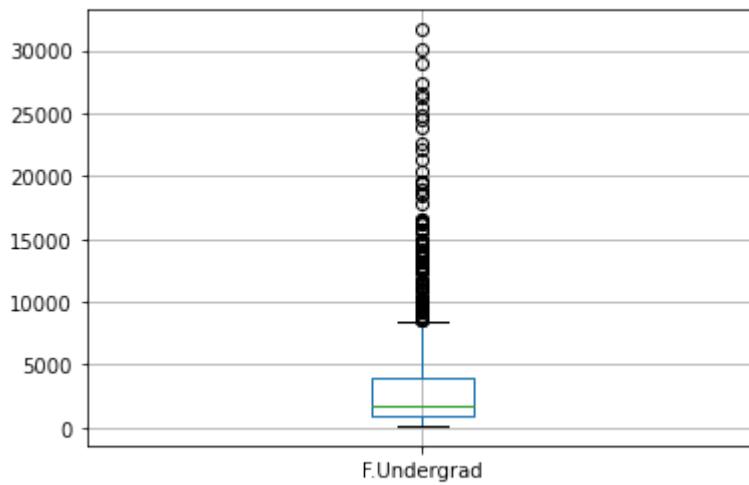
```
In [30]: df.boxplot(column=['Top10perc'])
plt.show()
```



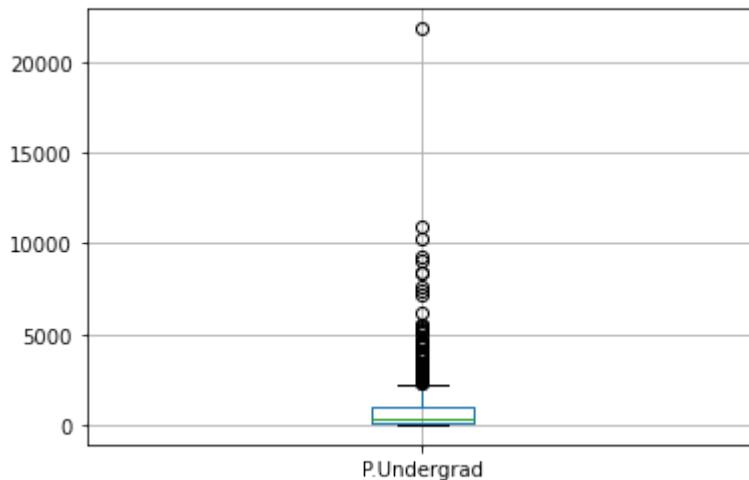
```
In [31]: df.boxplot(column=['Top25perc'])
plt.show()
```



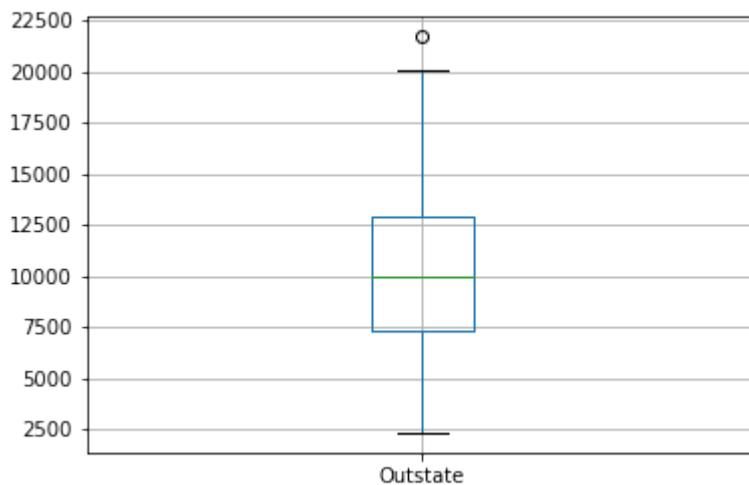
```
In [32]: df.boxplot(column=['F.Undergrad'])
plt.show()
```



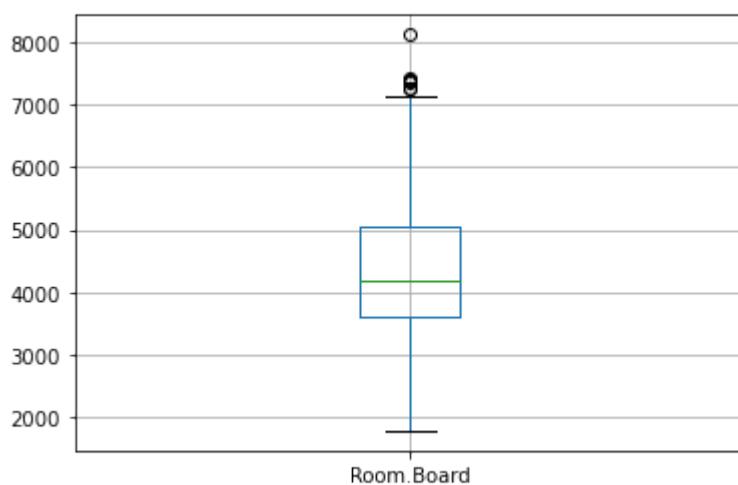
```
In [33]: df.boxplot(column=['P.Undergrad'])
plt.show()
```



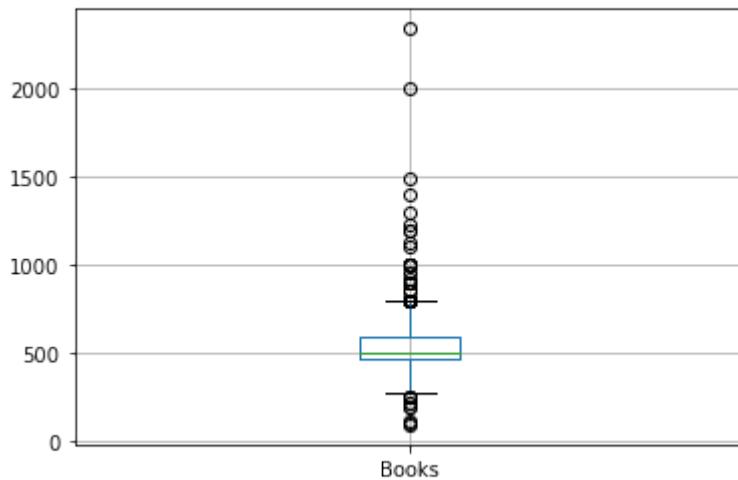
```
In [34]: df.boxplot(column=['Outstate'])
plt.show()
```



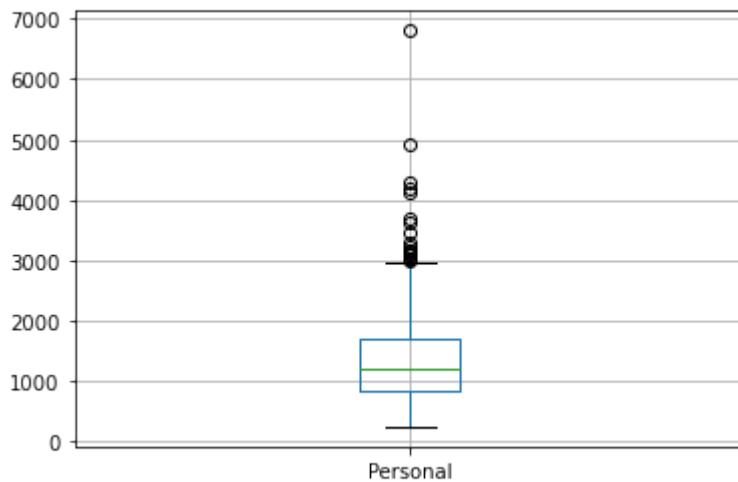
```
In [35]: df.boxplot(column=['Room.Board'])
plt.show()
```



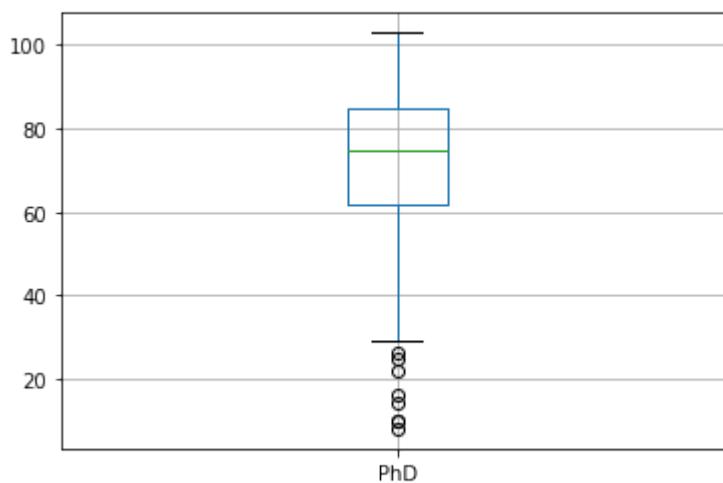
```
In [36]: df.boxplot(column=['Books'])
plt.show()
```



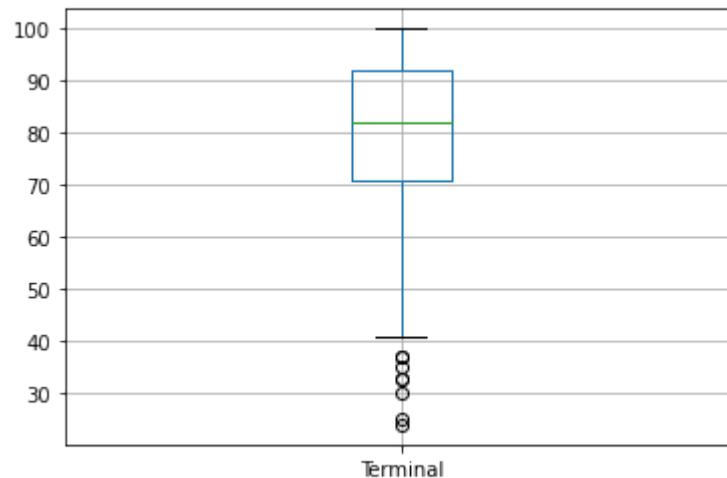
```
In [37]: df.boxplot(column=['Personal'])
plt.show()
```



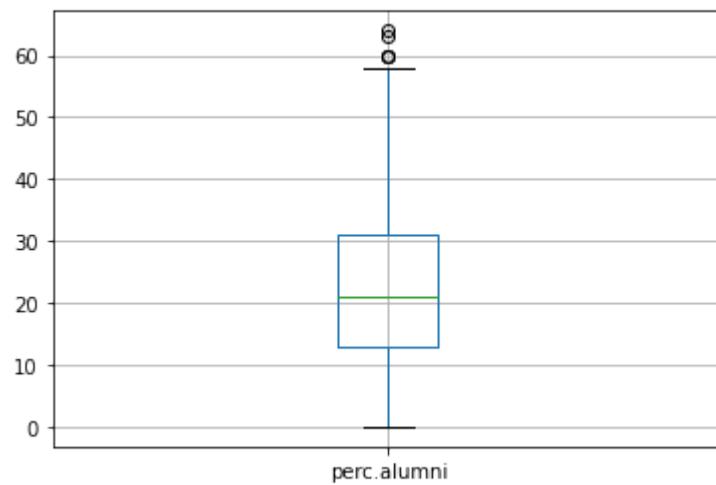
```
In [38]: df.boxplot(column=['PhD'])
plt.show()
```



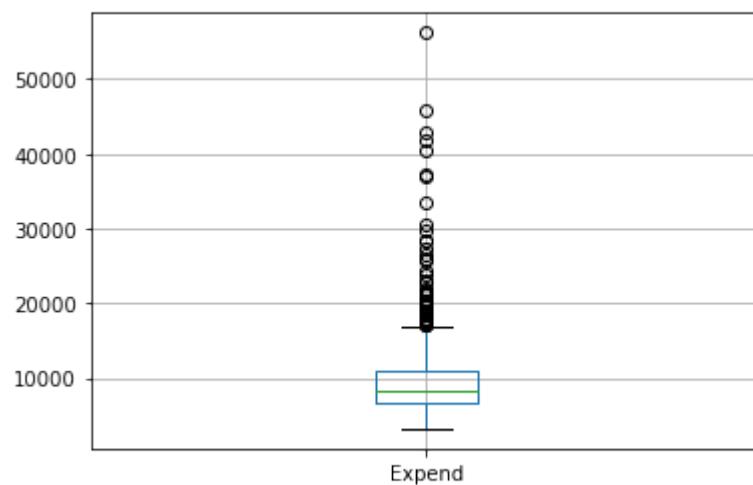
```
In [39]: df.boxplot(column=['Terminal'])
plt.show()
```



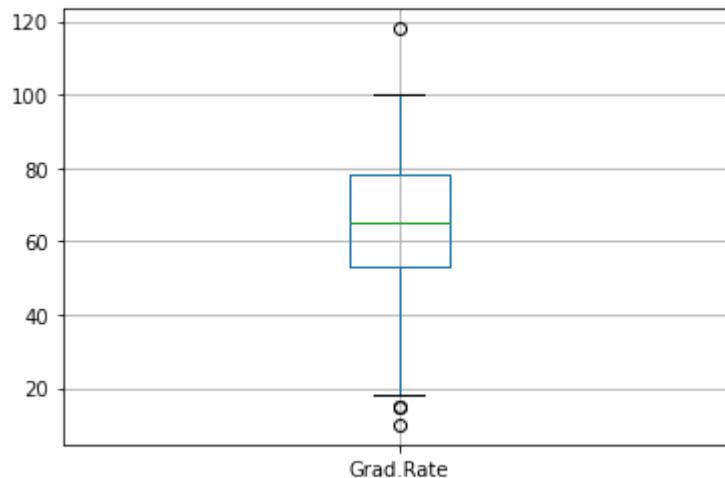
```
In [40]: df.boxplot(column=['perc.alumni'])
plt.show()
```



```
In [41]: df.boxplot(column=['Expend'])
plt.show()
```



```
In [42]: df.boxplot(column=['Grad.Rate'])
plt.show()
```



Looking at the box plot, it seems that all the variables are having outliers except for Top25perc and Outstate.

