In [2]:
```
import pandas as pd
import os
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from scipy import stats
%matplotlib inline
sns.set_style('darkgrid')
%matplotlib inline
from sklearn.preprocessing import MinMaxScaler
from scipy.stats import zscore
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn import model_selection
import warnings
warnings.filterwarnings("ignore")



import imblearn
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTENC
from imblearn.over_sampling import SMOTE
```

Part A - 30 Marks

• DOMAIN: Medical

• CONTEXT:

Medical research university X is undergoing a deep research on patients with certain conditions. University has an internal AI team. Due to confidentiality the patient's details and the conditions are masked by the client by providing different datasets to the AI team for developing a AIML model which can predict the condition of the patient depending on the received test results.

• **DATA DESCRIPTION:**

The data consists of biomechanics features of the patients according to their current conditions. Each patient is represented in the data set by six biomechanics attributes derived from the shape and orientation of the condition to their body part.

• **PROJECT OBJECTIVE:**

To Demonstrate the ability to fetch, process and leverage data to generate useful predictions by training Supervised Learning algorithms.

• **STEPS AND TASK [30 Marks]:**

1. Data Understanding: [5 Marks]

1.A.Read all the 3 CSV files as DataFrame and store them into 3 separate variables. [1 Mark]

Answer

In [3]:
```
df1_normal=pd.read_csv("C:\\Users\\CHIRAG\\Desktop\\AIML project - Supervised learning\\PART 1 CSV\\Normal.csv")
```

In [4]:
```
df1_normal.head()
```

Out[4]:

	P_incidence	P_tilt	L_angle	S_slope	P_radius	S_Degree	Class
0	38.505273	16.964297	35.112814	21.540976	127.632875	7.986683	Normal
1	54.920858	18.968430	51.601455	35.952428	125.846646	2.001642	Normal
2	44.362490	8.945435	46.902096	35.417055	129.220682	4.994195	Normal
3	48.318931	17.452121	48.000000	30.866809	128.980308	-0.910941	Normal
4	45.701789	10.659859	42.577846	35.041929	130.178314	-3.388910	Normal

In [5]:
```
df2_typeH=pd.read_csv("C:\\Users\\CHIRAG\\Desktop\\AIML project - Supervised learning\\PART 1 CSV\\Type_H.csv")
```

In [6]:
```
df2_typeH.head()
```

Out[6]:

	P_incidence	P_tilt	L_angle	S_slope	P_radius	S_Degree	Class
0	63.027817	22.552586	39.609117	40.475232	98.672917	-0.254400	Type_H
1	39.056951	10.060991	25.015378	28.995960	114.405425	4.564259	Type_H
2	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	Type_H
3	69.297008	24.652878	44.311238	44.644130	101.868495	11.211523	Type_H
4	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	Type_H

In [7]:
```
df3_typeS=pd.read_csv("C:\\Users\\CHIRAG\\Desktop\\AIML project - Supervised learning\\PART 1 CSV\\Type_S.csv")
```

In [8]:
```
df3_typeS.head()
```

Out[8]:

	P_incidence	P_tilt	L_angle	S_slope	P_radius	S_Degree	Class
0	74.377678	32.053104	78.772013	42.324573	143.560690	56.125906	Type_S
1	89.680567	32.704435	83.130732	56.976132	129.955476	92.027277	Type_S
2	44.529051	9.433234	52.000000	35.095817	134.711772	29.106575	Type_S
3	77.690577	21.380645	64.429442	56.309932	114.818751	26.931841	Type_S
4	76.147212	21.936186	82.961502	54.211027	123.932010	10.431972	Type_S

here we have read all three dataframes and stored in different variable names df1,df2,and df3.

1.B. Print Shape and columns of all the 3 DataFrames. [1 Mark]

Answer

In [9]:
```
# reading shape of the dataframe
df1_normal.shape
```

Out[9]:
```
(100, 7)
```

.we have 100 raws and 7 columns for data frame df1_normal

the name of the columns in df1_normal is as follows

In [10]:
```
print("the name of the columns in df1_normal is as folows")
df1_normal.columns
```

Out[10]:
```
the name of the columns in df1_normal is as folows
Index(['P_incidence', 'P_tilt', 'L_angle', 'S_slope', 'P_radius', 'S_Degree',
       'Class'],
      dtype='object')
```

for data set df2_typeH

In [11]:
```
df2_typeH.shape
```

Out[11]:
```
(60, 7)
```

.we have 60 raws and 7 columns for dataframe df2_typeH

The name of the columns in df2_typeH is as follows

In [12]:
```
df2_typeH.columns
```

Out[12]:
```
Index(['P_incidence', 'P_tilt', 'L_angle', 'S_slope', 'P_radius', 'S_Degree',
       'Class'],
      dtype='object')
```

for dataset df3_typeS

In [13]:
```
df3_typeS.shape
```

Out[13]:
```
(150, 7)
```

.we have 150 raws and 7 columns for dataframe df3_typeS

The name of the columns in df3_typeS is as follows

In [14]:
```
df3_typeS.columns
```

Out[14]:
```
Index(['P_incidence', 'P_tilt', 'L_angle', 'S_slope', 'P_radius', 'S_Degree',
       'Class'],
      dtype='object')
```

1.C. Compare Column names of all the 3 DataFrames and clearly write observations. [1 Mark]

In [15]:
```
df1_normal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   P_incidence  100 non-null    float64
 1   P_tilt       100 non-null    float64
 2   L_angle      100 non-null    float64
 3   S_slope      100 non-null    float64
 4   P_radius     100 non-null    float64
 5   S_Degree     100 non-null    float64
 6   Class        100 non-null    object 
dtypes: float64(6), object(1)
memory usage: 5.6+ KB
```

In [16]:
```
df2_typeH.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   P_incidence  60 non-null     float64
 1   P_tilt       60 non-null     float64
 2   L_angle      60 non-null     float64
 3   S_slope      60 non-null     float64
 4   P_radius     60 non-null     float64
 5   S_Degree     60 non-null     float64
 6   Class        60 non-null     object 
dtypes: float64(6), object(1)
memory usage: 3.4+ KB
```

In [17]:
```
df3_typeS.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   P_incidence  150 non-null    float64
 1   P_tilt       150 non-null    float64
 2   L_angle      150 non-null    float64
 3   S_slope      150 non-null    float64
 4   P_radius     150 non-null    float64
 5   S_Degree     150 non-null    float64
 6   Class        150 non-null    object 
dtypes: float64(6), object(1)
memory usage: 8.3+ KB
```

Answer

> From the observation it states that

> All data frame has similar column name

> Number of columns are also same that is 7

> Outof 7 column 6 has float type of data and 1 column has object type that is string data type in all dataframe

> name of columns that contains float type object is as follows

```
['P_incidence', 'P_tilt', 'L_angle', 'S_slope', 'P_radius', 'S_Degree']
```

> name of the column that has object type data name is 'Class' for all dataframes

1.D. Print DataTypes of all the 3 DataFrames. [1 Mark]

In [18]:
```
df1_normal.dtypes
```

Out[18]:
```
P_incidence    float64
P_tilt         float64
L_angle        float64
S_slope        float64
P_radius       float64
S_Degree       float64
Class           object
dtype: object
```

> for df1_normal dataframe 6 column has float data type and one column has object string type data

In [19]:
```
df2_typeH.dtypes
```

Out[19]:
```
P_incidence    float64
P_tilt         float64
L_angle        float64
S_slope        float64
P_radius       float64
S_Degree       float64
Class           object
dtype: object
```

> for df2_typeH dataframe 6 column has float data type and one column has object string type data

In [20]:
```
df3_typeS.dtypes
```

Out[20]:
```
P_incidence    float64
P_tilt         float64
L_angle        float64
S_slope        float64
P_radius       float64
S_Degree       float64
Class           object
dtype: object
```

> for df3_typeS dataframe 6 column has float data type and one column has object string type data

1.E. Observe and share variation in 'Class' feature of all the 3 DaraFrames. [1 Mark]¶

In [21]:
```
df1_normal['Class'].value_counts()
```

Out[21]:
```
Class
Normal    73
Nrmal     27
Name: count, dtype: int64
```

In [22]:
```
df2_typeH['Class'].value_counts()
```

Out[22]:
```
Class
Type_H    37
type_h    23
Name: count, dtype: int64
```

In [23]:
```
df3_typeS['Class'].value_counts()
```

Out[23]:
```
Class
Type_S    133
tp_s       17
Name: count, dtype: int64
```

Answer

from observation

> dataframe df1_normal has two type of variation name is 'normal' has 73 value counts and 'Nrmal' has 27 value counts

> dataframe df2_typeH has two type of variation name is 'Type_H' has 37 value counts and 'type_h' has 23 value counts

> dataframe df3_typeS has two type of variation name is 'Type_S' has 133 value counts and 'tp_s' has 17 value counts

summary:

Here 1) tp_s and Type_S, 2) Normal and Nrmal,TypeH and 3) type h represents same class.

2. Data Preparation and Exploration: [5 Marks]

2.A. Unify all the variations in 'Class' feature for all the 3 DataFrames. [1 Marks]

In [24]:
```
df1_normal.loc[df1_normal['Class']=='Nrmal','Class']='Normal'
df1_normal['Class'].value_counts()
```

Out[24]:
```
Class
Normal    100
Name: count, dtype: int64
```

In [25]:
```
df2_typeH.loc[df2_typeH['Class']=='type_h','Class']='Type_H'
df2_typeH['Class'].value_counts()
```

Out[25]:
```
Class
Type_H    60
Name: count, dtype: int64
```

In [26]:
```
df3_typeS.loc[df3_typeS['Class']=='tp_s','Class']='Type_S'
df3_typeS['Class'].value_counts()
```

Out[26]:
```
Class
Type_S    150
Name: count, dtype: int64
```

Answer

here we unified all dataframe variations in 'Class' feature

2.B. Combine all the 3 DataFrames to form a single DataFrame [1 Marks]

In [27]:
```
df=pd.concat([df1_normal,df2_typeH,df3_typeS])
```

In [28]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 310 entries, 0 to 149
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   P_incidence  310 non-null    float64
 1   P_tilt       310 non-null    float64
 2   L_angle      310 non-null    float64
 3   S_slope      310 non-null    float64
 4   P_radius     310 non-null    float64
 5   S_Degree     310 non-null    float64
 6   Class        310 non-null    object 
dtypes: float64(6), object(1)
memory usage: 19.4+ KB
```

In [29]:
```
df.shape
```

Out[29]:
```
(310, 7)
```

Answer

from shape we can say that all data frame is combine in single variable name is df

2.C. Print 5 random samples of this DataFrame [1 Marks]

In [30]:
```
df.head()
```

Out[30]:

	P_incidence	P_tilt	L_angle	S_slope	P_radius	S_Degree	Class
0	38.505273	16.964297	35.112814	21.540976	127.632875	7.986683	Normal
1	54.920858	18.968430	51.601455	35.952428	125.846646	2.001642	Normal
2	44.362490	8.945435	46.902096	35.417055	129.220682	4.994195	Normal
3	48.318931	17.452121	48.000000	30.866809	128.980308	-0.910941	Normal
4	45.701789	10.659859	42.577846	35.041929	130.178314	-3.388910	Normal

Answer

first five samples are as above of dataframe df

2.D. Print Feature-wise percentage of Null values. [1 Mark]

In [31]:
```
df.isnull().sum()/df.shape[0]*100
```

Out[31]:
```
P_incidence    0.0
P_tilt         0.0
L_angle        0.0
S_slope        0.0
P_radius       0.0
S_Degree       0.0
Class          0.0
dtype: float64
```

Answer

There is no missing value in the dataset

2.E. Check 5-point summary of the new DataFrame. [1 Mark]

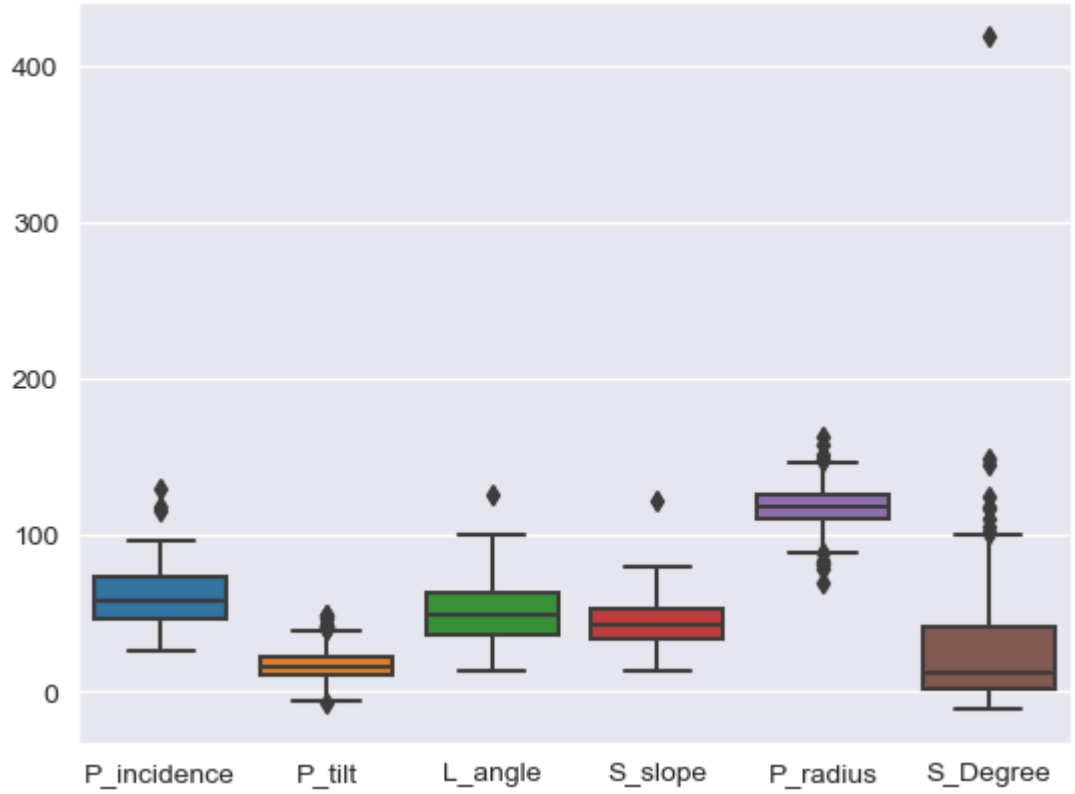
In [32]:
```
df.describe().T
```

Out[32]:

	count	mean	std	min	25%	50%	75%	max
P_incidence	310.0	60.496653	17.236520	26.147921	46.430294	58.691038	72.877696	129.834041
P_tilt	310.0	17.542822	10.008330	-6.554948	10.667069	16.357689	22.120395	49.431864
L_angle	310.0	51.930930	18.554064	14.000000	37.000000	49.562398	63.000000	125.742385
S_slope	310.0	42.953831	13.423102	13.366931	33.347122	42.404912	52.695888	121.429566
P_radius	310.0	117.920655	13.317377	70.082575	110.709196	118.268178	125.467674	163.071041
S_Degree	310.0	26.296694	37.559027	-11.058179	1.603727	11.767934	41.287352	418.543082

In [33]:
```
sns.boxplot(data=df)
```

Out[33]:
```
<Axes: >
```



Answer

> P_incidence:

Mean and Median are nearly equal .

Distribution might be normal. we have 75 % of values are less than 72 but maxiumum value is 129

> P_tilt:

Mean and median are nearly equal.

Distribution might be normal.

It contains negative values

75 % of values are less than 22 but maximum value is 49 so there might be little right skewness

> L_angle:

Mean and Median are nearly equal. There is no deviation.

Distribution might be normal

There might be few outliers because of the maximum value

> S_slope:

Mean and Median are nearly equal.

Towards the end there is little devation. 75% of values are lesser than 52 but maximum value is 121.

> P_radius:

Distribution might be normal.

There is no much Deviation.

> S_Degree:

Mean is greater than Median so there might be right skewness in the data .

We can see 75% of values are less than 41 but maximum value is 418 so there is obvious outliers in the data.

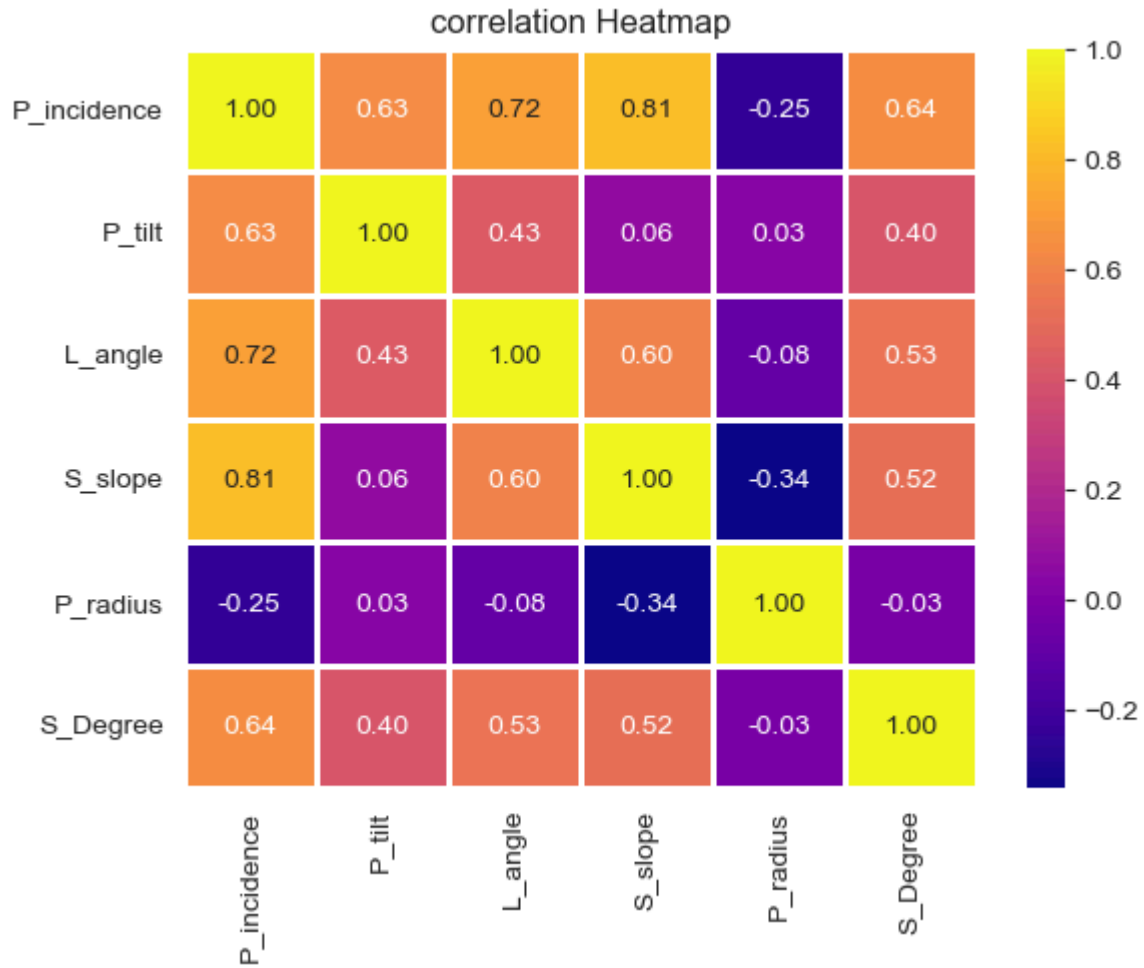
3. Data Analysis: [10 Marks]

3.A. Visualize a heatmap to understand correlation between all features [2 Marks]

In [34]:
```
df.columns
```

Out[34]:
```
Index(['P_incidence', 'P_tilt', 'L_angle', 'S_slope', 'P_radius', 'S_Degree',
       'Class'],
      dtype='object')
```

In [35]:
```
sns.heatmap(data=df[['P_incidence', 'P_tilt', 'L_angle', 'S_slope', 'P_radius', 'S_Degree']].corr(),annot=True,cmap='plasma',fmt=
plt.yticks(rotation = 0)
plt.xticks(rotation = 90)
plt.title('correlation Heatmap')
plt.show()
```



Answer

.positive correlation summary

> P_tilt and P_incidence has positive correlation

> L_large and P_incidence has Positive correlation

> L_large and P_tilt has moderate positive correlation

> P_tilt has no strong relation between L_angle,S*slope, and P* radius

> S_slope and P_incedednce has higest positive correlation

> Correlation between s_degree and p_incidence have high positive correlation.

. negatrive correlation summary

> S_degree and p_radius has negative correlation

> L angle and P_radius has neagtive correlation

> P_radius and S_degree has negative correlation

> S_slope and P_radius has higest negative correlation

3.B. Share insights on correlation. [2 Marks]

Answer

.positive correlation summary

> P_tilt and P_incidence has positive correlation with 63%(of data show positive relation)

> L_angle and P_incidence has Positive correlation with 72%

> L_angle and P_tilt has moderate positive correlation 43%

> P_tilt has no strong relation between,S*slope, and P* radius 6% and 3% approx

> S_slope and P_incedednce has higest positive correlation 81%

> Correlation between s_degree and p_incidence have high positive correlation. 64% same as P_tilt and P_incidence

. negatrive correlation summary

> S_degree and p_radius has slightlynegative correlation -0.026 that is 2.5% approx data shows negative correlation

> L_angle and P_radius has neagtive correlation -0.08

> S_degree and P_radius has negative correlation 2.6% -0.026

> S_slope and P_radius has higest negative correlation -0.34 that is 34%

> P_incidence and P_radius has negative correlation -0.25 that is 25%

> p_tilt and S slope has weak + ve correlation 0.062 ie 6.2%

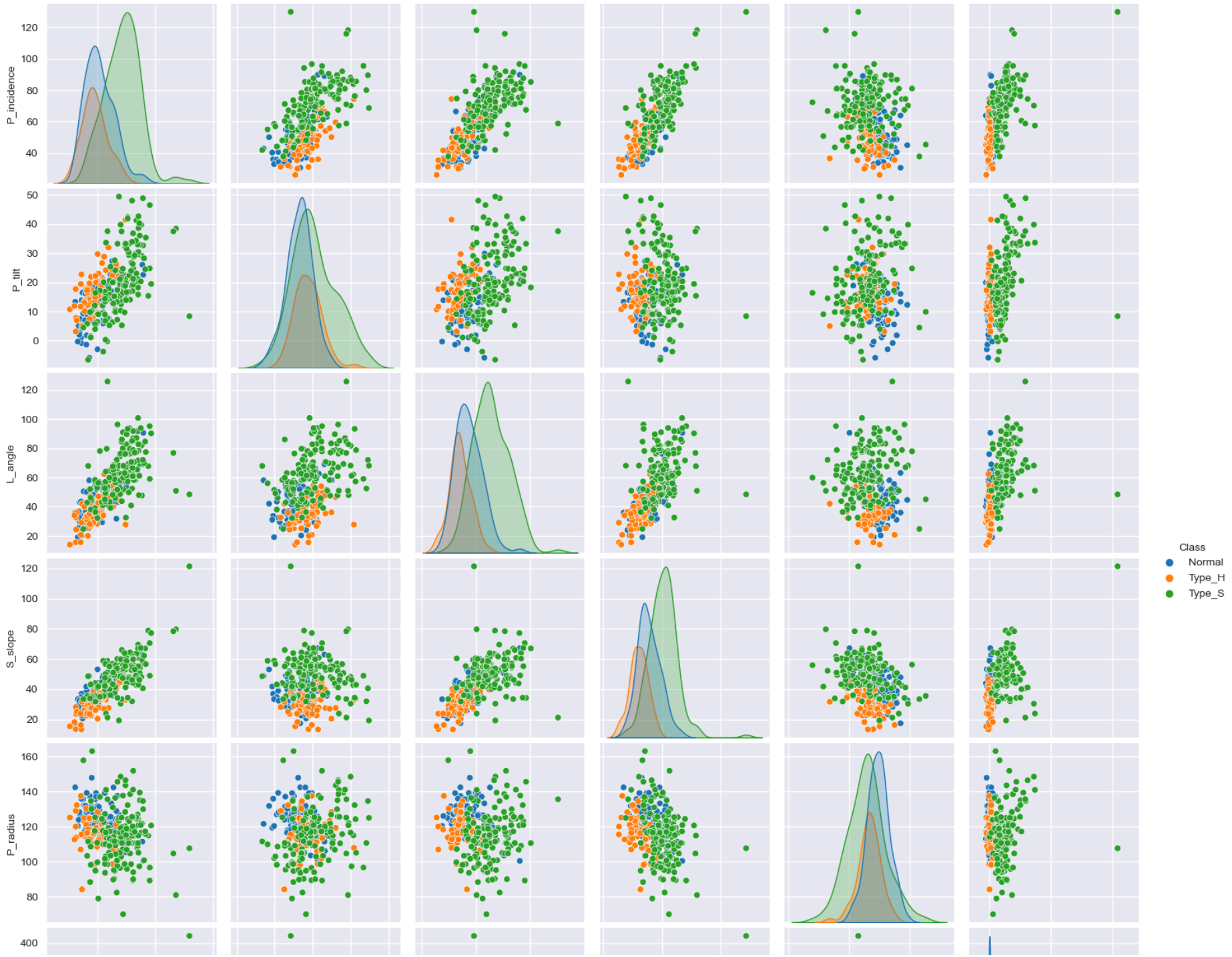
> p_tilt and P_radius has weak +ve correlation 0.033 is 3.3%

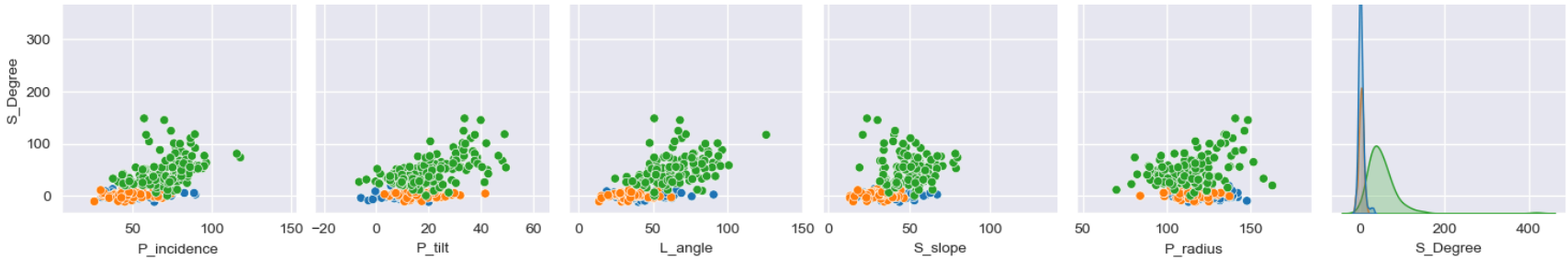
> L_angle and P_radius has weak -ve correlation 0.08 is 8%

3.C. Visualize a pairplot with 3 classes distinguished by colors and share insights. [2 Marks]

In [37]:
```
sns.pairplot(df,hue='Class')
```

Out[37]:
```
<seaborn.axisgrid.PairGrid at 0x28496b93350>
```


P_incidence
P_tilt
L_angle
S_slope
P_radius
Class
Normal
Type_H
Type_S



Answer

insights

Along the diagonal we can see distribution of variable for three claases are not same.We can prove that statistically as well¶

It is evident that type_s class is more compared to other two

Normal class has higher values compared to Type_H

Along the diagonal we can see the distribution of individual variable

P_incidence has postive realtionship with all variables except P_radius. Relationship is higher for S_slope and L_angle

P_tilt has Higher Relationship with P_incidence and L_angle.There is no Relationship with s_slope and p_radius

L_angle has postive Relationship with p_tilt,s_slope and s_degree. It has no Relationship with P_radius

s_slope has positive Relationship with L_angle and s_degree

p_radius has no Relationship with s_degree,p_tilt,l_angle.

S_degree has no strong positive Relationship with any of the variables.

In [38]:
```
class_summary=df.groupby('Class') #getting mean values of each class for all independent variables
class_summary.mean().reset_index()
```

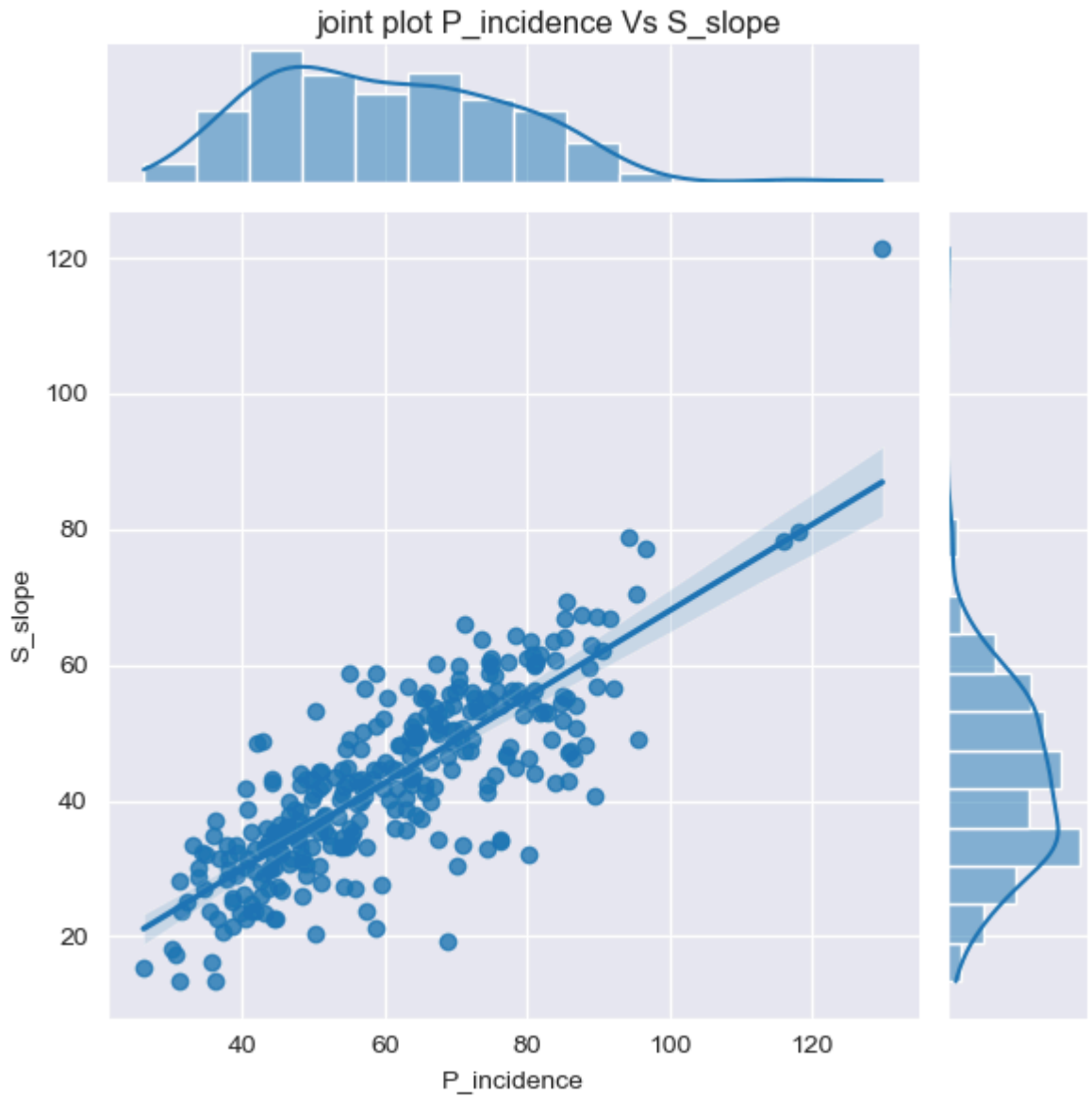
Out[38]:

	Class	P_incidence	P_tilt	L_angle	S_slope	P_radius	S_Degree
0	Normal	51.685244	12.821414	43.542605	38.863830	123.890834	2.186572
1	Type_H	47.638407	17.398795	35.463524	30.239612	116.474968	2.480251
2	Type_S	71.514224	20.748038	64.110108	50.766186	114.518810	51.896687

It is clear that s_Degree of Type_S contains larger values.

3.D. Visualize a jointplot for 'P_incidence' and 'S_slope' and share insights. [2 Marks]

In [40]:
```
p=sns.jointplot(data=df,x='P_incidence',y='S_slope',kind='reg')
p.fig.suptitle("joint plot P_incidence Vs S_slope")
p.fig.tight_layout()
p.fig.subplots_adjust(top=0.95) # Reduce plot to make room
```



Answer

insights

> from joint plot it is clear that there is positive correlation between P_incidence and S_slope

> S_slope increses with increse in value of P_incidence

> this data can be use for liner regreession model

> P_incidence data is slightlt rightly skewed distributed

> S_slope is slightly left skewed distributed

3.E. Visualize a boxplot to check distribution of the features and share insights. [2 Marks]

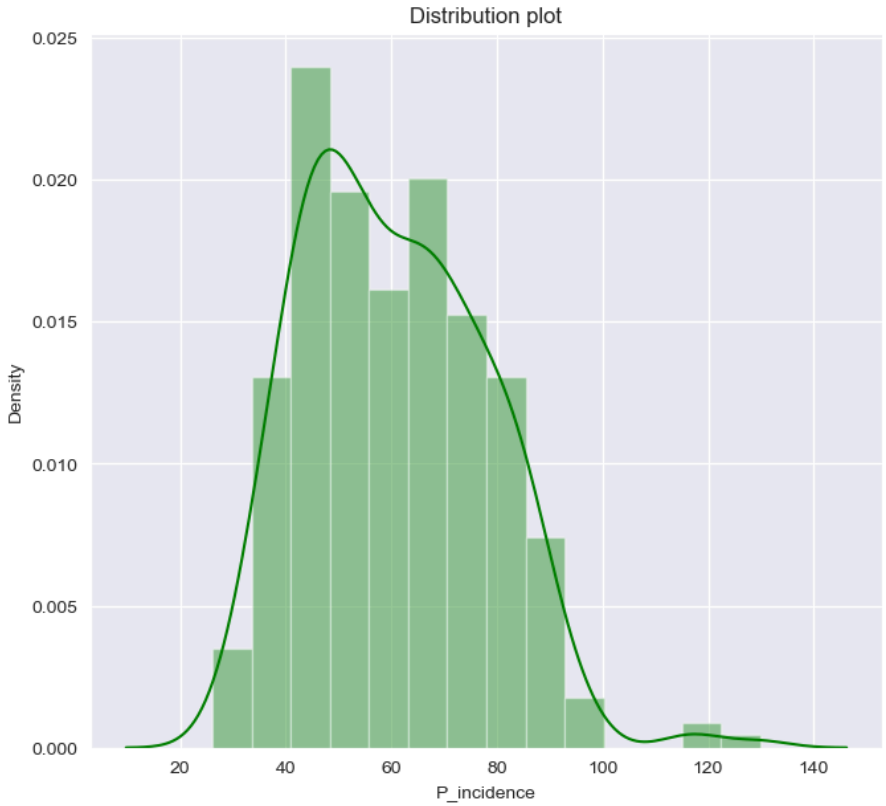
Answer

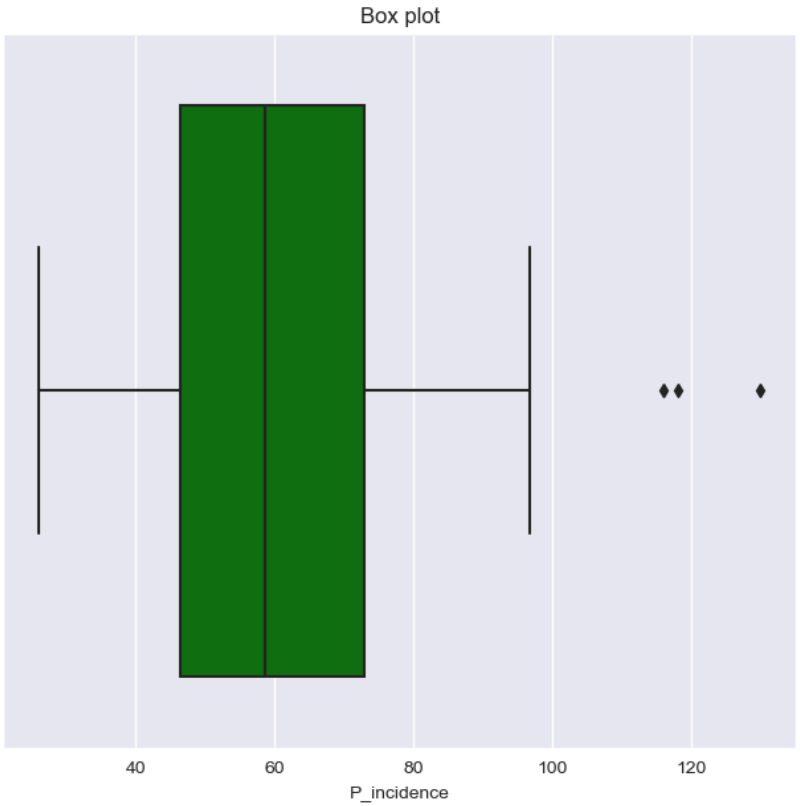
Box plot for Univariate Analysis

Distribution and outlier analysis of numerical variables

P_incidence

In [41]:
```
f, axes = plt.subplots(1, 2, figsize=(17,7))
sns.boxplot(x = 'P_incidence', data=df,  orient='h' , ax=axes[1],color='Green')
sns.distplot(df['P_incidence'],  ax=axes[0],color='Green')
axes[0].set_title('Distribution plot')
axes[1].set_title('Box plot')
plt.show()
#checking count of outliers.
q25,q75=np.percentile(df['P_incidence'],25),np.percentile(df['P_incidence'],75)
IQR=q75-q25
Threshold=IQR*1.5
lower,upper=q25-Threshold,q75+Threshold
Outliers=[i for i in df['P_incidence'] if i < lower or i > upper]
print('{} Total Number of outliers in P_incidence: {}'.format('\033[1m',len(Outliers)))
```





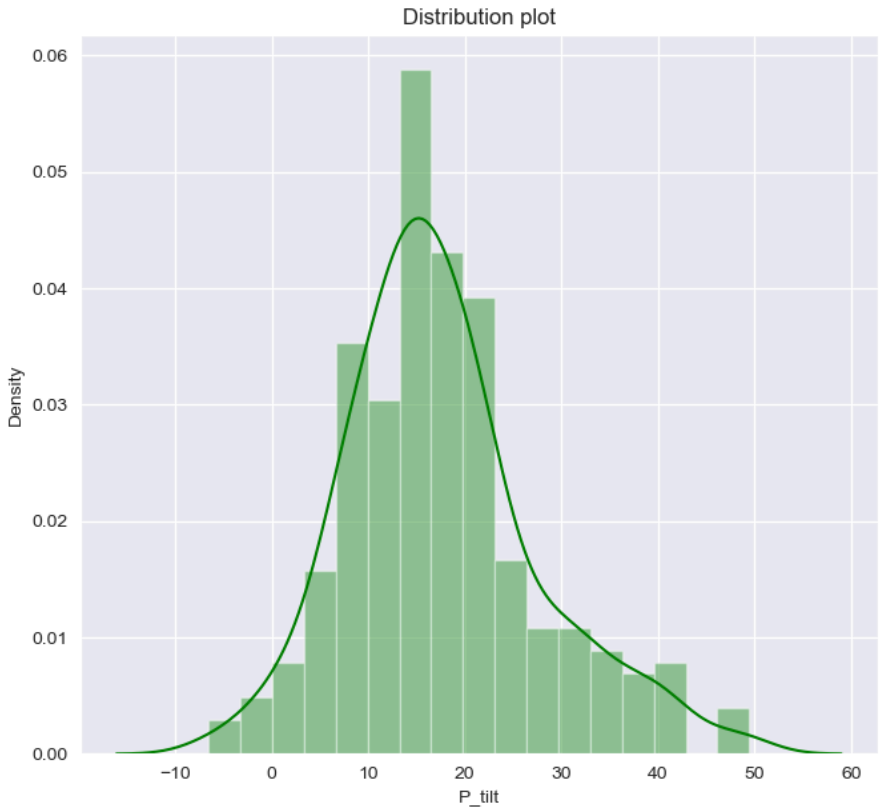
Total Number of outliers in P_incidence: 3

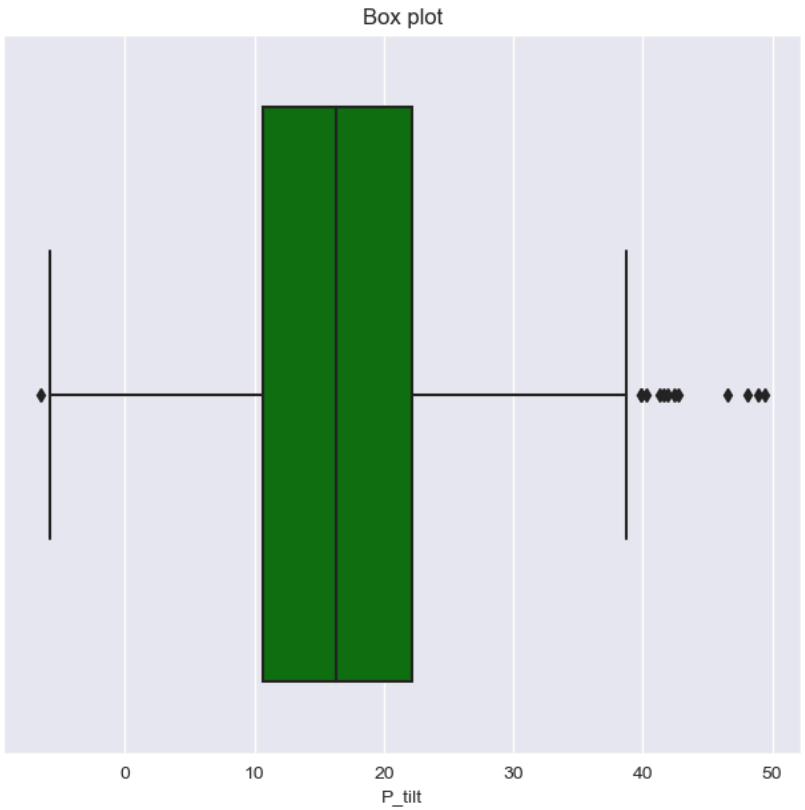
1. Normality is maintained with very less extreme values
2. We can see three outliers exists in the column

P_tilt

In [42]:
```
f, axes = plt.subplots(1, 2, figsize=(17,7))
sns.boxplot(x = 'P_tilt', data=df,  orient='h' , ax=axes[1],color='Green')
sns.distplot(df['P_tilt'],  ax=axes[0],color='Green')
axes[0].set_title('Distribution plot')
axes[1].set_title('Box plot')
plt.show()
#checking count of outliers.
q25,q75=np.percentile(df['P_tilt'],25),np.percentile(df['P_tilt'],75)
IQR=q75-q25
```

```
Threshold=IQR*1.5
lower,upper=q25-Threshold,q75+Threshold
Outliers=[i for i in df['P_tilt'] if i < lower or i > upper]
print('{} Total Number of outliers in P_tilt: {}'.format('\033[1m',len(Outliers)))
```



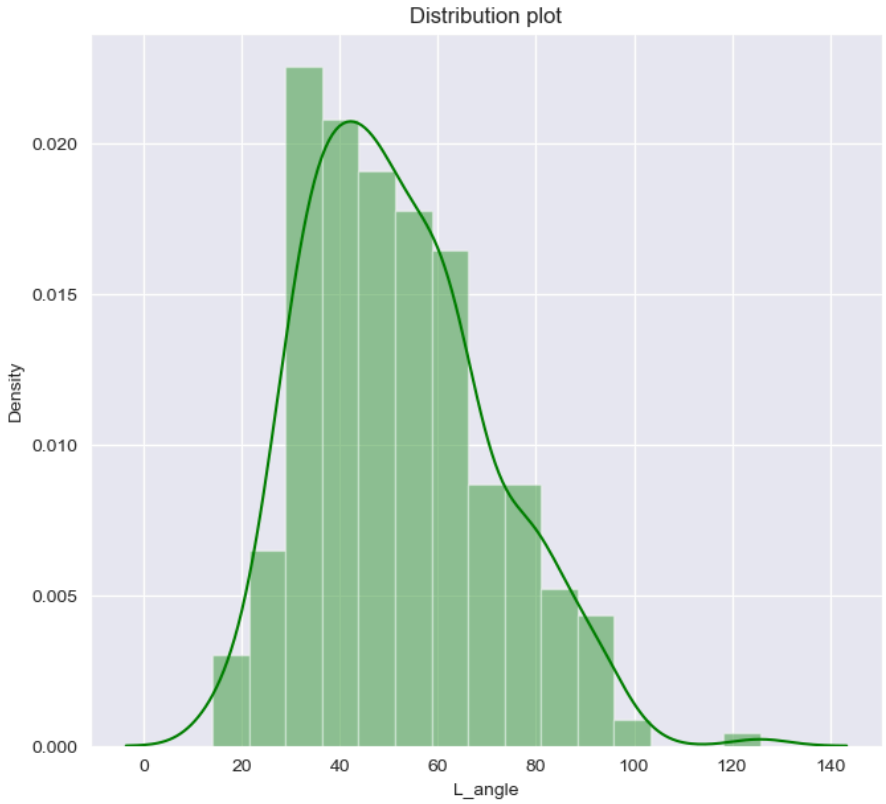


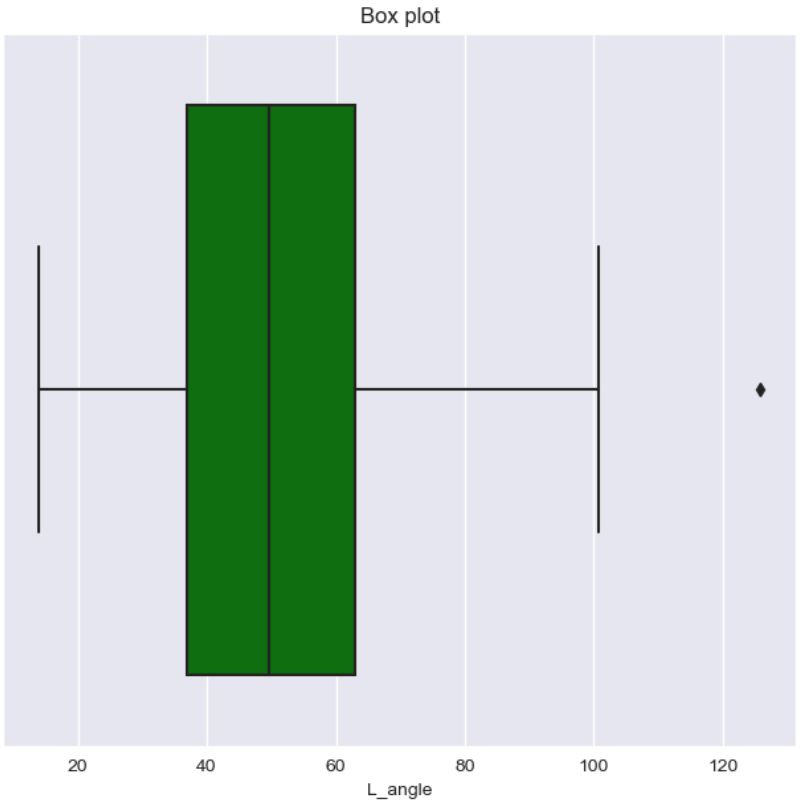
Total Number of outliers in P_tilt: 13

1. Data is Normally distributed and we can see one peakness in the center
2. It is has little skewness towards right side
3. We can see one outlier in negative end and few outliers in positive end.

L_angle

In [43]:
```
f, axes = plt.subplots(1, 2, figsize=(17,7))
sns.boxplot(x = 'L_angle', data=df,  orient='h' , ax=axes[1],color='Green')
sns.distplot(df['L_angle'],  ax=axes[0],color='Green')
axes[0].set_title('Distribution plot')
axes[1].set_title('Box plot')
plt.show()
#checking count of outliers.
q25,q75=np.percentile(df['L_angle'],25),np.percentile(df['L_angle'],75)
IQR=q75-q25
Threshold=IQR*1.5
lower,upper=q25-Threshold,q75+Threshold
Outliers=[i for i in df['L_angle'] if i < lower or i > upper]
print('{} Total Number of outliers in L_angle: {}'.format('\033[1m',len(Outliers)))
```





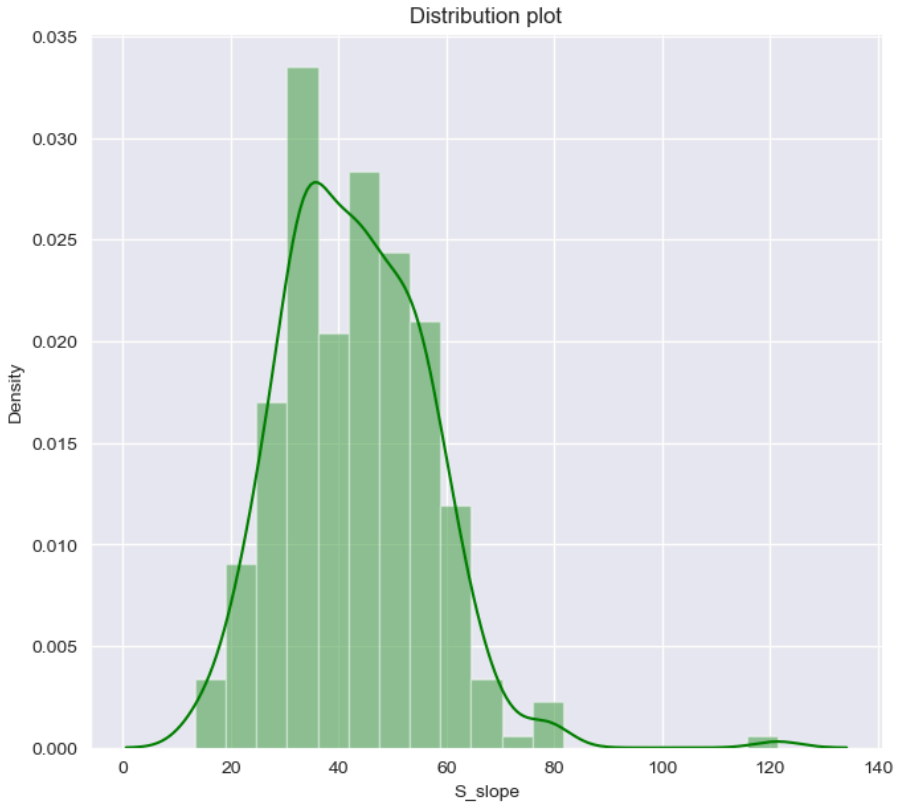
Total Number of outliers in L_angle: 1

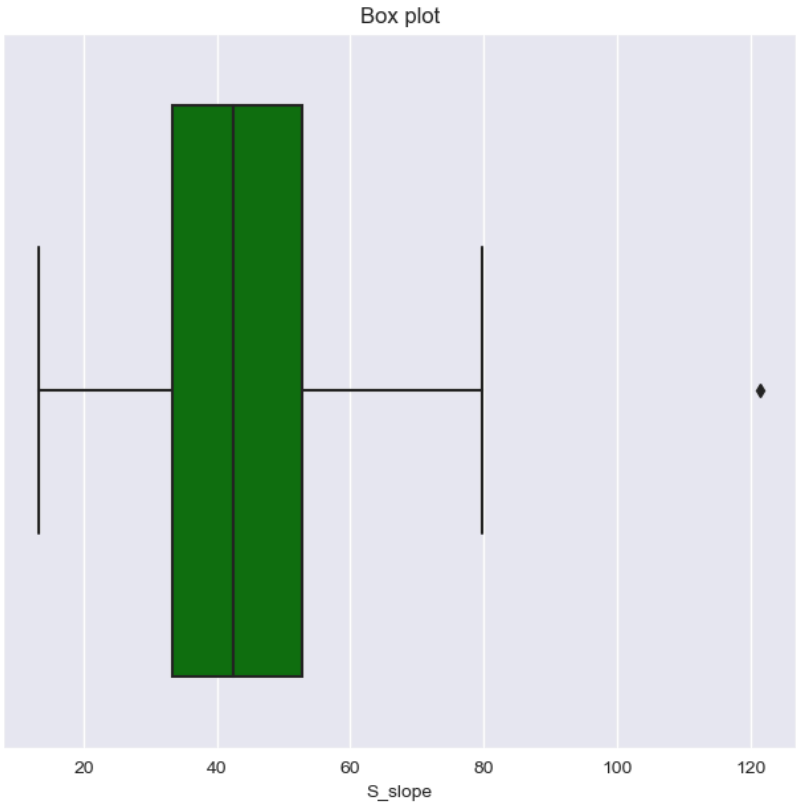
It is Normally distributed

Little right skewness because of one outlier

S_slope

In [44]:
```
f, axes = plt.subplots(1, 2, figsize=(17,7))
sns.boxplot(x = 'S_slope', data=df,  orient='h' , ax=axes[1],color='Green')
sns.distplot(df['S_slope'],  ax=axes[0],color='Green')
axes[0].set_title('Distribution plot')
axes[1].set_title('Box plot')
plt.show()
#checking count of outliers.
q25,q75=np.percentile(df['S_slope'],25),np.percentile(df['S_slope'],75)
IQR=q75-q25
Threshold=IQR*1.5
lower,upper=q25-Threshold,q75+Threshold
Outliers=[i for i in df['S_slope'] if i < lower or i > upper]
print('{} Total Number of outliers in S_slope: {}'.format('\033[1m',len(Outliers)))
```





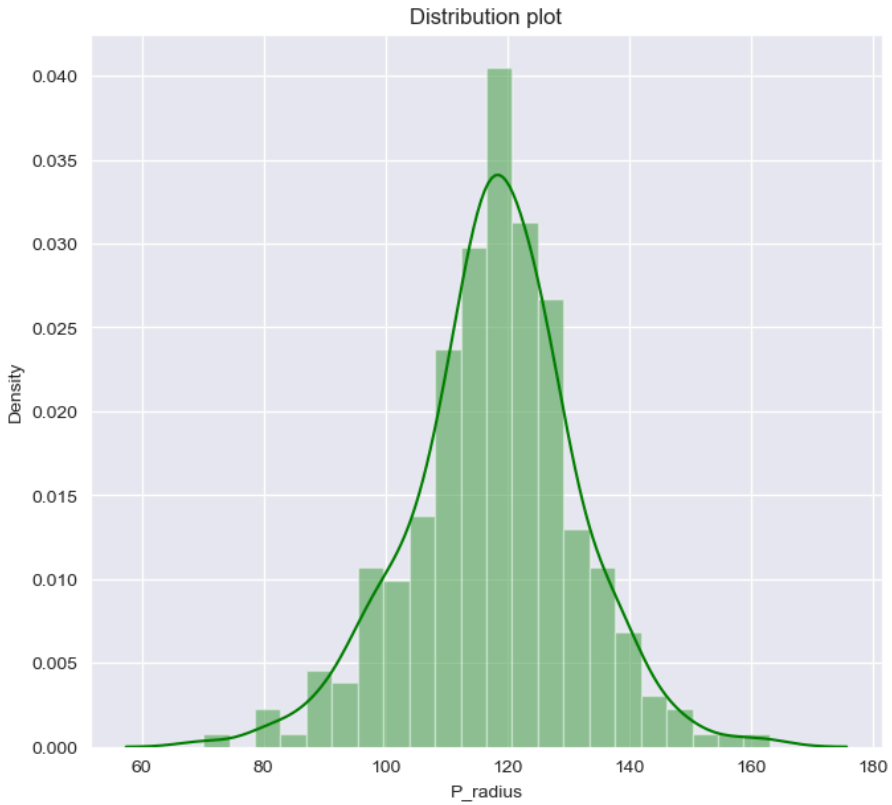
Total Number of outliers in S_slope: 1

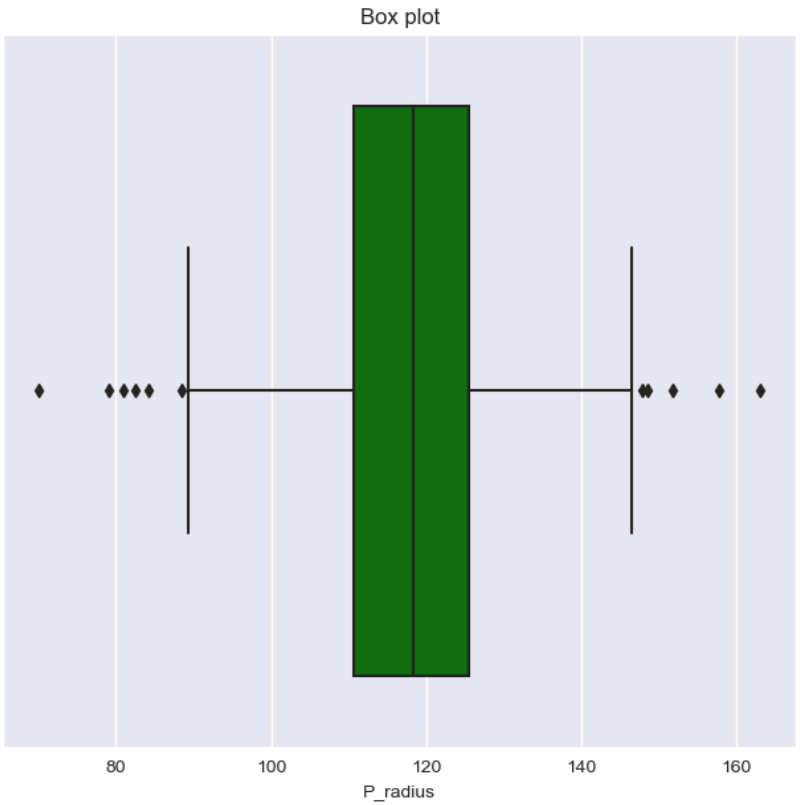
There is right skewness due to one outlier

P_radius

In [45]:
```
f, axes = plt.subplots(1, 2, figsize=(17,7))
sns.boxplot(x = 'P_radius', data=df,  orient='h' , ax=axes[1],color='Green')
sns.distplot(df['P_radius'],  ax=axes[0],color='Green')
axes[0].set_title('Distribution plot')
axes[1].set_title('Box plot')
plt.show()
#checking count of outliers.
q25,q75=np.percentile(df['P_radius'],25),np.percentile(df['P_radius'],75)
IQR=q75-q25
Threshold=IQR*1.5
lower,upper=q25-Threshold,q75+Threshold
```

```
Outliers=[i for i in df['P_radius'] if i < lower or i > upper]
print('{} Total Number of outliers in P_radius: {}'.format('\033[1m',len(Outliers)))
```





Total Number of outliers in P_radius: 11

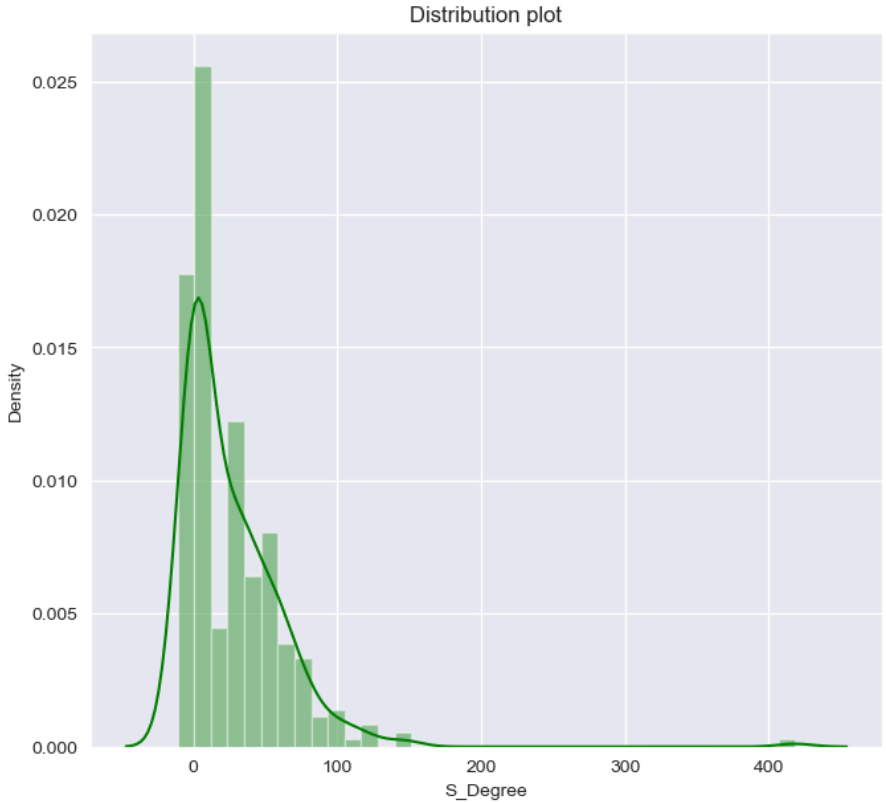
Data is normally distributed

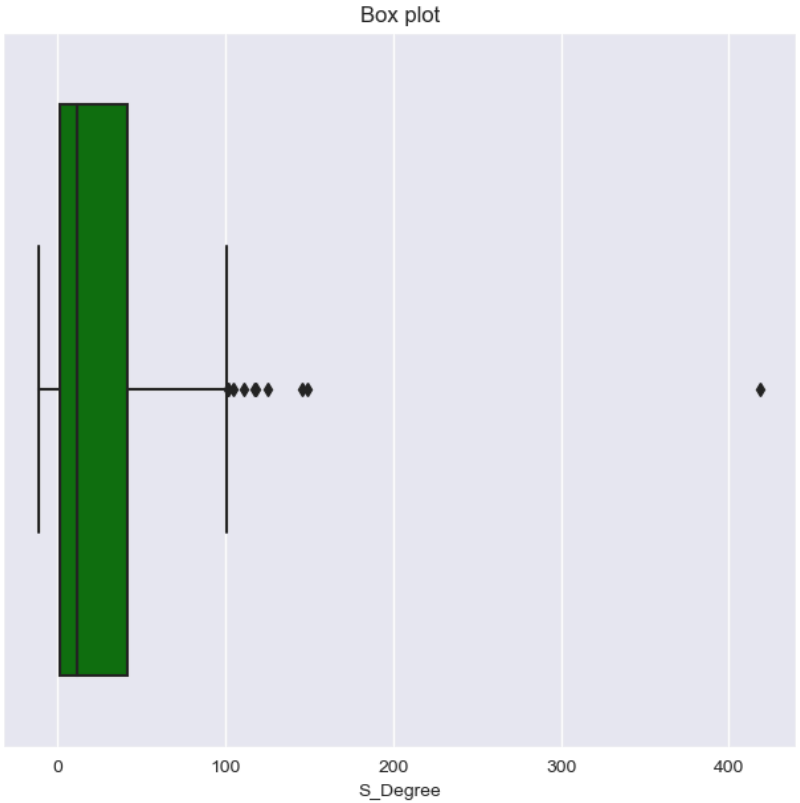
We can see outliers at both the ends.

S_Degree

In [46]:
```
f, axes = plt.subplots(1, 2, figsize=(17,7))
sns.boxplot(x = 'S_Degree', data=df,  orient='h' , ax=axes[1],color='Green')
sns.distplot(df['S_Degree'],  ax=axes[0],color='Green')
axes[0].set_title('Distribution plot')
axes[1].set_title('Box plot')
plt.show()
```

```
#checking count of outliers.
q25,q75=np.percentile(df['S_Degree'],25),np.percentile(df['S_Degree'],75)
IQR=q75-q25
Threshold=IQR*1.5
lower,upper=q25-Threshold,q75+Threshold
Outliers=[i for i in df['S_Degree'] if i < lower or i > upper]
print('{} Total Number of outliers in S_Degree: {}'.format('\033[1m',len(Outliers)))
```





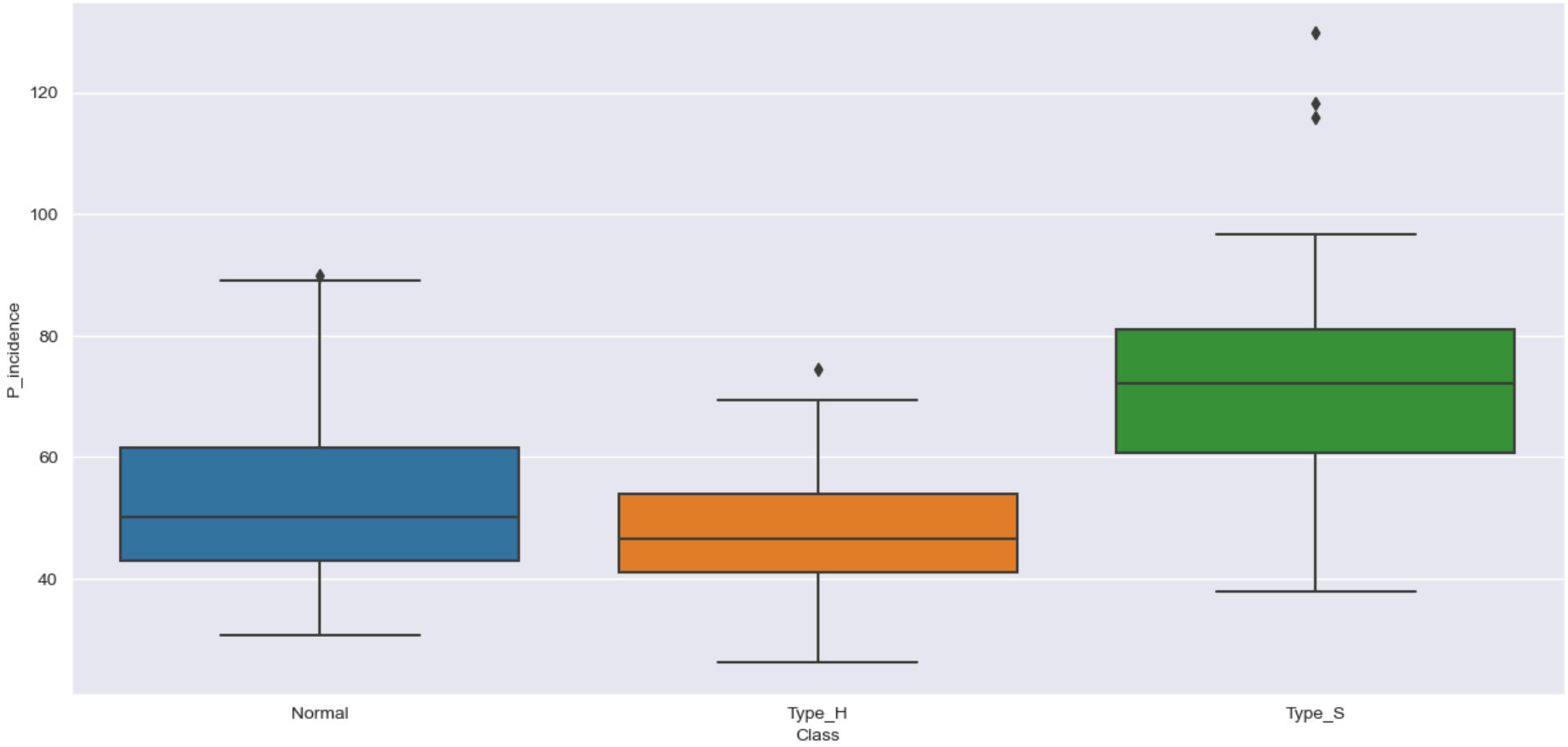
Total Number of outliers in S_Degree: 10

There is Positive Skewness in the data

Hugely affected by Outliers

boxplot for Bi variate Analysis

In [47]:
```
plt.figure(figsize=(15,7))
sns.boxplot(x='Class', y='P_incidence', data= df)
plt.show()
```

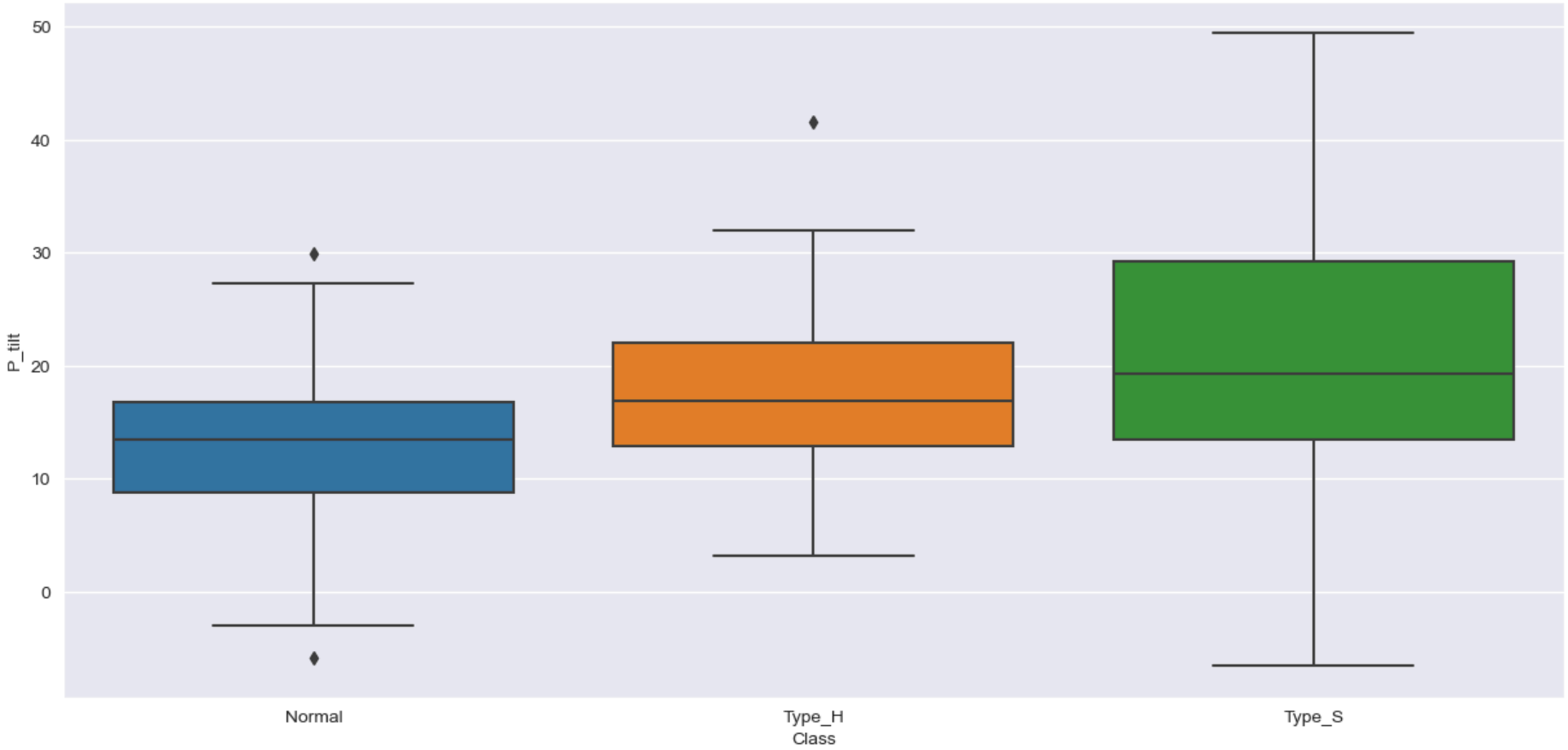


P_Incidence Value is larger for Type_S Class. We can see some extreme values as well

Normal Value is slightly higher than Type_H

Class vs P_tilt

In [48]:
```
plt.figure(figsize=(15,7))
sns.boxplot(x='Class', y='P_tilt', data= df)
plt.show()
```

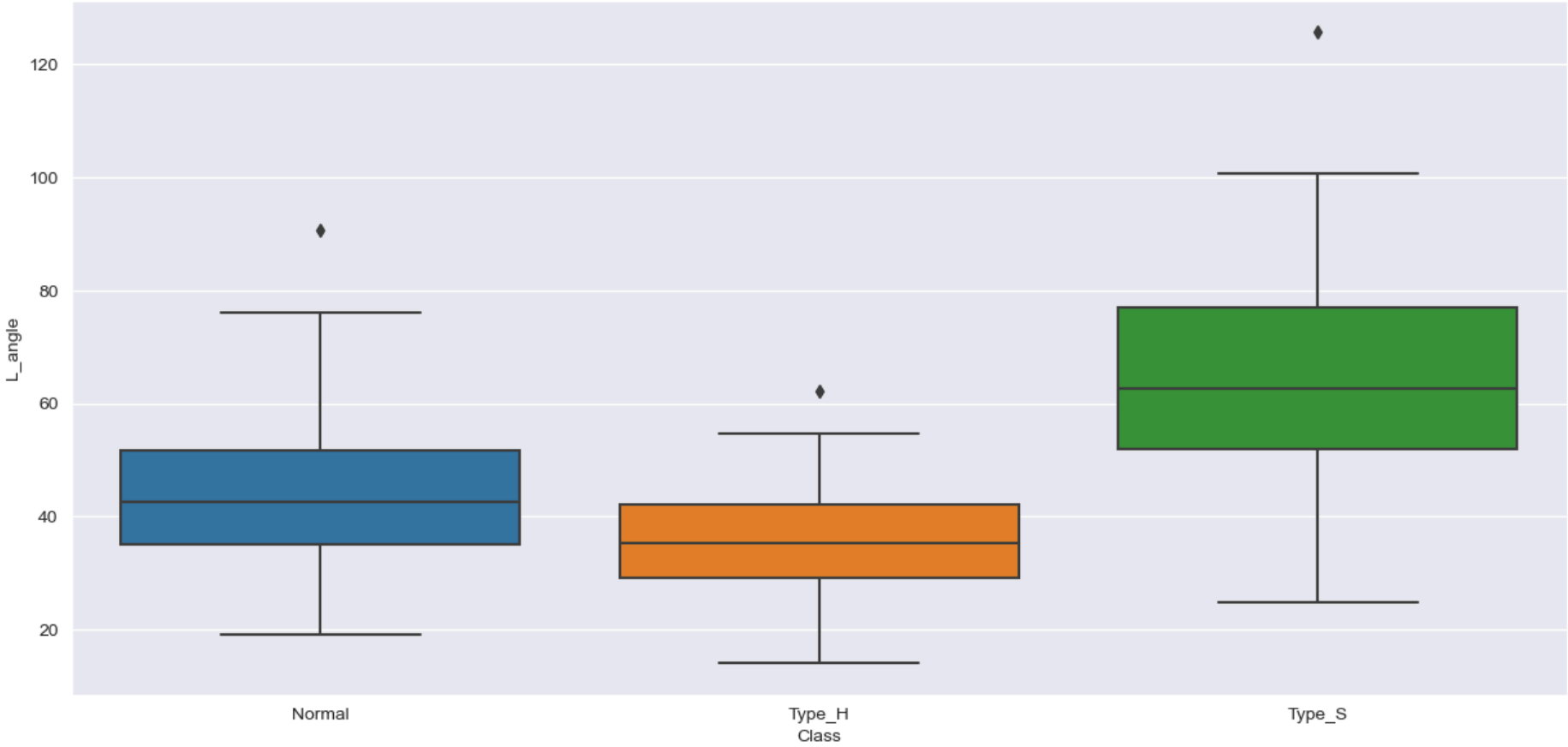


Mean of Type_S is slightly higher than rest two

Few cases Normal and Type_H also has huge values

Class vs L_angle

In [49]:
```
plt.figure(figsize=(15,7))
sns.boxplot(x='Class', y='L_angle', data= df)
plt.show()
```



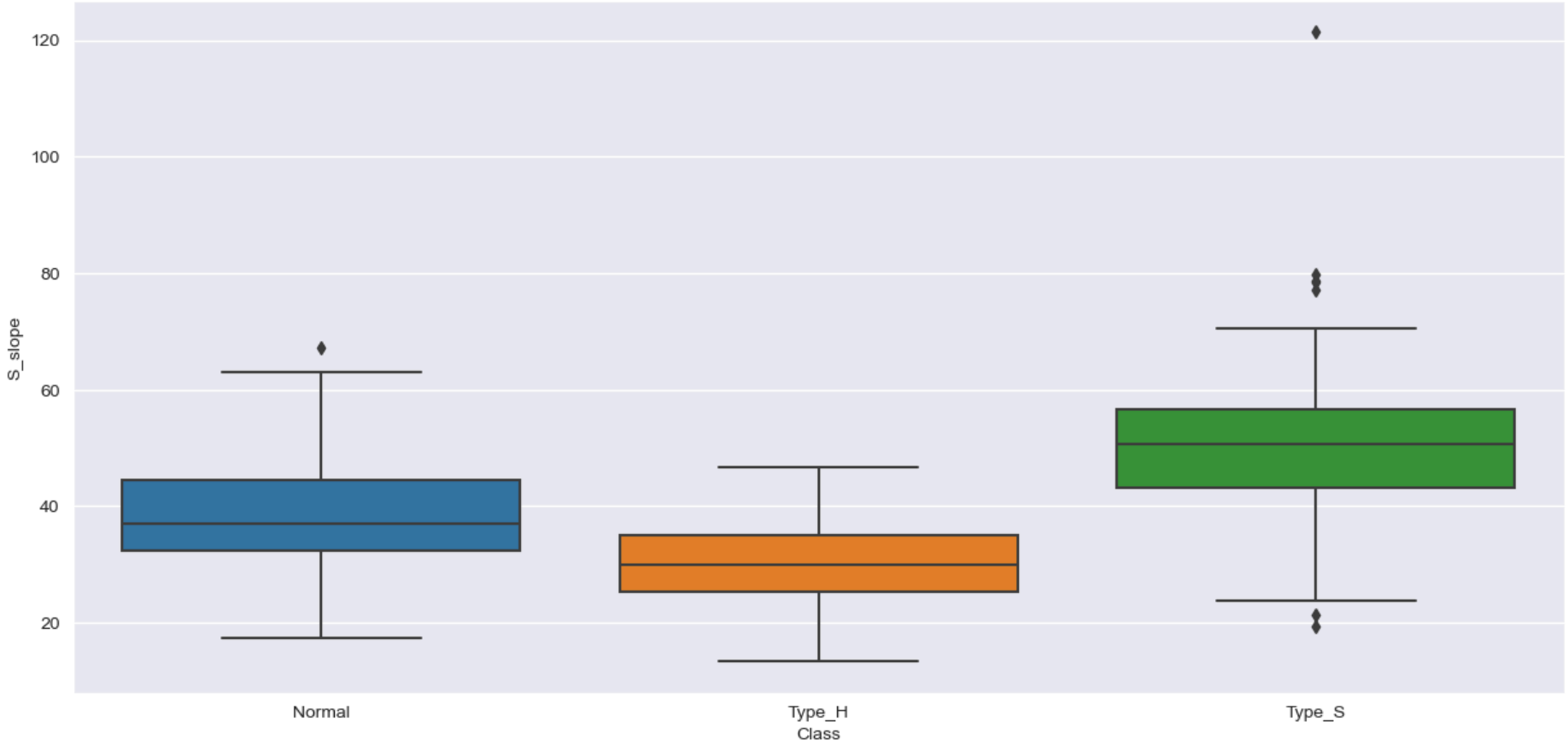
L_Angle has higher value for Type_S Class

We can see Normal class has higher values compared to type_H class

Each class contains one outlier

Class vs S_slope

In [50]:
```
plt.figure(figsize=(15,7))
sns.boxplot(x='Class', y='S_slope', data= df)
plt.show()
```

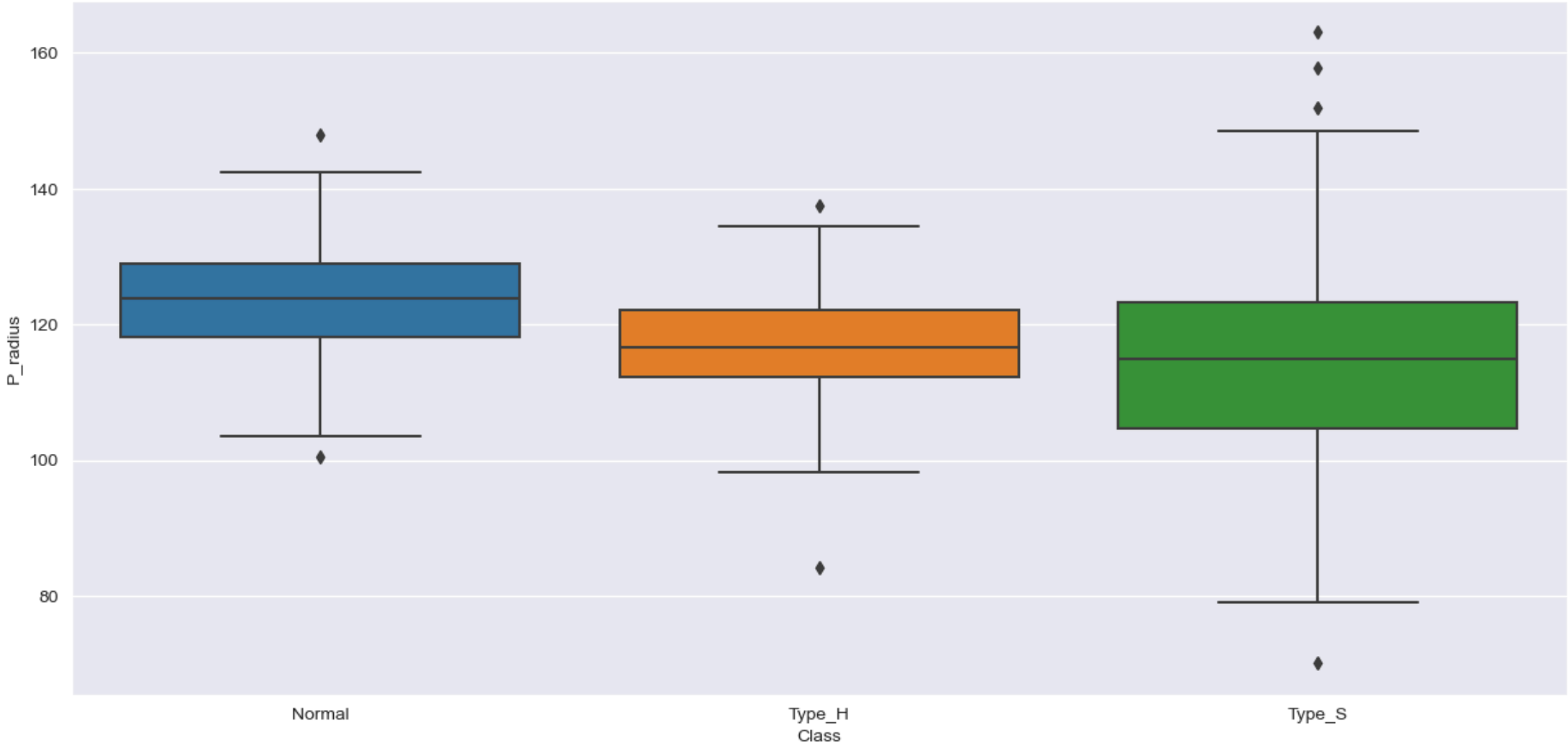


S_slope has huge values for Type_S class

Normal class has high s_slope compared to Type_H

Class vs P_radius

In [51]:
```
plt.figure(figsize=(15,7))
sns.boxplot(x='Class', y='P_radius', data= df)
plt.show()
```



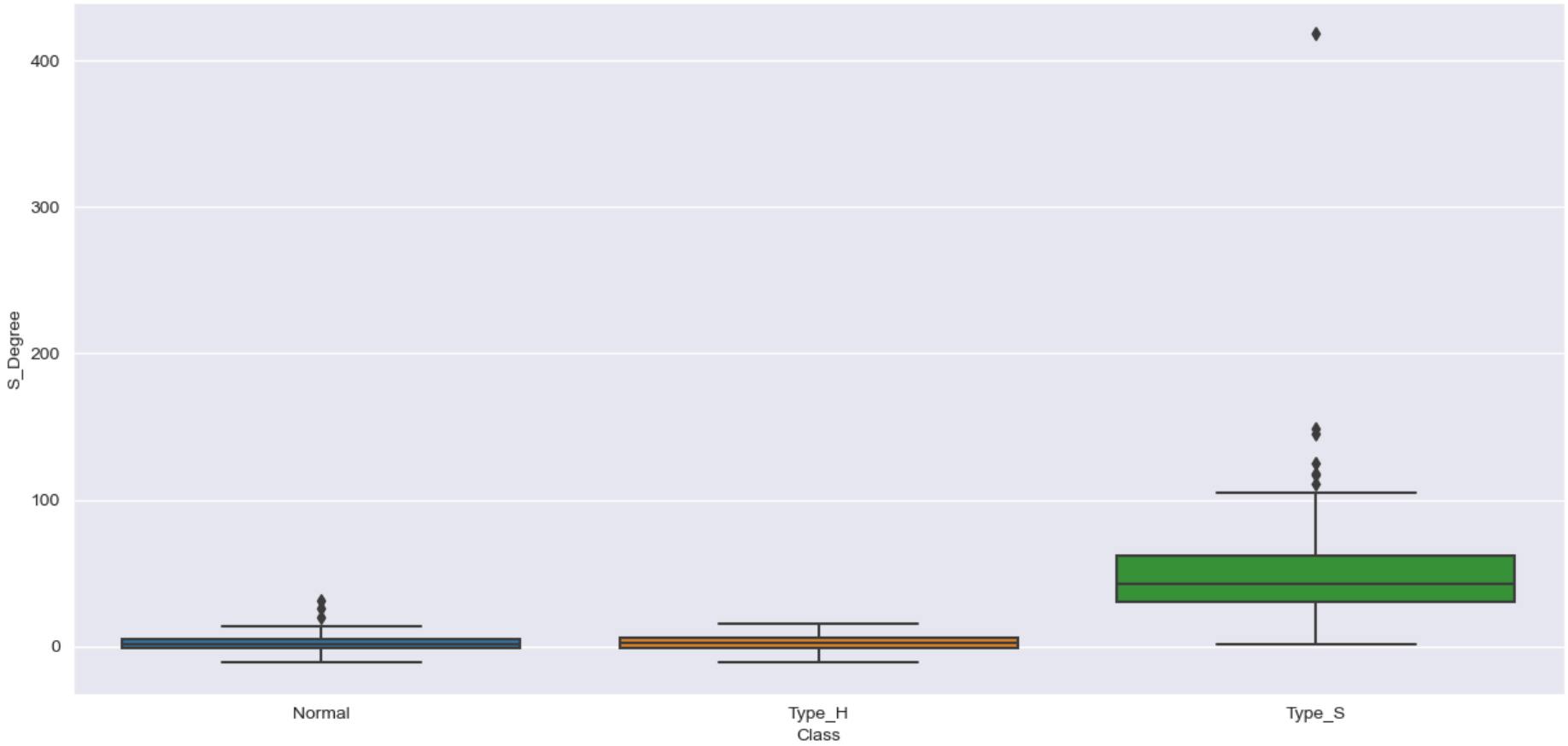
We can see P_radius value is more for Normal Class

There is some extreme values for Type_s class

All classes has higher and lower Value

Class vs S_Degree

In [52]:
```
plt.figure(figsize=(15,7))
sns.boxplot(x='Class', y='S_Degree', data= df)
plt.show()
```



S_Degree has extreme values for type_S Class

Few Normal class also has huge values for S_Degree

Answer

insights of boxplot

boxplot with univariate analysis

P_incidence

1. Total Number of outliers in P_incidence: 3
2. Normality is maintained with very less extreme values
3. We can see three outliers exists in the column

P_tilt

1. Total Number of outliers in P_tilt: 13
2. Data is Normally distributed and we can see one peakness in the center
3. It is has little skewness towards right side
4. We can see one outlier in negative end and few outliers in positive end.

L_angle

1. Total Number of outliers in L_angle: 1
2. It is Normally distributed
3. Little right skewness because of one outlier

S_slope

1. Total Number of outliers in S_slope: 1
2. There is right skewness due to one outlier

P_radius

1. Total Number of outliers in P_radius: 11
2. Data is normally distributed

3. We can see outliers at both the ends.

S_Degree

1. Total Number of outliers in S_Degree: 10

2. There is Positive Skewness in the data
3. Hugely affected by Outliers

boxplot with Bi variate analysis

Class vs P_incidence

_Incidence Value is larger for Type_S Class. We can see some extreme values as well Normal Value is slightly higher than Type_H

Class vs P_tilt

Mean of Type_S is slightly higher than rest two Few cases Normal and Type_H also has huge values

Class vs L_angle

L_Angle has higher value for Type_S Class We can see Normal class has higher values compared to type_H class Each class contains one outlier

Class vs S_slope

S_slope has huge values for Type_S class Normal class has high s_slope compared to Type_H

Class vs P_radius

We can see P_radius value is more for Normal Class There is some extreme values for Type_s class All classes has higher and lower Value

Class vs S_Degree

S_Degree has extreme values for type_S Class Few Normal class also has huge values for S_Degree

4. Model Building: [6 Marks]

A. Split data into X and Y. [1 Marks]

before doing that lets perform hypotesis testing

Is the distribution of independent variables across normal,type_H and type_s, the same?

Here we are using one-way anova to do statistical test.

In [53]:
```
col=['P_incidence','P_tilt','L_angle','S_slope','P_radius','S_Degree']
for i in col:
    print('{} Ho: Class types does not affect the {}'.format('\033[1m',i))
    print('\n')
    print('{} H1: Class types affect the {}'.format('\033[1m',i))
    print('\n')
    df_normal=df[df.Class=='Normal'][i]
    df_typeH=df[df.Class=='Type_H'][i]
    df_typeS=df[df.Class=='Type_S'][i]
    f_stats,p_value=stats.f_oneway(df_normal,df_typeH,df_typeS)
    print('{} F_stats: {}'.format('\033[1m',f_stats))
    print('{} p_value: {}'.format('\033[1m',p_value))
    if p_value < 0.05:  # Setting our significance level at 5%
        print('{} Rejecting Null Hypothesis.Class types has efect on {}'.format('\033[1m',i))
    else:
        print('{} Fail to Reject Null Hypothesis.Class types has no effect on {}'.format('\033[1m',i))
    print('\n')
```

Ho: Class types does not affect the P_incidence

H1: Class types affect the P_incidence

F_stats: 98.53970917437489
p_value: 8.752848964938295e-34
Rejecting Null Hypothesis.Class types has efect on P_incidence

Ho: Class types does not affect the P_tilt

H1: Class types affect the P_tilt

F_stats: 21.29919432898912
p_value: 2.176879152985521e-09
Rejecting Null Hypothesis.Class types has efect on P_tilt

Ho: Class types does not affect the L_angle

H1: Class types affect the L_angle

F_stats: 114.98284047330316
p_value: 5.357329394004833e-38
Rejecting Null Hypothesis.Class types has efect on L_angle

Ho: Class types does not affect the S_slope

H1: Class types affect the S_slope

F_stats: 89.64395329777523
p_value: 2.175670364983569e-31
Rejecting Null Hypothesis.Class types has efect on S_slope

```
Ho: Class types does not affect the P_radius


H1: Class types affect the P_radius


F_stats: 16.86693475538487
p_value: 1.1219959042394205e-07
Rejecting Null Hypothesis.Class types has efect on P_radius


Ho: Class types does not affect the S_Degree


H1: Class types affect the S_Degree


F_stats: 119.12288060759808
p_value: 5.1147320770401214e-39
Rejecting Null Hypothesis.Class types has efect on S_Degree
```

We can see class type affects each and every independent variables

also performimg oulier analsis and data preprocessing

bAs we have seen in our EDA we have very less outliers which needs to be handled

We are imputing outiers with mean

In [54]:
```
for c in col:
    #getting upper lower quartile values
    q25,q75=np.percentile(df[c],25),np.percentile(df[c],75)
    IQR=q75-q25
    Threshold=IQR*1.5
    lower,upper=q25-Threshold,q75+Threshold
    Outliers=[i for i in df[c] if i < lower or i > upper]
    print('{} Total Number of outliers in {} Before Imputing : {}'.format('\033[1m',c,len(Outliers)))
    print('\n')
```

```
    #taking mean of a column without considering outliers
    df_include = df.loc[(df[c] >= lower) & (df[c] <= upper)]
    mean=int(df_include[c].mean())
    print('{} Mean of {} is {}'.format('\033[1m',c,mean))
    print('\n')
    #imputing outliers with mean
    df[c]=np.where(df[c]>upper,mean,df[c])
    df[c]=np.where(df[c]<lower,mean,df[c])
    Outliers=[i for i in df[c] if i < lower or i > upper]
    print('{} Total Number of outliers in {} After Imputing : {}'.format('\033[1m',c,len(Outliers)))
    print('\n')
```

Total Number of outliers in P_incidence Before Imputing : 3

Mean of P_incidence is 59

Total Number of outliers in P_incidence After Imputing : 0

Total Number of outliers in P_tilt Before Imputing : 13

Mean of P_tilt is 16

Total Number of outliers in P_tilt After Imputing : 0

Total Number of outliers in L_angle Before Imputing : 1

Mean of L_angle is 51

Total Number of outliers in L_angle After Imputing : 0

Total Number of outliers in S_slope Before Imputing : 1

Mean of S_slope is 42

Total Number of outliers in S_slope After Imputing : 0

Total Number of outliers in P_radius Before Imputing : 11

Mean of P_radius is 118

Total Number of outliers in P_radius After Imputing : 0

```
Total Number of outliers in S_Degree Before Imputing : 10

Mean of S_Degree is 22

Total Number of outliers in S_Degree After Imputing : 0
```

Encoding Target Variable

In [55]:
```
le=LabelEncoder()
df['Class']=le.fit_transform(df['Class'])
df['Class'].value_counts()
```

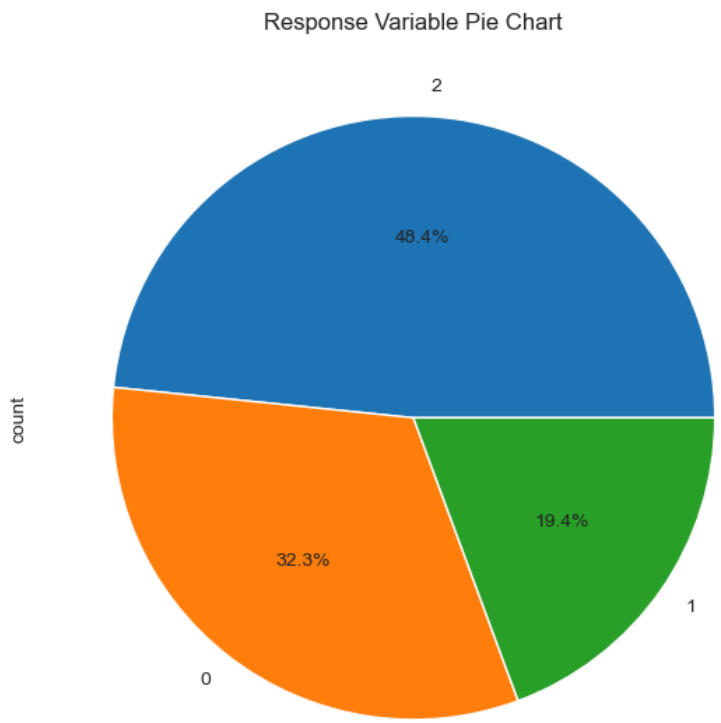
Out[55]:
```
Class
2    150
0    100
1     60
Name: count, dtype: int64
```

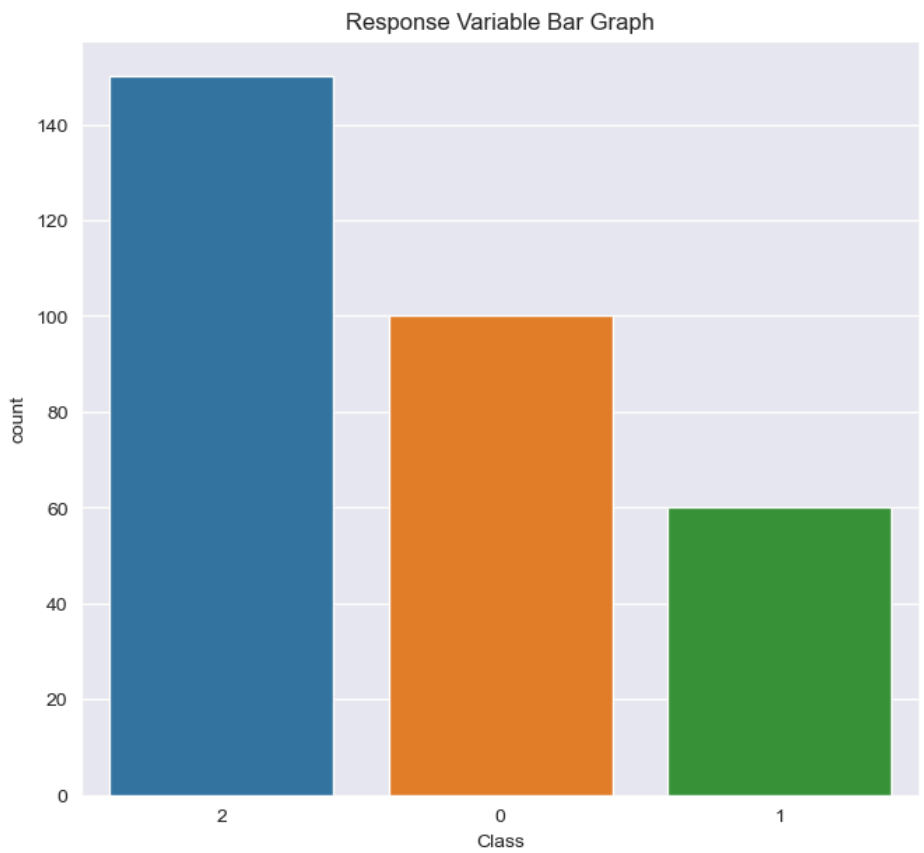
1. Normal: 0
2. Type_H: 1
3. Type_S: 2

In [56]:
```
df['Class']=df['Class'].astype('category') #changing datatype to category.
```

Checking on Target Imbalance

In [57]:
```
f,axes=plt.subplots(1,2,figsize=(17,7))
df['Class'].value_counts().plot.pie(autopct='%1.1f%%',ax=axes[0])
sns.countplot(x='Class',data=df,ax=axes[1],order=[2,0,1])
axes[0].set_title('Response Variable Pie Chart')
axes[1].set_title('Response Variable Bar Graph')
plt.show()
```





1. We have imbalanced target variable
2. Every class is not equally distributed.
3. 48% of data is occupied by Type_S
4. When you have imbalance dataset model does not learn about less distributed classes. This gives poor performance in unseen data

Train - Test Split

4A. Split data into X and Y. [1 Marks]

In [58]:
```
# Arrange data into independent variables and dependent variables
X=df.drop(columns='Class')
y=df['Class'] #target
```

In [59]:
```
X.describe()
```

Out[59]:

	P_incidence	P_tilt	L_angle	S_slope	P_radius	S_Degree
count	310.000000	310.000000	310.000000	310.000000	310.000000	310.000000
mean	59.893743	16.548519	51.689825	42.697607	118.061242	22.193516
std	16.139975	8.404101	18.071145	12.656481	11.342178	25.230932
min	26.147921	-5.845994	14.000000	13.366931	89.307547	-11.058179
25%	46.430294	10.705426	37.000000	33.347122	111.295804	1.603727
50%	58.691038	16.000000	49.562398	42.349084	118.000000	11.767934
75%	72.313279	21.021167	62.964777	52.475365	125.196027	38.144544
max	96.657315	38.750670	100.744220	79.695154	146.466001	100.292107

In [60]:
```
X_Scaled=X.apply(zscore)
X_Scaled.describe().T
```

Out[60]:

	count	mean	std	min	25%	50%	75%	max
P_incidence	310.0	1.375244e-16	1.001617	-2.094203	-0.835517	-0.074638	0.770733	2.281479
P_tilt	310.0	0.000000e+00	1.001617	-2.669021	-0.696391	-0.065374	0.533059	2.646095
L_angle	310.0	4.584147e-17	1.001617	-2.089008	-0.814203	-0.117915	0.624929	2.718904
S_slope	310.0	-9.168293e-17	1.001617	-2.321190	-0.739985	-0.027582	0.773799	2.927936
P_radius	310.0	-9.626708e-16	1.001617	-2.539211	-0.597449	-0.005408	0.630066	2.508397
S_Degree	310.0	4.584147e-17	1.001617	-1.320025	-0.817373	-0.413874	0.633223	3.100356

We have scaled independent variables to corresponding z-score.

We can see Mean becomes close to zero and Standard Deviation becomes 1

4.B. Split data into train and test with 80:20 proportion. [1 Marks]

In [65]:
```
# Split X and y into training and test set in 80:20 ratio

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=10)
```

4.C. Train a Supervised Learning Classification base model using KNN classifier. [2 Marks]

In [66]:
```
KNN = KNeighborsClassifier(n_neighbors= 5 , metric = 'euclidean' ) #Building knn with 5 neighbors
```

In [67]:
```
KNN.fit(X_train, y_train)
predicted_labels = KNN.predict(X_test)
```

Classification Accuracy

In [68]:
```
print('Accuracy on Training data:',KNN.score(X_train, y_train) )
print('Accuracy on Test data:',KNN.score(X_test, y_test) )
```

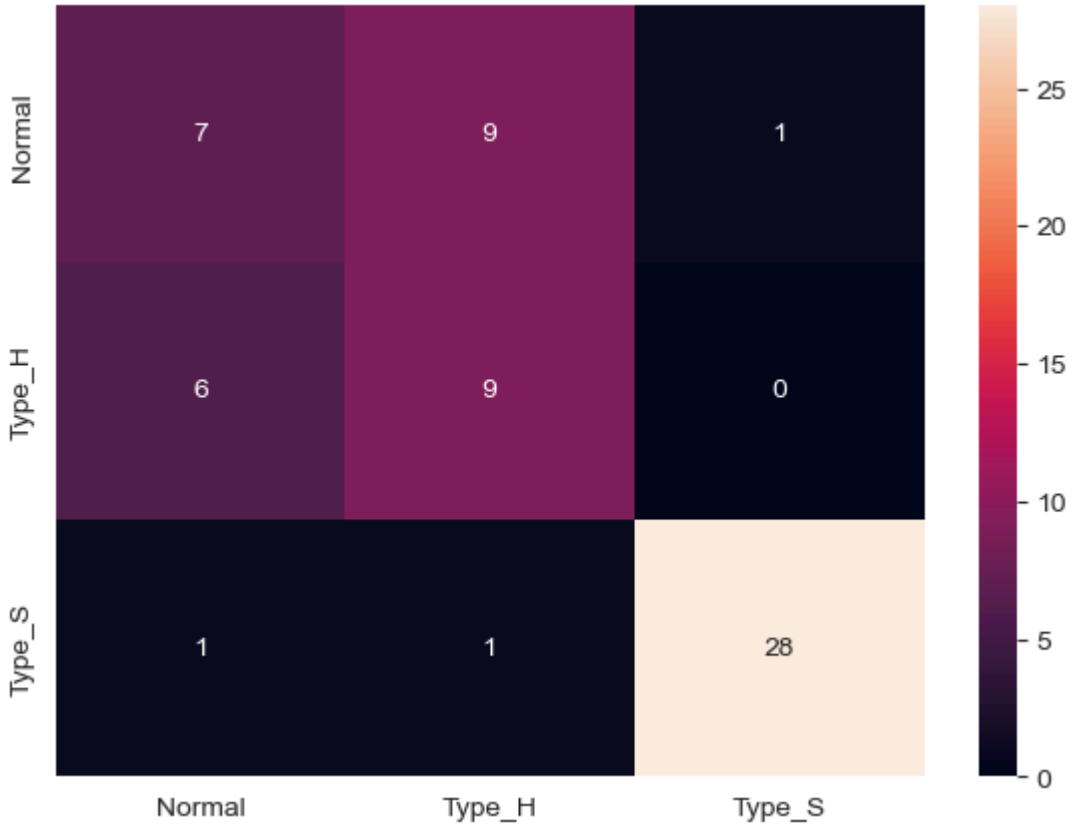
```
Accuracy on Training data: 0.9153225806451613
Accuracy on Test data: 0.7096774193548387
```

Training Acuracy is 0.91 and Testing Accuracy is 0.70. Performance is less in test data.

This is due to overfitting of data

4.D. Print all the possible performance metrics for both train and test data. [2 Marks]

In [69]:
```
cm = confusion_matrix(y_test, predicted_labels, labels=[0, 1,2])

df_cm = pd.DataFrame(cm, index = [i for i in ["Normal","Type_H","Type_S"]],
                  columns = [i for i in ["Normal","Type_H","Type_S"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True ,fmt='g')
plt.show()
```



Our model predicts Type_S correctly most of the time.

Only two misclassification on this class

Misclassification of labels are more when predicting normal class

In [70]:
```
print("classification  Matrix:\n",classification_report(y_test,predicted_labels))
```

```
classification  Matrix:
               precision    recall  f1-score   support

           0       0.50      0.41      0.45        17
           1       0.47      0.60      0.53        15
           2       0.97      0.93      0.95        30

    accuracy                           0.71        62
   macro avg       0.65      0.65      0.64        62
weighted avg       0.72      0.71      0.71        62
```

classification report

Precision for Normal class: It tells,out of all predicted normal class what fraction are predicted correctly

Recall(sensitivity or TPR) for Normal class: Out of all actual Normal class how much fraction we identified correctly

class 0 predicted correctly for 68% of time. similary for class 1 48% and class 2 98%

By F1 score we can say that precison and recall is balanced for class 0 by 60% and for class 1 by 56 %

We have maximum F1 score for class 2.

5. Performance Improvement: [4 Marks]

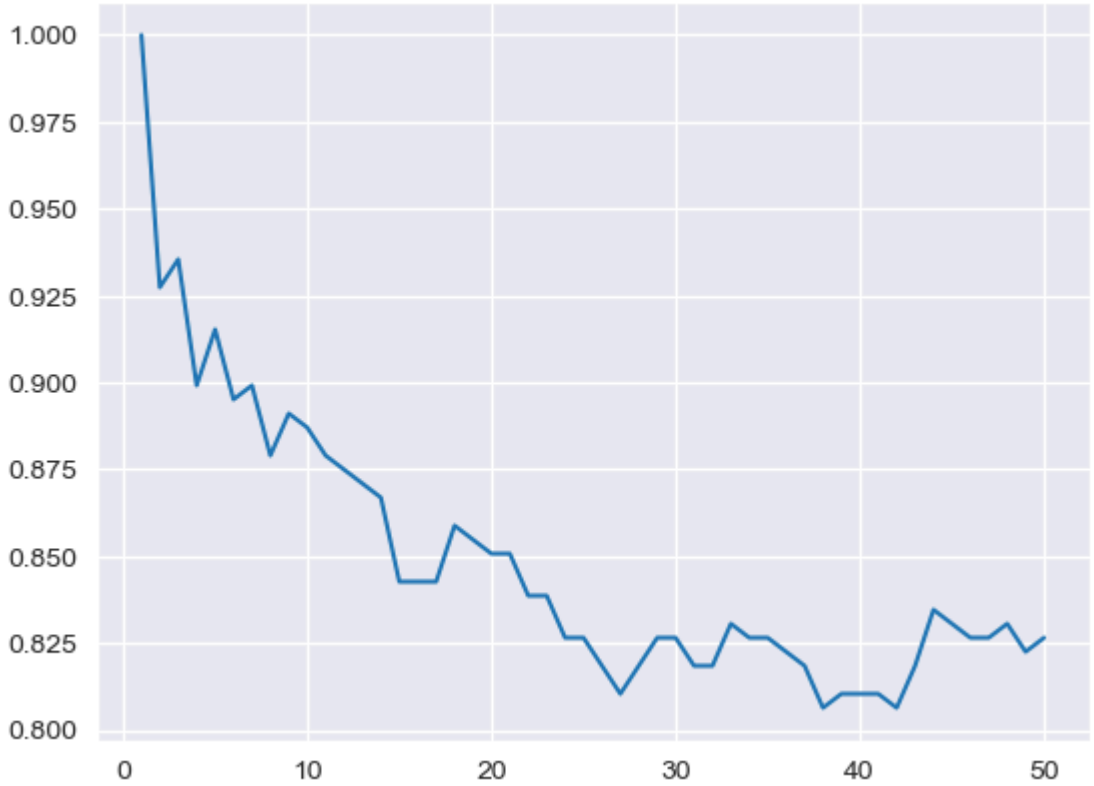
5 A. Experiment with various parameters to improve performance of the base model. [2 Marks]

(Optional: Experiment with various Hyperparameters - Research required)

In [71]:
```
train_score=[]
test_score=[]
for k in range(1,51):
    KNN = KNeighborsClassifier(n_neighbors= k , metric = 'euclidean' )
    KNN.fit(X_train, y_train)
```

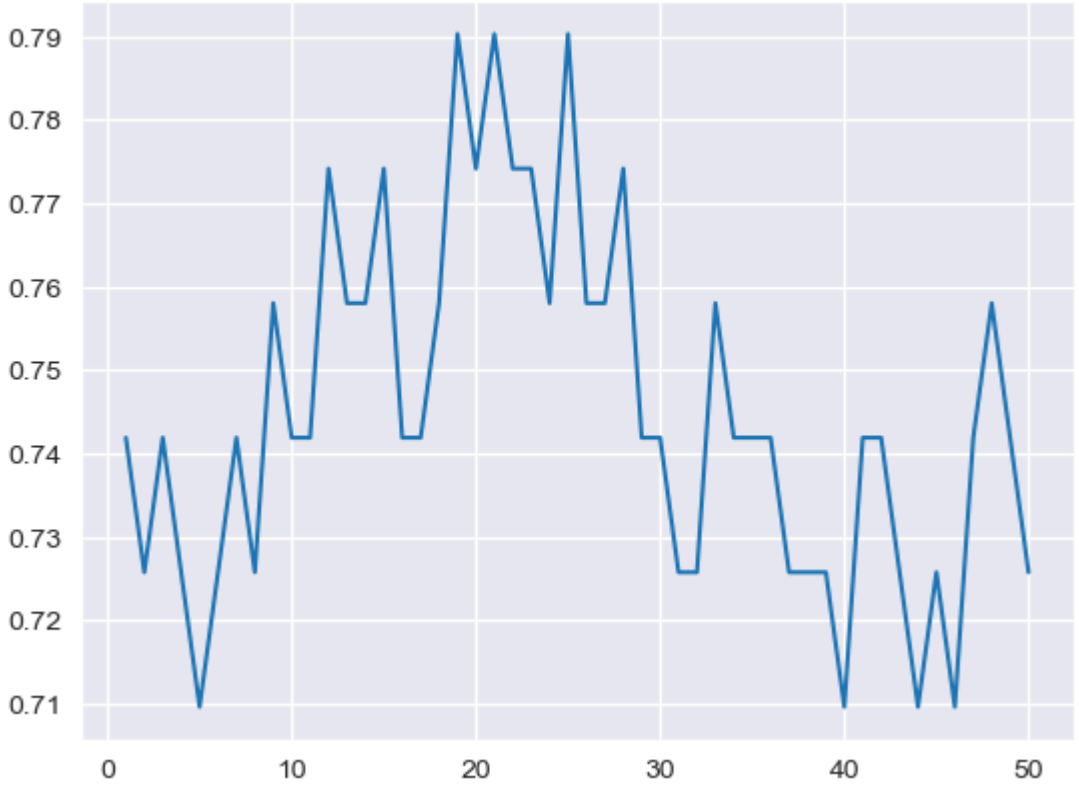
```
    train_score.append(KNN.score(X_train, y_train))
    test_score.append(KNN.score(X_test, y_test))
```

In [72]:
```
plt.plot(range(1,51),train_score)
plt.show()
```



Here training accuracy decreases when increase k value

In [73]:
```
plt.plot(range(1,51),test_score)
plt.show()
```



The maximum accuracy occures when k is less than 20.

We will fix k value as less than 20.

In [74]:
```
k=[1,3,5,7,9,11,13,15,17,19]
for i in k:
    KNN = KNeighborsClassifier(n_neighbors=i, metric = 'euclidean' ) #Building knn with 5 neighbors
    KNN.fit(X_train, y_train)
    predicted_labels = KNN.predict(X_test)
    print('Accuracy on Training data for k {} is {}:'.format(i,KNN.score(X_train, y_train)))
    print('Accuracy on Test data for k {} is {}:'.format(i,KNN.score(X_test, y_test)))
    print("classification  Matrix:\n",classification_report(y_test,predicted_labels))
```

```
Accuracy on Training data for k 1 is 1.0:
Accuracy on Test data for k 1 is 0.7419354838709677:
classification  Matrix:
               precision    recall  f1-score   support

           0       0.57      0.47      0.52        17
           1       0.53      0.67      0.59        15
           2       0.97      0.93      0.95        30

    accuracy                           0.74        62
   macro avg       0.69      0.69      0.68        62
weighted avg       0.75      0.74      0.74        62

Accuracy on Training data for k 3 is 0.9354838709677419:
Accuracy on Test data for k 3 is 0.7419354838709677:
classification  Matrix:
               precision    recall  f1-score   support

           0       0.57      0.47      0.52        17
           1       0.53      0.67      0.59        15
           2       0.97      0.93      0.95        30

    accuracy                           0.74        62
   macro avg       0.69      0.69      0.68        62
weighted avg       0.75      0.74      0.74        62

Accuracy on Training data for k 5 is 0.9153225806451613:
Accuracy on Test data for k 5 is 0.7096774193548387:
classification  Matrix:
               precision    recall  f1-score   support

           0       0.50      0.41      0.45        17
           1       0.47      0.60      0.53        15
           2       0.97      0.93      0.95        30

    accuracy                           0.71        62
   macro avg       0.65      0.65      0.64        62
weighted avg       0.72      0.71      0.71        62

Accuracy on Training data for k 7 is 0.8991935483870968:
Accuracy on Test data for k 7 is 0.7419354838709677:
classification  Matrix:
               precision    recall  f1-score   support
```

```
              0       0.57      0.47      0.52        17
              1       0.53      0.67      0.59        15
              2       0.97      0.93      0.95        30

    accuracy                           0.74        62
   macro avg       0.69      0.69      0.68        62
weighted avg       0.75      0.74      0.74        62

Accuracy on Training data for k 9 is 0.8911290322580645:
Accuracy on Test data for k 9 is 0.7580645161290323:
classification  Matrix:
               precision    recall  f1-score   support

           0       0.62      0.47      0.53        17
           1       0.55      0.73      0.63        15
           2       0.97      0.93      0.95        30

    accuracy                           0.76        62
   macro avg       0.71      0.71      0.70        62
weighted avg       0.77      0.76      0.76        62

Accuracy on Training data for k 11 is 0.8790322580645161:
Accuracy on Test data for k 11 is 0.7419354838709677:
classification  Matrix:
               precision    recall  f1-score   support

           0       0.60      0.53      0.56        17
           1       0.53      0.67      0.59        15
           2       0.96      0.90      0.93        30

    accuracy                           0.74        62
   macro avg       0.70      0.70      0.69        62
weighted avg       0.76      0.74      0.75        62

Accuracy on Training data for k 13 is 0.8709677419354839:
Accuracy on Test data for k 13 is 0.7580645161290323:
classification  Matrix:
               precision    recall  f1-score   support

           0       0.64      0.53      0.58        17
           1       0.55      0.73      0.63        15
           2       0.96      0.90      0.93        30

    accuracy                           0.76        62
```

```
   macro avg       0.72      0.72      0.71        62
weighted avg       0.78      0.76      0.76        62

Accuracy on Training data for k 15 is 0.842741935483871:
Accuracy on Test data for k 15 is 0.7741935483870968:
classification  Matrix:
              precision    recall  f1-score   support

           0       0.67      0.59      0.62        17
           1       0.58      0.73      0.65        15
           2       0.96      0.90      0.93        30

    accuracy                           0.77        62
   macro avg       0.74      0.74      0.73        62
weighted avg       0.79      0.77      0.78        62

Accuracy on Training data for k 17 is 0.842741935483871:
Accuracy on Test data for k 17 is 0.7419354838709677:
classification  Matrix:
              precision    recall  f1-score   support

           0       0.56      0.59      0.57        17
           1       0.53      0.60      0.56        15
           2       1.00      0.90      0.95        30

    accuracy                           0.74        62
   macro avg       0.69      0.70      0.69        62
weighted avg       0.76      0.74      0.75        62

Accuracy on Training data for k 19 is 0.8548387096774194:
Accuracy on Test data for k 19 is 0.7903225806451613:
classification  Matrix:
              precision    recall  f1-score   support

           0       0.61      0.65      0.63        17
           1       0.62      0.67      0.65        15
           2       1.00      0.93      0.97        30

    accuracy                           0.79        62
   macro avg       0.75      0.75      0.75        62
weighted avg       0.80      0.79      0.80        62
```

For K=13 we have balanced train and test error

we can use k value as 13 because when we increase this value the precision becomes100% for class 2

5 B. Clearly showcase improvement in performance achieved. [1 Marks]

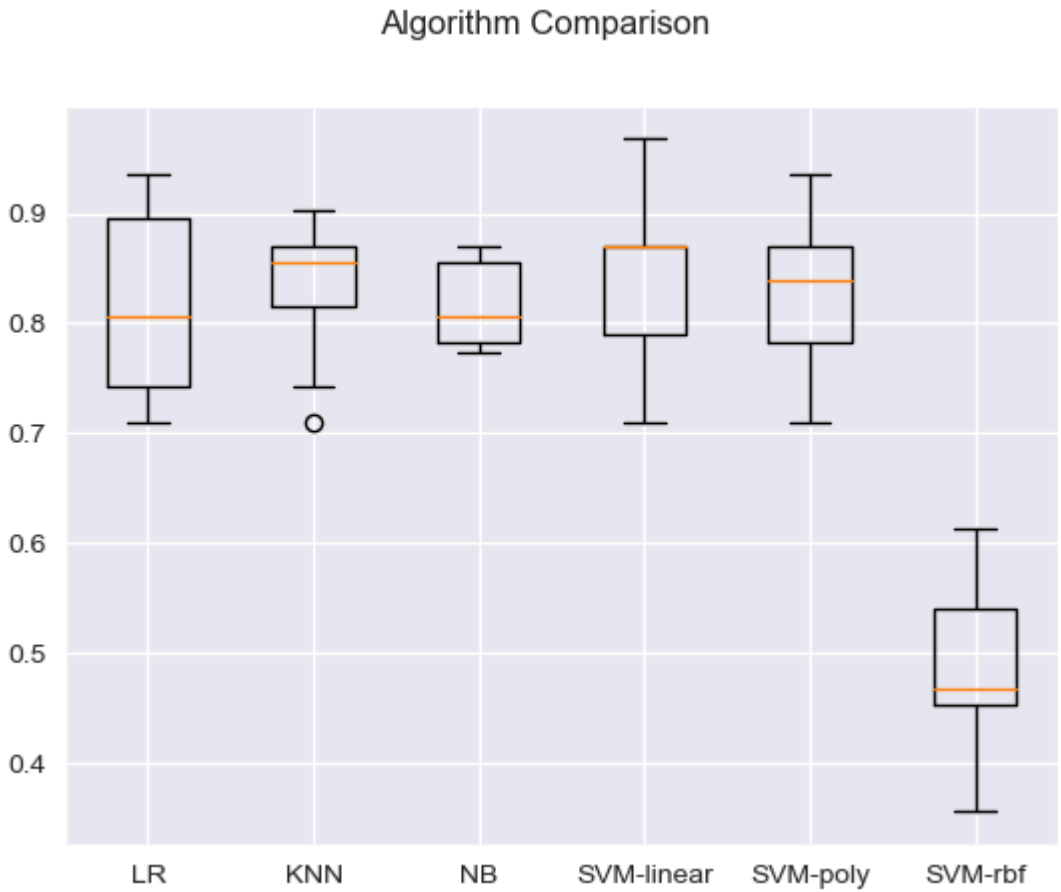
K-Fold CV for finding best model

In [75]:
```
LR_model=LogisticRegression()
KNN_model=KNeighborsClassifier(n_neighbors=13)
GN_model=GaussianNB()
svc_model_linear = SVC(kernel='linear',C=1,gamma=.6)
svc_model_rbf = SVC(kernel='rbf',degree=2,C=.009)
svc_model_poly  = SVC(kernel='poly',degree=2,gamma=0.1,C=.01)
```

In [76]:
```
seed = 7
# prepare models
models = []
models.append(('LR', LR_model))
models.append(('KNN', KNN_model))
models.append(('NB', GN_model))
models.append(('SVM-linear', svc_model_linear))
models.append(('SVM-poly', svc_model_poly))
models.append(('SVM-rbf', svc_model_rbf))
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
        cv_results = model_selection.cross_val_score(model,  X,y, cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
# boxplot algorithm comparison
```

```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

```
LR: 0.816129 (0.079082)
KNN: 0.832258 (0.059130)
NB: 0.816129 (0.038304)
SVM-linear: 0.841935 (0.068354)
SVM-poly: 0.822581 (0.069561)
SVM-rbf: 0.483871 (0.070674)
```



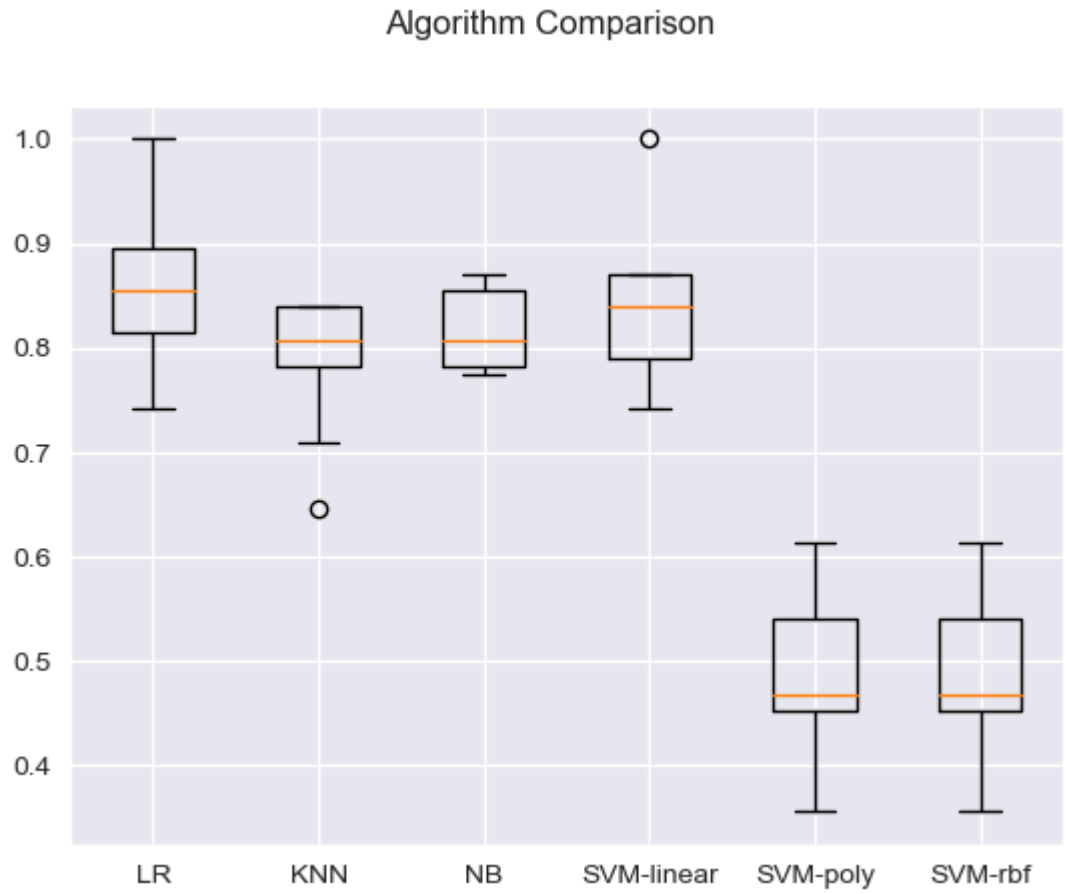
Accuracy is more for KNN,LR and svm-linear. However the standard deviation is less for svm-linear model.

We can tell svm-linear be a better algorithm for this dataset because of high accuracy and less Standard deviation

We will check with scaled values to see whether there is improvement in model

In [77]:
```
seed = 7
# prepare models
models = []
models.append(('LR', LR_model))
models.append(('KNN', KNN_model))
models.append(('NB', GN_model))
models.append(('SVM-linear', svc_model_linear))
models.append(('SVM-poly', svc_model_poly))
models.append(('SVM-rbf', svc_model_rbf))
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
        cv_results = model_selection.cross_val_score(model,X_Scaled,y, cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

```
LR: 0.854839 (0.076677)
KNN: 0.790323 (0.061629)
NB: 0.816129 (0.038304)
SVM-linear: 0.841935 (0.068354)
SVM-poly: 0.483871 (0.070674)
SVM-rbf: 0.483871 (0.070674)
```



When the scaled values are used instead of normal values Logistic regression is performing well.

Logistic Regression gives 81% accuracy with little standard deviation.

5C. Clearly state which parameters contributed most to improve model performance. [1 Marks

Conclusion and improvisation:

All the variables has significant effect on target class

class belongs to type_s has higher mean value for alomst all variables

Class belongs to normal has lower values for all variables

For almost all variables the distribution is normal

For Knn, k=13 we are getting balanced train and test error this parameter contributed most to improve model performance

We can use KNN as a final model because of balanced train and test error also the recall and precision values are good

Clear description on each variables may help to understand problem statement better because of medical domain

In [ ]:

In [ ]:

==========END PART A ==========

Part B - 30 Mark

• DOMAIN:

Banking, Marketing

• CONTEXT:

A bank X is on a massive digital transformation for all its departments. Bank has a growing customer base whee majority of them are liability customers (depositors) vs borrowers (asset customers). The bank is interested in expanding the borrowers base rapidly to bring in more business via loan interests. A campaign that the bank ran in last quarter showed an average single digit conversion rate. Digital transformation being the core strength of the business strategy, marketing department wants to devise effective campaigns with better target marketing to increase the conversion ratio to double digit with same budget as per last campaign.

DATA DICTIONARY:

1. ID: Customer ID
2. Age: Customer's approximate age.
3. CustomerSince: Customer of the bank since. [unit is masked]
4. HighestSpend: Customer's highest spend so far in one transaction. [unit is masked]
5. ZipCode: Customer's zip code.
6. HiddenScore: A score associated to the customer which is masked by the bank as an IP.
7. MonthlyAverageSpend: Customer's monthly average spend so far. [unit is masked]
8. Level: A level associated to the customer which is masked by the bank as an IP.
9. Mortgage: Customer's mortgage. [unit is masked]
10. Security: Customer's security asset with the bank. [unit is masked]
11. FixedDepositAccount: Customer's fixed deposit account with the bank. [unit is masked]
12. InternetBanking: if the customer uses internet banking.
13. CreditCard: if the customer uses bank's credit card.
14. LoanOnCard: if the customer has a loan on credit card

PROJECT OBJECTIVE:

Build a Machine Learning model to perform focused marketing by predicting the potential customers who will convert using the historical datase

STEPS AND TASK [30 Marks]:

1. Data Understanding and Preparation: [5 Marks]

1.A. Read both the Datasets 'Data1' and 'Data 2' as DataFrame and store them into two separate variables. [1 Marks]

data1

In [124…
```
newdf1=pd.read_csv("C:\\Users\\CHIRAG\\Desktop\\AIML project - Supervised learning\\PART 2 CSV\\Data1.csv")
```

In [125…
```
newdf1.head()
```

Out[125]:

	ID	Age	CustomerSince	HighestSpend	ZipCode	HiddenScore	MonthlyAverageSpend	Level
0	1	25	1	49	91107	4	1.6	1
1	2	45	19	34	90089	3	1.5	1
2	3	39	15	11	94720	1	1.0	1
3	4	35	9	100	94112	1	2.7	2
4	5	35	8	45	91330	4	1.0	2

data2

In [126…
```
newdf2=pd.read_csv("C:\\Users\\CHIRAG\\Desktop\\AIML project - Supervised learning\\PART 2 CSV\\Data2.csv")
```

In [127…
```
newdf2.head()
```

Out[127]:

	ID	Mortgage	Security	FixedDepositAccount	InternetBanking	CreditCard	LoanOnCard
0	1	0	1	0	0	0	NaN
1	2	0	1	0	0	0	NaN
2	3	0	0	0	0	0	NaN
3	4	0	0	0	0	0	NaN
4	5	0	0	0	0	1	NaN

1.B. Print shape and Column Names and DataTypes of both the Dataframes. [1 Marks]

data1

In [128...
```
print('the shape of dataframe 1 is = ',newdf1.shape)
```

```
the shape of dataframe 1 is =  (5000, 8)
```

In [129...
```
print('The column of dataframe 1 is =\n',newdf1.columns,newdf1.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   5000 non-null   int64  
 1   Age                  5000 non-null   int64  
 2   CustomerSince        5000 non-null   int64  
 3   HighestSpend         5000 non-null   int64  
 4   ZipCode              5000 non-null   int64  
 5   HiddenScore          5000 non-null   int64  
 6   MonthlyAverageSpend  5000 non-null   float64
 7   Level                5000 non-null   int64  
dtypes: float64(1), int64(7)
memory usage: 312.6 KB
The column of dataframe 1 is =
 Index(['ID', 'Age', 'CustomerSince', 'HighestSpend', 'ZipCode', 'HiddenScore',
       'MonthlyAverageSpend', 'Level'],
      dtype='object') None
```

In [130…

```
print('The data type of dataframe 1 is=\n',newdf1.dtypes)
```

```
The data type of dataframe 1 is=
 ID                     int64
Age                    int64
CustomerSince          int64
HighestSpend           int64
ZipCode                int64
HiddenScore            int64
MonthlyAverageSpend    float64
Level                  int64
dtype: object
```

data2

In [131…

```
print('The shape of dataframe 2 is = ',newdf2.shape)
```

```
The shape of dataframe 2 is =  (5000, 7)
```

In [132…

```
print('The column of dataframe 2 is =\n',newdf2.columns,newdf2.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   5000 non-null   int64  
 1   Mortgage             5000 non-null   int64  
 2   Security             5000 non-null   int64  
 3   FixedDepositAccount  5000 non-null   int64  
 4   InternetBanking      5000 non-null   int64  
 5   CreditCard           5000 non-null   int64  
 6   LoanOnCard           4980 non-null   float64
dtypes: float64(1), int64(6)
memory usage: 273.6 KB
The column of dataframe 2 is =
 Index(['ID', 'Mortgage', 'Security', 'FixedDepositAccount', 'InternetBanking',
       'CreditCard', 'LoanOnCard'],
      dtype='object') None
```

In [133…
```
print('The data type of dataframe 2 is=\n',newdf2.dtypes)
```

```
The data type of dataframe 2 is=
 ID                      int64
Mortgage                int64
Security                int64
FixedDepositAccount     int64
InternetBanking         int64
CreditCard              int64
LoanOnCard            float64
dtype: object
```

1C. Merge both the Dataframes on ‘ID’ feature to form a single DataFrame [2 Marks]

In [142…
```
newdf=newdf1.merge(newdf2,left_on='ID',right_on='ID')
#id is common in both dataframe
newdf.shape
```

Out[142]:
```
(5000, 14)
```

In [143…
```
newdf.describe().T
```

Out[143]:

	count	mean	std	min	25%	50%	75%	max
ID	5000.0	2500.500000	1443.520003	1.0	1250.75	2500.5	3750.25	5000.0
Age	5000.0	45.338400	11.463166	23.0	35.00	45.0	55.00	67.0
CustomerSince	5000.0	20.104600	11.467954	-3.0	10.00	20.0	30.00	43.0
HighestSpend	5000.0	73.774200	46.033729	8.0	39.00	64.0	98.00	224.0
ZipCode	5000.0	93152.503000	2121.852197	9307.0	91911.00	93437.0	94608.00	96651.0
HiddenScore	5000.0	2.396400	1.147663	1.0	1.00	2.0	3.00	4.0
MonthlyAverageSpend	5000.0	1.937938	1.747659	0.0	0.70	1.5	2.50	10.0
Level	5000.0	1.881000	0.839869	1.0	1.00	2.0	3.00	3.0
Mortgage	5000.0	56.498800	101.713802	0.0	0.00	0.0	101.00	635.0
Security	5000.0	0.104400	0.305809	0.0	0.00	0.0	0.00	1.0
FixedDepositAccount	5000.0	0.060400	0.238250	0.0	0.00	0.0	0.00	1.0
InternetBanking	5000.0	0.596800	0.490589	0.0	0.00	1.0	1.00	1.0
CreditCard	5000.0	0.294000	0.455637	0.0	0.00	0.0	1.00	1.0
LoanOnCard	4980.0	0.096386	0.295149	0.0	0.00	0.0	0.00	1.0

In [144…

```
newdf_copy=newdf.copy()
```

1D. Change Datatype of below features to ‘Object’ [1 Marks]

‘CreditCard’,

‘InternetBanking’,

‘FixedDepositAccount’,

‘Security’,

‘Level’,

'HiddenScore'.

[Reason behind performing this operation:- Values in these features are binary i.e. 1/0. But DataType is 'int'/'float' which is not expected.

In [145…
```
col=['CreditCard','InternetBanking','FixedDepositAccount','Security','Level','HiddenScore']
for i in col:
    newdf[i]=newdf[i].astype('category')
```

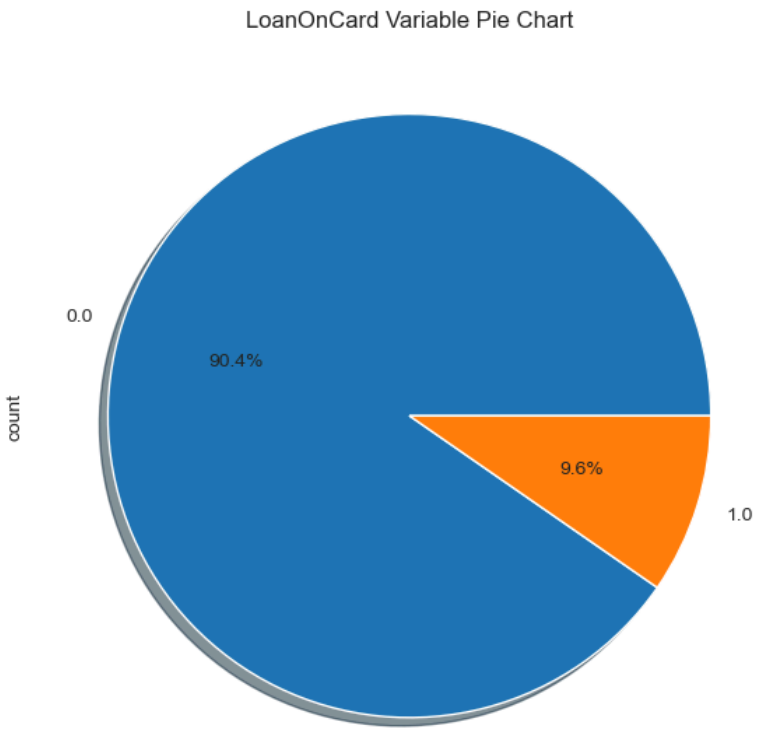
In [146…
```
newdf.dtypes
```

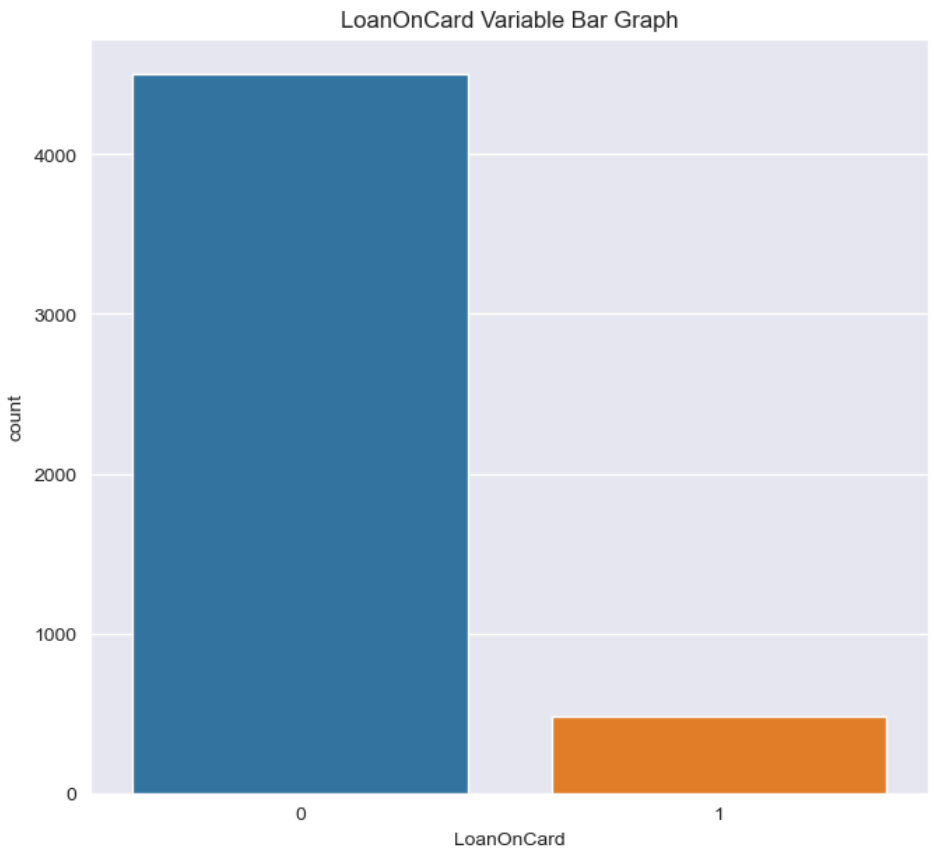
Out[146]:
```
ID                      int64
Age                     int64
CustomerSince           int64
HighestSpend            int64
ZipCode                 int64
HiddenScore          category
MonthlyAverageSpend   float64
Level                category
Mortgage                int64
Security             category
FixedDepositAccount  category
InternetBanking      category
CreditCard           category
LoanOnCard            float64
dtype: object
```

2. Data Exploration and Analysis: [5 Marks]

2.A. Visualize distribution of Target variable 'LoanOnCard' and clearly share insights. [2 Marks]

In [147…
```
f,axes=plt.subplots(1,2,figsize=(17,7))
newdf['LoanOnCard'].value_counts().plot.pie(autopct='%1.1f%%',ax=axes[0],shadow=True)
sns.countplot(x='LoanOnCard',data=newdf,ax=axes[1],order=[0,1])
axes[0].set_title('LoanOnCard Variable Pie Chart')
axes[1].set_title('LoanOnCard Variable Bar Graph')
plt.show()
```





answer

insights

We can see 90% of people does not have loan on credit card

morethan 4000 people does not have loan on credit card

2 B. Check the percentage of missing values and impute if required. [1 Marks]

In [148…

```
newdf.isnull().sum()/newdf.shape[0]*100
```

Out[148]:
```
ID                   0.0
Age                  0.0
CustomerSince        0.0
HighestSpend         0.0
ZipCode              0.0
HiddenScore          0.0
MonthlyAverageSpend  0.0
Level                0.0
Mortgage             0.0
Security             0.0
FixedDepositAccount  0.0
InternetBanking      0.0
CreditCard           0.0
LoanOnCard           0.4
dtype: float64
```

we have very less missing value so we will drop the missing records.

In [149…
```
newdf.dropna(axis=0,inplace=True)
```

In [150…
```
newdf.isnull().sum()
```

Out[150]:
```
ID                   0
Age                  0
CustomerSince        0
HighestSpend         0
ZipCode              0
HiddenScore          0
MonthlyAverageSpend  0
Level                0
Mortgage             0
Security             0
FixedDepositAccount  0
InternetBanking      0
CreditCard           0
LoanOnCard           0
dtype: int64
```

after droping missing value we have no missing value left

2C. Check for unexpected values in each categorical variable and impute with best suitable value. [2 Marks]

[Unexpected values means if all values in a feature are 0/1 then '?', 'a', 1.5 are unexpected values which needs treatment

In [151…
```
newdf.columns
```

Out[151]:
```
Index(['ID', 'Age', 'CustomerSince', 'HighestSpend', 'ZipCode', 'HiddenScore',
       'MonthlyAverageSpend', 'Level', 'Mortgage', 'Security',
       'FixedDepositAccount', 'InternetBanking', 'CreditCard', 'LoanOnCard'],
      dtype='object')
```

In [152…
```
newdf.describe()
```

Out[152]:

	ID	Age	CustomerSince	HighestSpend	ZipCode	MonthlyAverageSpend	Mortgage	LoanOnCard
count	4980.000000	4980.000000	4980.000000	4980.00000	4980.000000	4980.000000	4980.000000	4980.000000
mean	2510.345382	45.352610	20.117671	73.85241	93152.420482	1.939536	56.589759	0.096386
std	1438.011129	11.464212	11.468716	46.07009	2123.660073	1.750006	101.836758	0.295149
min	10.000000	23.000000	-3.000000	8.00000	9307.000000	0.000000	0.000000	0.000000
25%	1265.750000	35.000000	10.000000	39.00000	91911.000000	0.700000	0.000000	0.000000
50%	2510.500000	45.000000	20.000000	64.00000	93407.000000	1.500000	0.000000	0.000000
75%	3755.250000	55.000000	30.000000	98.00000	94608.000000	2.525000	101.000000	0.000000
max	5000.000000	67.000000	43.000000	224.00000	96651.000000	10.000000	635.000000	1.000000

3. Data Preparation and model building: [10 Marks]

A. Split data into X and Y. [1 Marks]

[Recommended to drop ID & ZipCode. LoanOnCard is target Variable]

In [153…
```
class_summary=newdf_copy.groupby('LoanOnCard') #getting mean values of each class for all independent variables
class_summary.mean().reset_index()
```

Out[153]:

	LoanOnCard	ID	Age	CustomerSince	HighestSpend	ZipCode	HiddenScore	MonthlyAverageSpend	Level	Mortgage	Securit
0	0.0	2523.112889	45.383111	20.146889	66.290444	93152.337111	2.372444	1.729849	1.843333	51.869111	0.10222
1	1.0	2390.650000	45.066667	19.843750	144.745833	93153.202083	2.612500	3.905354	2.233333	100.845833	0.12500

In [154…
```
col=list(newdf_copy.select_dtypes(include=['int64','float64']).columns)
```

In [155…
```
for i in col:
    x = np.array(newdf_copy[newdf_copy.LoanOnCard == 0][i])
    y = np.array(newdf_copy[newdf_copy.LoanOnCard == 1][i])
    t, p_value  = stats.ttest_ind(x,y, axis = 0,equal_var=False)
    print('{} P_Value:{}'.format('\033[1m',p_value))
    if p_value < 0.05:  # Setting our significance level at 5%
        print('{} Rejecting Null Hypothesis.{} of Loan holders and non-Loan holders are not same'.format('\033[1m',i))
    else:
        print('{} Fail to Reject Null Hypothesis.{} of Loan holders and non-Loan holders are  same'.format('\033[1m',i))
    print('\n')
```

P_Value:0.049093722429066254
Rejecting Null Hypothesis.ID of Loan holders and non-Loan holders are not same

P_Value:0.5694160158774422
Fail to Reject Null Hypothesis.Age of Loan holders and non-Loan holders are same

P_Value:0.5855242526574542
Fail to Reject Null Hypothesis.CustomerSince of Loan holders and non-Loan holders are same

P_Value:1.527529731162187e-227
Rejecting Null Hypothesis.HighestSpend of Loan holders and non-Loan holders are not same

P_Value:0.9920253364424511
Fail to Reject Null Hypothesis.ZipCode of Loan holders and non-Loan holders are same

P_Value:9.399870098381216e-06
Rejecting Null Hypothesis.HiddenScore of Loan holders and non-Loan holders are not same

P_Value:2.4144099931230367e-77
Rejecting Null Hypothesis.MonthlyAverageSpend of Loan holders and non-Loan holders are not same

P_Value:1.916152850102825e-24
Rejecting Null Hypothesis.Level of Loan holders and non-Loan holders are not same

P_Value:1.3389598194359617e-10
Rejecting Null Hypothesis.Mortgage of Loan holders and non-Loan holders are not same

P_Value:0.14922270750798364
Fail to Reject Null Hypothesis.Security of Loan holders and non-Loan holders are same

P_Value:3.934086798293139e-30
Rejecting Null Hypothesis.FixedDepositAccount of Loan holders and non-Loan holders are not same

```
P_Value:0.6696352242515428
Fail to Reject Null Hypothesis.InternetBanking of Loan holders and non-Loan holders are  same


P_Value:0.8585872788176462
Fail to Reject Null Hypothesis.CreditCard of Loan holders and non-Loan holders are  same


P_Value:0.0
Rejecting Null Hypothesis.LoanOnCard of Loan holders and non-Loan holders are not same
```

In [156…
```
newdf.drop(['Age','CustomerSince','ZipCode','ID'],axis=1,inplace=True)
```

In [157…
```
crosstab=pd.crosstab(newdf['LoanOnCard'],newdf['HiddenScore'])
print(crosstab)
```

```
HiddenScore     1     2    3     4
LoanOnCard
0.0          1359  1187  873  1081
1.0           107   106  133   134
```

In [158…
```
chi,p_value,dof,expected=stats.chi2_contingency(crosstab)
print('P_Value:', p_value)
```

```
P_Value: 1.5107064617649127e-06
```

In [159…
```
if p_value < 0.05:  # Setting our significance level at 5%
    print('{} Rejecting Null Hypothesis. \n There is significant difference in hidden score for different category of target vari
else:
    print('{} Fail to Reject Null Hypothesis.\n There is no significant difference in hidden score for different category of targ
```

```
Rejecting Null Hypothesis.
There is significant difference in hidden score for different category of target variable(Loan on card)
```

In [160…
```
cat_col=list(newdf.select_dtypes(include=['category']).columns)
```

In [161…
```
for i in cat_col:
    crosstab=pd.crosstab(newdf['LoanOnCard'],newdf[i])
    chi,p_value,dof,expected=stats.chi2_contingency(crosstab)
    if p_value < 0.05:  # Setting our significance level at 5%
        print('{} Rejecting Null Hypothesis. \n There is significant difference in {} Feature for different category of target va
    else:
```

```
        print('{} Fail to Reject Null Hypothesis.\n There is no significant difference in {} Feature for different category of ta
    print('\n')
```

```
Rejecting Null Hypothesis.
There is significant difference in HiddenScore Feature for different category of target variable(Loan on card)


Rejecting Null Hypothesis.
There is significant difference in Level Feature for different category of target variable(Loan on card)


Fail to Reject Null Hypothesis.
There is no significant difference in Security Feature for different category of target variable(Loan on card)


Rejecting Null Hypothesis.
There is significant difference in FixedDepositAccount Feature for different category of target variable(Loan on card)


Fail to Reject Null Hypothesis.
There is no significant difference in InternetBanking Feature for different category of target variable(Loan on card)


Fail to Reject Null Hypothesis.
There is no significant difference in CreditCard Feature for different category of target variable(Loan on card)
```

In [162…
```
newdf.drop(['CreditCard','InternetBanking','Security'],axis=1,inplace=True)
```

we are imputing outliner with mean

In [163…
```
col=['HighestSpend','MonthlyAverageSpend','Mortgage']
```

In [164…
```
for c in col:
    #getting upper lower quartile values
    q25,q75=np.percentile(newdf[c],25),np.percentile(newdf[c],75)
    IQR=q75-q25
    Threshold=IQR*1.5
    lower,upper=q25-Threshold,q75+Threshold
    Outliers=[i for i in newdf[c] if i < lower or i > upper]
```

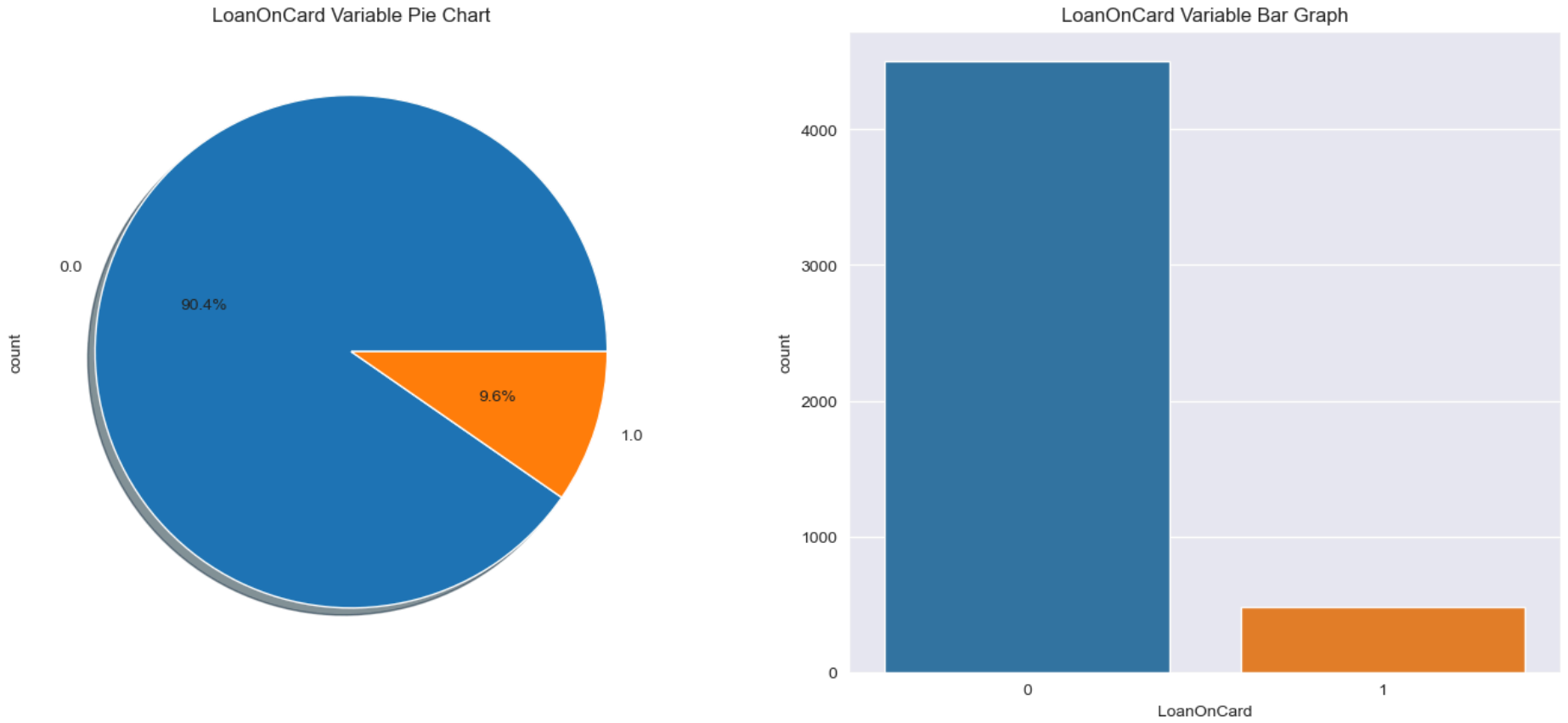
```
    print('{} Total Number of outliers in {} Before Imputing : {}'.format('\033[1m',c,len(Outliers)))
    print('\n')
    #taking mean of a column without considering outliers
    df_include = newdf.loc[(newdf[c] >= lower) & (newdf[c] <= upper)]
    mean=int(df_include[c].mean())
    print('{} Mean of {} is {}'.format('\033[1m',c,mean))
    print('\n')
    #imputing outliers with mean
    newdf[c]=np.where(newdf[c]>upper,mean,newdf[c])
    newdf[c]=np.where(newdf[c]<lower,mean,newdf[c])
    Outliers=[i for i in newdf[c] if i < lower or i > upper]
    print('{} Total Number of outliers in {} After Imputing : {}'.format('\033[1m',c,len(Outliers)))
    print('\n')
```

```
Total Number of outliers in HighestSpend Before Imputing : 96


Mean of HighestSpend is 71


Total Number of outliers in HighestSpend After Imputing : 0


Total Number of outliers in MonthlyAverageSpend Before Imputing : 324


Mean of MonthlyAverageSpend is 1


Total Number of outliers in MonthlyAverageSpend After Imputing : 0


Total Number of outliers in Mortgage Before Imputing : 291


Mean of Mortgage is 38


Total Number of outliers in Mortgage After Imputing : 0
```

In [165…

```
f,axes=plt.subplots(1,2,figsize=(17,7))
newdf['LoanOnCard'].value_counts().plot.pie(autopct='%1.1f%%',ax=axes[0],shadow=True)
```

```
sns.countplot(x='LoanOnCard',data=newdf,ax=axes[1],order=[0,1])
axes[0].set_title('LoanOnCard Variable Pie Chart')
axes[1].set_title('LoanOnCard Variable Bar Graph')
plt.show()
```



In [178…

```
# Arrange data into independent variables and dependent variables
X=newdf.drop(columns='LoanOnCard')
y=newdf['LoanOnCard'] #target
```

3 B. Split data into train and test. Keep 25% data reserved for testing. [1 Marks]

In [179…

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=10)
```

3 C. Train a Supervised Learning Classification base model - Logistic Regression. [2 Marks]

In [180…

```
logit = LogisticRegression()
logit.fit(X_train, y_train)
logit_pred = logit.predict(X_test)

print('Accuracy on Training data:',logit.score(X_train, y_train) )
print('Accuracy on Test data:',logit.score(X_test, y_test) )
```

```
Accuracy on Training data: 0.9502008032128514
Accuracy on Test data: 0.940562248995984
```

95% accuracy on training set and 94% accuracy on test set. Here training set accuracy and testing accuracy are balanced when model is built without sampling also accuracy is good

3 D. Print evaluation metrics for the model and clearly share insights. [1 Marks]

consusion matrix

In [181…

```
cm = confusion_matrix(y_test, logit_pred, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in ["Non-Loan holders","Loan holders"]],
                  columns = [i for i in ["Non-Loan holders","Loan holders"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True ,fmt='g')
plt.show()
```

In [182…

```
print("classification  Matrix:\n",classification_report(y_test,logit_pred))
```

```
classification  Matrix:
               precision    recall  f1-score   support

         0.0       0.95      0.98      0.97      1118
         1.0       0.77      0.59      0.67       127

    accuracy                           0.94      1245
   macro avg       0.86      0.79      0.82      1245
weighted avg       0.94      0.94      0.94      1245
```

Here you can see model is poor in predicting class 1 compared to class 0

Accuracy is good but in this case we need to look on recall value

Here Recall tells that only 48% class 1 is predicted correctly from actual values

we dont have enough sample of class 1 to train the model.

We will do the sampling and check how recall values improves in this case.

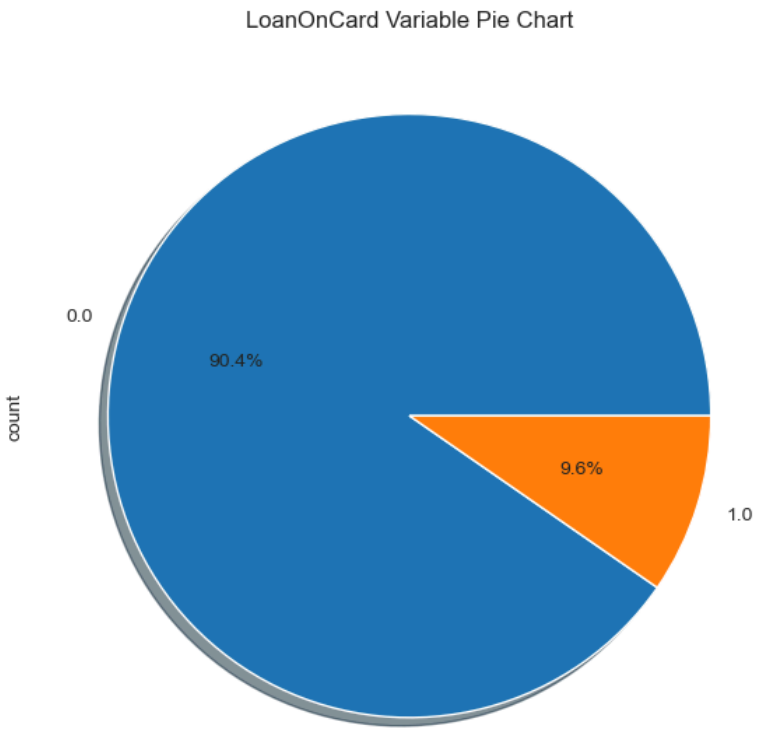
3 E. Balance the data using the right balancing technique. [2 Marks]

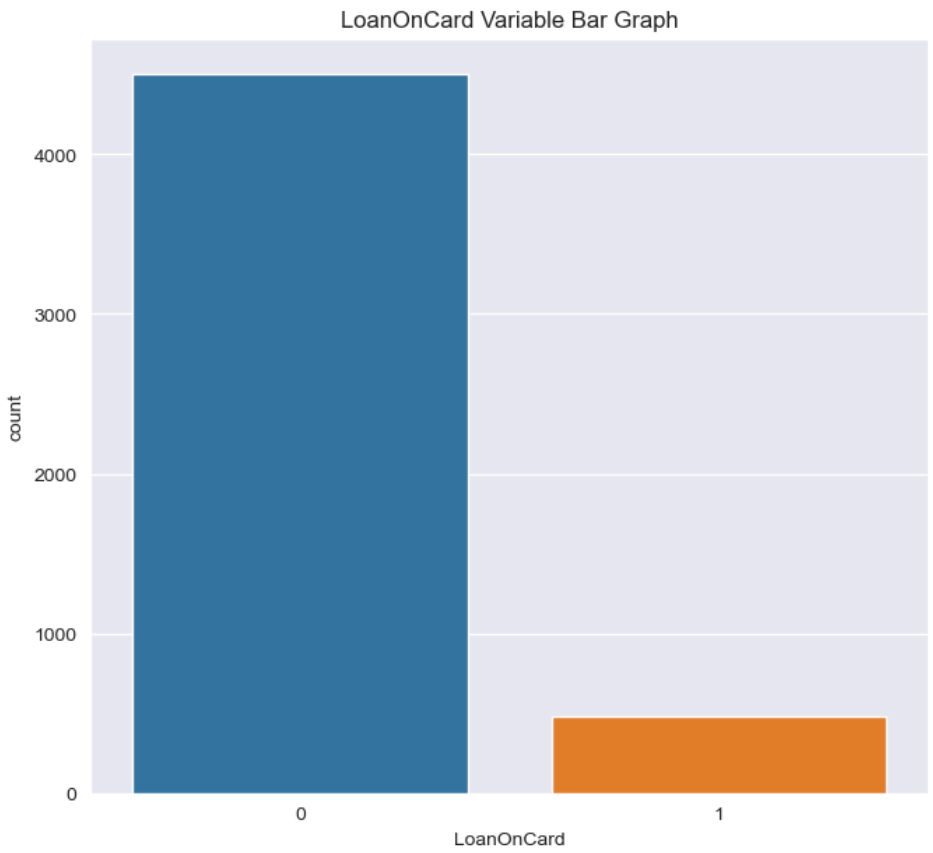
3E i .i. Check distribution of the target variable

Checking on Target Imbalance

In [184…

```
f,axes=plt.subplots(1,2,figsize=(17,7))
newdf['LoanOnCard'].value_counts().plot.pie(autopct='%1.1f%%',ax=axes[0],shadow=True)
sns.countplot(x='LoanOnCard',data=newdf,ax=axes[1],order=[0,1])
axes[0].set_title('LoanOnCard Variable Pie Chart')
axes[1].set_title('LoanOnCard Variable Bar Graph')
plt.show()
```





There is huge imbalance in target variable.

If the imbalanced data is not treated beforehand, then this will degrade the performance of the classifier model. Most of the predictions will correspond to the majority class and treat the minority class features as noise in the data and ignore them. This will result in a high bias in the model.

A widely adopted technique for dealing with highly unbalanced datasets is called resampling

Two widely used resampling methods:

Undersampling: It is the process where you randomly delete some of the observations from the majority class in order to match the numbers with the minority class.

Oversampling: It is the process of generating synthetic data that tries to randomly generate a sample of the attributes from observations in the minority class

Here we will use oversampling because undersampling may remove important information from the dataset

3E iii. Here you need to balance the target variable as 50:50

3E iv. Try appropriate method to achieve the same.

SMOTE for target imbalance

Here we are building model without sampling to achive balance data

Performing SMOTE for all data

In [196…
```
smote_nc=SMOTENC(categorical_features=[1,3,5],random_state=42) #specifying categorical column numbers
x_s,y_s=smote_nc.fit_resample(X,y)
```

In [197…
```
print('Before sampling:')
print(y.value_counts())
```

```
Before sampling:
LoanOnCard
0.0    4500
1.0     480
Name: count, dtype: int64
```

In [198…
```
print('After sampling:')
print(y_s.value_counts())
```

```
After sampling:
LoanOnCard
1.0    4500
0.0    4500
Name: count, dtype: int64
```

we can see the target is balanced after sampling

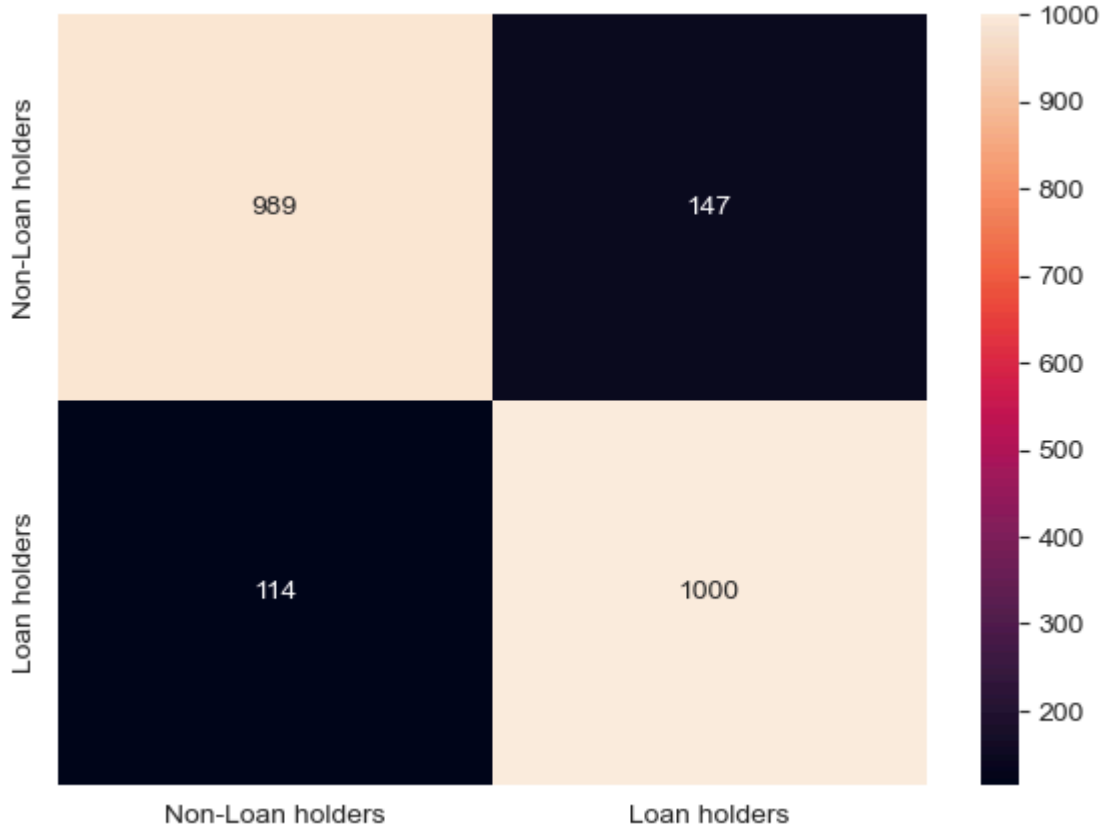
In [199…
```
# Split X and y into training and test set in 50:50 ratio
X_train, X_test, y_train, y_test = train_test_split(x_s, y_s, test_size=0.25, random_state=10)
```

In [201…
```
logit = LogisticRegression()
logit.fit(X_train, y_train)
logit_pred = logit.predict(X_test)

print('Accuracy on Training data:',logit.score(X_train, y_train) )
print('Accuracy on Test data:',logit.score(X_test, y_test) )
```

```
Accuracy on Training data: 0.8746666666666667
Accuracy on Test data: 0.884
```

Here both accuracy is reduced after sampling. Let us check on the classification report.

In [202…
```
cm = confusion_matrix(y_test, logit_pred, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in ["Non-Loan holders","Loan holders"]],
                  columns = [i for i in ["Non-Loan holders","Loan holders"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True ,fmt='g')
plt.show()
```



In [195…

```
print("classification  Matrix:\n",classification_report(y_test,logit_pred))
```

```
classification  Matrix:
               precision    recall  f1-score   support

         0.0       0.88      0.88      0.88      2264
         1.0       0.88      0.88      0.88      2236

    accuracy                           0.88      4500
   macro avg       0.88      0.88      0.88      4500
weighted avg       0.88      0.88      0.88      4500
```

Now we can see recall value is improved after sampling.

So whenever we have imbalance target we will use sampling method to balance the data.

If we do smote on entire data it may leak information to validation data as well. we need to test the model with unseen information. so we will do sampling only on training data.

3F. Again train the same previous model on balanced data. [1 Marks]

we are doing smote only for training data

In [227…
```
# Split X and y into training and test set in 50:50 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.50, random_state=10)
```

In [228…
```
smote_nc=SMOTENC(categorical_features=[1,3,5],random_state=42)
x_train_res, y_train_res = smote_nc.fit_resample(X_train, y_train)
```

In [229…
```
logit = LogisticRegression()
logit.fit(x_train_res, y_train_res)
logit_pred = logit.predict(X_test)

print('Accuracy on Training data:',logit.score(X_train, y_train) )
print('Accuracy on Test data:',logit.score(X_test, y_test) )
```

```
Accuracy on Training data: 0.8650602409638555
Accuracy on Test data: 0.8534136546184738
```

we can see there is decrease in test accuracy.

3 G. Print evaluation metrics and clearly share differences observed. [2 Marks]

In [230…
```
cm = confusion_matrix(y_test, logit_pred, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in ["Non-Loan holders","Loan holders"]],
                  columns = [i for i in ["Non-Loan holders","Loan holders"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True ,fmt='g')
plt.show()
```

In [231…

```
print("classification  Matrix:\n",classification_report(y_test,logit_pred))
```

```
classification  Matrix:
               precision    recall  f1-score   support

         0.0       0.98      0.85      0.91      2253
         1.0       0.38      0.87      0.53       237

    accuracy                           0.85      2490
   macro avg       0.68      0.86      0.72      2490
weighted avg       0.93      0.85      0.88      2490
```

difference found

After doing sampling only on training data we can see difference in values

We are getting good recall value but the precision value is reduced

We will do sampling oly on training data to check real performance of the model

4. Performance Improvement: [10 Marks]

A. Train a base model each for SVM, KNN. [4 Marks]

using Naive Bayes Model

In [47]:
```
g_model = GaussianNB()
g_model.fit(x_train_res, y_train_res.ravel())
g_pred = g_model.predict(X_test)

print('Accuracy on Training data:',g_model.score(X_train, y_train) )
print('Accuracy on Test data:',g_model.score(X_test, y_test) )
```

```
Accuracy on Training data: 0.8795180722891566
Accuracy on Test data: 0.8670682730923694
```

Here accuracy in test data slightly less compared to training data

In [48]:
```
cm = confusion_matrix(y_test, g_pred, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in ["Non-Loan holders","Loan holders"]],
                  columns = [i for i in ["Non-Loan holders","Loan holders"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True ,fmt='g')
plt.show()
```

In [49]:
```
print("classification  Matrix:\n",classification_report(y_test,g_pred))
```

```
classification  Matrix:
               precision    recall  f1-score   support

         0.0       0.98      0.87      0.92      2253
         1.0       0.40      0.82      0.54       237

    accuracy                           0.87      2490
   macro avg       0.69      0.85      0.73      2490
weighted avg       0.92      0.87      0.89      2490
```

Recall value is good for both the classes

Recall value for class 1 is less in naive bayes model compared to logistic regression.

4 B. Tune parameters for each of the models wherever required and finalize a model. [3 Marks]

(Optional: Experiment with various Hyperparameters - Research required

In [239…

```
 # SVM - to find the best parameters

from sklearn.svm import SVC

c=np.arange(0.1,1.1,0.1)  # Range of C values

kernels=['linear','rbf','sigmoid','poly']  # Range of kernels

# Loop to find the best parameters for C and Kernels

for k in range (len(kernels)):
    for i in range (len(c)):
        SVM_Classifier = SVC(C=c[i],kernel=kernels[k])
        SVM_Classifier.fit(X_train, y_train)
        print ('C=',round(c[i],2),"\tKernel=",kernels[k],'\tScore=',SVM_Classifier.score(X_test, y_test))
```

```
C= 0.1  Kernel= linear  Score= 0.9485943775100402
C= 0.2  Kernel= linear  Score= 0.9485943775100402
C= 0.3  Kernel= linear  Score= 0.9485943775100402
C= 0.4  Kernel= linear  Score= 0.9493975903614458
C= 0.5  Kernel= linear  Score= 0.9502008032128514
C= 0.6  Kernel= linear  Score= 0.9502008032128514
C= 0.7  Kernel= linear  Score= 0.9502008032128514
C= 0.8  Kernel= linear  Score= 0.9497991967871486
C= 0.9  Kernel= linear  Score= 0.9497991967871486
C= 1.0  Kernel= linear  Score= 0.9502008032128514
C= 0.1  Kernel= rbf     Score= 0.9048192771084337
C= 0.2  Kernel= rbf     Score= 0.9048192771084337
C= 0.3  Kernel= rbf     Score= 0.9048192771084337
C= 0.4  Kernel= rbf     Score= 0.9048192771084337
C= 0.5  Kernel= rbf     Score= 0.9048192771084337
C= 0.6  Kernel= rbf     Score= 0.9048192771084337
C= 0.7  Kernel= rbf     Score= 0.9048192771084337
C= 0.8  Kernel= rbf     Score= 0.9052208835341365
C= 0.9  Kernel= rbf     Score= 0.9048192771084337
C= 1.0  Kernel= rbf     Score= 0.9048192771084337
C= 0.1  Kernel= sigmoid         Score= 0.9048192771084337
C= 0.2  Kernel= sigmoid         Score= 0.9048192771084337
C= 0.3  Kernel= sigmoid         Score= 0.9048192771084337
C= 0.4  Kernel= sigmoid         Score= 0.9044176706827309
C= 0.5  Kernel= sigmoid         Score= 0.9020080321285141
C= 0.6  Kernel= sigmoid         Score= 0.9020080321285141
C= 0.7  Kernel= sigmoid         Score= 0.9008032128514056
C= 0.8  Kernel= sigmoid         Score= 0.9004016064257028
C= 0.9  Kernel= sigmoid         Score= 0.8979919678714859
C= 1.0  Kernel= sigmoid         Score= 0.8967871485943775
C= 0.1  Kernel= poly    Score= 0.9048192771084337
C= 0.2  Kernel= poly    Score= 0.9048192771084337
C= 0.3  Kernel= poly    Score= 0.9048192771084337
C= 0.4  Kernel= poly    Score= 0.9096385542168675
C= 0.5  Kernel= poly    Score= 0.9180722891566265
C= 0.6  Kernel= poly    Score= 0.921285140562249
C= 0.7  Kernel= poly    Score= 0.9244979919678715
C= 0.8  Kernel= poly    Score= 0.927710843373494
C= 0.9  Kernel= poly    Score= 0.9293172690763052
C= 1.0  Kernel= poly    Score= 0.9293172690763052
```

In [240…

```
#KNN - choosing the K value

# creating odd list of K for KNN
```

```
myList = list(range(2,20))

# subsetting just the odd ones
neighbors = list(filter(lambda x: x % 2 != 0, myList))

# empty list that will hold accuracy scores
ac_scores = []

# perform accuracy metrics for values from 1,3,5....19
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    # predict the response
    y_pred = knn.predict(X_test)
    # evaluate accuracy
    scores = accuracy_score(y_test, y_pred)
    ac_scores.append(scores)

# changing to misclassification error
MSE = [1 - x for x in ac_scores]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print("The optimal number of neighbors is %d" % optimal_k)
```

```
The optimal number of neighbors is 3
```

In [62]:

```
# KNN - Model using the best parameters form above

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

KNN = KNeighborsClassifier(n_neighbors=3)
KNN.fit(X_train, y_train)
# predict the response
y_pred = KNN.predict(X_test)
# evaluate accuracy
KNN_Accuracy=accuracy_score(y_test, y_pred)
print("\nAccuracy using KNN : ", KNN_Accuracy)


#Pickle the file
import pickle
if KNN_Accuracy>=0.1:
```

```
    print("\nInitiating pickling the model")
    KNN_Pickled = pickle.dumps(KNN)
    print("\nCompleted pickling the model")
else:
    print("\nModel with poor performance")

print("\nReload the pickled model")
KNN_Pickled_Load = pickle.loads(KNN_Pickled)

y_pred_1 = KNN_Pickled_Load.predict(X_test)
KNN_Accuracy_Pickled = accuracy_score(y_true=y_test, y_pred=y_pred_1)
print("\nAccuracy using KNN pickled model :", KNN_Accuracy_Pickled)
```

```
Accuracy using KNN :  0.934136546184739

Initiating pickling the model

Completed pickling the model

Reload the pickled model

Accuracy using KNN pickled model : 0.934136546184739
```

K-Fold CV for finding best model

In [241…

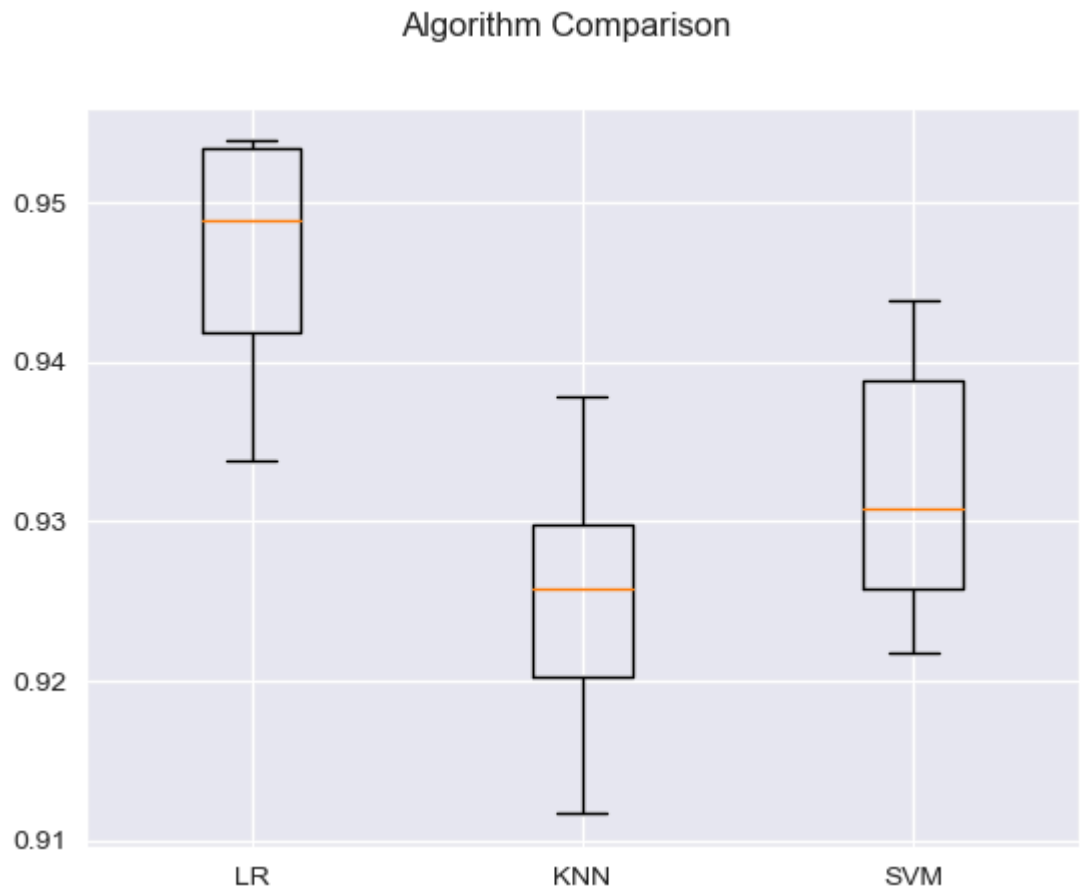
```
LR_model=LogisticRegression()
KNN_model=KNeighborsClassifier(n_neighbors=13)
SVM_model=SVM_Classifier
```

In [242…

```
models = []
models.append(('LR', LR_model))
models.append(('KNN', KNN_model))
models.append(('SVM', SVM_model))

# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
        kfold = model_selection.KFold(n_splits=10)
        cv_results = model_selection.cross_val_score(model,  X,y, cv=kfold, scoring=scoring)
        results.append(cv_results)
```

```
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

```
LR: 0.946787 (0.007254)
KNN: 0.925100 (0.006901)
SVM: 0.932129 (0.007449)
```



here we finalizing LR model

4 C. Print evaluation metrics for final model. [1 Marks

In [246...

```
cm = confusion_matrix(y_test, logit_pred, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in ["Non-Loan holders","Loan holders"]],
                  columns = [i for i in ["Non-Loan holders","Loan holders"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True ,fmt='g')
plt.show()
```

In [247…
```
print("classification  Matrix:\n",classification_report(y_test,logit_pred))
```

```
classification  Matrix:
               precision    recall  f1-score   support

         0.0       0.98      0.85      0.91      2253
         1.0       0.38      0.87      0.53       237

    accuracy                           0.85      2490
   macro avg       0.68      0.86      0.72      2490
weighted avg       0.93      0.85      0.88      2490
```

4D. Share improvement achieved from base model to final model. [2 Marks

6.Conclusion and improvisation.

We are selecting final model as logistic regression as it performs well in training and testing test.

Logistic Regression is not affected by overfitting and it is also has good recall value.

Logistic regression performed well in k-fold cross validation as well.

Deviation also less in logistic regression.

Sampling improved to predict minority classes as well

Suggesting to collect data equally for both the classes.

Few customers does't have credit card but those customer having loan on card. This data error can be avoided

========== End Part B ==========

In [ ]: