★ AI / ML Project - Horse Survival Prognostication ★

Domain: Healthcare

Description:

Predict whether or not a horse can survive based upon past medical conditions.

Noted by the "outcome" variable in the data.

Content:

All of the binary representation have been converted into the words they actually represent. However, a fuller description is provided by the data dictionary (datadict.txt).

There are a lot of NA's in the data. This is the real struggle here. Try to find a way around it through imputation or other means.

Attribute Information:

1: surgery? 1 = Yes, it had surgery 2 = It was treated without surgery

2: Age 1 = Adult horse 2 = Young (< 6 months)

3: Hospital Number

- numeric id
- the case number assigned to the horse (may not be unique if the horse is treated > 1 time)

4: rectal temperature

- linear
- in degrees celsius.
- An elevated temp may occur due to infection.
- temperature may be reduced when the animal is in late shock
- normal temp is 37.8
- this parameter will usually change as the problem progresses, eg. may start out normal, then become elevated because of the lesion, passing back through the normal range as the horse goes into shock 5: pulse
- linear
- the heart rate in beats per minute
- is a reflection of the heart condition: 30 -40 is normal for adults
- rare to have a lower than normal rate although athletic horses may have a rate of 20-25
- animals with painful lesions or suffering from circulatory shock may have an elevated heart rate

6: respiratory rate

- linear
- normal rate is 8 to 10
- usefulness is doubtful due to the great fluctuations

7: temperature of extremities

- a subjective indication of peripheral circulation
- possible values: 1 = Normal 2 = Warm 3 = Cool 4 = Cold
- cool to cold extremities indicate possible shock
- hot extremities should correlate with an elevated rectal temp.

8: peripheral pulse

- subjective
- possible values are: 1 = normal 2 = increased 3 = reduced 4 = absent
- normal or increased p.p. are indicative of adequate circulation while reduced or absent indicate poor perfusion

9: mucous membranes

- a subjective measurement of colour
- possible values are: 1 = normal pink 2 = bright pink 3 = pale pink 4 = pale cyanotic 5 = bright red / injected 6 = dark cyanotic
- 1 and 2 probably indicate a normal or slightly increased circulation
- 3 may occur in early shock
- 4 and 6 are indicative of serious circulatory compromise
- 5 is more indicative of a septicemia

10: capillary refill time

- a clinical judgement. The longer the refill, the poorer the circulation
- possible values 1 = < 3 seconds 2 = >= 3 seconds

11: pain - a subjective judgement of the horse's pain level

- possible values: 1 = alert, no pain 2 = depressed 3 = intermittent mild pain 4 = intermittent severe pain 5 = continuous severe pain
- should NOT be treated as a ordered or discrete variable!
- In general, the more painful, the more likely it is to require surgery
- prior treatment of pain may mask the pain level to some extent

12: peristalsis

- an indication of the activity in the horse's gut. As the gut becomes more distended or the horse becomes more toxic, the activity decreases
- possible values: 1 = hypermotile 2 = normal 3 = hypomotile 4 = absent

13: abdominal distension

- An IMPORTANT parameter.
- possible values 1 = none 2 = slight 3 = moderate 4 = severe
- an animal with abdominal distension is likely to be painful and have reduced gut motility.
- a horse with severe abdominal distension is likely to require surgery just tio relieve the pressure

14: nasogastric tube

- this refers to any gas coming out of the tube
- possible values: 1 = none 2 = slight 3 = significant
- a large gas cap in the stomach is likely to give the horse discomfort

15: nasogastric reflux

- possible values 1 = none 2 = > 1 liter 3 = < 1 liter
- the greater amount of reflux, the more likelihood that there is some serious obstruction to the fluid passage from the rest of the intestine

16: nasogastric reflux PH

- linear
- scale is from 0 to 14 with 7 being neutral
- normal values are in the 3 to 4 range

17: rectal examination - feces

- possible values 1 = normal 2 = increased 3 = decreased 4 = absent
- absent feces probably indicates an obstruction

18: abdomen

- possible values 1 = normal 2 = other 3 = firm feces in the large intestine 4 = distended small intestine 5 = distended large intestine
- 3 is probably an obstruction caused by a mechanical impaction and is normally treated medically
- 4 and 5 indicate a surgical lesion

19: packed cell volume

- linear
- the # of red cells by volume in the blood
- normal range is 30 to 50. The level rises as the circulation becomes compromised or as the animal becomes dehydrated.

20: total protein

- linear
- normal values lie in the 6-7.5 (gms/dL) range
- the higher the value the greater the dehydration

21: abdominocentesis appearance

- a needle is put in the horse's abdomen and fluid is obtained from the abdominal cavity
- possible values: 1 = clear 2 = cloudy 3 = serosanguinous
- normal fluid is clear while cloudy or serosanguinous indicates a compromised gut

22: abdomcentesis total protein

- linear
- the higher the level of protein the more likely it is to have a compromised gut. Values are in gms/dL

23: outcome

- what eventually happened to the horse?
- possible values: 1 = lived 2 = died 3 = was euthanized

24: surgical lesion?

- retrospectively, was the problem (lesion) surgical?
- all cases are either operated upon or autopsied so that this value and the lesion type are always known
- possible values: 1 = Yes 2 = No

25, 26, 27: type of lesion

- first number is site of lesion 1 = gastric 2 = sm intestine 3 = lg colon 4 = lg colon and cecum 5 = cecum 6 = transverse colon 7 = retum/descending colon 8 = uterus 9 = bladder 11 = all intestinal sites 00 = none
- second number is type 1 = simple 2 = strangulation 3 = inflammation 4 = other
- third number is subtype 1 = mechanical 2 = paralytic 0 = n/a
- fourth number is specific code 1 = obturation 2 = intrinsic 3 = extrinsic 4 = adynamic 5 = volvulus/torsion 6 = intussuption 7 = thromboembolic 8 = hernia 9 = lipoma/slenic incarceration 10 = displacement 0 = n/a 28: cp_data
- is pathology data present for this case? 1 = Yes 2 = No
- this variable is of no significance since pathology data is not included or collected for these cases

Acknowledgements:

This dataset was originally published by the UCI Machine Learning Database:\ http://archive.ics.uci.edu/ml/datasets/Horse+Colic

Objective:

- Understand the Dataset & cleanup (if required).
- Build classification model to predict weather the horse will survive or not.
- Also fine-tune the hyperparameters & compare the evaluation metrics of vaious classification algorithms.

Stractegic Plan of Action:

We aim to solve the problem statement by creating a plan of action, Here are some of the necessary steps:

1. Data Exploration
2. Exploratory Data Analysis (EDA)
3. Data Pre-processing
4. Data Manipulation
5. Feature Selection/Extraction
6. Predictive Modelling
7. Project Outcomes & Conclusion

---

1. Data Exploration

In [1]:
```
#Importing the basic librarires

import os
import math
```

```
import scipy
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import tree
from scipy.stats import randint
from scipy.stats import loguniform
from IPython.display import display

from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import RFE
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold

from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, \
f1_score, roc_auc_score, roc_curve, precision_score, recall_score

import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10,6]

import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_columns', 50)
```

In [2]:

```
#Importing the dataset

df = pd.read_csv('horse.csv')
df.drop(['hospital_number'],axis=1, inplace=True)

target = 'outcome'
labels = ['died', 'euthanized', 'lived']
features = [i for i in df.columns.values if i not in [target]]

original_df = df.copy(deep=True)
display(df.head(8))

print('\n\033[1mInference:\033[0m The Datset consists of {} features & {} samples.'.format(df.shape[1], df.shape[0]))
```

	surgery	age	rectal_temp	pulse	respiratory_rate	temp_of_extremities	peripheral_pulse	mucous_membrane	capillary_refill_time	pain	peris
0	no	adult	38.5	66.0	28.0	cool	reduced	NaN	more_3_sec	extreme_pain	a
1	yes	adult	39.2	88.0	20.0	NaN	NaN	pale_cyanotic	less_3_sec	mild_pain	a
2	no	adult	38.3	40.0	24.0	normal	normal	pale_pink	less_3_sec	mild_pain	hypor
3	yes	young	39.1	164.0	84.0	cold	normal	dark_cyanotic	more_3_sec	depressed	a
4	no	adult	37.3	104.0	35.0	NaN	NaN	dark_cyanotic	more_3_sec	NaN	
5	no	adult	NaN	NaN	NaN	warm	normal	pale_pink	less_3_sec	depressed	hypor
6	yes	adult	37.9	48.0	16.0	normal	normal	normal_pink	less_3_sec	mild_pain	hypor
7	yes	adult	NaN	60.0	NaN	cool	NaN	NaN	less_3_sec	NaN	a

Inference: The Datset consists of 27 features & 299 samples.

In [3]:

```
#Checking the dtypes of all the columns

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 27 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   surgery                299 non-null    object 
 1   age                    299 non-null    object 
 2   rectal_temp            239 non-null    float64
 3   pulse                  275 non-null    float64
```

```
 4   respiratory_rate       241 non-null    float64
 5   temp_of_extremities    243 non-null    object
 6   peripheral_pulse       230 non-null    object
 7   mucous_membrane        252 non-null    object
 8   capillary_refill_time  267 non-null    object
 9   pain                   244 non-null    object
 10  peristalsis            255 non-null    object
 11  abdominal_distention   243 non-null    object
 12  nasogastric_tube       195 non-null    object
 13  nasogastric_reflux     193 non-null    object
 14  nasogastric_reflux_ph  53 non-null     float64
 15  rectal_exam_feces      197 non-null    object
 16  abdomen                181 non-null    object
 17  packed_cell_volume     270 non-null    float64
 18  total_protein          266 non-null    float64
 19  abdomo_appearance      134 non-null    object
 20  abdomo_protein         101 non-null    float64
 21  outcome                299 non-null    object
 22  surgical_lesion        299 non-null    object
 23  lesion_1               299 non-null    int64
 24  lesion_2               299 non-null    int64
 25  lesion_3               299 non-null    int64
 26  cp_data                299 non-null    object
dtypes: float64(7), int64(3), object(17)
memory usage: 63.2+ KB
```

In [4]:

```
#Checking number of unique rows in each feature

df.nunique().sort_values()
```

Out[4]:

```
surgery                 2
surgical_lesion         2
lesion_3                2
cp_data                 2
age                     2
nasogastric_reflux      3
outcome                 3
abdomo_appearance       3
capillary_refill_time   3
nasogastric_tube        3
abdominal_distention    4
rectal_exam_feces       4
peripheral_pulse        4
temp_of_extremities     4
peristalsis             4
pain                    5
abdomen                 5
mucous_membrane         6
lesion_2                6
nasogastric_reflux_ph  20
abdomo_protein         37
rectal_temp            40
respiratory_rate       40
packed_cell_volume     50
pulse                  52
lesion_1               61
total_protein          80
dtype: int64
```

In [5]:

```
#Checking number of unique rows in each feature

nu = df[features].nunique().sort_values()
nf = []; cf = []; nnf = 0; ncf = 0; #numerical & categorical features

for i in range(df[features].shape[1]):
    if nu.values[i]<=15:cf.append(nu.index[i])
    else: nf.append(nu.index[i])

print('\n\033[1mInference:\033[0m The Datset has {} numerical & {} categorical features.'.format(len(nf),len(cf)))
```

Inference: The Datset has 8 numerical & 18 categorical features.

In [6]:

```
#Checking the stats of all the columns

display(df.describe())
```

	rectal_temp	pulse	respiratory_rate	nasogastric_reflux_ph	packed_cell_volume	total_protein	abdomo_protein	lesion_1	lesion_2	
count	239.000000	275.000000	241.000000	53.000000	270.000000	266.000000	101.000000	299.000000	299.000000	29
mean	38.168619	72.000000	30.460581	4.707547	46.307407	24.274436	3.039604	3659.709030	90.528428	
std	0.733744	28.646219	17.666102	1.982311	10.436743	27.364194	1.967947	5408.472421	650.637139	12
min	35.400000	30.000000	8.000000	1.000000	23.000000	3.300000	0.100000	0.000000	0.000000	
25%	37.800000	48.000000	18.000000	3.000000	38.000000	6.500000	2.000000	2111.500000	0.000000	
50%	38.200000	64.000000	25.000000	5.000000	45.000000	7.500000	2.300000	2322.000000	0.000000	
75%	38.500000	88.000000	36.000000	6.500000	52.000000	56.750000	3.900000	3209.000000	0.000000	

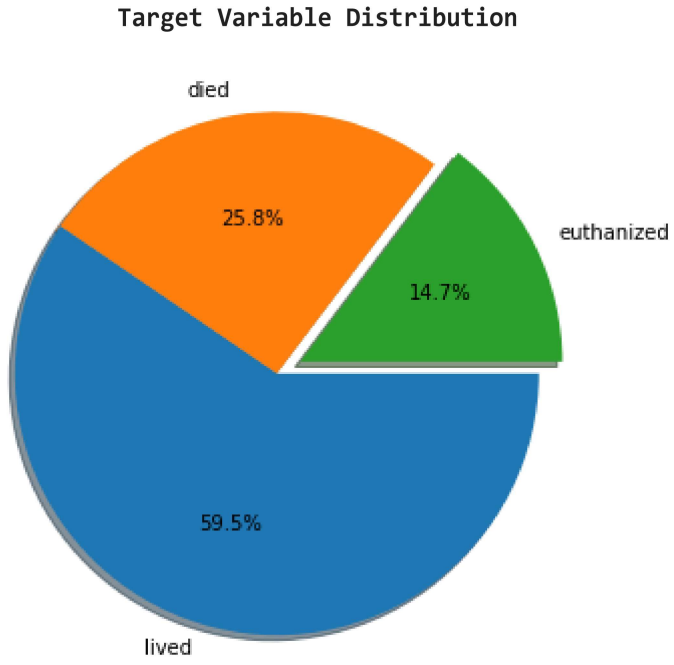
	rectal_temp	pulse	respiratory_rate	nasogastric_reflux_ph	packed_cell_volume	total_protein	abdomo_protein	lesion_1	lesion_2	
max	40.800000	184.000000	96.000000	7.500000	75.000000	89.000000	10.100000	41110.000000	7111.000000	220

Inference: The stats seem to be fine, let us gain more undestanding by visualising the dataset.

---

2. Exploratory Data Analysis (EDA)

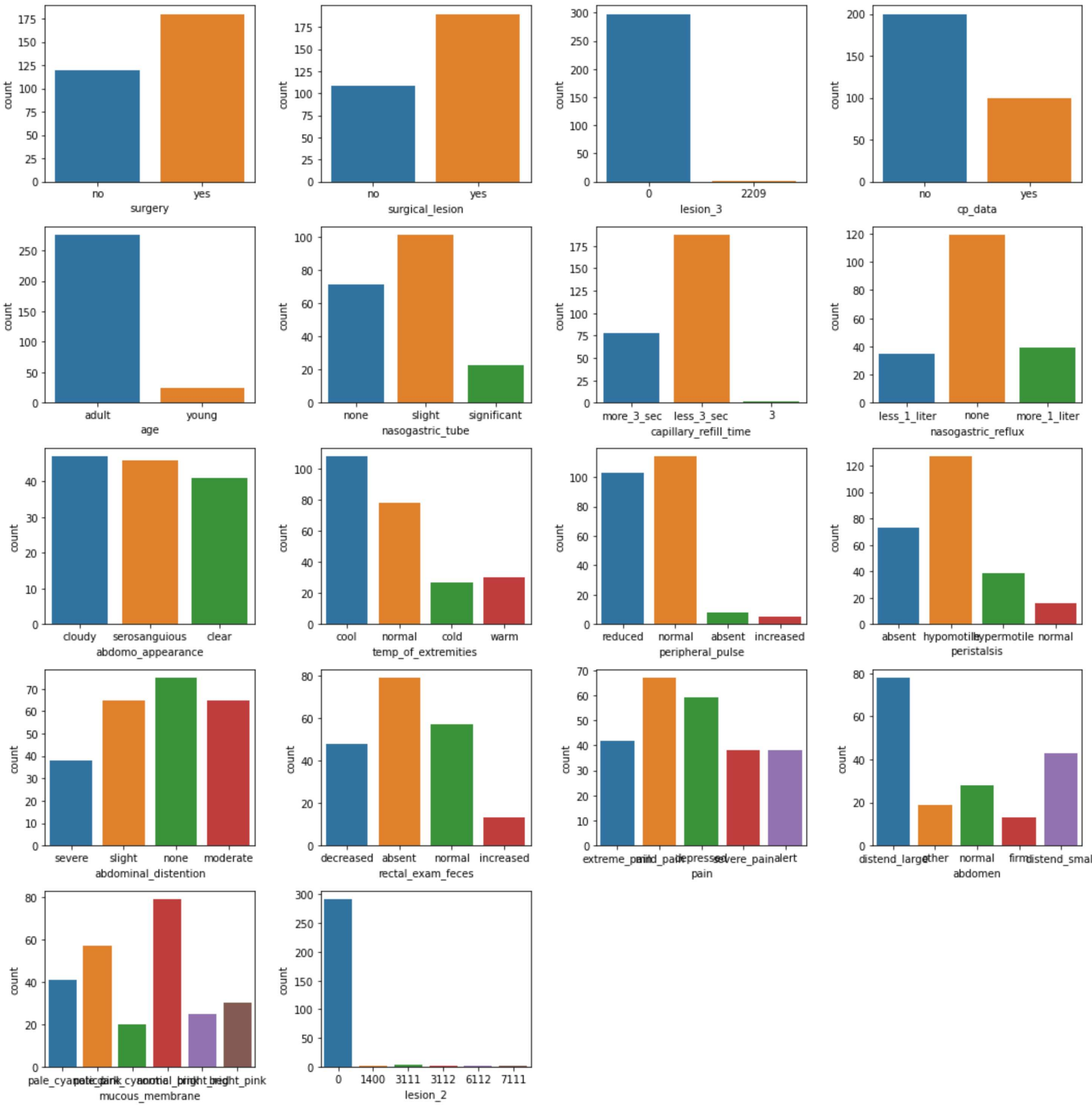
In [7]:
```
#Let us first analyze the distribution of the target variable

MAP={}
for e, i in enumerate(df[target].unique()):
    MAP[i]=labels[e]
#MAP={0:'Not-Survived',1:'Survived'}
df1 = df.copy()
df1[target]=df1[target].map(MAP)
explode=np.zeros(len(labels))
explode[-1]=0.1
print('\033[1mTarget Variable Distribution'.center(55))
plt.pie(df1[target].value_counts(), labels=df1[target].value_counts().index, counterclock=False, shadow=True,
        explode=explode, autopct='%1.1f%%', radius=1, startangle=0)
plt.show()
```



Inference: The Target Variable seems to be slightly imbalanced! Hence we shall try to perform data augmentation.

In [8]:
```
#Visualising the categorical features

print('\033[1mVisualising Categorical Features:'.center(100))

n=4
plt.figure(figsize=[15,3*math.ceil(len(cf)/n)])

for i in range(len(cf)):
    if df[cf[i]].nunique()<=15:
        plt.subplot(math.ceil(len(cf)/n),n,i+1)
        sns.countplot(df[cf[i]])
    #else:
    #    plt.subplot(2,2,i)
    #    sns.countplot(df[cf[i]])
plt.tight_layout()
plt.show()
```

Visualising Categorical Features:

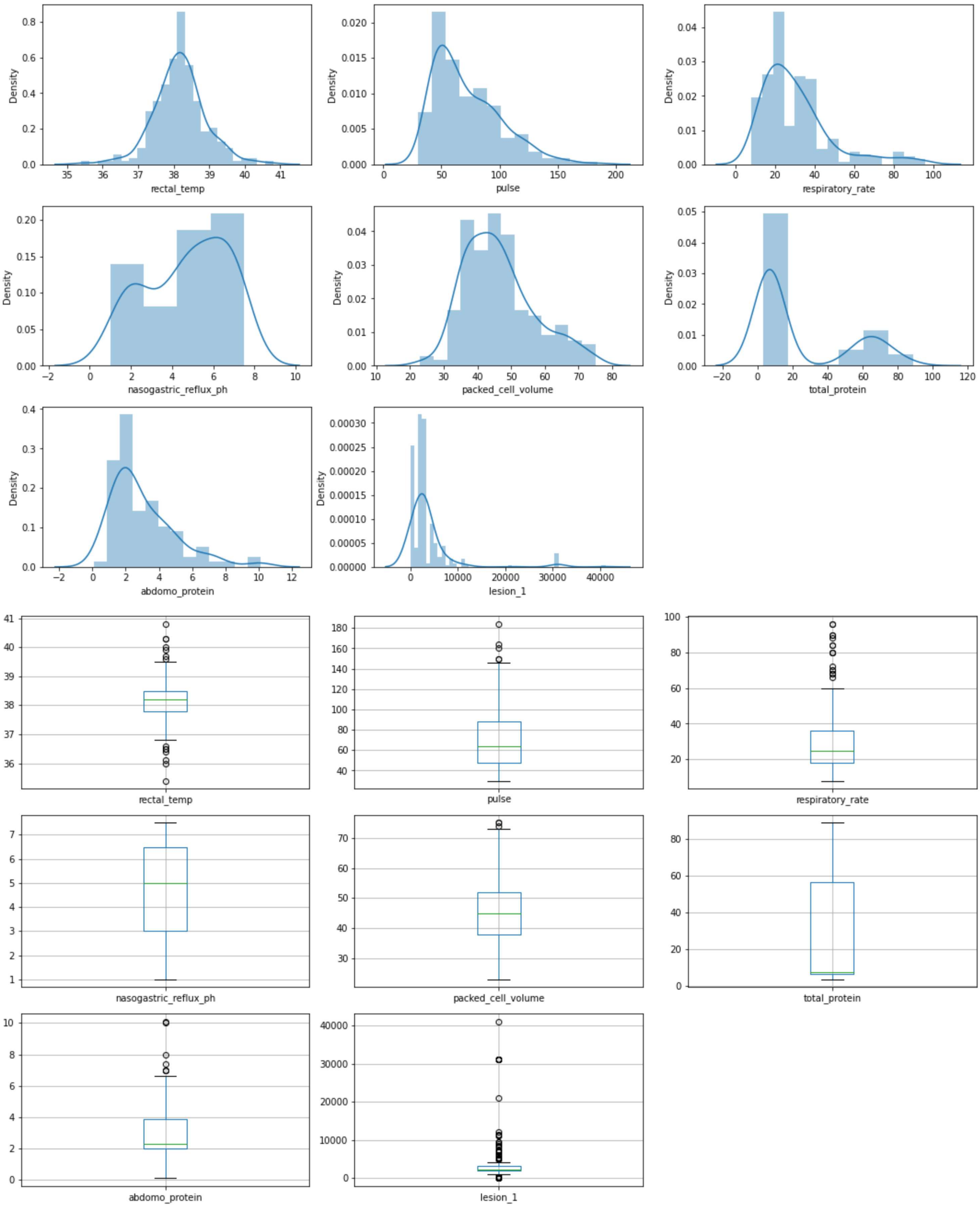


Inference: Visualizing the categorical features reveal lot of information about the dataset.

In [9]:

```
#Understanding the feature set

print('\033[1mFeatures Distribution'.center(100))

n=3
nf = [i for i in features if i not in cf]

plt.figure(figsize=[15,3*math.ceil(len(features)/n)])
for c in range(len(nf)):
    plt.subplot(math.ceil(len(features)/n),n,c+1)
    sns.distplot(df[nf[c]])
plt.tight_layout()
plt.show()

plt.figure(figsize=[15,3*math.ceil(len(features)/n)])
for c in range(len(nf)):
    plt.subplot(math.ceil(len(features)/n),n,c+1)
    df.boxplot(nf[c])
plt.tight_layout()
plt.show()
```

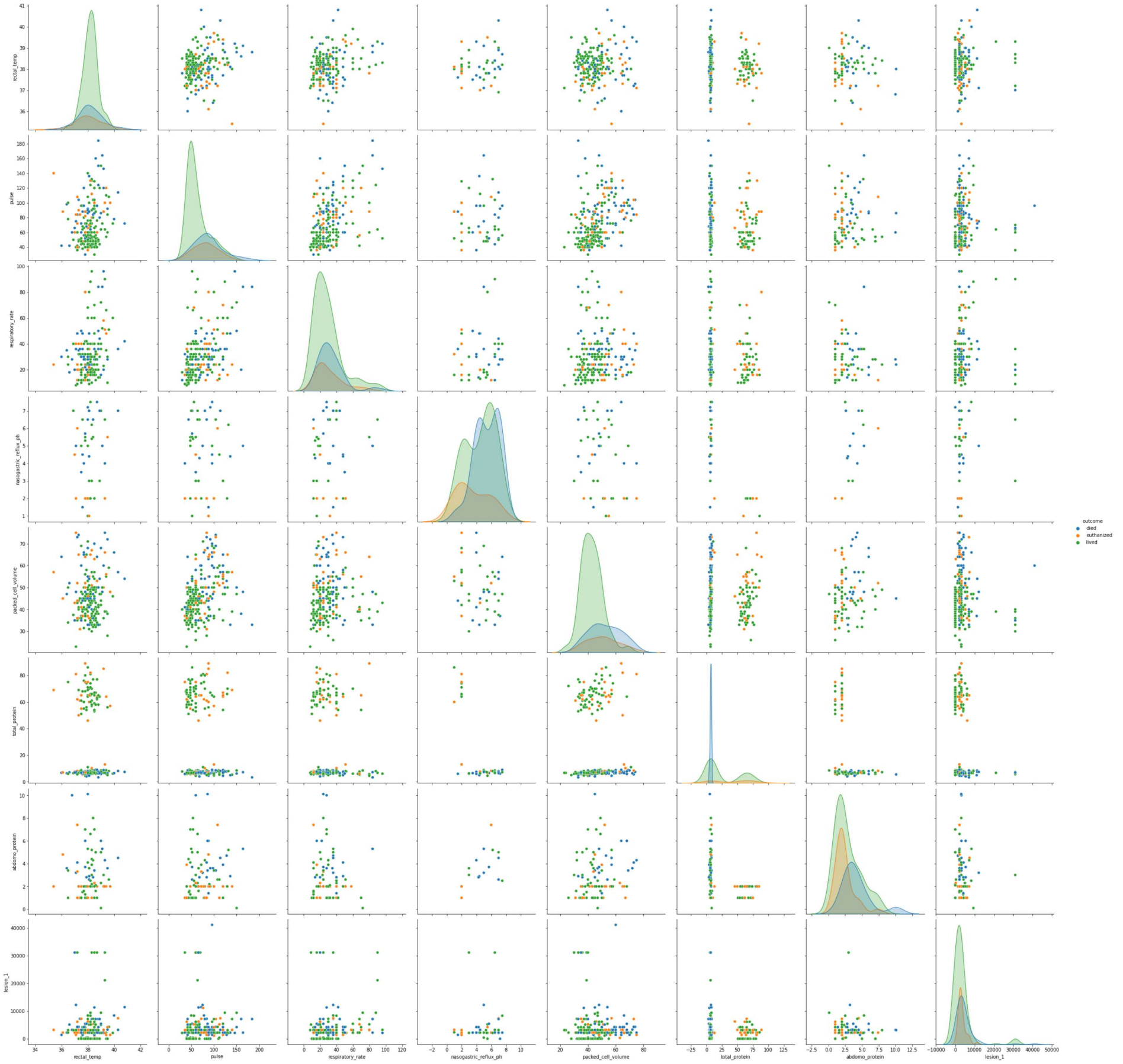
Features Distribution



Inference: The data is somewhat normally distributed. And there are many outliers present in the dataset. We shall fix these outliers..

In [10]:

```
#Understanding the relationship between all the features

ppc=[i for i in df1.columns if i not in cf]
g=sns.pairplot(df1[ppc], hue=target, size=4)
#g.map_upper(sns.kdeplot, levels=1, color=".2")
plt.show()
```



Inference: The data samples of most of the features do show some patterns. Also they seem to have lot of overlap for the outcome classes, making it difficult to be distingusihable. Let is proceed to perform cleanup on the data to remove the irregularities...

---

3. Data Preprocessing

In [86]:
```
#Removal of any Duplicate rows (if any)

counter = 0
r,c = original_df.shape

df1 = df.copy()
df1.drop_duplicates(inplace=True)
df1.reset_index(drop=True,inplace=True)

if df1.shape==(r,c):
    print('\n\033[1mInference:\033[0m The dataset doesn\'t have any duplicates')
else:
    print(f'\n\033[1mInference:\033[0m Number of duplicates dropped ---> {r-df1.shape[0]}')
```

Inference: The dataset doesn't have any duplicates

In [87]:
```
#Check for empty elements

nvc = pd.DataFrame(df.isnull().sum().sort_values(), columns=['Total Null Values'])
nvc['Percentage'] = round(nvc['Total Null Values']/df.shape[0],3)*100
print(nvc)
```

```
                      Total Null Values  Percentage
surgery                               0         0.0
lesion_2                              0         0.0
lesion_1                              0         0.0
surgical_lesion                       0         0.0
```

outcome	0	0.0
lesion_3	0	0.0
cp_data	0	0.0
age	0	0.0
pulse	24	8.0
packed_cell_volume	29	9.7
capillary_refill_time	32	10.7
total_protein	33	11.0
peristalsis	44	14.7
mucous_membrane	47	15.7
pain	55	18.4
abdominal_distention	56	18.7
temp_of_extremities	56	18.7
respiratory_rate	58	19.4
rectal_temp	60	20.1
peripheral_pulse	69	23.1
rectal_exam_feces	102	34.1
nasogastric_tube	104	34.8
nasogastric_reflux	106	35.5
abdomen	118	39.5
abdomo_appearance	165	55.2
abdomo_protein	198	66.2
nasogastric_reflux_ph	246	82.3

Inference: There are many outliers in the dataset. Let us try to impute the missing values

In [88]:
```
#Converting categorical Columns to Numeric

df1 = df.copy()
ecc = nvc[nvc['Percentage']!=0].index.values
dcc = [i for i in df.columns if i not in ecc]

#Target Variable
MAP={}
for i,e in enumerate(df1[target].unique()):
    MAP[e]=i
df1[target]=df1[target].map(MAP)
print('Mapping Target variable --->',MAP)

df3 = df1[dcc]
fcc = [i for i in cf if i not in ecc]

#One-Hot Binay Encoding
oh=True
dm=True
for i in fcc:
    #print(i)
    if df3[i].nunique()==2:
        if oh==True: print("\033[1m\nOne-Hot Encoding on features:\033[0m")
        print(i);oh=False
        df3[i]=pd.get_dummies(df3[i], drop_first=True, prefix=str(i))
    if (df3[i].nunique()>2 and df3[i].nunique()<17):
        if dm==True: print("\n\033[1mDummy Encoding on features:\033[0m")
        print(i);dm=False
        df3 = pd.concat([df3.drop([i], axis=1), pd.DataFrame(pd.get_dummies(df3[i], drop_first=True, prefix=str(i)))],axis=1)

df3.shape
```

```
Mapping Target variable ---> {'died': 0, 'euthanized': 1, 'lived': 2}

One-Hot Encoding on features:
surgery
surgical_lesion
lesion_3
cp_data
age

Dummy Encoding on features:
lesion_2
```

Out[88]:
```
(299, 12)
```

In [89]:
```
# Fixing Empty Categorical Columns

for x in [i for i in ecc if i in cf]:
    a = df1[x]
    b=[]; c=[]

    for i,e in enumerate(a):
        if e!=e:
            b.append(i)
        else:
            c.append(i)

    RF = RandomForestClassifier()
    RF.fit(df3.loc[c],a[c])
    d = RF.predict(df3.loc[b])

    df3[x] = a
    f=0
    for i,e in enumerate(df3[x]):
        if e!=e:
```

```
            df3.loc[i,x] = d[f]
            f+=1
    df3 = pd.concat([df3.drop([x], axis=1), pd.DataFrame(pd.get_dummies(df3[x], drop_first=True, prefix=str(x)))],axis=1)
df3
```

Out[89]:

	surgery	age	outcome	surgical_lesion	lesion_1	lesion_3	cp_data	lesion_2_1400	lesion_2_3111	lesion_2_3112	lesion_2_6112	lesion_2_7111	capilla
0	0	0	0	0	11300	0	0	0	0	0	0	0	
1	1	0	1	0	2208	0	0	0	0	0	0	0	
2	0	0	2	0	0	0	1	0	0	0	0	0	
3	1	1	0	1	2208	0	1	0	0	0	0	0	
4	0	0	0	0	4300	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
294	1	0	1	0	3205	0	0	0	0	0	0	0	
295	0	0	1	1	2208	0	1	0	0	0	0	0	
296	1	0	0	1	3205	0	0	0	0	0	0	0	
297	1	0	2	1	2208	0	1	0	0	0	0	0	
298	1	0	1	0	6112	0	0	0	0	0	0	0	

299 rows × 48 columns

In [90]:

```
# Fixing Empty Numerical Columns

for x in [i for i in ecc if i not in cf]:
    a = df1[x]
    b=[]; c=[]

    for i,e in enumerate(a):
        if e!=e:
            b.append(i)
        else:
            c.append(i)

    LR = LinearRegression()
    LR.fit(df3.loc[c],a[c])
    d = LR.predict(df3.loc[b])

    df3[x] = a
    f=0
    for i,e in enumerate(df3[x]):
        if e!=e:
            df3.loc[i,x] = d[f]
            f+=1
    #df3 = pd.concat([df3.drop([x], axis=1), pd.DataFrame(pd.get_dummies(df3[x], drop_first=True, prefix=str(x)))],axis=1)
df3
```

Out[90]:

	surgery	age	outcome	surgical_lesion	lesion_1	lesion_3	cp_data	lesion_2_1400	lesion_2_3111	lesion_2_3112	lesion_2_6112	lesion_2_7111	capilla
0	0	0	0	0	11300	0	0	0	0	0	0	0	
1	1	0	1	0	2208	0	0	0	0	0	0	0	
2	0	0	2	0	0	0	1	0	0	0	0	0	
3	1	1	0	1	2208	0	1	0	0	0	0	0	
4	0	0	0	0	4300	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
294	1	0	1	0	3205	0	0	0	0	0	0	0	
295	0	0	1	1	2208	0	1	0	0	0	0	0	
296	1	0	0	1	3205	0	0	0	0	0	0	0	
297	1	0	2	1	2208	0	1	0	0	0	0	0	
298	1	0	1	0	6112	0	0	0	0	0	0	0	

299 rows × 55 columns

In [109…

```
#Removal of outlier:

df4 = df3.copy()

for i in [i for i in df4.columns]:
    if df4[i].nunique()>=12:
        Q1 = df4[i].quantile(0.20)
        Q3 = df4[i].quantile(0.80)
```

```
        IQR = Q3 - Q1
        df4 = df4[df4[i] <= (Q3+(1.5*IQR))]
        df4 = df4[df4[i] >= (Q1-(1.5*IQR))]
df4 = df4.reset_index(drop=True)
display(df4.head())
print('\n\033[1mInference:\033[0m Before removal of outliers, The dataset had {} samples.'.format(df1.shape[0]))
print('\033[1mInference:\033[0m After removal of outliers, The dataset now has {} samples.'.format(df4.shape[0]))
```

	surgery	age	outcome	surgical_lesion	lesion_1	lesion_3	cp_data	lesion_2_1400	lesion_2_3111	lesion_2_3112	lesion_2_6112	lesion_2_7111	capillary
0	1	0	1	0	2208	0	0	0	0	0	0	0	
1	0	0	2	0	0	0	1	0	0	0	0	0	
2	0	0	0	0	4300	0	0	0	0	0	0	0	
3	0	0	2	0	0	0	0	0	0	0	0	0	
4	1	0	2	1	3124	0	0	0	0	0	0	0	

5 rows × 55 columns

Inference: Before removal of outliers, The dataset had 299 samples.
Inference: After removal of outliers, The dataset now has 269 samples.

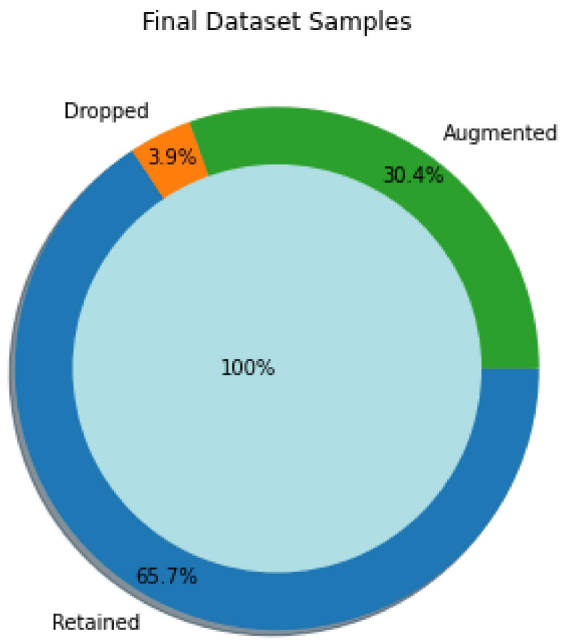
In [110…

```
#Fixing the imbalance using SMOTE Technique

df5 = df4.copy()

print('Original class distribution:')
print(df5[target].value_counts())

xf = df5.columns
X = df5.drop([target],axis=1)
Y = df5[target]

smote = SMOTE()
X, Y = smote.fit_resample(X, Y)

df5 = pd.DataFrame(X, columns=xf)
df5[target] = Y

print('\nClass distribution after applying SMOTE Technique:',)
print(Y.value_counts())
```

```
Original class distribution:
2    167
0     63
1     39
Name: outcome, dtype: int64

Class distribution after applying SMOTE Technique:
1    167
2    167
0    167
Name: outcome, dtype: int64
```

In [111…

```
#Final Dataset size after performing Preprocessing

df = df5.copy()
plt.title('Final Dataset Samples')
plt.pie([df.shape[0], original_df.shape[0]-df4.shape[0], df5.shape[0]-df4.shape[0]], radius = 1, shadow=True,
        labels=['Retained','Dropped','Augmented'], counterclock=False, autopct='%1.1f%%', pctdistance=0.9, explode=[0,0,0])
plt.pie([df.shape[0]], labels=['100%'], labeldistance=-0, radius=0.78, shadow=True, colors=['powderblue'])
plt.show()

print('\n\033[1mInference:\033[0mThe final dataset after cleanup has {} samples & {} columns.'.format(df.shape[0], df.shape[1]))
```



Inference: The final dataset after cleanup has 501 samples & 55 columns.

---

4. Data Manipulation

In [112…

```
#Splitting the data intro training & testing sets

df = df5.copy()

X = df.drop([target],axis=1)
Y = df[target]
Train_X, Test_X, Train_Y, Test_Y = train_test_split(X, Y, train_size=0.8, test_size=0.2, random_state=0)

print('Original set  ---> ',X.shape,Y.shape,'\nTraining set  ---> ',Train_X.shape,Train_Y.shape,'\nTesting set   ---> ', Test_X.sh
```

```
Original set  --->  (501, 54) (501,)
Training set  --->  (400, 54) (400,)
Testing set   --->  (101, 54)  (101,)
```

In [113…

```
#Feature Scaling (Standardization)

std = StandardScaler()

print('\033[1mStandardardization on Training set'.center(100))
Train_X_std = std.fit_transform(Train_X)
Train_X_std = pd.DataFrame(Train_X_std, columns=X.columns)
display(Train_X_std.describe())

print('\n','\033[1mStandardardization on Testing set'.center(100))
Test_X_std = std.transform(Test_X)
Test_X_std = pd.DataFrame(Test_X_std, columns=X.columns)
display(Test_X_std.describe())
```

Standardardization on Training set

	surgery	age	surgical_lesion	lesion_1	lesion_3	cp_data	lesion_2_1400	lesion_2_3111	lesion_2_3112	lesion_2_611
count	4.000000e+02	4.000000e+02	4.000000e+02	4.000000e+02	4.000000e+02	4.000000e+02	4.000000e+02	4.000000e+02	4.000000e+02	400.00000
mean	-2.886580e-17	-4.440892e-17	-1.465494e-16	5.107026e-17	1.776357e-17	4.440892e-17	1.776357e-17	-1.998401e-17	1.776357e-17	0.00000
std	1.001252e+00	1.001252e+00	1.001252e+00	1.001252e+00	1.001252e+00	1.001252e+00	1.001252e+00	1.001252e+00	1.001252e+00	1.00125
min	-1.116777e+00	-2.170724e-01	-1.231147e+00	-1.610614e+00	-5.006262e-02	-5.194625e-01	-5.006262e-02	-8.692914e-02	-5.006262e-02	-0.05006
25%	-1.116777e+00	-2.170724e-01	-1.231147e+00	-4.278308e-01	-5.006262e-02	-5.194625e-01	-5.006262e-02	-8.692914e-02	-5.006262e-02	-0.05006
50%	8.954339e-01	-2.170724e-01	8.122506e-01	1.163077e-01	-5.006262e-02	-5.194625e-01	-5.006262e-02	-8.692914e-02	-5.006262e-02	-0.05006
75%	8.954339e-01	-2.170724e-01	8.122506e-01	1.707077e-01	-5.006262e-02	-5.194625e-01	-5.006262e-02	-8.692914e-02	-5.006262e-02	-0.05006
max	8.954339e-01	4.606758e+00	8.122506e-01	3.607343e+00	1.997498e+01	1.925067e+00	1.997498e+01	1.150362e+01	1.997498e+01	19.97498

8 rows × 54 columns

Standardardization on Testing set

	surgery	age	surgical_lesion	lesion_1	lesion_3	cp_data	lesion_2_1400	lesion_2_3111	lesion_2_3112	lesion_2_6112	lesion_2_7
count	101.000000	101.000000	101.000000	101.000000	1.010000e+02	101.000000	1.010000e+02	1.010000e+02	1.010000e+02	1.010000e+02	101.00
mean	0.118442	-0.169312	0.043447	-0.063842	-5.006262e-02	0.085619	-5.006262e-02	-8.692914e-02	-5.006262e-02	-5.006262e-02	0.00

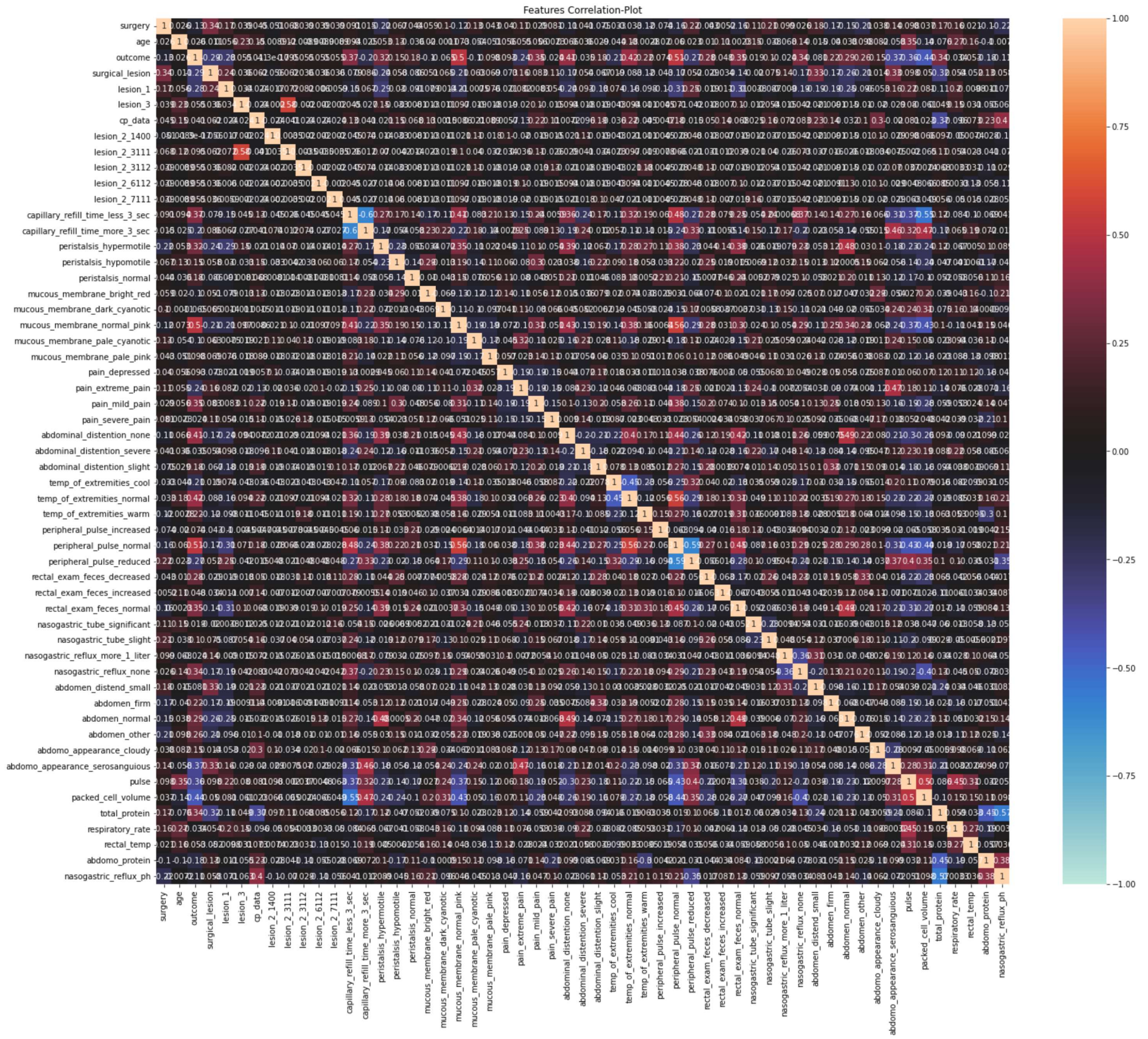
	surgery	age	surgical_lesion	lesion_1	lesion_3	cp_data	lesion_2_1400	lesion_2_3111	lesion_2_3112	lesion_2_6112	lesion_2_7
std	0.984557	0.479989	0.994842	0.870181	1.394700e-17	1.060258	1.394700e-17	2.789401e-17	1.394700e-17	1.394700e-17	0.09
min	-1.116777	-0.217072	-1.231147	-1.610614	-5.006262e-02	-0.519462	-5.006262e-02	-8.692914e-02	-5.006262e-02	-5.006262e-02	0.00
25%	-1.116777	-0.217072	-1.231147	-0.386615	-5.006262e-02	-0.519462	-5.006262e-02	-8.692914e-02	-5.006262e-02	-5.006262e-02	0.00
50%	0.895434	-0.217072	0.812251	0.116308	-5.006262e-02	-0.519462	-5.006262e-02	-8.692914e-02	-5.006262e-02	-5.006262e-02	0.00
75%	0.895434	-0.217072	0.812251	0.168487	-5.006262e-02	-0.519462	-5.006262e-02	-8.692914e-02	-5.006262e-02	-5.006262e-02	0.00
max	0.895434	4.606758	0.812251	2.391115	-5.006262e-02	1.925067	-5.006262e-02	-8.692914e-02	-5.006262e-02	-5.006262e-02	1.00

8 rows × 54 columns

5. Feature Selection/Extraction

In [114…

```
#Checking the correlation
features = df.columns
plt.figure(figsize=[24,20])
plt.title('Features Correlation-Plot')
sns.heatmap(df[features].corr(), vmin=-1, vmax=1, center=0, annot=True) #,
plt.show()
```



Inference: \ Correlation plt between the variables convey lot of information about the realationship betweem them. Especially in case of gender & survived.Hence it is clear that probably women were given more importance to save first. Similiary we have obvious strong correlation between fare &

Passenger-Class.

Let us check with different techniques if we can improve the model's performance by performing Feature Selection/Extraction steps to take care of these multi-collinearity...

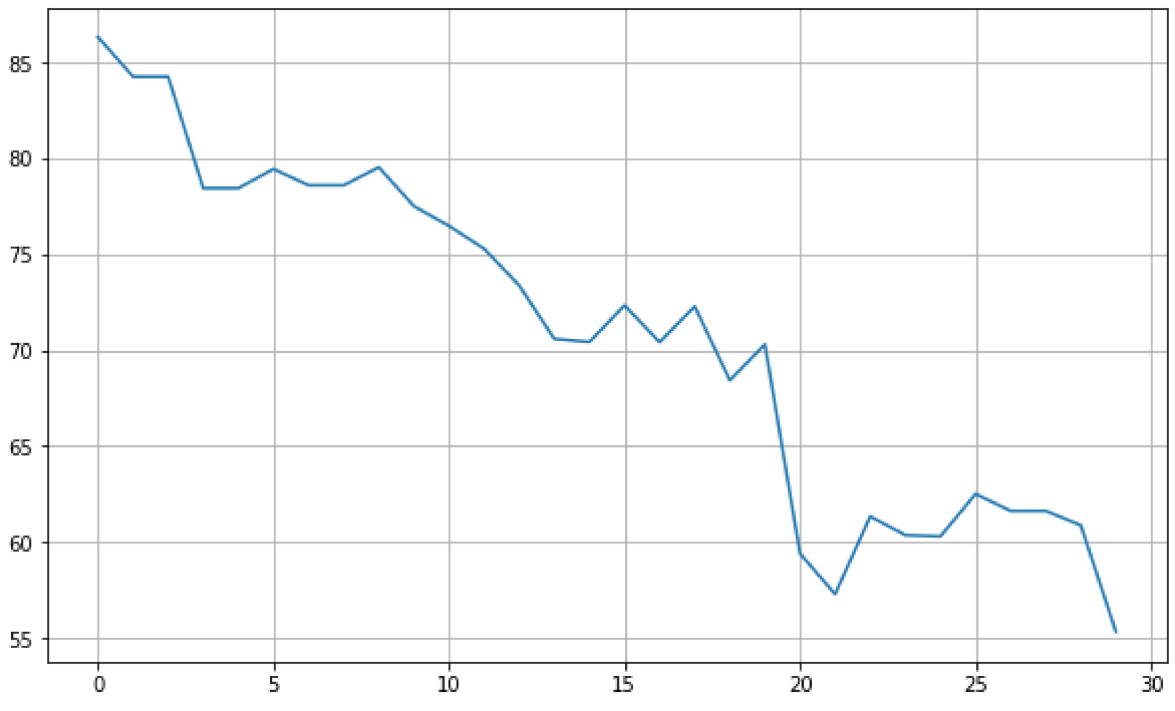
Strategy: \ We can fix these multicollinearity with two techniques:

1. Manual Method - Variance Inflation Factor (VIF)
2. Automatic Method - Recursive Feature Elimination (RFE)
3. Decomposition Method - Principle Component Analysis (PCA)

5a. Manual Method - VIF

In [115…

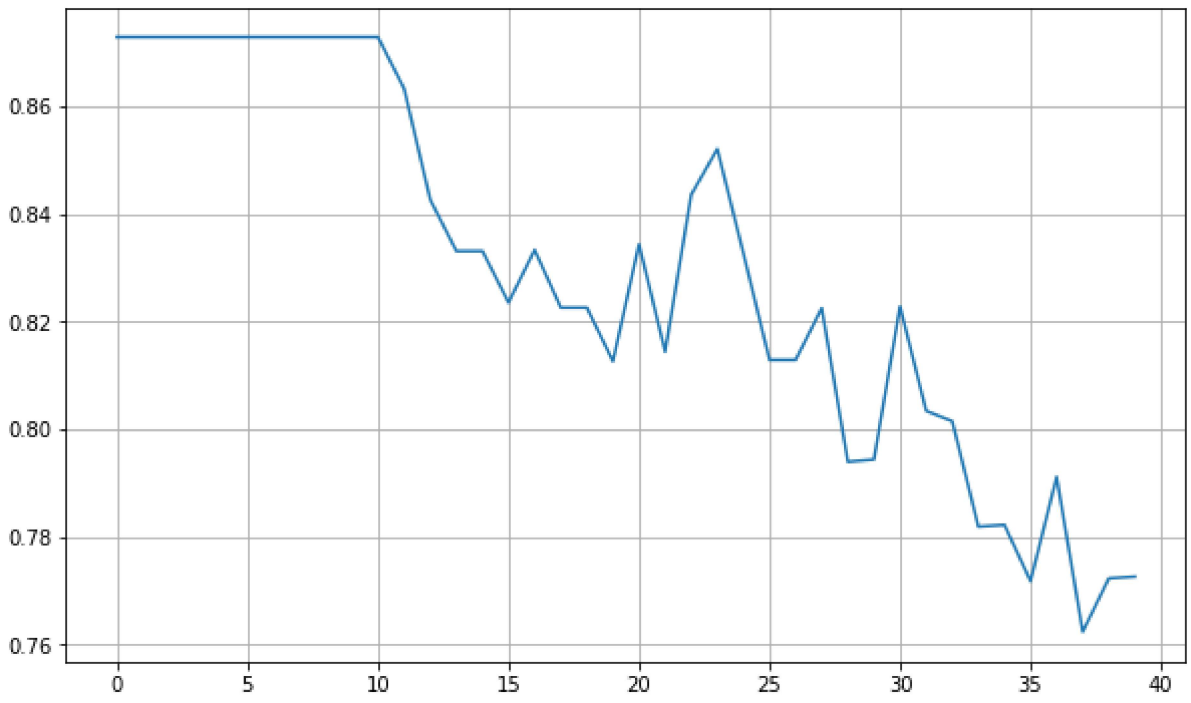
```
# Calculate the VIFs to remove multicollinearity

DROP=[]; scores=[]
#scores.append(f1_score(Test_Y,LogisticRegression().fit(Train_X_std, Train_Y).predict(Test_X_std)))

for i in range(len(X.columns.values)-10):
    vif = pd.DataFrame()
    Xs = X.drop(DROP,axis=1)
    #print(DROP)
    vif['Features'] = Xs.columns
    vif['VIF'] = [variance_inflation_factor(Xs.values, i) for i in range(Xs.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    vif.reset_index(drop=True, inplace=True)
    DROP.append(vif.Features[0])
    if vif.VIF[0]>1.5:
        scores.append(f1_score(Test_Y,LogisticRegression().fit(Train_X_std.drop(DROP,axis=1), Train_Y).predict(Test_X_std.drop(DRO
    #print(scores)

plt.plot(scores)
#plt.ylim([0.7,0.85])
plt.grid()
plt.show()
```



5b. Automatic Method - RFE

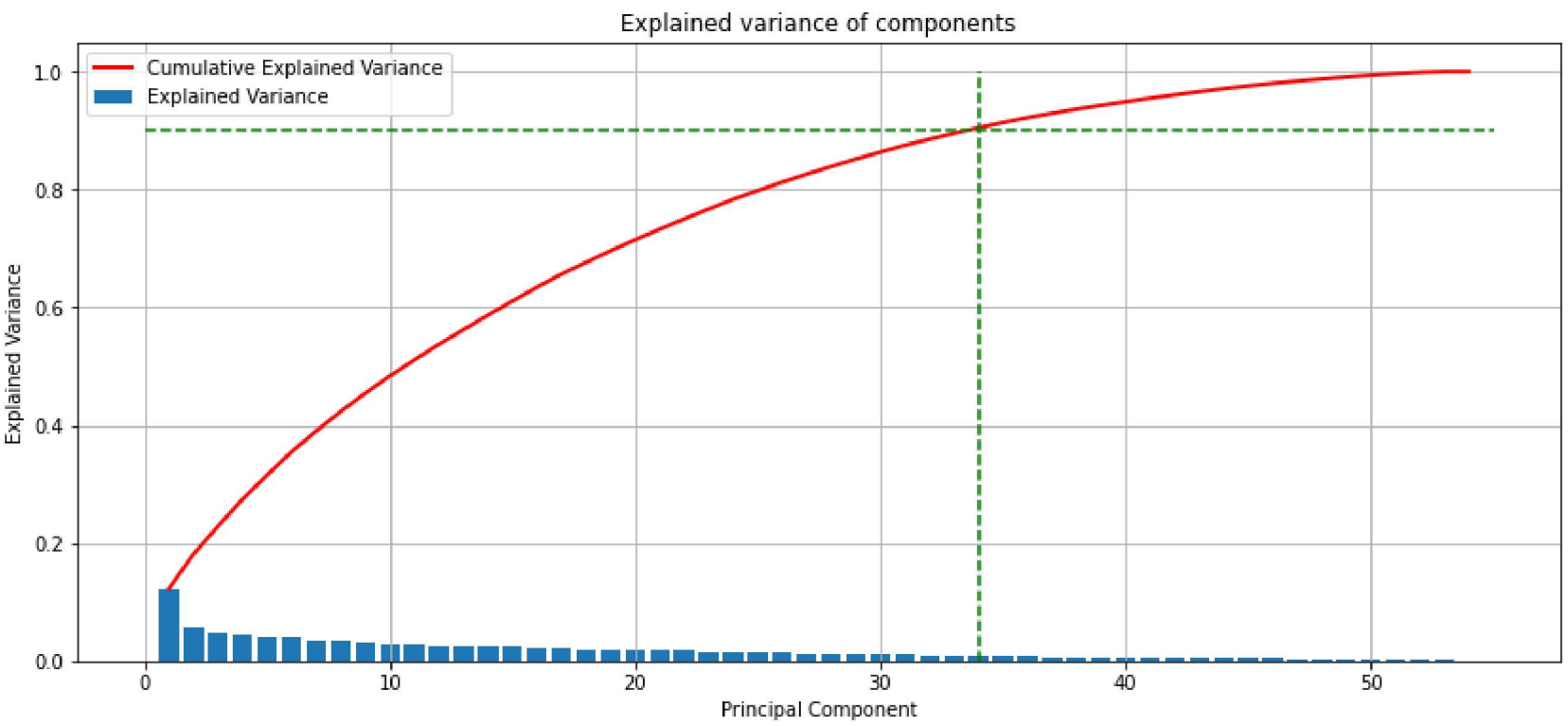
In [116…

```
# Applying Recurrsive Feature Elimination

# Running RFE with the output number of the variable equal to 10
LR = LogisticRegression()#.fit(Train_X_std, Train_Y)
scores=[]

for i in range(40):
    rfe = RFE(LR,n_features_to_select=len(Train_X_std.columns)-i)
    rfe = rfe.fit(Train_X_std, Train_Y)
    scores.append(f1_score(Test_Y,LogisticRegression().fit(Train_X_std[Train_X_std.columns[rfe.support_]], Train_Y).predict(Test_X
    #print(scores)

plt.plot(scores)
#plt.ylim([0.80,0.84])
plt.grid()
plt.show()
```