These outliers value needs to be treated and there are several ways of treating them:

- Drop the outlier value
- Replace the outlier value using the IQR

Created a user defined function for finding the lower and upper range for a variable so that outlier can be treated.

```
In [ ]: def remove_outliers(col):
    sorted(col)
    Q1, Q3 = col.quantile([0.25, 0.75])
    IQR = Q3 - Q1
    lower_range = Q1 - (1.5 * IQR)
    upper_range = Q3 + (1.5 * IQR)
    return lower_range, upper_range
```

```
In [218]: lrapps, urapps = remove_outliers(df['Apps'])
lrapps
```

Out[218]: -0.5059719646553912

```
In [219]: lrapps, urapps = remove_outliers(df['Apps'])
urapps
```

Out[219]: -0.22743959988606752


```
In [356]: lrapps,urapps = remove_outliers(df['Apps'])
df['Apps'] = np.where(df['Apps'] > urapps, urapps, df['Apps'])

lraccept,uraccept = remove_outliers(df['Accept'])
df['Accept'] = np.where(df['Accept'] > uraccept, uraccept, df['Accept'])

lrenroll,urenroll = remove_outliers(df['Enroll'])
df['Enroll'] = np.where(df['Enroll'] > urenroll, urenroll, df['Enroll'])

lrTop10perc,urTop10perc = remove_outliers(df['Top10perc'])
df['Top10perc'] = np.where(df['Top10perc'] > urTop10perc, urTop10perc, df['Top10perc'])

lrTop25perc,urTop25perc = remove_outliers(df['Top25perc'])
df['Top25perc'] = np.where(df['Top25perc'] > urTop25perc, urTop25perc, df['Top25perc'])

lrFUndergrad,urFUndergrad = remove_outliers(df['F.Undergrad'])
df['F.Undergrad'] = np.where(df['F.Undergrad'] > urFUndergrad, urFUndergrad, df['F.Undergrad'])

lrPUndergrad,urPUndergrad = remove_outliers(df['P.Undergrad'])
df['P.Undergrad'] = np.where(df['P.Undergrad'] > urPUndergrad, urPUndergrad, df['P.Undergrad'])

lrOutstate,urOutstate = remove_outliers(df['Outstate'])
df['Outstate'] = np.where(df['Outstate'] > urOutstate, urOutstate, df['Outstate'])

lrRoomBoard,urRoomBoard = remove_outliers(df['Room.Board'])
df['Room.Board'] = np.where(df['Room.Board'] > urRoomBoard, urRoomBoard, df['Room.Board'])

lrBooks,urBooks = remove_outliers(df['Books'])
df['Books'] = np.where(df['Books'] > urBooks, urBooks, df['Books'])
df['Books'] = np.where(df['Books'] < lrBooks, lrBooks, df['Books'])

lrPersonal,urPersonal = remove_outliers(df['Personal'])
df['Personal'] = np.where(df['Personal'] > urPersonal, urPersonal, df['Personal'])

lrPhD,urPhD = remove_outliers(df['PhD'])
df['PhD'] = np.where(df['PhD'] > urPhD, urPhD, df['PhD'])
df['PhD'] = np.where(df['PhD'] < lrPhD, lrPhD, df['PhD'])

lrTerminal,urTerminal = remove_outliers(df['Terminal'])
df['Terminal'] = np.where(df['Terminal'] > urTerminal, urTerminal, df['Terminal'])
df['Terminal'] = np.where(df['Terminal'] < lrTerminal, lrTerminal, df['Terminal'])

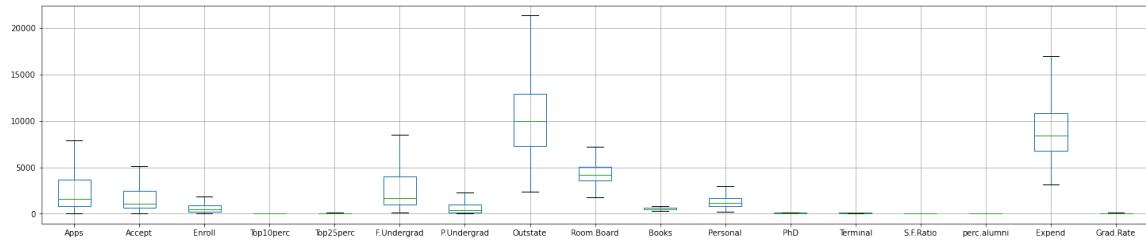
lrSFRatio,urSFRatio = remove_outliers(df['S.F.Ratio'])
df['S.F.Ratio'] = np.where(df['S.F.Ratio'] > urSFRatio, urSFRatio, df['S.F.Ratio'])
df['S.F.Ratio'] = np.where(df['S.F.Ratio'] < lrSFRatio, lrSFRatio, df['S.F.Ratio'])

lrpercalumni,urpercalumni = remove_outliers(df['perc.alumni'])
df['perc.alumni'] = np.where(df['perc.alumni'] > urpercalumni, urpercalumni, df['perc.alumni'])

lrExpend,urExpend = remove_outliers(df['Expend'])
df['Expend'] = np.where(df['Expend'] > urExpend, urExpend, df['Expend'])

lrGradRate,urGradRate = remove_outliers(df['Grad.Rate'])
df['Grad.Rate'] = np.where(df['Grad.Rate'] < lrGradRate, lrGradRate, df['Grad.Rate'])
df['Grad.Rate'] = np.where(df['Grad.Rate'] > urGradRate, urGradRate, df['Grad.Rate'])
```

```
In [357]: plt.figure(figsize=(25,5))
df.boxplot()
plt.show()
```



box plots above,post treating the outlier there are no outliers in all these columns.

Check for missing value

```
In [268]: # Check for missing value in any column
df.isnull().sum()
```

```
Out[268]: 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
```

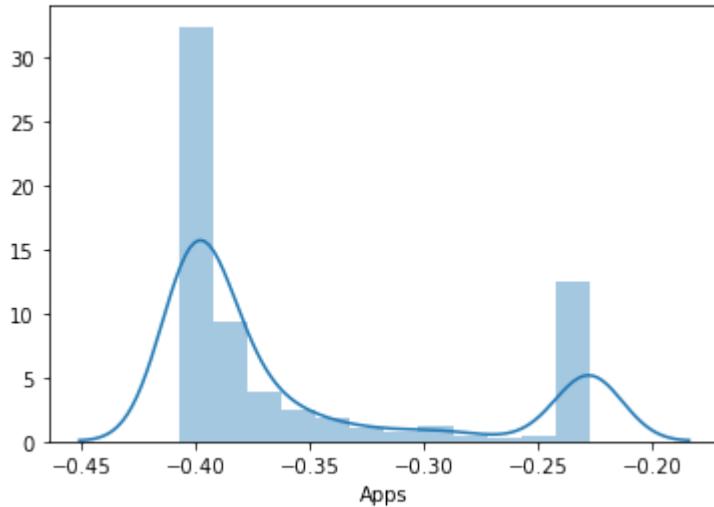
Univariate Analysis

Univariate Analysis is done on various variables

Number of application received

```
In [270]: sns.distplot(df["Apps"])
```

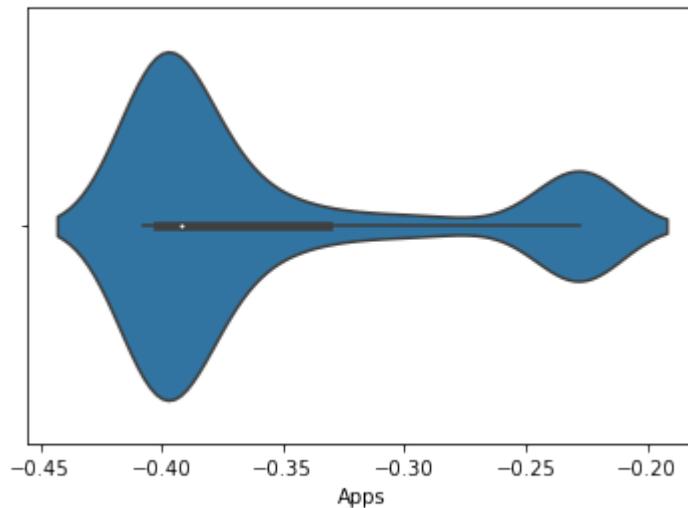
```
Out[270]: <AxesSubplot:xlabel='Apps'>
```



From above figure, we can say that the Apps parameter is highly right skewed

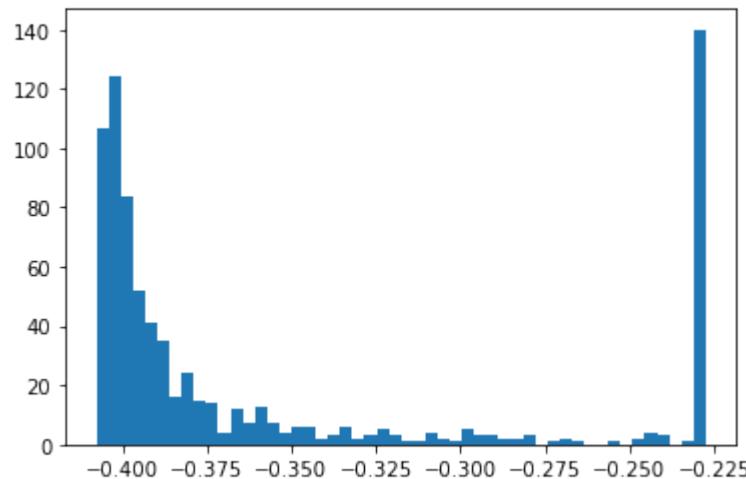
```
In [271]: sns.violinplot(df['Apps'])
```

```
Out[271]: <AxesSubplot:xlabel='Apps'>
```



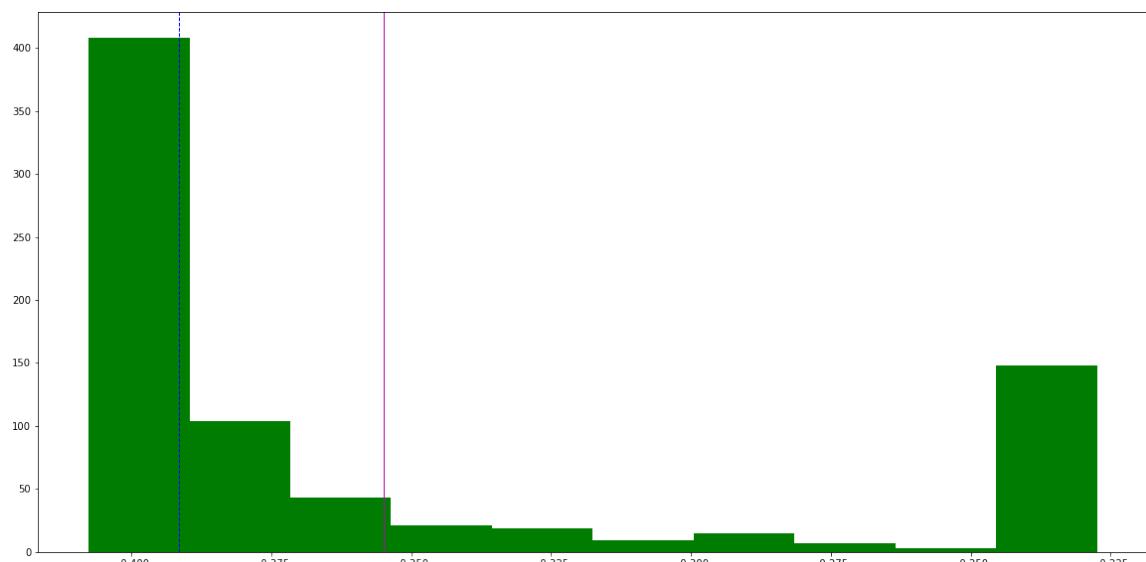
In [272]: `plt.hist(df['Apps'], bins=50)`

Out[272]: (array([107., 124., 84., 52., 41., 35., 16., 24., 15., 14., 4., 12., 7., 13., 7., 4., 6., 6., 2., 3., 6., 2., 3., 5., 3., 1., 1., 4., 2., 1., 5., 3., 3., 2., 2., 3., 0., 1., 2., 1., 0., 0., 1., 0., 2., 4., 3., 0., 1., 140.]), array([-0.40768319, -0.40407832, -0.40047345, -0.39686858, -0.39326371, -0.38965883, -0.38605396, -0.38244909, -0.37884422, -0.37523935, -0.37163447, -0.3680296, -0.36442473, -0.36081986, -0.35721499, -0.35361011, -0.35000524, -0.34640037, -0.3427955, -0.33919063, -0.33558576, -0.33198088, -0.32837601, -0.32477114, -0.32116627, -0.3175614, -0.31395652, -0.31035165, -0.30674678, -0.30314191, -0.29953704, -0.29593217, -0.29232729, -0.28872242, -0.28511755, -0.28151268, -0.27790781, -0.27430293, -0.27069806, -0.26709319, -0.26348832, -0.25988345, -0.25627857, -0.2526737, -0.24906883, -0.24546396, -0.24185909, -0.23825422, -0.23464934, -0.23104447, -0.2274396]), <BarContainer object of 50 artists>)



In [273]: `plt.figure(figsize=(20,10))
plt.hist(df["Apps"], color='g')
plt.axvline(df['Apps'].mean(), color='m', linewidth=1)
plt.axvline(df['Apps'].median(), color='b', linestyle='dashed', linewidth=1)
plt.axvline(df['Apps'].mode()[0], color='w', linestyle='dashed', linewidth=1)`

Out[273]: <matplotlib.lines.Line2D at 0x2223fb3a670>

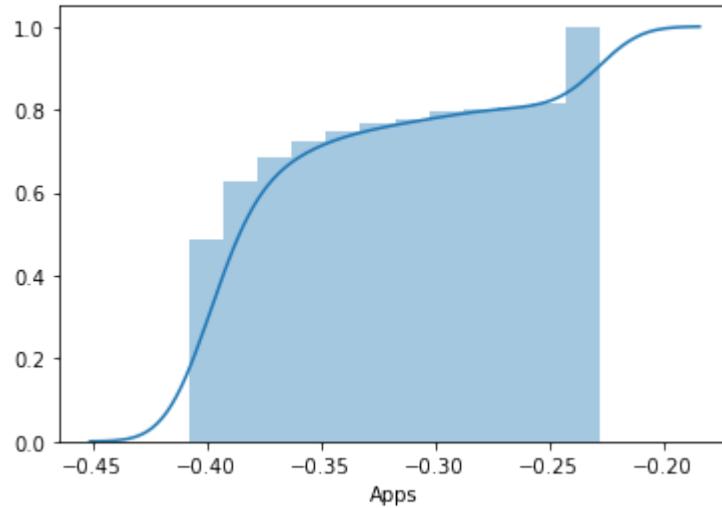


In the above histogram the first bin with class limits (1041.26, 6802.82) has the most number of observations. A bulk of the observations lie within the first 2 classes. The rest of the 3 classes contain only a very few observations. In this case we are measuring various parameters. We can say based on visual observation that most of the parameters present on the data are less than 10000.

Cumulative distribution

```
In [274]: sns.distplot(df['Apps'], hist_kws=dict(cumulative=True), kde_kws=dict(cumula
```

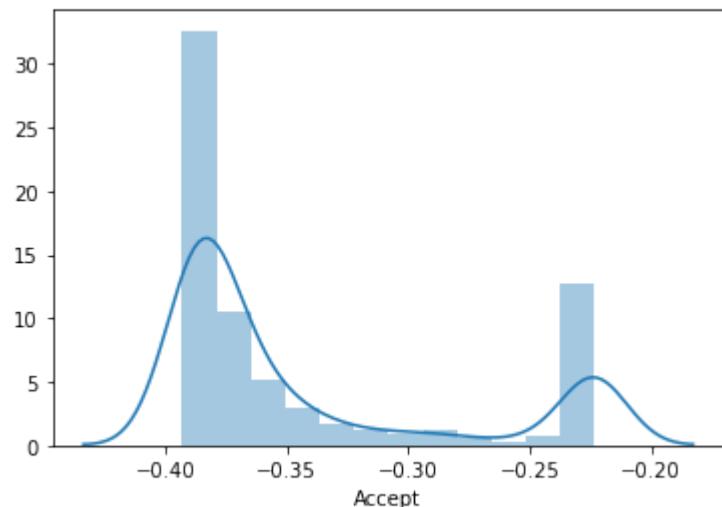
```
Out[274]: <AxesSubplot:xlabel='Apps'>
```



Number of application accepted

```
In [275]: sns.distplot(df["Accept"])
```

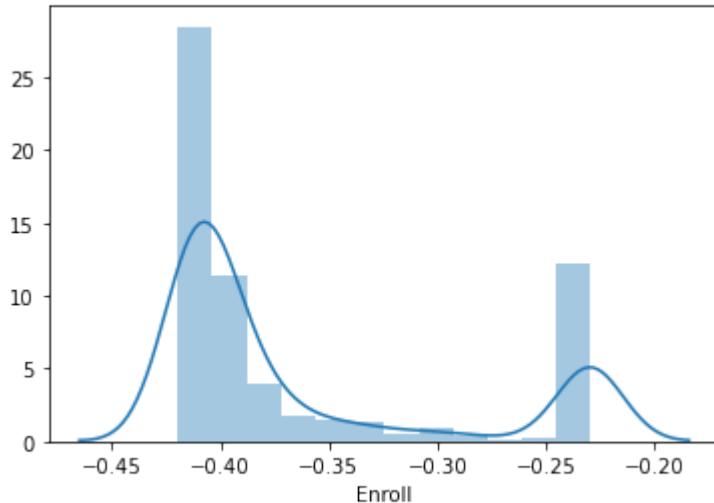
```
Out[275]: <AxesSubplot:xlabel='Accept'>
```



Number of new students enrolled

```
In [276]: sns.distplot(df["Enroll"])
```

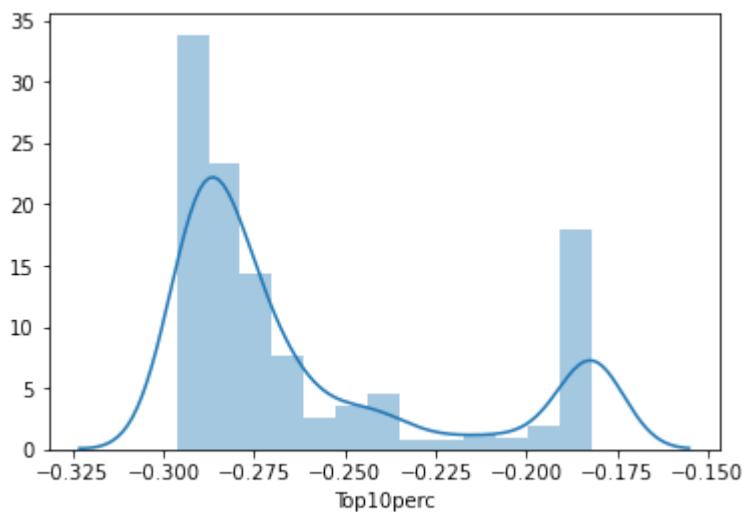
```
Out[276]: <AxesSubplot:xlabel='Enroll'>
```



Percentage of new students from top 10% of Higher Secondary class

```
In [277]: sns.distplot(df["Top10perc"])
```

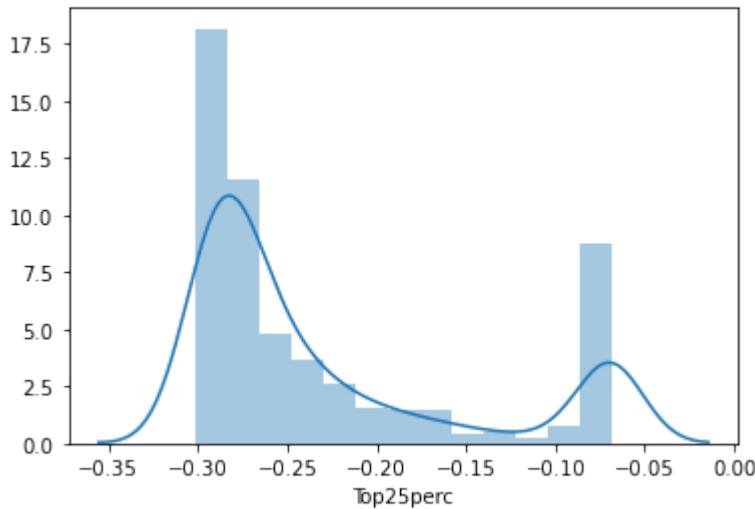
```
Out[277]: <AxesSubplot:xlabel='Top10perc'>
```



Percentage of new students from top 25% of Higher Secondary class

```
In [278]: sns.distplot(df["Top25perc"])
```

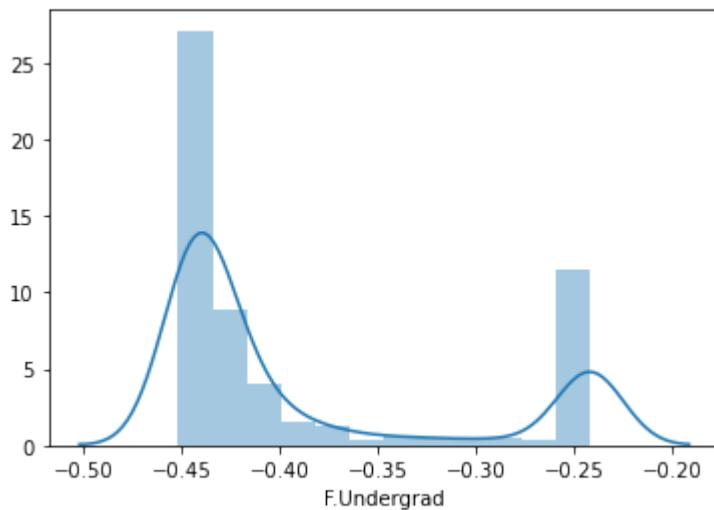
```
Out[278]: <AxesSubplot:xlabel='Top25perc'>
```



Number of full-time undergraduate students

```
In [279]: sns.distplot(df["F.Undergrad"])
```

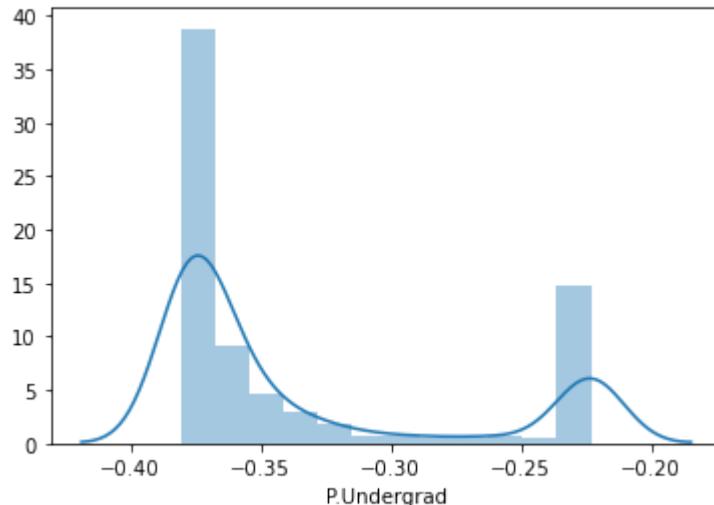
```
Out[279]: <AxesSubplot:xlabel='F.Undergrad'>
```



Number of part-time undergraduate students

```
In [280]: sns.distplot(df["P.Undergrad"])
```

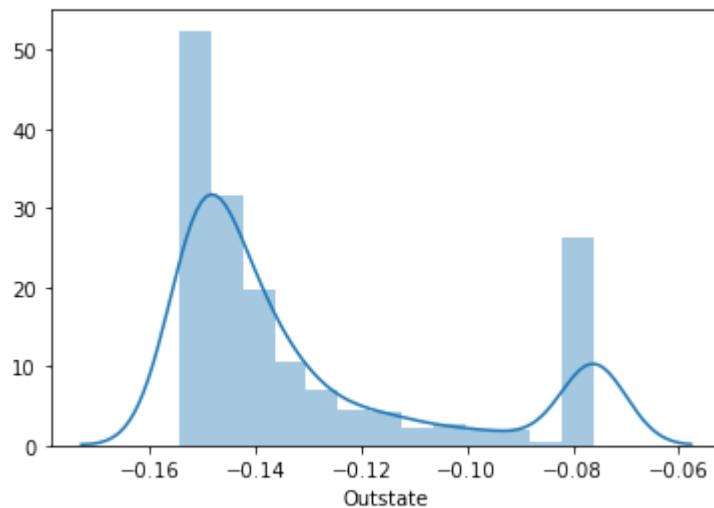
```
Out[280]: <AxesSubplot:xlabel='P.Undergrad'>
```



Number of students for whom the particular college or university is Out-of-state tuition

```
In [281]: sns.distplot(df["Outstate"])
```

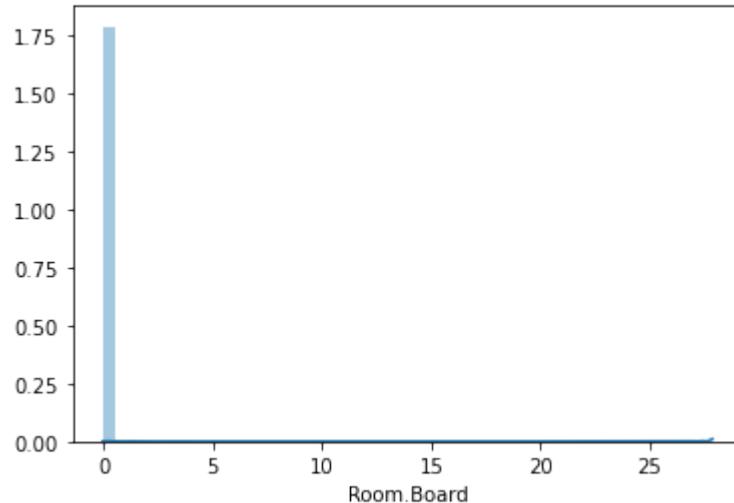
```
Out[281]: <AxesSubplot:xlabel='Outstate'>
```



Cost of Room and board

```
In [282]: sns.distplot(df["Room.Board"])
```

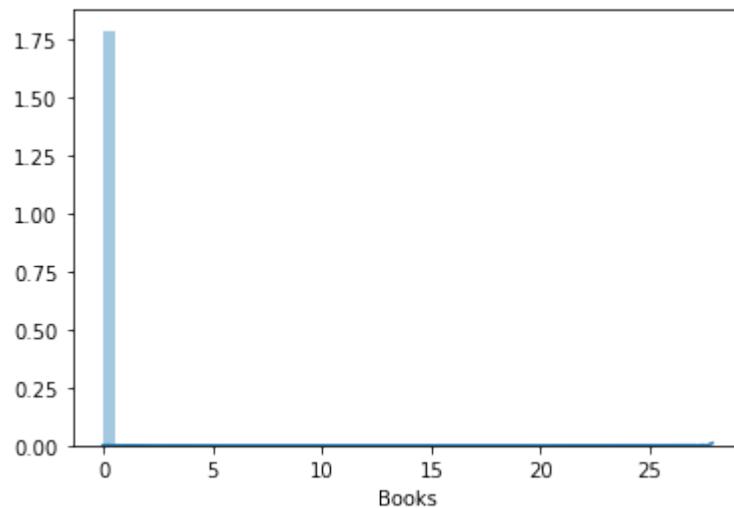
```
Out[282]: <AxesSubplot:xlabel='Room.Board'>
```



Estimated book costs for a student

```
In [283]: sns.distplot(df["Books"])
```

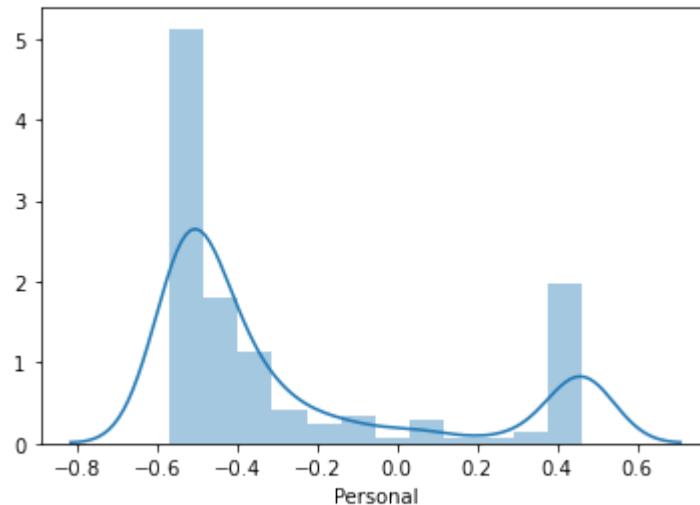
```
Out[283]: <AxesSubplot:xlabel='Books'>
```



Estimated personal spending for a student

```
In [284]: sns.distplot(df["Personal"])
```

```
Out[284]: <AxesSubplot:xlabel='Personal'>
```

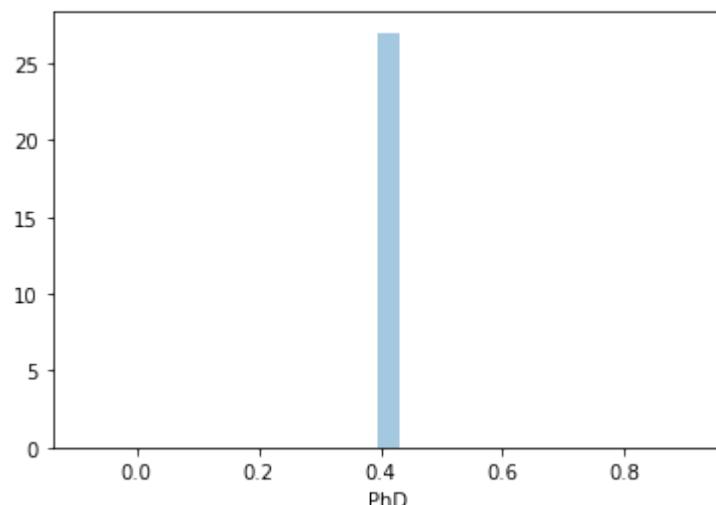


Percentage of faculties with Ph.D.'s

```
In [285]: sns.distplot(df["PhD"])
```

```
C:\Users\Admin\anaconda4\lib\site-packages\seaborn\distributions.py:369: UserWarning: Default bandwidth for data is 0; skipping density estimation.  
warnings.warn(msg, UserWarning)
```

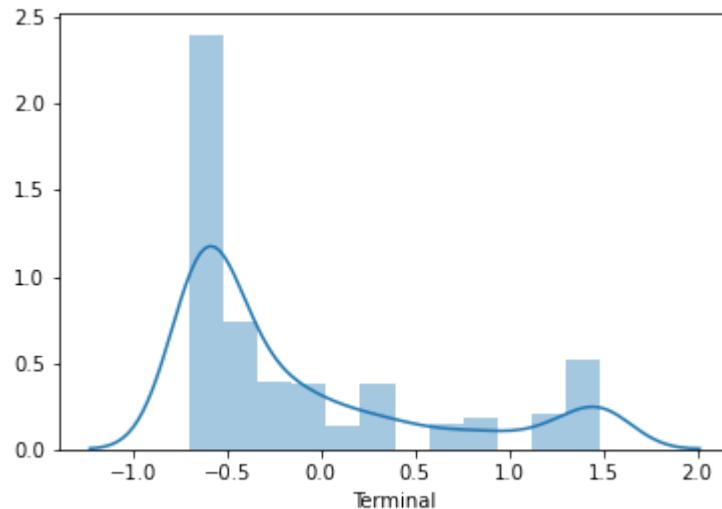
```
Out[285]: <AxesSubplot:xlabel='PhD'>
```



Percentage of faculties with terminal degree

```
In [286]: sns.distplot(df["Terminal"])
```

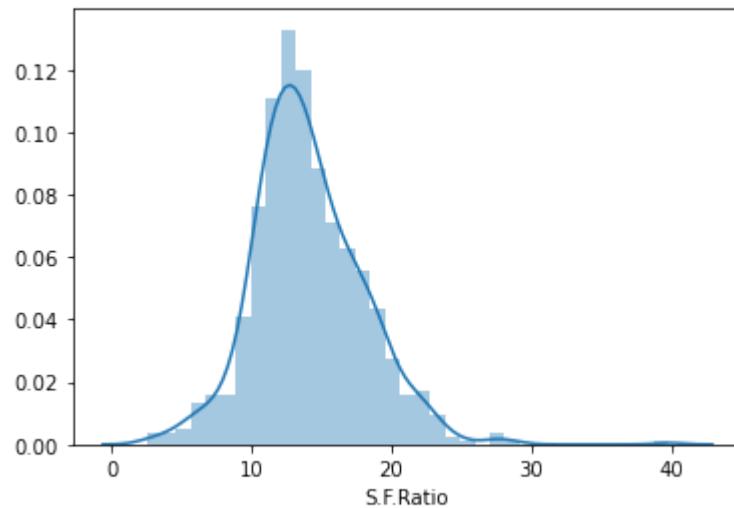
```
Out[286]: <AxesSubplot:xlabel='Terminal'>
```



Student/faculty ratio

```
In [287]: sns.distplot(df["S.F.Ratio"])
```

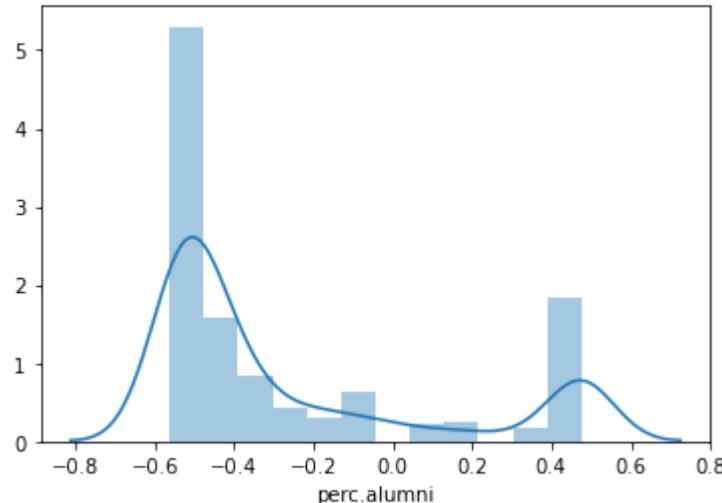
```
Out[287]: <AxesSubplot:xlabel='S.F.Ratio'>
```



Percentage of alumni who donate

```
In [288]: sns.distplot(df["perc.alumni"])
```

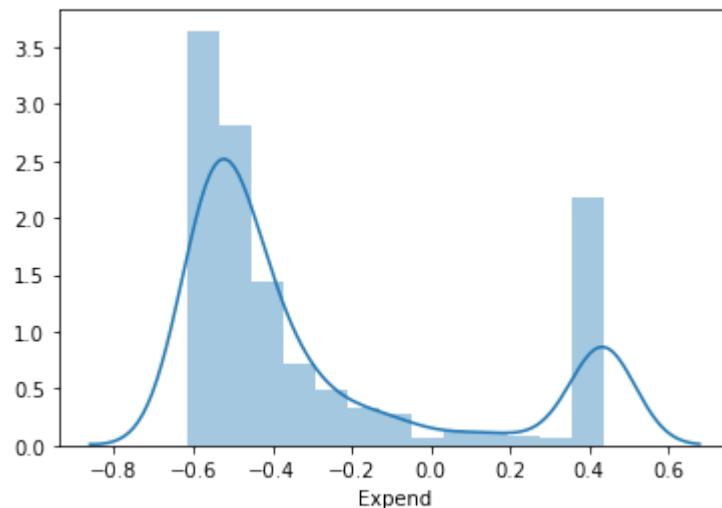
```
Out[288]: <AxesSubplot:xlabel='perc.alumni'>
```



The Instructional expenditure per student

```
In [289]: sns.distplot(df["Expend"])
```

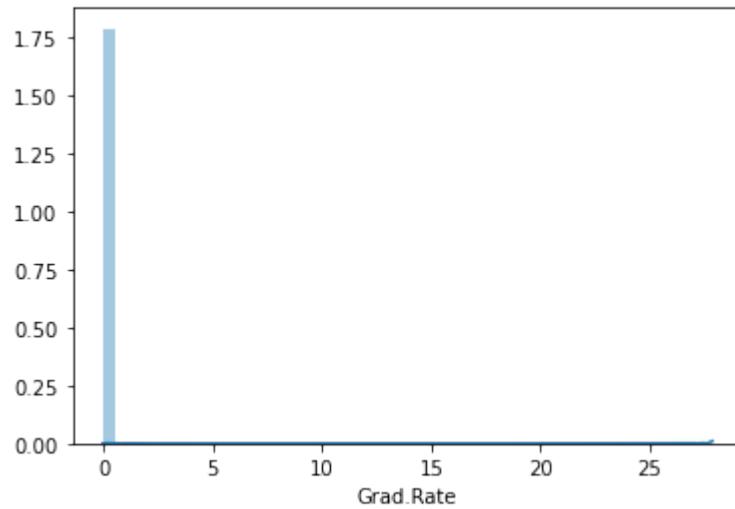
```
Out[289]: <AxesSubplot:xlabel='Expend'>
```



Graduation rate

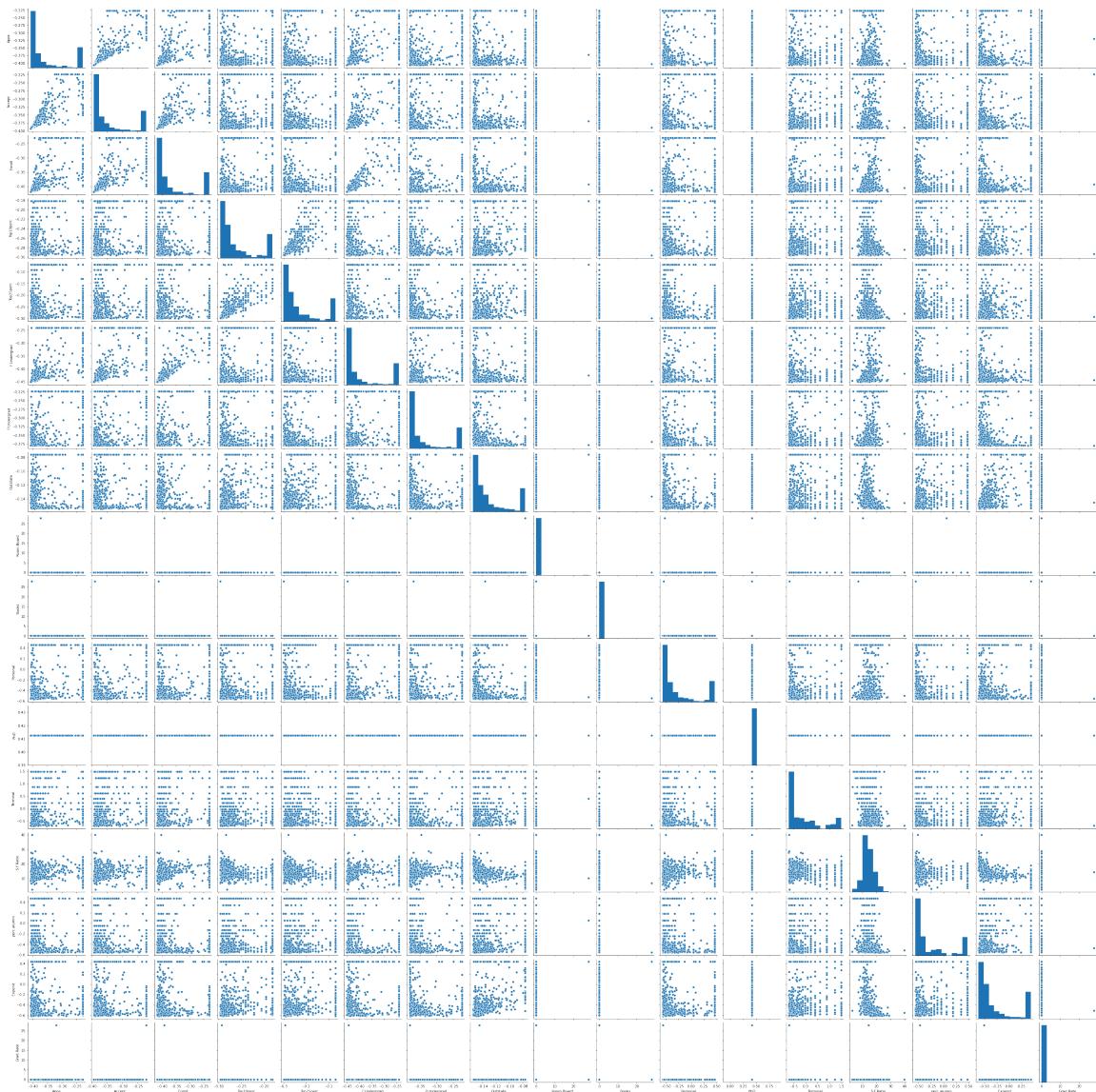
```
In [290]: sns.distplot(df["Grad.Rate"])
```

```
Out[290]: <AxesSubplot:xlabel='Grad.Rate'>
```



Bivariate Analysis

```
In [292]: sns.pairplot(df);      #Pairplot of all variables
```



In the above plot scatter diagrams are plotted for all the numerical columns in the dataset. A scatter plot is a visual representation of the degree of correlation between any two columns. The pair plot function in seaborn makes it very easy to generate joint scatter plots for all the columns in the data.

In [163]: df.corr()

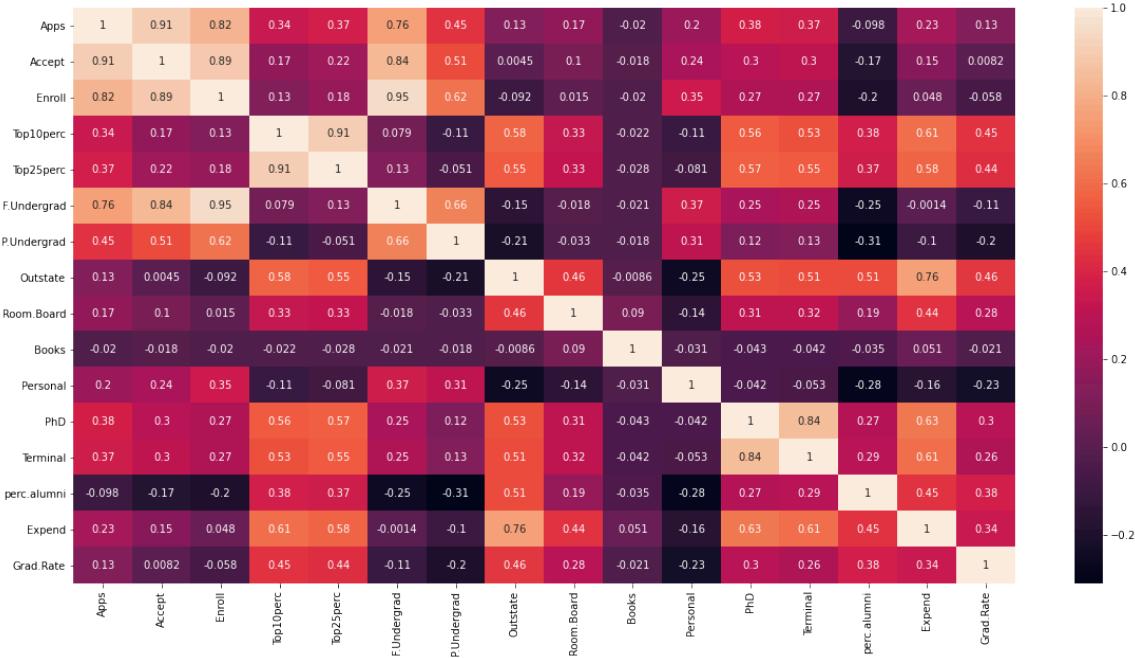
Out[163]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad
Apps	1.000000	0.905832	0.821709	0.341434	0.374126	0.762954	0.445846
Accept	0.905832	1.000000	0.891860	0.171606	0.216153	0.836145	0.511618
Enroll	0.821709	0.891860	1.000000	0.133346	0.183667	0.950968	0.618496
Top10perc	0.341434	0.171606	0.133346	1.000000	0.911219	0.078718	-0.108971
Top25perc	0.374126	0.216153	0.183667	0.911219	1.000000	0.128763	-0.051297
F.Undergrad	0.762954	0.836145	0.950968	0.078718	0.128763	1.000000	0.657372
P.Undergrad	0.445846	0.511618	0.618496	-0.108971	-0.051297	0.657372	1.000000
Outstate	0.132916	0.004480	-0.092102	0.576708	0.554635	-0.145010	-0.211103
Room.Board	0.171813	0.102521	0.014764	0.332950	0.327937	-0.017834	-0.032876
Books	-0.019822	-0.017775	-0.019541	-0.021818	-0.028204	-0.020972	-0.018422
Personal	0.204271	0.243302	0.351389	-0.108558	-0.081242	0.369988	0.312653
PhD	0.384999	0.302116	0.268291	0.559219	0.565023	0.245846	0.119797
Terminal	0.367588	0.304608	0.267850	0.533230	0.551476	0.254479	0.126406
perc.alumni	-0.097793	-0.172961	-0.203800	0.377161	0.367680	-0.249811	-0.309651
Expend	0.230684	0.146255	0.048412	0.606885	0.579190	-0.001391	-0.100884
Grad.Rate	0.125807	0.008189	-0.057899	0.447156	0.435555	-0.109682	-0.198985

Correlation Heatmap

In [164]: `plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), annot=True)`

Out[164]: <AxesSubplot:>



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

Normalizing and Scaling

Often the variables of the data set are of different scales i.e. one variable is in millions and other in only 100. For e.g. in our data set Income is having values in thousands and age in just two digits. Since the data in these variables are of different scales, it is tough to compare these variables.

Feature scaling (also known as data normalization) is the method used to standardize the range of features of data. Since, the range of values of data may vary widely, it becomes a necessary step in data preprocessing while using machine learning algorithms.

In this method, we convert variables with different scales of measurements into a single scale.

StandardScaler normalizes the data using the formula $(x - \text{mean})/\text{standard deviation}$.

In [165]: df.head()

Out[165]:

	Names	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad
0	Abilene Christian University	0.686264	0.713317	1.112245	0.781312	0.825449	0.981406	0.846976
1	Adelphi University	0.852830	1.123831	0.770712	0.498431	0.258228	0.910624	2.220775
2	Adrian College	0.623544	0.652783	0.565897	0.732717	0.746113	0.494620	0.459335
3	Agnes Scott College	0.410658	0.399368	0.399072	8.408153	5.352758	0.407022	0.436806
4	Alaska Pacific University	0.374362	0.349511	0.345579	0.498431	0.550994	0.369498	1.346792



In [166]: #Scales the data. Essentially returns the z-scores of every attribute

```
from sklearn.preprocessing import StandardScaler  
  
std_scale = StandardScaler()  
  
std_scale
```

Out[166]: StandardScaler()

```
In [402]: df['Apps'] = std_scale.fit_transform(df[['Apps']])# returns z-scores of the
df['Accept'] = std_scale.fit_transform(df[['Accept']])
df['Enroll'] = std_scale.fit_transform(df[['Enroll']])
df['Top10perc'] = std_scale.fit_transform(df[['Top10perc']])

df['Top25perc'] = std_scale.fit_transform(df[['Top25perc']])

df['F.Undergrad'] = std_scale.fit_transform(df[['F.Undergrad']])

df['P.Undergrad'] = std_scale.fit_transform(df[['P.Undergrad']])

df['Outstate'] = std_scale.fit_transform(df[['Outstate']])

df['Room.Board'] = std_scale.fit_transform(df[['Room.Board']])

df['Books'] = std_scale.fit_transform(df[['Books']])

df['Personal'] = std_scale.fit_transform(df[['Personal']])

df['PhD'] = std_scale.fit_transform(df[['PhD']])

df['Terminal'] = std_scale.fit_transform(df[['Terminal']])

df['S.F.Ratio'] = std_scale.fit_transform(df[['S.F.Ratio']])

df['perc.alumni'] = std_scale.fit_transform(df[['perc.alumni']])

df['Expend'] = std_scale.fit_transform(df[['Expend']])

df['Grad.Rate'] = std_scale.fit_transform(df[['Grad.Rate']])
```

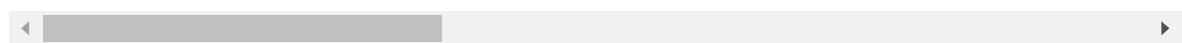
```
In [403]: df.head()
```

	Names	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad
0	Abilene Christian University	-0.376493	-0.337830	0.106380	-0.246780	-0.191827	-0.018769	-0.166083
1	Adelphi University	-0.159195	0.116744	-0.260441	-0.696290	-1.353911	-0.093626	0.797856
2	Adrian College	-0.472336	-0.426511	-0.569343	-0.310996	-0.292878	-0.703966	-0.777974
3	Agnes Scott College	-0.889994	-0.917871	-0.918613	2.129202	1.677612	-0.898889	-0.828267
4	Alaska Pacific University	-0.982532	-1.051221	-1.062533	-0.696290	-0.596031	-0.995610	0.297726

In [404]: `df.describe()`

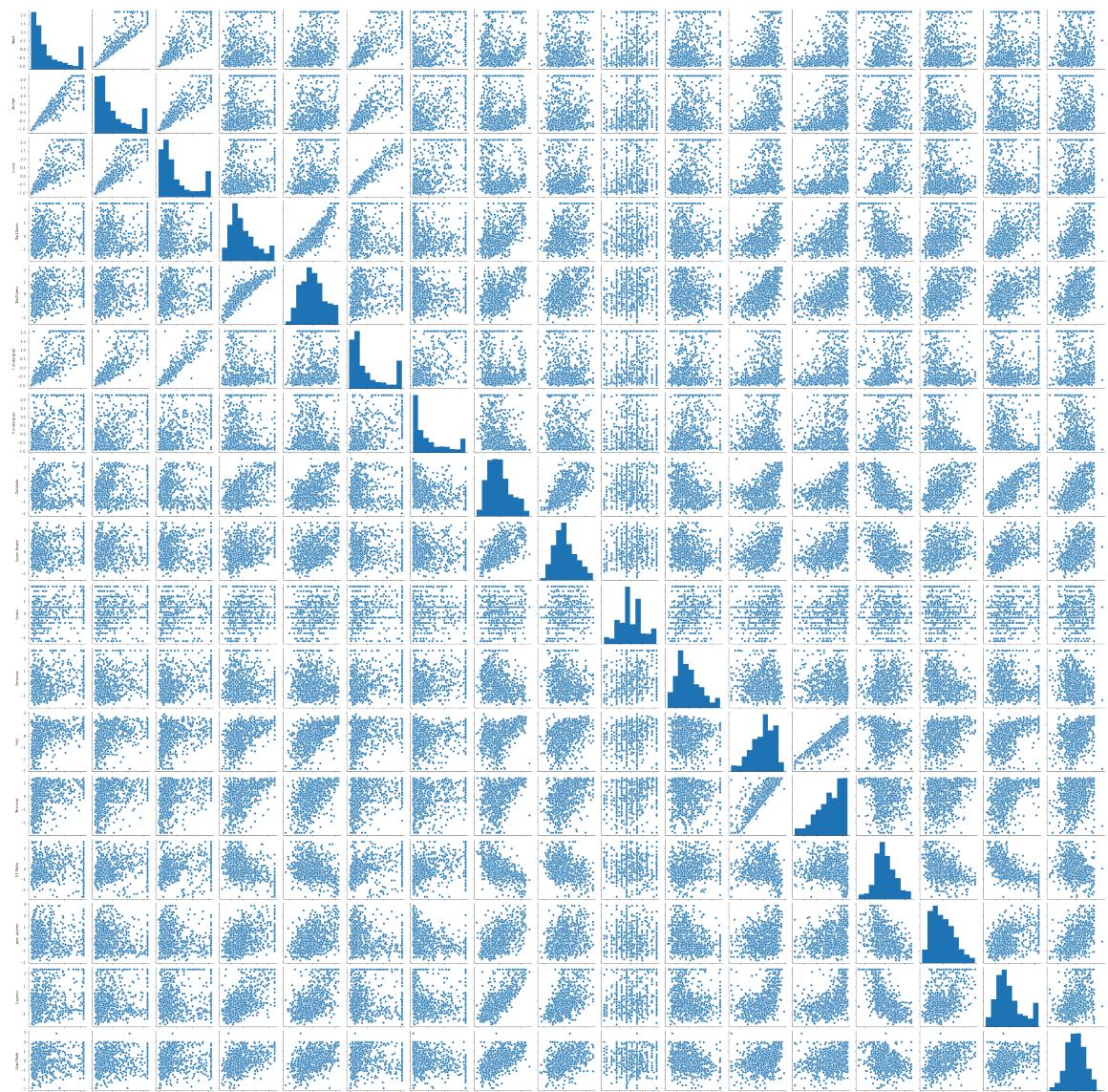
Out[404]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad
count	7.770000e+02	7.770000e+02	7.770000e+02	7.770000e+02	7.770000e+02	7.770000e+02
mean	1.234534e-16	1.340626e-16	1.521645e-16	-2.250452e-18	-1.546739e-16	-1.911679e-16
std	1.000644e+00	1.000644e+00	1.000644e+00	1.000644e+00	1.000644e+00	1.000644e+00
min	-1.028801e+00	-1.099832e+00	-1.097636e+00	-1.659526e+00	-2.364419e+00	-1.036373e+00
25%	-7.416863e-01	-7.503620e-01	-7.343246e-01	-7.605060e-01	-7.476067e-01	-7.202711e-01
50%	-4.186307e-01	-4.179715e-01	-3.973405e-01	-2.467801e-01	-9.077663e-02	-4.553086e-01
75%	4.348639e-01	4.451926e-01	4.240582e-01	5.238086e-01	6.671042e-01	3.962772e-01
max	2.199689e+00	2.238524e+00	2.161632e+00	2.450281e+00	2.233391e+00	2.071100e+00



In [406]: `sns.pairplot(df)`

Out[406]: <seaborn.axisgrid.PairGrid at 0x2225835b850>



In [407]: `df.corr()`

Out[407]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad
Apps	1.000000	0.955307	0.896883	0.321342	0.364491	0.861002	0.519823
Accept	0.955307	1.000000	0.935277	0.223298	0.273681	0.897034	0.572691
Enroll	0.896883	0.935277	1.000000	0.171756	0.230434	0.967302	0.641595
Top10perc	0.321342	0.223298	0.171756	1.000000	0.913875	0.111215	-0.180009
Top25perc	0.364491	0.273681	0.230434	0.913875	1.000000	0.181196	-0.099295
F.Undergrad	0.861002	0.897034	0.967302	0.111215	0.181196	1.000000	0.696130
P.Undergrad	0.519823	0.572691	0.641595	-0.180009	-0.099295	0.696130	1.000000
Outstate	0.065337	-0.005002	-0.155655	0.562160	0.489569	-0.226166	-0.354216
Room.Board	0.187475	0.119586	-0.023846	0.357366	0.330987	-0.054476	-0.067638
Books	0.236138	0.208705	0.202057	0.153452	0.169761	0.207879	0.122529
Personal	0.229948	0.256346	0.339348	-0.116730	-0.086810	0.359783	0.344053
PhD	0.463924	0.427341	0.381540	0.544048	0.551461	0.361564	0.127663
Terminal	0.434478	0.403409	0.354379	0.506748	0.527654	0.335054	0.122152
S.F.Ratio	0.126411	0.188506	0.274269	-0.387926	-0.297233	0.324504	0.370607
perc.alumni	-0.101158	-0.165516	-0.222723	0.455797	0.416832	-0.285457	-0.419334
Expend	0.242935	0.161808	0.054221	0.657039	0.572905	0.000371	-0.201929
Grad.Rate	0.150803	0.078982	-0.023251	0.493670	0.478985	-0.082239	-0.265158

Exponential Transformation

In [171]: `from sklearn.preprocessing import FunctionTransformer
exp_transformer = FunctionTransformer(np.exp) # Exponential transform

exp_transformer`

Out[171]: `FunctionTransformer(func=<ufunc 'exp'>)`

```
In [408]: df['Apps'] = exp_transformer.fit_transform(df[['Apps']])
df['Accept'] = exp_transformer.fit_transform(df[['Accept']])
df['Enroll'] = exp_transformer.fit_transform(df[['Enroll']])
df['Top10perc'] = exp_transformer.fit_transform(df[['Top10perc']])
df['Top25perc'] = exp_transformer.fit_transform(df[['Top25perc']])
df['F.Undergrad'] = exp_transformer.fit_transform(df[['F.Undergrad']])
df['P.Undergrad'] = exp_transformer.fit_transform(df[['P.Undergrad']])
df['Outstate'] = exp_transformer.fit_transform(df[['Outstate']])
df['Room.Board'] = exp_transformer.fit_transform(df[['Room.Board']])
df['Books'] = exp_transformer.fit_transform(df[['Books']])
df['Personal'] = exp_transformer.fit_transform(df[['Personal']])
df['PhD'] = exp_transformer.fit_transform(df[['PhD']])
df['Terminal'] = exp_transformer.fit_transform(df[['Terminal']])
df['S.F.Ratio'] = exp_transformer.fit_transform(df[['S.F.Ratio']])
df['perc.alumni'] = exp_transformer.fit_transform(df[['perc.alumni']])
df['Expend'] = exp_transformer.fit_transform(df[['Expend']])
df['Grad.Rate'] = exp_transformer.fit_transform(df[['Grad.Rate']])

#returns the exponential transform of the data
```

```
In [409]: df.head()
```

	Names	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad
0	Abilene Christian University	0.686264	0.713317	1.112245	0.781312	0.825449	0.981406	0.846976
1	Adelphi University	0.852830	1.123831	0.770712	0.498431	0.258228	0.910624	2.220775
2	Adrian College	0.623544	0.652783	0.565897	0.732717	0.746113	0.494620	0.459335
3	Agnes Scott College	0.410658	0.399368	0.399072	8.408153	5.352758	0.407022	0.436806
4	Alaska Pacific University	0.374362	0.349511	0.345579	0.498431	0.550994	0.369498	1.346792

```
In [297]: df.describe()
```

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad
count	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000
mean	0.702824	0.711228	0.697356	0.771383	0.797822	0.680225	0.716669
std	0.048994	0.045871	0.051336	0.031260	0.066926	0.056904	0.044585
min	0.665190	0.675056	0.657310	0.743427	0.740028	0.636521	0.683653
25%	0.669300	0.679530	0.662614	0.748897	0.750129	0.641928	0.685414
50%	0.676000	0.685932	0.670130	0.755863	0.764869	0.648386	0.691962
75%	0.717567	0.725296	0.712742	0.781741	0.818683	0.695919	0.729135
max	0.796571	0.799787	0.795135	0.833728	0.933439	0.785539	0.800000

In [410]: `df.corr()`

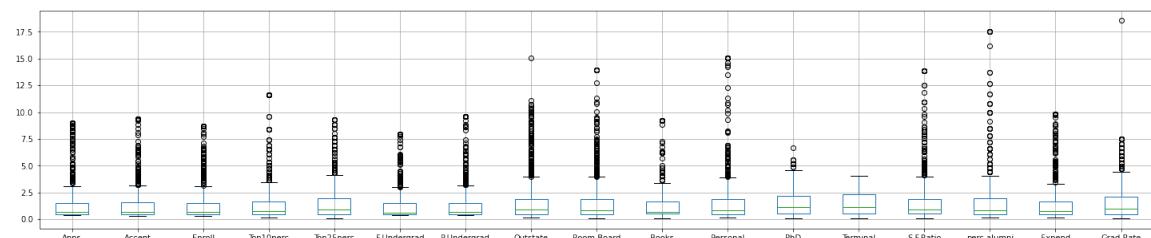
Out[410]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad
Apps	1.000000	0.905832	0.821709	0.341434	0.374126	0.762954	0.445846
Accept	0.905832	1.000000	0.891860	0.171606	0.216153	0.836145	0.511618
Enroll	0.821709	0.891860	1.000000	0.133346	0.183667	0.950968	0.618496
Top10perc	0.341434	0.171606	0.133346	1.000000	0.911219	0.078718	-0.108971
Top25perc	0.374126	0.216153	0.183667	0.911219	1.000000	0.128763	-0.051297
F.Undergrad	0.762954	0.836145	0.950968	0.078718	0.128763	1.000000	0.657372
P.Undergrad	0.445846	0.511618	0.618496	-0.108971	-0.051297	0.657372	1.000000
Outstate	0.132916	0.004480	-0.092102	0.576708	0.554635	-0.145010	-0.211103
Room.Board	0.192476	0.115243	0.017649	0.357313	0.336911	-0.018062	-0.034702
Books	0.148029	0.147019	0.142501	0.103459	0.142225	0.139734	0.073980
Personal	0.133968	0.181456	0.262421	-0.085780	-0.068912	0.295893	0.303030
PhD	0.384999	0.302116	0.268291	0.559219	0.565023	0.245846	0.119797
Terminal	0.367062	0.303764	0.266823	0.534287	0.552265	0.253421	0.125533
S.F.Ratio	0.119816	0.146627	0.227474	-0.189006	-0.165043	0.273443	0.319609
perc.alumni	-0.055582	-0.138333	-0.147226	0.420168	0.405392	-0.180146	-0.225461
Expend	0.265421	0.146660	0.061760	0.687139	0.627762	0.015255	-0.089111
Grad.Rate	0.128861	0.007452	-0.058226	0.458544	0.446950	-0.110923	-0.202168

The data above shows that the Exponential transformation is best suited for scaling for the given data

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

In [413]: `plt.figure(figsize=(25,5))
df.boxplot()
plt.show()`



In []: The above graph shows that after scaling there are many outliers in the given

In [303]: `data_df=pd.read_excel('Education - Post 12th Standard-1.xlsx')
data_df.head()`

Out[303]:

	Names	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstat
0	Abilene Christian University	1660	1232	721	23	52	2885	537	744
1	Adelphi University	2186	1924	512	16	29	2683	1227	1228
2	Adrian College	1428	1097	336	22	50	1036	99	1125
3	Agnes Scott College	417	349	137	60	89	510	63	1296
4	Alaska Pacific University	193	146	55	16	44	249	869	756

In [302]: `data_df.describe()`

Out[302]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Und
count	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	777
mean	3001.638353	2018.804376	779.972973	27.558559	55.796654	3699.907336	855
std	3870.201484	2451.113971	929.176190	17.640364	19.804778	4850.420531	1522
min	81.000000	72.000000	35.000000	1.000000	9.000000	139.000000	1
25%	776.000000	604.000000	242.000000	15.000000	41.000000	992.000000	95
50%	1558.000000	1110.000000	434.000000	23.000000	54.000000	1707.000000	353
75%	3624.000000	2424.000000	902.000000	35.000000	69.000000	4005.000000	967
max	48094.000000	26330.000000	6392.000000	96.000000	100.000000	31643.000000	21836

In [304]: `data_df['Names']`

Out[304]:

```
0      Abilene Christian University
1      Adelphi University
2      Adrian College
3      Agnes Scott College
4      Alaska Pacific University
      ...
772     Worcester State College
773     Xavier University
774     Xavier University of Louisiana
775     Yale University
776     York College of Pennsylvania
Name: Names, Length: 777, dtype: object
```

Check for any null values

```
In [305]: data_df.isnull().sum()
```

```
Out[305]: 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
```

There are no null values

```
In [307]: #check for duplicates  
data_df.duplicated().sum()
```

```
Out[307]: 0
```

There are no duplicate values

Univariate Analysis


```
In [323]: fig,axes = plt.subplots(nrows=18,ncols=2)
fig.set_size_inches(15,40)

a= sns.distplot(data_df[ 'Apps' ], ax=axes[0][0])
a.set_title('Apps Distribution',fontsize =15)

a= sns.boxplot(data_df[ 'Apps' ], orient = 'v', ax=axes[0][1])
a.set_title('Apps Distribution', fontsize =15)

a= sns.distplot(data_df[ 'Accept' ], ax=axes[1][0])
a.set_title('Accept Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'Accept' ], orient = 'v', ax=axes[1][1])
a.set_title('Accept Distribution', fontsize =15)

a= sns.distplot(data_df[ 'Enroll' ], ax=axes[2][0])
a.set_title('Enroll Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'Enroll' ], orient = 'v', ax=axes[2][1])
a.set_title('Enroll Distribution', fontsize =15)

a= sns.distplot(data_df[ 'Top10perc' ], ax=axes[3][0])
a.set_title('Top10perc Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'Top10perc' ], orient = 'v', ax=axes[3][1])
a.set_title('Top10perc Distribution', fontsize =15)

a= sns.distplot(data_df[ 'Top25perc' ], ax=axes[3][0])
a.set_title('Top25perc Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'Top25perc' ], orient = 'v', ax=axes[3][1])
a.set_title('Top25perc Distribution', fontsize =15)

a= sns.distplot(data_df[ 'F.Undergrad' ], ax=axes[4][0])
a.set_title('F.Undergrad Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'F.Undergrad' ], orient = 'v', ax=axes[4][1])
a.set_title('F.Undergrad Distribution', fontsize =15)

a= sns.distplot(data_df[ 'P.Undergrad' ], ax=axes[5][0])
a.set_title('P.Undergrad Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'P.Undergrad' ], orient = 'v', ax=axes[5][1])
a.set_title('P.Undergrad Distribution', fontsize =15)

a= sns.distplot(data_df[ 'Outstate' ], ax=axes[6][0])
a.set_title('Outstate Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'Outstate' ], orient = 'v', ax=axes[6][1])
a.set_title('Outstate Distribution', fontsize =15)

a= sns.distplot(data_df[ 'Room.Board' ], ax=axes[6][0])
a.set_title('Room.Board Distribution', fontsize =15)
```

```
a= sns.boxplot(data_df[ 'Room.Board'], orient = 'v', ax=axes[6][1])
a.set_title('Room.Board Distribution', fontsize =15)

a= sns.distplot(data_df[ 'Books'], ax=axes[7][0])
a.set_title('Books Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'Books'], orient = 'v', ax=axes[7][1])
a.set_title('Books Distribution', fontsize =15)

a= sns.distplot(data_df[ 'Personal'], ax=axes[8][0])
a.set_title('Personal Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'Personal'], orient = 'v', ax=axes[8][1])
a.set_title('Personal Distribution', fontsize =15)

a= sns.distplot(data_df[ 'PhD'], ax=axes[9][0])
a.set_title('PhD Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'PhD'], orient = 'v', ax=axes[9][1])
a.set_title('PhD Distribution', fontsize =15)

a= sns.distplot(data_df[ 'Terminal'], ax=axes[10][0])
a.set_title('Terminal Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'Terminal'], orient = 'v', ax=axes[10][1])
a.set_title('Terminal Distribution', fontsize =15)

a= sns.distplot(data_df[ 'S.F.Ratio'], ax=axes[11][0])
a.set_title('S.F.Ratio Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'S.F.Ratio'], orient = 'v', ax=axes[11][1])
a.set_title('S.F.Ratio Distribution', fontsize =15)

a= sns.distplot(data_df[ 'perc.alumni'], ax=axes[12][0])
a.set_title('perc.alumni Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'perc.alumni'], orient = 'v', ax=axes[12][1])
a.set_title('perc.alumni Distribution', fontsize =15)

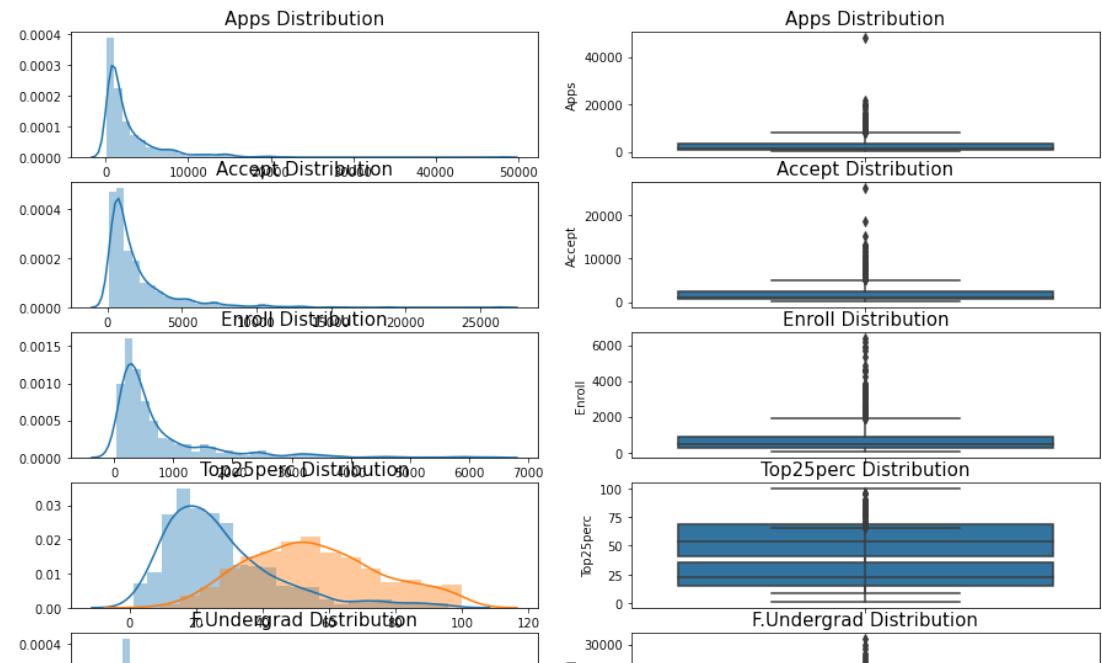
a= sns.distplot(data_df[ 'Expend'], ax=axes[13][0])
a.set_title('Expend Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'Expend'], orient = 'v', ax=axes[13][1])
a.set_title('Expend Distribution', fontsize =15)

a= sns.distplot(data_df[ 'Grad.Rate'], ax=axes[14][0])
a.set_title('Grad.Rate Distribution', fontsize =15)

a= sns.boxplot(data_df[ 'Grad.Rate'], orient = 'v', ax=axes[14][1])
a.set_title('Grad.Rate Distribution', fontsize =15)

plt.show()
```



Bi variate Analysis

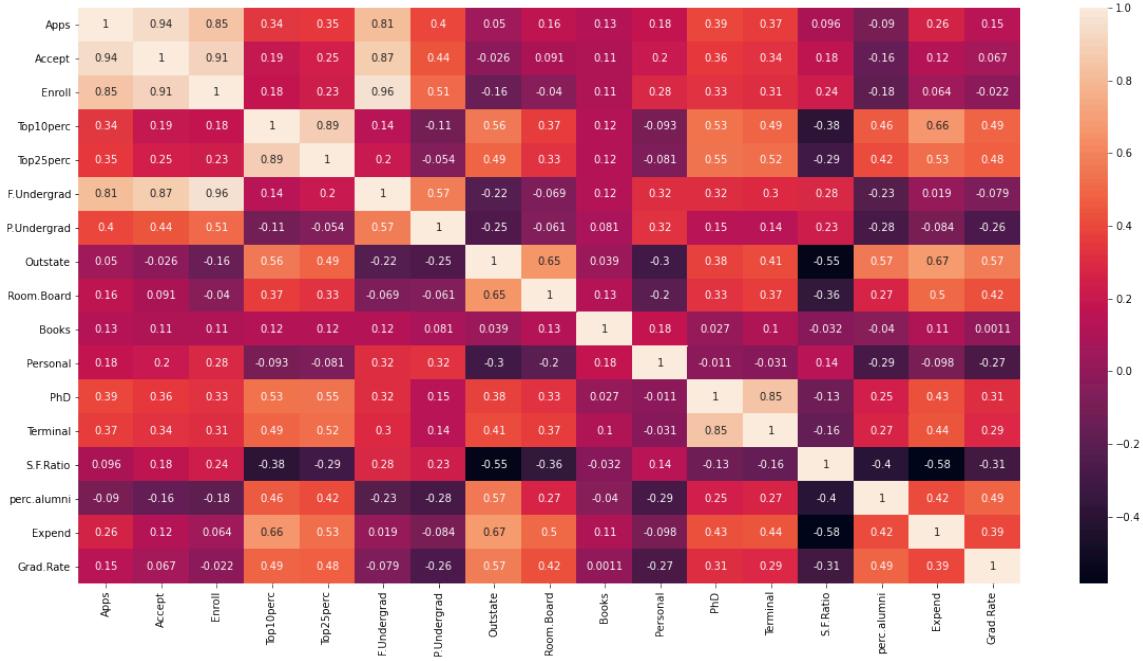
In [324]: `data_df.corr()`

Out[324]:

	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	0.149112
Terminal	0.369491	0.337583	0.308274	0.491135	0.524749	0.300019	0.141902
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

```
In [327]: plt.figure(figsize=(20,10))
sns.heatmap(data_df.corr(), annot=True)
```

Out[327]: <AxesSubplot:>



Lighter the color , it is highly corelated. Hence we need to perform PCA


```
In [361]: lrapps,urapps = remove_outliers(data_df['Apps'])
data_df['Apps'] = np.where(data_df['Apps'] > urapps, urapps, data_df['Apps'])

lraccept,uraccept = remove_outliers(data_df['Accept'])
data_df['Accept'] = np.where(data_df['Accept'] > uraccept, uraccept, data_df['Accept'])

lrenroll,urenroll = remove_outliers(data_df['Enroll'])
data_df['Enroll'] = np.where(data_df['Enroll'] > urenroll, urenroll, data_df['Enroll'])

lrTop10perc,urTop10perc = remove_outliers(data_df['Top10perc'])
data_df['Top10perc'] = np.where(data_df['Top10perc'] > urTop10perc, urTop10perc, data_df['Top10perc'])

lrTop25perc,urTop25perc = remove_outliers(data_df['Top25perc'])
data_df['Top25perc'] = np.where(data_df['Top25perc'] > urTop25perc, urTop25perc, data_df['Top25perc'])

lrFUndergrad,urFUndergrad = remove_outliers(data_df['F.Undergrad'])
data_df['F.Undergrad'] = np.where(data_df['F.Undergrad'] > urFUndergrad, urFUndergrad, data_df['F.Undergrad'])

lrPUndergrad,urPUndergrad = remove_outliers(data_df['P.Undergrad'])
data_df['P.Undergrad'] = np.where(data_df['P.Undergrad'] > urPUndergrad, urPUndergrad, data_df['P.Undergrad'])

lrOutstate,urOutstate = remove_outliers(data_df['Outstate'])
data_df['Outstate'] = np.where(data_df['Outstate'] > urOutstate, urOutstate, data_df['Outstate'])

lrRoomBoard,urRoomBoard = remove_outliers(data_df['Room.Board'])
data_df['Room.Board'] = np.where(data_df['Room.Board'] > urRoomBoard, urRoomBoard, data_df['Room.Board'])

lrBooks,urBooks = remove_outliers(data_df['Books'])
data_df['Books'] = np.where(data_df['Books'] > urBooks, urBooks, data_df['Books'])
data_df['Books'] = np.where(data_df['Books'] < lrBooks, lrBooks, data_df['Books'])

lrPersonal,urPersonal = remove_outliers(data_df['Personal'])
data_df['Personal'] = np.where(data_df['Personal'] > urPersonal, urPersonal, data_df['Personal'])

lrPhD,urPhD = remove_outliers(data_df['PhD'])
data_df['PhD'] = np.where(data_df['PhD'] > urPhD, urPhD, data_df['PhD'])
data_df['PhD'] = np.where(data_df['PhD'] < lrPhD, lrPhD, data_df['PhD'])

lrTerminal,urTerminal = remove_outliers(data_df['Terminal'])
data_df['Terminal'] = np.where(data_df['Terminal'] > urTerminal, urTerminal, data_df['Terminal'])
data_df['Terminal'] = np.where(data_df['Terminal'] < lrTerminal, lrTerminal, data_df['Terminal'])

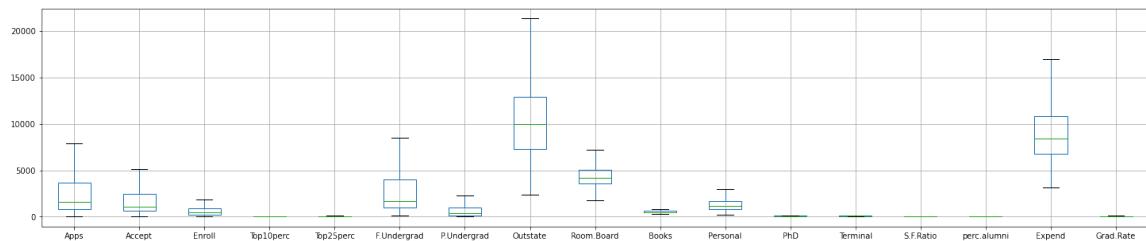
lrSFRatio,urSFRatio = remove_outliers(data_df['S.F.Ratio'])
data_df['S.F.Ratio'] = np.where(data_df['S.F.Ratio'] > urSFRatio, urSFRatio, data_df['S.F.Ratio'])
data_df['S.F.Ratio'] = np.where(data_df['S.F.Ratio'] < lrSFRatio, lrSFRatio, data_df['S.F.Ratio'])

lrpercalumni,urpercalumni = remove_outliers(data_df['perc.alumni'])
data_df['perc.alumni'] = np.where(data_df['perc.alumni'] > urpercalumni, urpercalumni, data_df['perc.alumni'])

lrExpend,urExpend = remove_outliers(data_df['Expend'])
data_df['Expend'] = np.where(data_df['Expend'] > urExpend, urExpend, data_df['Expend'])

lrGradRate,urGradRate = remove_outliers(data_df['Grad.Rate'])
data_df['Grad.Rate'] = np.where(data_df['Grad.Rate'] < lrGradRate, lrGradRate, data_df['Grad.Rate'])
data_df['Grad.Rate'] = np.where(data_df['Grad.Rate'] > urGradRate, urGradRate, data_df['Grad.Rate'])
```

```
In [362]: plt.figure(figsize=(25,5))
data_df.boxplot()
plt.show()
```



Drop Class column

```
In [365]: # Since Names column has unique values , so will remove it from the dataset.
data_df = data_df.drop(['Names'], axis =1)
```

```
In [366]: data_df.head()
```

```
Out[366]:
```

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.E
0	1660.0	1232.0	721.0	23.0	52.0	2885.0	537.0	7440.0	3
1	2186.0	1924.0	512.0	16.0	29.0	2683.0	1227.0	12280.0	6
2	1428.0	1097.0	336.0	22.0	50.0	1036.0	99.0	11250.0	3
3	417.0	349.0	137.0	60.0	89.0	510.0	63.0	12960.0	5
4	193.0	146.0	55.0	16.0	44.0	249.0	869.0	7560.0	4

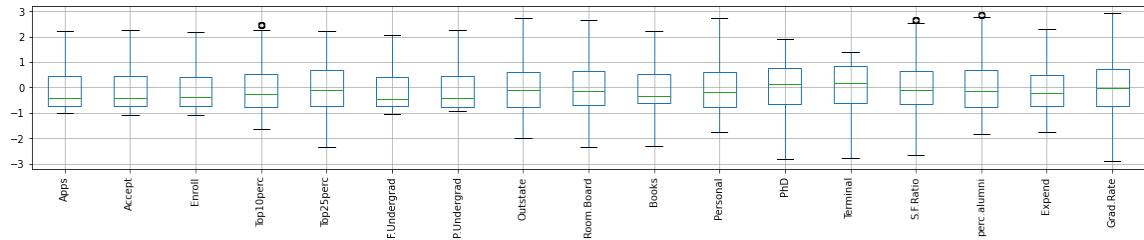
Standardising before processing PCA

```
In [370]: # All variables must be on same scale, hence we can omit scaling.
# Standardization
from scipy.stats import zscore
data_new = data_df.apply(zscore)
data_new.head()
```

```
Out[370]:
```

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate
0	-0.376493	-0.337830	0.106380	-0.246780	-0.191827	-0.018769	-0.166083	-0.746480
1	-0.159195	0.116744	-0.260441	-0.696290	-1.353911	-0.093626	0.797856	0.457762
2	-0.472336	-0.426511	-0.569343	-0.310996	-0.292878	-0.703966	-0.777974	0.201488
3	-0.889994	-0.917871	-0.918613	2.129202	1.677612	-0.898889	-0.828267	0.626954
4	-0.982532	-1.051221	-1.062533	-0.696290	-0.596031	-0.995610	0.297726	-0.716625

```
In [371]: data_new.boxplot(figsize=(20,3))
plt.xticks(rotation=90)
plt.show()
```



Create a covariance matrix for identifying Principal components

```
In [ ]:
```

```
In [372]: # PCA  
# Step 1 - Create covariance matrix  
  
cov_matrix = np.cov(data_new.T)  
print('Covariance Matrix \n%s', cov_matrix)
```

Covariance Matrix

```
%s [[ 1.00128866e+00  9.56537704e-01  8.98039052e-01  3.21756324e-01
    3.64960691e-01  8.62111140e-01  5.20492952e-01  6.54209711e-02
    1.87717056e-01  2.36441941e-01  2.30243993e-01  4.64521757e-01
    4.35037784e-01  1.26573895e-01  -1.01288006e-01  2.43248206e-01
    1.50997775e-01]
[ 9.56537704e-01  1.00128866e+00  9.36482483e-01  2.23586208e-01
  2.74033187e-01  8.98189799e-01  5.73428908e-01  -5.00874847e-03
  1.19740419e-01  2.08974091e-01  2.56676290e-01  4.27891234e-01
  4.03929238e-01  1.88748711e-01  -1.65728801e-01  1.62016688e-01
  7.90839722e-02]
[ 8.98039052e-01  9.36482483e-01  1.00128866e+00  1.71977357e-01
  2.30730728e-01  9.68548601e-01  6.42421828e-01  -1.55856056e-01
  -2.38762560e-02  2.02317274e-01  3.39785395e-01  3.82031198e-01
  3.54835877e-01  2.74622251e-01  -2.23009677e-01  5.42906862e-02
  -2.32810071e-02]
[ 3.21756324e-01  2.23586208e-01  1.71977357e-01  1.00128866e+00
  9.15052977e-01  1.11358019e-01  -1.80240778e-01  5.62884044e-01
  3.57826139e-01  1.53650150e-01  -1.16880152e-01  5.44748764e-01
  5.07401238e-01  -3.88425719e-01  4.56384036e-01  6.57885921e-01
  4.94306540e-01]
[ 3.64960691e-01  2.74033187e-01  2.30730728e-01  9.15052977e-01
  1.00128866e+00  1.81429267e-01  -9.94231153e-02  4.90200034e-01
  3.31413314e-01  1.69979808e-01  -8.69219644e-02  5.52172085e-01
  5.28333659e-01  -2.97616423e-01  4.17369123e-01  5.73643193e-01
  4.79601950e-01]
[ 8.62111140e-01  8.98189799e-01  9.68548601e-01  1.11358019e-01
  1.81429267e-01  1.00128866e+00  6.97027420e-01  -2.26457040e-01
  -5.45459528e-02  2.08147257e-01  3.60246460e-01  3.62030390e-01
  3.35485771e-01  3.24921933e-01  -2.85825062e-01  3.71119607e-04
  -8.23447851e-02]
[ 5.20492952e-01  5.73428908e-01  6.42421828e-01  -1.80240778e-01
  -9.94231153e-02  6.97027420e-01  1.00128866e+00  -3.54672874e-01
  -6.77252009e-02  1.22686416e-01  3.44495974e-01  1.27827147e-01
  1.22309141e-01  3.71084841e-01  -4.19874031e-01  -2.02189396e-01
  -2.65499420e-01]
[ 6.54209711e-02  -5.00874847e-03  -1.55856056e-01  5.62884044e-01
  4.90200034e-01  -2.26457040e-01  -3.54672874e-01  1.00128866e+00
  6.56333564e-01  5.11656377e-03  -3.26028927e-01  3.91824814e-01
  4.13110264e-01  -5.74421963e-01  5.66465309e-01  7.76326650e-01
  5.73195743e-01]
[ 1.87717056e-01  1.19740419e-01  -2.38762560e-02  3.57826139e-01
  3.31413314e-01  -5.45459528e-02  -6.77252009e-02  6.56333564e-01
  1.00128866e+00  1.09064551e-01  -2.19837042e-01  3.41908577e-01
  3.79759015e-01  -3.76915472e-01  2.72743761e-01  5.81370284e-01
  4.26338910e-01]
[ 2.36441941e-01  2.08974091e-01  2.02317274e-01  1.53650150e-01
  1.69979808e-01  2.08147257e-01  1.22686416e-01  5.11656377e-03
  1.09064551e-01  1.00128866e+00  2.40172145e-01  1.36566243e-01
  1.59523091e-01  -8.54689129e-03  -4.28870629e-02  1.50176551e-01
  -8.06107505e-03]
[ 2.30243993e-01  2.56676290e-01  3.39785395e-01  -1.16880152e-01
  -8.69219644e-02  3.60246460e-01  3.44495974e-01  -3.26028927e-01
  -2.19837042e-01  2.40172145e-01  1.00128866e+00  -1.16986124e-02
  -3.20117803e-02  1.74136664e-01  -3.06146886e-01  -1.63481407e-01
  -2.91268705e-01]
[ 4.64521757e-01  4.27891234e-01  3.82031198e-01  5.44748764e-01
  5.52172085e-01  3.62030390e-01  1.27827147e-01  3.91824814e-01
  3.41908577e-01  1.36566243e-01  -1.16986124e-02  1.00128866e+00
  8.64040263e-01  -1.29556494e-01  2.49197779e-01  5.11186852e-01
  3.10418895e-01]
```

```
[ 4.35037784e-01  4.03929238e-01  3.54835877e-01  5.07401238e-01
 5.28333659e-01  3.35485771e-01  1.22309141e-01  4.13110264e-01
 3.79759015e-01  1.59523091e-01  -3.20117803e-02  8.64040263e-01
 1.00128866e+00 -1.51187934e-01   2.66375402e-01  5.24743500e-01
 2.93180212e-01]
[ 1.26573895e-01  1.88748711e-01  2.74622251e-01  -3.88425719e-01
 -2.97616423e-01  3.24921933e-01  3.71084841e-01  -5.74421963e-01
 -3.76915472e-01  -8.54689129e-03  1.74136664e-01  -1.29556494e-01
 -1.51187934e-01  1.00128866e+00  -4.12632056e-01  -6.55219504e-01
 -3.08922187e-01]
[-1.01288006e-01 -1.65728801e-01 -2.23009677e-01  4.56384036e-01
 4.17369123e-01 -2.85825062e-01 -4.19874031e-01  5.66465309e-01
 2.72743761e-01 -4.28870629e-02 -3.06146886e-01  2.49197779e-01
 2.66375402e-01 -4.12632056e-01  1.00128866e+00  4.63518674e-01
 4.92040760e-01]
[ 2.43248206e-01  1.62016688e-01  5.42906862e-02  6.57885921e-01
 5.73643193e-01  3.71119607e-04  -2.02189396e-01  7.76326650e-01
 5.81370284e-01  1.50176551e-01  -1.63481407e-01  5.11186852e-01
 5.24743500e-01  -6.55219504e-01  4.63518674e-01  1.00128866e+00
 4.15826026e-01]
[ 1.50997775e-01  7.90839722e-02  -2.32810071e-02  4.94306540e-01
 4.79601950e-01  -8.23447851e-02  -2.65499420e-01  5.73195743e-01
 4.26338910e-01  -8.06107505e-03  -2.91268705e-01  3.10418895e-01
 2.93180212e-01  -3.08922187e-01  4.92040760e-01  4.15826026e-01
 1.00128866e+00]]
```

In []: #Even if we take the transpose of the covariance matrix it results in same \#Covariance is Less than the correlation

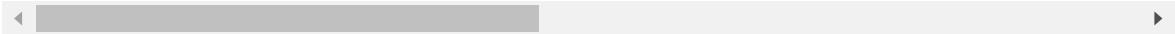
Comparing Correlation and Covariance Matrix

In [373]:

```
df_corr = data_df.copy()
df_corr.corr()
```

Out[373]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad
Apps	1.000000	0.955307	0.896883	0.321342	0.364491	0.861002	0.519823
Accept	0.955307	1.000000	0.935277	0.223298	0.273681	0.897034	0.572691
Enroll	0.896883	0.935277	1.000000	0.171756	0.230434	0.967302	0.641595
Top10perc	0.321342	0.223298	0.171756	1.000000	0.913875	0.111215	-0.180009
Top25perc	0.364491	0.273681	0.230434	0.913875	1.000000	0.181196	-0.099295
F.Undergrad	0.861002	0.897034	0.967302	0.111215	0.181196	1.000000	0.696130
P.Undergrad	0.519823	0.572691	0.641595	-0.180009	-0.099295	0.696130	1.000000
Outstate	0.065337	-0.005002	-0.155655	0.562160	0.489569	-0.226166	-0.354216
Room.Board	0.187475	0.119586	-0.023846	0.357366	0.330987	-0.054476	-0.067638
Books	0.236138	0.208705	0.202057	0.153452	0.169761	0.207879	0.122529
Personal	0.229948	0.256346	0.339348	-0.116730	-0.086810	0.359783	0.344053
PhD	0.463924	0.427341	0.381540	0.544048	0.551461	0.361564	0.127663
Terminal	0.434478	0.403409	0.354379	0.506748	0.527654	0.335054	0.122152
S.F.Ratio	0.126411	0.188506	0.274269	-0.387926	-0.297233	0.324504	0.370607
perc.alumni	-0.101158	-0.165516	-0.222723	0.455797	0.416832	-0.285457	-0.419334
Expend	0.242935	0.161808	0.054221	0.657039	0.572905	0.000371	-0.201929
Grad.Rate	0.150803	0.078982	-0.023251	0.493670	0.478985	-0.082239	-0.265158



In [374]: `data_new.corr() #with scaled data`

Out[374]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad
Apps	1.000000	0.955307	0.896883	0.321342	0.364491	0.861002	0.519823
Accept	0.955307	1.000000	0.935277	0.223298	0.273681	0.897034	0.572691
Enroll	0.896883	0.935277	1.000000	0.171756	0.230434	0.967302	0.641595
Top10perc	0.321342	0.223298	0.171756	1.000000	0.913875	0.111215	-0.180009
Top25perc	0.364491	0.273681	0.230434	0.913875	1.000000	0.181196	-0.099295
F.Undergrad	0.861002	0.897034	0.967302	0.111215	0.181196	1.000000	0.696130
P.Undergrad	0.519823	0.572691	0.641595	-0.180009	-0.099295	0.696130	1.000000
Outstate	0.065337	-0.005002	-0.155655	0.562160	0.489569	-0.226166	-0.354216
Room.Board	0.187475	0.119586	-0.023846	0.357366	0.330987	-0.054476	-0.067638
Books	0.236138	0.208705	0.202057	0.153452	0.169761	0.207879	0.122529
Personal	0.229948	0.256346	0.339348	-0.116730	-0.086810	0.359783	0.344053
PhD	0.463924	0.427341	0.381540	0.544048	0.551461	0.361564	0.127663
Terminal	0.434478	0.403409	0.354379	0.506748	0.527654	0.335054	0.122152
S.F.Ratio	0.126411	0.188506	0.274269	-0.387926	-0.297233	0.324504	0.370607
perc.alumni	-0.101158	-0.165516	-0.222723	0.455797	0.416832	-0.285457	-0.419334
Expend	0.242935	0.161808	0.054221	0.657039	0.572905	0.000371	-0.201929
Grad.Rate	0.150803	0.078982	-0.023251	0.493670	0.478985	-0.082239	-0.265158

Correlation is same for both scaled and non scaled data “Covariance” indicates the direction of the linear relationship between variables. “Correlation” on the other hand measures both the strength and direction of the linear relationship between two variables. Correlation is a function of the covariance. You can obtain the correlation coefficient of two variables by dividing the covariance of these variables by the product of the standard deviations of the same values.

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

Identify eigen values and eigen vector

In [376]: `# Step 2- Get eigen values and eigen vector
eig_vals, eig_vecs = np.linalg.eig(cov_matrix)`

In [377]: `eig_vals`

Out[377]: `array([5.6625219 , 4.89470815, 1.12636744, 1.00397659, 0.87218426,
0.7657541 , 0.58491404, 0.5445048 , 0.42352336, 0.38101777,
0.24701456, 0.02239369, 0.03789395, 0.14726392, 0.13434483,
0.09883384, 0.07469003])`

In [378]: eig_vecs

```
Out[378]: array([[ -2.62171542e-01,   3.14136258e-01,   8.10177245e-02,
   -9.87761685e-02,  -2.19898081e-01,   2.18800617e-03,
   -2.83715076e-02,  -8.99498102e-02,   1.30566998e-01,
   -1.56464458e-01,  -8.62132843e-02,   1.82169814e-01,
   -5.99137640e-01,   8.99775288e-02,   8.88697944e-02,
   5.49428396e-01,   5.41453698e-03],
 [-2.30562461e-01,   3.44623583e-01,   1.07658626e-01,
  -1.18140437e-01,  -1.89634940e-01,  -1.65212882e-02,
  -1.29584896e-02,  -1.37606312e-01,   1.42275847e-01,
  -1.49209799e-01,  -4.25899061e-02,  -3.91041719e-01,
  6.61496927e-01,   1.58861886e-01,   4.37945938e-02,
  2.91572312e-01,   1.44582845e-02],
 [-1.89276397e-01,   3.82813322e-01,   8.55296892e-02,
  -9.30717094e-03,  -1.62314818e-01,  -6.80794143e-02,
  -1.52403625e-02,  -1.44216938e-01,   5.08712481e-02,
  -6.48997860e-02,  -4.38408622e-02,   7.16684935e-01,
  2.33235272e-01,  -3.53988202e-02,  -6.19241658e-02,
  -4.17001280e-01,  -4.97908902e-02],
 [-3.38874521e-01,  -9.93191661e-02,  -7.88293849e-02,
  3.69115031e-01,  -1.57211016e-01,  -8.88656824e-02,
  -2.57455284e-01,  2.89538833e-01,  -1.22467790e-01,
  -3.58776186e-02,  1.77837341e-03,  -5.62053913e-02,
  2.21448729e-02,  -3.92277722e-02,   6.99599977e-02,
  8.79767299e-03,  -7.23645373e-01],
 [-3.34690532e-01,  -5.95055011e-02,  -5.07938247e-02,
  4.16824361e-01,  -1.44449474e-01,  -2.76268979e-02,
  -2.39038849e-01,  3.45643551e-01,  -1.93936316e-01,
  6.41786425e-03,  -1.02127328e-01,   1.96735274e-02,
  3.22646978e-02,  1.45621999e-01,  -9.70282598e-02,
  -1.07779150e-02,  6.55464648e-01],
 [-1.63293010e-01,  3.98636372e-01,   7.37077827e-02,
  -1.39504424e-02,  -1.02728468e-01,  -5.16468727e-02,
  -3.11751439e-02,  -1.08748900e-01,   1.45452749e-03,
  -1.63981359e-04,  -3.49993487e-02,  -5.42774834e-01,
  -3.67681187e-01,  -1.33555923e-01,  -8.71753137e-02,
  -5.70683843e-01,  2.53059904e-02],
 [-2.24797091e-02,  3.57550046e-01,   4.03568700e-02,
  -2.25351078e-01,  9.56790178e-02,  -2.45375721e-02,
  -1.00138971e-02,  1.23841696e-01,  -6.34774326e-01,
  5.46346279e-01,  2.52107094e-01,   2.95029745e-02,
  2.62494456e-02,  5.02487566e-02,   4.45537493e-02,
  1.46321060e-01,  -3.97146972e-02],
 [-2.83547285e-01,  -2.51863617e-01,   1.49394795e-02,
  -2.62975384e-01,  -3.72750885e-02,  -2.03860462e-02,
  9.45370782e-02,   1.12721477e-02,  -8.36648339e-03,
  -2.31799759e-01,  5.93433149e-01,   1.03393587e-03,
  -8.14247697e-02,  5.60392799e-01,   6.72405494e-02,
  -2.11561014e-01,  -1.59275617e-03],
 [-2.44186588e-01,  -1.31909124e-01,  -2.11379165e-02,
  -5.80894132e-01,  6.91080879e-02,   2.37267409e-01,
  9.45210745e-02,   3.89639465e-01,  -2.20526518e-01,
  -2.55107620e-01,  -4.75297296e-01,   9.85725168e-03,
  2.67779296e-02,  -1.07365653e-01,   1.77715010e-02,
  -1.00935084e-01,  -2.82578388e-02],
 [-9.67082754e-02,   9.39739472e-02,  -6.97121128e-01,
  3.61562884e-02,  -3.54056654e-02,   6.38604997e-01,
  -1.11193334e-01,  -2.39817267e-01,   2.10246624e-02,
  9.11624912e-02,   4.35697999e-02,   4.36086500e-03,
  1.04624246e-02,   5.16224550e-02,   3.54343707e-02,
  -2.86384228e-02,  -8.06259380e-03],
 [ 3.52299594e-02,   2.32439594e-01,  -5.30972806e-01,
```

```

1.14982973e-01, 4.75358244e-04, -3.81495854e-01,
6.39418106e-01, 2.77206569e-01, 1.73715184e-02,
-1.27647512e-01, 1.51627393e-02, -1.08725257e-02,
4.54572099e-03, 9.39409228e-03, -1.18604404e-02,
3.38197909e-02, 1.42590097e-03],
[-3.26410696e-01, 5.51390195e-02, 8.11134044e-02,
1.47260891e-01, 5.50786546e-01, 3.34444832e-03,
8.92320786e-02, -3.42628480e-02, 1.66510079e-01,
1.00975002e-01, -3.91865961e-02, 1.33146759e-02,
1.25137966e-02, -7.16590441e-02, 7.02656469e-01,
-6.38096394e-02, 8.31471932e-02],
[-3.23115980e-01, 4.30332048e-02, 5.89785929e-02,
8.90079921e-02, 5.90407136e-01, 3.54121294e-02,
9.16985445e-02, -9.03076644e-02, 1.12609034e-01,
8.60363025e-02, -8.48575651e-02, 7.38135022e-03,
-1.79275275e-02, 1.63820871e-01, -6.62488717e-01,
9.85019644e-02, -1.13374007e-01],
[ 1.63151642e-01, 2.59804556e-01, 2.74150657e-01,
2.59486122e-01, 1.42842546e-01, 4.68752604e-01,
1.52864837e-01, 2.42807562e-01, -1.53685343e-01,
-4.70527925e-01, 3.63042716e-01, 8.85797314e-03,
1.83059753e-02, -2.39902591e-01, -4.79006197e-02,
6.19970446e-02, 3.83160891e-03],
[-1.86610828e-01, -2.57092552e-01, 1.03715887e-01,
2.23982467e-01, -1.28215768e-01, 1.25669415e-02,
3.91400512e-01, -5.66073056e-01, -5.39235753e-01,
-1.47628917e-01, -1.73918533e-01, -2.40534190e-02,
-8.03169296e-05, -4.89753356e-02, 3.58875507e-02,
2.80805469e-02, -7.32598621e-03],
[-3.28955847e-01, -1.60008951e-01, -1.84205687e-01,
-2.13756140e-01, 2.24240837e-02, -2.31562325e-01,
-1.50501305e-01, -1.18823549e-01, 2.42371616e-02,
-8.04154875e-02, 3.93722676e-01, 1.05658769e-02,
5.60069250e-02, -6.90417042e-01, -1.26667522e-01,
1.28739213e-01, 1.45099786e-01],
[-2.38822447e-01, -1.67523664e-01, 2.45335837e-01,
3.61915064e-02, -3.56843227e-01, 3.13556243e-01,
4.68641965e-01, 1.80458508e-01, 3.15812873e-01,
4.88415259e-01, 8.72638706e-02, -2.51028410e-03,
1.48410810e-02, -1.59332164e-01, -6.30737002e-02,
-7.09643331e-03, -3.29024228e-03]])

```

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.

Cummulative Distribution of Eigen values

```
In [379]: tot = sum(eig_vals)
var_exp = [( i /tot ) * 100 for i in sorted(eig_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
print("Cumulative Variance Explained", cum_var_exp)
```

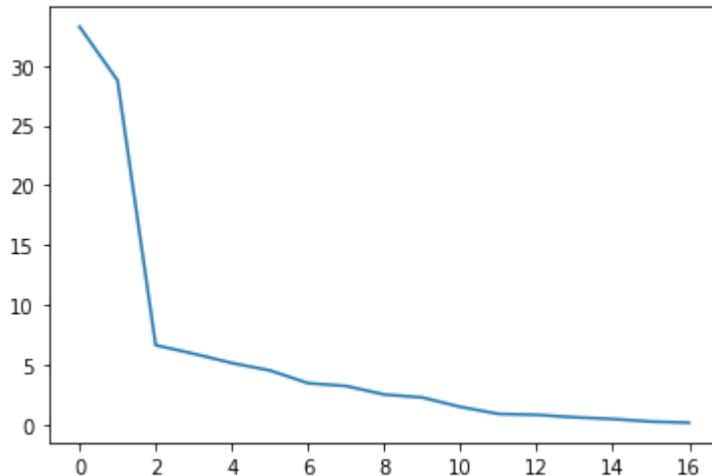
```
Cumulative Variance Explained [ 33.26608367  62.02142867  68.63859223  74.
53673619  79.66062886
84.15926753  87.59551019  90.79435736  93.28246491  95.52086136
96.97201814  97.83716159  98.62640821  99.20703552  99.64582321
99.86844192 100. ]
```

```
In [ ]:
```

Scree plot

```
In [380]: plt.plot(var_exp)
```

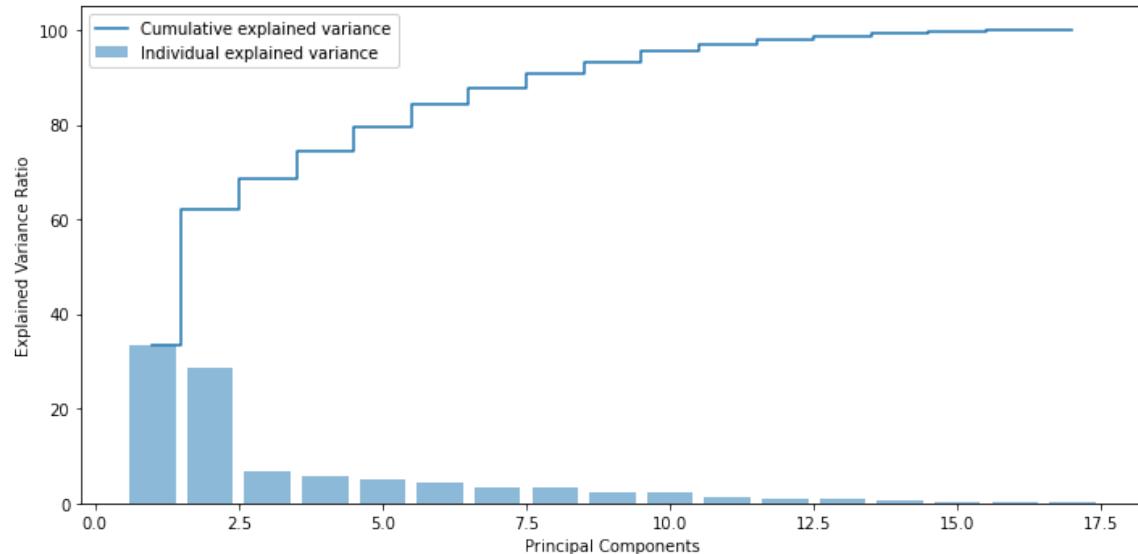
```
Out[380]: [
```



Visually we can observe that there is a steep drop in variance explained with increase in number of PC's. We will proceed with 4 components here. But depending on requirement 90% variation or 5 components will also do good

In [381]: # Plotting

```
plt.figure(figsize=(10 , 5))
plt.bar(range(1, eig_vals.size + 1), var_exp, alpha = 0.5, align = 'center')
plt.step(range(1, eig_vals.size + 1), cum_var_exp, where='mid', label = 'Cumulative explained variance')
plt.ylabel('Explained Variance Ratio')
plt.xlabel('Principal Components')
plt.legend(loc = 'best')
plt.tight_layout()
plt.show()
```



Use PCA command from sklearn and find Principal Components

In [384]:

```
# Using scikit Learn PCA here. It does all the above steps and maps data to
from sklearn.decomposition import PCA
# NOTE - we are generating only 4 PCA dimensions (dimensionality reduction)

pca = PCA(n_components = 4)
data_reduced = pca.fit_transform(data_new)
data_reduced.T
```

Out[384]: array([[-1.60249937, -1.80467547, -1.60828258, ..., -0.57688269,
 6.57095201, -0.4773931],
 [0.99368301, -0.07041499, -1.38279212, ..., 0.01779846,
 -1.18493014, 1.04394673],
 [0.03004373, 2.12213212, -0.5015119 , ..., 0.32216376,
 1.32596316, -1.42543193],
 [-1.0084203 , 3.13893281, -0.03637465, ..., -0.58726445,
 0.07770845, -1.30028602]])

In [401]: `pca.components_`

```
Out[401]: array([[ 0.26217154,  0.23056246,  0.1892764 ,  0.33887452,  0.33469053,
   0.16329301,  0.02247971,  0.28354729,  0.24418659,  0.09670828,
  -0.03522996,  0.3264107 ,  0.32311598, -0.16315164,  0.18661083,
  0.32895585,  0.23882245],
 [ 0.31413626,  0.34462358,  0.38281332, -0.09931917, -0.0595055 ,
  0.39863637,  0.35755005, -0.25186362, -0.13190912,  0.09397395,
  0.23243959,  0.05513902,  0.0430332 ,  0.25980456, -0.25709255,
  -0.16000895, -0.16752366],
 [-0.08101716, -0.10765826, -0.08553006,  0.07883042,  0.0507929 ,
  -0.07370849, -0.04035666, -0.0149397 ,  0.02113785,  0.69712111,
  0.53097284, -0.08111366, -0.05897824, -0.27415063, -0.10371585,
  0.18420555, -0.24533585],
 [ 0.09877485,  0.11813972,  0.00930802, -0.3691173 , -0.41682234,
  0.01395194,  0.22535062,  0.26297585,  0.58089427, -0.03615625,
  -0.11498306, -0.14726034, -0.08900876, -0.25948617, -0.22398255,
  0.21375646, -0.03619149]])
```

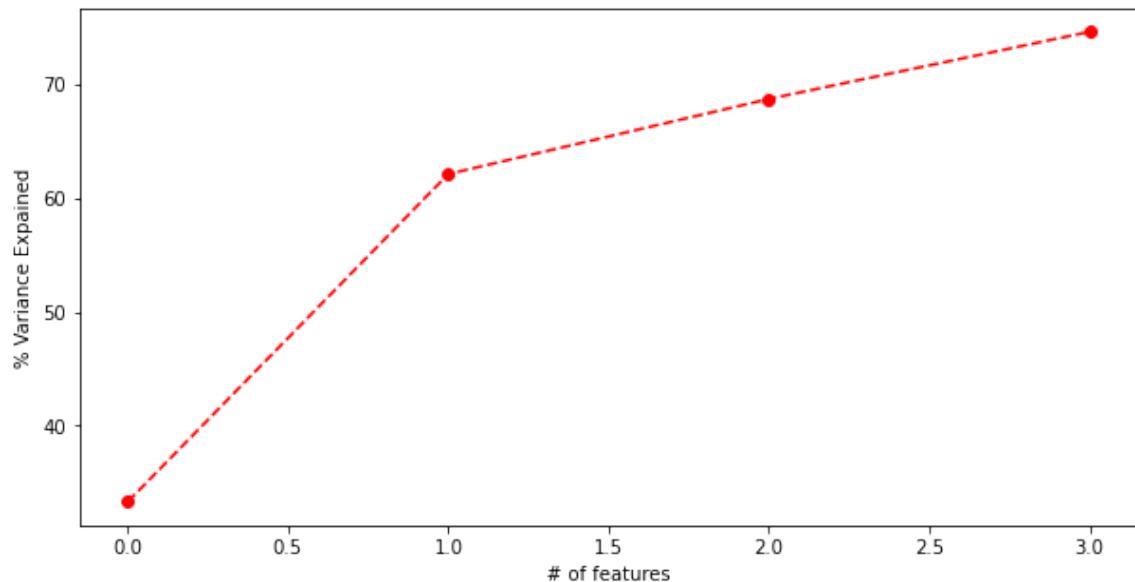
In [385]: `pca.explained_variance_ratio_`

```
Out[385]: array([0.33266084, 0.28755345, 0.06617164, 0.05898144])
```

In [386]: `var=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=3)*100)`
var #cumulative sum of variance explained with [n] features

```
Out[386]: array([33.3, 62.1, 68.7, 74.6])
```

In [389]: `plt.figure(figsize=(10 , 5))`
`plt.plot(var,marker='o',linestyle ='--',color='red')`
`plt.ylabel('% Variance Expained')`
`plt.xlabel('# of features')`
`plt.show()`



The Cumulative % gives the percentage of variance accounted for by the n components. For example, the cumulative percentage for the second component is the sum of the percentage of variance for the first and second components. It helps in deciding the number of components by selecting the components which explained the high variance

In the above array we see that the first feature explains 33.26% of the variance within our data set while the first two explain 62.02 and so on. If we employ 4 features we capture ~ 74% of the variance within the dataset, thus we gain very little by implementing an additional

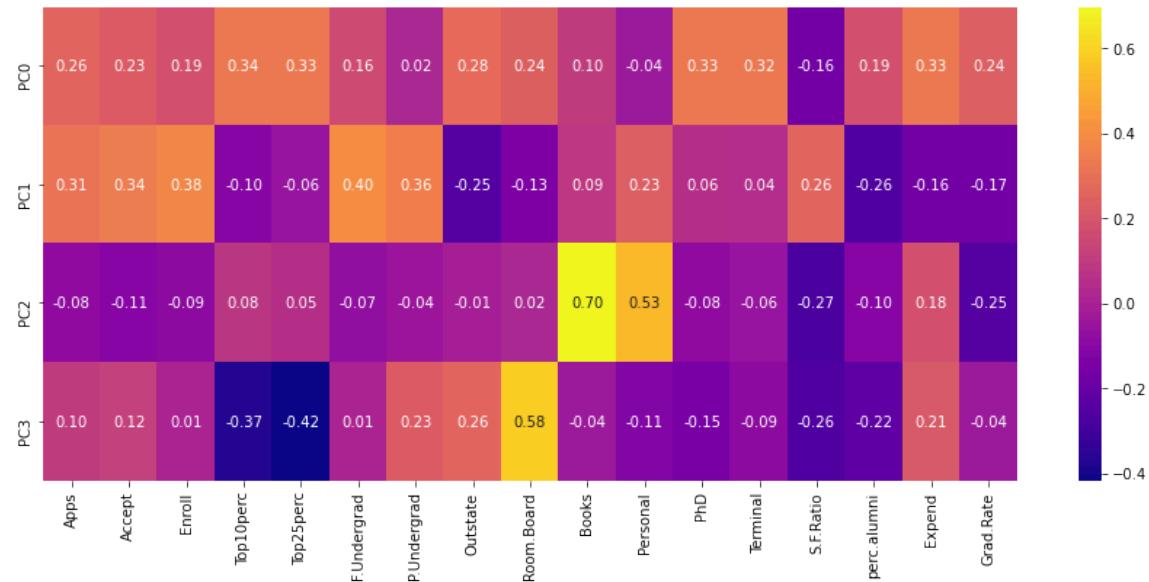
In []: `# Correlation between components and features`

In [390]: `df_comp = pd.DataFrame(pca.components_, columns=list(data_new))
df_comp.head()`

Out[390]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate
0	0.262172	0.230562	0.189276	0.338875	0.334691	0.163293	0.022480	0.283547
1	0.314136	0.344624	0.382813	-0.099319	-0.059506	0.398636	0.357550	-0.251864
2	-0.081017	-0.107658	-0.085530	0.078830	0.050793	-0.073708	-0.040357	-0.014940
3	0.098775	0.118140	0.009308	-0.369117	-0.416822	0.013952	0.225351	0.262976

In [400]: `plt.figure(figsize=(15,6))
sns.heatmap(df_comp,cmap='plasma',annot=True,fmt='.2f',yticklabels=['PC0',
plt.show()`



In [382]: `# This heatmap and the color bar basically represent the correlation between
and the principal component itself
Component 2 looks more related to aspect - We can label it as aspect prop
Depending on relationship, we could go ahead and label relationship with`

PCA is a statistical technique and uses orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables. PCA also is a tool to reduce multidimensional data to lower dimensions while retaining most of the information. Principal Component Analysis (PCA) is a well-established mathematical technique for reducing the dimensionality of data, while keeping as much variation as possible.

This PCA can only be done on continuous variables

In []: