

Description on the **Cancer_Final.py** code

Location of the **plot folder** needs to be given, to save plot output. Code has been created using **spyder**

Project Folder structure:

~F- Drive

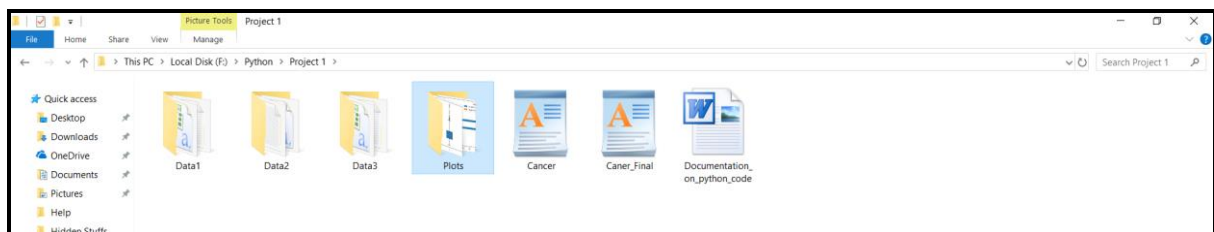
--Project 1

--Data1 (<https://archive.ics.uci.edu/ml/datasets/Breast+Cancer>)

--Data2 (Breast Cancer Wisconsin (Original) Data Set)

--Data3 (Breast Cancer Wisconsin (Diagnostic) Data Set)

--Plots



The complete code has been worked out using custom defined functions:

1. **def OnCreateDF:** Function defined for creating dataframe, function takes two arguments: 1st- location of the interested file, 2nd: list of column names

```
48 def OnCreateDF( fname, sep = ',', cols = None, index = None ):
49
50     data = pd.read_csv( fname, sep = sep, header = None, names = cols, index_col = index )
51     return data
```

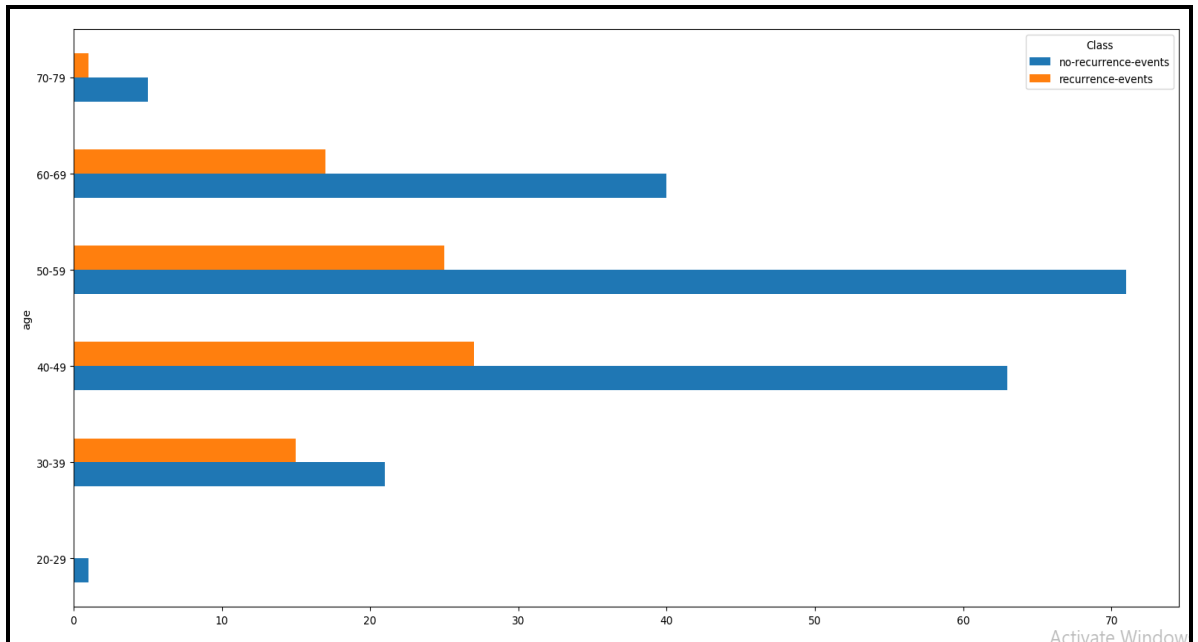
2. **def OnPlotCat:** Function defined for creating plot for **categorical/ range attributes**, function takes three arguments: 1st- dataframe, 2nd- categorical/range column name, 3rd- class (dependent) column name

```
54 def OnPlotCat( data, col1, col2, sav = False, name = 'out' ):
55
56     datav = data.groupby( col2 )[col1].value_counts().unstack(0)
57     ax = datav.plot.barh()
58     if sav:
59         fig = ax.get_figure()
60         fig.savefig( PLOTS + name + '.png', bbox_inches = 'tight', dpi = 1000 )
61
```

Output

I have use groupby function followed by bar chart plot for categorical attributes representation and analysis.

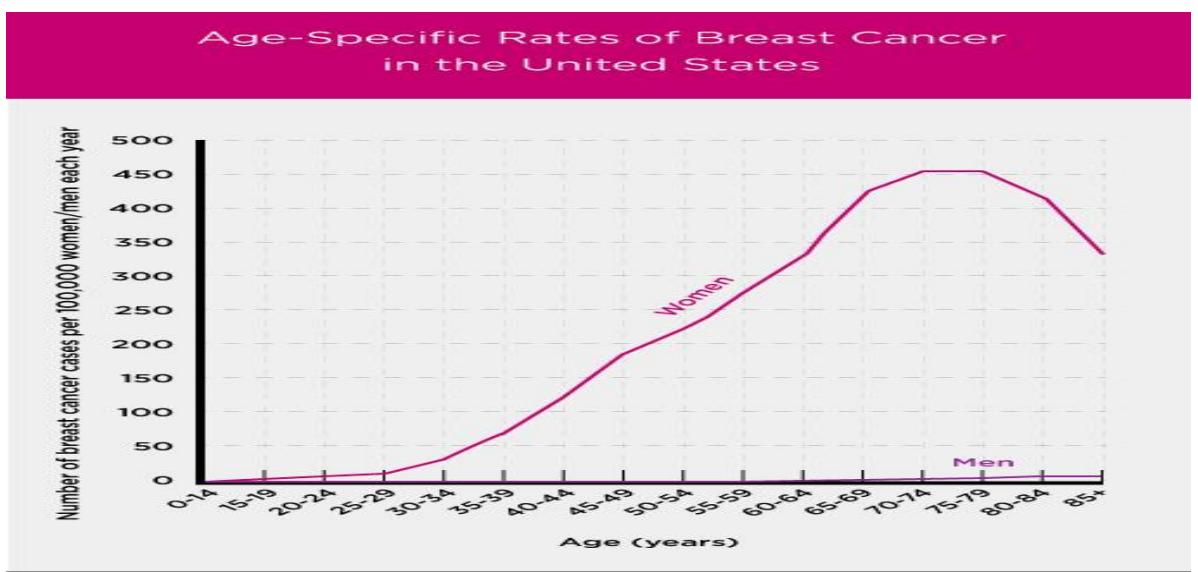
Age independent of menstruation information (start and end of menstrual cycle) is not an important predictor of breast cancer risk. As, we can see all age range has high proportion of **non-recurrence-event**. Also my pathway/literature analysis document (Pathway_Analysis.doc) points on the same thing.



At the same time, patient with the **age older than 45** and **late onset of menopause** have higher risk of breast and ovarian cancer, **due to more exposure of estrogen**.

From our barchart output, we can see patient in the range 40+ have higher rate of **recurrence-event** compared to lower age group.

From research article:



3. **def OnPlotBoxClass:** Function defined for creating boxplot for **feature ~ class** continuous features, function takes two arguments: 1st- dataframe, 2nd- class (dependent) column name.

```
92
93 def OnPlotBoxClass( data, cname = 'Class' ):
94
95     boxplot = data.boxplot( by = cname )
96     plt.show()
97
```

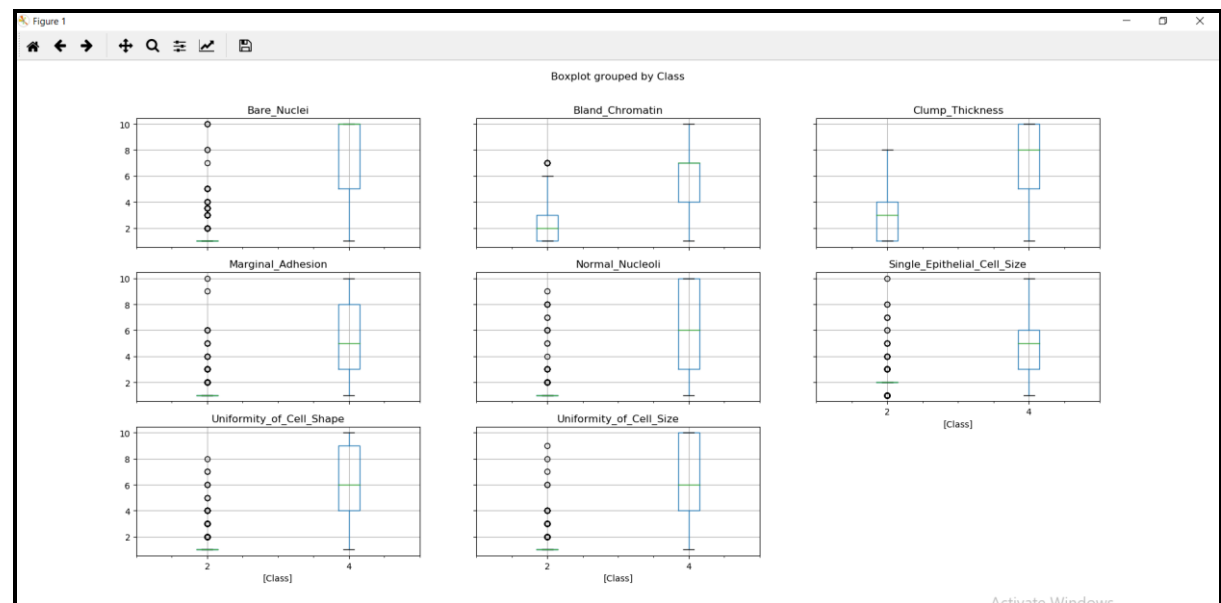
Output

Dataset2:

This so beautifully explains: **high values** of Clump thickness, Cell shape and size uniformity, Epithelial cell size box plot is **prognostic/ deterministic factor of Malignant tumor**. Also, we can see the above-mentioned features upper whisker and upper quartile touching each other, signifying point concentrating at the higher value. Also, my pathway/literature analysis document (**Pathway_Analysis.doc**) points on the same thing.

Contradiction: loss of tumor adhesion is a feature of malignant cancer, since loss of adhesion is required for tumor metastasis.

Low value should have been the prognostic factor for malignant tumor, while high value of adhesion should have been prognostic factor of benign tumor.



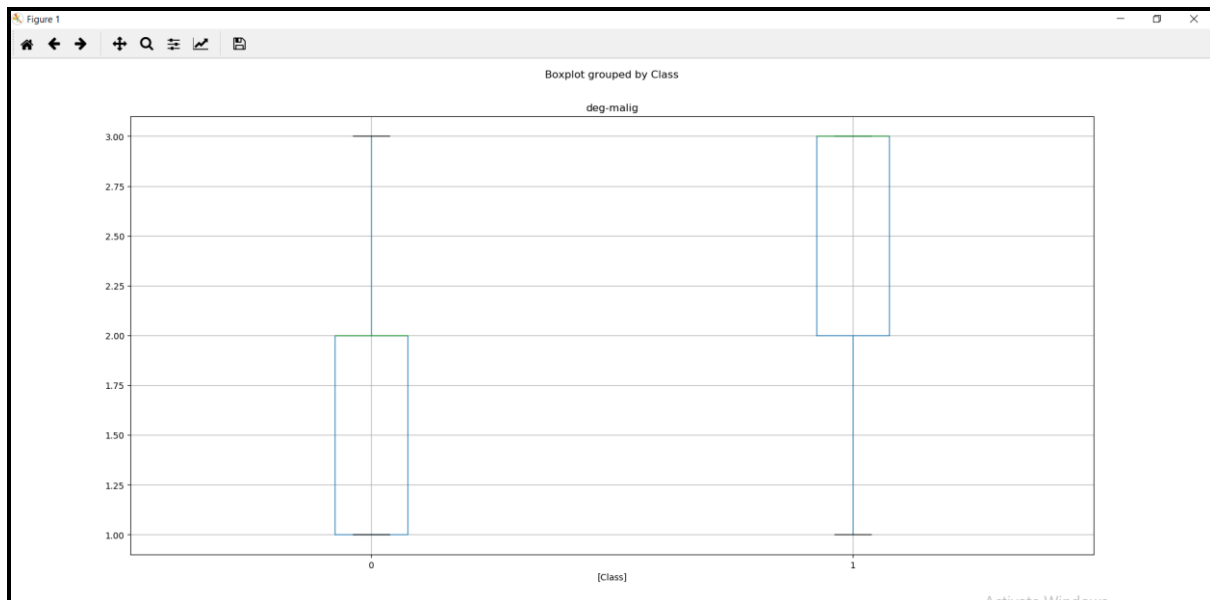
Class: 2- benign; 4- Malignant

Dataset 1:

Histology grade (deg- malig): Tumor grading deals with understanding of the tumor cell shape, spread, stage, abnormality, undifferentiated (lack normal tissue structures).

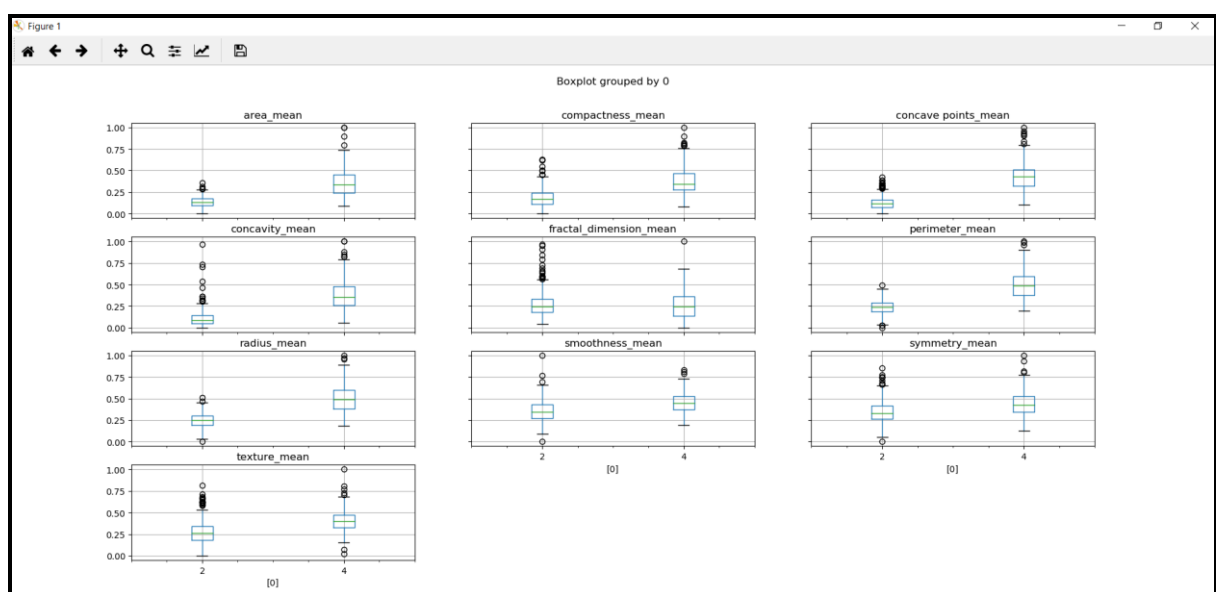
Grade 1 & 2- Normal growing cell; Grade 3 & 4- Rapidly growing and spreading cell.

High values of tumor grading (deg-malig) box plot are prognostic/ deterministic factor of recurrence event (can be linked with recurrence event based on biology). Also we can see the above mentioned features upper whisker and upper quartile touching each other, signifying point concentrating at the higher value. Also my pathway/literature analysis document (**Pathway_Analysis.doc**) points on the same thing.



Dataset 3

High values of texture, area, symmetry worse box plot are prognostic/ deterministic factor of malignant tumor. Also my pathway/literature analysis document (**Pathway_Analysis.doc**) points on the same thing.



4. **def OnPlotBox:** Function defined for creating plot for **categorical/ range attributes**, function takes three arguments: 1st- dataframe, 2nd- numeric column name

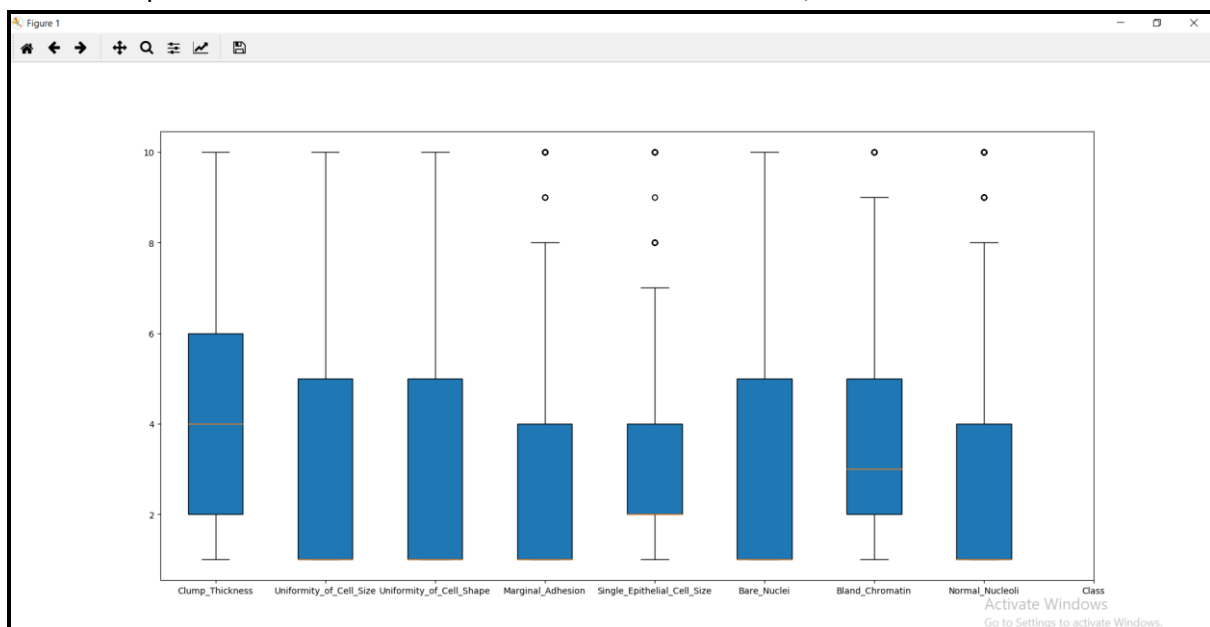
```

Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
F:\Python\Project 1
Editor - F:\Python\Project 1\Caner_Final2.py
Caner_Final.py X Caner_Final2.py X
89 """
90 OnPlotBox is used to create a Box Plot. The function creates a box plot of multiple numeric variables.
91
92 data = The DataFrame.
93 cols = List of columns names that will be represented by the plot.
94
95 """
96 def OnPlotBox( data, cols, sav = False, name = 'out2' ):
97
98     vect = []
99     for c in data.columns:
100         if c not in ['Class']:
101             vect.append( data[c] )
102
103     plt.boxplot( vect, patch_artist = True )
104     plt.xticks( list(range(1, len(cols)+1)), cols )
105     if sav:
106         plt.savefig( PLOTS + name + '.png', bbox_inches = 'tight', dpi = 1000 )
107

```

Output

Help us to understand how each feature is distributed, number of outliers etc.



5. **def OnViolinPlot:** OnViolinPlot is used to create a Violin Plot of the given data. Function takes two arguments, 1st- data = The DataFrame, 2nd - class_col = Either "NaN" or the dependent variable column as a List.

```

125 """
126 OnViolinPlot is used to create a Violin Plot of the given data.
127
128 data = The DataFrame.
129 class_col = Either "NaN" or the dependent variable column as a List.
130
131 """
132 def OnViolinPlot(data, class_col = 'NaN'):
133
134     if class_col != 'NaN':
135         data['Class'] = class_col
136     data = pd.melt( data, id_vars = 'Class', var_name = 'features', value_name = 'value' )
137     plt.figure( figsize = (10,10) )
138     sns.violinplot( x = 'features', y = 'value', hue = 'Class', data = data, split = True, inner = 'quart' )
139     plt.xticks( rotation = 90 )
140

```

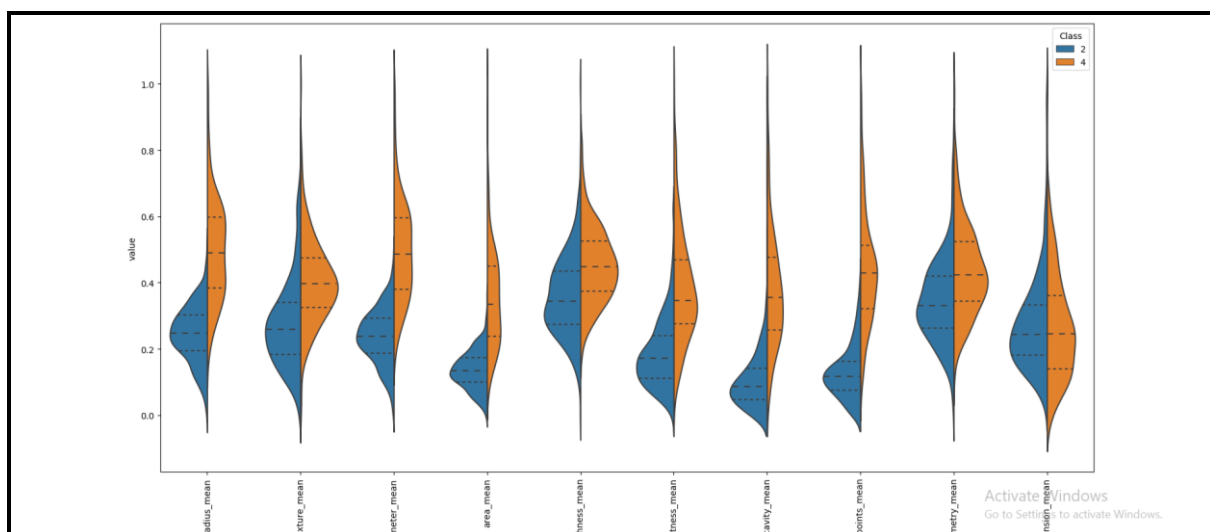
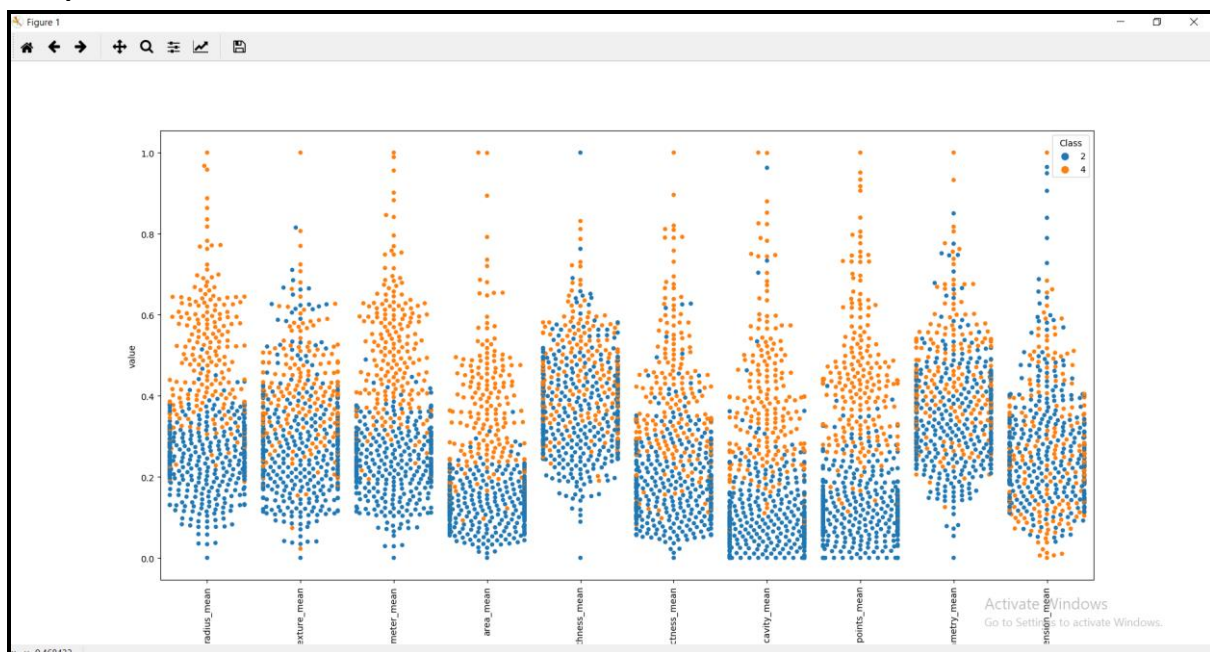
6. **def OnSwarmPlot:** OnSwarmPlot is used to create a Swarm plot. Swarm Plots are helpful to understand how finely values of independent variables are distributed among the categories of dependent variable. Function takes two arguments, 1st- data = The DataFrame, 2nd - class_col = Either "NaN" or the dependent variable column as a List.

```

144
145 """
146 OnSwarmPlot is used to create a Swarm plot. Swarm Plots are helpful to understand how finely values of
147 independent variables are distributed among the categories of dependent variable.
148
149 data = The DataFrame.
150 class_col = Either "NaN" or the dependent variable column as a List.
151
152 """
153 def OnSwarmPlot(data, class_col = 'NaN'):
154
155     if class_col != 'NaN':
156         data['Class'] = class_col
157     data = pd.melt( data, id_vars = 'Class', var_name = 'features', value_name = 'value' )
158     plt.figure( figsize = (10,10) )
159
160     sns.swarmplot( x = 'features', y = 'value', hue = 'Class', data = data )
161     plt.xticks( rotation = 90 )
162

```

Output



From above visualization especially Violin and Swamp Plot, it can be seen that **symmetry, fractal dimension, smoothness is not very cleanly distributed**. While perimeter, area, radius, concave points are **clearly distributed**. Hence, we can think of dropping the feature if required.

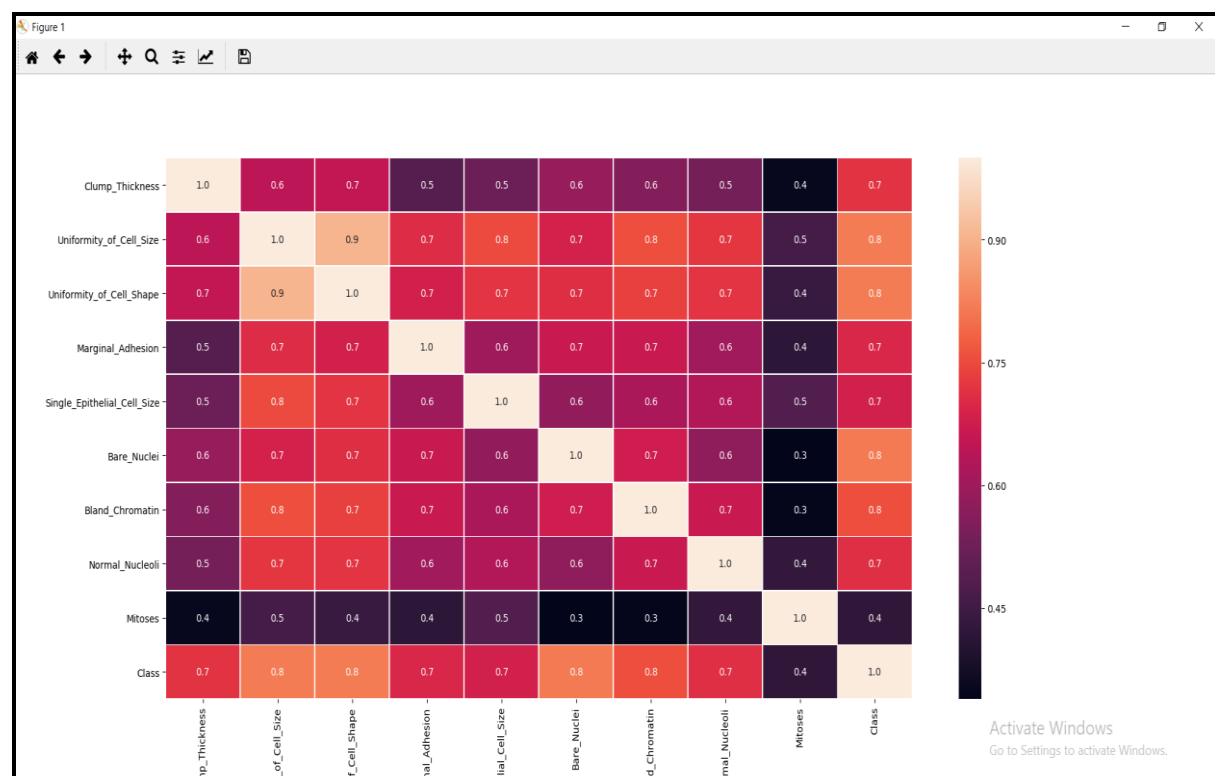
I ran the model without dropping the variable and with dropping the variable. I was not able to see any difference in the accuracy of the models.

7. **def OnPlotHeatMap:** OnPlotHeatMap creates a Heat Map. **The plot can be used for understanding the corelation among the variables that can help in Feature Selection.** Function takes two arguments, 1st- data = The DataFrame, 2nd - class_col = Either "NaN" or the dependent variable column as a List.

```
164 """
165 OnPlotHeatMap creates a Heat Map. The plot can be used for understanding the corelation among the
166 variables that can help in Feature Selection.
167
168 data = The DataFrame.
169 class_col = Either "NaN" or the dependent variable column as a List.
170 """
171
172 def OnPlotHeatMap(data):
173
174     f, ax = plt.subplots( figsize = (18,18) )
175     sns.heatmap( data = data.corr(), annot = True, linewidths = 0.5, fmt = '.1f' )
```

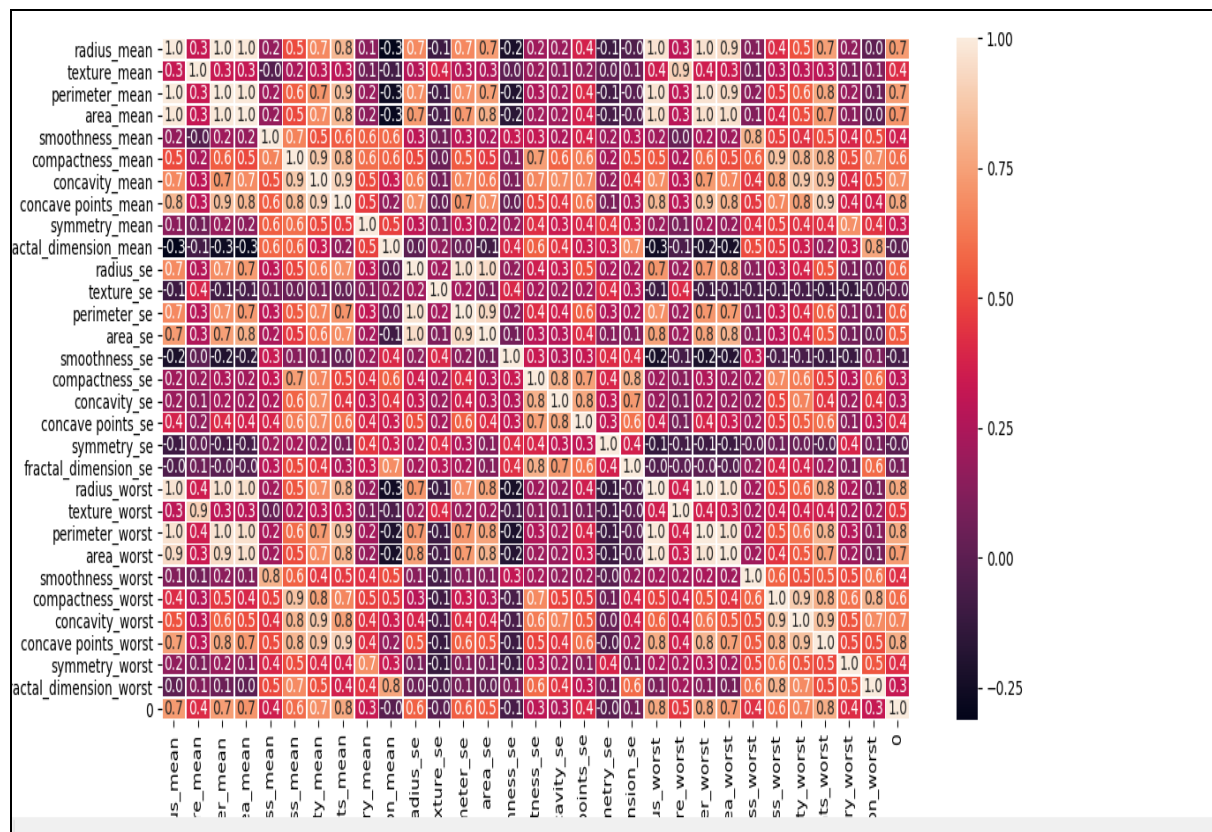
Output

Dataset 2:



From above it can be seen that **all the features except mitosis are strongly correlated.**
Contradiction: Mitosis is a feature of malignant cancer (from molecular biology), hence class and mitosis must have high correlation.

Dataset3



From above it can be seen that **radius, perimeter, area is strongly correlated with class variable, as radius, perimeter, and area are important factor in determining the morphology of cancer tissue and cell.**

Contradiction: texture is a feature of malignant cancer (from molecular biology), hence class and texture must have high correlation.

8. **def OnLabelEncode:** OnLabelEncode is used to Label encode Categorical Features. Function takes two arguments, 1st- data = The DataFrame, 2nd - class_col = Either "NaN" or the dependent variable column as a List.

```

186
187 OnLabelEncode is used to Label encode Categorical Features.
188
189 data = The DataFrame.
190 class_col = Either "all" or List of columns that must be Label Encoded.
191
192 """
193 def OnLabelEncode( data, cols = 'all' ):
194
195     lblenc = LBLENC();
196
197     if cols == 'all':
198         for c in data.columns:
199             data[c] = lblenc.fit_transform( data[c] )
200         return data
201     else:
202         for i in cols:
203             data.iloc[:, i] = lblenc.fit_transform( data.iloc[:, i] )
204         return data
205

```


9. **def OnSetDummyVars:** OnSetDummyVars is used to create Dummy Variables of Label Encoded Categorical Features. Function takes two arguments, 1st- data = The DataFrame, 2nd - class_col = Either "NaN" or the dependent variable column as a List.

Points:

1. Label Encode and Dummy Encode the categorical features. The second parameter of the OnDummyEncode()
2. is a List of indices of the columns that must be Dummified starts with 1 to ensure that the dependent variable ie. "Class" is not dummified.

```
208 OnSetDummyVars is used to create Dummy Variables of Label Encoded Categorical Features.
209
210 data = The DataFrame.
211 class_col = Either "all" or List of columns that must be converted to Dummy Variables.
212
213 """
214 def OnSetDummyVars( data, cols = 'all' ):
215
216     if cols == 'all':
217         for c in data.columns:
218             data = pd.get_dummies( data, prefix = [ c ], columns = [ c ] )
219
220     return data
221
222     else:
223         col_names = [ data.columns[i] for i in cols ]
224         for i in col_names:
225             data = pd.get_dummies( data, prefix = [ i ], columns = [ i ] )
226
227     return data
228
229
```

10. **def OnReplaceValue:** OnReplaceValue function is used to handel missing attributes in the dataset either Categorical or Numeric.

Input Parameters:

data = The DataFrame.

nvalue = String that represents missing values in the dataset.

myval = Custom value to replace the missing value with. Must be 'NaN' if not required.

cols = Either 'all' or the List of columns that must be processed.

dtype = Must be 'num' if given columns represent Numeric features or 'cat' if they represent Categorical Features.

strat = Represents the replacement strategy for missing values like mean, max, min, etc.

```
Spyder (Python 3.6)
File Edit Search Source Run Debug Cnsoles Projects Tools View Help
Editor - C:\Users\10646237\Desktop\va\cancer\Cancer_Final2.py
C:\Users\10646237\Desktop\va\cancer\Cancer_Final2.py
239 dtype = Must be 'num' if given columns represent Numeric features or 'cat' if they represent
240 Categorical Features.
241 strat = Represents the replacement strategy for missing values like mean, max, min, etc.
242
243 """
244 def OnReplaceValue( data, nvalue = '?', myval = 'NaN', cols = 'all', dtype = 'num', strat = 'mean' ):
245
246     if dtype == 'cat':
247         if cols == 'all':
248             for c in data.columns:
249                 if myval == 'NaN':
250                     data[c] = data[c].replace( nvalue, data[c].value_counts().idxmax() )
251                 else:
252                     data[c] = data[c].replace( nvalue, myval )
253
254         return data
255     else:
256         for i in cols:
257             if myval == 'NaN':
258                 data.iloc[:, i] = data.iloc[:, i].replace( nvalue, data.iloc[:, i].value_counts().idxmax() )
259             else:
260                 data[c] = data[c].replace( nvalue, myval )
261
262         return data
263
264     elif dtype == 'num':
265         if myval == 'NaN':
266             if cols == 'all':
```

Replacing strategy in dataset2

```
488
489 # Replace missing values. Notice that we replace the missing values that are represented by "?" by a custom
490 # number NaN. This is because firstly all the independent variable columns in the dataset are numeric.
491 # Second, all the columns are ranged between 1 - 10.
492 # Third, because the missing value is represented by "?" the columns containing missing values are
493 # of type String. This prevents us from using standard missing value replacement strategies like mean, max.
494 # Hence we first replace the missing values with NaN which is numeric.
495 # Next we convert the columns to type numeric. Next we replace them using standard methods.
496
497 data2 = OnReplaceValue( data2, myval = np.nan )
498 data2[data2.columns[5]] = data2[data2.columns[5]].apply(pd.to_numeric)
499 data2 = OnReplaceValue( data2, nvalue = np.nan, cols = [5] )
500
```

11. **def OnMinMaxScale:** OnMinMaxScale function is used to scale a DataFrame using Normalization.

Input Parameters:

data = The DataFrame.

```
296
297
298 """
299 OnMinMaxScale is used to scale a DataFrame using Normalization.
300
301 data = The DataFrame.
302
303 """
304 def OnMinMaxScale( data ):
305     mmscl = MMSCL()
306     return pd.DataFrame( mmscl.fit_transform(data) )
307
308
309
```

12. **def OnStdScale:** OnStdScale function is used to scale a DataFrame using Normalization.

Input Parameters:

data = The DataFrame.

```
310 """
311 OnStdScale is used to scale a DataFrame using Standardisation.
312
313 data = The DataFrame.
314
315 """
316 def OnStdScale( data ):
317     sscl = SSCL()
318     return pd.DataFrame( sscl.fit_transform(data) )
319
320
321
```

13. **def OnSplitData:** OnSplitData is used to split the dataset into Train/Test sets possessing equal proportion of all the categories of the dependent variable

Input Parameters:

data = The DataFrame.

t_size = Size of test set.

Strategy in the function:

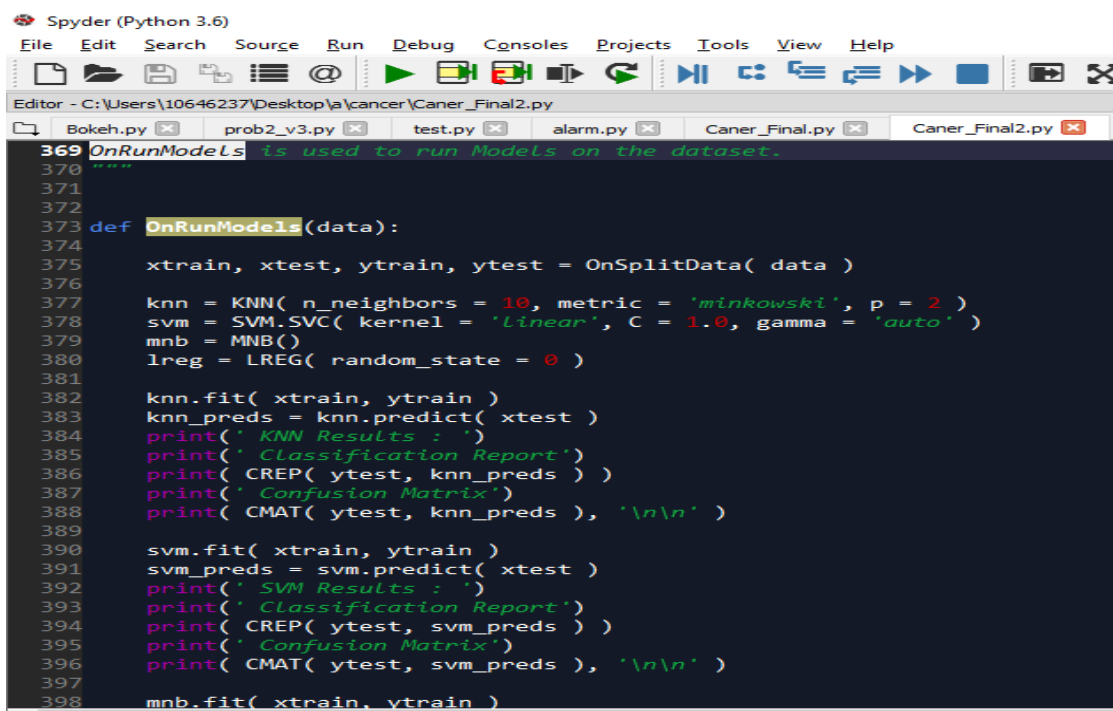
1. Assuming the dependent variable has two categories, we first create two independent dataframes each containing data of only one of the categories of dependent variable.
2. Next, we train_test_split each of the previously created dataset individually.
3. We concat the Train Sets and Test Sets of each categories obtained from previous step to a single Train and Test set.
4. Shuffle to maintain randomness.

```

322 """
323 OnSplitData is used to split the dataset into Train/Test sets possessing equal propotion of all the
324 categories of the dependent variable.
325
326 data = The DataFrame.
327 t_size = Size of test set.
328
329 """
330 def OnSplitData( data, t_size = 0.2 ):
331
332     """
333     Assuming the dependent variable has two categories, we first create two independent dataframes each
334     containing data of only one of the categories of dependent variable.
335     """
336     data_c1 = data.loc[ data['Class'] == data['Class'].unique()[0] ]
337     data_c2 = data.loc[ data['Class'] == data['Class'].unique()[1] ]
338
339     """
340     Next, we train_test_split each of the previously created dataset individually.
341     """
342     c1_xtrain, c1_xtest, c1_ytrain, c1_ytest = tts( data_c1, data_c1['Class'], test_size = t_size, random_sta
343     c2_xtrain, c2_xtest, c2_ytrain, c2_ytest = tts( data_c2, data_c2['Class'], test_size = t_size, random_sta
344
345     """
346     We concat the Train Sets and Test Sets of each categories obtained from previous step to a single
347     Train and Test set.
348     """
349     xtrain = pd.concat( [c1_xtrain, c2_xtrain] )

```

14. **def OnRunModels:** OnRunModels function is used to run KNN. Naïve Bayes, SVM and logistic regression model at one go.



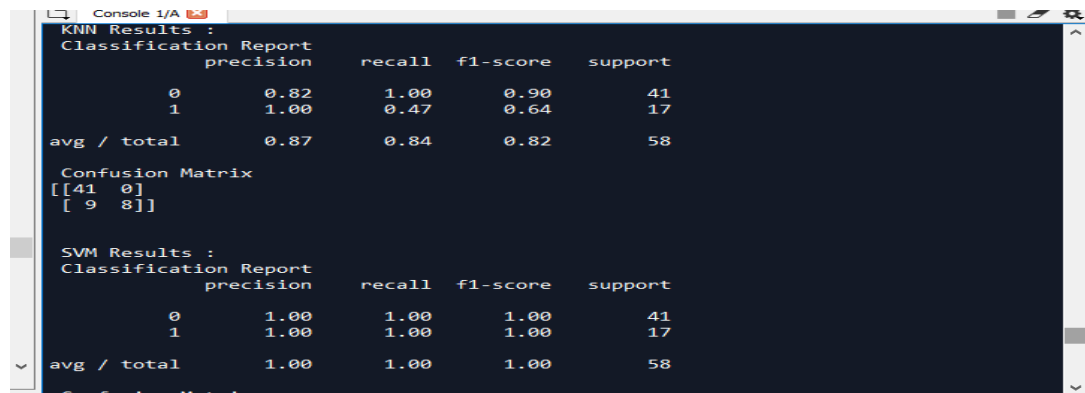
```

Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\10646237\Desktop\lab\cancer\Caner_Final2.py
Bokeh.py x prob2_v3.py x test.py x alarm.py x Caner_Final.py x Caner_Final2.py x
369 OnRunModels is used to run Models on the dataset.
370
371
372
373 def OnRunModels(data):
374
375     xtrain, xtest, ytrain, ytest = OnSplitData( data )
376
377     knn = KNN( n_neighbors = 10, metric = 'minkowski', p = 2 )
378     svm = SVM.SVC( kernel = 'Linear', C = 1.0, gamma = 'auto' )
379     mnb = MNB()
380     lreg = LREG( random_state = 0 )
381
382     knn.fit( xtrain, ytrain )
383     knn_preds = knn.predict( xtest )
384     print( ' KNN Results : ' )
385     print( ' Classification Report' )
386     print( CREP( ytest, knn_preds ) )
387     print( ' Confusion Matrix' )
388     print( CMAT( ytest, knn_preds ), '\n\n' )
389
390     svm.fit( xtrain, ytrain )
391     svm_preds = svm.predict( xtest )
392     print( ' SVM Results : ' )
393     print( ' Classification Report' )
394     print( CREP( ytest, svm_preds ) )
395     print( ' Confusion Matrix' )
396     print( CMAT( ytest, svm_preds ), '\n\n' )
397
398     mnb.fit( xtrain, ytrain )

```

Result:

1. I have used confusion matrix or classification report which calculates (accuracy, sensitivity, recall) for the given model.
2. I have identified all the four model gives almost same accuracy across three dataset. Of which SVM and logistic giving better accuracy than naïve bayes and KNN
3. I have model twice with all variables and after dropping variables showing low correlation based on plots, correlation matrix and some features which is not important for cancer. The model accuracy do not change much.



The screenshot shows the console output for KNN and SVM models. The KNN results show a precision of 0.87, recall of 0.84, and f1-score of 0.82. The SVM results show a precision of 1.00, recall of 1.00, and f1-score of 1.00.

```
KNN Results :
Classification Report
precision    recall  f1-score   support

0           0.82    1.00    0.90     41
1           1.00    0.47    0.64     17

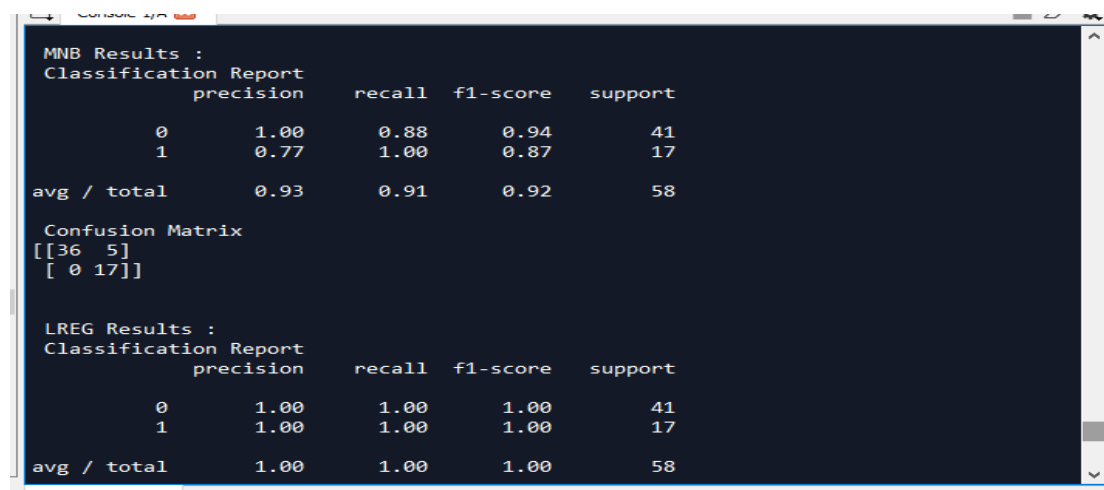
avg / total    0.87    0.84    0.82     58

Confusion Matrix
[[41  0]
 [ 9  8]]

SVM Results :
Classification Report
precision    recall  f1-score   support

0           1.00    1.00    1.00     41
1           1.00    1.00    1.00     17

avg / total    1.00    1.00    1.00     58
```



The screenshot shows the console output for MNB and LREG models. The MNB results show a precision of 0.93, recall of 0.91, and f1-score of 0.92. The LREG results show a precision of 1.00, recall of 1.00, and f1-score of 1.00.

```
MNB Results :
Classification Report
precision    recall  f1-score   support

0           1.00    0.88    0.94     41
1           0.77    1.00    0.87     17

avg / total    0.93    0.91    0.92     58

Confusion Matrix
[[36  5]
 [ 0 17]]

LREG Results :
Classification Report
precision    recall  f1-score   support

0           1.00    1.00    1.00     41
1           1.00    1.00    1.00     17

avg / total    1.00    1.00    1.00     58
```

SVM performed better in most of the dataset, possibly svm can be used in linear or non-linear ways with the use of a Kernel, when you have a limited set of points in many dimensions SVM tends to be very good because it should be able to find the linear separation that should exist. SVM is good with outliers as it will only use the most relevant points to find a linear separation (support vectors).

KNN is also very sensitive to bad features (attributes) so feature selection is also important. KNN is also sensitive to outliers, impacting the knn results.

Some features which are having poor correlation (i.e. texture), while some having high correlation in the datasets (adhesion), as mentioned above. Are contradicting with the biology concept.

Merging of datasets on Patient ID does not seem possible. This is because the first dataset lacks the information. While dataset 2 and 3 possess the Patient ID column there is not a single entry in the column of the 2 dataset that overlap/are similar, every single Patient ID in both the set is unique. Hence merging the dataset would simply mean concating the two frames and for every Patient ID of one dataset nullifying the values of all the columns of the other dataset.

Converting the current model into product would require:

1. Combining all the three dataset features into one, which I was not able to perform since the patient ID was not common across.
2. Working more into feature engg part: 1.e. converting all categorical/range variable into numeric and using it in model development:

Age1 = substr(data1\$age, 1,2)

Age1 = substr(data1\$age, 4,5)

Age_new = Age1 + Age2 # converting range into numeric.

3. Using Advance machine learning algorithms: Adaboost, Xgboost, Ripper etc.
 4. Using approach like LDA and then performing machine learning analysis on the Outcome of the same.
 5. The feature which are turning out to be poor correlated in the dataset, while the same feature is important prognostic factor in oncology, understanding the reason behind it.
-