Step 1 is to define the probem statement

Definig problem:

1) Data 1: Perform PCA on boston housing data

2) Data 2: Apply PCA on superstore sales data

Data 1: Housing Data

Step 1: Import necessary libraries and packages

These libraries will be used for both datasets

In [53]:

```
# importing necessary libraries

import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import sklearn.metrics as metrics
from sklearn.metrics import r2_score
import warnings

warnings.filterwarnings('ignore')
```

Step 2: Read the data file - here pandas library is used to read the data file

Firstly, reading the .txt file and then converting .txt to csv file.

In [54]:

```
#Reading the file
read_file = pd.read_csv (r'HousingData.txt', sep = '\s+|\t+|\s+\t+|\t+\s+', header = Non
e)
```

As the data file does not have a header. Therefore, a list of column names is added to the file and then converted to .csv file

In [55]:

```
read_file.columns = ['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX','PTRA
TIO','B_1000','LSTAT','MEDV']
#Converted to CSV
read_file.to_csv (r'HousingData.csv', index=None)
```

The housing _df defines the dataframe of HousingData

In [56]:

```
#Reading the csv file
housing_df = pd.read_csv('HousingData.csv')

#df1.head() is used to display the specified number of rows form the dataframe. Here the
```

```
number is 5
housing_df.head(5)
```

Out[56]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B_1000	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

Step 3: Summarizing the Data set

In [57]:

```
#housing_df.describe() displays the stats summary of the dataframe.
#Mean, Standard Deviation, Quantile, count, Minimum and Maximum

housing_df.describe()
```

Out[57]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TA
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.2371
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.5371
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.0000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.0000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.0000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.0000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.0000

In [58]:

```
#housing_df.info() defines the dataframe, the number of columns, rows, data type and coun
t of null values
housing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64  
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64  
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B_1000   506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

In [59]:

```python
# gives a count of the nan values present in each column
housing_df.isnull().sum()
```

Out[59]:

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B_1000     0
LSTAT      0
MEDV       0
dtype: int64
```

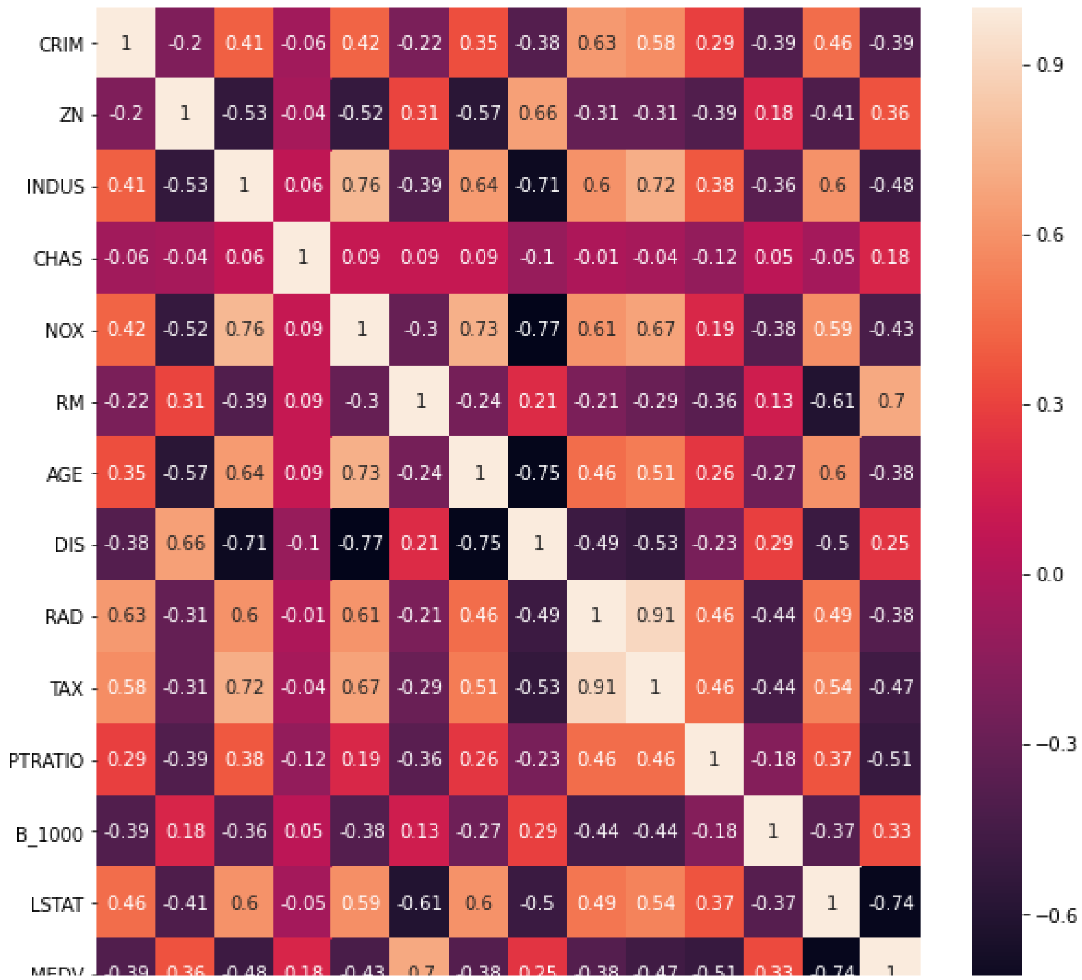
Step 4: Data Analysis

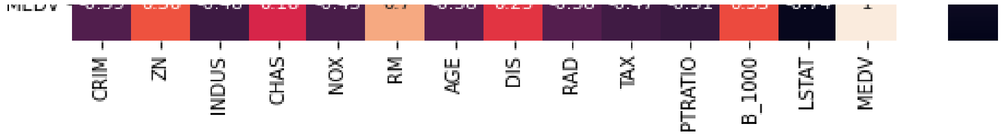
In [60]:

```python
# Correlation Matrix - states the relation between each column

correlation_matrix = housing_df.corr().round(2)

# annot = True to print the values inside the square
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(data=correlation_matrix, annot=True, ax = ax)
```

Out[60]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e8522eb3c8>
```





Correlation Matrix helps to select the features that have impact on our target varaible that is MEDV.

From the figure we can see RM has 0.7 correlation with MEDV. The other affecting variables on MEDV are the B_1000, DIS, CHAS and ZN.

While, LSTAT has the negative correlation (-0.74) with the MEDV. That means it is inversely proportional with MEDV. Also we can see form the figure, LSTAT is inveresly proportional to the RM (-0.61). Whereas, INDUS, NOX, PTRATIO, and TAX do have some linearity with the MEDV. DIS and has high effective rate on NOX, AGE and INDUS.

RAD and TAX have the highest impact on each other. There have a positive correlation of 0.91.

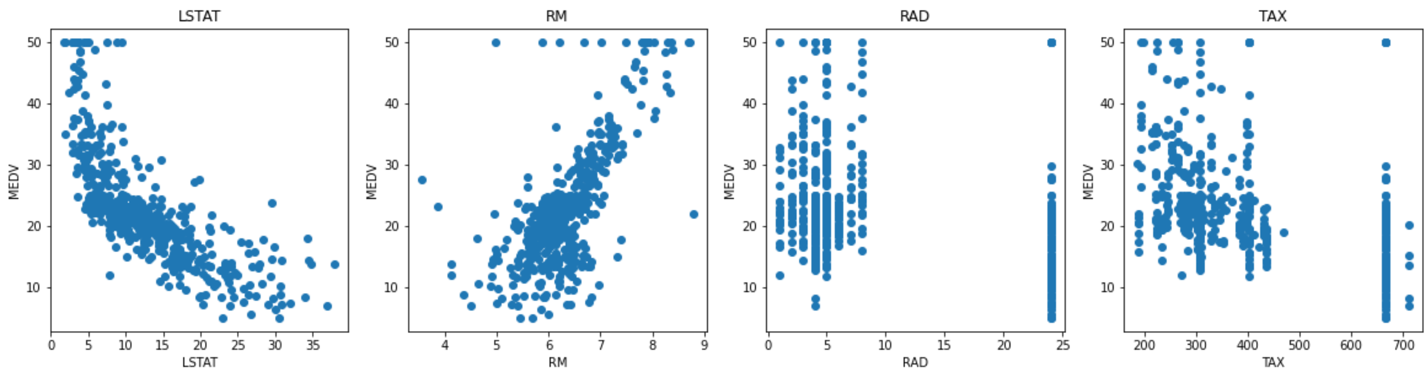
Correlation Matrix helps in feature selection that can work best for the model. From this model it can be concluded that RM and LSTAT are highly dependant variable.

Multicollinearity

Here, a lot of multicollinearity can be seen and this can make our model unstable. These independent variables can highly affect our model and the analysis results.

In [61]:

```
plt.figure(figsize=(20, 10))

features = ['LSTAT', 'RM', 'RAD' , 'TAX']
target = housing_df['MEDV']

for i, col in enumerate(features):
    plt.subplot(2, len(features) , i+1)
    x = housing_df[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
```



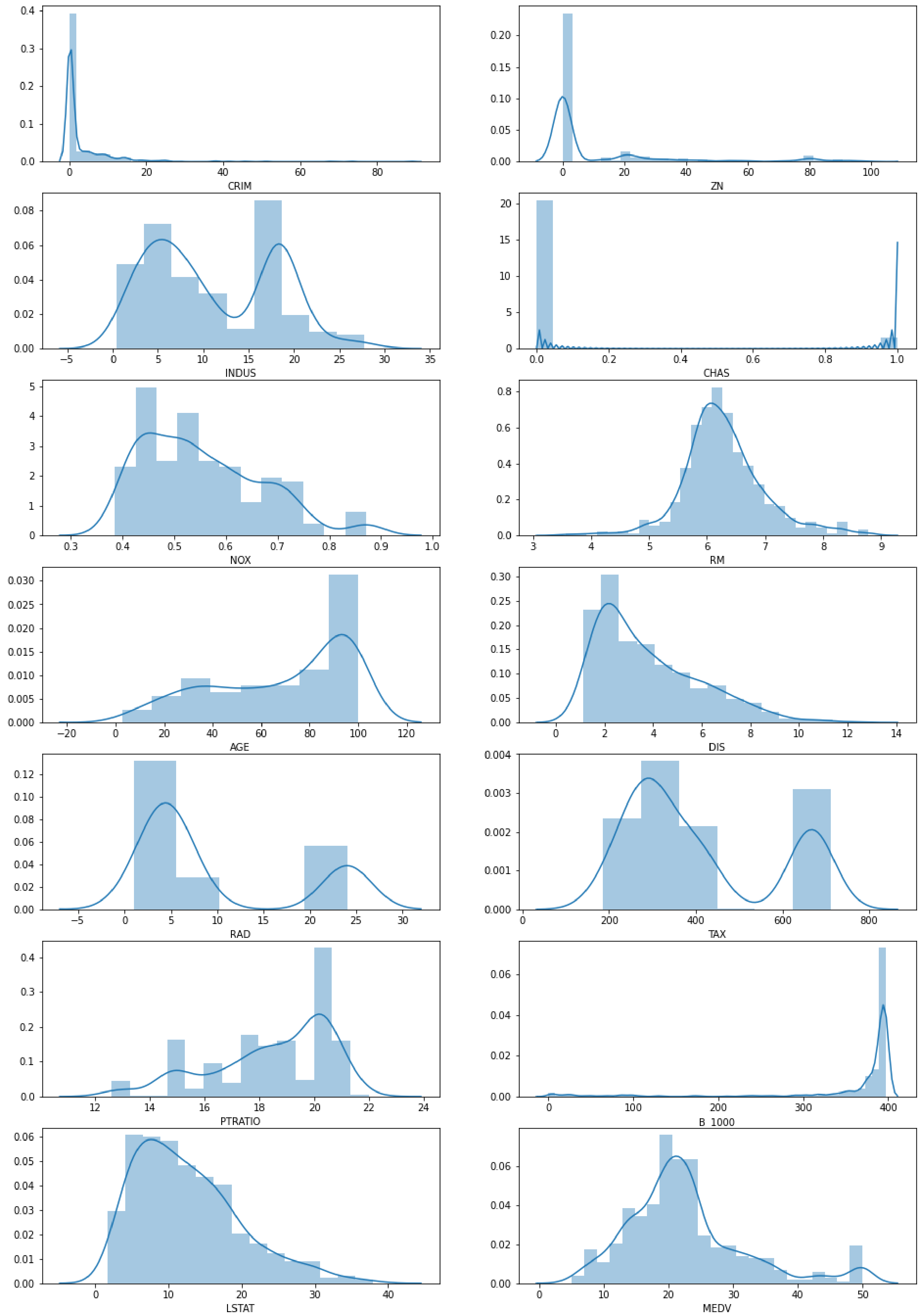
As we can see,

1) with increase in Lower state of popluation, Median value of owner houses are decreasing 2) Though we can see many outliers, most of the median rates are dependant on 5 to 7.5 Rooms 3) The average tax payer, is between range of 20 - 40 median value, paying $200-400$ tax per $10,000 property value.

In [62]:

```
# Lets look at the distribution plot of the features
pos = 1
fig = plt.figure(figsize=(16,24))
for i in housing_df.columns:
    ax = fig.add_subplot(7,2,pos)
```

```
    pos = pos + 1
    sns.distplot(housing_df[i],ax=ax)
```



PCA

PCA

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components.

PCA is used here for dimensionality reduction. For that we will first standardize the data, as our data comprises of many features and range of different scales

Calculating PCA involves following steps:

- **Calculating the covariance matrix**
- **Calculating the eigenvalues and eigenvector**
- **Forming Principal Components**
- **Projection into the new feature space**

Step 5: Standardizing of Data

In [63]:

```
# Lets build our function which will perform the normaliztion
def rescale(X):
    mean = X.mean()
    std = X.std()
    scaled_X = [(i - mean)/std for i in X]
    return pd.Series(scaled_X)
```

In [74]:

```
df_std = pd.DataFrame(columns=housing_df.columns)
for i in housing_df.columns:
    df_std[i] = rescale(housing_df[i])
```

In [75]:

```
# Lets look at the descriptive stats now
df_std.describe().iloc[1:3:]
```

Out[75]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
mean	8.326673e-17	3.466704e-16	-3.016965e-15	3.999875e-16	3.563575e-15	-1.149882e-14	-1.158274e-15	7.308603e-16
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00

In [76]:

```
# Calculating the mean and creating covariance matrix
mean_X = np.mean(df_std, axis=0)
cov_mat = (df_std - mean_X).T.dot((df_std - mean_X)) / (df_std.shape[0]-1)
print('Covariance matrix is {}'.format(cov_mat))
```

```
Covariance matrix is                 CRIM        ZN     INDUS      CHAS       NOX        RM
AGE  \
CRIM     1.000000 -0.200469  0.406583 -0.055892  0.420972 -0.219247  0.352734
ZN      -0.200469  1.000000 -0.533828 -0.042697 -0.516604  0.311991 -0.569537
INDUS    0.406583 -0.533828  1.000000  0.062938  0.763651 -0.391676  0.644779
CHAS    -0.055892 -0.042697  0.062938  1.000000  0.091203  0.091251  0.086518
NOX      0.420972 -0.516604  0.763651  0.091203  1.000000 -0.302188  0.731470
RM      -0.219247  0.311991 -0.391676  0.091251 -0.302188  1.000000 -0.240265
AGE      0.352734 -0.569537  0.644779  0.086518  0.731470 -0.240265  1.000000
DIS     -0.379670  0.664408 -0.708027 -0.099176 -0.769230  0.205246 -0.747881
RAD      0.625505 -0.311948  0.595129 -0.007368  0.611441 -0.209847  0.456022
TAX      0.582764 -0.314563  0.720760 -0.035587  0.668023 -0.292048  0.506456
PTRATIO  0.289946 -0.391679  0.383248 -0.121515  0.188933 -0.355501  0.261515
B 1000  -0.385064  0.175520 -0.356977  0.048788 -0.380051  0.128069 -0.273534
```

```
LSTAT    0.455621 -0.412995  0.603800 -0.053929  0.590879 -0.613808  0.602339
MEDV    -0.388305  0.360445 -0.483725  0.175260 -0.427321  0.695360 -0.376955

              DIS       RAD       TAX   PTRATIO    B_1000     LSTAT      MEDV
CRIM    -0.379670  0.625505  0.582764  0.289946 -0.385064  0.455621 -0.388305
ZN       0.664408 -0.311948 -0.314563 -0.391679  0.175520 -0.412995  0.360445
INDUS   -0.708027  0.595129  0.720760  0.383248 -0.356977  0.603800 -0.483725
CHAS    -0.099176 -0.007368 -0.035587 -0.121515  0.048788 -0.053929  0.175260
NOX     -0.769230  0.611441  0.668023  0.188933 -0.380051  0.590879 -0.427321
RM       0.205246 -0.209847 -0.292048 -0.355501  0.128069 -0.613808  0.695360
AGE     -0.747881  0.456022  0.506456  0.261515 -0.273534  0.602339 -0.376955
DIS      1.000000 -0.494588 -0.534432 -0.232471  0.291512 -0.496996  0.249929
RAD     -0.494588  1.000000  0.910228  0.464741 -0.444413  0.488676 -0.381626
TAX     -0.534432  0.910228  1.000000  0.460853 -0.441808  0.543993 -0.468536
PTRATIO -0.232471  0.464741  0.460853  1.000000 -0.177383  0.374044 -0.507787
B_1000   0.291512 -0.444413 -0.441808 -0.177383  1.000000 -0.366087  0.333461
LSTAT   -0.496996  0.488676  0.543993  0.374044 -0.366087  1.000000 -0.737663
MEDV     0.249929 -0.381626 -0.468536 -0.507787  0.333461 -0.737663  1.000000
```

In [77]:

```
print('covariance matrix is {}' .format(np.cov(df_std.T)))
```

```
covariance matrix is [[ 1.         -0.20046922  0.40658341 -0.05589158  0.42097171 -0.219
2467
   0.35273425 -0.37967009  0.62550515  0.58276431  0.28994558 -0.38506394
   0.45562148 -0.38830461]
 [-0.20046922  1.         -0.53382819 -0.04269672 -0.51660371  0.31199059
  -0.56953734  0.66440822 -0.31194783 -0.31456332 -0.39167855  0.17552032
  -0.41299457  0.36044534]
 [ 0.40658341 -0.53382819  1.          0.06293803  0.76365145 -0.39167585
   0.64477851 -0.70802699  0.59512927  0.72076018  0.38324756 -0.35697654
   0.60379972 -0.48372516]
 [-0.05589158 -0.04269672  0.06293803  1.          0.09120281  0.09125123
   0.08651777 -0.09917578 -0.00736824 -0.03558652 -0.12151517  0.04878848
  -0.0539293   0.17526018]
 [ 0.42097171 -0.51660371  0.76365145  0.09120281  1.         -0.30218819
   0.7314701  -0.76923011  0.61144056  0.6680232   0.18893268 -0.38005064
   0.59087892 -0.42732077]
 [-0.2192467   0.31199059 -0.39167585  0.09125123 -0.30218819  1.
  -0.24026493  0.20524621 -0.20984667 -0.29204783 -0.35550149  0.12806864
  -0.61380827  0.69535995]
 [ 0.35273425 -0.56953734  0.64477851  0.08651777  0.7314701  -0.24026493
   1.         -0.74788054  0.45602245  0.50645559  0.26151501 -0.27353398
   0.60233853 -0.37695457]
 [-0.37967009  0.66440822 -0.70802699 -0.09917578 -0.76923011  0.20524621
  -0.74788054  1.         -0.49458793 -0.53443158 -0.23247054  0.29151167
  -0.49699583  0.24992873]
 [ 0.62550515 -0.31194783  0.59512927 -0.00736824  0.61144056 -0.20984667
   0.45602245 -0.49458793  1.          0.91022819  0.46474118 -0.44441282
   0.48867633 -0.38162623]
 [ 0.58276431 -0.31456332  0.72076018 -0.03558652  0.6680232  -0.29204783
   0.50645559 -0.53443158  0.91022819  1.          0.46085304 -0.44180801
   0.54399341 -0.46853593]
 [ 0.28994558 -0.39167855  0.38324756 -0.12151517  0.18893268 -0.35550149
   0.26151501 -0.23247054  0.46474118  0.46085304  1.         -0.1773833
   0.37404432 -0.50778669]
 [-0.38506394  0.17552032 -0.35697654  0.04878848 -0.38005064  0.12806864
  -0.27353398  0.29151167 -0.44441282 -0.44180801 -0.1773833   1.
  -0.3660869   0.33346082]
 [ 0.45562148 -0.41299457  0.60379972 -0.0539293   0.59087892 -0.61380827
   0.60233853 -0.49699583  0.48867633  0.54399341  0.37404432 -0.3660869
   1.         -0.73766273]
 [-0.38830461  0.36044534 -0.48372516  0.17526018 -0.42732077  0.69535995
  -0.37695457  0.24992873 -0.38162623 -0.46853593 -0.50778669  0.33346082
  -0.73766273  1.        ]]
```

In [78]:

```
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
print('Eigenvectors: {}' .format(eig_vecs))
```

```
print('\nEigenvalues: {}' .format(eig_vals))
```

```
Eigenvectors: [[ 2.42284451e-01 -6.58731079e-02  3.95077419e-01  1.00366211e-01
   4.95765921e-03  2.24627030e-01  7.77083366e-01  1.57401402e-01
  -5.91141759e-02 -9.70323119e-02 -2.54211798e-01  7.13846149e-02
  -6.32761157e-02 -7.10687807e-02]
 [-2.45435005e-01 -1.48002653e-01  3.94545713e-01  3.42958421e-01
   1.14495002e-01  3.35746944e-01 -2.74178365e-01 -3.80314042e-01
   9.62968067e-02  1.32375830e-01 -3.82899480e-01 -2.45579673e-01
   2.21122101e-01 -1.27709065e-01]
 [ 3.31859746e-01  1.27075668e-01 -6.60819134e-02 -9.62693566e-03
  -2.25836917e-02  8.08249519e-02 -3.40273839e-01  1.71745781e-01
   2.35472877e-01 -8.37168543e-02 -6.27048264e-01  2.54827026e-01
  -3.48408284e-01  2.73797614e-01]
 [-5.02713285e-03  4.10668763e-01 -1.25305293e-01  7.00406497e-01
  -5.35197817e-01 -1.62649056e-01  7.40757751e-02 -3.29270041e-02
  -2.34889657e-02  4.99174539e-02  1.86429670e-02  4.17069157e-02
   1.90397469e-02 -9.96840221e-03]
 [ 3.25193880e-01  2.54276363e-01 -4.64755487e-02  5.37075825e-02
   1.94605570e-01  1.48991906e-01 -1.98092965e-01  4.74583814e-02
  -8.76491484e-02 -5.24974687e-01  4.30243906e-02  2.11620349e-01
   4.49093566e-01 -4.37475550e-01]
 [-2.02816554e-01  4.34005810e-01  3.53406095e-01 -2.93357309e-01
  -8.32048140e-03 -1.31080559e-01  7.40849381e-02 -4.37615662e-01
  -7.19051500e-03  4.98935961e-02  3.66694703e-03  5.26133916e-01
   1.25605540e-01  2.23951923e-01]
 [ 2.96976574e-01  2.60303205e-01 -2.00823078e-01 -7.84263261e-02
   1.49750092e-01  6.08695963e-02  1.18580363e-01 -5.88105687e-01
   3.82270273e-02  5.14625621e-02  4.32658224e-02 -2.45647942e-01
  -4.86339045e-01 -3.29630928e-01]
 [-2.98169809e-01 -3.59149977e-01  1.57068710e-01  1.84747787e-01
  -1.06219480e-01 -1.16233540e-02 -1.04397844e-01 -1.28230604e-01
  -4.71240287e-02 -5.52292172e-01  1.75802196e-01  2.99412026e-01
  -4.93568220e-01 -1.14600078e-01]
 [ 3.03412754e-01  3.11495955e-02  4.18510334e-01 -5.13743811e-02
  -2.30352185e-01  1.34937322e-01 -1.37080107e-01  7.46487153e-02
   6.34975332e-01  6.27847431e-03  4.63439397e-01 -1.15793486e-01
  -1.86364119e-02  4.22133485e-02]
 [ 3.24033052e-01  8.85140554e-03  3.43232194e-01 -2.68106947e-02
  -1.63425820e-01  1.88471462e-01 -3.13984433e-01  7.09921237e-02
  -6.98822190e-01  2.42987756e-01  1.79446555e-01  8.36641308e-03
  -1.70421793e-01  4.27940542e-02]
 [ 2.07679535e-01 -3.14623061e-01  3.99092044e-04 -3.42036328e-01
  -6.15707380e-01 -2.79017309e-01  1.48560832e-03 -2.83469595e-01
  -5.57381600e-02 -1.88347079e-01 -2.74525949e-01 -1.60474164e-01
   2.32148422e-01 -9.99918413e-02]
 [-1.96638358e-01  2.64810325e-02 -3.61375914e-01 -2.01741185e-01
  -3.67460674e-01  7.85907284e-01  7.48427805e-02 -4.44417533e-02
   1.61652795e-02  2.10781985e-02  6.09756507e-02  1.46292237e-01
   4.15288525e-02  3.91948578e-02]
 [ 3.11397955e-01 -2.01245177e-01 -1.61060336e-01  2.42621217e-01
   1.78358870e-01  9.19721068e-02  8.32130826e-02 -3.57482467e-01
  -8.31437946e-02 -2.49489863e-01  1.71810921e-01 -6.66472668e-02
   1.81892088e-01  6.83032690e-01]
 [-2.66636396e-01  4.44924411e-01  1.63188735e-01 -1.80297553e-01
  -5.06598928e-02  5.40280379e-02 -9.96497280e-03  1.52308790e-01
  -1.34127182e-01 -4.69629324e-01 -7.07510826e-02 -5.75547284e-01
  -9.82858002e-02  2.42001064e-01]]

Eigenvalues: [6.54598958 1.64953191 1.34890592 0.88653987 0.85089944 0.66001077
 0.5354108  0.40307658 0.06032666 0.1340097  0.27726358 0.25225744
 0.1829875  0.21279025]
```

In [79]:

```
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]
```

In [80]:

```
eig_pairs.sort(key=lambda x: x[0], reverse=True)
```

Explained variance(var_exp) is the amount of variance explained by each of the selected components. This attribute is associated with the sklearn PCA model as explained*variance*

Explained variance ratio is the percentage of variance explained by each of the selected components.

In [81]:

```
total = sum(eig_vals)
var_exp = [(i / total)*100 for i in sorted(eig_vals, reverse=True)]
var_exp

count = len(var_exp)
print(var_exp)
```

```
[46.75706841894138, 11.782370755269831, 9.635042314463803, 6.332427636794928, 6.077853139
077556, 4.714362667278552, 3.8243628603712976, 2.8791184448943112, 1.9804541698966356, 1.
801838852709612, 1.5199303537192448, 1.3070535668777132, 0.9572121182234968, 0.4309047014
8165766]
```

In [82]:

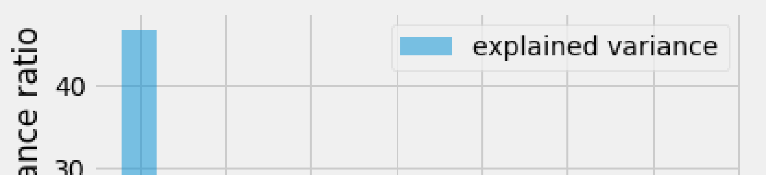
```
pd.DataFrame(var_exp, index = ['Component 1', 'Component 2', 'Component 3', 'Component 4'
,'Component 5',
                                'Component 6','Component 7','Component 8', 'Component 9',
'Component 10', 'Component 11',
                                'Component 12','Component 13','Component 14'], columns =
['Explained Variance'])
```

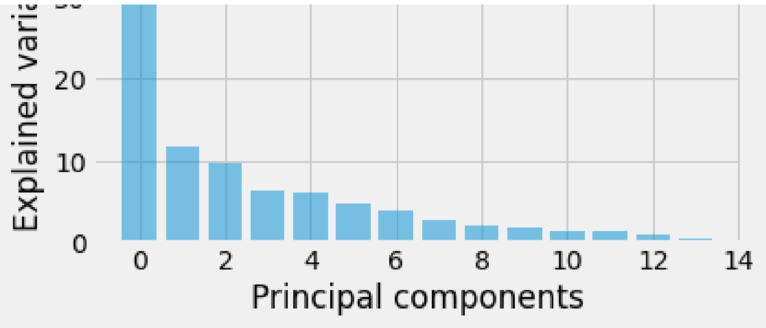
Out[82]:

	Explained Variance
Component 1	46.757068
Component 2	11.782371
Component 3	9.635042
Component 4	6.332428
Component 5	6.077853
Component 6	4.714363
Component 7	3.824363
Component 8	2.879118
Component 9	1.980454
Component 10	1.801839
Component 11	1.519930
Component 12	1.307054
Component 13	0.957212
Component 14	0.430905

In [83]:

```
with plt.style.context('fivethirtyeight'):
    plt.figure(figsize=(6, 4))
    plt.bar(range(count), var_exp, alpha=0.5, align='center', label='explained variance'
)
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.legend(loc='best')
    plt.tight_layout()
```





Applying PCA on the standardized data

In [84]:

```
pca = PCA(n_components=3)
X = df_std.drop('MEDV',axis=1)
X_pca = pca.fit_transform(X)
df_std_pca = pd.DataFrame(X_pca, columns=['PCA1','PCA2', 'PCA3'])
df_std_pca
```

Out[84]:

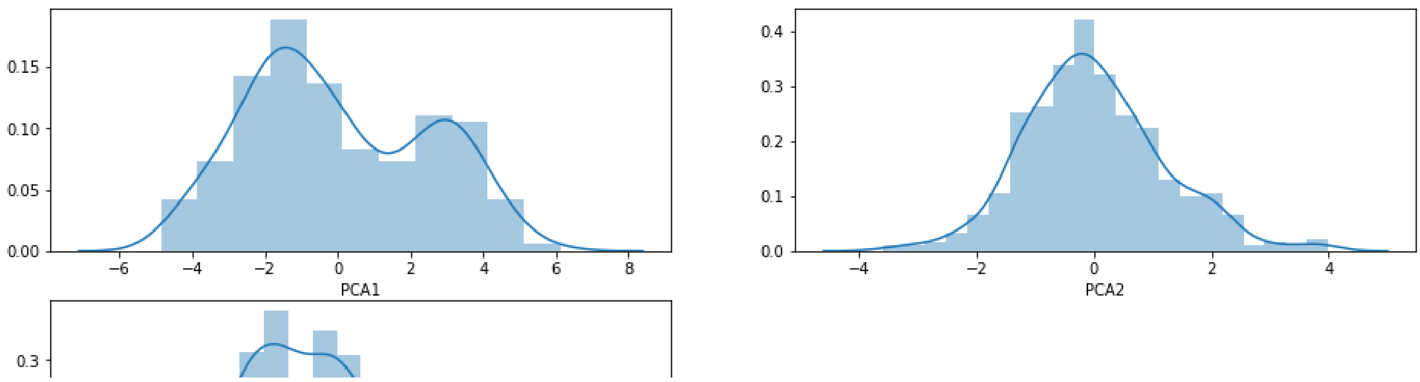
	PCA1	PCA2	PCA3
0	-2.096223	0.772348	0.342604
1	-1.455811	0.591400	-0.694512
2	-2.072547	0.599047	0.166956
3	-2.608922	-0.006864	-0.100185
4	-2.455755	0.097615	-0.075274
...	...	...	...
501	-0.314656	0.723568	-0.860045
502	-0.110404	0.758557	-1.254737
503	-0.312052	1.154104	-0.408194
504	-0.270252	1.040332	-0.584875
505	-0.125679	0.761225	-1.293602

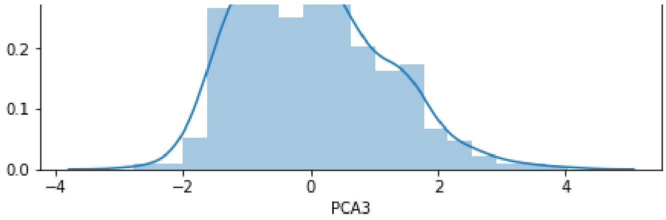
506 rows × 3 columns

Using PCA the multicollinearity is removed

In [85]:

```
# Lets look at the distribution of our features after applying PCA
pos = 1
fig = plt.figure(figsize=(16,24))
for i in df_std_pca.columns:
    ax = fig.add_subplot(7,2,pos)
    pos = pos + 1
    sns.distplot(df_std_pca[i],ax=ax)
```





In [ ]:

Data 2: Superstore Data

In [86]:

```
#Reading the csv file
store_data = pd.read_csv("superstore.csv")

#df1.head() is used to display the specified number of rows form the dataframe. Here the
number is 5
store_data.head(5)
```

Out[86]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Postal Code	Region	Product ID
0	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-BO-1000179
1	2	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-CH-1000045
2	3	CA-2016-138688	2016-06-12	2016-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	...	90036	West	OFF-LA-1000024
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	FUR-TA-1000057
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	OFF-ST-1000076

5 rows × 21 columns

In [87]:

```
# Describing the dataset using method describe()
store_data.describe()
```

Out[87]:

	Row ID	Postal Code	Sales	Quantity	Discount	Profit
count	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000
mean	4997.500000	55190.379428	229.858001	3.789574	0.156203	28.656896
std	2885.163629	32063.693350	623.245101	2.225110	0.206452	234.260108

	Row ID	Postal Code	Sales	Quantity	Discount	Profit
min	1.000000	1040.000000	0.444000	1.000000	0.000000	-6599.978000
25%	2499.250000	23223.000000	17.280000	2.000000	0.000000	1.728750
50%	4997.500000	56430.500000	54.490000	3.000000	0.200000	8.666500
75%	7495.750000	90008.000000	209.940000	5.000000	0.200000	29.364000
max	9994.000000	99301.000000	22638.480000	14.000000	0.800000	8399.976000

In [88]:

```
#extracting information of the dataset
store_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Row ID         9994 non-null   int64  
 1   Order ID       9994 non-null   object 
 2   Order Date     9994 non-null   object 
 3   Ship Date      9994 non-null   object 
 4   Ship Mode      9994 non-null   object 
 5   Customer ID    9994 non-null   object 
 6   Customer Name  9994 non-null   object 
 7   Segment        9994 non-null   object 
 8   Country        9994 non-null   object 
 9   City           9994 non-null   object 
 10  State          9994 non-null   object 
 11  Postal Code    9994 non-null   int64  
 12  Region         9994 non-null   object 
 13  Product ID     9994 non-null   object 
 14  Category       9994 non-null   object 
 15  Sub-Category   9994 non-null   object 
 16  Product Name   9994 non-null   object 
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64  
 19  Discount       9994 non-null   float64
 20  Profit         9994 non-null   float64
dtypes: float64(3), int64(3), object(15)
memory usage: 1.6+ MB
```

Step: Cleaning and Structuring the Dataset

In [89]:

```
# we can see there is only one country in the dataset
store_data['Country'].value_counts()
```

Out[89]:

```
United States    9994
Name: Country, dtype: int64
```

In [90]:

```
#Selecting only those columns that can be important for solving the Problem Statement
#Alternative - Instead of selecting you can also remove the non required columns

store_data = store_data[['State', 'Region', 'Category', 'Sales', 'Quantity', 'Discount', 'Profit']]
store_data.head(5)
```

Out[90]:

	State	Region	Category	Sales	Quantity	Discount	Profit
0	Kentucky	South	Furniture	261.9600	2	0.00	41.9136

	State	Region	Category	Sales	Quantity	Discount	Profit
1	Kentucky	South	Furniture	731.9400	3	0.00	219.5820
2	California	West	Office Supplies	14.6200	2	0.00	6.8714
3	Florida	South	Furniture	957.5775	5	0.45	-383.0310
4	Florida	South	Office Supplies	22.3680	2	0.20	2.5164

Step: Converting all the categorical columns to Numeric

In [91]:

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
```

In [92]:

```
# Converting the Categorical Columns of the dataframe to the Numeric using Label Encoder
columns = ['State', 'Category', 'Region']
lb_make = LabelEncoder()

#Building a list of dictionaries. Each dictionary has the keys as "Label" and the value a
s "Number assigned"
d = []
for i,value in enumerate(columns):
    store_data[value] = lb_make.fit_transform(store_data[value])
    L = list(lb_make.inverse_transform(store_data[value]))
    d.append(dict(zip(lb_make.classes_, lb_make.transform(lb_make.classes_))))

d_state = d[0]
d_region = d[2]

#printing d_state with keys and values
print(d_state)
```

```
{'Alabama': 0, 'Arizona': 1, 'Arkansas': 2, 'California': 3, 'Colorado': 4, 'Connecticut'
: 5, 'Delaware': 6, 'District of Columbia': 7, 'Florida': 8, 'Georgia': 9, 'Idaho': 10, '
Illinois': 11, 'Indiana': 12, 'Iowa': 13, 'Kansas': 14, 'Kentucky': 15, 'Louisiana': 16,
'Maine': 17, 'Maryland': 18, 'Massachusetts': 19, 'Michigan': 20, 'Minnesota': 21, 'Missi
ssippi': 22, 'Missouri': 23, 'Montana': 24, 'Nebraska': 25, 'Nevada': 26, 'New Hampshire'
: 27, 'New Jersey': 28, 'New Mexico': 29, 'New York': 30, 'North Carolina': 31, 'North Da
kota': 32, 'Ohio': 33, 'Oklahoma': 34, 'Oregon': 35, 'Pennsylvania': 36, 'Rhode Island':
37, 'South Carolina': 38, 'South Dakota': 39, 'Tennessee': 40, 'Texas': 41, 'Utah': 42, '
Vermont': 43, 'Virginia': 44, 'Washington': 45, 'West Virginia': 46, 'Wisconsin': 47, 'Wy
oming': 48}
```

Standarizing Dataset

In [93]:

```
store_data.head(10)
```

Out[93]:

	State	Region	Category	Sales	Quantity	Discount	Profit
0	15	2	0	261.9600	2	0.00	41.9136
1	15	2	0	731.9400	3	0.00	219.5820
2	3	3	1	14.6200	2	0.00	6.8714
3	8	2	0	957.5775	5	0.45	-383.0310
4	8	2	1	22.3680	2	0.20	2.5164
5	3	3	0	48.8600	7	0.00	14.1694
6	3	3	1	7.2800	4	0.00	1.9656
7	3	3	2	907.1520	6	0.20	90.7152

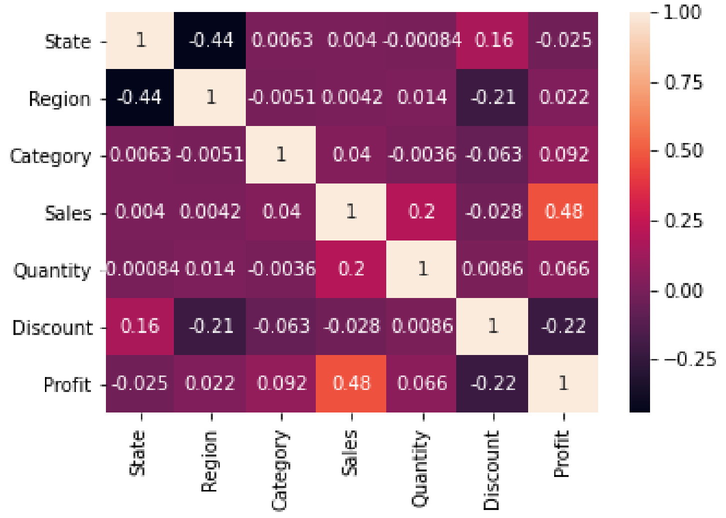
	State	Region	Category	Sales	Quantity	Discount	Profit
8	3	3	1	18.5040	3	0.20	5.7825
9	3	3	1	114.9000	5	0.00	34.4700

In [94]:

```
corrmat = store_data.corr()
sns.heatmap(corrmat,annot=True)
```

Out[94]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e852ff6dc8>
```



In [95]:

```
store_df_std = pd.DataFrame(columns=store_data.columns)
for i in store_data.columns:
    store_df_std[i] = rescale(store_data[i])
```

In [96]:

```
# Calculating the mean and creating covariance matrix
mean_X = np.mean(store_df_std, axis=0)
cov_mat = (store_df_std - mean_X).T.dot((store_df_std - mean_X)) / (store_df_std.shape[0
]-1)
print('Covariance matrix is {}'.format(cov_mat))
```

```
Covariance matrix is               State     Region  Category     Sales  Quantity  Discoun
t    Profit
State      1.000000 -0.444400  0.006285  0.004015 -0.000836  0.162552 -0.025315
Region    -0.444400  1.000000 -0.005148  0.004216  0.013506 -0.212769  0.022216
Category   0.006285 -0.005148  1.000000  0.040077 -0.003619 -0.062897  0.091506
Sales      0.004015  0.004216  0.040077  1.000000  0.200795 -0.028190  0.479064
Quantity  -0.000836  0.013506 -0.003619  0.200795  1.000000  0.008623  0.066253
Discount   0.162552 -0.212769 -0.062897 -0.028190  0.008623  1.000000 -0.219487
Profit    -0.025315  0.022216  0.091506  0.479064  0.066253 -0.219487  1.000000
```

In [97]:

```
print('covariance matrix is {}' .format(np.cov(df_std.T)))
```

```
covariance matrix is [[ 1.         -0.20046922  0.40658341 -0.05589158  0.42097171 -0.219
2467
   0.35273425 -0.37967009  0.62550515  0.58276431  0.28994558 -0.38506394
   0.45562148 -0.38830461]
 [-0.20046922  1.         -0.53382819 -0.04269672 -0.51660371  0.31199059
  -0.56953734  0.66440822 -0.31194783 -0.31456332 -0.39167855  0.17552032
  -0.41299457  0.36044534]
 [ 0.40658341 -0.53382819  1.          0.06293803  0.76365145 -0.39167585
   0.64477851 -0.70802699  0.59512927  0.72076018  0.38324756 -0.35697654
   0.60379972 -0.48372516]
 [-0.05589158 -0.04269672  0.06293803  1.          0.09120281  0.09125123
   0.08651777 -0.09917578 -0.00736824 -0.03558652 -0.12151517  0.04878848
  -0.0539293   0.17526018]
```

```
 [ 0.42097171 -0.51660371  0.76365145  0.09120281  1.         -0.30218819
   0.7314701  -0.76923011  0.61144056  0.6680232   0.18893268 -0.38005064
   0.59087892 -0.42732077]
 [-0.2192467   0.31199059 -0.39167585  0.09125123 -0.30218819  1.
  -0.24026493  0.20524621 -0.20984667 -0.29204783 -0.35550149  0.12806864
  -0.61380827  0.69535995]
 [ 0.35273425 -0.56953734  0.64477851  0.08651777  0.7314701  -0.24026493
   1.         -0.74788054  0.45602245  0.50645559  0.26151501 -0.27353398
   0.60233853 -0.37695457]
 [-0.37967009  0.66440822 -0.70802699 -0.09917578 -0.76923011  0.20524621
  -0.74788054  1.         -0.49458793 -0.53443158 -0.23247054  0.29151167
  -0.49699583  0.24992873]
 [ 0.62550515 -0.31194783  0.59512927 -0.00736824  0.61144056 -0.20984667
   0.45602245 -0.49458793  1.          0.91022819  0.46474118 -0.44441282
   0.48867633 -0.38162623]
 [ 0.58276431 -0.31456332  0.72076018 -0.03558652  0.6680232  -0.29204783
   0.50645559 -0.53443158  0.91022819  1.          0.46085304 -0.44180801
   0.54399341 -0.46853593]
 [ 0.28994558 -0.39167855  0.38324756 -0.12151517  0.18893268 -0.35550149
   0.26151501 -0.23247054  0.46474118  0.46085304  1.         -0.1773833
   0.37404432 -0.50778669]
 [-0.38506394  0.17552032 -0.35697654  0.04878848 -0.38005064  0.12806864
  -0.27353398  0.29151167 -0.44441282 -0.44180801 -0.1773833   1.
  -0.3660869   0.33346082]
 [ 0.45562148 -0.41299457  0.60379972 -0.0539293   0.59087892 -0.61380827
   0.60233853 -0.49699583  0.48867633  0.54399341  0.37404432 -0.3660869
   1.         -0.73766273]
 [-0.38830461  0.36044534 -0.48372516  0.17526018 -0.42732077  0.69535995
  -0.37695457  0.24992873 -0.38162623 -0.46853593 -0.50778669  0.33346082
  -0.73766273  1.        ]]
```

In [98]:

```python
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
print('Eigenvectors: {}' .format(eig_vecs))
print('\nEigenvalues: {}' .format(eig_vals))
```

```
Eigenvectors: [[ 0.38314092 -0.50625797 -0.0719046   0.68084466 -0.30243228 -0.16407226
   0.09900096]
 [-0.4058902   0.50948809 -0.14056074  0.70901219  0.153255    0.1464135
  -0.09107915]
 [-0.12745615 -0.10035843  0.03377778  0.01483944  0.14555209 -0.64883353
  -0.7281151 ]
 [-0.44624663 -0.4809089   0.64789888  0.12793803  0.31168202  0.16822809
   0.08946003]
 [-0.18593862 -0.23818263 -0.15146253 -0.06343459 -0.45282699  0.59723946
  -0.56567107]
 [ 0.41593474 -0.17696154 -0.27123251  0.05100182  0.74128882  0.34769689
  -0.22161319]
 [-0.51526709 -0.39336142 -0.67649873 -0.10265855  0.11025031 -0.15738374
   0.27322615]]

Eigenvalues: [1.67493514 1.48905252 0.46537829 0.55417761 0.82757951 1.05159229
 0.93728464]
```

In [99]:

```python
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]
eig_pairs.sort(key=lambda x: x[0], reverse=True)

total = sum(eig_vals)
var_exp = [(i / total)*100 for i in sorted(eig_vals, reverse=True)]
var_exp

count = len(var_exp)
```

In [100]:

```python
pd.DataFrame(var_exp, index = ['Component 1', 'Component 2', 'Component 3', 'Component 4'
,'Component 5',
                               'Component 6','Component 7'], columns = ['Explained Varia
```

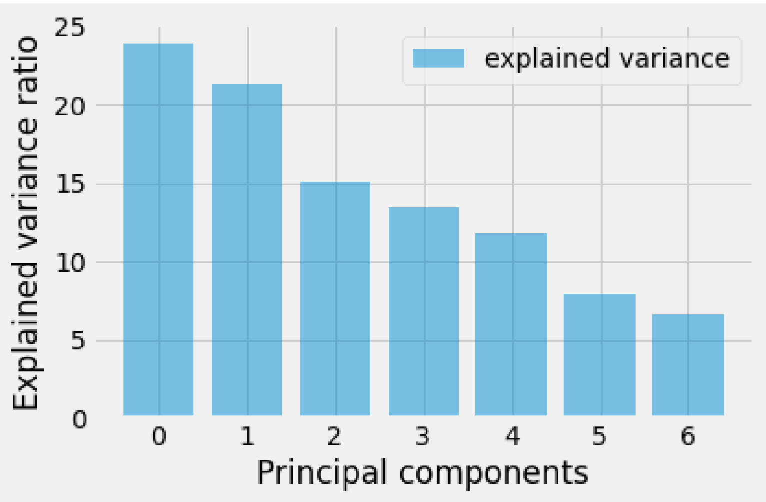
```
nce'])
```

Out[100]:

	Explained Variance
Component 1	23.927645
Component 2	21.272179
Component 3	15.022747
Component 4	13.389781
Component 5	11.822564
Component 6	7.916823
Component 7	6.648261

In [101]:

```
with plt.style.context('fivethirtyeight'):
    plt.figure(figsize=(6, 4))
    plt.bar(range(count), var_exp, alpha=0.5, align='center', label='explained variance'
)
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.legend(loc='best')
    plt.tight_layout()
```



In [102]:

```
pca = PCA(n_components=5)
X_pca = pca.fit_transform(store_df_std)
df_std_pca = pd.DataFrame(X_pca,columns=['PCA1','PCA2', 'PCA3', 'PCA4', 'PCA5'])
df_std_pca
```

Out[102]:

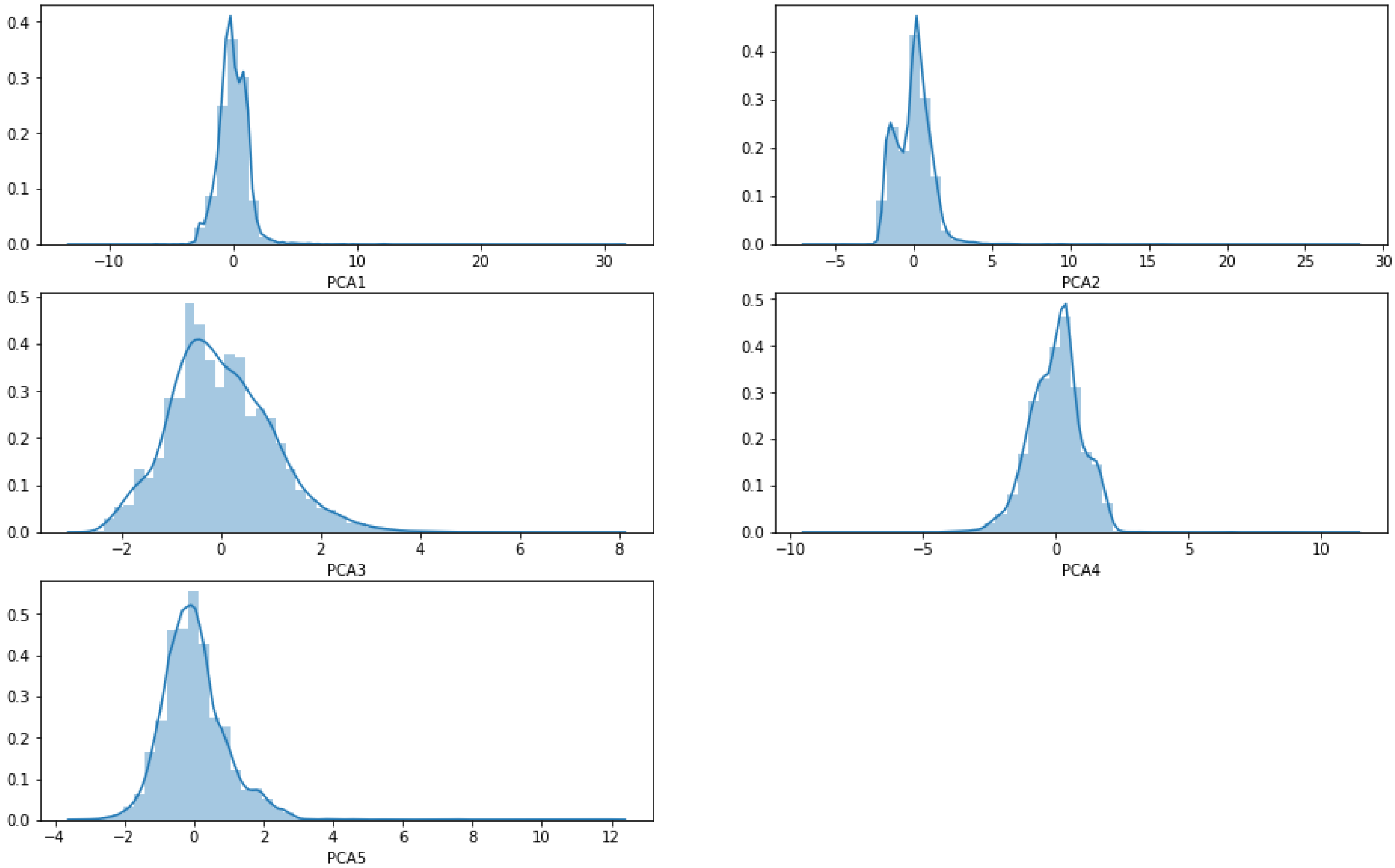
	PCA1	PCA2	PCA3	PCA4	PCA5
0	0.351786	-0.861438	0.390526	1.687025	-0.199394
1	1.162649	-0.093414	0.666430	1.707485	-0.084250
2	0.945812	-1.781877	-0.430382	0.299094	0.257546
3	-0.567770	-0.559952	2.500953	0.000836	1.090282
4	0.066083	-1.010185	-0.267411	0.190725	0.748112
...	...	...	...	...	...
9989	2.645166	-0.059775	0.573057	-0.005023	0.411847
9990	0.818002	-1.866901	0.615247	1.476989	0.069145
9991	0.947549	-1.241763	-1.066754	-1.022546	1.334765
9992	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

	PCA1	PCA2	PCA3	PCA4	PCA5
9992	1.137849	-1.545403	0.106147	-0.199678	-0.138943
9993	1.254786	-1.494577	-0.413086	0.408966	0.402936

9994 rows × 5 columns

In [103]:

```
# Lets look at the distribution of our features after applying PCA
pos = 1
fig = plt.figure(figsize=(16,24))
for i in df_std_pca.columns:
    ax = fig.add_subplot(7,2,pos)
    pos = pos + 1
    sns.distplot(df_std_pca[i],ax=ax)
```



In [ ]:

```

```