Code Snippets

```
import numpy as np import pandas as pd import seaborn as sns import
matplotlib.pyplot as plt from sklearn.model_selection import
train_test_split from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier from sklearn
import metrics from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score, roc_auc_score <importing
necessary libraries>
df        =        pd.read_csv("/content/drive/MyDrive/Parkinsson
disease.csv") <uploading the csv file>
df.head(100 shows the first 100
rows>
df.shape <shows the shape of the
dataset>
df.info()<shows concise summary of
dataset>
df.isnull().sum() <used to find the null values in a data
frame>
import seaborn as sns import matplotlib.pyplot as plt
sns.countplot(x='status', data=df) plt.title('Class Distribution')
plt.show() <Used to create a count plot using Seaborn to visualize
the distribution of classes in a dataset>

= df.drop(columns=['name', 'status'], axis=1) <X will contain all the
features except the 'name' & 'status' columns from the original Data
Frame>
= df['status'] <Y will contain the target variable values (status)
from the original Data Frame>

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= 0.2,
random_state= 42) <Used to call train_test_split function provided by the
Scikit-learn library>

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test) <performs feature scaling using
the StandardScaler from scikit-learn>
```

```
smote = SMOTE(random_state=42)
X_train_balanced, Y_train_balanced = smote.fit_resample(X_train_scaled,
Y_train) <Used to balance the imabalance data>
print(pd.Series(Y_train_balanced).value_counts()) <Used to print
the counts of unique values in the Pandas Series Y_train_balanced>
knn = KNeighborsClassifier(n_neighbors=5) <Used to create an instance
of the KNN classifier configured to consider the 5 nearest neighbors
when making predictions>
knn.fit(X_train_balanced, Y_train_balanced) <Used to train a
KNN classifier model using the training data>

Y_pred = knn.predict(X_test_scaled) <Used to make predictions on the test
data (X_test_scaled) using a trained KNN classifier >
print("Confusion Matrix:\n", confusion_matrix(Y_test, Y_pred))
print("\nClassification Report:\n", classification_report(Y_test,
Y_pred)) print("\nAccuracy Score:", accuracy_score(Y_test, Y_pred)) <
Used to evaluate the performance of a ML classifier by printing out the
confusion matrix, classification report, & accuracy score>
cm = confusion_matrix(Y_test, Y_pred)
sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues') plt.title('Confusion Matrix')
plt.xlabel('Predicted Label') plt.ylabel('True
Label')
plt.show() <creates a heatmap visualization of the confusion matrix using
Seaborn & Matplotlib libraries>

class_0_metrics = {'Precision': 0.60, 'Recall': 0.86, 'F1-score':
0.71} class_1_metrics = {'Precision': 0.97, 'Recall': 0.88, 'F1-
score': 0.92} overall_accuracy = 0.87 macro_avg_f1 = 0.81
weighted_avg_f1 = 0.88
categories =
list(class_0_metrics.keys())

class_0_values = [class_0_metrics[cat] for cat in categories]
class_1_values = [class_1_metrics[cat] for cat in categories]
bar_width =
0.35
index = range(len(categories))
```

Challenges faced during implementation and their solutions

```
fig, ax = plt.subplots() bar1 = ax.bar(index, class_0_values, bar_width,
label='Class 0', color='blue') bar2 = ax.bar([i + bar_width for i in index],
class_1_values, bar_width, label='Class 1', color='red')
ax.axhline(y=overall_accuracy, color='green', linestyle='-', label='Overall
Accuracy') ax.axhline(y=macro_avg_f1, color='orange', linestyle='--',
label='Macroaverage F1-score') ax.axhline(y=weighted_avg_f1, color='purple',
linestyle=':', label='Weighted average F1-score')
ax.set_xlabel('Metrics')
ax.set_ylabel('Percentage')
ax.set_title('Performance Metrics Comparison')
ax.set_xticks([i + bar_width/2 for i in index])
ax.set_xticklabels(categories) ax.legend()
plt.show() <visualizes the performance metrics comparison between two
classes in a bar chart, along with horizontal lines representing overall
accuracy, macro-average F1-score, & weighted average F1-score>
```