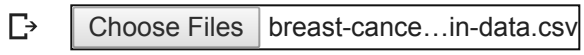
Principal Component Analysis with Cancer Data

```
#Import all the necessary modules
#Import all the necessary modules
import pandas as pd
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Q1. Load the Data file into Python DataFrame and view to

```
import io
```

```
from google.colab import files
uploaded  = files.upload()
```



- **breast-cancer-wisconsin-data.csv**(application/vnd.ms-excel) - 20732 bytes, last modified: 10/13/2019 - 1

```
Saving breast-cancer-wisconsin-data.csv to breast-cancer-wisconsin-data (1).csv
```

```
data_raw = pd.read_csv(io.BytesIO(uploaded['breast-cancer-wisconsin-data.csv']))
```

```
data_train = data_raw.copy(deep=True)
```

```
data_train.shape
```

```
(699, 11)
```

```
data_train.head(10)
```

	ID	ClumpThickness	Cell Size	Cell Shape	Marginal Adhesion	Single Epithelial
0	1000025	5	1	1	1	
1	1002945	5	4	4	5	
2	1015425	3	1	1	1	
3	1016277	6	8	8	1	
4	1017023	4	1	1	3	
5	1017122	8	10	10	8	
6	1018099	1	1	1	1	
7	1018561	2	1	2	1	
8	1033078	2	1	1	1	
9	1033078	4	2	1	1	

```
## ID column is of no use so will drop it

data_train = data_train.drop(columns='ID')
```

```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
ClumpThickness               699 non-null int64
Cell Size                    699 non-null int64
Cell Shape                   699 non-null int64
Marginal Adhesion            699 non-null int64
Single Epithelial Cell Size  699 non-null int64
Bare Nuclei                  699 non-null object
Normal Nucleoli              699 non-null int64
Bland Chromatin              699 non-null int64
Mitoses                      699 non-null int64
Class                        699 non-null int64
dtypes: int64(9), object(1)
memory usage: 54.7+ KB
```

```
data_train.isnull().any()
```

```
ClumpThickness                False
Cell Size                     False
Cell Shape                    False
Marginal Adhesion             False
Single Epithelial Cell Size   False
Bare Nuclei                   False
Normal Nucleoli               False
Bland Chromatin               False
Mitoses                       False
Class                         False
dtype: bool
```

```
# Id columns is to identify rows hence can be skipped in analysis
# All columns have numerical values
# Class would be the target variable. Should be removed when PCA is done
```

Q2 Print the datatypes of each column and the shape of t descriptive analysis

```
data_train.shape
```

```
(699, 10)
```

```
data_train.head(10)
```

	ClumpThickness	Cell Size	Cell Shape	Marginal Adhesion	Single Epithelial Cell Size
0	5	1	1	1	2
1	5	4	4	5	7
2	3	1	1	1	2
3	6	8	8	1	3
4	4	1	1	3	2
5	8	10	10	8	7
6	1	1	1	1	2
7	2	1	2	1	2
8	2	1	1	1	2
9	4	2	1	1	2

```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
ClumpThickness               699 non-null int64
Cell Size                    699 non-null int64
Cell Shape                   699 non-null int64
Marginal Adhesion            699 non-null int64
Single Epithelial Cell Size  699 non-null int64
Bare Nuclei                  699 non-null object
Normal Nucleoli              699 non-null int64
Bland Chromatin              699 non-null int64
Mitoses                      699 non-null int64
Class                        699 non-null int64
dtypes: int64(9), object(1)
memory usage: 54.7+ KB
```

```
## Bare Nuclei is showing as only column as Object so this have non numeric values other columns hav
```

```
data_train.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
ClumpThickness	699.0	4.417740	2.815741	1.0	2.0	4.0	6.0	10.0
Cell Size	699.0	3.134478	3.051459	1.0	1.0	1.0	5.0	10.0
Cell Shape	699.0	3.207439	2.971913	1.0	1.0	1.0	5.0	10.0
Marginal Adhesion	699.0	2.806867	2.855379	1.0	1.0	1.0	4.0	10.0
Single Epithelial Cell Size	699.0	3.216023	2.214300	1.0	2.0	2.0	4.0	10.0
Normal Nucleoli	699.0	3.437768	2.438364	1.0	2.0	3.0	5.0	10.0
Bland Chromatin	699.0	2.866953	3.053634	1.0	1.0	1.0	4.0	10.0
Mitoses	699.0	1.589413	1.715078	1.0	1.0	1.0	1.0	10.0
Class	699.0	2.689557	0.951273	2.0	2.0	2.0	4.0	4.0

Q3 Check for missing value check, incorrect data, duplica imputation with mean, median, mode as necessary.

```
## checking for the missing data in each column

data_train.isnull().sum()
```

```
ClumpThickness               0
Cell Size                    0
Cell Shape                   0
Marginal Adhesion            0
Single Epithelial Cell Size  0
Bare Nuclei                  0
Normal Nucleoli              0
Bland Chromatin              0
Mitoses                      0
Class                        0
dtype: int64
```

```
## checking for the duplicate data in each column
```

```
data_train.nunique()
```

```
ClumpThickness               10
Cell Size                    10
Cell Shape                   10
Marginal Adhesion            10
Single Epithelial Cell Size  10
Bare Nuclei                  11
Normal Nucleoli              10
Bland Chromatin              10
Mitoses                       9
Class                         2
dtype: int64
```

```
data_train.columns
```

```
Index(['ClumpThickness', 'Cell Size', 'Cell Shape', 'Marginal Adhesion',
       'Single Epithelial Cell Size', 'Bare Nuclei', 'Normal Nucleoli',
       'Bland Chromatin', 'Mitoses', 'Class'],
      dtype='object')
```

```
data_train = data_train.drop_duplicates(keep='first')
```

```
data_train.shape
```

```
(463, 10)
```

```
data_raw.shape
```

```
(699, 11)
```

```
## checking for the incorrect data in each column
```

```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 463 entries, 0 to 698
Data columns (total 10 columns):
ClumpThickness               463 non-null int64
Cell Size                    463 non-null int64
Cell Shape                   463 non-null int64
Marginal Adhesion            463 non-null int64
Single Epithelial Cell Size  463 non-null int64
Bare Nuclei                  463 non-null object
Normal Nucleoli              463 non-null int64
Bland Chromatin              463 non-null int64
Mitoses                      463 non-null int64
Class                        463 non-null int64
dtypes: int64(9), object(1)
memory usage: 39.8+ KB
```

```
data_train['Bare Nuclei'].unique()
```

```
array(['1', '10', '2', '4', '3', '9', '7', '?', '5', '8', '6'],
      dtype=object)
```

```
data_train['Bare Nuclei'].mode()
```

```
0    1
dtype: object
```

```
## Finding the non numeric values and replacing it with mode as it is
```

```
data_train[~data_train['Bare Nuclei'].str.isdigit()]['Bare Nuclei']
```

```
23     ?
40     ?
139    ?
145    ?
158    ?
164    ?
235    ?
249    ?
275    ?
292    ?
294    ?
297    ?
315    ?
617    ?
Name: Bare Nuclei, dtype: object
```

```
data_train['Bare Nuclei'] = data_train['Bare Nuclei'].replace(data_train[~data_train['Bare Nuclei'].
```

```
data_train['Bare Nuclei'].isnull().unique()
```

```
array([False,  True])
```

```
data_train['Bare Nuclei'].fillna(method='ffill',inplace=True)
```

```
## Checking the non numeric digits
data_train[~data_train['Bare Nuclei'].str.isdigit()]['Bare Nuclei']
```

```
Series([], Name: Bare Nuclei, dtype: object)
```

```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 463 entries, 0 to 698
Data columns (total 10 columns):
ClumpThickness               463 non-null int64
Cell Size                    463 non-null int64
Cell Shape                   463 non-null int64
Marginal Adhesion            463 non-null int64
Single Epithelial Cell Size  463 non-null int64
Bare Nuclei                  463 non-null object
Normal Nucleoli              463 non-null int64
Bland Chromatin              463 non-null int64
Mitoses                      463 non-null int64
Class                        463 non-null int64
dtypes: int64(9), object(1)
memory usage: 59.8+ KB
```

```
data_train['Bare Nuclei'] = data_train['Bare Nuclei'].astype(np.int64)
```

```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 463 entries, 0 to 698
Data columns (total 10 columns):
ClumpThickness               463 non-null int64
Cell Size                    463 non-null int64
Cell Shape                   463 non-null int64
Marginal Adhesion            463 non-null int64
Single Epithelial Cell Size  463 non-null int64
Bare Nuclei                  463 non-null int64
Normal Nucleoli              463 non-null int64
Bland Chromatin              463 non-null int64
Mitoses                      463 non-null int64
Class                        463 non-null int64
dtypes: int64(10)
memory usage: 59.8 KB
```

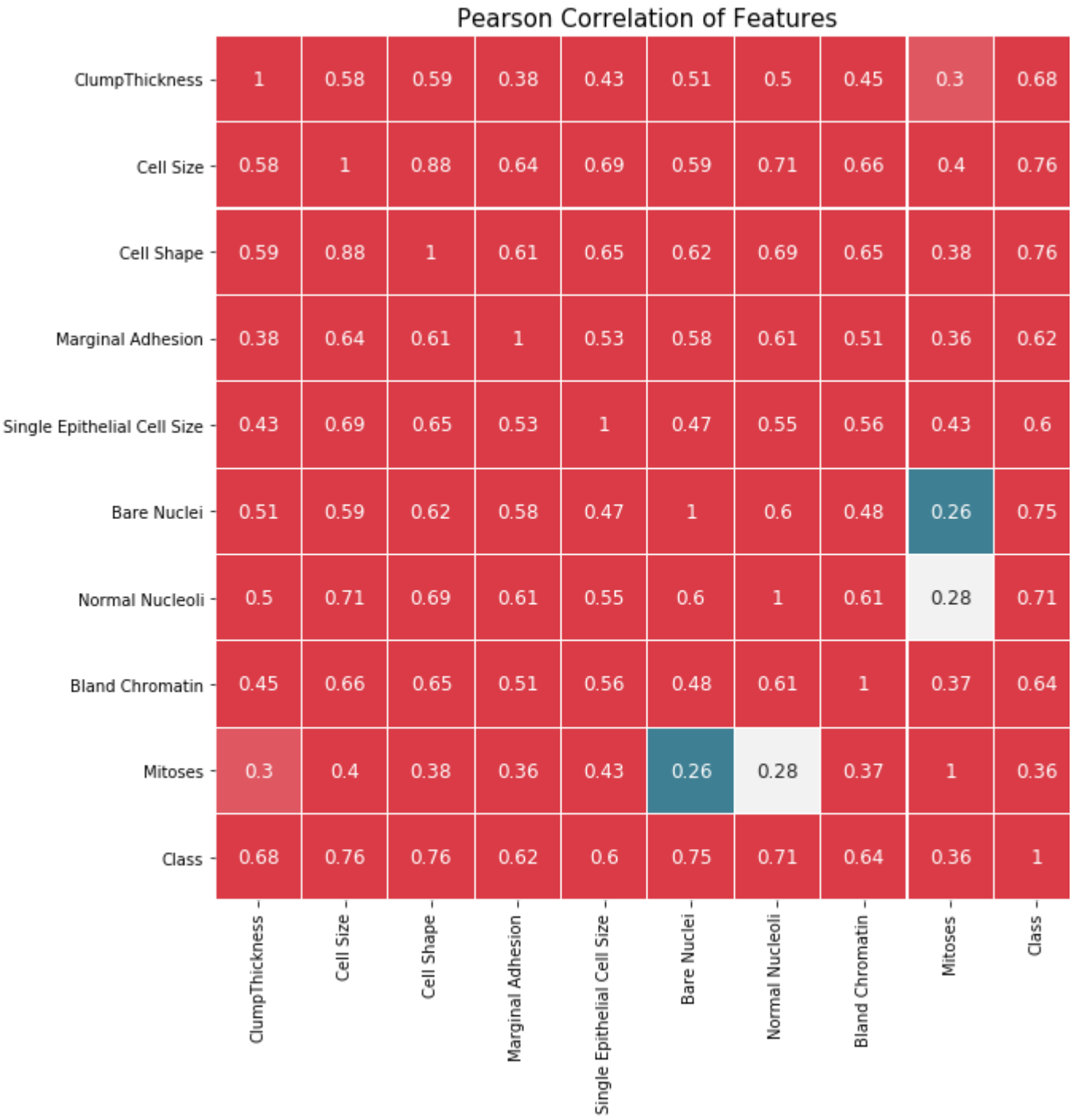
```
# We could see "?" values in column, this should be removed from data set

# Check for missing value in any other column


# No missing values found. So let us try to remove ? from bare nuclei column

# Get count of rows having ?



# 16 values are corrupted. We can either delete them as it forms roughly 2% of data.
# Here we would like to impute it with suitable values
```

Q4. Perform bi variate analysis including correlation, pair inferences.

```
# Check for correlation of variable
```

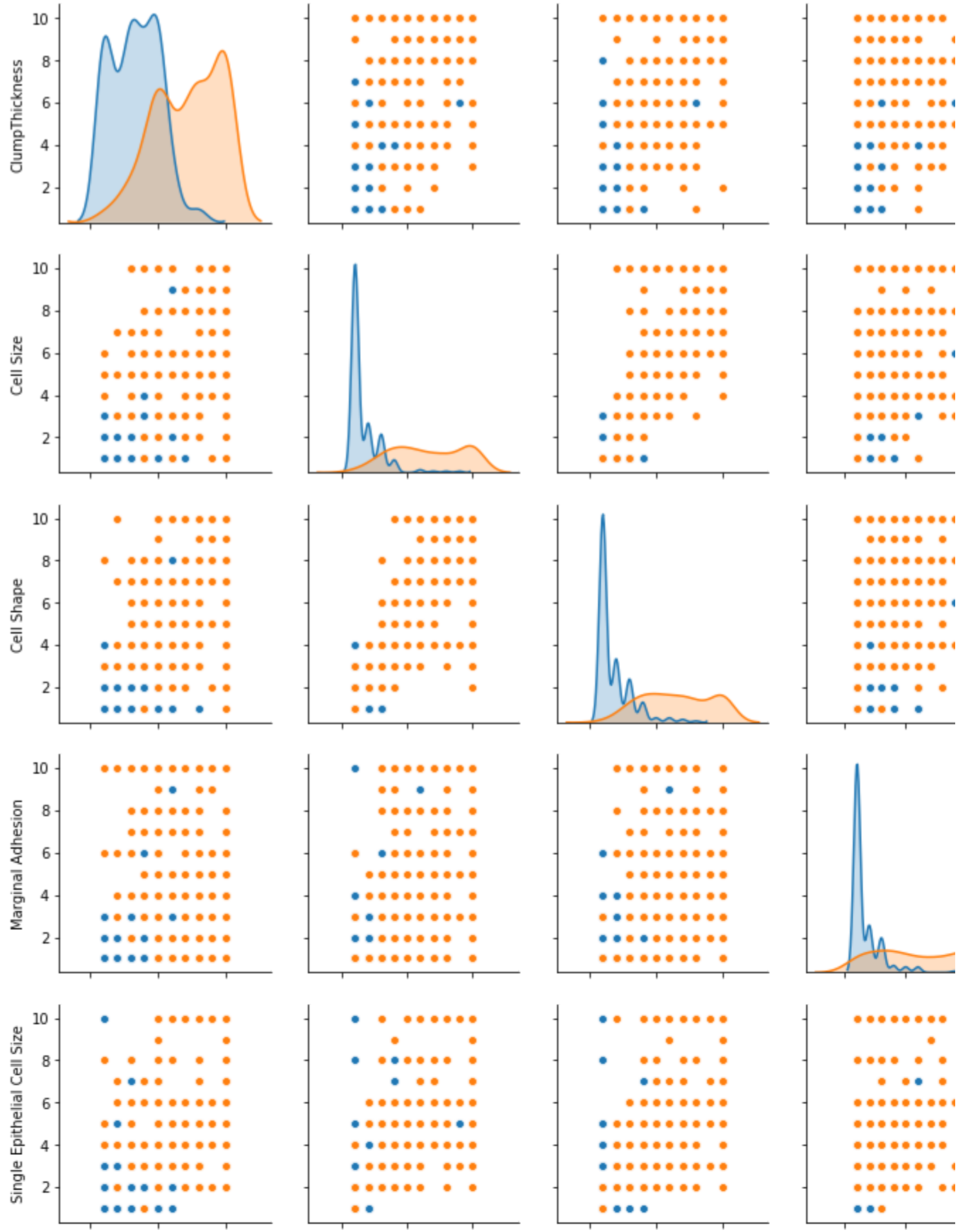
```
#################### 3- Correlation Matrix with Heatmap ##############################

#correlation heatmap of dataset
def correlation_heatmap(df):
    fig , ax = plt.subplots(figsize =(15,10))
    colormap = sns.diverging_palette(220, 10, as_cmap = True)

    _ = sns.heatmap(df.corr(),cmap = colormap,square=True,cbar_kws={'shrink':.5 }, ax=ax,annot=True,

    plt.title('Pearson Correlation of Features', y=1.05, size=15)
```

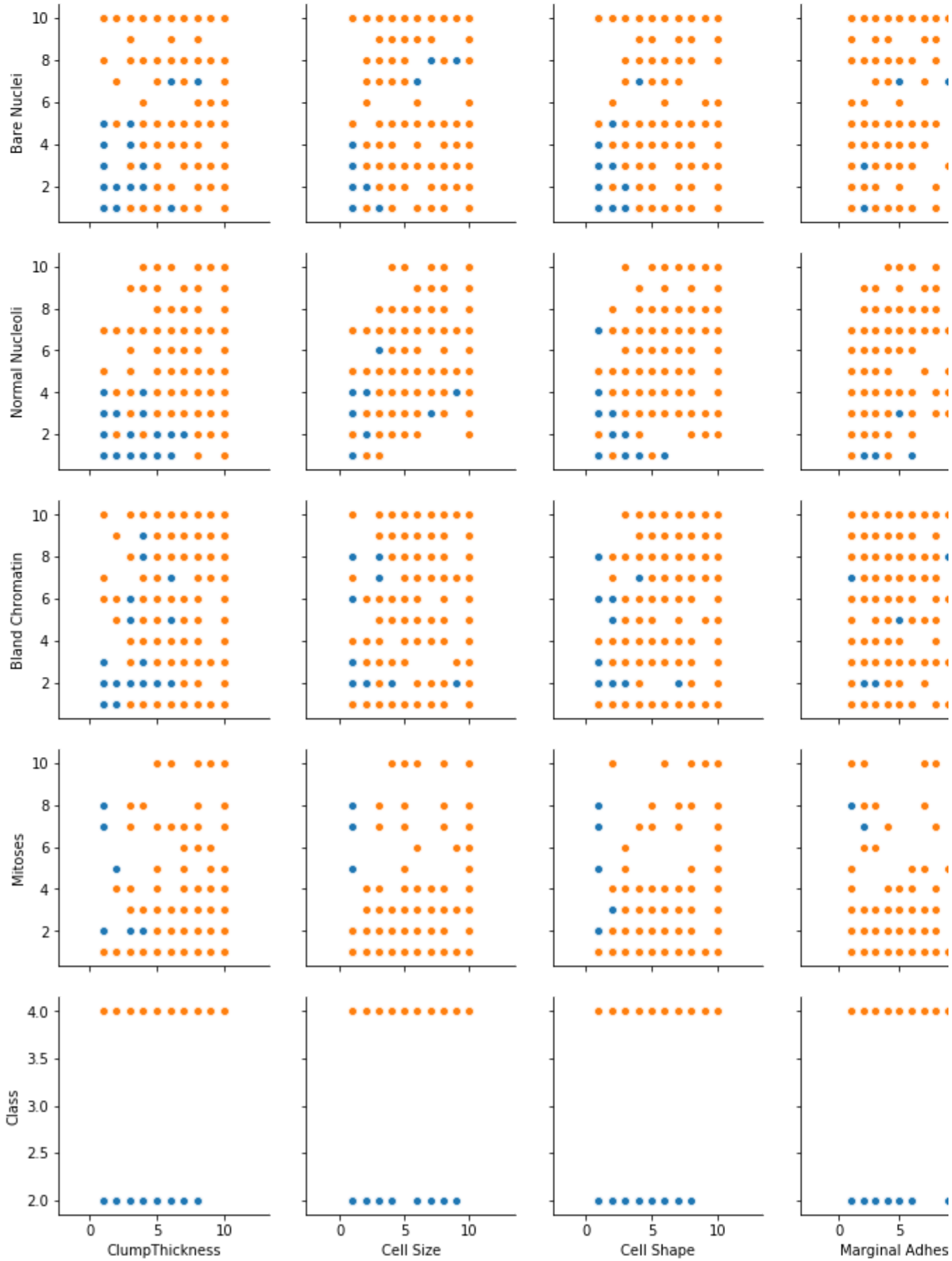
```
correlation_heatmap(data_train)
```




```
## we can see cell size is highly corrreated with cell shape so either of the columns can be dropped


# Cell size shows high significance with cell shape,marginal adhesion, single epithelial cell size,b
# and bland chromatin
# Target variable shows high correlation with most of these variables


#Let us check for pair plots


sns.pairplot(data_train,hue='Class')
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:487: RuntimeWarn
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kdetools.py:34: Runtime
  FAC1 = 2*(np.pi*bw/RANGE)**2
<seaborn.axisgrid.PairGrid at 0x7f1e7238c588>
```





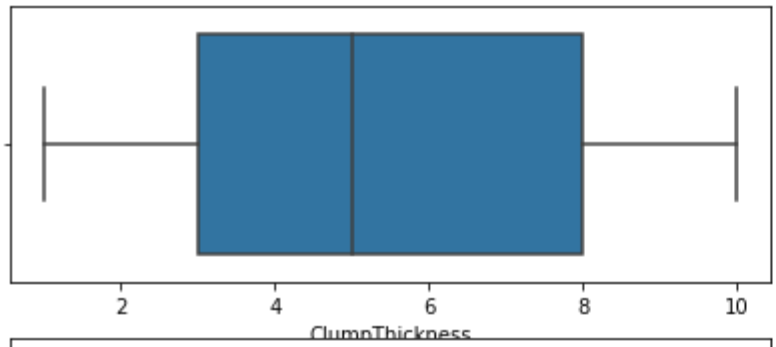

```
# Relationship between variables shows come correlation.
```

```
# Distribution of variables shows most of the values are concentrated on lower side, though range re
# Between 1 to 10
```

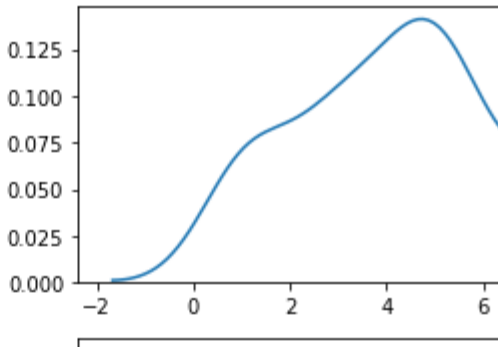
Q5 Remove any unwanted columns or outliers, standardi
processing step

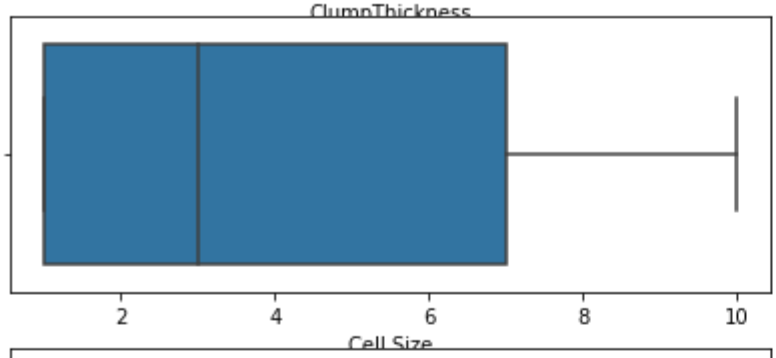
```
# We could see most of the outliers are now removed.
```

```
col_d = data_train.select_dtypes(exclude=['object']).columns
col=2
row = int(np.ceil(len(data_train.select_dtypes(exclude=['object']).columns)))
fig, qaxis = plt.subplots(row,col,figsize=(15,30))

i=0
j=0
k=0

for i in range (len(col_d)):

  ax1= sns.boxplot(x=data_train[col_d[i]], ax = qaxis[j,k])
  k=k+1

  ax2=sns.kdeplot(data_train[col_d[i]],ax = qaxis[j,k])

  k=k+1
  j=j
  if k>=col:
    j=j+1
    k=0
```

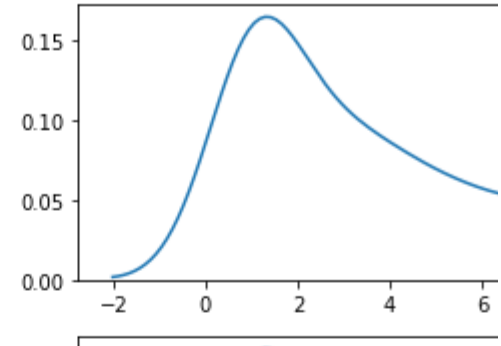


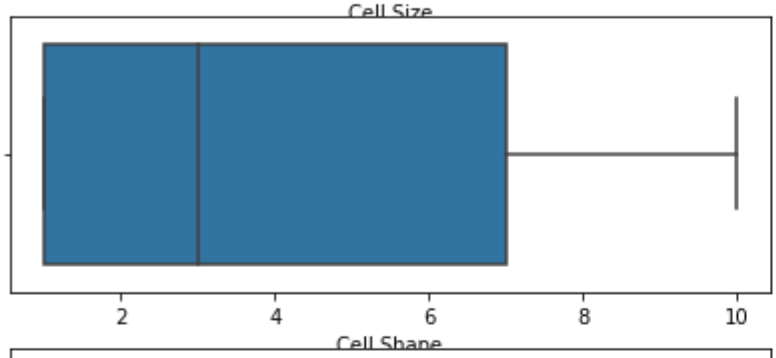
ClumpThickness



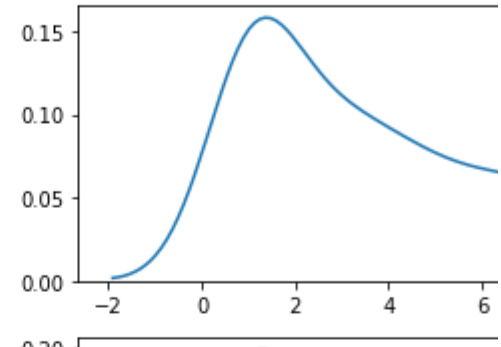


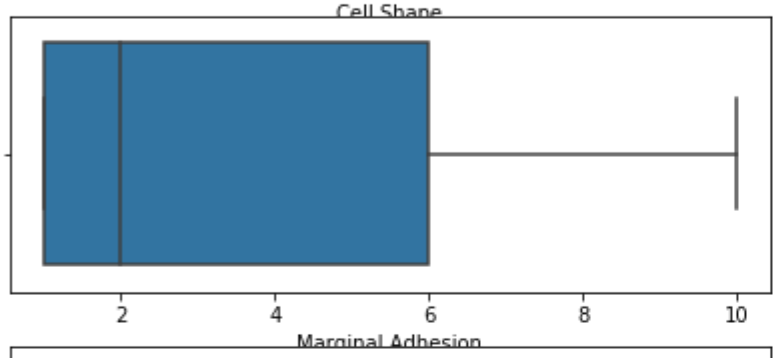
Cell Size



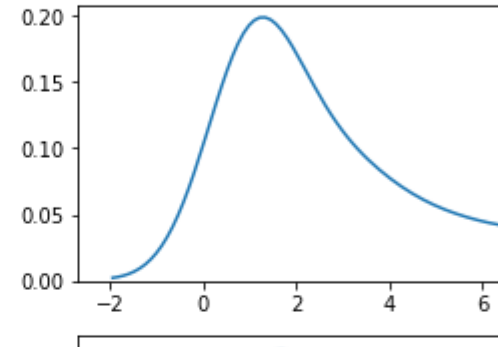


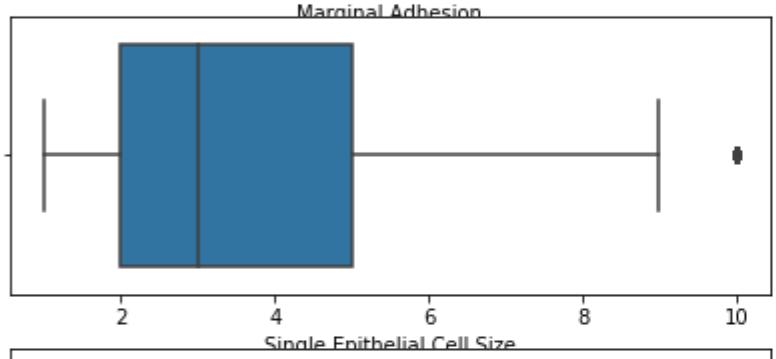
Cell Shape



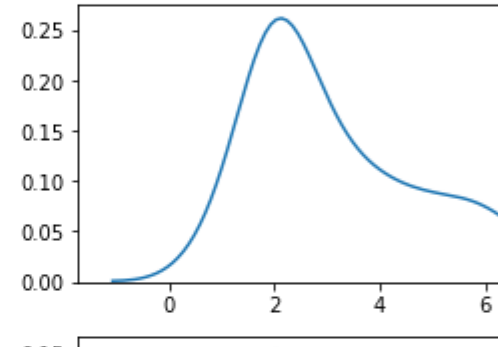


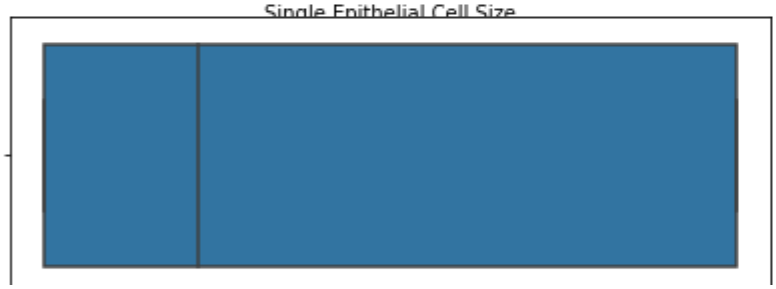
Marginal Adhesion

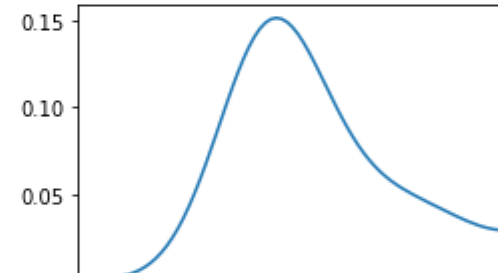


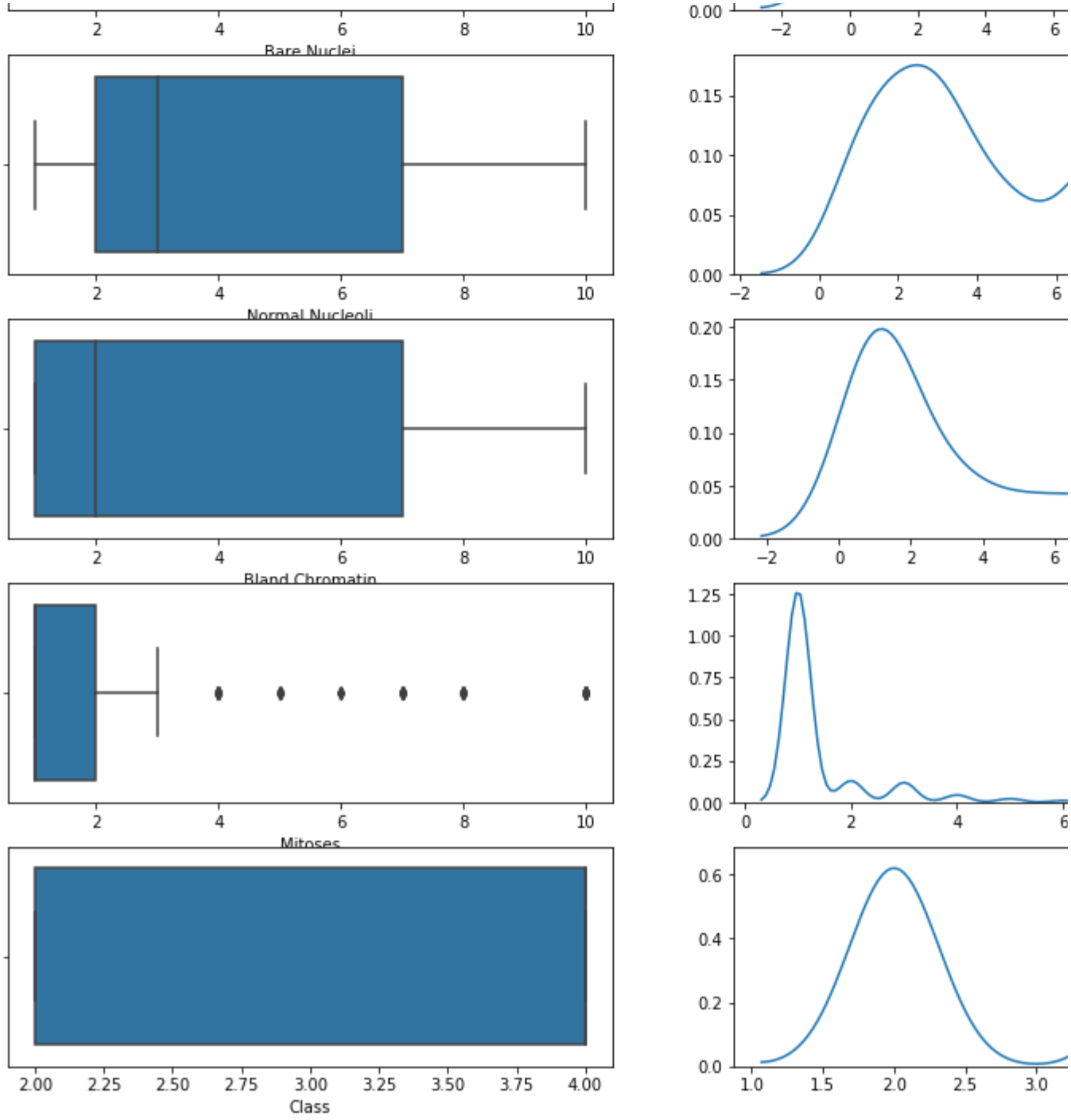


Single Epithelial Cell Size










```
################################ OUTLIERS DETECTION ##################################
out_compare = pd.DataFrame(columns=['Column_name'])

col=data_train.select_dtypes(exclude='object').columns

for i in range(len(col)):
  factor=2
  upper_lmt=data_train[col[i]].mean()+data_train[col[i]].std()*factor
  lower_lmt=data_train[col[i]].mean()-data_train[col[i]].std()*factor
  out_compare.loc[i,'Column_name']=col[i]
  out_compare.loc[i,'upper_lmt']=upper_lmt
  out_compare.loc[i,'lower_lmt']=lower_lmt
  out_compare.loc[i,'upper_outlier(%)']='{0:=5.2f} %'.format((data_train[(data_train[col[i]]>upper_l
  out_compare.loc[i,'lower_outlier(%)']='{0:<5.2f} %'.format((data_train[(data_train[col[i]]<lower_l


out_compare = out_compare.sort_values(by=['upper_outlier(%)','lower_outlier(%)'],ascending=False)
```

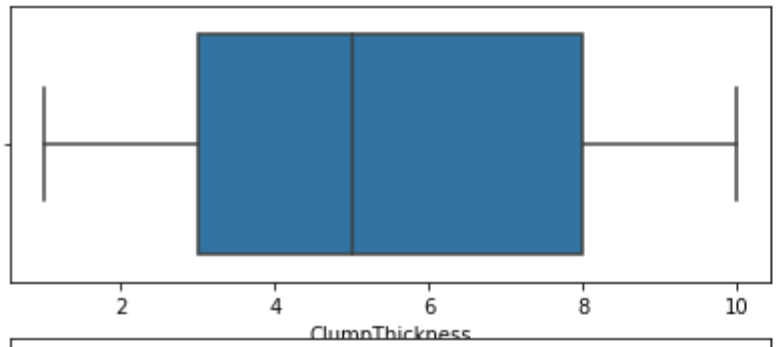
```
out_compare.set_index('Column_name',inplace=True)
out_compare

#Capping the outlier rows with Percentiles
#upper_lim = data['column'].quantile(.95)
#lower_lim = data['column'].quantile(.05)
```

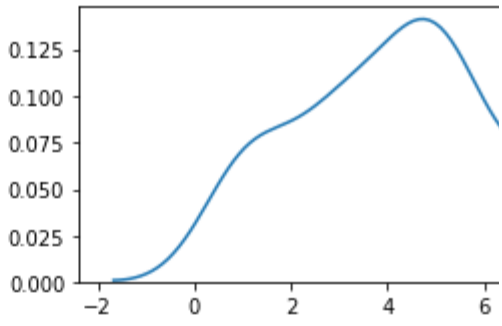
Column_name	upper_lmt	lower_lmt	upper_outlier(%)	lower_outlier(%)
Marginal Adhesion	9.995705	-2.613415	11.88 %	0.00 %
Mitoses	5.972180	-2.201122	7.34 %	0.00 %
Single Epithelial Cell Size	8.734000	-1.053655	6.91 %	0.00 %
Normal Nucleoli	9.448630	-1.107377	4.32 %	0.00 %
ClumpThickness	11.071478	-0.423530	0.00 %	0.00 %
Cell Size	10.654237	-2.304345	0.00 %	0.00 %
Cell Shape	10.483275	-2.003794	0.00 %	0.00 %
Bare Nuclei	12.506822	-2.977664	0.00 %	0.00 %
Bland Chromatin	10.574360	-2.967449	0.00 %	0.00 %
Class	5.029452	1.026704	0.00 %	0.00 %

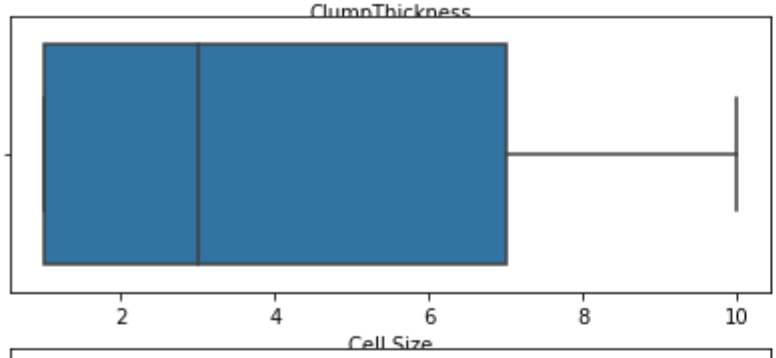
```
##################### OUTLIERS TREATMENT and Verifying it after the treatment  ##################

col=data_train.select_dtypes(exclude='object').columns
out_fx_compare = pd.DataFrame(columns=['Column_name'])

for i in range(len(col)):
  #repl_value = out_compare.loc[col[i],'upper_lmt']
  repl_value = data_train[col[i]].mean()
  #print('column {}:- has mean value {}'.format(col[i],repl_value))
  data_train[col[i]].replace(data_train[(data_train[col[i]]>out_compare.loc[col[i],'upper_lmt'])==Tr
  #data_train[col[i]].replace(data_train[(data_train[col[i]]<out_compare.loc[col[i],'lower_lmt'])==T

  out_fx_compare.loc[i,'Column_name']=col[i]
  out_fx_compare.loc[i,'upper_lmt_old']=out_compare.loc[col[i],'upper_lmt']
  out_fx_compare.loc[i,'lower_lmt_old']=out_compare.loc[col[i],'lower_lmt']
  out_fx_compare.loc[i,'upper_outlier_nw(%)']='{0:=5.2f} %'.format((data_train[(data_train[col[i]]>o
  out_fx_compare.loc[i,'lower_outlier_nw(%)']='{0:<5.2f} %'.format((data_train[(data_train[col[i]]<o

out_fx_compare = out_fx_compare.sort_values(by=['upper_lmt_old'],ascending=False)
out_fx_compare.set_index('Column_name',inplace=True)
out_fx_compare
```

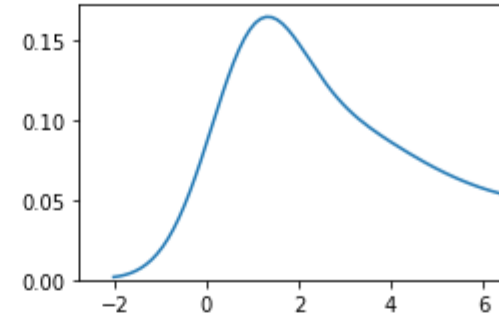
Column_name	upper_lmt_old	lower_lmt_old	upper_outlier_nw(%)	lower_outlie
Bare Nuclei	12.506822	-2.977664	0.00 %	
ClumpThickness	11.071478	-0.423530	0.00 %	
Cell Size	10.654237	-2.304345	0.00 %	
Bland Chromatin	10.574360	-2.967449	0.00 %	
Cell Shape	10.483275	-2.003794	0.00 %	
Marginal Adhesion	9.995705	-2.613415	0.00 %	

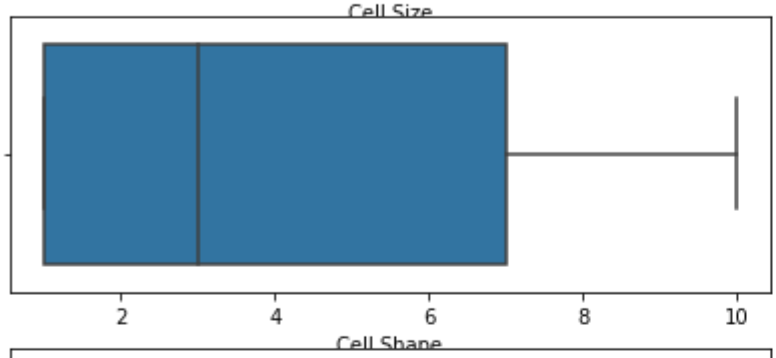
```
col_d = data_train.select_dtypes(exclude=['object']).columns
col=2
row = int(np.ceil(len(data_train.select_dtypes(exclude=['object']).columns)))
fig, qaxis = plt.subplots(row,col,figsize=(15,30))

i=0
j=0
k=0

for i in range (len(col_d)):

  ax1= sns.boxplot(x=data_train[col_d[i]], ax = qaxis[j,k])
  k=k+1

  ax2=sns.kdeplot(data_train[col_d[i]],ax = qaxis[j,k])

  k=k+1
  j=j
  if k>=col:
    j=j+1
    k=0
```

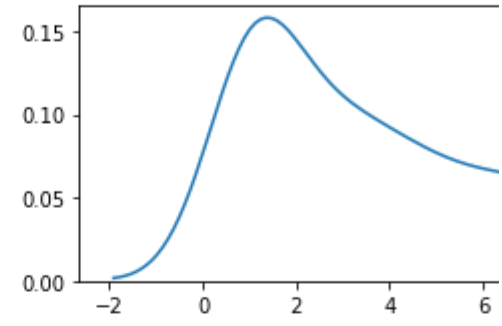

2
4
6
8
10
ClumpThickness

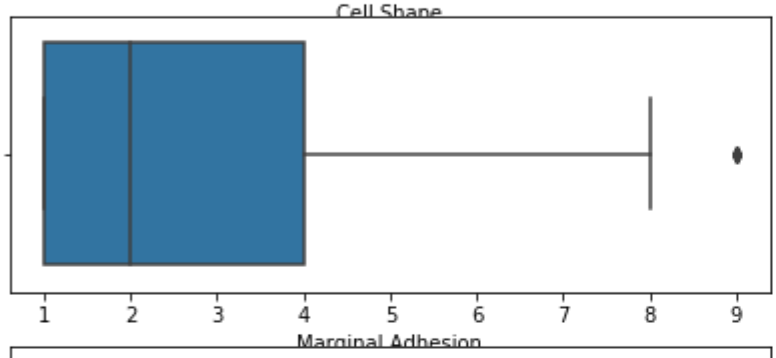

0.125
0.100
0.075
0.050
0.025
0.000
−2
0
2
4
6

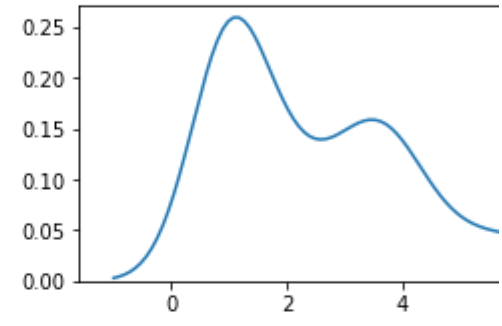

2
4
6
8
10
Cell Size

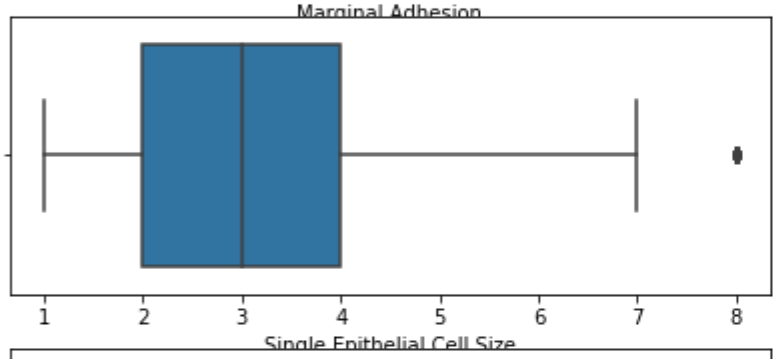

0.15
0.10
0.05
0.00
−2
0
2
4
6

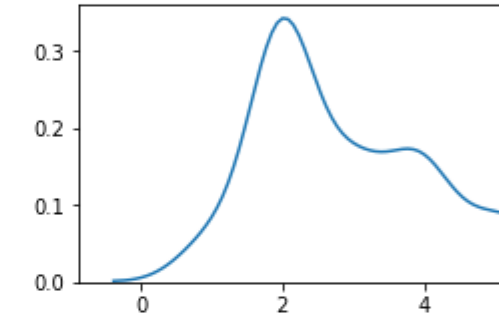

2
4
6
8
10
Cell Shape

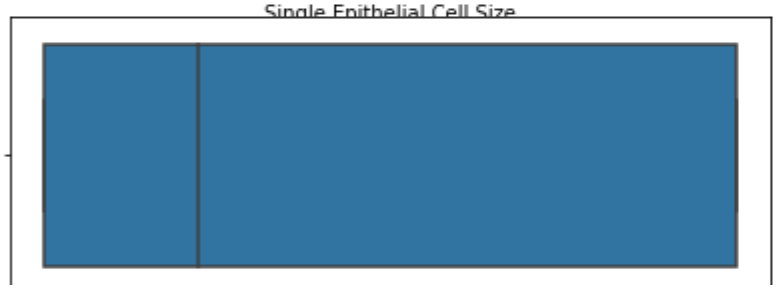

0.15
0.10
0.05
0.00
−2
0
2
4
6

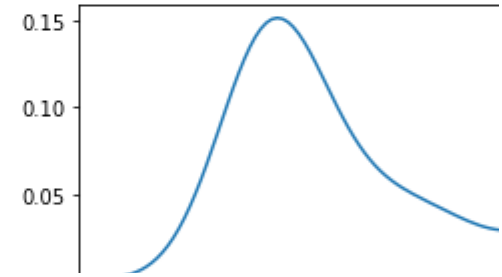

1
2
3
4
5
6
7
8
9
Marginal Adhesion

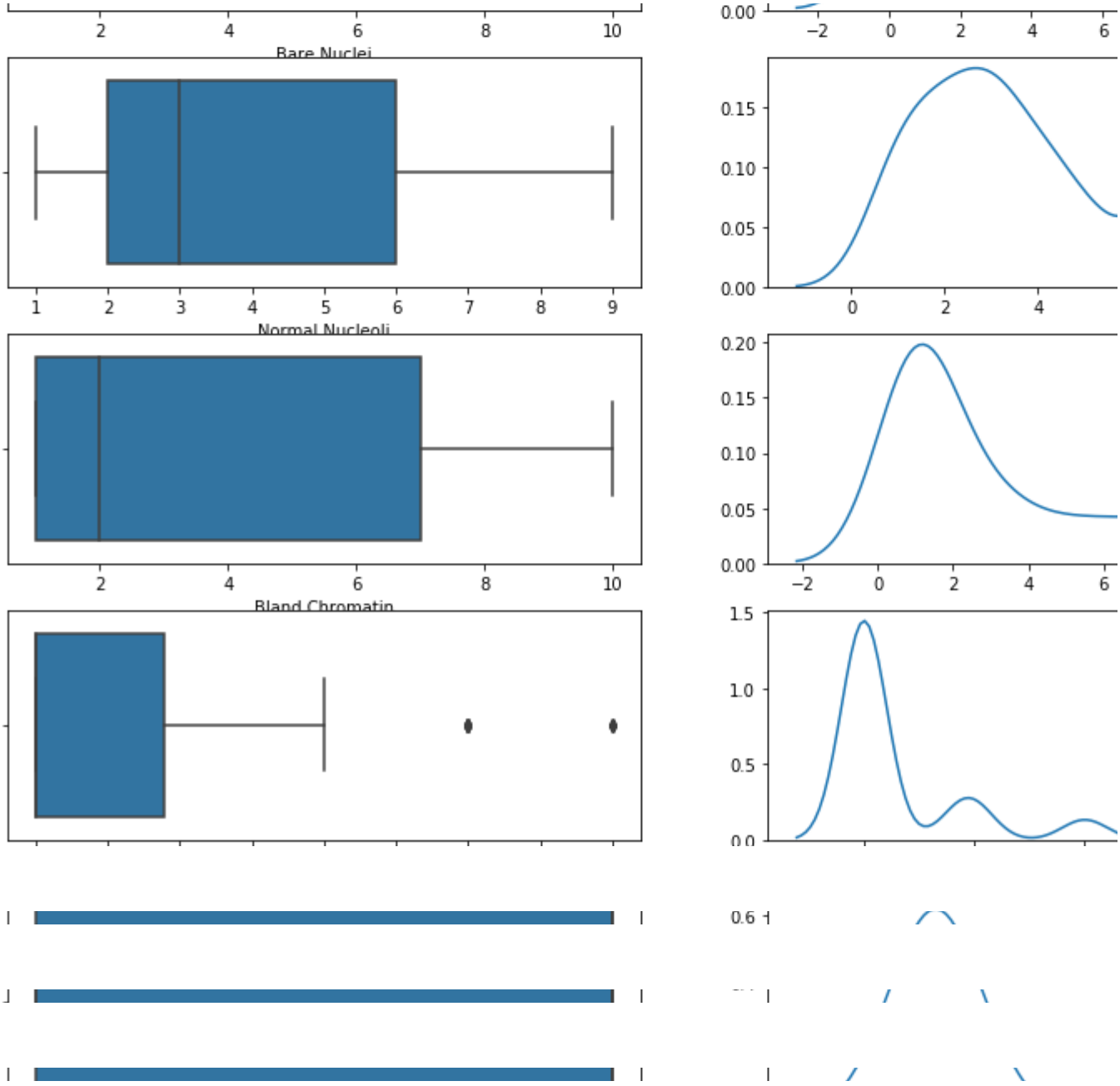

0.25
0.20
0.15
0.10
0.05
0.00
0
2
4


1
2
3
4
5
6
7
8
Single Epithelial Cell Size


0.3
0.2
0.1
0.0
0
2
4




0.15
0.10
0.05



Q6 Create a covariance matrix for identifying Principal co

```
# PCA
# Step 1 - Create covariance matrix
```

```
data_train.columns
```

```
Index(['ClumpThickness', 'Cell Size', 'Cell Shape', 'Marginal Adhesion',
       'Single Epithelial Cell Size', 'Bare Nuclei', 'Normal Nucleoli',
       'Bland Chromatin', 'Mitoses', 'Class'],
      dtype='object')
```

```
X = np.array(data_train[['ClumpThickness', 'Cell Size', 'Cell Shape', 'Marginal Adhesion',
```

```
        'Single Epithelial Cell Size', 'Bare Nuclei', 'Normal Nucleoli',
        'Bland Chromatin', 'Mitoses']])
Y = np.array(data_train[['Class']])
```

```
from sklearn.model_selection import train_test_split
```

```
test_size = 0.30 # taking 70:30 training and test set
seed = 7  # Random numbmer seeding for reapeatability of the code
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
```

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
```

```
X_train_std = sc.fit_transform(X)  # scale training and test data independently to prevent data leak
```

```
cov_matrix = np.cov(X_train_std.T)

print('Covariance Matrix \n%s', cov_matrix)
```

```
Covariance Matrix
%s [[1.0021645  0.57940715 0.59023092 0.36776197 0.40615245 0.50722844
  0.49605714 0.45182864 0.3053708 ]
 [0.57940715 1.0021645  0.87930297 0.5872503  0.65802677 0.59086656
  0.64758246 0.65859238 0.35414853]
 [0.59023092 0.87930297 1.0021645  0.54874667 0.62856071 0.61641993
  0.62504238 0.65614155 0.31386109]
 [0.36776197 0.5872503  0.54874667 1.0021645  0.49541234 0.5307154
  0.572818   0.49938679 0.24635951]
 [0.40615245 0.65802677 0.62856071 0.49541234 1.0021645  0.49763307
  0.52336077 0.59485519 0.34020424]
 [0.50722844 0.59086656 0.61641993 0.5307154  0.49763307 1.0021645
  0.61345391 0.47910267 0.32399339]
 [0.49605714 0.64758246 0.62504238 0.572818   0.52336077 0.61345391
  1.0021645  0.58354739 0.26007403]
 [0.45182864 0.65859238 0.65614155 0.49938679 0.59485519 0.47910267
  0.58354739 1.0021645  0.32993549]
 [0.3053708  0.35414853 0.31386109 0.24635951 0.34020424 0.32399339
  0.26007403 0.32993549 1.0021645 ]]
```

Q7 Identify eigen values and eigen vector

```
# Step 2- Get eigen values and eigen vector
```

```
eig_vals, eig_vecs = np.linalg.eig(cov_matrix)
print('Eigen Vectors \n%s', eig_vecs)
print('\n Eigen Values \n%s', eig_vals)
```

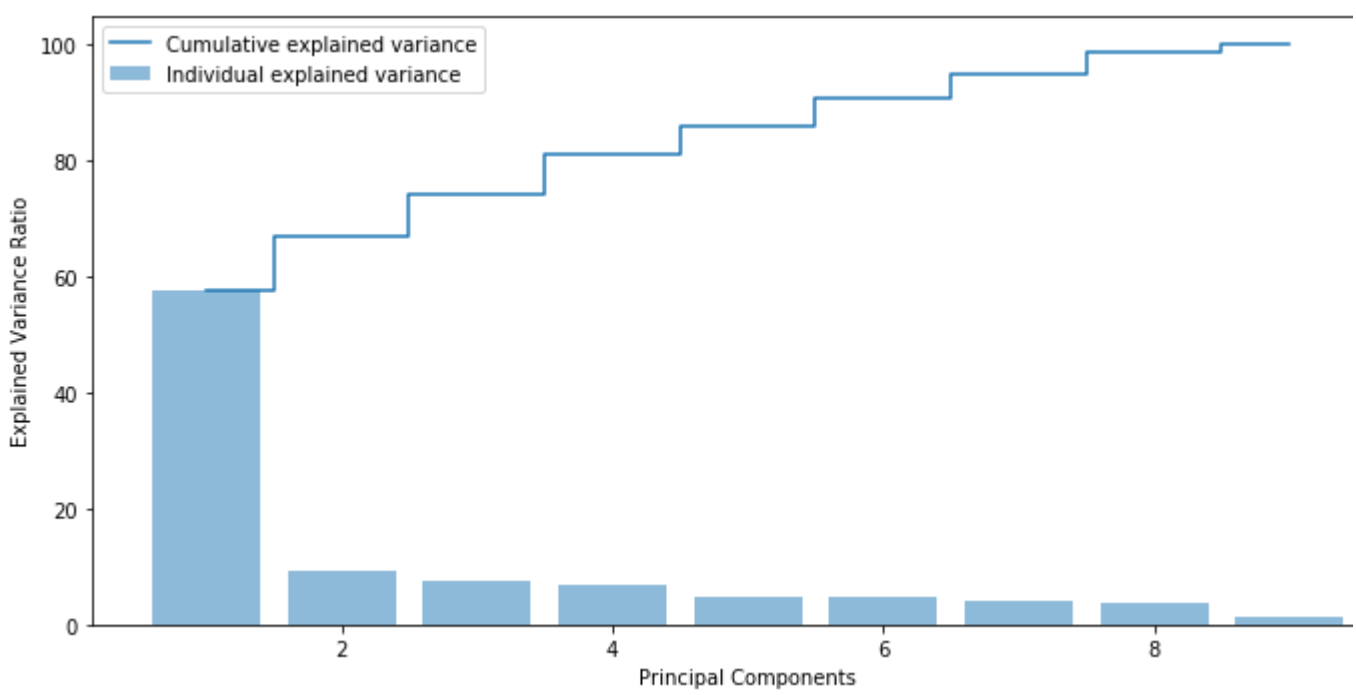
```
Eigen Vectors
%s [[ 0.30271964 -0.08533972  0.00875547  0.77005931  0.12002929 -0.06615621
  -0.44459013  0.25943931  0.15574705]
 [ 0.39230729  0.06464043  0.71607756  0.01053363  0.24561847  0.19783783
   0.41789617  0.02087923  0.2334463 ]
 [ 0.38695785  0.09972318 -0.68497896  0.09052587  0.27037636  0.06651213
   0.49245676 -0.04926675  0.20114595]
 [ 0.31496826  0.22935583 -0.05368879 -0.35055369 -0.52394019 -0.12554081
  -0.13186947  0.4709439   0.43928235]
 [ 0.33501608 -0.03345486 -0.04235198 -0.38973947  0.32084631  0.12732012
  -0.57641019 -0.48139432  0.22318711]
 [ 0.33481544  0.04344812  0.0843919   0.21072965 -0.49450488 -0.44498161
   0.1264908  -0.60161883 -0.11829226]
 [ 0.34873602  0.20922646 -0.05457941  0.02784333 -0.27502646  0.64764847
  -0.0964188   0.04996269 -0.5691857 ]
 [ 0.34309799  0.01493694  0.03579774 -0.2538083   0.34758893 -0.53965401
  -0.03838694  0.32115788 -0.5474487 ]
 [ 0.20455862 -0.93754154 -0.04360654 -0.11965424 -0.18361042  0.10679493
   0.09370387  0.09335721 -0.01841267]]

 Eigen Values
%s [5.19056052 0.84571993 0.11681916 0.66687039 0.61267966 0.34050845
 0.37734441 0.43917215 0.42980586]
```

Q8 Find variance and cumulative variance by each eigen

```
tot = sum(eig_vals)
var_exp = [( i /tot ) * 100 for i in sorted(eig_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)   # array of size =  as many PC dimensions
print("Cumulative Variance Explained", cum_var_exp)
```

```
Cumulative Variance Explained [ 57.54833119  66.92492362  74.31859104  81.11143972  85.9
  90.74589707  94.92955717  98.70481279 100.        ]
```

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



Q9 Use PCA command from sklearn and find Principal Cc data to components formed

```
from sklearn.decomposition import PCA


pca = PCA()


X = pca.fit_transform(X)



explained_variance = pca.explained_variance_ratio_
explained_variance
```

```
array([0.64146085, 0.10654322, 0.07407465, 0.05891574, 0.0412067 ,
       0.02982652, 0.02286171, 0.01711763, 0.00799298])
```

Q10 Find correlation between components and features

```
data = pd.DataFrame(X)


data.corr()
```

	0	1	2	3	4	5	
0	1.000000e+00	5.121110e-18	2.974911e-17	-4.820692e-17	1.374580e-17	-1.028383e-16	-3
1	5.121110e-18	1.000000e+00	2.288959e-16	-2.921934e-17	-7.614643e-17	5.195125e-17	5
2	2.974911e-17	2.288959e-16	1.000000e+00	5.251139e-17	2.922771e-16	-1.704121e-17	5
3	-4.820692e-17	-2.921934e-17	5.251139e-17	1.000000e+00	-2.702019e-16	-3.226976e-17	-8
4	1.374580e-17	-7.614643e-17	2.922771e-16	-2.702019e-16	1.000000e+00	-2.990586e-17	1
5	-1.028383e-16	5.195125e-17	-1.704121e-17	-3.226976e-17	-2.990586e-17	1.000000e+00	3
6	-3.454798e-17	5.792652e-18	5.453922e-17	-8.777742e-17	1.535533e-16	3.200319e-16	1.
7	1.179479e-17	1.559300e-17	-6.221223e-17	-3.173888e-16	-3.069459e-16	3.188164e-17	3
8	3.797835e-18	1.326132e-16	-5.313973e-17	5.269687e-17	2.491190e-16	-1.305128e-16	1

Content Based Recommendation System - Optional (Q11 graded)

Q11 Read the Dataset `movies_metadata.csv`

1 cell hidden

Q12 Create a new column with name 'description' combin `'tagline'` columns in the given dataset

1 cell hidden

Q13 Lets drop the null values in `description` column

1 cell hidden

Q14 Keep the first occurance and drop duplicates of eacl

1 cell hidden

Q15 As we might have dropped a few rows with duplicate just reset the index [make sure you are not adding any ne dataframe while doing reset index]

2 cells hidden

Q16 Create cosine similarity matrix

1 cell hidden

Q17 Write a function with name `recommend` which takes t returns a list of 10 recommended title names in the outpu cosine similarities

Hint:

```
titles = df['title']
indices = pd.Series(df.index, index=df['title'])

def recommend(title):
idx = indices[title]
sim_scores = list(enumerate(cosine_similarities[idx]))
sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
sim_scores = sim_scores[1:31]
movie_indices = [i[0] for i in sim_scores]
return titles.iloc[movie_indices]
```

Q18 Give the recommendations from above functions for `Godfather` and `The Dark Knight Rises`

2 cells hidden

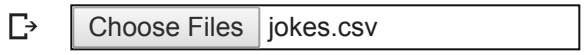
Popularity Based Recommendation System

3 cells hidden

Q19 Read the dataset(jokes.csv)

Take care about the header in read_csv() as there are no column names given in the dataset.

```
import io


from google.colab import files
uploaded1  = files.upload()
```



- **jokes.csv**(application/vnd.ms-excel) - 11738238 bytes, last modified: 10/13/2019 - 100% done

```
Saving jokes.csv to jokes (3).csv
```

```
data_raw_jk = pd.read_csv(io.BytesIO(uploaded1['jokes.csv']))

data_train_jk = data_raw_jk.copy(deep=True)
```

Q20 Consider `ratings` named dataframe with only first 2 columns from 1(first column is 0) of dataset

```
data_train_jk.head()
```

	NumJokes	Joke1	Joke2	Joke3	Joke4	Joke5	Joke6	Joke7	Joke8	Joke9	Joke10	Joke1
0	74	-7.82	8.79	-9.66	-8.16	-7.52	-8.50	-9.85	4.17	-8.98	-4.76	-8.5
1	100	4.08	-0.29	6.36	4.37	-2.38	-9.66	-0.73	-5.34	8.88	9.22	6.7
2	49	99.00	99.00	99.00	99.00	9.03	9.27	9.03	9.27	99.00	99.00	7.3
3	48	99.00	8.35	99.00	99.00	1.80	8.16	-2.82	6.21	99.00	1.84	7.3
4	91	8.50	4.61	-4.17	-5.39	1.36	1.60	7.04	4.61	-0.44	5.73	8.2

5 rows × 101 columns

```
data_train_jk.shape
```

```
(24983, 101)
```

```
data_train_jk = data_train_jk.drop(columns='NumJokes')
```

```
data_train_jk1 = data_train_jk.iloc[0:200,0:101]
```

```
data_train_jk1.shape
```

```
(200, 100)
```

Q21 Change the column indices from 0 to 99

```
value = np.arange(0,100)
```

```
value
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

```
col= data_train_jk1.columns
```