In [119…

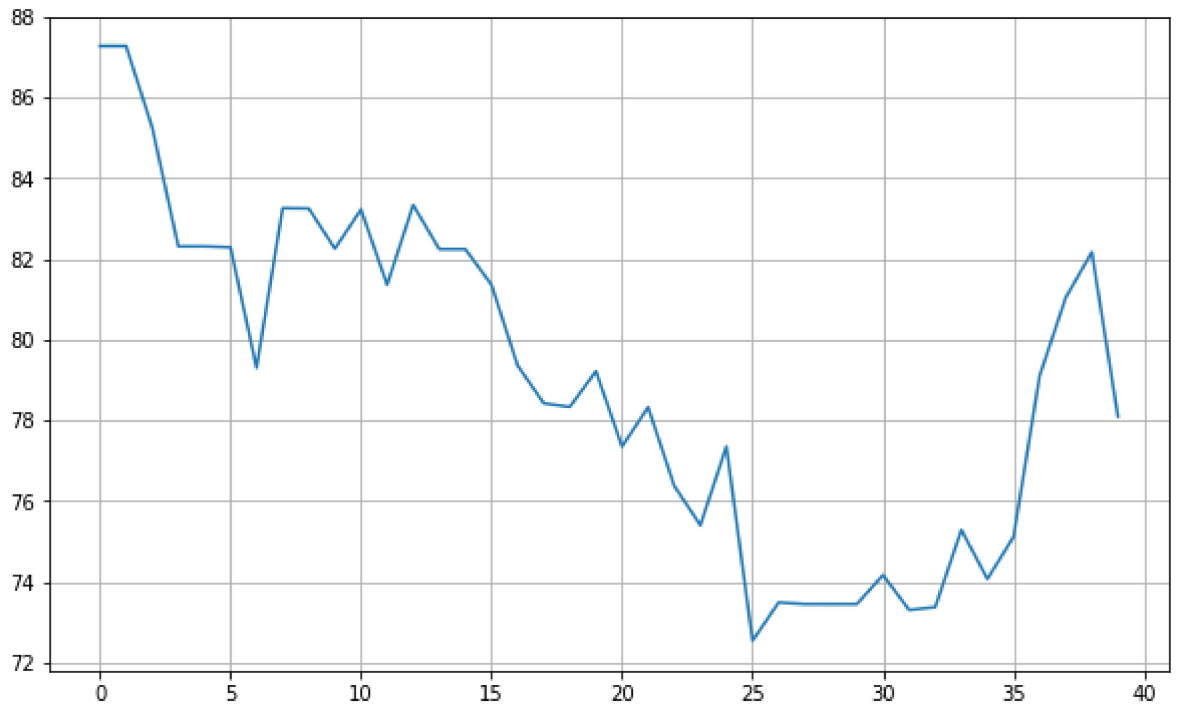
```
from sklearn.decomposition import PCA

pca = PCA().fit(Train_X_std)

fig, ax = plt.subplots(figsize=(14,6))
x_values = range(1, pca.n_components_+1)
ax.bar(x_values, pca.explained_variance_ratio_, lw=2, label='Explained Variance')
ax.plot(x_values, np.cumsum(pca.explained_variance_ratio_), lw=2, label='Cumulative Explained Variance', color='red')
plt.plot([0,pca.n_components_+1],[0.90,0.90],'g--')
plt.plot([34,34],[0,1], 'g--')
ax.set_title('Explained variance of components')
ax.set_xlabel('Principal Component')
ax.set_ylabel('Explained Variance')
plt.grid()
plt.legend()
plt.show()
```



Inference: We shall avoid performing dimensionality reduction for the current problem.

In [118…

```
#Applying PCA Transformations

scores=[]
for i in range(40):
    pca = PCA(n_components=Train_X_std.shape[1]-i)
    Train_X_std_pca = pca.fit_transform(Train_X_std)
    #print('The shape of final transformed training feature set:')
    #print(Train_X_std_pca.shape)
    Train_X_std_pca = pd.DataFrame(Train_X_std_pca)

    Test_X_std_pca = pca.transform(Test_X_std)
    #print('\nThe shape of final transformed testing feature set:')
    #print(Test_X_std_pca.shape)
    Test_X_std_pca = pd.DataFrame(Test_X_std_pca)

    scores.append(f1_score(Test_Y,LogisticRegression().fit(Train_X_std_pca, Train_Y).predict(Test_X_std_pca),average='weighted')*1

plt.plot(scores)
#plt.ylim([0.80,0.84])
plt.grid()
plt.show()
```



Inference: In VIF, RFE & PCA Techniques, we did notice any better scores upon dropping some multicollinear features. But in order to avoid the curse of dimensionality, we can capture top 90% of the data Variance explained by top 33 PCA components.

In [124…

```
#Finalising the shortlisted features

rfe = RFE(LR,n_features_to_select=len(Train_X_std.columns)-10)
rfe = rfe.fit(Train_X_std, Train_Y)

Train_X_std = Train_X_std[Train_X_std.columns[rfe.support_]]
Test_X_std = Test_X_std[Test_X_std.columns[rfe.support_]]

print(Train_X_std.shape)
print(Test_X_std.shape)
```

```
(400, 44)
(101, 44)
```

---

6. Predictive Modeling

In [128…

```
#Let us create first create a table to store the results of various models

Evaluation_Results = pd.DataFrame(np.zeros((8,5)), columns=['Accuracy', 'Precision','Recall','F1-score','AUC-ROC score'])
Evaluation_Results.index=['Logistic Regression (LR)','Decision Tree Classifier (DT)','Random Forest Classifier (RF)','Naïve Bayes
                          'Support Vector Machine (SVM)','K Nearest Neighbours (KNN)', 'Gradient Boosting (GB)','Extreme Gradient B
Evaluation_Results
```

Out[128…

	Accuracy	Precision	Recall	F1-score	AUC-ROC score
Logistic Regression (LR)	0.0	0.0	0.0	0.0	0.0
Decision Tree Classifier (DT)	0.0	0.0	0.0	0.0	0.0
Random Forest Classifier (RF)	0.0	0.0	0.0	0.0	0.0
Naïve Bayes Classifier (NB)	0.0	0.0	0.0	0.0	0.0
Support Vector Machine (SVM)	0.0	0.0	0.0	0.0	0.0
K Nearest Neighbours (KNN)	0.0	0.0	0.0	0.0	0.0
Gradient Boosting (GB)	0.0	0.0	0.0	0.0	0.0
Extreme Gradient Boosting (XGB)	0.0	0.0	0.0	0.0	0.0

In [135…

```
#Let us define functions to summarise the Prediction's scores .

#Classification Summary Function
def Classification_Summary(pred,pred_prob,i):
    Evaluation_Results.iloc[i]['Accuracy']=round(accuracy_score(Test_Y, pred),3)*100
    Evaluation_Results.iloc[i]['Precision']=round(precision_score(Test_Y, pred, average='weighted'),3)*100 #
    Evaluation_Results.iloc[i]['Recall']=round(recall_score(Test_Y, pred, average='weighted'),3)*100 #
    Evaluation_Results.iloc[i]['F1-score']=round(f1_score(Test_Y, pred, average='weighted'),3)*100 #
    Evaluation_Results.iloc[i]['AUC-ROC score']=round(roc_auc_score(Test_Y, pred_prob, multi_class='ovr'),3)*100 #[:, 1]
    print('{}{}\033[1m Evaluating {} \033[0m{}{}\n'.format('<'*3,'-'*35,Evaluation_Results.index[i], '-'*35,'>'*3))
    print('Accuracy = {}%'.format(round(accuracy_score(Test_Y, pred),3)*100))
    print('F1 Score = {}%'.format(round(f1_score(Test_Y, pred, average='weighted'),3)*100)) #
    print('\n \033[1mConfusiton Matrix:\033[0m\n',confusion_matrix(Test_Y, pred))
    print('\n\033[1mClassification Report:\033[0m\n',classification_report(Test_Y, pred))

    auc_roc(Test_Y, pred_prob, curves=['each_class'])
    plt.show()

#Visualising Function
def AUC_ROC_plot(Test_Y, pred):
    ref = [0 for _ in range(len(Test_Y))]
```

```
    ref_auc = roc_auc_score(Test_Y, ref)
    lr_auc = roc_auc_score(Test_Y, pred)

    ns_fpr, ns_tpr, _ = roc_curve(Test_Y, ref)
    lr_fpr, lr_tpr, _ = roc_curve(Test_Y, pred)

    plt.plot(ns_fpr, ns_tpr, linestyle='--')
    plt.plot(lr_fpr, lr_tpr, marker='.', label='AUC = {}'.format(round(roc_auc_score(Test_Y, pred)*100,2)))
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()
```

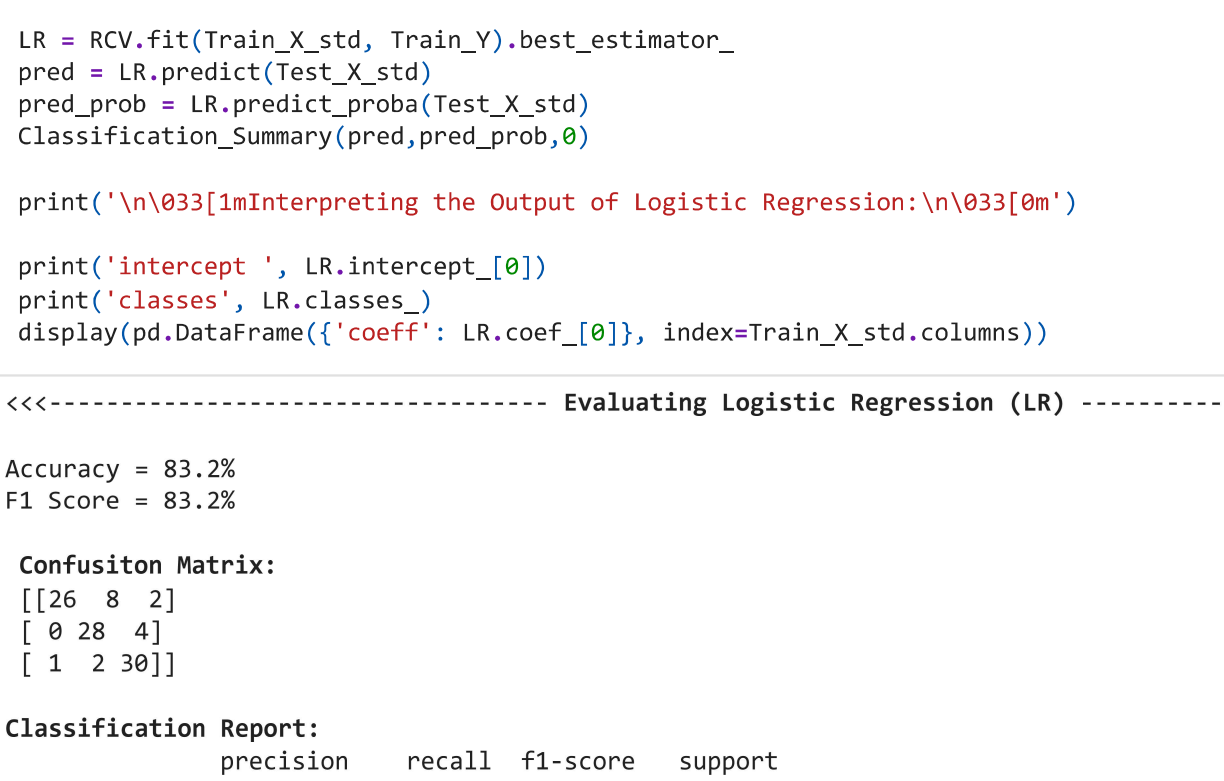
---

1. Logistic Regression:

In [136…

```
# Building Logistic Regression Classifier

LR_model = LogisticRegression()

space = dict()
space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
space['C'] = loguniform(1e-5, 100)

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(LR_model, space, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

LR = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = LR.predict(Test_X_std)
pred_prob = LR.predict_proba(Test_X_std)
Classification_Summary(pred,pred_prob,0)

print('\n\033[1mInterpreting the Output of Logistic Regression:\n\033[0m')

print('intercept ', LR.intercept_[0])
print('classes', LR.classes_)
display(pd.DataFrame({'coeff': LR.coef_[0]}, index=Train_X_std.columns))
```

```
<<<---------------------------------- Evaluating Logistic Regression (LR) ----------------------------------->>>

Accuracy = 83.2%
F1 Score = 83.2%

 Confusiton Matrix:
 [[26  8  2]
 [ 0 28  4]
 [ 1  2 30]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.72      0.83        36
           1       0.74      0.88      0.80        32
           2       0.83      0.91      0.87        33

    accuracy                           0.83       101
   macro avg       0.84      0.84      0.83       101
weighted avg       0.85      0.83      0.83       101
```



Interpreting the Output of Logistic Regression:

```
intercept  -113.42992166530351
classes [0 1 2]
```

	coeff
surgery	49.904420
age	18.539828
surgical_lesion	-56.111282
lesion_1	128.902677
cp_data	-21.041914
lesion_2_3111	9.778063
capillary_refill_time_less_3_sec	107.286553
capillary_refill_time_more_3_sec	-130.477140
peristalsis_hypermotile	-47.378380
peristalsis_hypomotile	17.106468
peristalsis_normal	-14.490431
mucous_membrane_bright_red	66.698144
mucous_membrane_dark_cyanotic	-25.504627
mucous_membrane_normal_pink	-147.225742
mucous_membrane_pale_cyanotic	-86.311840
mucous_membrane_pale_pink	-106.637220
pain_depressed	-93.299865
pain_extreme_pain	112.620482
pain_mild_pain	-130.187159
pain_severe_pain	57.172807
abdominal_distention_none	-152.077052
abdominal_distention_severe	-94.877823
abdominal_distention_slight	90.454070
temp_of_extremities_cool	73.842611
temp_of_extremities_normal	10.849007
peripheral_pulse_increased	-24.208206
peripheral_pulse_normal	-47.359983
peripheral_pulse_reduced	-84.462422
rectal_exam_feces_decreased	-20.005110
rectal_exam_feces_increased	13.172308
rectal_exam_feces_normal	-100.482489
nasogastric_tube_slight	8.610889
nasogastric_reflux_more_1_liter	-24.587464
nasogastric_reflux_none	39.000865
abdomen_firm	-81.127833
abdomen_normal	4.977446
abdomen_other	-126.136887
abdomo_appearance_cloudy	-1.096022
abdomo_appearance_serosanguious	76.919202
packed_cell_volume	123.047790
total_protein	-269.787713
rectal_temp	-13.462604
abdomo_protein	119.373373
nasogastric_reflux_ph	-40.973568

---

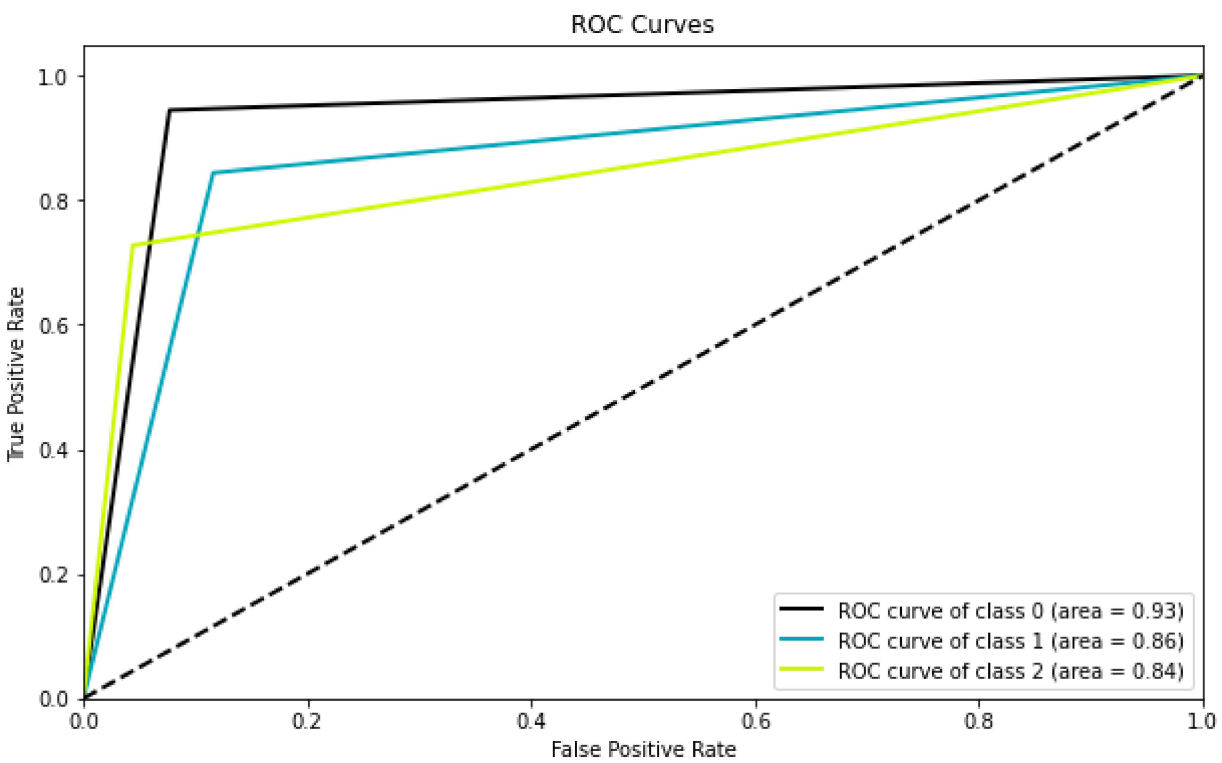
2. Decisoin Tree Classfier:

In [137…

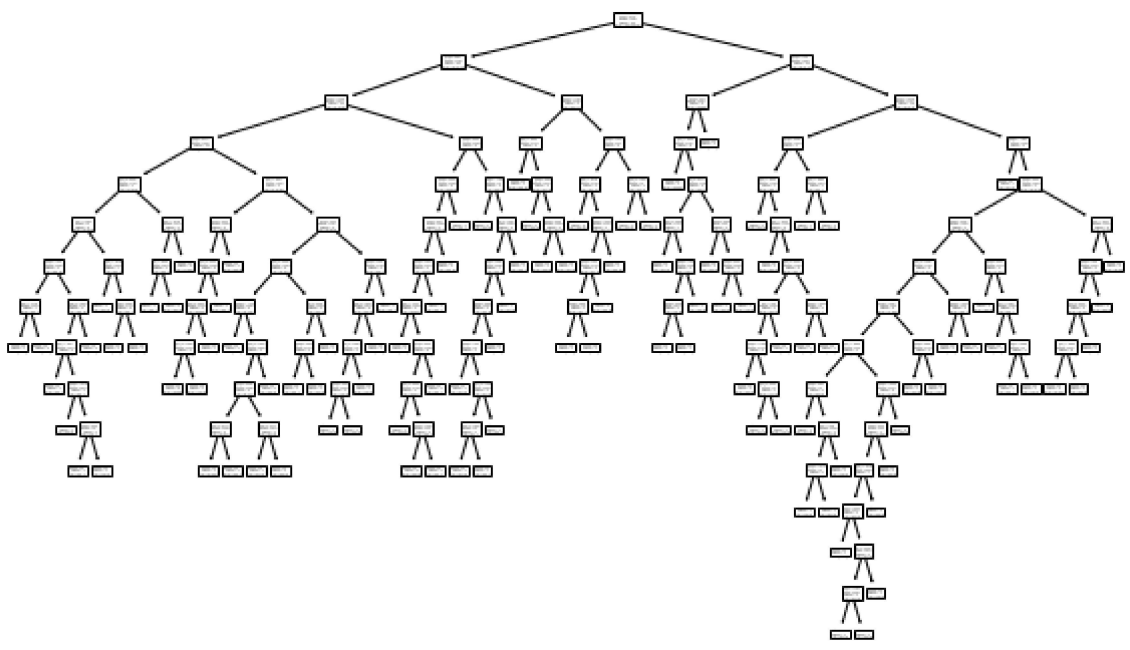
```
#Building Decision Tree Classifier

DT_model = DecisionTreeClassifier()

param_dist = {"max_depth": [3, None],
              "max_features": randint(1, 9),
              "min_samples_leaf": randint(1, 9),
              "criterion": ["gini", "entropy"]}
```

```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(DT_model, param_dist, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

DT = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = DT.predict(Test_X_std)
pred_prob = DT.predict_proba(Test_X_std)
Classification_Summary(pred,pred_prob,1)

print('\n\033[1mInterpreting the output of Decision Tree:\n\033[0m')
tree.plot_tree(DT)
plt.show()
```

```
<<<---------------------------------- Evaluating Decision Tree Classifier (DT) ----------------------------------->>>

Accuracy = 84.2%
F1 Score = 84.0%

 Confusiton Matrix:
 [[34  1  1]
 [ 3 27  2]
 [ 2  7 24]]

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.94      0.91        36
           1       0.77      0.84      0.81        32
           2       0.89      0.73      0.80        33

    accuracy                           0.84       101
   macro avg       0.84      0.84      0.84       101
weighted avg       0.85      0.84      0.84       101
```



Interpreting the output of Decision Tree:



---

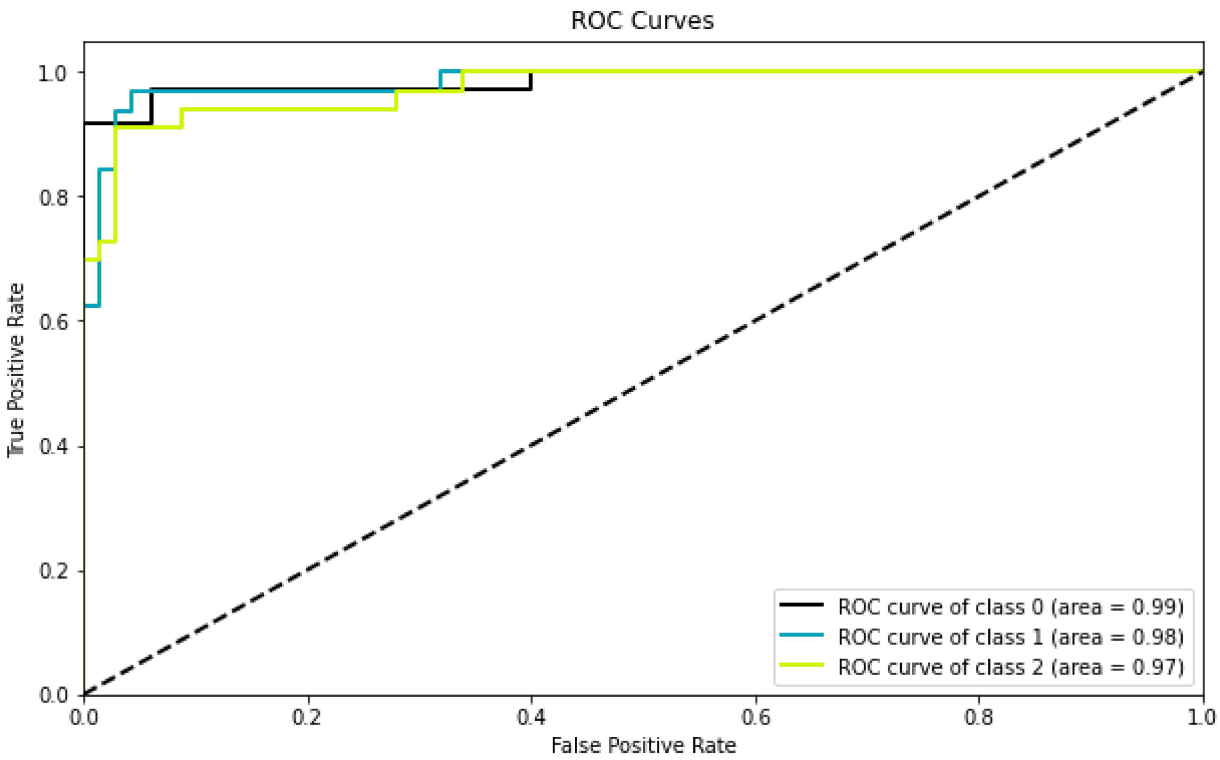
3. Random Forest Classfier:

In [138…

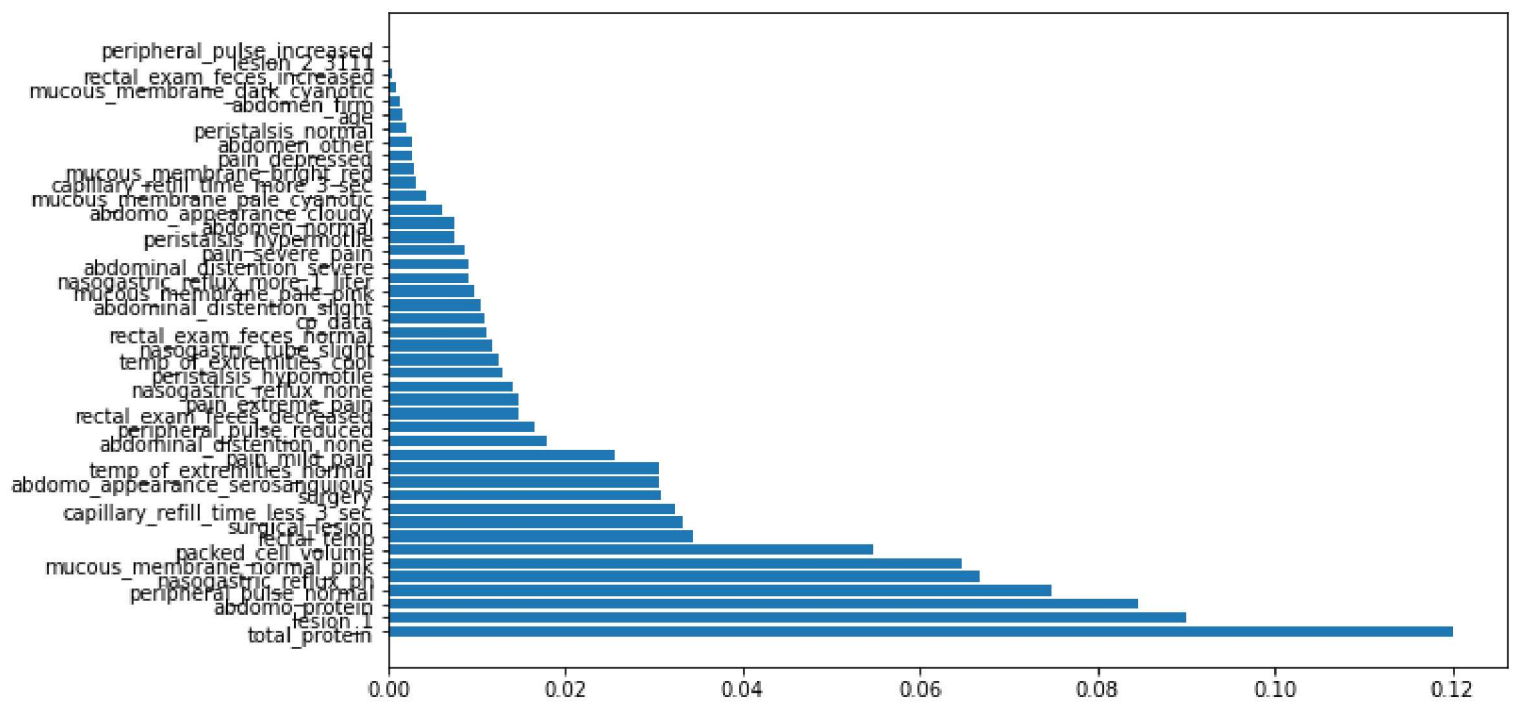
```
# Building Random-Forest Classifier

RF_model = RandomForestClassifier()
```

```
param_dist={'bootstrap': [True, False],
            'max_depth': [10, 20, 50, 100, None],
            'max_features': ['auto', 'sqrt'],
            'min_samples_leaf': [1, 2, 4],
            'min_samples_split': [2, 5, 10],
            'n_estimators': [50, 100]}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(RF_model, param_dist, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

RF = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = RF.predict(Test_X_std)
pred_prob = RF.predict_proba(Test_X_std)
Classification_Summary(pred,pred_prob,2)

print('\n\033[1mInterpreting the output of Random Forest:\n\033[0m')
rfi=pd.Series(RF.feature_importances_, index=Train_X_std.columns).sort_values(ascending=False)
plt.barh(rfi.index,rfi.values)
plt.show()
```

```
<<<---------------------------------- Evaluating Random Forest Classifier (RF) ----------------------------------->>>

Accuracy = 93.10000000000001%
F1 Score = 93.10000000000001%

 Confusiton Matrix:
 [[33  1  2]
 [ 1 30  1]
 [ 1  1 31]]

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.92      0.93        36
           1       0.94      0.94      0.94        32
           2       0.91      0.94      0.93        33

    accuracy                           0.93       101
   macro avg       0.93      0.93      0.93       101
weighted avg       0.93      0.93      0.93       101
```



Interpreting the output of Random Forest:

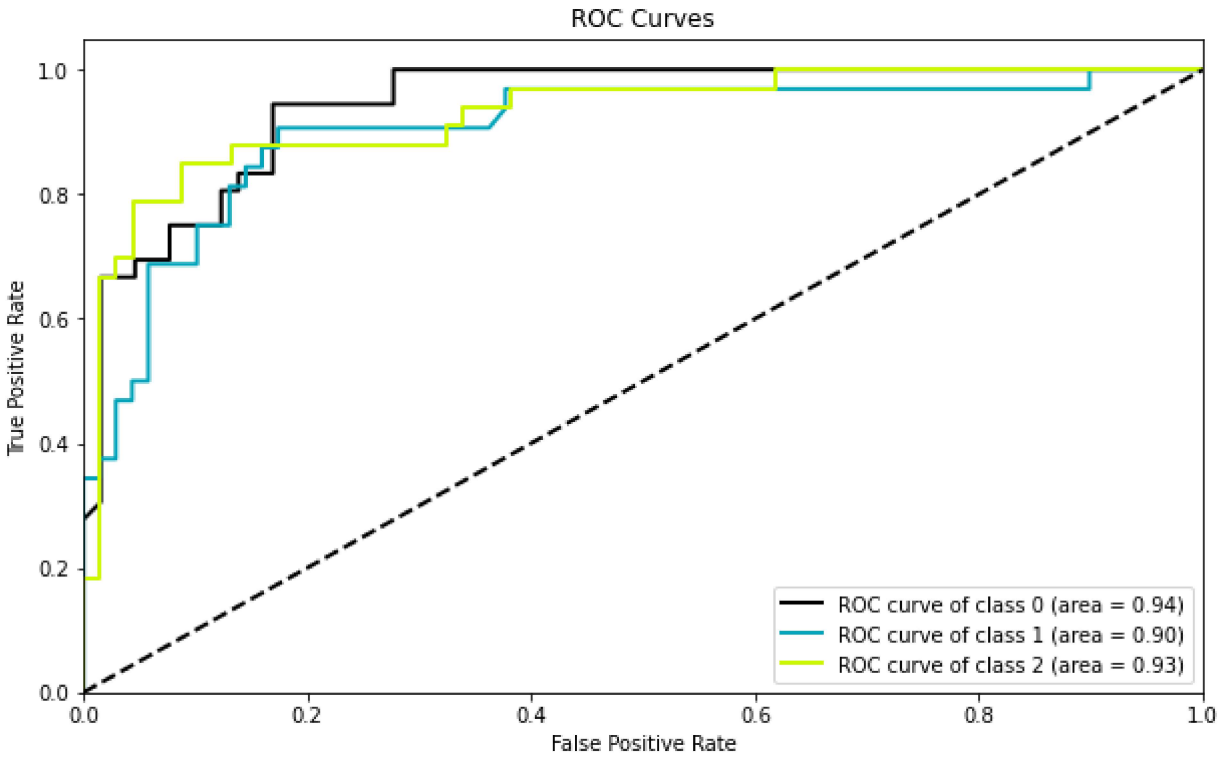


4. Naive Bayes Classfier:

In [139…

```
# Building Naive Bayes Classifier

NB_model = BernoulliNB()

params = {'alpha': [0.01, 0.1, 0.5, 1.0, 10.0]}
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(NB_model, params, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

NB = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = NB.predict(Test_X_std)
pred_prob = NB.predict_proba(Test_X_std)
Classification_Summary(pred,pred_prob,3)
```

```
<<<---------------------------------- Evaluating Naïve Bayes Classifier (NB) ----------------------------------->>>

Accuracy = 78.2%
F1 Score = 78.3%

 Confusiton Matrix:
 [[27  5  4]
 [ 4 26  2]
 [ 2  5 26]]

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.75      0.78        36
           1       0.72      0.81      0.76        32
           2       0.81      0.79      0.80        33

    accuracy                           0.78       101
   macro avg       0.78      0.78      0.78       101
weighted avg       0.79      0.78      0.78       101
```



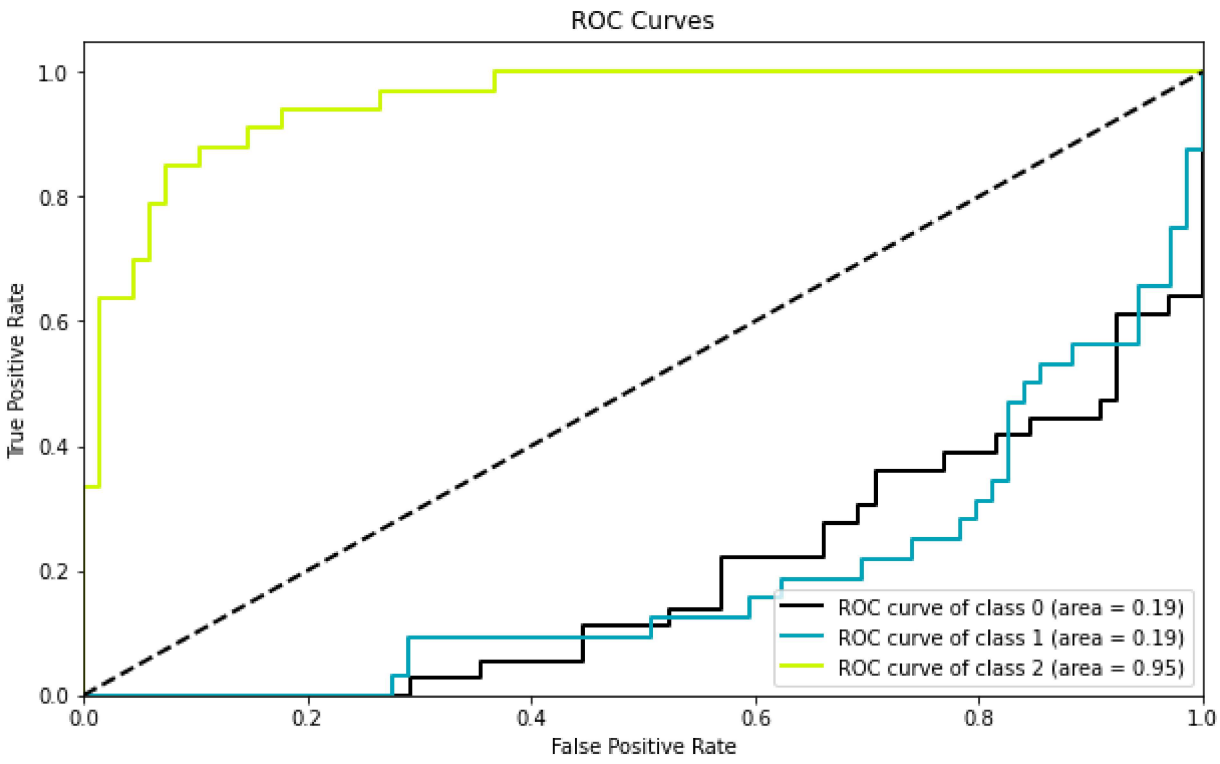
5. Support Vector Machine Classfier:

In [140…

```
# Building Support Vector Machine Classifier

SVM_model = SVC(probability=True).fit(Train_X_std, Train_Y)

svm_param = {"C": [.01, .1, 1, 5, 10, 100],
             "gamma": [.01, .1, 1, 5, 10, 100],
             "kernel": ["rbf"],
             "random_state": [1]}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(SVM_model, svm_param, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

SVM = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = SVM.predict(Test_X_std)
pred_prob = SVM.predict_proba(Test_X_std)
Classification_Summary(pred,pred_prob,4)
```

```
<<<---------------------------------- Evaluating Support Vector Machine (SVM) ---------------------------------->>>

Accuracy = 31.7%
F1 Score = 15.2%

 Confusiton Matrix:
```

```
 [[ 0 36  0]
 [ 0 32  0]
 [ 0 33  0]]

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        36
           1       0.32      1.00      0.48        32
           2       0.00      0.00      0.00        33

    accuracy                           0.32       101
   macro avg       0.11      0.33      0.16       101
weighted avg       0.10      0.32      0.15       101
```



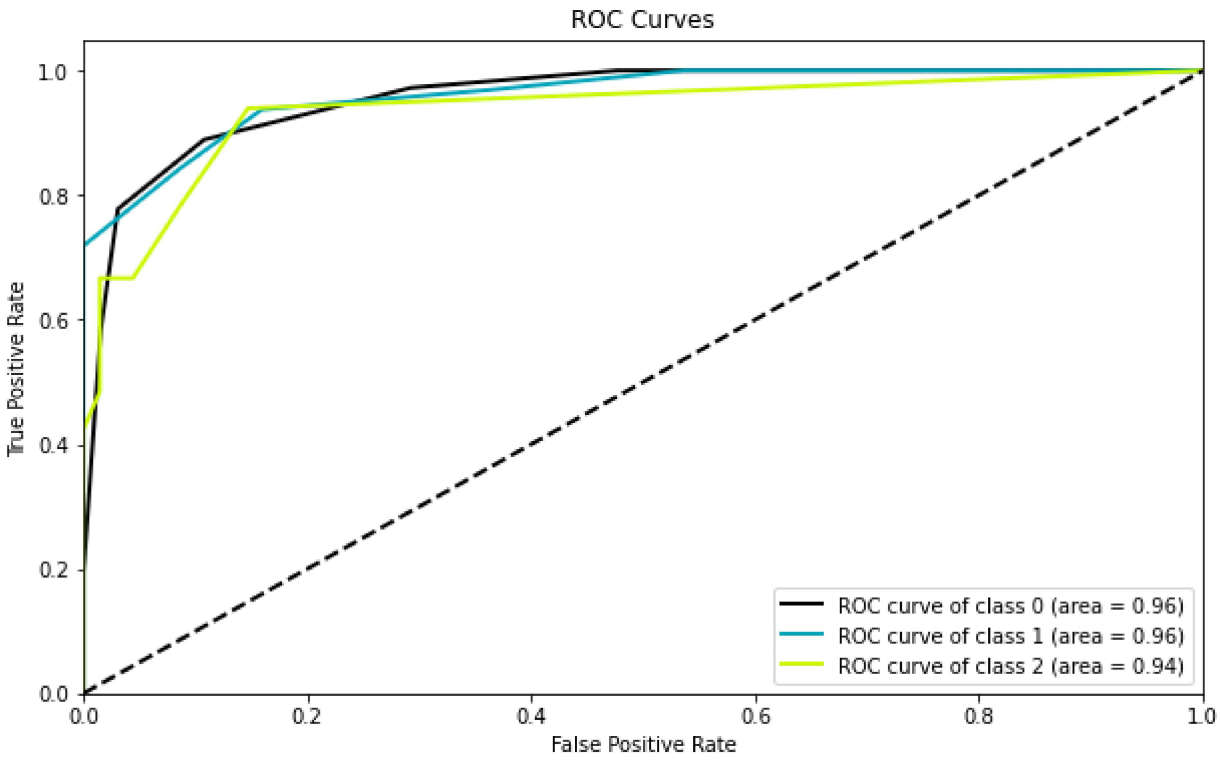
---

6. K-Nearest Neighbours Classfier:

In [141…

```
# Building K-Neareset Neighbours Classifier

KNN_model = KNeighborsClassifier()

knn_param = {"n_neighbors": [i for i in range(1,30,5)],
             "weights": ["uniform", "distance"],
             "algorithm": ["ball_tree", "kd_tree", "brute"],
             "leaf_size": [1, 10, 30],
             "p": [1,2]}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(KNN_model, knn_param, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

KNN = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = KNN.predict(Test_X_std)
pred_prob = KNN.predict_proba(Test_X_std)
Classification_Summary(pred,pred_prob,5)
```

```
<<<---------------------------------- Evaluating K Nearest Neighbours (KNN) ----------------------------------->>>

Accuracy = 80.2%
F1 Score = 80.10000000000001%

 Confusiton Matrix:
 [[32  4  0]
 [ 4 27  1]
 [ 6  5 22]]

Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.89      0.82        36
           1       0.75      0.84      0.79        32
           2       0.96      0.67      0.79        33

    accuracy                           0.80       101
   macro avg       0.82      0.80      0.80       101
weighted avg       0.82      0.80      0.80       101
```



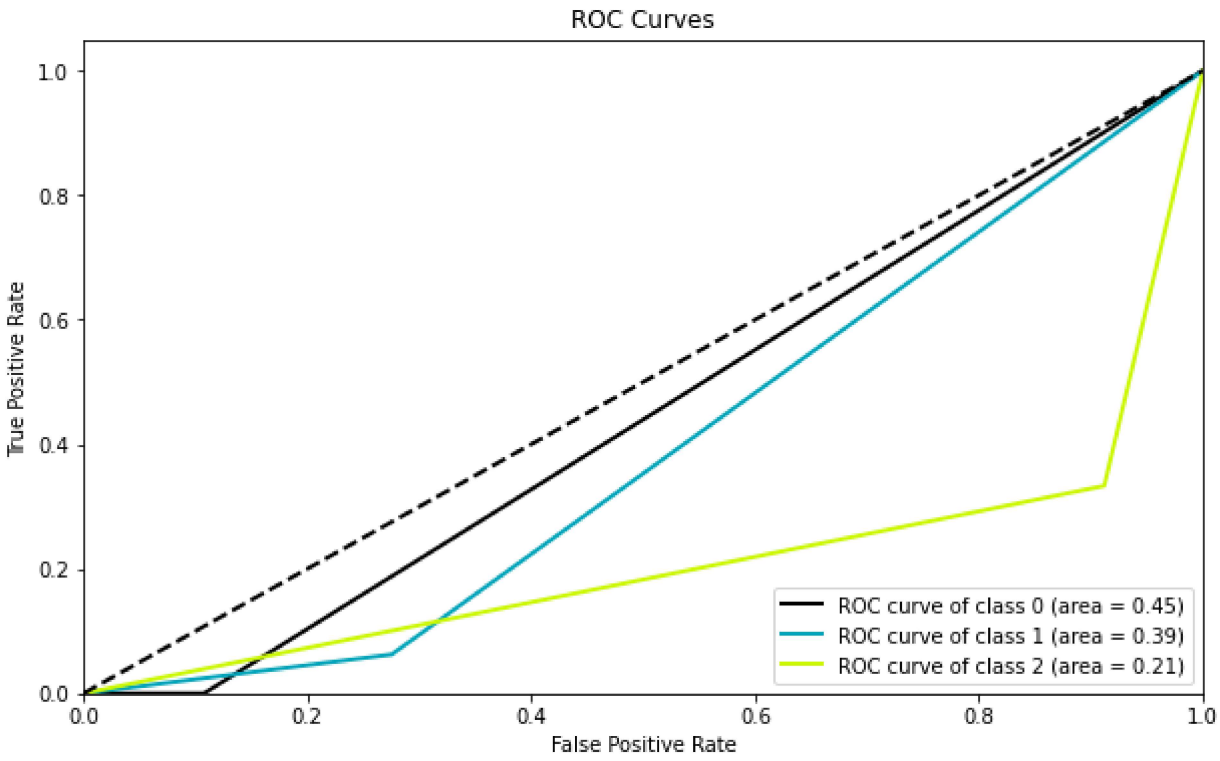
---

7. Gradient Boosting Classfier:

In [142…

```
# Building Gradient Boosting Classifier

GB_model = GradientBoostingClassifier().fit(Train_X_std, Train_Y)
param_dist = {
    "n_estimators":[5,20,100,500],
    "max_depth":[1,3,5,7,9],
    "learning_rate":[0.01,0.1,1,10,100]
}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(GB_model, param_dist, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

GB = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = GB.predict(Test_X_std)
pred_prob = GB.predict_proba(Test_X_std)
Classification_Summary(pred,pred_prob,6)
```

```
<<<---------------------------------- Evaluating Gradient Boosting (GB) ---------------------------------->>>

Accuracy = 12.9%
F1 Score = 9.2%

 Confusiton Matrix:
[[ 0  4 32]
 [ 0  2 30]
 [ 7 15 11]]

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        36
           1       0.10      0.06      0.08        32
           2       0.15      0.33      0.21        33

    accuracy                           0.13       101
   macro avg       0.08      0.13      0.09       101
weighted avg       0.08      0.13      0.09       101
```



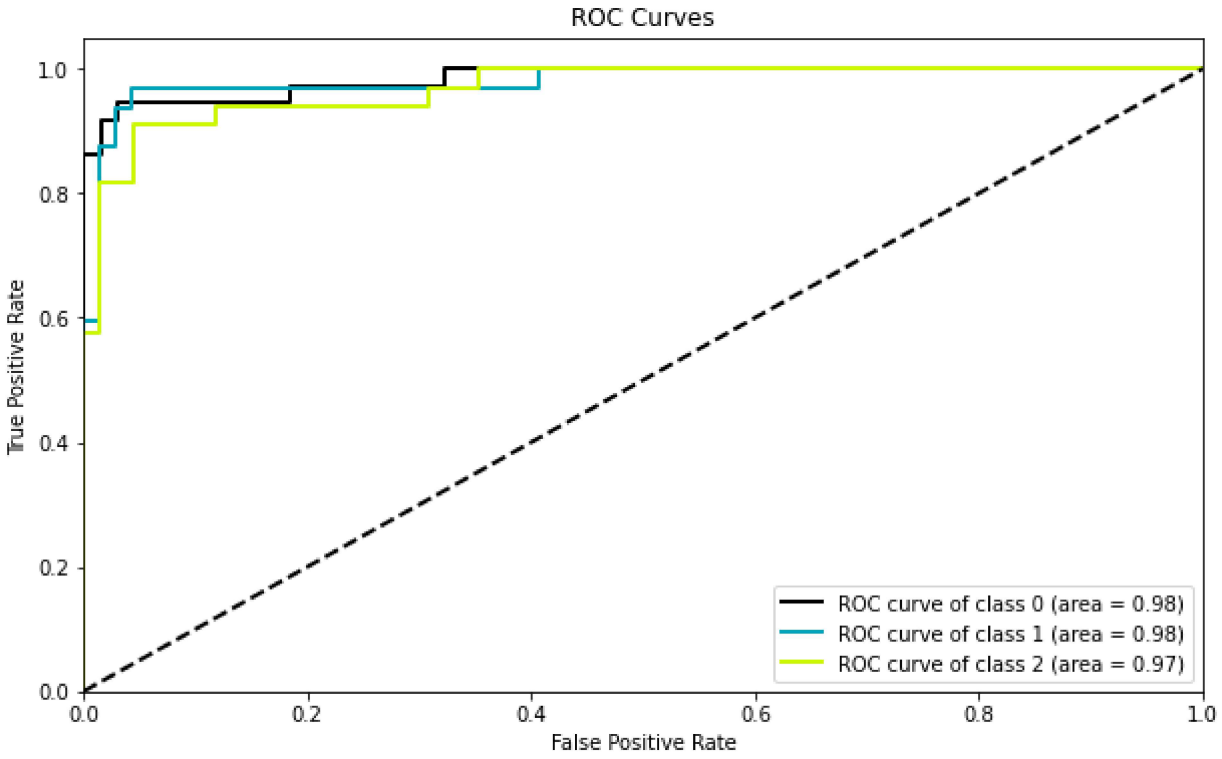
---

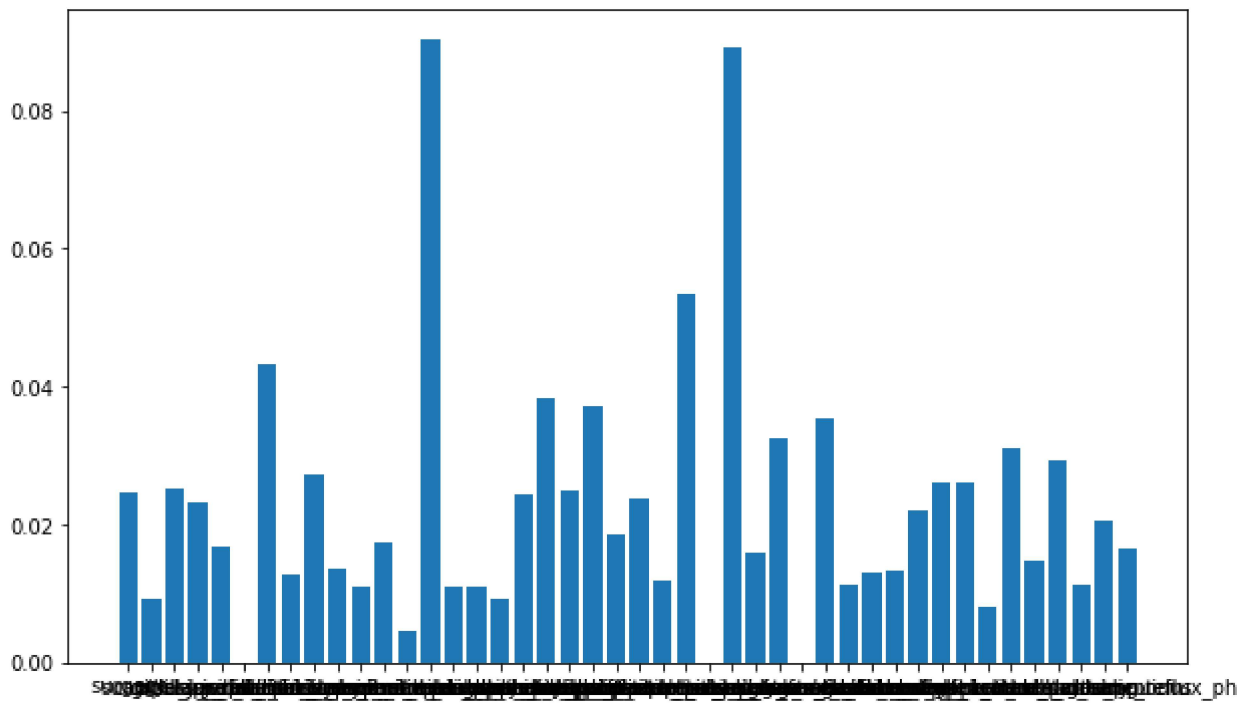
8. Extreme Gradient Boosting Classfier:

In [143…

```
# Building Extreme Gradient Boosting Classifier

XGB_model = XGBClassifier().fit(Train_X_std, Train_Y)

param_dist = {
 "learning_rate" : [0.05,0.10,0.15,0.20,0.25,0.30],
 "max_depth" : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma": [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]
}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

RCV = RandomizedSearchCV(XGB_model, param_dist, n_iter=50, scoring='roc_auc', n_jobs=-1, cv=5, random_state=1)

XGB = RCV.fit(Train_X_std, Train_Y).best_estimator_
pred = XGB.predict(Test_X_std)
pred_prob = XGB.predict_proba(Test_X_std)
Classification_Summary(pred,pred_prob,7)

plt.bar( Train_X_std.columns,XGB.feature_importances_,)
plt.show()
```

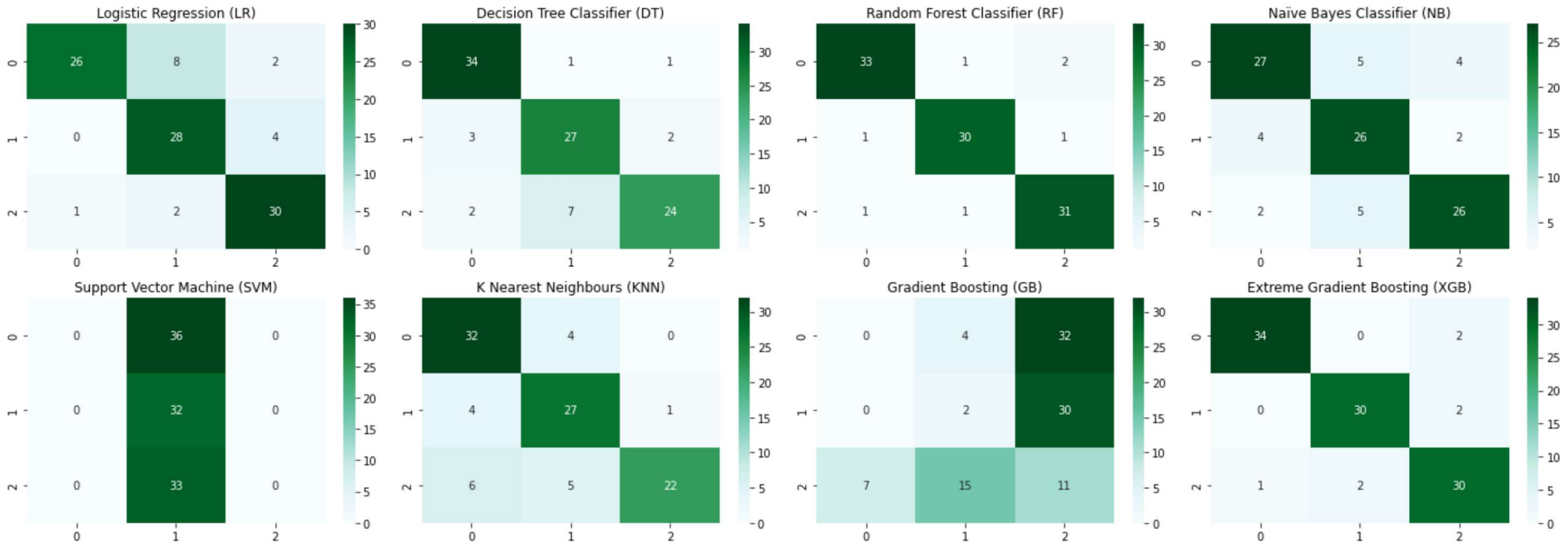
```
[23:50:54] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, t
he default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval
_metric if you'd like to restore the old behavior.
[23:51:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, t
he default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval
_metric if you'd like to restore the old behavior.
<<<---------------------------------- Evaluating Extreme Gradient Boosting (XGB) ----------------------------------->>>

Accuracy = 93.10000000000001%
F1 Score = 93.10000000000001%

 Confusiton Matrix:
 [[34  0  2]
 [ 0 30  2]
 [ 1  2 30]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.94      0.96        36
           1       0.94      0.94      0.94        32
           2       0.88      0.91      0.90        33

    accuracy                           0.93       101
   macro avg       0.93      0.93      0.93       101
weighted avg       0.93      0.93      0.93       101
```





In [144…

```
#Plotting Confusion-Matrix of all the predictive Models

def plot_cm(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred, labels=np.unique(y_true))
    cm_sum = np.sum(cm, axis=1, keepdims=True)
    cm_perc = cm / cm_sum.astype(float) * 100
    annot = np.empty_like(cm).astype(str)
    nrows, ncols = cm.shape
    for i in range(nrows):
        for j in range(ncols):
            c = cm[i, j]
            p = cm_perc[i, j]
            if i == j:
                s = cm_sum[i]
                annot[i, j] = '%.1f%%\n%d/%d' % (p, c, s)
            elif c == 0:
                annot[i, j] = ''
            else:
                annot[i, j] = '%.1f%%\n%d' % (p, c)
    cm = pd.DataFrame(cm, index=np.unique(y_true), columns=np.unique(y_true))
    cm.columns=labels
    cm.index=labels
    cm.index.name = 'Actual'
    cm.columns.name = 'Predicted'
    #fig, ax = plt.subplots()
    sns.heatmap(cm, annot=annot, fmt='')# cmap= "GnBu"

def conf_mat_plot(all_models):
    plt.figure(figsize=[20,3.5*math.ceil(len(all_models)*len(labels)/14)])

    for i in range(len(all_models)):
        if len(labels)<=4:
            plt.subplot(2,4,i+1)
        else:
            plt.subplot(math.ceil(len(all_models)/3),3,i+1)
        pred = all_models[i].predict(Test_X_std)
        #plot_cm(Test_Y, pred)
        sns.heatmap(confusion_matrix(Test_Y, pred), annot=True, cmap='BuGn', fmt='.0f') #vmin=0,vmax=5
        plt.title(Evaluation_Results.index[i])
    plt.tight_layout()
    plt.show()

conf_mat_plot([LR,DT,RF,NB,SVM,KNN,GB,XGB])
```



In [148…

```
#Comparing all the models Scores

#plt.figure(figsize=[12,5])
sns.heatmap(Evaluation_Results, annot=True, vmin=50, vmax=98, cmap='BuGn', fmt='.1f')
plt.show()
```

Insights: For the current problem statement, it is more important to focus on the F1 score. We can note from the above heatmap that the Random Forest & Extreme Gradient Boosting Model Performed well on the current dataset...

---

7. Project Outcomes & Conclusions

Here are some of the key outcomes of the project:

- The Dataset was quiet small totalling around 300 samples & after preprocessing 3.9% of the datasamples were dropped.
- The samples were slightly imbalanced after processing, hence SMOTE Technique was applied on the data to balance the classes, adding 30.4% more samples to the dataset.
- Visualising the distribution of data & their relationships, helped us to get some insights on the relationship between the feature-set.
- Feature Selection/Eliminination was carried out and appropriate features were shortlisted.
- Testing multiple algorithms with fine-tuning hyperparamters gave us some understanding on the model performance for various algorithms on this specific dataset.
- The XG-Boosting & Random Forest Classifier performed exceptionally well on the current dataset, considering F1-Score as the key-metric.
- Yet it wise to also consider simpler model like Logistic Regression as it is more generalisable & is computationally less expensive, but comes at the cost of slight misclassifications.

In [ ]:

```
<<<----------------------------------------------------------THE END----------------------------------------------------------
```