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
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473

Enter your authorization code:
··········
Mounted at /content/drive
```

```
data = pd.read_csv("/content/drive/My Drive/13oct/breast-cancer-wisconsin-data.csv")
```

```
data
```

28	1067444	2	1	1	1	2	1
29	1070935	1	1	3	1	2	1
...	...	...	...	...	...	...	...
669	1350423	5	10	10	8	5	5
670	1352848	3	10	7	8	5	8
671	1353092	3	2	1	2	2	1
672	1354840	2	1	1	1	2	1
673	1354840	5	3	2	1	3	1
674	1355260	1	1	1	1	2	1
675	1365075	4	1	4	1	2	1
676	1365328	1	1	2	1	2	1
677	1368267	5	1	1	1	2	1
678	1368273	1	1	1	1	2	1
679	1368882	2	1	1	1	2	1
680	1369821	10	10	10	10	5	10
681	1371026	5	10	10	10	4	10
682	1371920	5	1	1	1	2	1
683	466906	1	1	1	1	2	1
684	466906	1	1	1	1	2	1
685	534555	1	1	1	1	2	1
686	536708	1	1	1	1	2	1
687	566346	3	1	1	1	2	1
688	603148	4	1	1	1	2	1
689	654546	1	1	1	1	2	1
690	654546	1	1	1	3	2	1
691	695091	5	10	10	5	4	5
692	714039	3	1	1	1	2	1
693	763235	3	1	1	1	2	1
694	776715	3	1	1	1	3	2
695	841769	2	1	1	1	2	1
696	888820	5	10	10	3	7	3
697	897471	4	8	6	4	3	4
698	897471	4	8	8	5	4	5

699 rows × 11 columns

Q1. Load the Data file into Python DataFrame and view

```
print(data.shape)

data.head(10)
```

```
(699, 11)
```

	ID	ClumpThickness	Cell Size	Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Nor Nucle
0	1000025	5	1	1	1	2	1	
1	1002945	5	4	4	5	7	10	
2	1015425	3	1	1	1	2	2	
3	1016277	6	8	8	1	3	4	
4	1017023	4	1	1	3	2	1	
5	1017122	8	10	10	8	7	10	
6	1018099	1	1	1	1	2	10	
7	1018561	2	1	2	1	2	1	
8	1033078	2	1	1	1	2	1	

```
data['Bare Nuclei'].unique() # Bare nuclei is not available might have missing values.

data = data.replace('?', np.nan) # replacing ? with NAN

data = data.apply(lambda x: x.fillna(x.median()),axis=0)

# converting the Bare Nuclei column from object / string type to int
data['Bare Nuclei'] = data['Bare Nuclei'].astype('int64')


  # 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


data.isnull().any()
```

```
ID                             False
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
```

```
data.dtypes
```

```
ID                           int64
ClumpThickness               int64
Cell Size                    int64
Cell Shape                   int64
Marginal Adhesion            int64
Single Epithelial Cell Size  int64
```

```
data = data.replace('?', 0)
```

```
Bland Chromatin              int64


dtype: object
```

```
data["Bare Nuclei"] = pd.to_numeric(data["Bare Nuclei"])
```

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

```
print(data.shape)
```

```
(699, 11)
```

```
data.describe
```

1	1002945	5	4	...	2	1	2
2	1015425	3	1	...	1	1	2
3	1016277	6	8	...	7	1	2
4	1017023	4	1	...	1	1	2
5	1017122	8	10	...	7	1	4
6	1018099	1	1	...	1	1	2
7	1018561	2	1	...	1	1	2
8	1033078	2	1	...	1	5	2
9	1033078	4	2	...	1	1	2
10	1035283	1	1	...	1	1	2
11	1036172	2	1	...	1	1	2
12	1041801	5	3	...	4	1	4
13	1043999	1	1	...	1	1	2
14	1044572	8	7	...	5	4	4
15	1047630	7	4	...	3	1	4
16	1048672	4	1	...	1	1	2
17	1049815	4	1	...	1	1	2
18	1050670	10	7	...	1	2	4
19	1050718	6	1	...	1	1	2
20	1054590	7	3	...	4	4	4
21	1054593	10	5	...	10	1	4
22	1056784	3	1	...	1	1	2
23	1057013	8	4	...	3	1	4
24	1059552	1	1	...	1	1	2
25	1065726	5	2	...	6	1	4
26	1066373	3	2	...	1	1	2
27	1066979	5	1	...	1	1	2
28	1067444	2	1	...	1	1	2
29	1070935	1	1	...	1	1	2
..	...	...	...	...	...	...	...
669	1350423	5	10	...	10	1	4
670	1352848	3	10	...	4	1	4
671	1353092	3	2	...	1	1	2
672	1354840	2	1	...	1	1	2
673	1354840	5	3	...	1	1	2
674	1355260	1	1	...	1	1	2
675	1365075	4	1	...	1	1	2
676	1365328	1	1	...	1	1	2
677	1368267	5	1	...	1	1	2
678	1368273	1	1	...	1	1	2
679	1368882	2	1	...	1	1	2
680	1369821	10	10	...	10	7	4
681	1371026	5	10	...	6	3	4
682	1371920	5	1	...	2	1	2
683	466906	1	1	...	1	1	2
684	466906	1	1	...	1	1	2
685	534555	1	1	...	1	1	2
686	536708	1	1	...	1	1	2
687	566346	3	1	...	3	1	2
688	603148	4	1	...	1	1	2
689	654546	1	1	...	1	8	2
690	654546	1	1	...	1	1	2
691	695091	5	10	...	4	1	4
692	714039	3	1	...	1	1	2
693	763235	3	1	...	1	2	2
694	776715	3	1	...	1	1	2
695	841769	2	1	...	1	1	2
696	888820	5	10	...	10	2	4
697	897471	4	8	...	6	1	4
698	897471	4	8	...	4	1	4

[699 rows x 11 columns]>

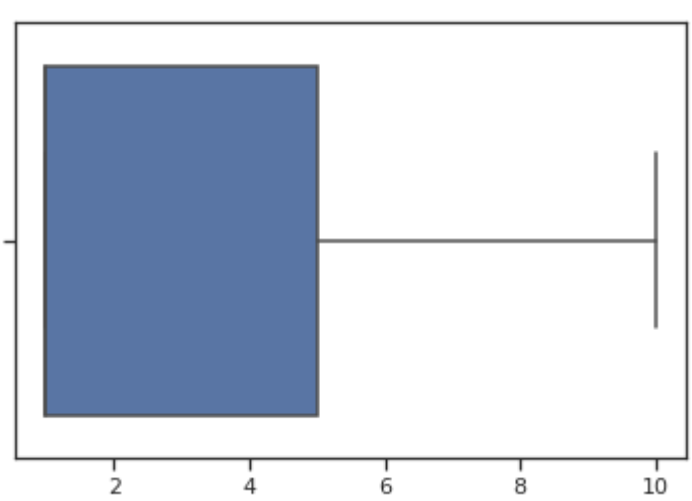
```
data.describe().transpose()
```

	count	mean	std	min	25%	50
ID	699.0	1.071704e+06	617095.729819	61634.0	870688.5	1171710.
ClumpThickness	699.0	4.417740e+00	2.815741	1.0	2.0	4.
Cell Size	699.0	3.134478e+00	3.051459	1.0	1.0	1.
Cell Shape	699.0	3.207439e+00	2.971913	1.0	1.0	1.
Marginal Adhesion	699.0	2.806867e+00	2.855379	1.0	1.0	1.
Single Epithelial Cell Size	699.0	3.216023e+00	2.214300	1.0	2.0	2.
Bare Nuclei	699.0	3.486409e+00	3.621929	1.0	1.0	1.
Normal Nucleoli	699.0	3.437768e+00	2.438364	1.0	2.0	3.
Bland Chromatin	699.0	2.866953e+00	3.053634	1.0	1.0	1.
Mitoses	699.0	1.589413e+00	1.715078	1.0	1.0	1.
Class	699.0	2.689557e+00	0.951273	2.0	2.0	2.

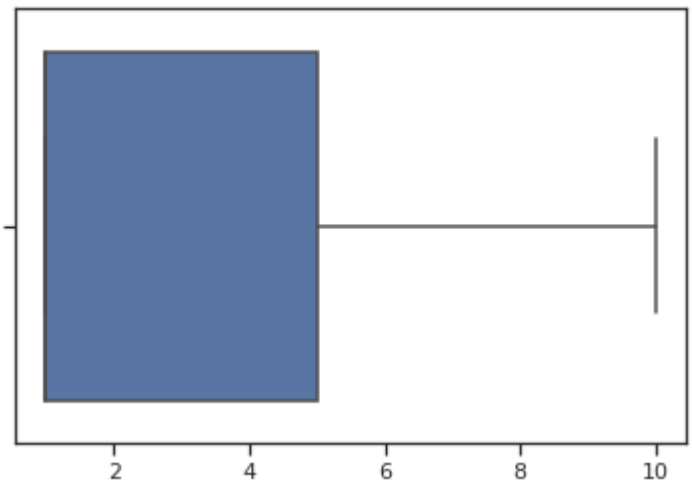
Q3 Check for missing value check, incorrect data, dupli perform imputation with mean, median, mode as neces

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

# Check for missing value in any other column

data = data.replace('?', 0)

plt = sns.boxplot(data[['ClumpThickness']])
```

```
plt = sns.boxplot(data[['Cell Size']])
```




```
plt = sns.boxplot(data[['Cell Shape']])
```

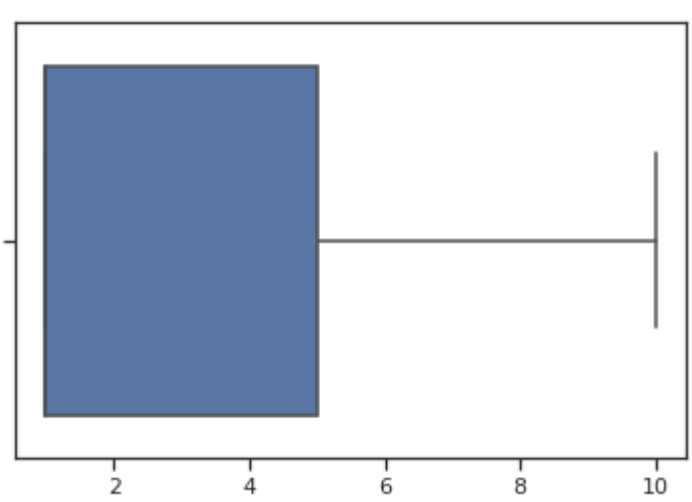



```
plt = sns.boxplot(data[['Marginal Adhesion']])
```

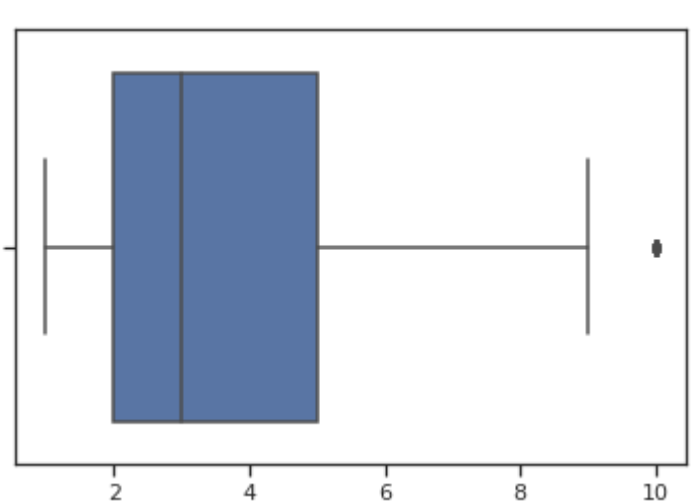
```
data['Marginal Adhesion']=np.where(data['Marginal Adhesion']>8,4 ,data['Marginal Adhesion'])
```

```
plt = sns.boxplot(data[['Single Epithelial Cell Size']])
```

```
plt = sns.boxplot(data[['Bare Nuclei']])
```

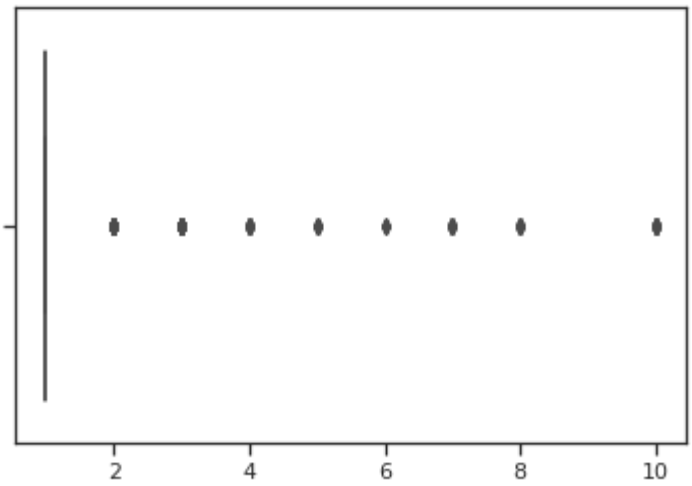



```
plt = sns.boxplot(data[['Normal Nucleoli']])
```




```
plt = sns.boxplot(data[['Bland Chromatin']])
```

```
plt = sns.boxplot(data[['Mitoses']])
```




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

# Get count of rows having ?


print(data.shape)
```

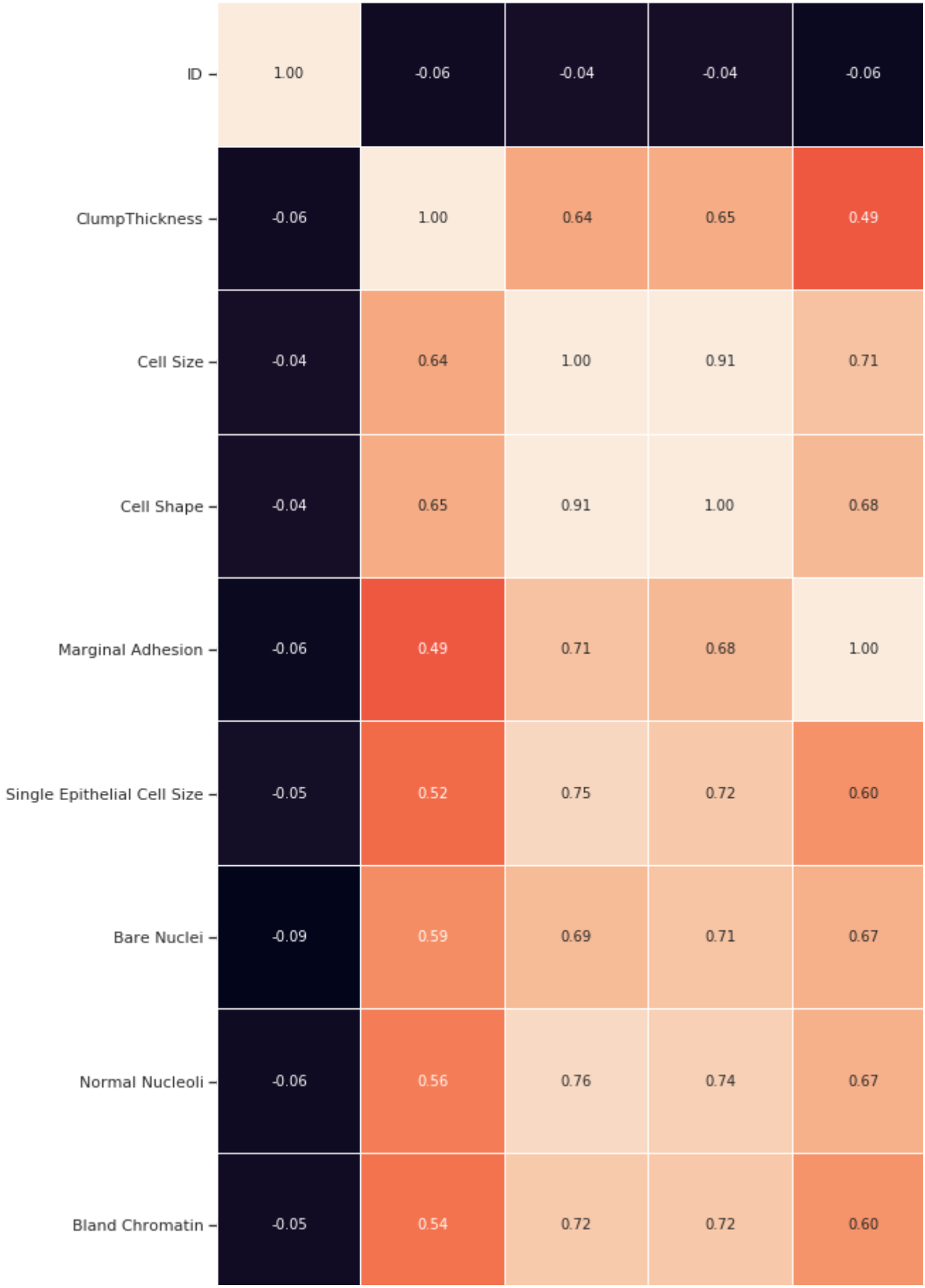
```
(699, 11)
```

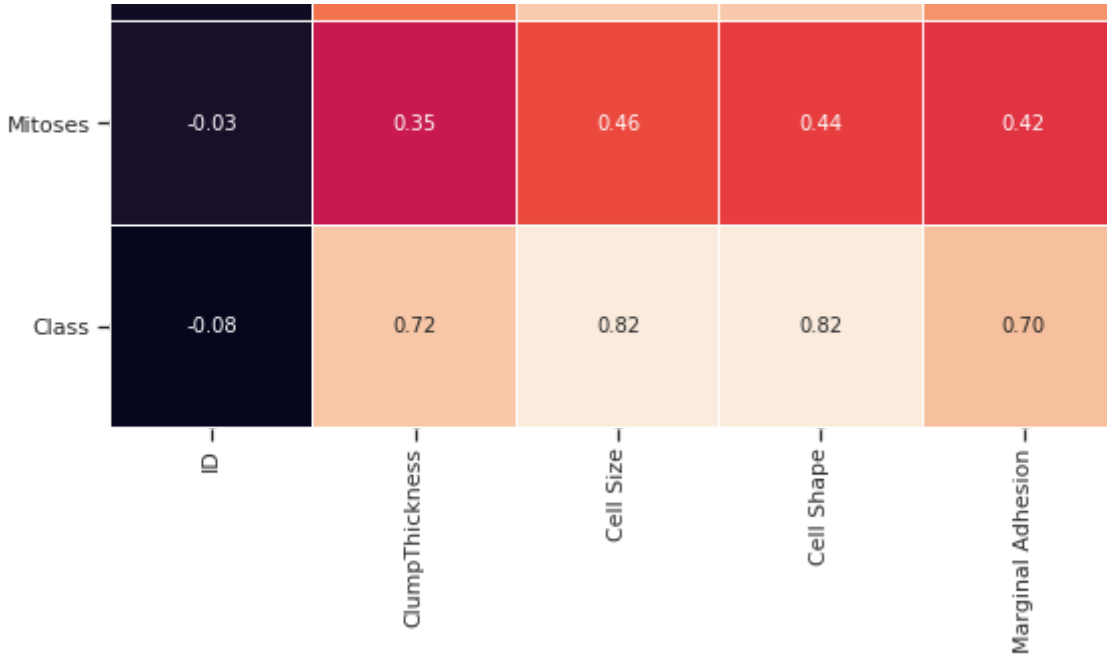
```
# 16 values are corrupted. We can either delete them as it forms roughly 2% of data.
# Here we would like to impute it with suitable values
```

Q4. Perform bi variate analysis including correlation, pa the inferences.

```
# Check for correlation of variable


import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from scipy.stats import zscore
from sklearn.preprocessing import Imputer
from sklearn.metrics import accuracy_score
import seaborn as sns
import os
%matplotlib inline
import pandas as pd
```

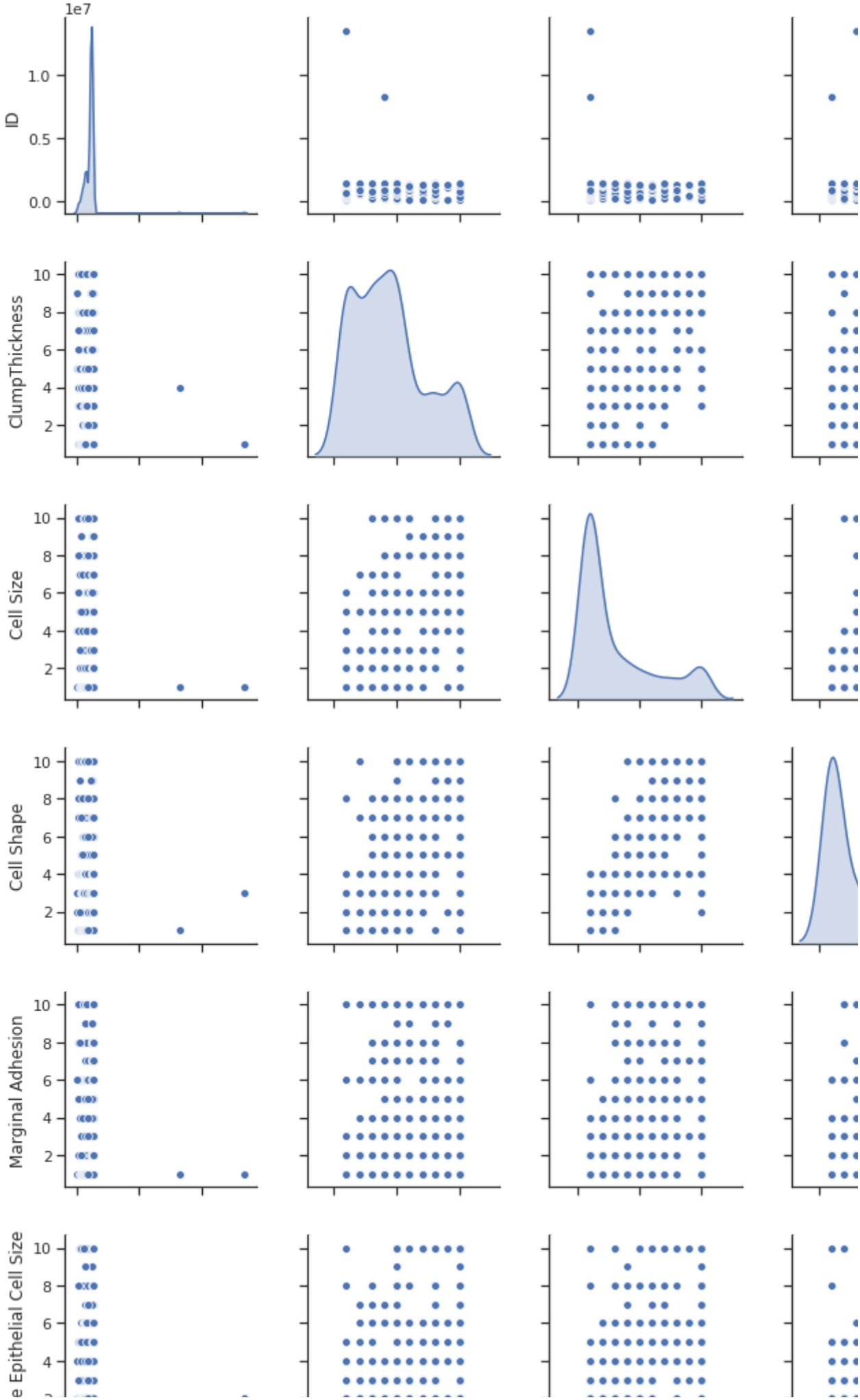
```
import numpy as np
import matplotlib as mp
import seaborn as sns
%matplotlib inline
sns.set(style="ticks")

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from scipy.stats import zscore
from sklearn.preprocessing import Imputer
from sklearn.metrics import accuracy_score
import seaborn as sns
import os
%matplotlib inline
from sklearn import metrics

from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt



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


from matplotlib import pyplot as plt
plt.figure(figsize=(25, 25))
ax = sns.heatmap(data.corr(), vmax=.8, square=True, fmt='.2f', annot=True, linecolor='white', lin
plt.title('Correlation')
plt.show()
```

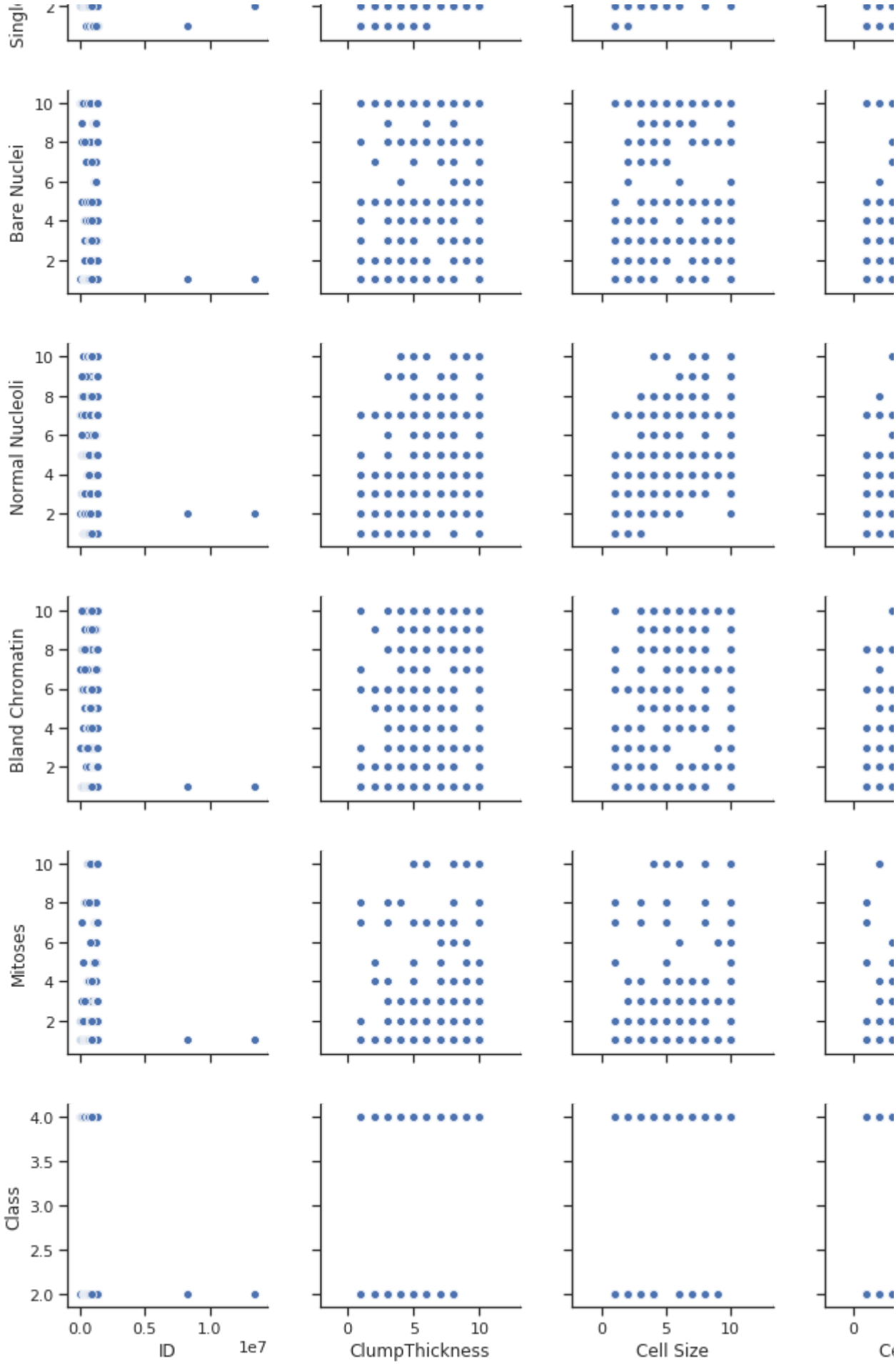

ID
ClumpThickness
Cell Size
Cell Shape
Marginal Adhesion
Single Epithelial Cell Size
Bare Nuclei
Normal Nucleoli
Bland Chromatin
1.00 -0.06 -0.04 -0.04 -0.06
-0.06 1.00 0.64 0.65 0.49
-0.04 0.64 1.00 0.91 0.71
-0.04 0.65 0.91 1.00 0.68
-0.06 0.49 0.71 0.68 1.00
-0.05 0.52 0.75 0.72 0.60
-0.09 0.59 0.69 0.71 0.67
-0.06 0.56 0.76 0.74 0.67
-0.05 0.54 0.72 0.72 0.60




```
#Let us check for pair plots

sns.pairplot(data, diag_kind ='kde')
```

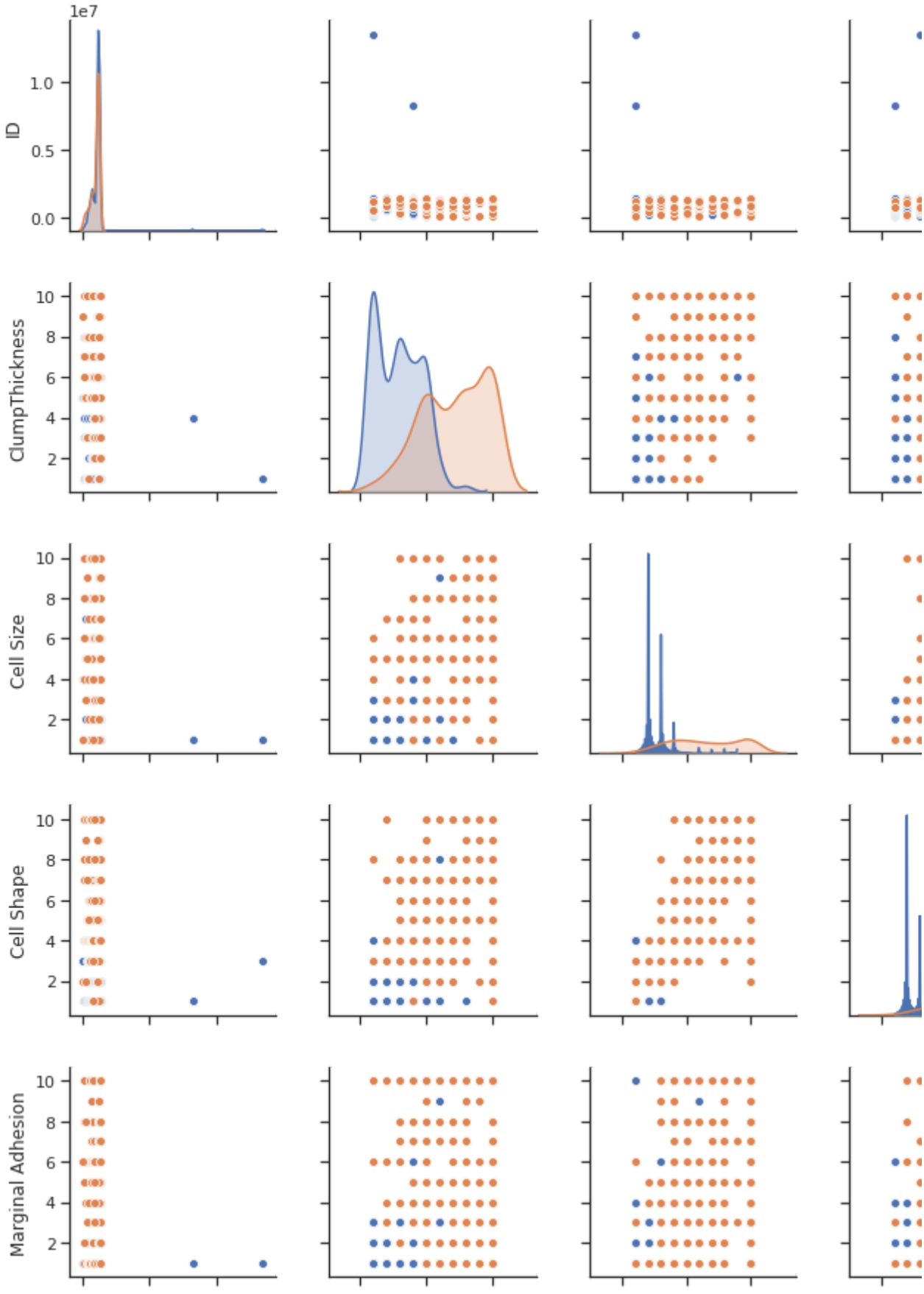
```
<seaborn.axisgrid.PairGrid at 0x7f1f22d9c2b0>
```

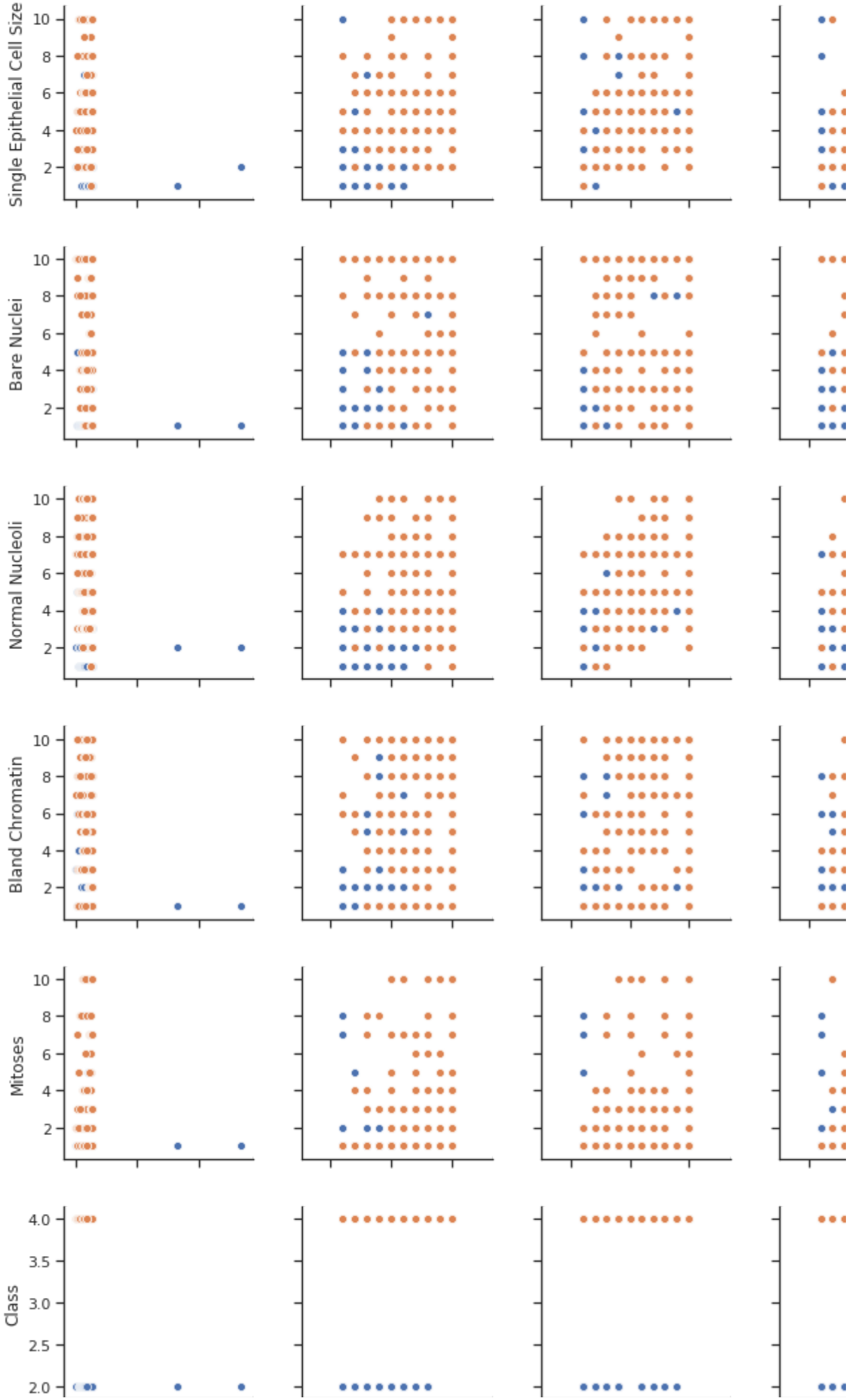


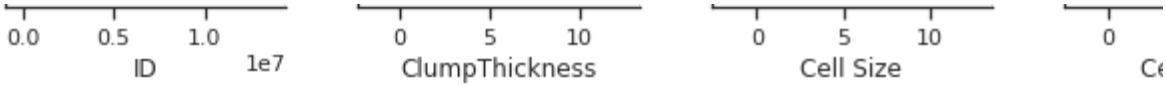

Bare Nuclei
Normal Nucleoli
Bland Chromatin
Mitoses
Class
ID
1e7
ClumpThickness
Cell Size

```
sns.pairplot(data , hue='Class' , diag_kind = 'kde')
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:447: RuntimeW
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:447: RuntimeW
  X = X[np.logical_and(X > clip[0], X < clip[1])] # won't work for two columns.
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:487: RuntimeW
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kdetools.py:34: Runt
  FAC1 = 2*(np.pi*bw/RANGE)**2
<seaborn.axisgrid.PairGrid at 0x7f1f2591ce10>
```




Single Epithelial Cell Size
Bare Nuclei
Normal Nucleoli
Bland Chromatin
Mitoses
Class



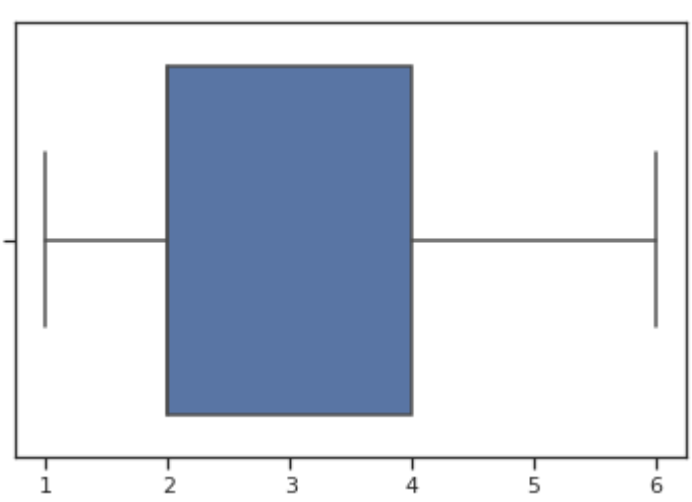

```
# Relationship between variables shows come correlation.
# Distribution of variables shows most of the values are concentrated on lower side, though range
# Between 1 to 10
```

Q5 Remove any unwanted columns or outliers, standard pre-processing step

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

```
data['Marginal Adhesion']=np.where(data['Marginal Adhesion']>8,4 ,data['Marginal Adhesion'])
```

```
plt = sns.boxplot(data[['Single Epithelial Cell Size']])
```

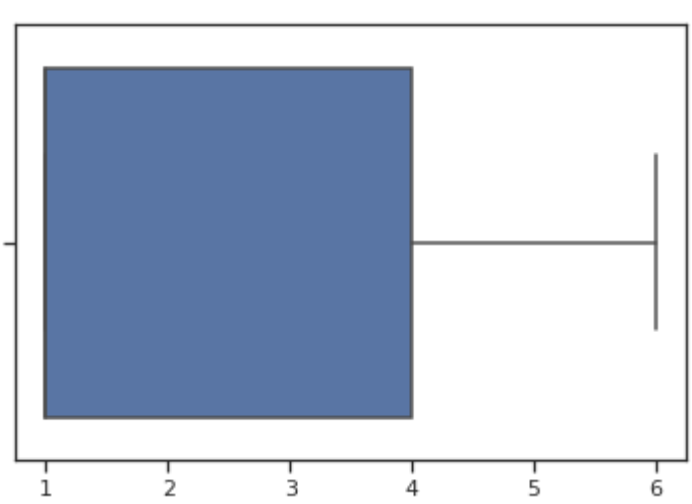



```
data['Single Epithelial Cell Size']=np.where(data['Single Epithelial Cell Size']>6,4 ,data['Singl
```

```
plt = sns.boxplot(data[['Normal Nucleoli']])
```

```
data['Normal Nucleoli']=np.where(data['Normal Nucleoli']>6,5 ,data['Normal Nucleoli'])
```

```
plt = sns.boxplot(data[['Bland Chromatin']])
```




```
data['Bland Chromatin']=np.where(data['Bland Chromatin']>6,5 ,data['Bland Chromatin'])
```

Q6 Create a covariance matrix for identifying Principal

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



import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler


from sklearn.decomposition import PCA
from scipy.stats import zscore
```

Q7 Identify eigen values and eigen vector

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


from sklearn.model_selection import train_test_split


X,y = data.iloc[:, 1:].values , data.iloc[:,0].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5 , random_state=0)

sc = StandardScaler()
```

```
X_train_std = sc.fit_transform(X_train)  # scale training and test data independently to prevent
X_test_std = sc.fit_transform(X_test)


cov_matrix = np.cov(X_train_std.T)

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

```
Covariance Matrix
%s [[1.00287356 0.61712893 0.63455658 0.48013973 0.53880998 0.56095073
  0.56244792 0.50761727 0.3247337  0.70326794]
 [0.61712893 1.00287356 0.89496652 0.73713988 0.73413458 0.65787247
  0.69830771 0.64581003 0.42924575 0.82453015]
 [0.63455658 0.89496652 1.00287356 0.7208573  0.71348043 0.66458122
  0.66002332 0.6501502  0.44279443 0.8193686 ]
 [0.48013973 0.73713988 0.7208573  1.00287356 0.64034269 0.65565412
  0.58873308 0.57384696 0.36226629 0.71860428]
 [0.53880998 0.73413458 0.71348043 0.64034269 1.00287356 0.66143739
  0.61451869 0.64013496 0.39529604 0.77607289]
 [0.56095073 0.65787247 0.66458122 0.65565412 0.66143739 1.00287356
  0.63649024 0.61830145 0.37442731 0.83143973]
 [0.56244792 0.69830771 0.66002332 0.58873308 0.61451869 0.63649024
  1.00287356 0.63475119 0.26169154 0.74812363]
 [0.50761727 0.64581003 0.6501502  0.57384696 0.64013496 0.61830145
  0.63475119 1.00287356 0.34244091 0.73390585]
 [0.3247337  0.42924575 0.44279443 0.36226629 0.39529604 0.37442731
  0.26169154 0.34244091 1.00287356 0.4112888 ]
 [0.70326794 0.82453015 0.8193686  0.71860428 0.77607289 0.83143973
  0.74812363 0.73390585 0.4112888  1.00287356]]
```

Q8 Find variance and cumulative variance by each eige

```
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 [[-0.28488528 -0.07250969  0.13208618 -0.0394129   0.85720613  0.27053069
  -0.15870294  0.19572552  0.140123    0.0588542 ]
 [-0.35170978  0.00880952  0.46062119  0.57803147 -0.08318062 -0.32591494
  -0.35504479 -0.07180592 -0.02337554 -0.2916016 ]
 [-0.34947583  0.04050158 -0.20658915 -0.65329236 -0.03075509 -0.45622182
```

```
print("Eigen Values:")
pd.DataFrame(eig_vals).transpose()
```

Eigen Values:

	0	1	2	3	4	5	6	7	
0	6.61478	0.811785	0.096973	0.105292	0.554373	0.262907	0.478395	0.331771	0.36281

Q9 Use PCA command from sklearn and find Principal C

Transform data to components formed

```
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 [ 65.95826434  74.05285452  79.58069622  84.35094167  8
  92.05340977  95.36161149  97.98314912  99.03305    100.        ]
```

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

```
# Step 3 (continued): Sort eigenvalues in descending order

# Make a set of (eigenvalue, eigenvector) pairs
eig_pairs = [(eig_vals[index], eig_vecs[:,index]) for index in range(len(eig_vals))]


# Sort the (eigenvalue, eigenvector) pairs from highest to lowest with respect to eigenvalue
eig_pairs.sort()
eig_pairs.reverse()

# Note: always form pair of eigen vector and values  first before sorting...

# Extract the descending ordered eigenvalues and eigenvectors
eigvalues_sort = [eig_pairs[index][0] for index in range(len(eig_vals))]
eigvectors_sort = [eig_pairs[index][1] for index in range(len(eig_vals))]



P_reduce = np.array(eigvectors_sort[0:10]).transpose()   # Selecting first few eigen vectors of a

Proj_train_data = np.dot(X_train_std,P_reduce)   # projecting training data onto the eight eigen

Proj_test_data = np.dot(X_test_std,P_reduce)    # projecting test data onto the eight eigen vecto



# complete matrix operation
print(y_train.shape)
X_train_std.shape, P_reduce.shape, Proj_train_data.shape
```

```
(349,)
((349, 10), (10, 10), (349, 10))
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(Proj_train_data, y_train)
#predicted_values = model.predict(Proj_test_data)

result = model.score(Proj_test_data, y_test)
print(result)
```

```
0.005714285714285714
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432: FutureWa
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:469: FutureWa
  "this warning.", FutureWarning)
```

```
from sklearn.decomposition import PCA
pca = PCA()
X = pca.fit_transform(X)
explained_variance = pca.explained_variance_ratio_
explained_variance
data=pd.DataFrame(X)
data.corr
```

```
<bound method DataFrame.corr of             0         1         2  ...         7
0    -3.880834  0.639743  2.012276  ... -0.648421 -0.035018  0.225753
1     4.941779 -4.783347 -0.050481  ...  0.645561  0.548477  1.388748
2    -4.069887 -0.744483  0.312226  ... -0.607645 -0.043604  0.188803
3     5.365617  3.353837 -1.343852  ...  1.365767 -0.890028  1.306544
4    -3.761819  0.244657  0.775440  ... -0.853708 -0.236329  0.203854
5    13.436348 -0.076738 -1.651973  ...  0.150049 -1.221093  0.428810
6    -0.658092 -7.801359 -0.927770  ...  0.167916 -0.296488  0.691221
7    -4.491635  0.078110 -0.888996  ... -1.125661  0.062999  0.088210
8    -4.788713  0.585288 -0.885423  ...  0.353080  0.135064 -0.081606
9    -3.969714  0.745376  0.844459  ...  0.491604 -0.139599  0.105189
10   -5.452859 -0.536107 -1.445803  ... -0.894614 -0.932273 -0.069874
11   -5.128507 -0.187569 -0.606930  ... -0.011685  0.066444 -0.041608
12    0.560618  0.319425  0.367987  ... -0.443245 -1.180960 -1.130017
13   -4.258939 -2.128708 -1.387824  ... -0.566870 -0.052190  0.151853
14    8.791722 -1.068462  0.965466  ...  0.702728 -0.505831  0.058130
15    2.697472  3.773455  0.557686  ... -1.193707  2.760421 -0.814208
16   -4.425048  0.386277  1.158842  ...  0.052508  0.040130  0.072394
17   -4.232564  0.352819  1.129390  ... -0.680518 -0.021861  0.168751
18   10.022139 -1.373633  2.514939  ... -0.288401  0.233014 -0.034545
19   -3.529104  0.926666  2.895162  ... -0.616324 -0.048176  0.282754
20    5.742921 -4.527038  2.188645  ... -0.005716  1.175276 -0.151065
21    7.251332 -0.036210  3.489563  ...  0.277193  1.899141  0.027860
22   -4.776778  0.099354  0.275956  ...  0.020412  0.053287  0.015393
23    1.430756  3.777401  2.488520  ... -1.725124 -0.851104 -1.017452
24   -5.287753 -0.507950 -1.519268  ... -0.776808  0.017610 -0.002252
25    2.649971 -3.190559  0.600869  ...  0.755147 -1.605450 -0.802797
26   -4.486550  0.430296  0.035038  ...  0.341701 -1.076324 -0.019434
27   -4.073318  0.673200  2.041728  ...  0.084605  0.026973  0.129396
28   -5.128507 -0.187569 -0.606930  ... -0.011685  0.066444 -0.041608
29   -4.783946  0.157239 -1.965593  ... -0.072656  0.258685 -0.128044
..         ...       ...       ...  ...       ...       ...       ...
669   9.974232  3.142544 -4.702706  ... -0.353282 -0.057241 -0.059836
670   9.258744 -0.815475 -5.415036  ...  0.754692 -0.101574 -0.108369
671  -3.893587  0.370914 -0.244854  ... -0.360114 -0.295667  0.162096
672  -4.936023 -0.221027 -0.636382  ... -0.744712  0.004453  0.054749
673  -2.745641  1.752148  1.116334  ...  1.432678  0.737936  0.199711
674  -5.480237 -0.474492 -1.489816  ... -0.043782  0.079601 -0.098610
675  -3.284370  1.317144  0.430451  ... -0.357315  0.277760  0.076421
676  -5.035849 -0.175356 -1.742430  ... -0.424732  0.138147 -0.065148
677  -4.265802  0.706658  2.071180  ...  0.817631  0.088965  0.033038
678  -5.672721 -0.441034 -1.460364  ...  0.689244  0.141593 -0.194967
679  -5.320992 -0.154111 -0.577478  ...  0.721341  0.128436 -0.137965
680  14.161842  1.850688  0.286315  ...  0.126038  0.141352  0.624766
681  11.927895 -0.295856 -3.845137  ...  0.501598 -0.959062  0.281084
682  -3.658748  0.695369  1.913843  ... -0.362831 -0.244708  0.291057
683  -5.672721 -0.441034 -1.460364  ...  0.689244  0.141593 -0.194967
684  -5.672721 -0.441034 -1.460364  ...  0.689244  0.141593 -0.194967
685  -5.672721 -0.441034 -1.460364  ...  0.689244  0.141593 -0.194967
686  -5.672721 -0.441034 -1.460364  ...  0.689244  0.141593 -0.194967
687  -4.332605  0.210608  0.079090  ...  0.591591 -0.366091  0.146003
688  -4.617532  0.419735  1.188294  ...  0.785535  0.102122 -0.023963
689  -4.741234  0.852913 -1.999268  ...  0.044788  0.153192 -0.096339
690  -5.201976 -0.549197 -1.814314  ...  0.516054 -0.072874 -0.159864
691   8.688438  3.254462 -3.970431  ...  0.236133 -0.413742 -0.341774
692  -4.969262  0.132812  0.305408  ...  0.753438  0.115279 -0.080964
693  -4.643708  0.284204  0.198969  ... -0.071654  0.054944  0.029483
694  -4.289749 -0.649410  0.297664  ...  0.976213  1.030262  0.063710
695  -5.320992 -0.154111 -0.577478  ...  0.721341  0.128436 -0.137965
696   7.736520  5.190401 -3.952797  ... -0.340117 -0.399499 -0.355230
697   5.370262  1.966895 -3.255694  ...  0.551618 -1.564430 -0.533469
```

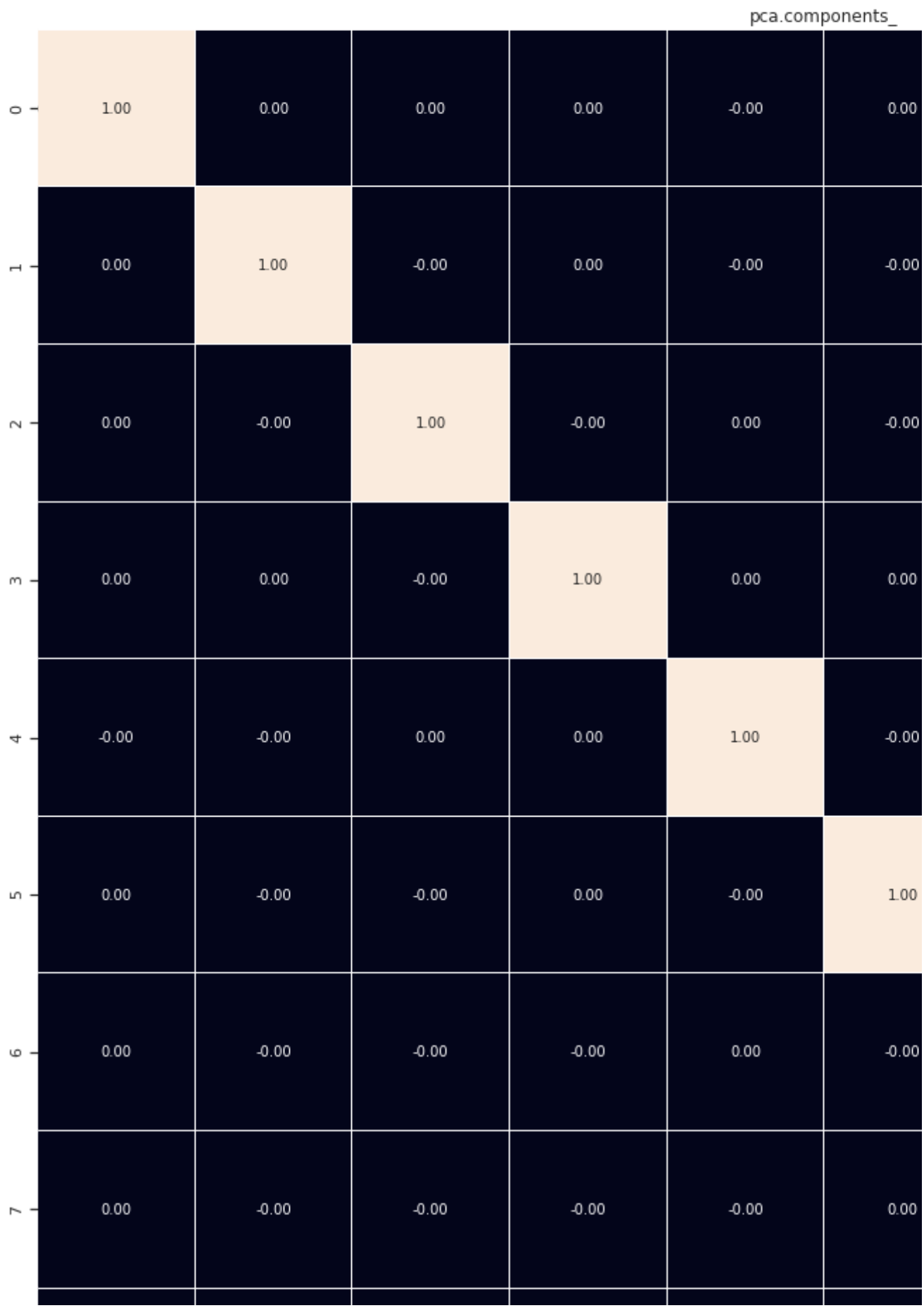
```
698   6.729750  1.617611 -3.748774  ... -0.645281 -0.220210 -0.434930

[699 rows x 10 columns]>
```

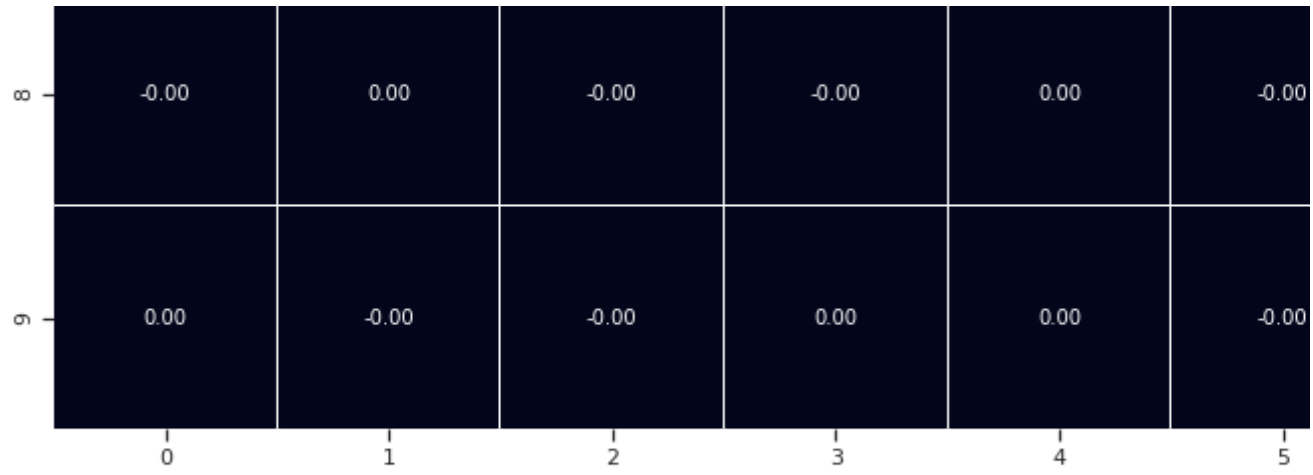
Q10 Find correlation between components and features

```
pca.components_
```

```
array([[ 0.35172969,  0.45533385,  0.44438748,  0.23537241,  0.16510645,
         0.51440668,  0.1924843 ,  0.2220863 ,  0.13306954,  0.14185382],
       [ 0.28692314,  0.35909853,  0.29913639, -0.0540813 ,  0.02815658,
        -0.81037881, -0.03345788,  0.05562673,  0.18484967, -0.02064492],
       [ 0.882886  , -0.31438319, -0.2526142 , -0.1769748 , -0.07346544,
         0.06572201, -0.02945203, -0.09843308, -0.07698632,  0.01848144],
       [-0.04310829, -0.11014488, -0.12608298, -0.17307526, -0.03884056,
         0.12103639, -0.09490807, -0.01057406,  0.95648883, -0.00303107],
       [ 0.06428528, -0.16125958, -0.39630321,  0.77870637,  0.15603631,
        -0.19724415,  0.12996567,  0.3327376 ,  0.12096019,  0.03055448],
       [ 0.0620316 ,  0.14213767,  0.07044057,  0.47747073, -0.11250477,
         0.0508752 , -0.32562854, -0.78310884,  0.06265867, -0.06740674],
       [-0.01350319,  0.52949036, -0.56713142, -0.16349987,  0.0854887 ,
        -0.00596283,  0.52780241, -0.28985006,  0.00951815,  0.03262593],
       [ 0.03209688,  0.43909552, -0.38094998, -0.08659497,  0.1178056 ,
         0.10496934, -0.73302615,  0.28558959, -0.09206511, -0.03580342],
       [-0.01315707, -0.17972888,  0.05854597, -0.1072337 ,  0.94988289,
        -0.03489971, -0.06199167, -0.20968906,  0.00165693,  0.03587492],
       [ 0.0570013 ,  0.03279471,  0.03346126,  0.01755148,  0.06762153,
         0.07705266,  0.09635712,  0.06530476,  0.01408973, -0.98488067]])
```

```
from matplotlib import pyplot as plt
plt.figure(figsize=(25, 25))
ax = sns.heatmap(data.corr(), vmax=.8, square=True, fmt='.2f', annot=True, linecolor='white', lin
plt.title('pca.components_')
plt.show()
```


pca.components_
0
1
2
3
4
5
6
7
1.00 0.00 0.00 0.00 -0.00 0.00
0.00 1.00 -0.00 0.00 -0.00 -0.00
0.00 -0.00 1.00 -0.00 0.00 -0.00
0.00 0.00 -0.00 1.00 0.00 0.00
-0.00 -0.00 0.00 0.00 1.00 -0.00
0.00 -0.00 -0.00 0.00 -0.00 1.00
0.00 -0.00 -0.00 -0.00 0.00 -0.00
0.00 -0.00 -0.00 -0.00 -0.00 0.00



Content Based Recommendation System - Optional (Q be graded)

Q11 Read the Dataset `movies_metadata.csv`

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

Q13 Lets drop the null values in `description` column

Q14 Keep the first occurance and drop duplicates of ea `title`

Q15 As we might have dropped a few rows with duplica step, just reset the index [make sure you are not adding to the dataframe while doing reset index]

Q16 Create cosine similarity matrix

Q17 Write a function with name `recommend` which takes argument and returns a list of 10 recommended title na based on the above 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]
```

1 cell hidden

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

Popularity Based Recommendation System

About Dataset

Anonymous Ratings on jokes.

1. Ratings are real values ranging from -10.00 to +10.00 (the value "99" corresponds to "null" = "not rated").
2. One row per user
3. The first column gives the number of jokes rated by that user. The next 100 columns give the ratings for jo

Q19 Read the dataset(jokes.csv)

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

```
data1 = pd.read_csv("/content/drive/My Drive/13oct/jokes.csv")
```

```
data1
```

0	74	-7.82	8.79	-9.66	-8.16	-7.52	-8.50	-9.85	4.17	-8.98	-4.7
1	100	4.08	-0.29	6.36	4.37	-2.38	-9.66	-0.73	-5.34	8.88	9.2
2	49	99.00	99.00	99.00	99.00	9.03	9.27	9.03	9.27	99.00	99.0
3	48	99.00	8.35	99.00	99.00	1.80	8.16	-2.82	6.21	99.00	1.8
4	91	8.50	4.61	-4.17	-5.39	1.36	1.60	7.04	4.61	-0.44	5.7
5	100	-6.17	-3.54	0.44	-8.50	-7.09	-4.32	-8.69	-0.87	-6.65	-1.8
6	47	99.00	99.00	99.00	99.00	8.59	-9.85	7.72	8.79	99.00	99.0
7	100	6.84	3.16	9.17	-6.21	-8.16	-1.70	9.27	1.41	-5.19	-4.4
8	100	-3.79	-3.54	-9.42	-6.89	-8.74	-0.29	-5.29	-8.93	-7.86	-1.6
9	72	3.01	5.15	5.15	3.01	6.41	5.15	8.93	2.52	3.01	8.1
10	36	-2.91	4.08	99.00	99.00	-5.73	99.00	2.48	-5.29	99.00	1.4
11	100	1.31	1.80	2.57	-2.38	0.73	0.73	-0.97	5.00	-7.23	-1.3
12	47	99.00	99.00	99.00	99.00	5.87	99.00	5.58	0.53	99.00	7.1
13	100	9.22	9.27	9.22	8.30	7.43	0.44	3.50	8.16	5.97	8.9
14	100	8.79	-5.78	6.02	3.69	7.77	-5.83	8.69	8.59	-5.92	7.5
15	100	-3.50	1.55	2.33	-4.13	4.22	-2.28	-2.96	-0.49	2.91	1.9
16	51	99.00	-9.27	99.00	99.00	-7.38	99.00	8.74	-6.31	99.00	2.3
17	100	3.16	7.62	3.79	8.25	4.22	7.62	2.43	0.97	0.53	0.8
18	49	4.22	3.64	99.00	99.00	2.52	99.00	4.13	-5.19	99.00	7.9
19	53	99.00	7.62	99.00	99.00	-8.64	2.43	8.93	-6.60	99.00	-9.4
20	55	2.57	-0.73	99.00	99.00	2.57	99.00	-4.22	2.67	99.00	-1.3
21	50	7.28	5.39	99.00	99.00	-4.22	99.00	8.93	3.50	99.00	6.1
22	72	2.72	4.17	6.50	-3.54	-3.30	2.86	7.04	4.95	6.17	8.9
23	100	2.09	-7.57	4.17	-8.40	-6.31	5.34	-5.19	-5.05	-8.20	2.2
24	74	-3.93	-1.17	2.28	-0.44	-2.28	2.38	-3.93	4.51	-0.34	7.7
25	67	99.00	5.49	3.45	99.00	0.15	4.76	0.10	4.13	8.01	4.7
26	60	99.00	5.83	99.00	99.00	5.34	99.00	-4.47	2.18	-3.98	-3.2
27	72	0.49	0.15	0.15	-9.56	-9.76	-9.61	0.29	1.65	-4.61	-4.7
28	54	99.00	99.00	99.00	99.00	-1.70	99.00	8.83	-1.89	99.00	99.0
29	46	99.00	4.08	99.00	99.00	5.87	99.00	0.44	2.28	99.00	99.0
...	...	...	...	...	...	...	...	...	...	...	..
24953	59	99.00	99.00	99.00	99.00	-7.62	3.59	2.86	0.73	99.00	3.3
24954	68	99.00	99.00	99.00	2.18	3.54	5.53	8.59	-5.63	4.08	1.7
24955	73	-2.82	-2.14	-3.30	-7.72	-1.02	-6.21	2.23	-8.20	-8.54	6.3

24956	73	3.54	8.98	6.80	-1.21	8.11	-0.10	3.74	2.18	-0.19	-0.5
24957	36	99.00	99.00	99.00	99.00	3.79	99.00	1.55	2.38	99.00	99.0
24958	100	-2.91	-1.89	-2.86	-1.99	-7.52	-4.66	-7.52	6.02	-3.83	-4.4
24959	49	99.00	-7.48	99.00	99.00	-0.29	-1.02	6.70	8.01	99.00	5.1
24960	51	99.00	99.00	99.00	99.00	0.39	99.00	-7.09	7.18	99.00	99.0
24961	36	99.00	99.00	99.00	99.00	-2.52	99.00	1.07	-1.99	99.00	99.0
24962	100	4.85	3.83	4.13	-5.53	2.14	-4.66	0.10	-1.75	0.44	0.3
24963	69	8.16	7.52	7.52	8.16	1.80	8.45	7.57	7.57	8.16	7.7
24964	72	-2.09	1.50	0.24	-8.06	1.75	1.41	-6.36	-7.72	-6.17	1.3
24965	38	99.00	99.00	99.00	99.00	-6.70	99.00	-0.87	-8.74	99.00	99.0
24966	84	4.13	9.13	7.23	4.08	2.96	5.29	6.07	6.60	5.44	7.4
24967	100	-0.29	-0.29	-0.29	-0.29	-1.84	0.10	-3.01	-0.29	0.49	-0.2
24968	72	8.88	9.32	8.20	3.40	0.44	7.86	2.23	1.26	5.87	8.9
24969	72	6.41	1.12	-5.15	-7.18	-7.38	1.07	2.28	-2.09	-6.31	-6.8
24970	47	99.00	99.00	99.00	99.00	-0.10	-4.22	1.46	8.83	99.00	6.7
24971	100	4.17	4.17	9.03	-0.29	2.04	3.79	7.86	-7.86	4.17	5.7
24972	48	99.00	99.00	99.00	99.00	-6.02	-8.20	-3.93	2.52	99.00	7.8
24973	62	1.80	-9.42	-8.20	99.00	-8.54	-0.05	-1.12	-8.45	99.00	-8.1
24974	51	99.00	-9.22	99.00	99.00	5.97	8.45	-3.30	5.49	99.00	2.3
24975	52	99.00	8.20	99.00	99.00	8.79	8.40	9.08	2.09	99.00	99.0
24976	78	1.36	1.75	2.38	1.84	4.66	5.00	-0.19	-0.24	9.13	1.0
24977	36	99.00	99.00	99.00	99.00	7.67	99.00	1.02	-3.74	99.00	99.0
24978	100	0.44	7.43	9.08	2.33	3.20	6.75	-8.79	-0.53	-8.74	7.2
24979	91	9.13	-8.16	8.59	9.08	0.87	-8.93	-3.50	5.78	-8.11	4.9
24980	39	99.00	99.00	99.00	99.00	-7.77	99.00	6.70	-6.75	99.00	99.0
24981	37	99.00	99.00	99.00	99.00	-9.71	99.00	4.56	-8.30	99.00	99.0
24982	72	2.43	2.67	-3.98	4.27	-2.28	7.33	2.33	4.56	6.75	4.6

24983 rows × 101 columns

Q20 Consider `ratings` named dataframe with only first

```
ratingdata=data1.iloc[0:200,1:]
ratingdata
```

	Joke1	Joke2	Joke3	Joke4	Joke5	Joke6	Joke7	Joke8	Joke9	Joke10	Joke11	J
0	-7.82	8.79	-9.66	-8.16	-7.52	-8.50	-9.85	4.17	-8.98	-4.76	-8.50	
1	4.08	-0.29	6.36	4.37	-2.38	-9.66	-0.73	-5.34	8.88	9.22	6.75	
2	99.00	99.00	99.00	99.00	9.03	9.27	9.03	9.27	99.00	99.00	7.33	
3	99.00	8.35	99.00	99.00	1.80	8.16	-2.82	6.21	99.00	1.84	7.33	
4	8.50	4.61	-4.17	-5.39	1.36	1.60	7.04	4.61	-0.44	5.73	8.25	
5	-6.17	-3.54	0.44	-8.50	-7.09	-4.32	-8.69	-0.87	-6.65	-1.80	-6.80	
6	99.00	99.00	99.00	99.00	8.59	-9.85	7.72	8.79	99.00	99.00	4.27	
7	6.84	3.16	9.17	-6.21	-8.16	-1.70	9.27	1.41	-5.19	-4.42	8.20	
8	-3.79	-3.54	-9.42	-6.89	-8.74	-0.29	-5.29	-8.93	-7.86	-1.60	-2.91	
9	3.01	5.15	5.15	3.01	6.41	5.15	8.93	2.52	3.01	8.16	5.53	
10	-2.91	4.08	99.00	99.00	-5.73	99.00	2.48	-5.29	99.00	1.46	99.00	
11	1.31	1.80	2.57	-2.38	0.73	0.73	-0.97	5.00	-7.23	-1.36	3.83	
12	99.00	99.00	99.00	99.00	5.87	99.00	5.58	0.53	99.00	7.14	7.04	
13	9.22	9.27	9.22	8.30	7.43	0.44	3.50	8.16	5.97	8.98	3.74	
14	8.79	-5.78	6.02	3.69	7.77	-5.83	8.69	8.59	-5.92	7.52	-4.85	
15	-3.50	1.55	2.33	-4.13	4.22	-2.28	-2.96	-0.49	2.91	1.99	-1.99	
16	99.00	-9.27	99.00	99.00	-7.38	99.00	8.74	-6.31	99.00	2.33	99.00	
17	3.16	7.62	3.79	8.25	4.22	7.62	2.43	0.97	0.53	0.83	3.50	
18	4.22	3.64	99.00	99.00	2.52	99.00	4.13	-5.19	99.00	7.91	99.00	
19	99.00	7.62	99.00	99.00	-8.64	2.43	8.93	-6.60	99.00	-9.47	-2.77	
20	2.57	-0.73	99.00	99.00	2.57	99.00	-4.22	2.67	99.00	-1.31	0.73	
21	7.28	5.39	99.00	99.00	-4.22	99.00	8.93	3.50	99.00	6.12	0.87	
22	2.72	4.17	6.50	-3.54	-3.30	2.86	7.04	4.95	6.17	8.93	8.88	
23	2.09	-7.57	4.17	-8.40	-6.31	5.34	-5.19	-5.05	-8.20	2.28	1.17	
24	-3.93	-1.17	2.28	-0.44	-2.28	2.38	-3.93	4.51	-0.34	7.77	-2.48	
25	99.00	5.49	3.45	99.00	0.15	4.76	0.10	4.13	8.01	4.76	7.33	
26	99.00	5.83	99.00	99.00	5.34	99.00	-4.47	2.18	-3.98	-3.25	-3.40	
27	0.49	0.15	0.15	-9.56	-9.76	-9.61	0.29	1.65	-4.61	-4.71	-5.78	
28	99.00	99.00	99.00	99.00	-1.70	99.00	8.83	-1.89	99.00	99.00	-5.39	
29	99.00	4.08	99.00	99.00	5.87	99.00	0.44	2.28	99.00	99.00	99.00	
...	...	...	...	...	...	...	...	...	...	...	...	
170	99.00	99.00	99.00	99.00	4.27	99.00	4.81	-6.21	99.00	99.00	99.00	
171	4.56	3.74	2.67	99.00	3.40	0.63	5.34	0.44	5.34	4.08	0.83	