manubhai_asign1

May 24, 2024

PART-A

1 Data understanding

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
df_1=pd.read_csv(r"C:\Users\LENOVO\Desktop\f1.csv")
```

[3]:
```
df_2=pd.read_csv(r"C:\Users\LENOVO\Desktop\f2.csv")
```

[4]:
```
df_3=pd.read_csv(r"C:\Users\LENOVO\Desktop\f3.csv")
```

[5]:
```
df_1.shape
```

[5]:
```
(150, 7)
```

[6]:
```
df_2.shape
```

[6]:
```
(60, 7)
```

[7]:
```
df_3.shape
```

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

[12]:
```
df_1.columns
```

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

[13]:
```
df_2.columns
```

[13]:
```
Index(['P_incidence', 'P_tilt', 'L_angle', 'S_slope', 'P_radius', 'S_Degree',
       'Class'],
      dtype='object')
```

[14]:
```
df_3.columns
```

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

[15]:
```
check=(df_1.columns==df_2.columns).all() and (df_2.columns==df_3.columns).all()
```

[16]:
```
if check:
    print("all data frames have similar columns with same order")
else:
    print("all data frames are distinct")
```

```
all data frames have similar columns with same order
```

The output confirms that all DataFrames have the same column names in the same order, you can proceed with merging the datasets by rows confidently.

[20]:
```
df_1.dtypes
```

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

[21]:
```
df_2.dtypes
```

[21]:
```
P_incidence    float64
P_tilt         float64
L_angle        float64
S_slope        float64
P_radius       float64
S_Degree       float64
Class           object
dtype: object
```

[22]:
```
df_3.dtypes
```

[22]:
```
P_incidence    float64
P_tilt         float64
L_angle        float64
S_slope        float64
P_radius       float64
S_Degree       float64
Class           object
dtype: object
```

[24]:
```
df_1['Class'].unique()
```

[24]:
```
array(['Type_S', 'tp_s'], dtype=object)
```

[25]:
```
df_2['Class'].unique()
```

[25]:
```
array(['Type_H', 'type_h'], dtype=object)
```

[26]:
```
df_3['Class'].unique()
```

[26]:
```
array(['Normal', 'Nrmal'], dtype=object)
```

2 Data Preparation and Exploration

[27]:
```
df_1['Class']='type_s'
```

[28]:
```
df_2['Class']='type_h'
```

[29]:
```
df_3['Class']='normal'
```

[58]:
```
df_4=pd.concat([df_1,df_2,df_3],axis=0)
```

[59]:
```
df_4.shape
```

[59]:
```
(310, 7)
```

[60]:
```
df_4.sample(5)
```

[60]:
```
     P_incidence     P_tilt    L_angle    S_slope    P_radius   S_Degree  \
118    80.654320  26.344379  60.898118  54.309940  120.103493  52.467552
12     56.103774  13.106307  62.637020  42.997467  116.228503  31.172767
42     53.854798  19.230643  32.779060  34.624155  121.670915   5.329843
56     46.374088  10.215902  42.700000  36.158185  121.247657  -0.542022
8      72.076278  18.946176  51.000000  53.130102  114.213013   1.010041

      Class
118  type_s
12   normal
42   type_h
56   normal
8    type_s
```

[61]:
```
null_percentage=(df_4.isnull().mean())*100
print(null_percentage)
```

```
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
```

[62]:
```python
df_4.describe()
```

[62]:

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

[49]:
```python
import seaborn as sns
```

[74]:
```python
# Assuming df_4 is your existing DataFrame
df_5 = df_4.iloc[:,0:6]
```

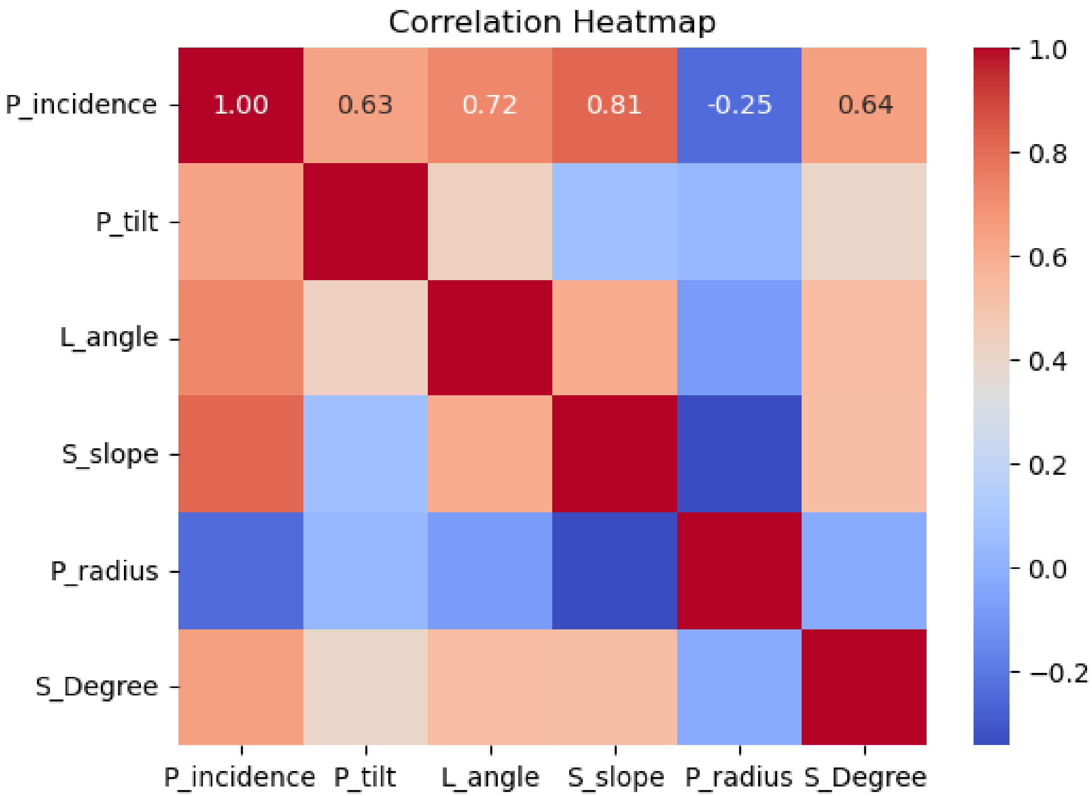
[75]:
```python
df_5
```

[75]:

	P_incidence	P_tilt	L_angle	S_slope	P_radius	S_Degree
0	74.377678	32.053104	78.772013	42.324573	143.560690	56.125906
1	89.680567	32.704435	83.130732	56.976132	129.955476	92.027277
2	44.529051	9.433234	52.000000	35.095817	134.711772	29.106575
3	77.690577	21.380645	64.429442	56.309932	114.818751	26.931841
4	76.147212	21.936186	82.961502	54.211027	123.932010	10.431972
..	…	…	…	…	…	…
95	47.903565	13.616688	36.000000	34.286877	117.449062	-4.245395
96	53.936748	20.721496	29.220534	33.215251	114.365845	-0.421010
97	61.446597	22.694968	46.170347	38.751628	125.670725	-2.707880
98	45.252792	8.693157	41.583126	36.559635	118.545842	0.214750
99	33.841641	5.073991	36.641233	28.767649	123.945244	-0.199249

[310 rows x 6 columns]

[76]:
```python
sns.heatmap(df_5.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```

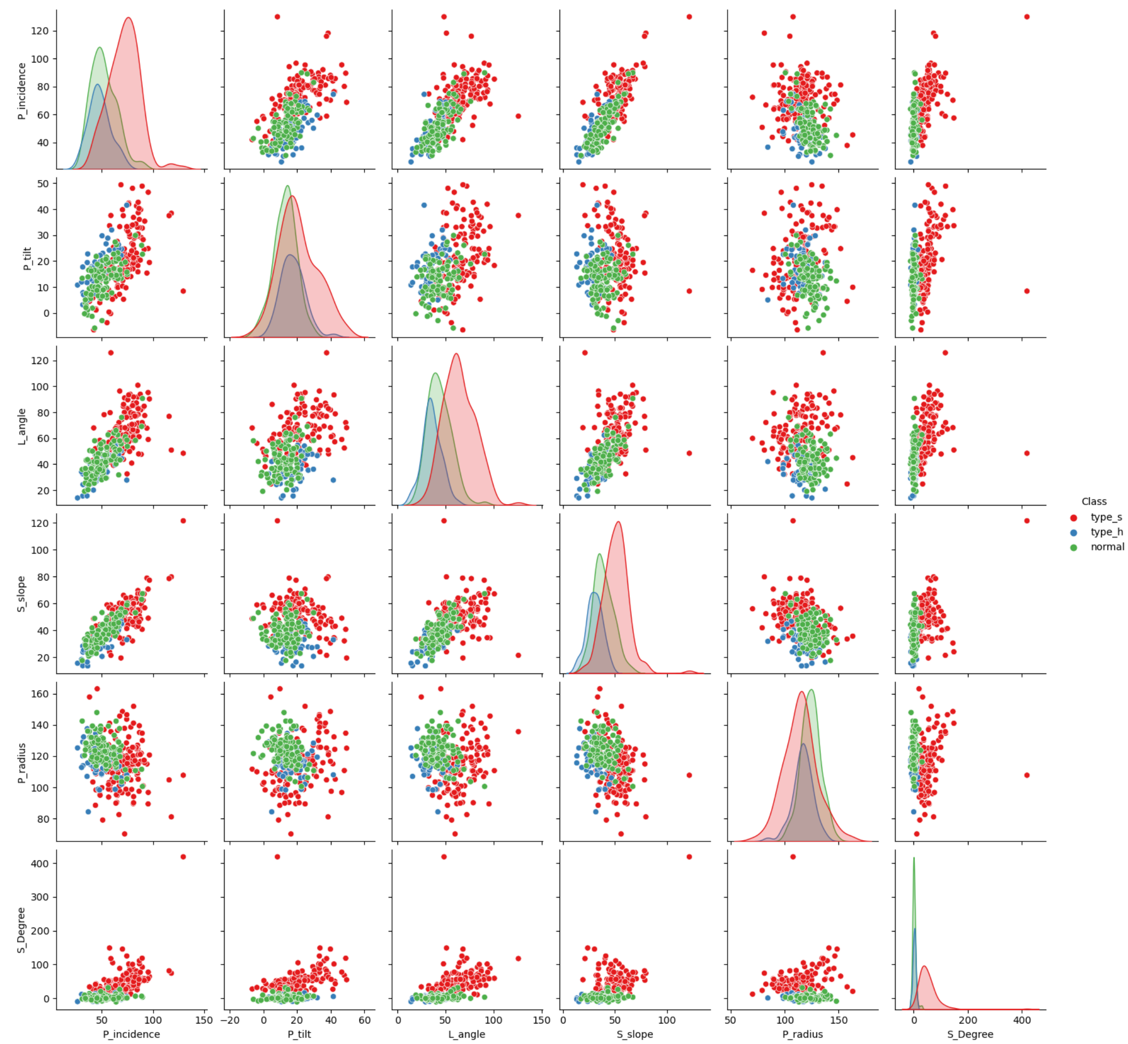


1)POSITIVE CORRELATION S_slope and P_incidence are highly correlated attributes followed by L_angle and P_incidence 2)NEGATIVE CORRELATION P_radius and P_incidence are negatively correlated showing increase in the value of one will result in decrease in teh value of the other

[79]:
```
sns.pairplot(df_4, hue='Class', palette='Set1')
plt.show()
```

```
C:\Users\LENOVO\Documents\anaconda\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\LENOVO\Documents\anaconda\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\LENOVO\Documents\anaconda\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\LENOVO\Documents\anaconda\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
```

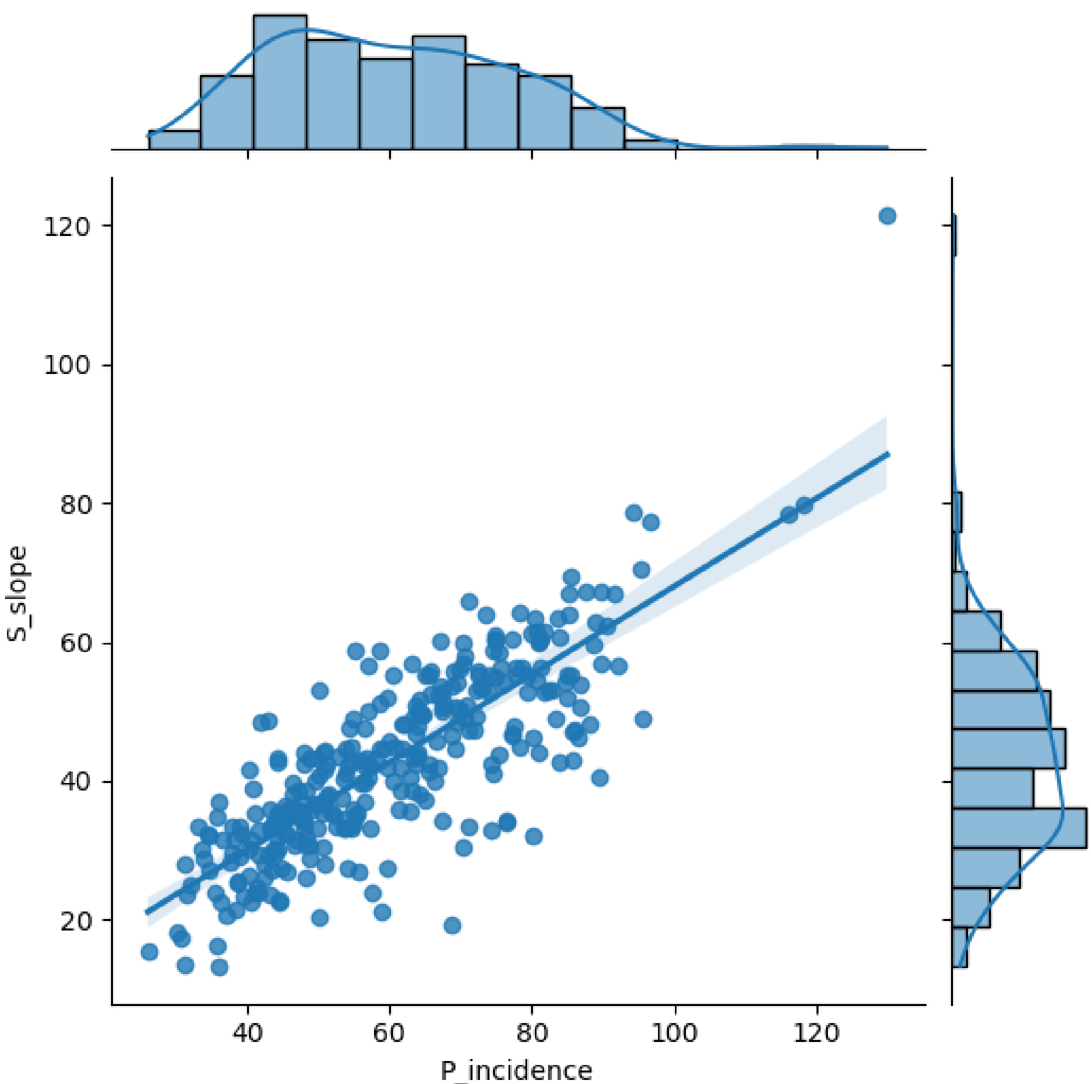
```
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\LENOVO\Documents\anaconda\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\LENOVO\Documents\anaconda\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



The above pair plot shows normal distribution curves for every attribute except for S_degree as it has left skewness All the values except for S_Degree lies in the mean range of the population

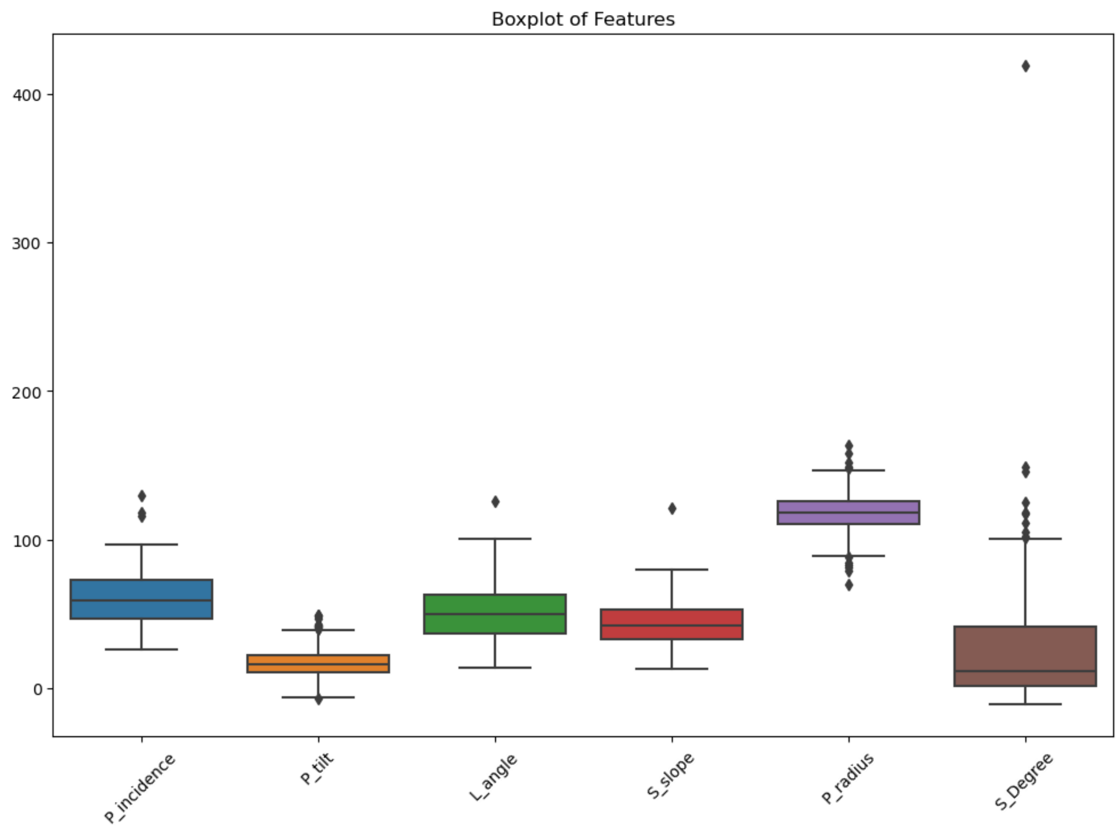
[87]:
```
pd.set_option('mode.use_inf_as_na', False)
sns.jointplot(x='P_incidence', y='S_slope', data=df_5, kind='reg')
plt.show()
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_24552\565059688.py:1:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  pd.set_option('mode.use_inf_as_na', False)
C:\Users\LENOVO\Documents\anaconda\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\LENOVO\Documents\anaconda\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



As shown above the variables P_incidence and S_slope are having a linear relationship showing high density of data points around the best fit line Some outliers are also present as indicated by the pendant data point The histogram aty the top seems to be normally distributed but show a slight left skewness showing some values of P_incidence have values between 40 to 60 Histogram on right hand side clearly shows left skewness showing majority of data points have low values of S-slope

[88]:
```
plt.figure(figsize=(12, 8))
sns.boxplot(data=df_4)
plt.title('Boxplot of Features')
plt.xticks(rotation=45)
plt.show()
```



The points lying beyond the wiskers of every box plot of an atrribute is an outlier Every attribute has its median close to its mean value except for S_degree where the median is greater than its mean value

4 MODEL BUILDING

[138]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
X=df_4.iloc[:,:-1]
Y=df_4['Class']
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=6)
```

[139]:
```
knn=KNeighborsClassifier(n_neighbors =5)
```

[140]:
```
knn.fit(x_train,y_train)
```

[140]:
```
KNeighborsClassifier()
```

[141]:
```
y_predict=knn.predict(x_test)
```

[142]:
```
from sklearn.metrics import classification_report
```

[143]:
```
report=classification_report(y_predict,y_test)
print(report)
```

```
              precision    recall  f1-score   support

      normal       0.81      0.85      0.83        26
      type_h       0.75      0.82      0.78        11
      type_s       0.96      0.88      0.92        25

    accuracy                           0.85        62
   macro avg       0.84      0.85      0.84        62
weighted avg       0.86      0.85      0.86        62
```

[144]:
```
y_pred=knn.predict(x_train)
```

[145]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix
train_accuracy = accuracy_score(y_train, y_pred)
test_accuracy = accuracy_score(y_test, y_predict)
train_precision = precision_score(y_train, y_pred, average='weighted')
test_precision = precision_score(y_test, y_predict, average='weighted')
train_recall = recall_score(y_train, y_pred, average='weighted')
test_recall = recall_score(y_test, y_predict, average='weighted')
train_f1_score = f1_score(y_train, y_pred, average='weighted')
test_f1_score = f1_score(y_test, y_predict, average='weighted')
train_confusion_matrix = confusion_matrix(y_train, y_pred)
test_confusion_matrix = confusion_matrix(y_test, y_predict)

print("Train Data Metrics:")
print("Accuracy:", train_accuracy)
print("Precision:", train_precision)
```

```
print("Recall:", train_recall)
print("F1-score:", train_f1_score)
print("Confusion Matrix:")
print(train_confusion_matrix)

print("\nTest Data Metrics:")
print("Accuracy:", test_accuracy)
print("Precision:", test_precision)
print("Recall:", test_recall)
print("F1-score:", test_f1_score)
print("Confusion Matrix:")
print(test_confusion_matrix)
```

```
Train Data Metrics:
Accuracy: 0.9112903225806451
Precision: 0.913231780167264
Recall: 0.9112903225806451
F1-score: 0.9108742959549411
Confusion Matrix:
[[ 66   7   0]
 [ 13  35   0]
 [  2   0 125]]

Test Data Metrics:
Accuracy: 0.8548387096774194
Precision: 0.8532957365215429
Recall: 0.8548387096774194
F1-score: 0.8530601938835817
Confusion Matrix:
[[22  2  3]
 [ 3  9  0]
 [ 1  0 22]]
```

[146]:
```
from sklearn.model_selection import GridSearchCV
knn_1 = KNeighborsClassifier()
param_grid = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [10, 30, 50],
    'metric': ['euclidean', 'manhattan']
}
grid_search = GridSearchCV(knn_1, param_grid, cv=5, scoring='accuracy',␣
 ↪verbose=1, n_jobs=-1)
grid_search.fit(X, Y)

print("Best Parameters:", grid_search.best_params_)
```

```
print("Best Score:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
Best Parameters: {'algorithm': 'auto', 'leaf_size': 10, 'metric': 'euclidean',
'n_neighbors': 5, 'weights': 'uniform'}
Best Score: 0.8419354838709678
```

[210]:
```
knn_2 = KNeighborsClassifier(algorithm='ball_tree', leaf_size=50,␣
↪metric='euclidean', n_neighbors=5, weights='distance')
```

[211]:
```
knn_2.fit(x_train,y_train)
```

[211]:
```
KNeighborsClassifier(algorithm='ball_tree', leaf_size=50, metric='euclidean',
                     weights='distance')
```

[212]:
```
y_predict1=knn_2.predict(x_test)
```

[213]:
```
report_1=classification_report(y_predict1,y_test)
print(report_1)
```

```
              precision    recall  f1-score   support

      normal       0.81      0.85      0.83        26
      type_h       0.75      0.82      0.78        11
      type_s       0.96      0.88      0.92        25

    accuracy                           0.85        62
   macro avg       0.84      0.85      0.84        62
weighted avg       0.86      0.85      0.86        62
```

[214]:
```
y_pred1=knn_2.predict(x_train)
```

[215]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, confusion_matrix
train_accuracy1 = accuracy_score(y_train, y_pred1)
test_accuracy1 = accuracy_score(y_test, y_predict1)
train_precision1 = precision_score(y_train, y_pred1, average='weighted')
test_precision1 = precision_score(y_test, y_predict1, average='weighted')
train_recall1 = recall_score(y_train, y_pred1, average='weighted')
test_recall1 = recall_score(y_test, y_predict1, average='weighted')
train_f1_score1 = f1_score(y_train, y_pred1, average='weighted')
test_f1_score1 = f1_score(y_test, y_predict1, average='weighted')
train_confusion_matrix1 = confusion_matrix(y_train, y_pred1)
test_confusion_matrix1 = confusion_matrix(y_test, y_predict1)

print("Train Data Metrics:")
print("Accuracy:", train_accuracy1)
```

```
print("Precision:", train_precision1)
print("Recall:", train_recall1)
print("F1-score:", train_f1_score1)
print("Confusion Matrix:")
print(train_confusion_matrix1)

print("\nTest Data Metrics:")
print("Accuracy:", test_accuracy1)
print("Precision:", test_precision1)
print("Recall:", test_recall1)
print("F1-score:", test_f1_score1)
print("Confusion Matrix:")
print(test_confusion_matrix1)
```

```
Train Data Metrics:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
Confusion Matrix:
[[ 73   0   0]
 [  0  48   0]
 [  0   0 127]]

Test Data Metrics:
Accuracy: 0.8548387096774194
Precision: 0.8532957365215429
Recall: 0.8548387096774194
F1-score: 0.8530601938835817
Confusion Matrix:
[[22  2  3]
 [ 3  9  0]
 [ 1  0 22]]
```

At the end of multiple hit and try attempts the above combinations provides best results We find that the parameters n_neighbors,weights and metric cause the most of the impact on the model performance

[ ]:

```

```