In [*]:
```
#Parkinson's Disease Detection using Machine Learning
```

In [*]:
```
# Data manipulation
import numpy as np
import pandas as pd
import seaborn as sns

# Model training and evaluation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import accuracy_score
```

In [ ]:
```

```

In [*]:
```
# loading the data from csv file to a Pandas DataFrame
parkisons_data = pd.read_csv("parkisons.csv")
```

In [*]:
```
# number of rows and columns in the dataframe
parkisons_data.shape
```

In [*]:
```
# printing the first 5 rows of the dataframe
parkisons_data.head()
```

In [*]:
```
# getting more information about the dataset
parkisons_data.info()
```

In [*]:
```
# checking for missing values in each column
parkisons_data.isnull().sum()
```

In [*]:
```
# getting some statistical measures about the data
parkisons_data.describe()
```

In [*]:
```
# Exploratory Data Analysis (EDA)
```

In [*]:
```
# Check for missing values
print(parkisons_data.isnull().sum())

# Visualize the distribution of the target variable
sns.countplot(x='status', data=parkisons_data)
plt.title("Distribution of Parkinson's Disease Status")
plt.xlabel("Status")
plt.ylabel("Count")
plt.show()

# Correlation heatmap
plt.figure(figsize=(15, 10))
sns.heatmap(parkisons_data.corr(), annot=True, cmap="coolwarm")
plt.title("Feature Correlation Matrix")
plt.show()
```

In [*]:
```
# distribution of target Variable
parkisons_data['status'].value_counts()
```

In [*]:
```
#1 --> Parkinson's Positive

#0 --> Healthy
```

In [*]:
```
import matplotlib.pyplot as plt

parkisons_data['status'].value_counts().plot(kind='bar')
plt.title('Distribution of Parkinson\'s Disease Status')
plt.xlabel('Status')
plt.ylabel('Count')
plt.show()
```

In [*]:
```
# grouping the data bas3ed on the target variable
parkisons_data.groupby('status').mean()
```

In [*]:
```
#Data Pre-Processing

#Separating the features & Target
```

In [*]:
```
X = parkisons_data.drop(columns=['name','status'], axis=1)# Drop the target column to g
Y = parkisons_data['status']# Target column
```

In [*]:
```
print(X)
```

In [*]:
```
print(Y)
```

In [*]:
```
#Splitting the data to training data & Test data
```

In [*]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2
```

In [29]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(195, 22) (156, 22) (39, 22)
```

In [31]:
```
#Data Standardization
```

In [*]:
```
scaler = StandardScaler()
```

In [33]:
```
scaler.fit(X_train)
```

Out[33]:
```
StandardScaler
StandardScaler()
```

In [*]:
```
X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)
```

In [*]:
```
print(X_train)
```

In [*]:
```
# Feature Engineering - Create feature matrix X and target vector y
X = parkisons_data.drop(columns=['name', 'status'])
y = parkisons_data['status']

# Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # Ensure X_scaled is defined here

# Train-Test Split
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_
```

In [*]:
```
# Hyperparameter tuning using GridSearchCV for SVM
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf']
}

grid_search = GridSearchCV(SVC(probability=True), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Display best parameters and model evaluation
print("Best Parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

In [37]:
```
model = svm.SVC(kernel='linear')
```

In [*]:
```
 #training the SVM model with training data
model.fit(X_train, Y_train)
```

In [*]:
```
#Model Evaluation

#Accuracy Score
```

In [*]:
```
# accuracy score on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
```

In [41]:
```
print('Accuracy score of training data : ', training_data_accuracy)
```

```
Accuracy score of training data :  0.8846153846153846
```

In [42]:
```
# accuracy score on training data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
```

In [43]:
```
print('Accuracy score of test data : ', test_data_accuracy)
```

```
Accuracy score of test data :  0.8717948717948718
```

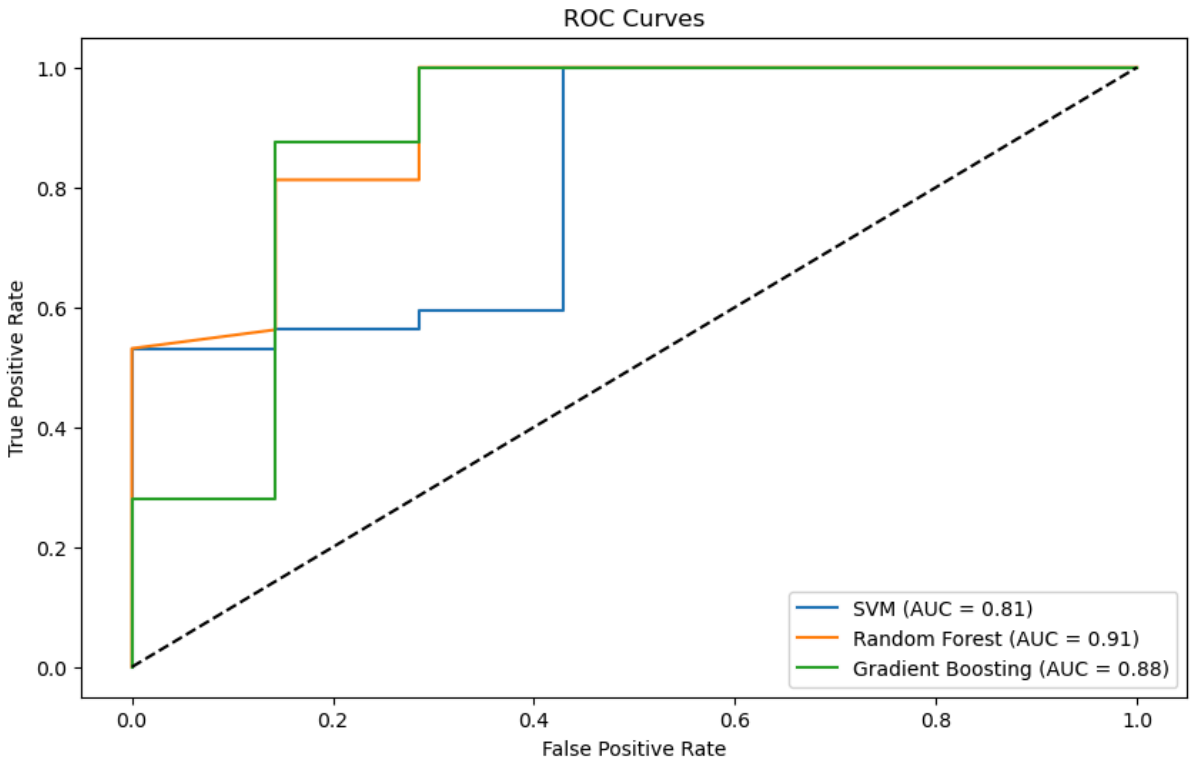
In [44]:
```
#Building a Predictive System
```

In [45]:
```
input_data = (197.07600,206.89600,192.05500,0.00289,0.00001,0.00166,0.00168,0.00498,0.0

# changing input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the data
std_data = scaler.transform(input_data_reshaped)

prediction = model.predict(std_data)
print(prediction)


if (prediction[0] == 0):
  print("The Person does not have Parkinsons Disease")

else:
  print("The Person has Parkinsons")
```

```
[0]
The Person does not have Parkinsons Disease

C:\Users\HP\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not h
ave valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

In [*]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

# Initialize and train models
rf_model = RandomForestClassifier(random_state=42)
gb_model = GradientBoostingClassifier(random_state=42)

rf_model.fit(X_train, y_train)
gb_model.fit(X_train, y_train)

# Evaluate performance
for name, model in [("Random Forest", rf_model), ("Gradient Boosting", gb_model)]:
    y_pred = model.predict(X_test)
    print(f"{name} Performance:")
    print(classification_report(y_test, y_pred))
```

In [74]:
```
from sklearn.metrics import roc_curve, roc_auc_score

# Generate ROC curves for each model
models = {"SVM": best_model, "Random Forest": rf_model, "Gradient Boosting": gb_model}
plt.figure(figsize=(10, 6))

for name, model in models.items():
    y_prob = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    auc_score = roc_auc_score(y_test, y_prob)
    plt.plot(fpr, tpr, label=f"{name} (AUC = {auc_score:.2f})")

plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curves")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```



In [*]:
```
import pickle

# Save the trained model (replace 'best_model' with your actual model variable)
with open("trained_model.pkl", "wb") as model_file:
    pickle.dump(best_model, model_file)  # Use your actual model variable name

# Save the scaler (replace 'scaler' with your actual scaler variable)
with open("scaler.pkl", "wb") as scaler_file:
    pickle.dump(scaler, scaler_file)  # Use your actual scaler variable name
```

In [*]:
```
%%writefile app.py
import numpy as np
import pandas as pd
import streamlit as st
import pickle

# Load the trained model and scaler
with open("trained_model.pkl", "rb") as model_file:
    model = pickle.load(model_file)

with open("scaler.pkl", "rb") as scaler_file:
    scaler = pickle.load(scaler_file)

# Streamlit UI
st.title("Parkinson's Disease Detection App")

# User input
user_input = st.text_input("Enter 22 feature values separated by commas")

if st.button("Predict"):
    if user_input:
        try:
            # Convert input to numpy array
            input_data = np.array([float(x) for x in user_input.split(",")]).reshape(1,
            input_data_scaled = scaler.transform(input_data)
            prediction = model.predict(input_data_scaled)

            result = "Positive for Parkinson's Disease" if prediction[0] == 1 else "Hea
            st.write("Prediction:", result)
        except ValueError:
            st.error("Please enter valid numeric values.")
    else:
        st.warning("Please enter the feature values.")
```

In [*]:

In [*]:

In [*]:

In [*]:

In [*]:

In [*]:

In [*]:

In [*]:

In [*]:

In [*]:

In [*]:

In [*]:

In [ ]: