Data Mining Project Report

Aim: Internal positioning using Wi-Fi signal strengths

Group Members: Prajual Pillai(193100069)
Sudip Walter Thomas(193100063)
Amitkumar Dubey(193100062)
Vivek Revi(193100076)

Description:

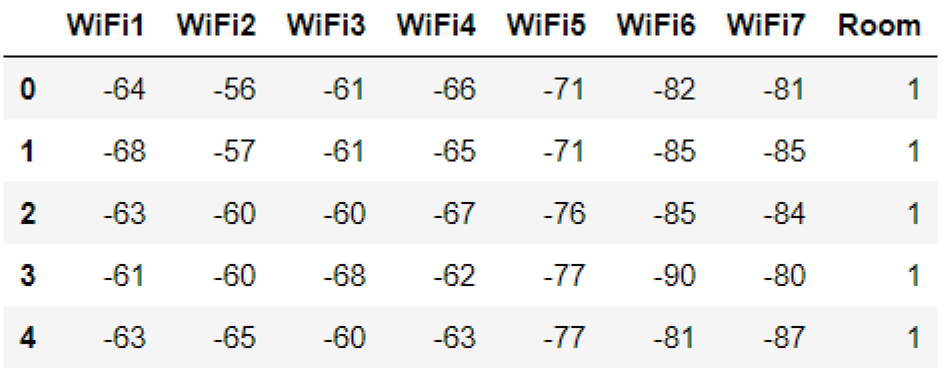
We have a data set containing signal strengths of seven different WiFi routers located in four rooms. The number of observations are 2000. Based upon the signal strength of all the seven WiFi signals recorded in the device at a particular moment we aim to determine the location of the device using various classification methods.

Data Information:

- 2000 rows, 8 columns
- 7 features (WiFi1,WiFi2,WiFi3,WiFi4,WiFi5,WiFi6,WiFi7)
- 1 output (Room No. 1/2/3/4)
- Link to Data : https://code.datasciencedojo.com/datasciencedojo/datasets/tree/master/Wireless%20Indoor%20Localization

Original Data Set:

The data collected from all the four rooms have been stored in an excel file named “proj” where each column shows the signal strength at a particular location of different WiFi signals.
The first 5 rows of the original data set has been shown for reference.



	WiFi1	WiFi2	WiFi3	WiFi4	WiFi5	WiFi6	WiFi7	Room
0	-64	-56	-61	-66	-71	-82	-81	1
1	-68	-57	-61	-65	-71	-85	-85	1
2	-63	-60	-60	-67	-76	-85	-84	1
3	-61	-60	-68	-62	-77	-90	-80	1
4	-63	-65	-60	-63	-77	-81	-87	1

Knocking off 10% Data from the Data Set:

Here we are knocking off 10% of the data from our data set in order to simulate a real life scenario.

```
for i in range(1,8):
    X[f'WiFi{i}'] = X[f'WiFi{i}'].sample(frac=0.9)
```

	WiFi1	WiFi2	WiFi3	WiFi4	WiFi5	WiFi6	WiFi7
0	-64.0	-56.0	-61.0	-66.0	-71.0	-82.0	-81.0
1	NaN	-57.0	-61.0	-65.0	-71.0	-85.0	-85.0
2	-63.0	-60.0	NaN	-67.0	-76.0	-85.0	-84.0
3	NaN	-60.0	-68.0	-62.0	-77.0	-90.0	-80.0
4	NaN	-65.0	-60.0	-63.0	-77.0	-81.0	-87.0

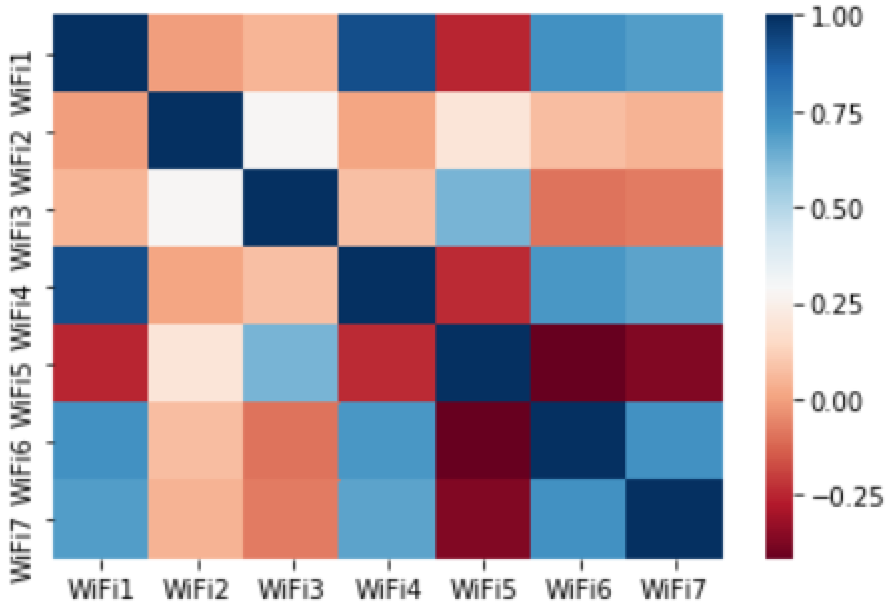
Table above shows the first 5 rows

The obtained file is exported as a csv file for further use and renamed as “proj1.csv”.

```
df_n.to_csv("proj1.csv", header=True, index = False, mode="w")
```

Plotting the Correlation Heatmap between features:

We are plotting the Correlation Heatmap to find if the features are related or not i.e, if the signal strengths are dependent on each other.



From the above heatmap we observe that majority of the features shows some correlation among themselves. But since the number of features are less in this dataset, we cannot afford to delete the features.

Imputing Data:

Here we are imputing missing data using four methods

- Mean
- Median
- KNN regression
- Decision Tree

Creating the model using various classification techniques:

After imputing the data using the above mentioned methods, we try to fit the model using various classification techniques such as

- LDA
- QDA
- SVM
- Decision tree
- KNN

At the end of each classification technique, we create a confusion matrix and an accuracy score to determine how good that model is in predicting the results. Finally, we take the model with the highest accuracy and apply K Fold Cross Validation to boost its accuracy further.

Imputing using Mean and Predicting the Model:

The missing values are imputed using the mean of the data points in the column.

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp.fit(df)
```

	0	1	2	3	4	5	6	7
0	-64.000000	-56.0	-61.000	-66.0	-71.0	-82.0	-81.0	1.0
1	-52.362222	-57.0	-61.000	-65.0	-71.0	-85.0	-85.0	1.0
2	-63.000000	-60.0	-54.955	-67.0	-76.0	-85.0	-84.0	1.0
3	-52.362222	-60.0	-68.000	-62.0	-77.0	-90.0	-80.0	1.0
4	-52.362222	-65.0	-60.000	-63.0	-77.0	-81.0	-87.0	1.0

Table above shows first 5 rows for reference

All the missing values have been imputed using SimpleImputer from sklearn library choosing the strategy as mean.

Converting the Output values into Binary:

This is done for using 'OneVsRestClassfier' library to print ROC and AUC curve later on.

```
X=df.iloc[:,:-1].values
y=df.iloc[:,7].values
y = label_binarize(y, classes=[1,2,3,4])
```

Splitting the Data Set into Train and Test data:

The overall dataset has been split into 70% training set and 30% testing test for determining the accuracy and obtaining the confusion matrix.

```
# Split data to test and train
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, random_state = 2)
```

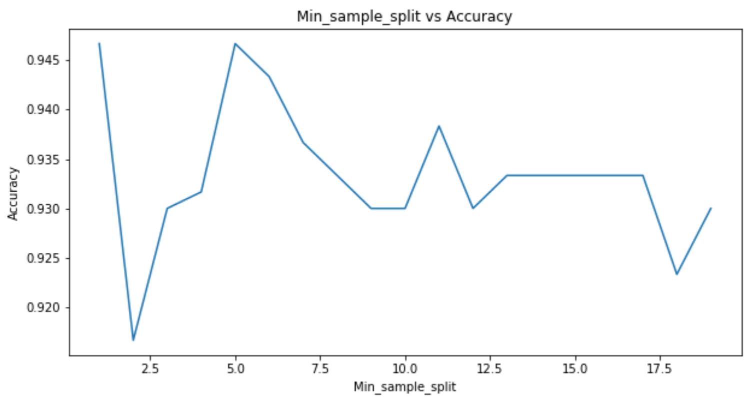
Creating a Model with Decision Trees:

Here we are implementing Decision Tree with the help of sklearn library. In order to obtain the optimum sample split and depth of the Decision Tree model, we have done hyper-parametric tuning. The optimum values are then chosen from the obtained results.

To find the variation of accuracy with sample split we use:

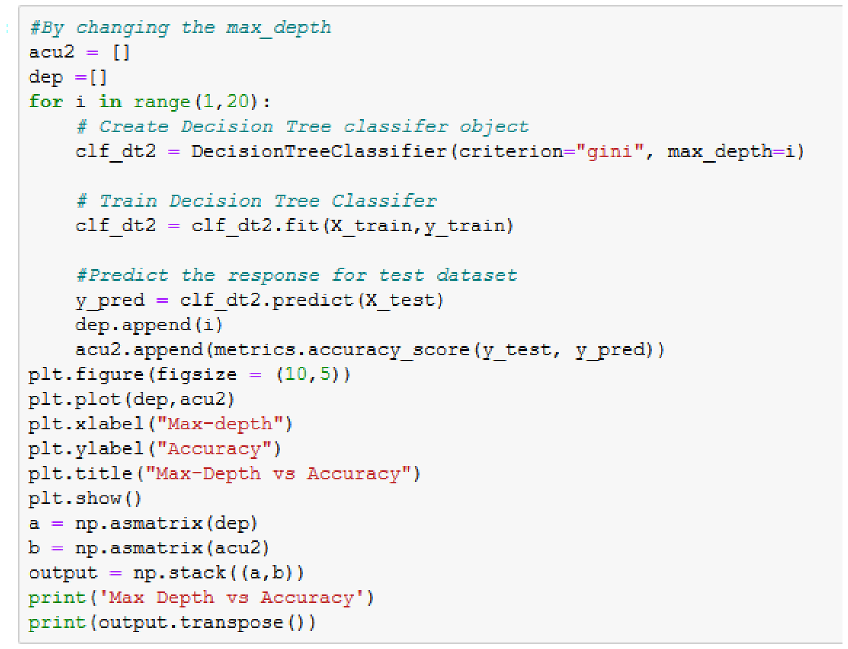
```python
# By changing the min_sample_split
acu1 = []
splits =[]
for i in range(2,20):
    # Create Decision Tree classifer object
    clf_dt1 = DecisionTreeClassifier(criterion="gini", min_samples_split=i)

    # Train Decision Tree Classifer
    clf_dt1 = clf_dt1.fit(X_train,y_train)

    #Predict the response for test dataset
    y_pred = clf_dt1.predict(X_test)
    splits.append(i)
    acu1.append(metrics.accuracy_score(y_test, y_pred))
plt.figure(figsize = (10,5))
plt.plot(splits,acu1)
plt.xlabel("Min_sample_split")
plt.ylabel("Accuracy")
plt.title("Min_sample_split vs Accuracy")
plt.show()

a = np.asmatrix(splits)
b = np.asmatrix(acu1)
output = np.stack((a,b))
print('  split vs Accuracy')
print(output.transpose())
```

The variation of sample split with accuracy is plotted.



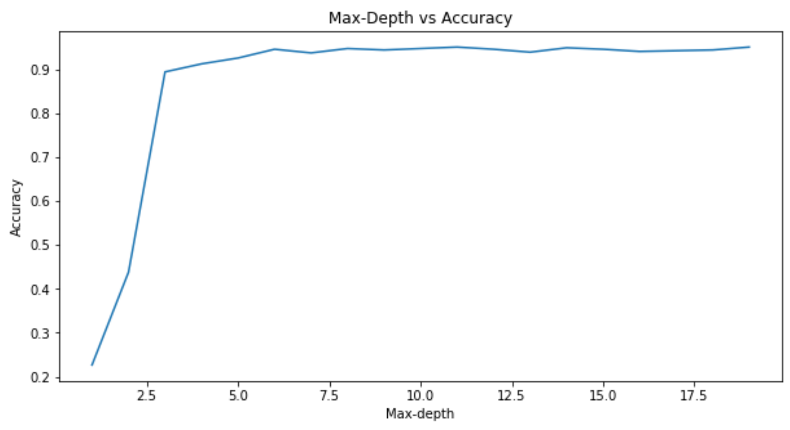
```
Sample_split vs Accuracy
[[ 1.          0.22666667]
 [ 2.          0.43833333]
 [ 3.          0.895     ]
 [ 4.          0.91333333]
 [ 5.          0.93      ]
 [ 6.          0.94333333]
 [ 7.          0.93666667]
 [ 8.          0.95166667]
 [ 9.          0.94833333]
 [10.          0.94833333]
 [11.          0.95      ]
 [12.          0.95166667]
 [13.          0.94666667]
 [14.          0.94833333]
 [15.          0.94166667]
 [16.          0.94333333]
 [17.          0.95166667]
 [18.          0.945     ]
 [19.          0.95166667]]
```

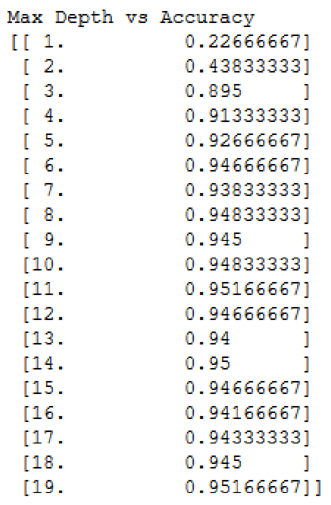
Similarly the variation of max depth with accuracy is calculated by



```
#By changing the max_depth
acu2 = []
dep =[]
for i in range(1,20):
    # Create Decision Tree classifer object
    clf_dt2 = DecisionTreeClassifier(criterion="gini", max_depth=i)

    # Train Decision Tree Classifer
    clf_dt2 = clf_dt2.fit(X_train,y_train)

    #Predict the response for test dataset
    y_pred = clf_dt2.predict(X_test)
    dep.append(i)
    acu2.append(metrics.accuracy_score(y_test, y_pred))
plt.figure(figsize = (10,5))
plt.plot(dep,acu2)
plt.xlabel("Max-depth")
plt.ylabel("Accuracy")
plt.title("Max-Depth vs Accuracy")
plt.show()
a = np.asmatrix(dep)
b = np.asmatrix(acu2)
output = np.stack((a,b))
print('Max Depth vs Accuracy')
print(output.transpose())
```

The variation of max depth with accuracy is plotted,





```
Max Depth vs Accuracy
[[ 1.          0.22666667]
 [ 2.          0.43833333]
 [ 3.          0.895     ]
 [ 4.          0.91333333]
 [ 5.          0.92666667]
 [ 6.          0.94666667]
 [ 7.          0.93833333]
 [ 8.          0.94833333]
 [ 9.          0.945     ]
 [10.          0.94833333]
 [11.          0.95166667]
 [12.          0.94666667]
 [13.          0.94      ]
 [14.          0.95      ]
 [15.          0.94666667]
 [16.          0.94166667]
 [17.          0.94333333]
 [18.          0.945     ]
 [19.          0.95166667]]
```

Now the best model will be the one with max accuracy from both the cases of sample split and depth.

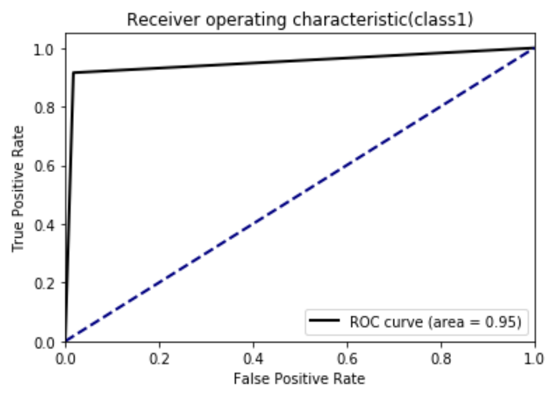
```
if max(acu2)>max(acu1):
    i = acu2.index(max(acu2))
    clf_dt = DecisionTreeClassifier(criterion = 'gini', max_depth = dep[i])
    clf_dt = clf_dt.fit(X_train,y_train)
    #Predicting results using testing data
    y_pred_dt = clf_dt.predict(X_test)
else:
    i = acu1.index(max(acu1))
    clf_dt = DecisionTreeClassifier(criterion = 'gini', min_samples_split = splits[i])
    clf_dt = clf.fit(X_train,y_train)
    #Predicting results using testing data
    y_pred_dt = clf_dt.predict(X_test)

#Creating confusion matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_dt.argmax(axis=1))
#Printing accuracy
print(f"Accuracy: {asc(y_pred_dt, y_test)}")
print(("\nConfusion Matrix:"))
print(cm)
```

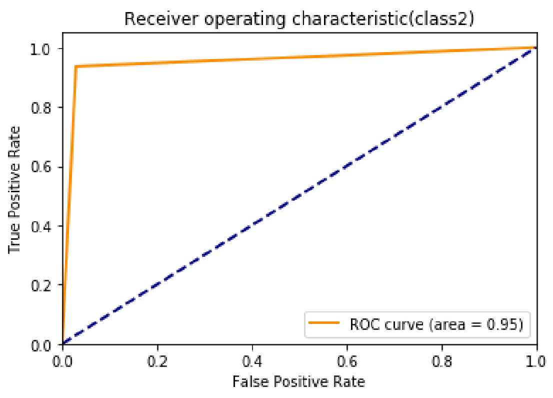
Accuracy and confusion matrix of the obtained model is:

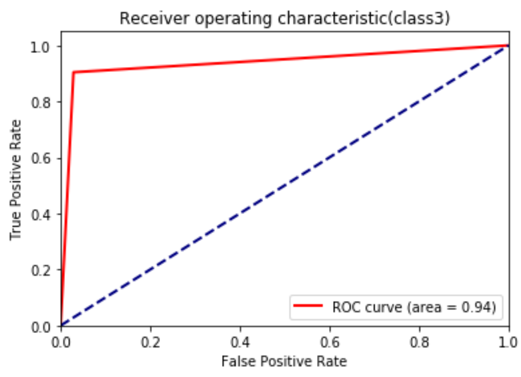
```
Accuracy: 0.95

Confusion Matrix:
[[134   0   1   7]
 [  0 148   9   0]
 [  2   7 133   4]
 [  0   0   0 155]]
```

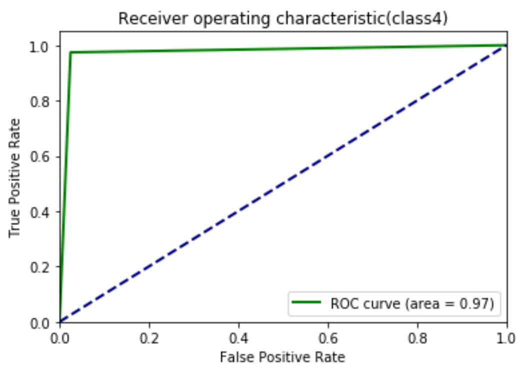
ROC and AUC of Decision Tree model:

```
classifier = OneVsRestClassifier(clf_dt)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```









The ROC curve obtained shows that it a good model and AUC of more than 0.95 confirms this. This is the reason behind a good accuracy score of 0.95 for this model.

Creating a Model with Support Vector Machines:

Here we are implementing SVM with the help of sklearn library.

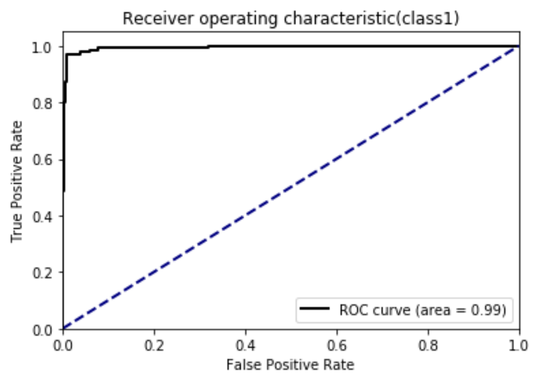
```
from sklearn.svm import SVC
clf_svm = SVC(kernel = 'rbf')
clf_svm.fit(X_train, y_train.argmax(axis=1))

# Predicting the test set result
y_pred_svm = clf_svm.predict(X_test)

# Making the Confusion Matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_svm)

print(f"Accuracy: {asc(y_pred_svm, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

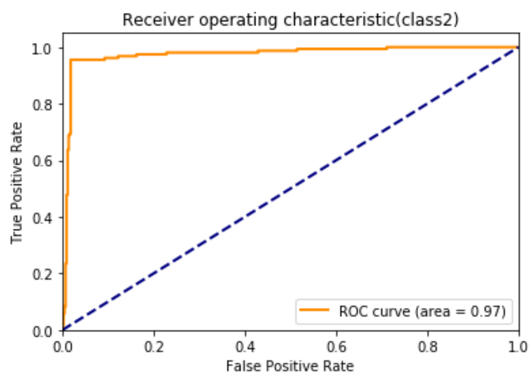
The confusion matrix and accuracy obtained using this model are:

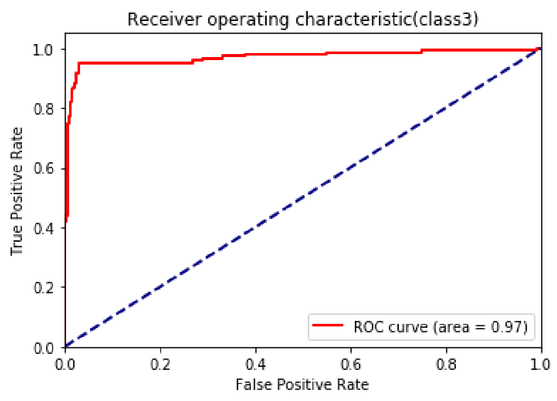
```
Accuracy: 0.7016666666666667

Confusion Matrix:
[[ 86  56   0   0]
 [  0 155   2   0]
 [  0  59  87   0]
 [  1  61   0  93]]
```

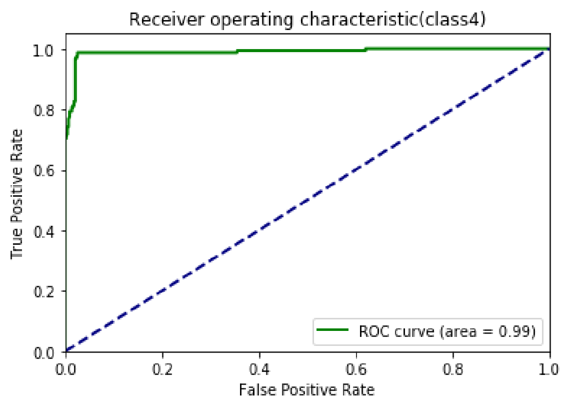
ROC and AUC for SVM Model:

```
classifier = OneVsRestClassifier(SVC(kernel='rbf',probability = True))
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```









Creating a Model with Linear Discriminant Analysis:

Here we are implementing LDA with the help of sklearn library.

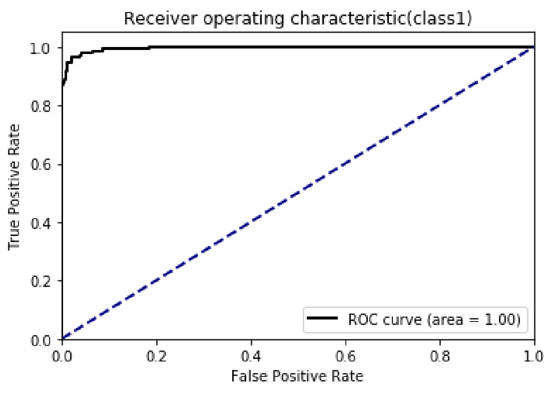
```
# Create model
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
clf_lda = LinearDiscriminantAnalysis()
clf_lda.fit(X_train, y_train.argmax(axis=1))

# Predicting the test set result
y_pred_lda = clf_lda.predict(X_test)
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_lda)

print(f"Accuracy: {asc(y_pred_lda, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

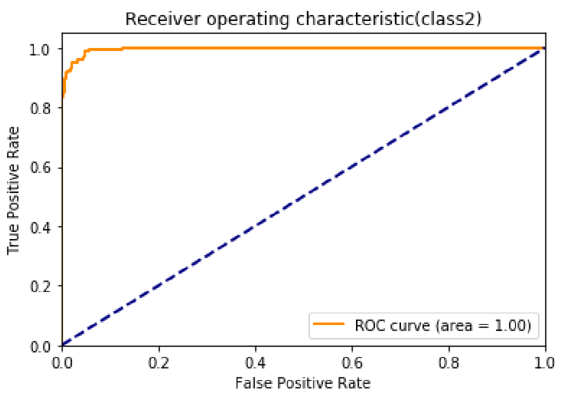
The confusion matrix and accuracy obtained using this model are:

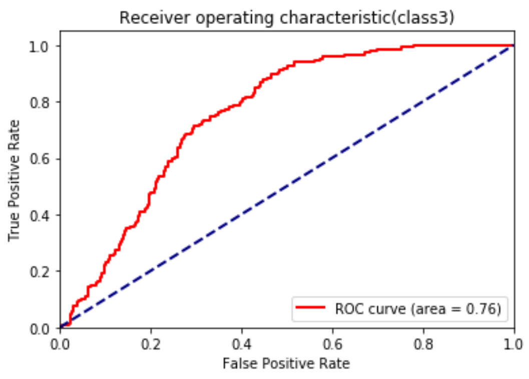
```
Accuracy: 0.9333333333333333

Confusion Matrix:
[[137   0   4   1]
 [  1 136  20   0]
 [  1   1 141   3]
 [  2   0   7 146]]
```

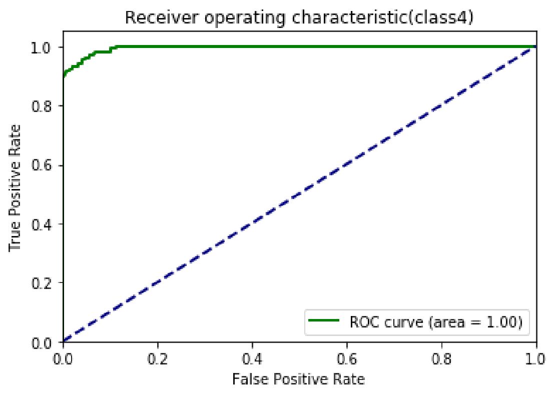
ROC and AUC for LDA Model:

```
classifier = OneVsRestClassifier(clf_lda)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```









Creating a Model with Quadratic Discriminant Analysis:

Here we are implementing QDA with the help of sklearn library.

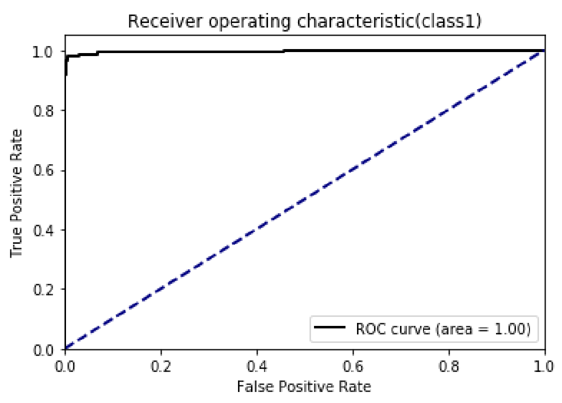
```
# Create model
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
clf_qda = QuadraticDiscriminantAnalysis()
clf_qda.fit(X_train, y_train.argmax(axis=1))

# Predicting the test set result
y_pred_qda = clf_qda.predict(X_test)

# Making the Confusion Matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_qda)

print(f"Accuracy: {asc(y_pred_qda, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

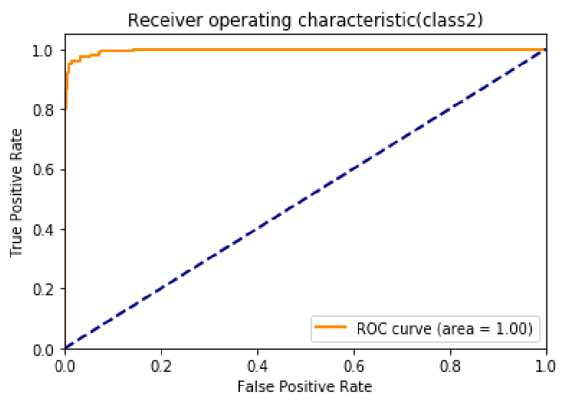
The confusion matrix and accuracy obtained using this model are:

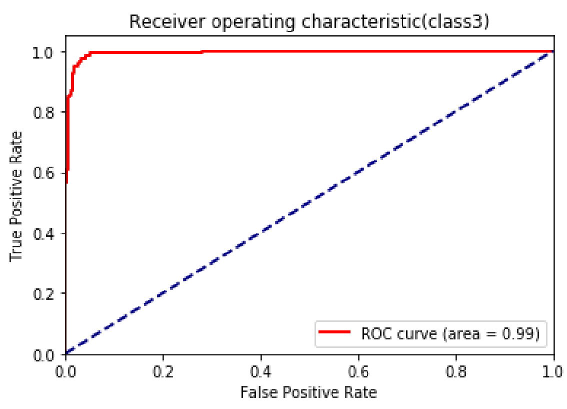
```
Accuracy: 0.9683333333333334

Confusion Matrix:
[[140   0   1   1]
 [  0 151   6   0]
 [  3   2 136   5]
 [  0   0   1 154]]
```

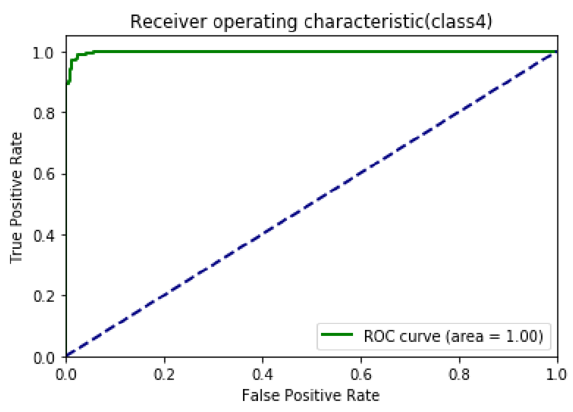
ROC and AUC for QDA Model:

```
classifier = OneVsRestClassifier(clf_lda)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```



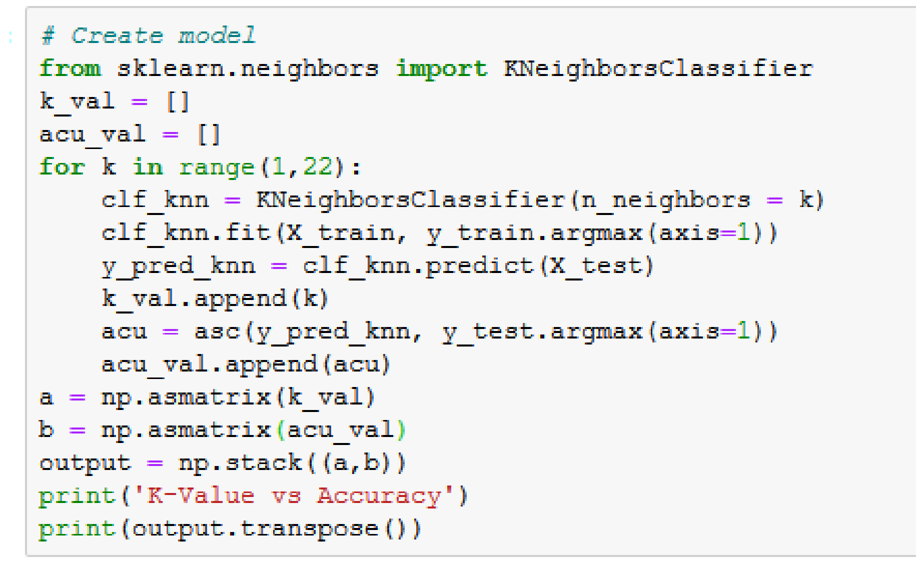




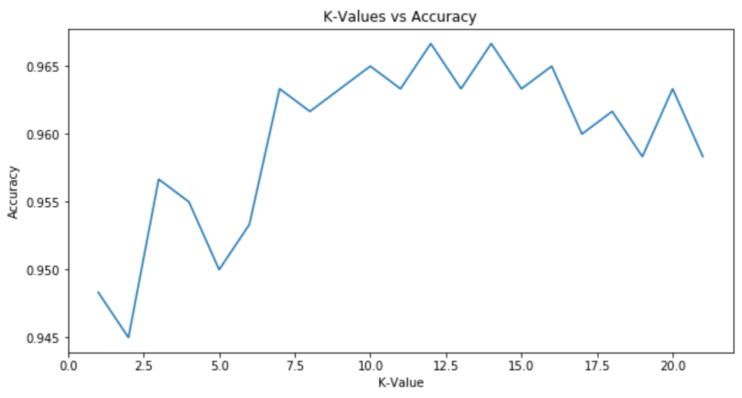


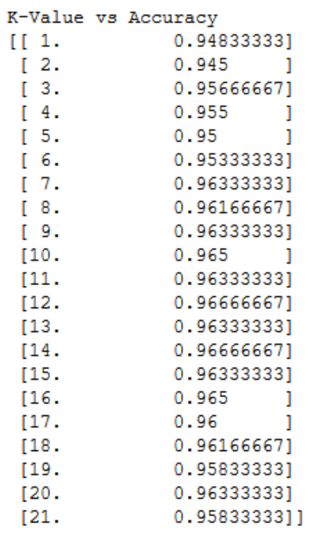
Creating a Model with K-Nearest Neighbours:

Here we are implementing KNN with the help of sklearn library. Inorder to find the optimum K value, we have done hyper-parametric tuning for the value of K in the range of 1 to 22.



```
# Create model
from sklearn.neighbors import KNeighborsClassifier
k_val = []
acu_val = []
for k in range(1,22):
    clf_knn = KNeighborsClassifier(n_neighbors = k)
    clf_knn.fit(X_train, y_train.argmax(axis=1))
    y_pred_knn = clf_knn.predict(X_test)
    k_val.append(k)
    acu = asc(y_pred_knn, y_test.argmax(axis=1))
    acu_val.append(acu)
a = np.asmatrix(k_val)
b = np.asmatrix(acu_val)
output = np.stack((a,b))
print('K-Value vs Accuracy')
print(output.transpose())
```





```
K-Value vs Accuracy
[[ 1.          0.94833333]
 [ 2.          0.945     ]
 [ 3.          0.95666667]
 [ 4.          0.955     ]
 [ 5.          0.95      ]
 [ 6.          0.95333333]
 [ 7.          0.96333333]
 [ 8.          0.96166667]
 [ 9.          0.96333333]
 [10.          0.965     ]
 [11.          0.96333333]
 [12.          0.96666667]
 [13.          0.96333333]
 [14.          0.96666667]
 [15.          0.96333333]
 [16.          0.965     ]
 [17.          0.96      ]
 [18.          0.96166667]
 [19.          0.95833333]
 [20.          0.96333333]
 [21.          0.95833333]]
```

The variation of K value with accuracy is plotted to find the optimum value of K for our model.

Now the best model will be the one with maximum accuracy for a given K value is found out.

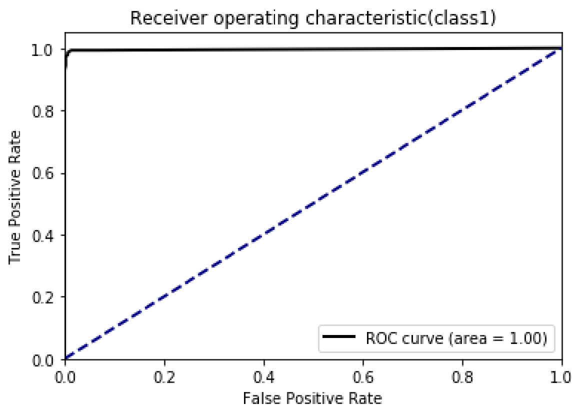
```
i = acu_val.index(max(acu_val))
clf_knn = KNeighborsClassifier(n_neighbors = k_val[i])
clf_knn.fit(X_train, y_train.argmax(axis=1))
y_pred_knn = clf_knn.predict(X_test)

# Making the Confusion Matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_knn)
print(f"Accuracy: {asc(y_pred_knn, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

Accuracy and confusion matrix of the obtained model is as follows

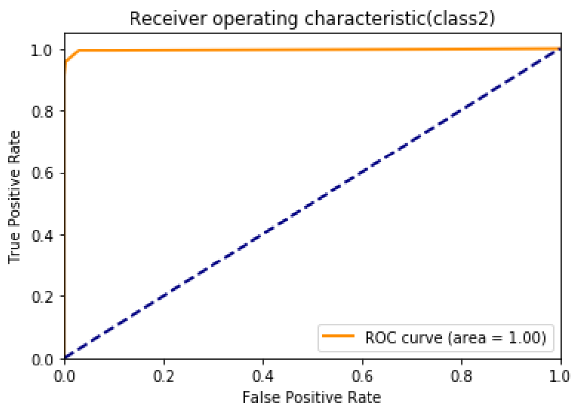
```
Accuracy: 0.9666666666666667

Confusion Matrix:
[[139   0   1   2]
 [  0 150   7   0]
 [  1   1 141   3]
 [  2   0   3 150]]
```

ROC and AUC for KNN Model:

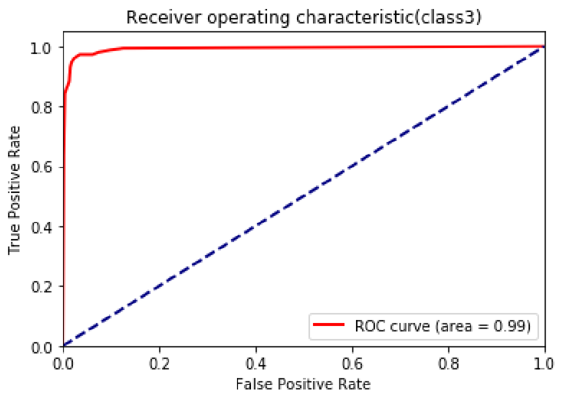
```
classifier = OneVsRestClassifier(clf_knn)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```



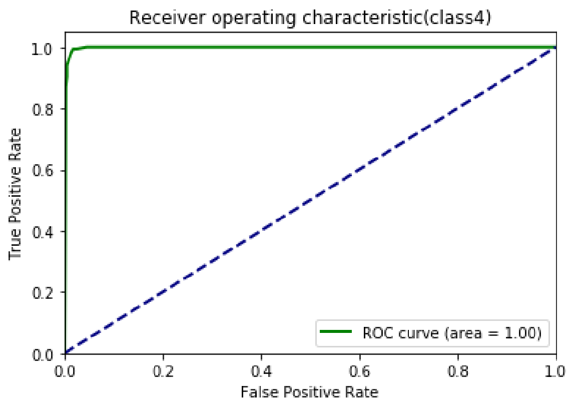
Receiver operating characteristic(class1)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 1.00)



Receiver operating characteristic(class2)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 1.00)



Receiver operating characteristic(class3)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.99)



Receiver operating characteristic(class4)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 1.00)

Tabulating the Results:

The results from all the four models created after imputation using mean method is shown below.

Model	**Accuracy Score**	**Confusion Matrix**

Decision Tree	0.950	`Confusion Matrix:`
		`[[134   0   1   7]`
		`[  0 148   9   0]`
		`[  2   7 133   4]`
		`[  0   0   0 155]]`

SVM	0.701	`Confusion Matrix:`
		`[[ 86  56   0   0]`
		`[  0 155   2   0]`
		`[  0  59  87   0]`
		`[  1  61   0  93]]`

LDA	0.933	`Confusion Matrix:`
		`[[137   0   4   1]`
		`[  1 136  20   0]`
		`[  1   1 141   3]`
		`[  2   0   7 146]]`

QDA	0.968	`Confusion Matrix:`
		`[[140   0   1   1]`
		`[  0 151   6   0]`
		`[  3   2 136   5]`
		`[  0   0   1 154]]`

KNN	0.966	`Confusion Matrix:`
		`[[139   0   1   2]`
		`[  0 150   7   0]`
		`[  1   1 141   3]`
		`[  2   0   3 150]]`

From the above table we can see that QDA model produced the maximum accuracy.

Performing K-fold cross validation on model with best accuracy:

```
# The best model is QDA
a = 0
for i in range(2,500):
    scores_res = model_selection.cross_val_score(clf_qda, X, y.argmax(axis=1), cv=i)
    if a < scores_res.mean():
        a = scores_res.mean()
        j = i
print(f'Max Accuracy = {a:.3}\nNo. of Folds = {j}')
    #print(scores_res.mean())
```

The max accuracy and the number of folds at which we obtain the max accuracy is as follows:

```
Max Accuracy = 0.969
No. of Folds = 340
```

Imputing using K Nearest Neighbours and predicting the model:

Here we are imputing the missing data using K Nearest Neighbours imputation technique.

```
from sklearn.neighbors import KNeighborsRegressor
Xt = data.drop(["Room"],axis=1)
imp = IterativeImputer(estimator = KNeighborsRegressor(n_neighbors=15),max_iter=20, random_state=0)
imp.fit(Xt)
```

```
features = pd.DataFrame(imp.transform(Xt), columns = ['WiFi1','WiFi2','WiFi3','WiFi4','WiFi5','WiFi6','WiFi7'])
df = features
df["Room"] = data.Room
y = data.Room
df
```

	WiFi1	WiFi2	WiFi3	WiFi4	WiFi5	WiFi6	WiFi7	Room
0	-64.000000	-56.000000	-61.000000	-66.000000	-71.000000	-82.000000	-81.000000	1
1	-62.266667	-57.000000	-61.000000	-65.000000	-71.000000	-85.000000	-85.000000	1
2	-63.000000	-60.000000	-62.933333	-67.000000	-76.000000	-85.000000	-84.000000	1
3	-65.066667	-60.000000	-68.000000	-62.000000	-77.000000	-90.000000	-80.000000	1
4	-64.800000	-65.000000	-60.000000	-63.000000	-77.000000	-81.000000	-87.000000	1
5	-64.000000	-55.000000	-63.000000	-66.000000	-76.000000	-83.466667	-83.000000	1

Table above shows first 5 rows for reference

All the missing values have been imputed using KNeighboursRegressor from sklearn library choosing the number of neighbors as 15.

Converting the Output values into Binary:

This is done for using ‘OneVsRestClassfier’ library to print ROC and AUC curve later on.

```
X=df.iloc[:,:-1].values
y=df.iloc[:,7].values
y = label_binarize(y, classes=[1,2,3,4])
```

Splitting the Data Set into Train and Test data:

The overall dataset has been split into 70% training set and 30% testing test for determining the accuracy and obtaining the confusion matrix.

```
# Splitting into test and train data
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=1)
```

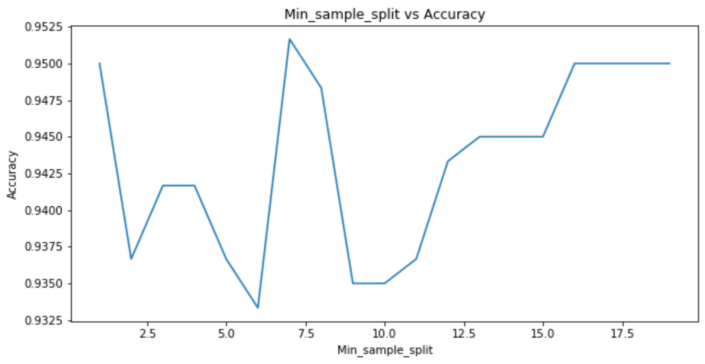
Creating a Model with Decision Trees:

Here we are implementing Decision Tree with the help of sklearn library. In order to obtain the optimum sample split and depth of the Decision Tree model, we have done hyper-parametric tuning. The optimum values are then chosen from the obtained results.

To find the variation of accuracy with sample split we use:

```
# By changing the min_sample_split
acu1 = []
splits =[]
for i in range(1,20):
    # Create Decision Tree classifer object
    clf_dt1 = DecisionTreeClassifier(criterion="gini", min_samples_split=i)

    # Train Decision Tree Classifer
    clf_dt1 = clf_dt1.fit(X_train,y_train)

    #Predict the response for test dataset
    y_pred = clf_dt1.predict(X_test)
    splits.append(i)
    acu1.append(metrics.accuracy_score(y_test, y_pred))
plt.figure(figsize = (10,5))
plt.plot(splits,acu1)
plt.xlabel("Min_sample_split")
plt.ylabel("Accuracy")
plt.title("Min_sample_split vs Accuracy")
plt.show()
a = np.asmatrix(splits)
b = np.asmatrix(acu1)
output = np.stack((a,b))
print('no. of splits vs Accuracy')
print(output.transpose())
```

The variation of sample split with accuracy is as follows :

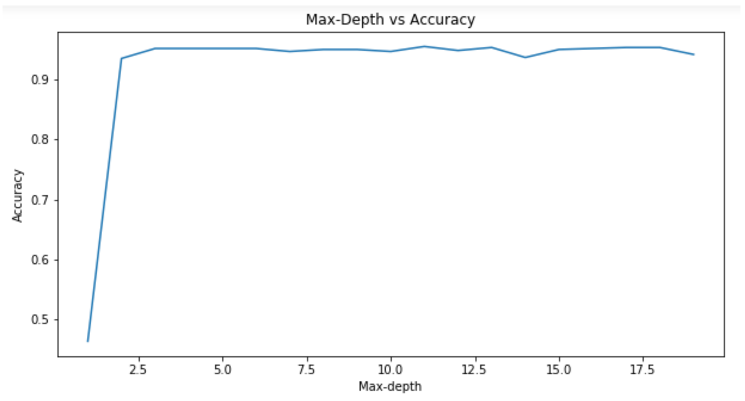


```
no. of splits vs Accuracy
[[ 1.          0.95      ]
 [ 2.          0.93666667]
 [ 3.          0.94166667]
 [ 4.          0.94166667]
 [ 5.          0.93666667]
 [ 6.          0.93333333]
 [ 7.          0.95166667]
 [ 8.          0.94833333]
 [ 9.          0.935     ]
 [10.          0.935     ]
 [11.          0.93666667]
 [12.          0.94333333]
 [13.          0.945     ]
 [14.          0.945     ]
 [15.          0.945     ]
 [16.          0.95      ]
 [17.          0.95      ]
 [18.          0.95      ]
 [19.          0.95      ]]
```

Similarly the variation of max depth with accuracy is calculated by

```python
#By changing the max_depth
acu2 = []
dep =[]
for i in range(1,20):
    # Create Decision Tree classifer object
    clf_dt2 = DecisionTreeClassifier(criterion="gini", max_depth=i)

    # Train Decision Tree Classifer
    clf_dt2 = clf_dt2.fit(X_train,y_train)

    #Predict the response for test dataset
    y_pred = clf_dt2.predict(X_test)
    dep.append(i)
    acu2.append(metrics.accuracy_score(y_test, y_pred))
plt.figure(figsize = (10,5))
plt.plot(dep,acu2)
plt.xlabel("Max-depth")
plt.ylabel("Accuracy")
plt.title("Max-Depth vs Accuracy")
plt.show()
a = np.asmatrix(dep)
b = np.asmatrix(acu2)
output = np.stack((a,b))
print('Maximum Depth vs Accuracy')
print(output.transpose())
```

The variation of maximum depth with accuracy is plotted:



```
Maximum Depth vs Accuracy
[[ 1.          0.46333333]
 [ 2.          0.935     ]
 [ 3.          0.95166667]
 [ 4.          0.95166667]
 [ 5.          0.95166667]
 [ 6.          0.95166667]
 [ 7.          0.94666667]
 [ 8.          0.95      ]
 [ 9.          0.95      ]
 [10.          0.94666667]
 [11.          0.955     ]
 [12.          0.94833333]
 [13.          0.95333333]
 [14.          0.93666667]
 [15.          0.95      ]
 [16.          0.95166667]
 [17.          0.95333333]
 [18.          0.95333333]
 [19.          0.94166667]]
```

Now the best model will be the one with max accuracy from both the cases of sample split and depth.

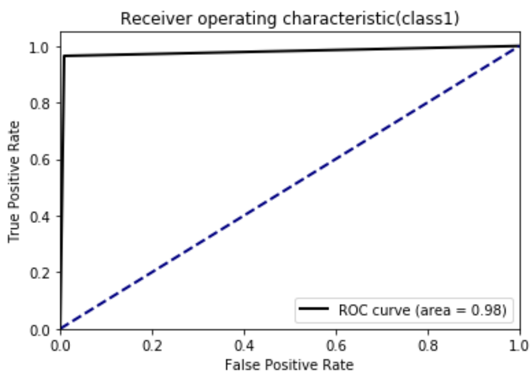
```
if max(acu2)>max(acu1):
    i = acu2.index(max(acu2))
    clf_dt = DecisionTreeClassifier(criterion = 'gini', max_depth = dep[i])
    clf_dt = clf_dt.fit(X_train,y_train)
    #Predicting results using testing data
    y_pred_dt = clf_dt.predict(X_test)
else:
    i = acu1.index(max(acu1))
    clf_dt = DecisionTreeClassifier(criterion = 'gini', min_samples_split = splits[i])
    clf_dt = clf.fit(X_train,y_train)
    #Predicting results using testing data
    y_pred_dt = clf_dt.predict(X_test)

#Creating confusion matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_dt.argmax(axis=1))
#Printing accuracy
print(f"Accuracy: {asc(y_pred_dt, y_test)}")
print(("\nConfusion Matrix:"))
print(cm)
```

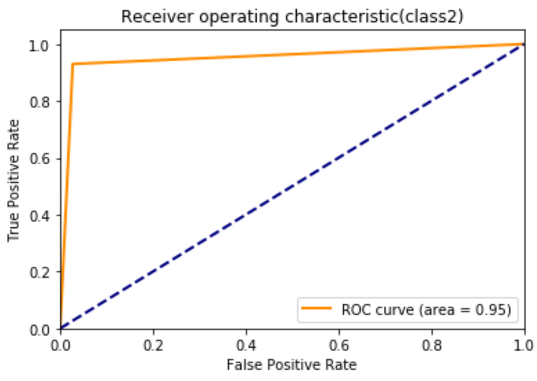
Accuracy and confusion matrix of the obtained model is:

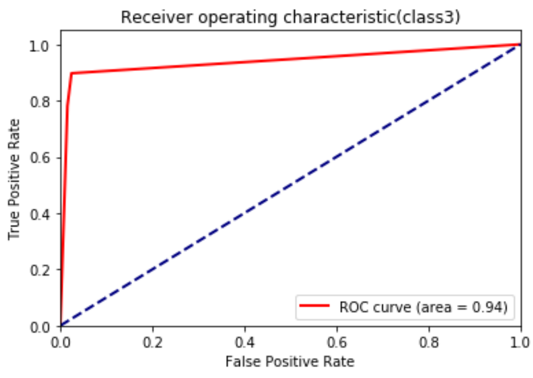
```
Accuracy: 0.9533333333333334

Confusion Matrix:
[[135   0   1   6]
 [  0 149   8   0]
 [  2   7 134   3]
 [  0   0   1 154]]
```

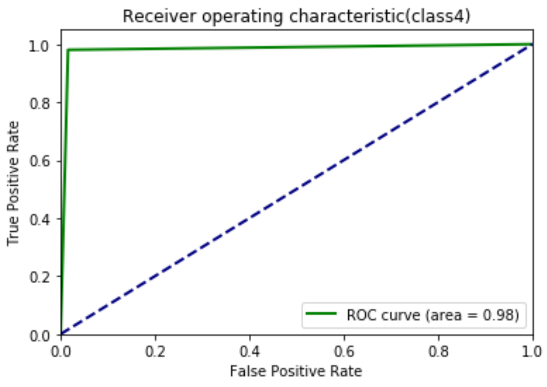
ROC and AUC of Decision Tree model:

```
classifier = OneVsRestClassifier(clf_dt)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```





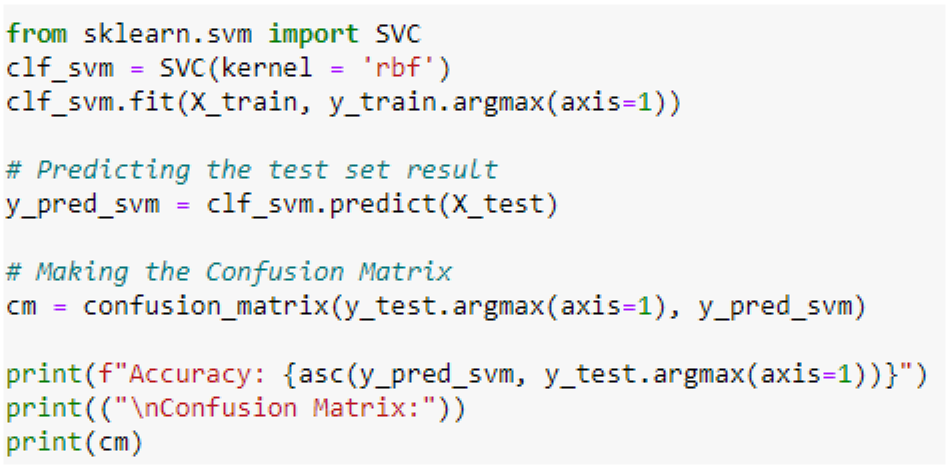




The ROC curve obtained shows that it a good model and AUC of more than 0.9533 confirms this. This is the reason behind a good accuracy score of 0.9533 for this model.

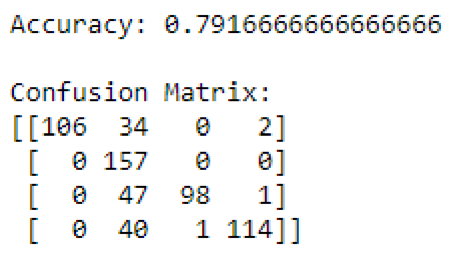
Creating a Model with Support Vector Machines:

Here we are implementing SVM with the help of sklearn library.



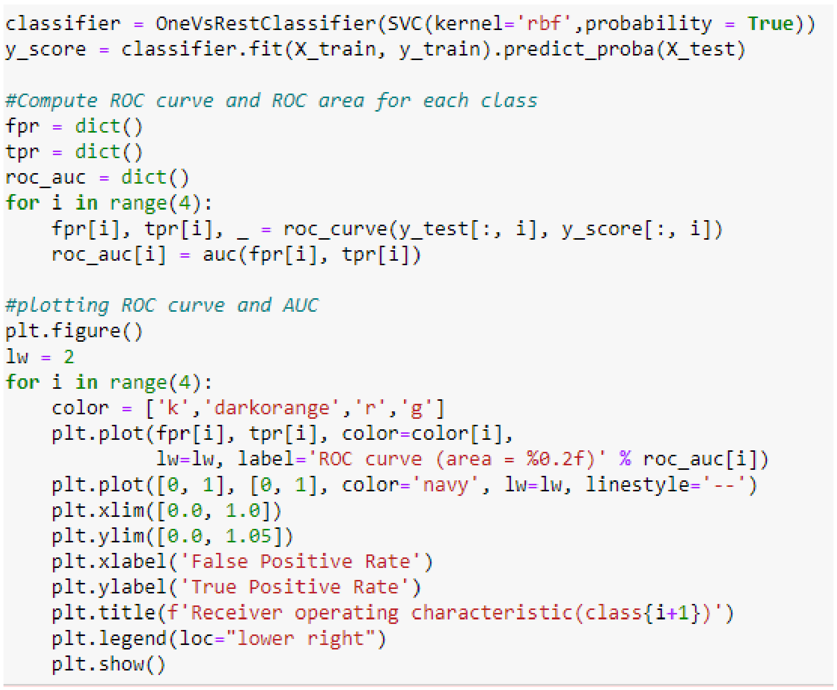
```python
from sklearn.svm import SVC
clf_svm = SVC(kernel = 'rbf')
clf_svm.fit(X_train, y_train.argmax(axis=1))

# Predicting the test set result
y_pred_svm = clf_svm.predict(X_test)

# Making the Confusion Matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_svm)

print(f"Accuracy: {asc(y_pred_svm, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

The confusion matrix and accuracy obtained using this model are:

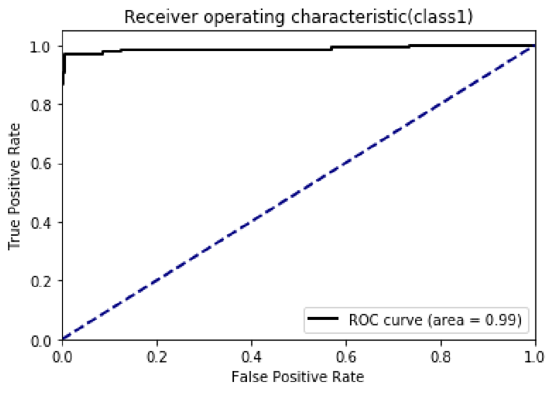


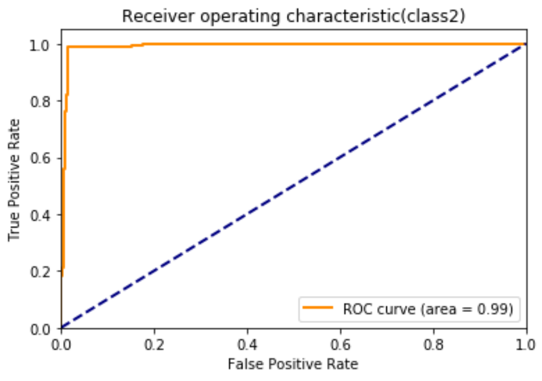
```
Accuracy: 0.7916666666666666

Confusion Matrix:
[[106  34   0   2]
 [  0 157   0   0]
 [  0  47  98   1]
 [  0  40   1 114]]
```

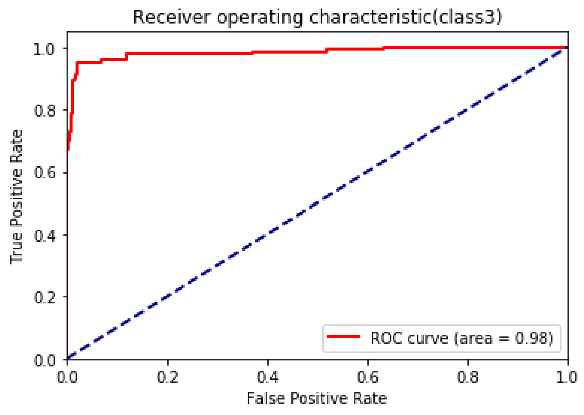
ROC and AUC for SVM Model:

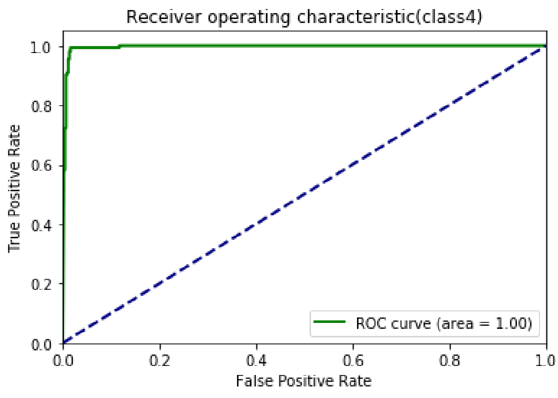


```python
classifier = OneVsRestClassifier(SVC(kernel='rbf',probability = True))
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```


Receiver operating characteristic(class1)
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver operating characteristic(class2)
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver operating characteristic(class3)
True Positive Rate
False Positive Rate
ROC curve (area = 0.98)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver operating characteristic(class4)
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0

Creating a Model with Linear Discriminant Analysis:

Here we are implementing LDA with the help of sklearn library.

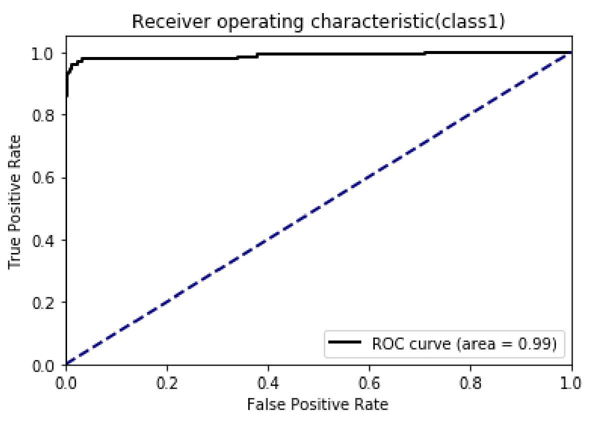
```
# Create model
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
clf_lda = LinearDiscriminantAnalysis()
clf_lda.fit(X_train, y_train.argmax(axis=1))

# Predicting the test set result
y_pred_lda = clf_lda.predict(X_test)
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_lda)

print(f"Accuracy: {asc(y_pred_lda, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

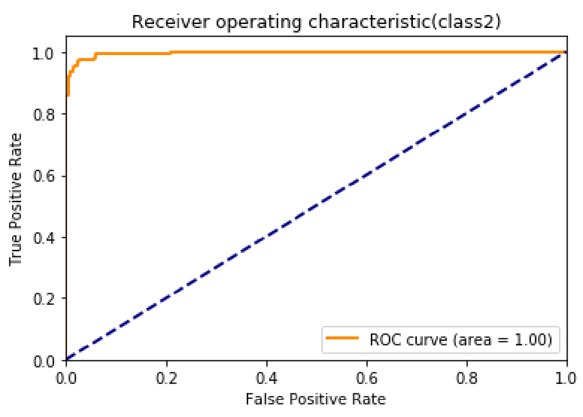
The confusion matrix and accuracy obtained using this model are:

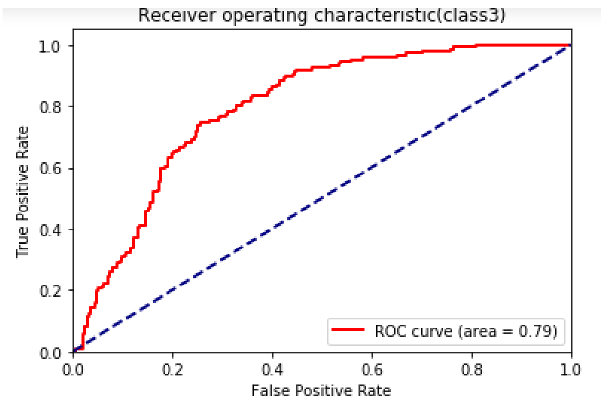
```
Accuracy: 0.96

Confusion Matrix:
[[138   0   2   2]
 [  0 141  16   0]
 [  1   0 143   2]
 [  0   0   1 154]]
```

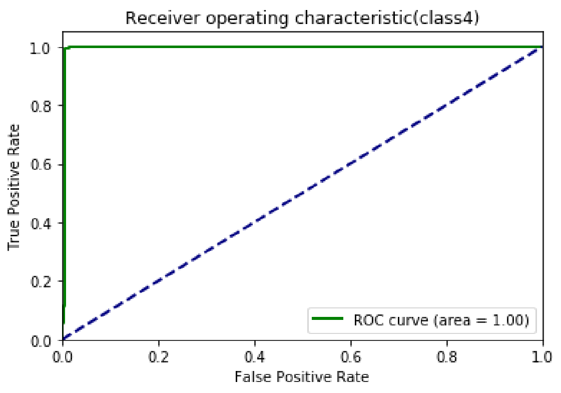
ROC and AUC for LDA Model:

```
classifier = OneVsRestClassifier(clf_lda)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```









Creating a Model with Quadratic Discriminant Analysis:

Here we are implementing QDA with the help of sklearn library.

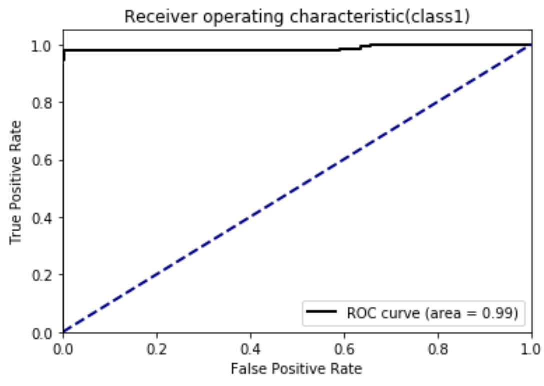
```
# Create model
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
clf_qda = QuadraticDiscriminantAnalysis()
clf_qda.fit(X_train, y_train.argmax(axis=1))

# Predicting the test set result
y_pred_qda = clf_qda.predict(X_test)

# Making the Confusion Matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_qda)

print(f"Accuracy: {asc(y_pred_qda, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

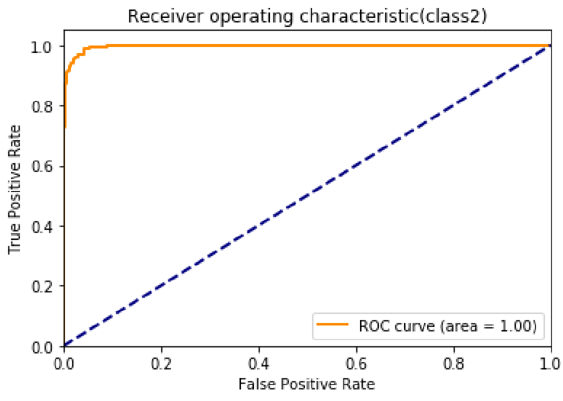
The confusion matrix and accuracy obtained using this model are:

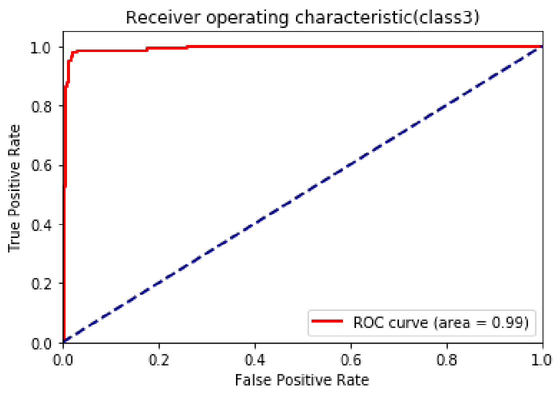
```
Accuracy: 0.9733333333333334

Confusion Matrix:
[[139   0   1   2]
 [  0 152   5   0]
 [  1   2 139   4]
 [  0   0   1 154]]
```

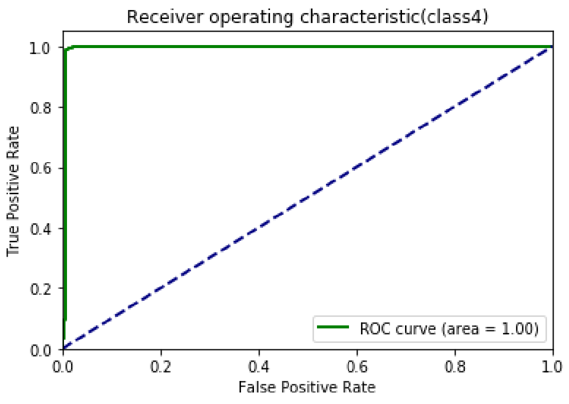
ROC and AUC for QDA Model:

```
classifier = OneVsRestClassifier(clf_qda)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```



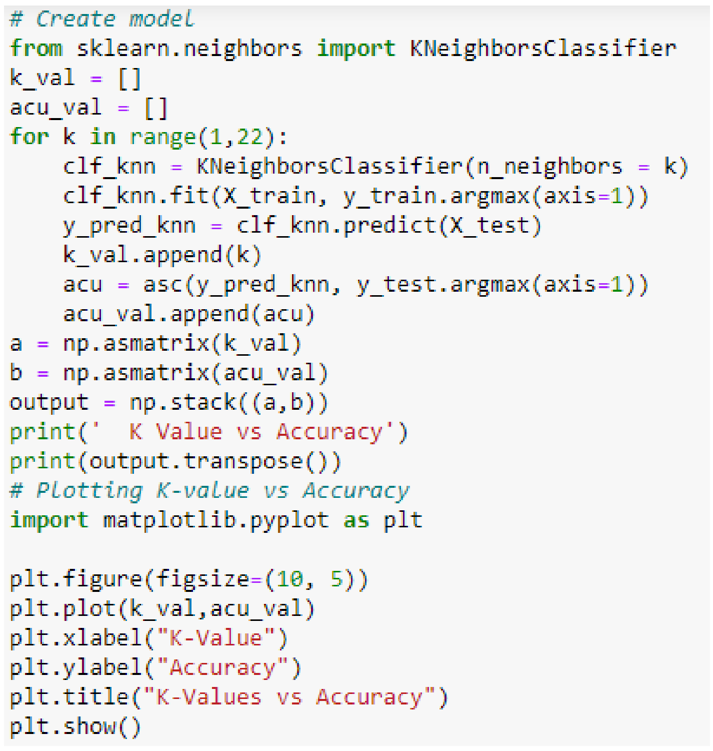






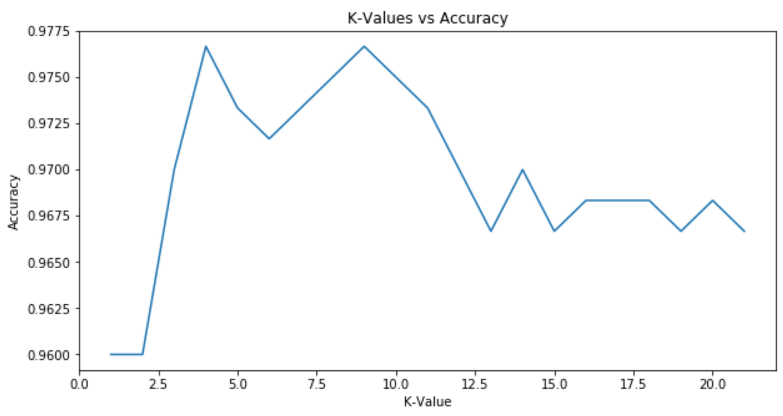
Creating a Model with K-Nearest Neighbours:

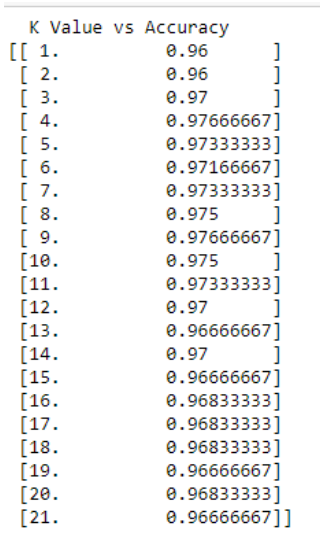
Here we are implementing KNN with the help of sklearn library. Inorder to find the optimum K value, we have done hyper-parametric tuning for the value of K in the range of 1 to 22.



```python
# Create model
from sklearn.neighbors import KNeighborsClassifier
k_val = []
acu_val = []
for k in range(1,22):
    clf_knn = KNeighborsClassifier(n_neighbors = k)
    clf_knn.fit(X_train, y_train.argmax(axis=1))
    y_pred_knn = clf_knn.predict(X_test)
    k_val.append(k)
    acu = asc(y_pred_knn, y_test.argmax(axis=1))
    acu_val.append(acu)
a = np.asmatrix(k_val)
b = np.asmatrix(acu_val)
output = np.stack((a,b))
print('  K Value vs Accuracy')
print(output.transpose())
# Plotting K-value vs Accuracy
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
plt.plot(k_val,acu_val)
plt.xlabel("K-Value")
plt.ylabel("Accuracy")
plt.title("K-Values vs Accuracy")
plt.show()
```

The variation of K value with accuracy is plotted to find the optimum value of K for our model.





```
  K Value vs Accuracy
[[ 1.          0.96      ]
 [ 2.          0.96      ]
 [ 3.          0.97      ]
 [ 4.          0.97666667]
 [ 5.          0.97333333]
 [ 6.          0.97166667]
 [ 7.          0.97333333]
 [ 8.          0.975     ]
 [ 9.          0.97666667]
 [10.          0.975     ]
 [11.          0.97333333]
 [12.          0.97      ]
 [13.          0.96666667]
 [14.          0.97      ]
 [15.          0.96666667]
 [16.          0.96833333]
 [17.          0.96833333]
 [18.          0.96833333]
 [19.          0.96666667]
 [20.          0.96833333]
 [21.          0.96666667]]
```

Now the best model will be the one with maximum accuracy for a given K value is found out.

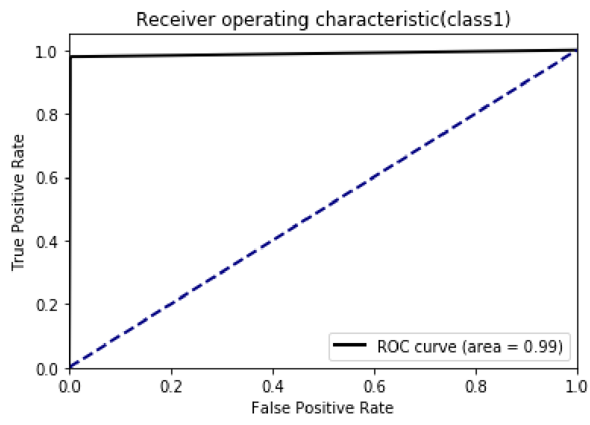
```python
i = acu_val.index(max(acu_val))
clf_knn = KNeighborsClassifier(n_neighbors = k_val[i])
clf_knn.fit(X_train, y_train.argmax(axis=1))
y_pred_knn = clf_knn.predict(X_test)

# Making the Confusion Matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_knn)
print(f"Accuracy: {asc(y_pred_knn, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

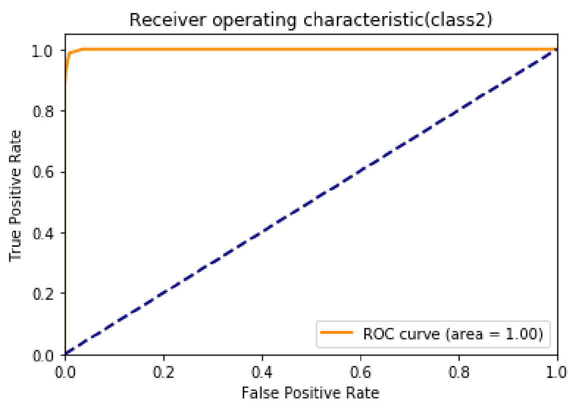
Accuracy and confusion matrix of the obtained model is as follows:

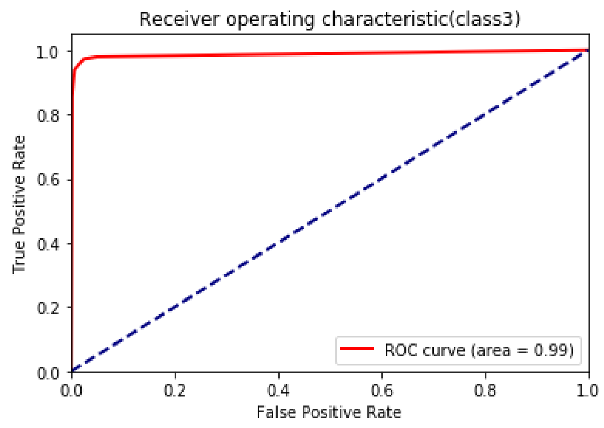
```
Accuracy: 0.9766666666666667

Confusion Matrix:
[[139   0   1   2]
 [  0 155   2   0]
 [  2   4 138   2]
 [  0   0   1 154]]
```

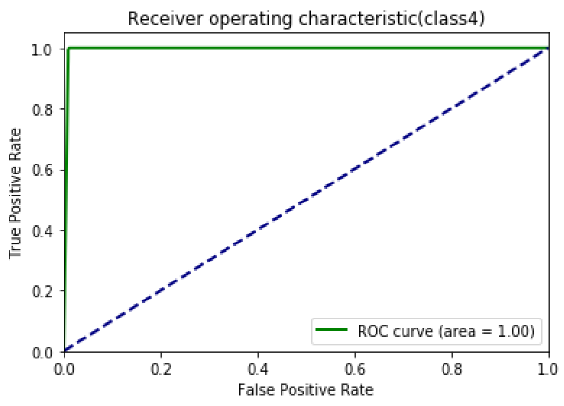
ROC and AUC for KNN Model:

```
classifier = OneVsRestClassifier(clf_knn)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```



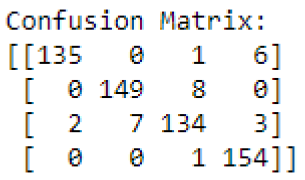
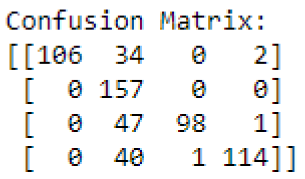
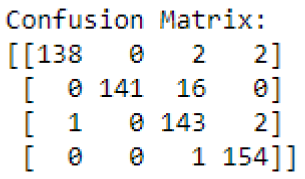
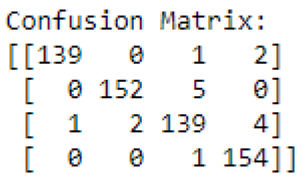
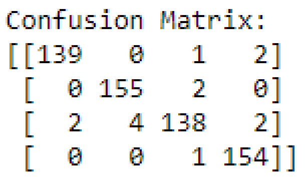






Tabulating the Results:

The results from all the four models created after imputation using mean method is shown below.

Model	**Accuracy Score**	**Confusion Matrix**
Decision Tree	0.953	Confusion Matrix: [[135 0 1 6] [0 149 8 0] [2 7 134 3] [0 0 1 154]]
SVM	0.791	Confusion Matrix: [[106 34 0 2] [0 157 0 0] [0 47 98 1] [0 40 1 114]]
LDA	0.960	Confusion Matrix: [[138 0 2 2] [0 141 16 0] [1 0 143 2] [0 0 1 154]]
QDA	0.973	Confusion Matrix: [[139 0 1 2] [0 152 5 0] [1 2 139 4] [0 0 1 154]]
KNN	0.976	Confusion Matrix: [[139 0 1 2] [0 155 2 0] [2 4 138 2] [0 0 1 154]]

From the above table we can see that KNN model produced the maximum accuracy.

Performing K fold cross validation on the model with maximum accuracy:

By performing K fold cross validation we are trying to increase the accuracy of the best fitted model.

```
a=0
for i in range(2,500):
    scores_res = model_selection.cross_val_score(clf_knn, X, y.argmax(axis=1), cv=i)
    if a < scores_res.mean():
        a = scores_res.mean()
        j=i
print(f'Max accuracy = {a}, coooresponding to K value of = {j}')
```

The results are as follows:

```
Max accuracy = 0.9779624277456648, coooresponding to K value of = 346
```

Hence, we can see that the final model using KNN regression as an imputation technique and KNN classification as the modelling technique, we are getting a maximum accuracy of 0.977 corresponding to K value of 346.

Imputing using Decision Tree and Predicting the Model:

The missing values are imputed using decision tree library present in the iterative imputer library of python of the data points in the column.

```
Xt = data.drop(["Room"],axis=1)
imp = IterativeImputer(estimator =  DecisionTreeRegressor(max_features='sqrt', random_state=0),max_iter=100, random_state=0)
imp.fit(Xt)
```

	WiFi1	WiFi2	WiFi3	WiFi4	WiFi5	WiFi6	WiFi7	Room
0	-64.0	-56.0	-61.0	-66.0	-71.0	-82.0	-81.0	1
1	-64.0	-57.0	-61.0	-65.0	-71.0	-85.0	-85.0	1
2	-63.0	-60.0	-64.0	-67.0	-76.0	-85.0	-84.0	1
3	-65.0	-60.0	-68.0	-62.0	-77.0	-90.0	-80.0	1
4	-60.0	-65.0	-60.0	-63.0	-77.0	-81.0	-87.0	1

Table above shows first 5 rows for reference

All the missing values have been imputed using IterativeImputer from sklearn library with estimator as Decision Tree.

Converting the Output values into Binary:

This is done for using 'OneVsRestClassfier' library to print ROC and AUC curve later on.

```
X=df.iloc[:,:-1].values
y=df.iloc[:,7].values
y = label_binarize(y, classes=[1,2,3,4])
```

Splitting the Data Set into Train and Test data:

The overall dataset has been split into 70% training set and 30% testing test for determining the accuracy and obtaining the confusion matrix.

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=1)
```

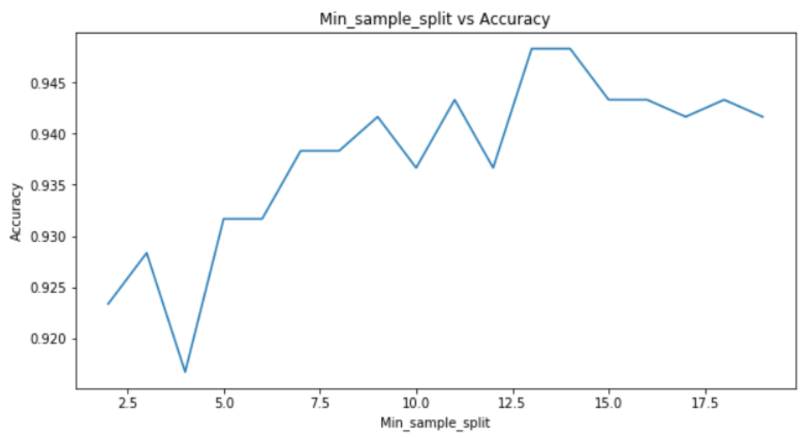
Creating model using Decision Tree:

Here we are implementing Decision Tree with the help of sklearn library. In order to obtain the optimum sample split and depth of the Decision Tree model, we have done hyper-parametric tuning. The optimum values are then chosen from the obtained results.

To find the variation of accuracy with sample split we use

```
# By changing the min_sample_split
acu1 = []
splits =[]
for i in range(2,20):
    # Create Decision Tree classifer object
    clf_dt1 = DecisionTreeClassifier(criterion="gini", min_samples_split=i)

    # Train Decision Tree Classifer
    clf_dt1 = clf_dt1.fit(X_train,y_train)

    #Predict the response for test dataset
    y_pred = clf_dt1.predict(X_test)
    splits.append(i)
    acu1.append(metrics.accuracy_score(y_test, y_pred))
plt.figure(figsize = (10,5))
plt.plot(splits,acu1)
plt.xlabel("Min_sample_split")
plt.ylabel("Accuracy")
plt.title("Min_sample_split vs Accuracy")
plt.show()

a = np.asmatrix(splits)
b = np.asmatrix(acu1)
output = np.stack((a,b))
print('  split vs Accuracy')
print(output.transpose())
```

The variation of sample split with accuracy is plotted.

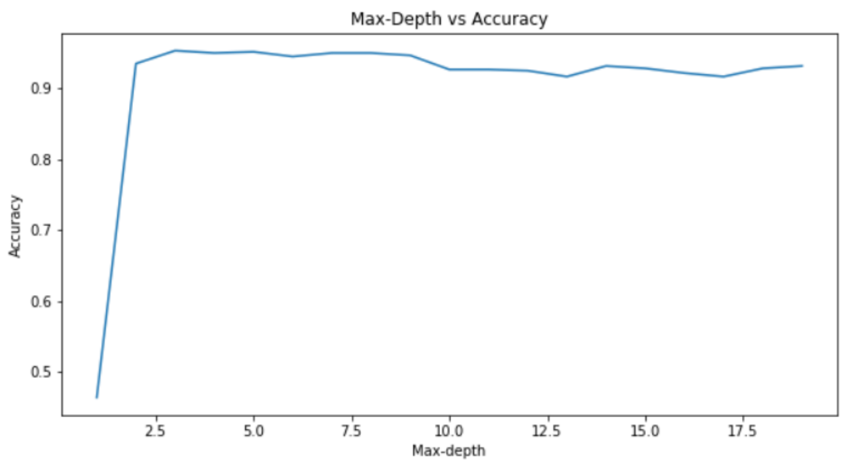


```
 split vs Accuracy
[[ 2.          0.92333333]
 [ 3.          0.92833333]
 [ 4.          0.91666667]
 [ 5.          0.93166667]
 [ 6.          0.93166667]
 [ 7.          0.93833333]
 [ 8.          0.93833333]
 [ 9.          0.94166667]
 [10.          0.93666667]
 [11.          0.94333333]
 [12.          0.93666667]
 [13.          0.94833333]
 [14.          0.94833333]
 [15.          0.94333333]
 [16.          0.94333333]
 [17.          0.94166667]
 [18.          0.94333333]
 [19.          0.94166667]]
```

Similarly the variation of max depth with accuracy is found by

```python
#By changing the max_depth
acu2 = []
dep =[]
for i in range(1,20):
    # Create Decision Tree classifer object
    clf_dt2 = DecisionTreeClassifier(criterion="gini", max_depth=i)

    # Train Decision Tree Classifer
    clf_dt2 = clf_dt2.fit(X_train,y_train)

    #Predict the response for test dataset
    y_pred = clf_dt2.predict(X_test)
    dep.append(i)
    acu2.append(metrics.accuracy_score(y_test, y_pred))
plt.figure(figsize = (10,5))
plt.plot(dep,acu2)
plt.xlabel("Max-depth")
plt.ylabel("Accuracy")
plt.title("Max-Depth vs Accuracy")
plt.show()

a = np.asmatrix(dep)
b = np.asmatrix(acu2)
output = np.stack((a,b))
print('  Depth vs Accuracy')
print(output.transpose())
```

The variation of max depth with accuracy is plotted.



```
 Depth vs Accuracy
[[ 1.          0.46333333]
 [ 2.          0.935     ]
 [ 3.          0.95333333]
 [ 4.          0.95      ]
 [ 5.          0.95166667]
 [ 6.          0.945     ]
 [ 7.          0.95      ]
 [ 8.          0.95      ]
 [ 9.          0.94666667]
 [10.          0.92666667]
 [11.          0.92666667]
 [12.          0.925     ]
 [13.          0.91666667]
 [14.          0.93166667]
 [15.          0.92833333]
 [16.          0.92166667]
 [17.          0.91666667]
 [18.          0.92833333]
 [19.          0.93166667]]
```

Now the best model will be the one with max accuracy from both the cases of sample split and depth.

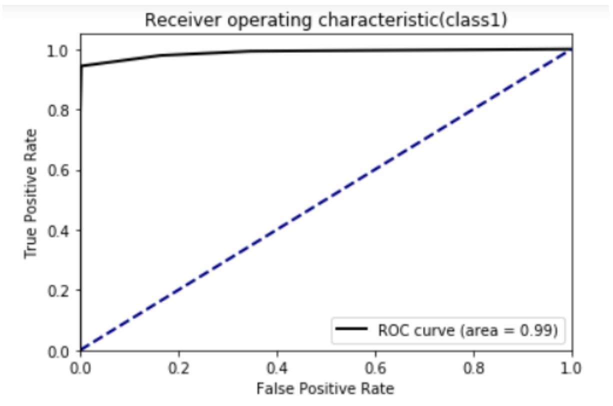
```
if max(acu2)>max(acu1):
    i = acu2.index(max(acu2))
    clf_dt = DecisionTreeClassifier(criterion = 'gini', max_depth = dep[i])
    clf_dt = clf.fit(X_train,y_train)
    #Predicting results using testing data
    y_pred_dt = clf_dt.predict(X_test)
else:
    i = acu1.index(max(acu1))
    clf_dt = DecisionTreeClassifier(criterion = 'gini', min_samples_split = splits[i])
    clf_dt = clf.fit(X_train,y_train)
    #Predicting results using testing data
    y_pred_dt = clf_dt.predict(X_test)

#Creating confusion matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_dt.argmax(axis=1))
#Printing accuracy
print(f"Accuracy: {asc(y_pred_dt, y_test)}")
print(("\nConfusion Matrix:"))
print(cm)
```

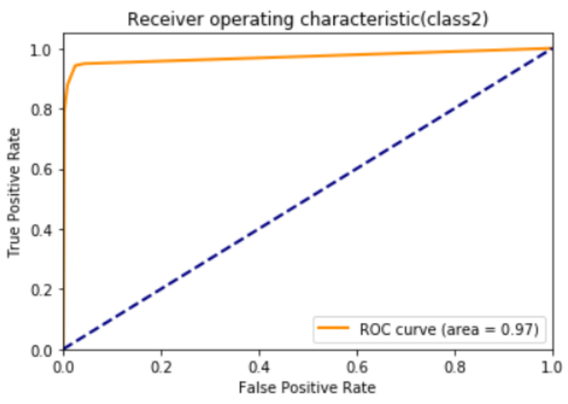
Accuracy and confusion matrix of the obtained model is.

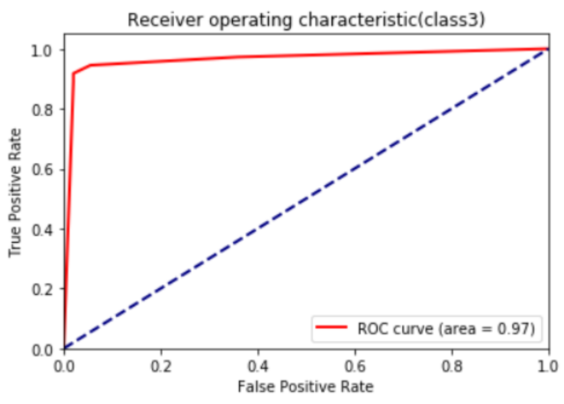
```
Accuracy: 0.9533333333333334

Confusion Matrix:
[[137   0   1   4]
 [  0 150   7   0]
 [  4   7 131   4]
 [  0   0   1 154]]
```

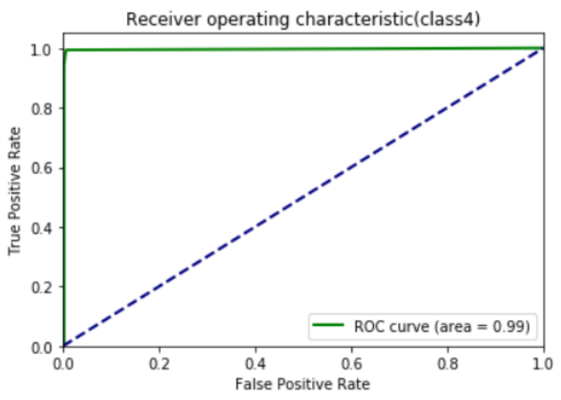
ROC and AUC for Decision Tree model:

```
classifier = OneVsRestClassifier(clf_dt)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```









Creating a Model with Linear Discriminant Analysis:

Here again, we are implementing LDA with the help of sklearn library.

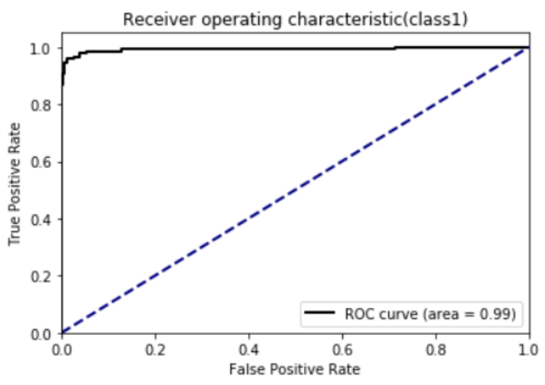
```
# Create model
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
clf_lda = LinearDiscriminantAnalysis()
clf_lda.fit(X_train, y_train.argmax(axis=1))

# Predicting the test set result
y_pred_lda = clf_lda.predict(X_test)
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_lda)

print(f"Accuracy: {asc(y_pred_lda, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

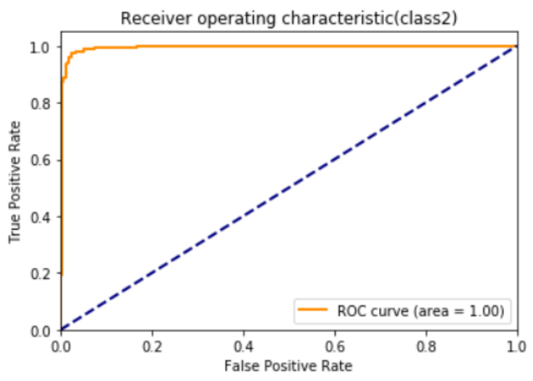
The accuracy and confusion matrix given by the above modelling technique are as follows:

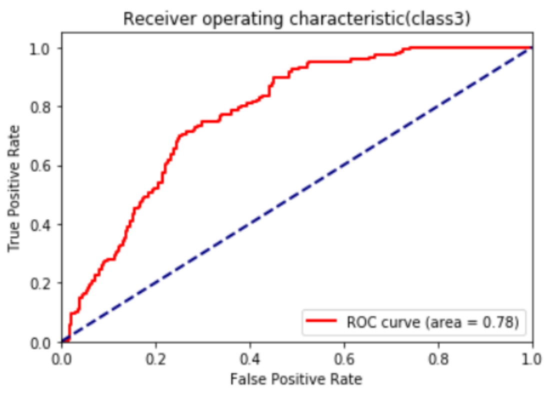
```
Accuracy: 0.9583333333333334

Confusion Matrix:
[[139   0   2   1]
 [  0 141  16   0]
 [  2   1 141   2]
 [  0   0   1 154]]
```

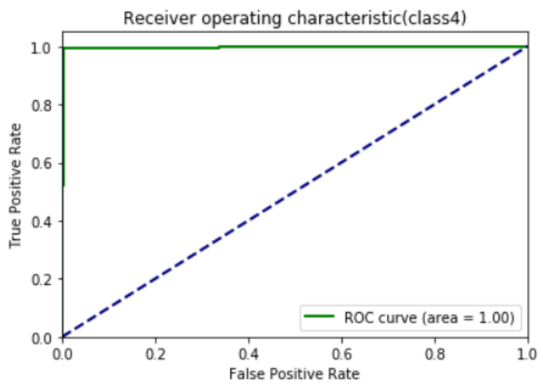
ROC and AUC for LDA model:

```
classifier = OneVsRestClassifier(clf_lda)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```









Creating a Model with Quadratic Discriminant Analysis:

Here we are implementing QDA with the help of sklearn library.

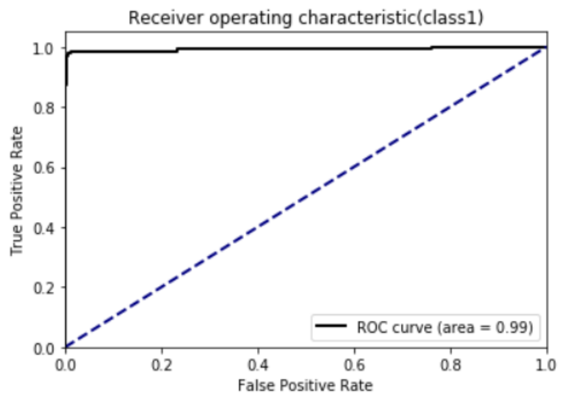
```
# Create model
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
clf_qda = QuadraticDiscriminantAnalysis()
clf_qda.fit(X_train, y_train.argmax(axis=1))

# Predicting the test set result
y_pred_qda = clf_qda.predict(X_test)

# Making the Confusion Matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_qda)

print(f"Accuracy: {asc(y_pred_qda, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

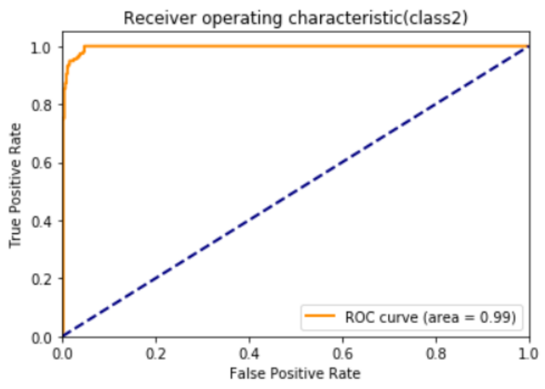
The confusion matrix and accuracy obtained using this model are.

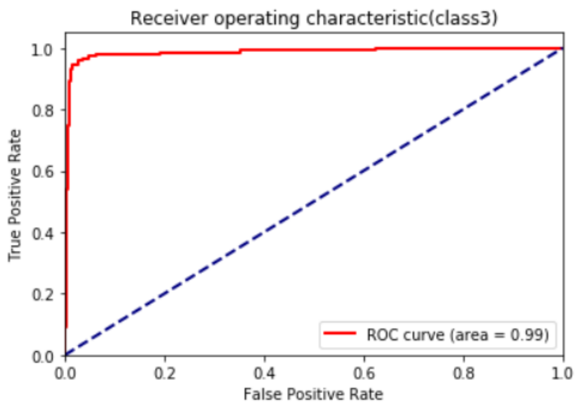
```
Accuracy: 0.9683333333333334

Confusion Matrix:
[[140   0   1   1]
 [  0 153   4   0]
 [  2   5 134   5]
 [  0   0   1 154]]
```

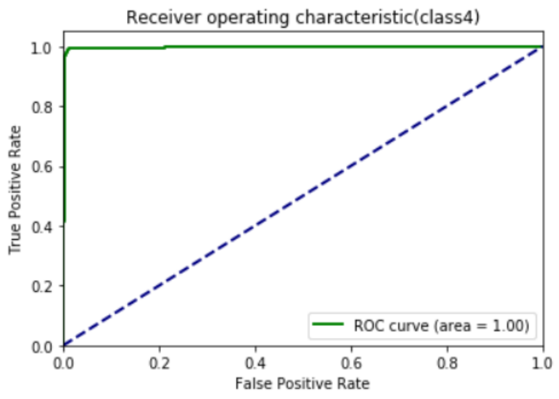
ROC and AUC for QDA model:

```
classifier = OneVsRestClassifier(clf_qda)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```









Creating a Model with Support Vector Machines:

Here we are implementing SVM with the help of sklearn library.

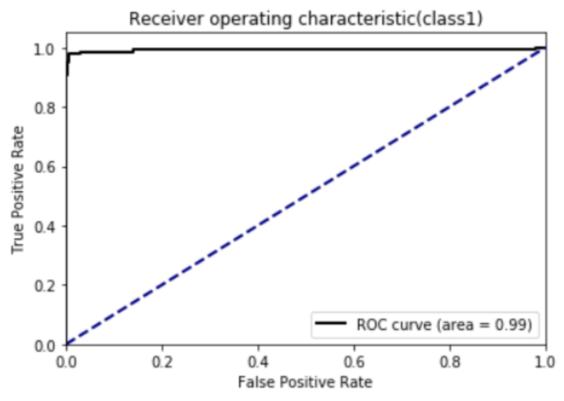
```
from sklearn.svm import SVC
clf_svm = SVC(kernel = 'rbf')
clf_svm.fit(X_train, y_train.argmax(axis=1))

# Predicting the test set result
y_pred_svm = clf_svm.predict(X_test)

# Making the Confusion Matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_svm)

print(f"Accuracy: {asc(y_pred_svm, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

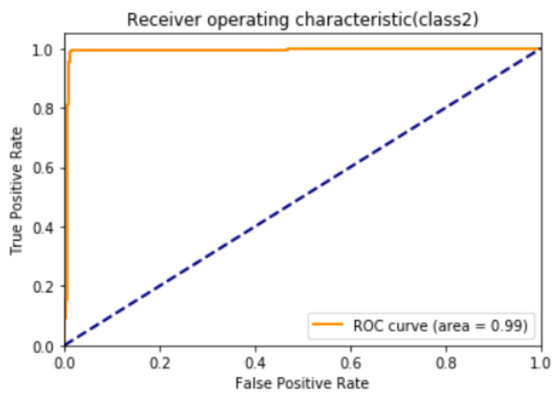
The accuracy and confusion matrix given by the above modelling technique are as follows:

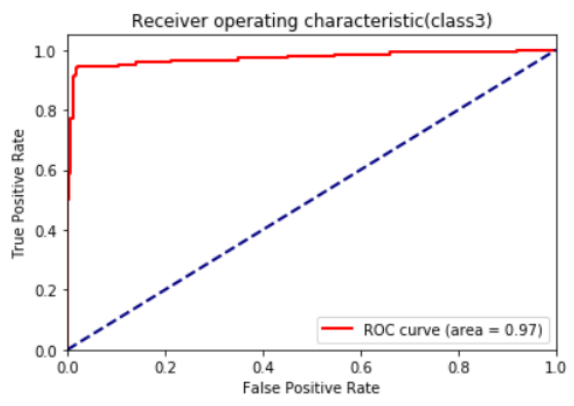
```
Accuracy: 0.77

Confusion Matrix:
[[ 99  42   0   1]
 [  0 156   1   0]
 [  0  53  91   2]
 [  0  39   0 116]]
```

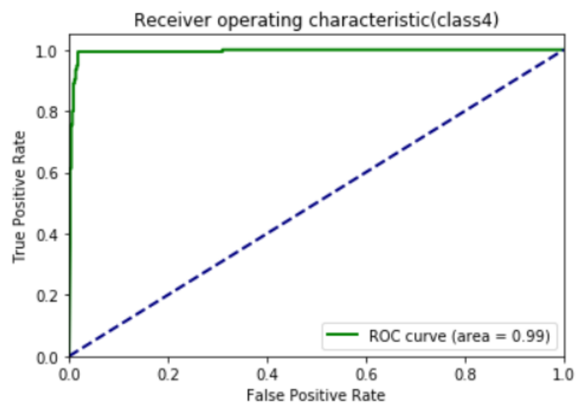
ROC and AUC for SVM model:

```
classifier = OneVsRestClassifier(SVC(kernel='rbf',probability = True))
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```









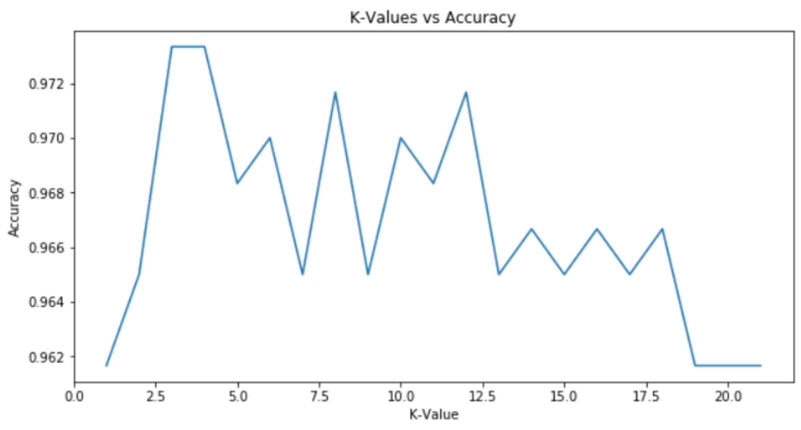
Creating a Model with K-Nearest Neighbours:

Here we are implementing KNN with the help of sklearn library. Inorder to find the optimum K value, we have done hyper-parametric tuning of K in the range 1 to 22.

```python
# Create model
from sklearn.neighbors import KNeighborsClassifier
k_val = []
acu_val = []
for k in range(1,22):
    clf_knn = KNeighborsClassifier(n_neighbors = k)
    clf_knn.fit(X_train, y_train.argmax(axis=1))
    y_pred_knn = clf_knn.predict(X_test)
    k_val.append(k)
    acu = asc(y_pred_knn, y_test.argmax(axis=1))
    acu_val.append(acu)

a = np.asmatrix(k_val)
b = np.asmatrix(acu_val)
output = np.stack((a,b))
print('  kvalue vs Accuracy')
print(output.transpose())
```

The variation of K value with accuracy is plotted to find the optimum value of K for our model.

```python
# Plotting K-value vs Accuracy
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
plt.plot(k_val,acu_val)
plt.xlabel("K-Value")
plt.ylabel("Accuracy")
plt.title("K-Values vs Accuracy")
plt.show()
```



```
 kvalue vs Accuracy
[[ 1.          0.96166667]
 [ 2.          0.965     ]
 [ 3.          0.97333333]
 [ 4.          0.97333333]
 [ 5.          0.96833333]
 [ 6.          0.97      ]
 [ 7.          0.965     ]
 [ 8.          0.97166667]
 [ 9.          0.965     ]
 [10.          0.97      ]
 [11.          0.96833333]
 [12.          0.97166667]
 [13.          0.965     ]
 [14.          0.96666667]
 [15.          0.965     ]
 [16.          0.96666667]
 [17.          0.965     ]
 [18.          0.96666667]
 [19.          0.96166667]
 [20.          0.96166667]
 [21.          0.96166667]]
```

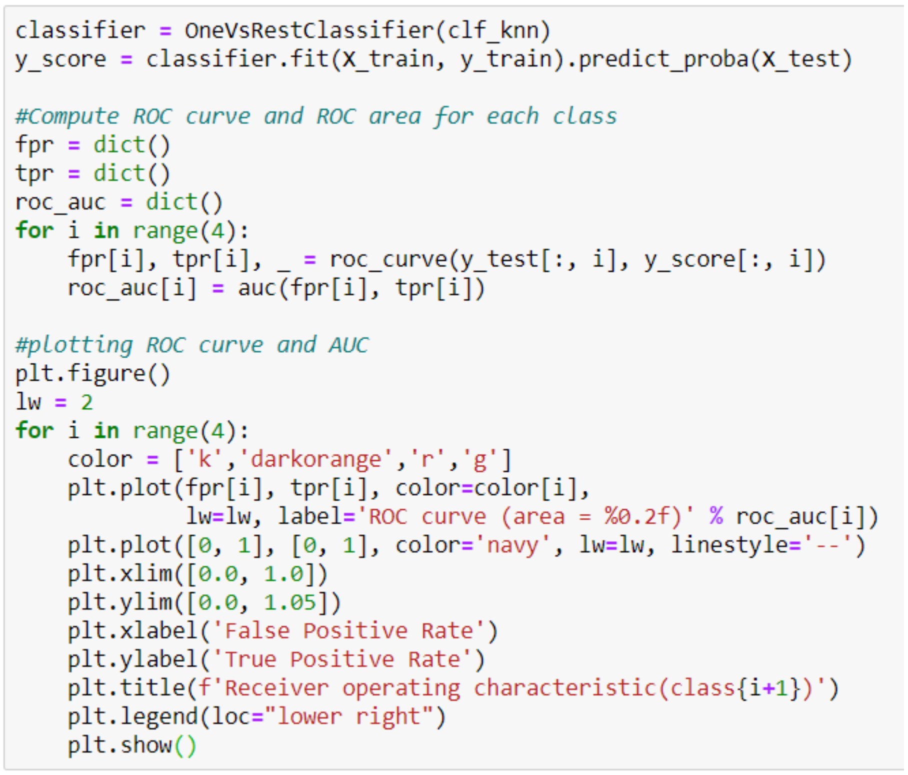
Now the best model will be the one with maximum accuracy for a given K value.

```python
i = acu_val.index(max(acu_val))
clf_knn = KNeighborsClassifier(n_neighbors = k_val[i])
clf_knn.fit(X_train, y_train)
y_pred_knn = clf_knn.predict(X_test)

# Making the Confusion Matrix
cm = confusion_matrix(y_test, y_pred_knn)
print(f"Accuracy: {asc(y_pred_knn, y_test)}")
print(("\nConfusion Matrix:"))
print(cm)
```

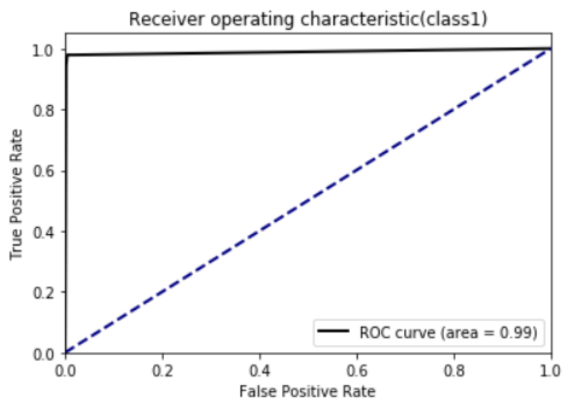
Now the accuracy and confusion matrix of the obtained model is as follows:

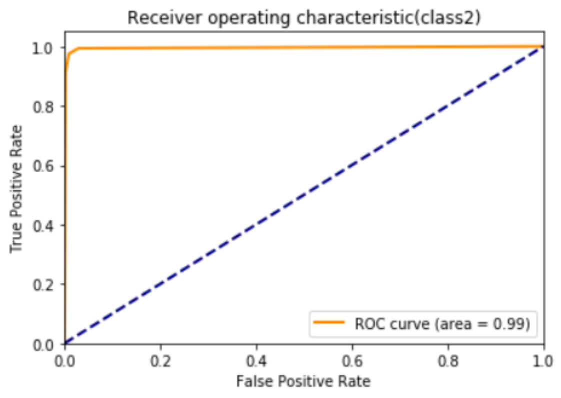
```
Accuracy: 0.9733333333333334

Confusion Matrix:
[[139   1   0   2]
 [  0 153   4   0]
 [  2   3 138   3]
 [  0   0   1 154]]
```

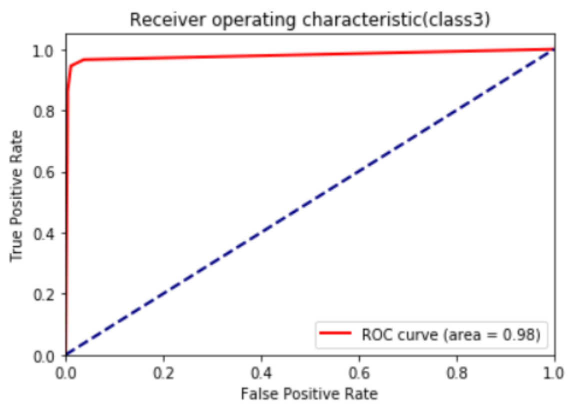
ROC and AUC for KNN model:

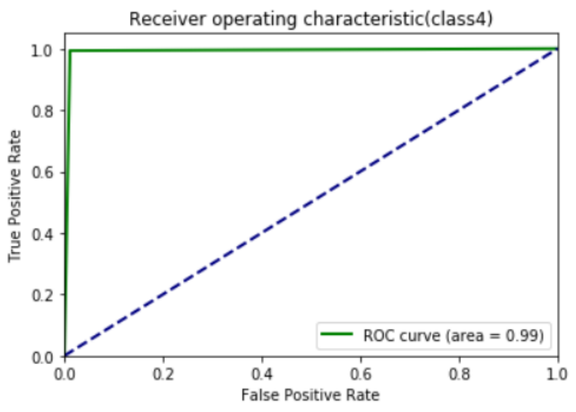


```
classifier = OneVsRestClassifier(clf_knn)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```









Tabulating the Results:

The results from all the four models created after imputation using decision tree method is shown below.

Model	Accuracy Score	Confusion Matrix
Decision Tree	0.953	Confusion Matrix: [[137 0 1 4] [0 150 7 0] [4 7 131 4] [0 0 1 154]]
SVM	0.77	Confusion Matrix: [[99 42 0 1] [0 156 1 0] [0 53 91 2] [0 39 0 116]]
LDA	0.958	Confusion Matrix: [[139 0 2 1] [0 141 16 0] [2 1 141 2] [0 0 1 154]]
QDA	0.968	Confusion Matrix: [[140 0 1 1] [0 153 4 0] [2 5 134 5] [0 0 1 154]]
KNN	0.973	Confusion Matrix: [[139 1 0 2] [0 153 4 0] [2 3 138 3] [0 0 1 154]]

From the above table we can see that KNN model produced the maximum accuracy.

Performing K-fold cross validation on model with best accuracy:

```
a=0
for i in range(2,500):
    scores_res = model_selection.cross_val_score(clf_knn, X, y.argmax(axis=1), cv=i)
    if a < scores_res.mean():
        a = scores_res.mean()
        j=i
print('max accuracy = ',a,'\nfolds = ', j)
```

The max accuracy and the number of folds at which we obtain the max accuracy is as follows:

```
max accuracy =  0.975
folds =  340
```

Imputing using Median and Predicting the Model:

The missing values are imputed using the median of the data points in the column.

```
from sklearn.impute import SimpleImputer
Xt = data.drop(["Room"],axis=1)
imp = SimpleImputer(missing_values=np.nan, strategy='median')
imp.fit(Xt)
```

	WiFi1	WiFi2	WiFi3	WiFi4	WiFi5	WiFi6	WiFi7	Room
0	-64.0	-56.0	-61.0	-66.0	-71.0	-82.0	-81.0	1
1	-55.0	-57.0	-61.0	-65.0	-71.0	-85.0	-85.0	1
2	-63.0	-60.0	-55.0	-67.0	-76.0	-85.0	-84.0	1
3	-55.0	-60.0	-68.0	-62.0	-77.0	-90.0	-80.0	1
4	-55.0	-65.0	-60.0	-63.0	-77.0	-81.0	-87.0	1

Table above shows first 5 rows for reference

All the missing values have been imputed using SimpleImputer from sklearn library choosing the strategy as median.

Converting the Output values into Binary:

This is done for using ‘OneVsRestClassfier’ library to print ROC and AUC curve later on.

```
X=df.iloc[:,:-1].values
y=df.iloc[:,7].values
y = label_binarize(y, classes=[1,2,3,4])
```

Splitting the Data Set into Train and Test data:

The overall dataset has been split into 70% training set and 30% testing test for determining the accuracy and obtaining the confusion matrix.

```
# Split data to test and train
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, random_state = 2)
```

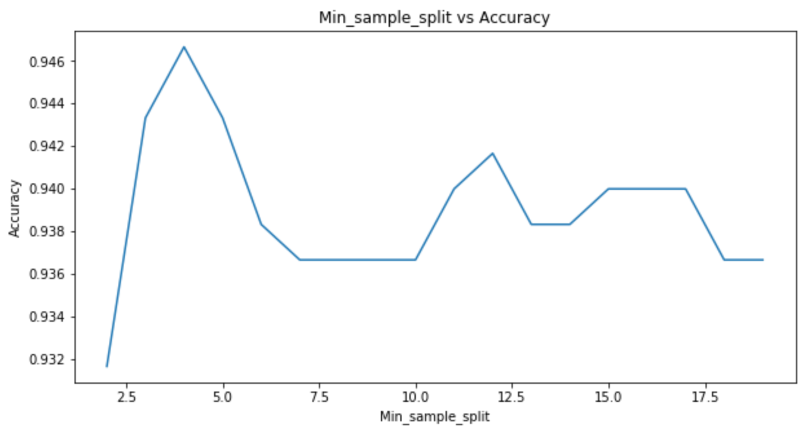
Creating a Model with Decision Trees:

Here we are implementing Decision Tree with the help of sklearn library. In order to obtain the optimum sample split and depth of the Decision Tree model, we have done hyper-parametric tuning. The optimum values are then chosen from the obtained results.

To find the variation of accuracy with sample split we use

```
# By changing the min_sample_split
acu1 = []
splits =[]
for i in range(2,20):
    # Create Decision Tree classifer object
    clf_dt1 = DecisionTreeClassifier(criterion="gini", min_samples_split=i)

    # Train Decision Tree Classifer
    clf_dt1 = clf_dt1.fit(X_train,y_train)

    #Predict the response for test dataset
    y_pred = clf_dt1.predict(X_test)
    splits.append(i)
    acu1.append(metrics.accuracy_score(y_test, y_pred))
plt.figure(figsize = (10,5))
plt.plot(splits,acu1)
plt.xlabel("Min_sample_split")
plt.ylabel("Accuracy")
plt.title("Min_sample_split vs Accuracy")
plt.show()

a = np.asmatrix(splits)
b = np.asmatrix(acu1)
output = np.stack((a,b))
print('  split vs Accuracy')
print(output.transpose())
```

The variation of sample split with accuracy is plotted.

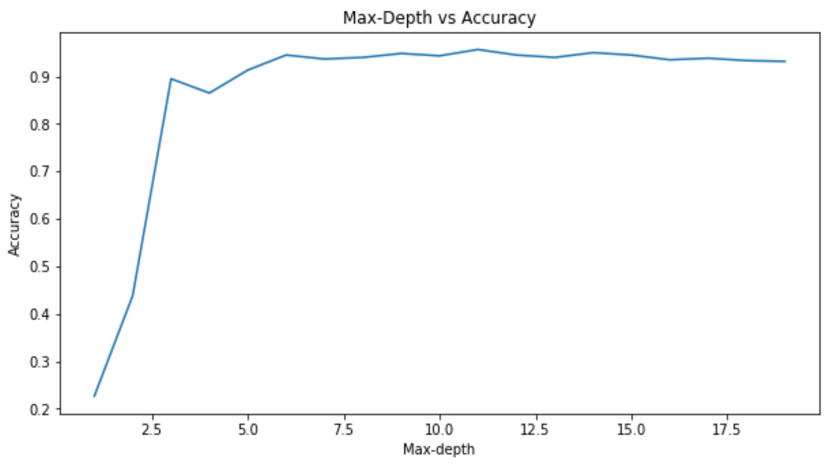


```
Sample_split vs Accuracy
[[ 2.          0.93166667]
 [ 3.          0.94333333]
 [ 4.          0.94666667]
 [ 5.          0.94333333]
 [ 6.          0.93833333]
 [ 7.          0.93666667]
 [ 8.          0.93666667]
 [ 9.          0.93666667]
 [10.          0.93666667]
 [11.          0.94      ]
 [12.          0.94166667]
 [13.          0.93833333]
 [14.          0.93833333]
 [15.          0.94      ]
 [16.          0.94      ]
 [17.          0.94      ]
 [18.          0.93666667]
 [19.          0.93666667]]
```

Similarly the variation of max depth with accuracy is found by

```
#By changing the max_depth
acu2 = []
dep =[]
for i in range(1,20):
    # Create Decision Tree classifer object
    clf_dt2 = DecisionTreeClassifier(criterion="gini", max_depth=i)

    # Train Decision Tree Classifer
    clf_dt2 = clf_dt2.fit(X_train,y_train)

    #Predict the response for test dataset
    y_pred = clf_dt2.predict(X_test)
    dep.append(i)
    acu2.append(metrics.accuracy_score(y_test, y_pred))
plt.figure(figsize = (10,5))
plt.plot(dep,acu2)
plt.xlabel("Max-depth")
plt.ylabel("Accuracy")
plt.title("Max-Depth vs Accuracy")
plt.show()
a = np.asmatrix(dep)
b = np.asmatrix(acu2)
output = np.stack((a,b))
print('Max Depth vs Accuracy')
print(output.transpose())
```

The variation of max depth with accuracy is plotted.



```
Max Depth vs Accuracy
[[ 1.          0.22666667]
 [ 2.          0.43833333]
 [ 3.          0.895     ]
 [ 4.          0.865     ]
 [ 5.          0.91333333]
 [ 6.          0.945     ]
 [ 7.          0.93666667]
 [ 8.          0.94      ]
 [ 9.          0.94833333]
 [10.          0.94333333]
 [11.          0.95666667]
 [12.          0.945     ]
 [13.          0.94      ]
 [14.          0.95      ]
 [15.          0.945     ]
 [16.          0.935     ]
 [17.          0.93833333]
 [18.          0.93333333]
 [19.          0.93166667]]
```

Now the best model will be the one with max accuracy from both the cases of sample split and depth.

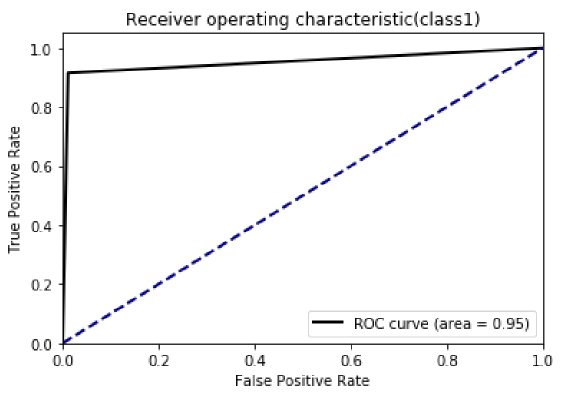
```
if max(acu2)>max(acu1):
    i = acu2.index(max(acu2))
    clf_dt = DecisionTreeClassifier(criterion = 'gini', max_depth = dep[i])
    clf_dt = clf_dt.fit(X_train,y_train)
    #Predicting results using testing data
    y_pred_dt = clf_dt.predict(X_test)
else:
    i = acu1.index(max(acu1))
    clf_dt = DecisionTreeClassifier(criterion = 'gini', min_samples_split = splits[i])
    clf_dt = clf.fit(X_train,y_train)
    #Predicting results using testing data
    y_pred_dt = clf_dt.predict(X_test)

#Creating confusion matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_dt.argmax(axis=1))
#Printing accuracy
print(f"Accuracy: {asc(y_pred_dt, y_test)}")
print(("\nConfusion Matrix:"))
print(cm)
```

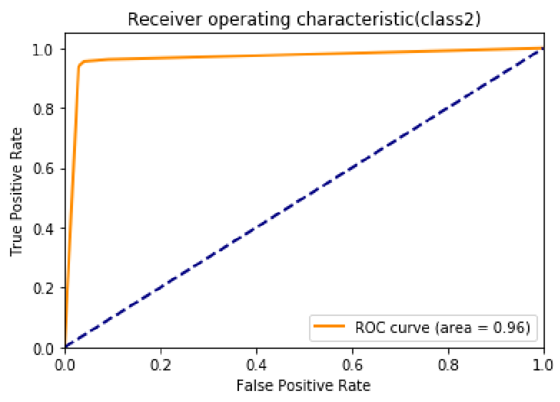
Accuracy and confusion matrix of the obtained model is.

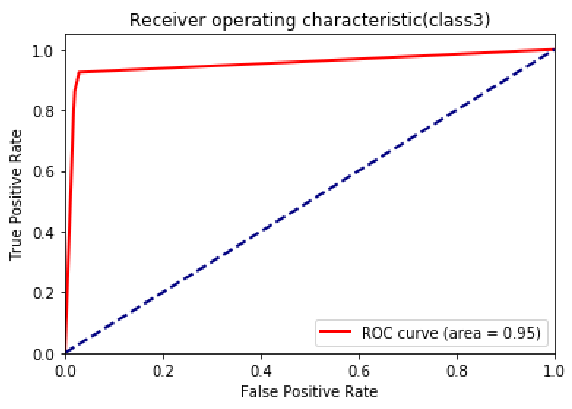
```
Accuracy: 0.9483333333333334

Confusion Matrix:
[[137   0   1   4]
 [  0 147  10   0]
 [  1   7 133   5]
 [  1   0   2 152]]
```

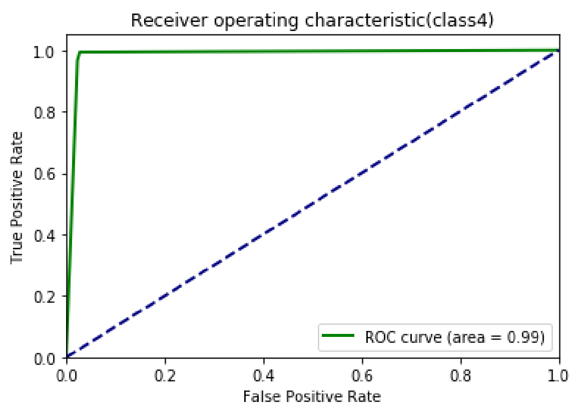
ROC and AUC of Decision Tree model:

```
classifier = OneVsRestClassifier(clf_dt)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```









The ROC curve obtained shows that it a good model and AUC of more than 0.95 confirms this. This is the reason behind a good accuracy score of 0.948 for this model.

Creating a Model with Support Vector Machines:

Here we are implementing SVM with the help of sklearn library.

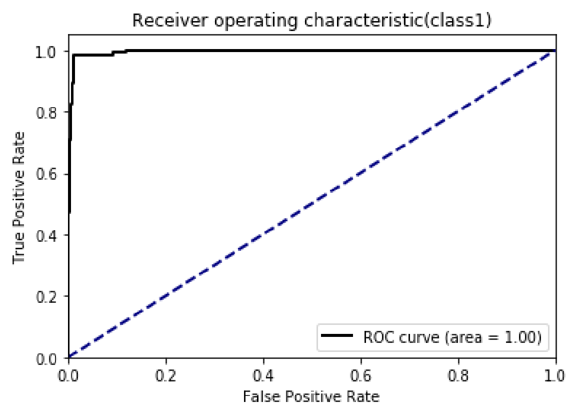
```
from sklearn.svm import SVC
clf_svm = SVC(kernel = 'rbf')
clf_svm.fit(X_train, y_train.argmax(axis=1))

# Predicting the test set result
y_pred_svm = clf_svm.predict(X_test)

# Making the Confusion Matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_svm)

print(f"Accuracy: {asc(y_pred_svm, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

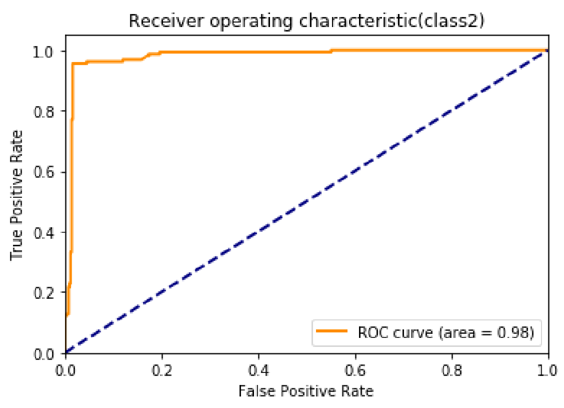
The confusion matrix and accuracy obtained using this model are.

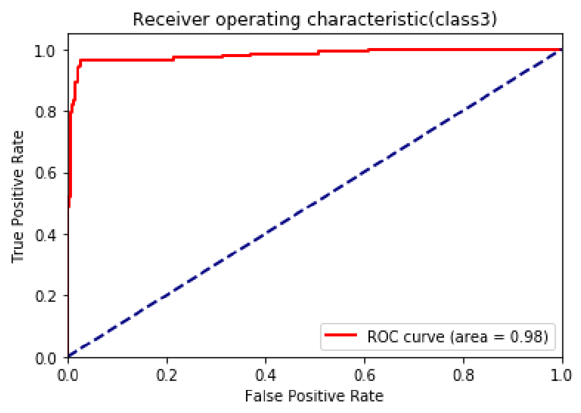
```
Accuracy: 0.7083333333333334

Confusion Matrix:
[[ 90  52   0   0]
 [  0 156   1   0]
 [  0  62  83   1]
 [  1  58   0  96]]
```

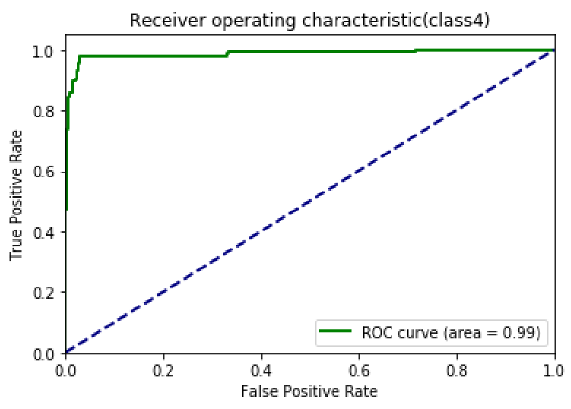
ROC and AUC for SVM Model:

```
classifier = OneVsRestClassifier(SVC(kernel='rbf',probability = True))
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```


Receiver operating characteristic(class1)
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate


Receiver operating characteristic(class2)
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
ROC curve (area = 0.98)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate


Receiver operating characteristic(class3)
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
ROC curve (area = 0.98)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate


Receiver operating characteristic(class4)
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

Creating a Model with Linear Discriminant Analysis:

Here we are implementing LDA with the help of sklearn library.

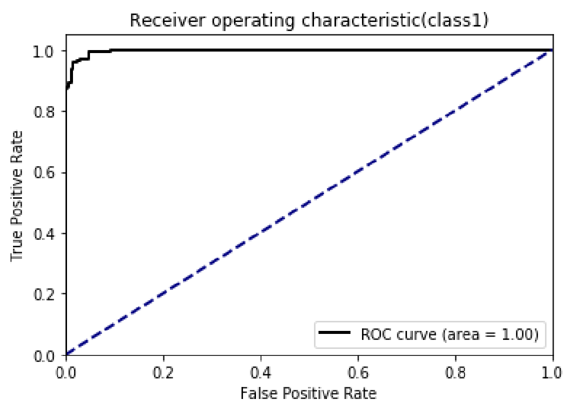
```
# Create model
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
clf_lda = LinearDiscriminantAnalysis()
clf_lda.fit(X_train, y_train.argmax(axis=1))

# Predicting the test set result
y_pred_lda = clf_lda.predict(X_test)
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_lda)

print(f"Accuracy: {asc(y_pred_lda, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

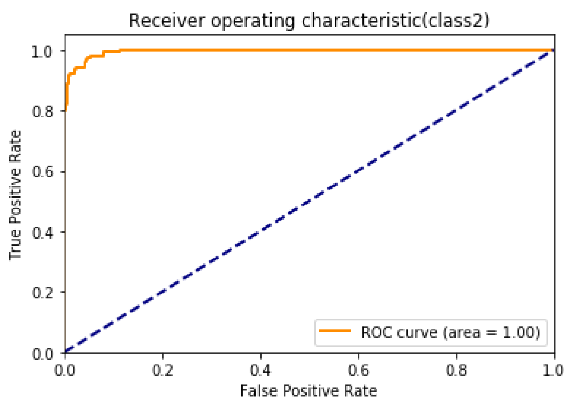
The confusion matrix and accuracy obtained using this model are.

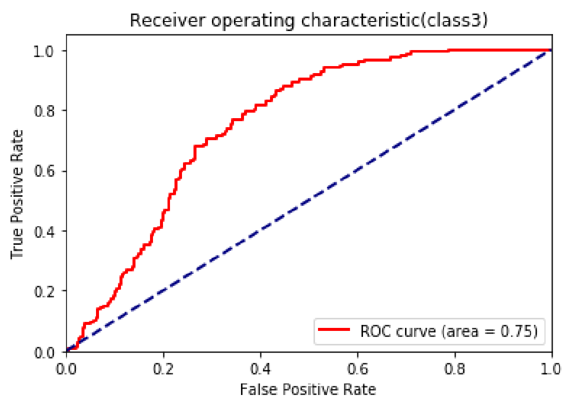
```
Accuracy: 0.9283333333333333

Confusion Matrix:
[[138   0   4   0]
 [  2 133  22   0]
 [  1   1 141   3]
 [  2   0   8 145]]
```

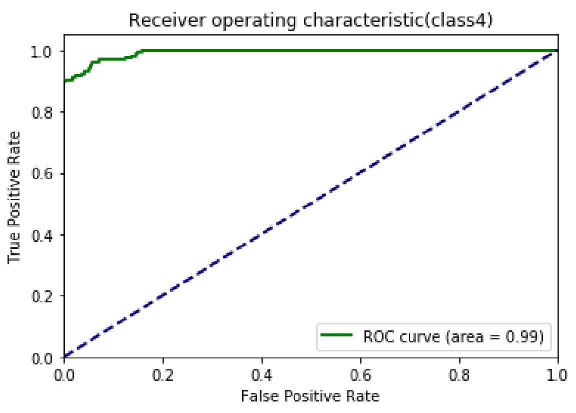
ROC and AUC for LDA Model:

```
classifier = OneVsRestClassifier(clf_lda)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```









Creating a Model with Quadratic Discriminant Analysis:

Here we are implementing QDA with the help of sklearn library.

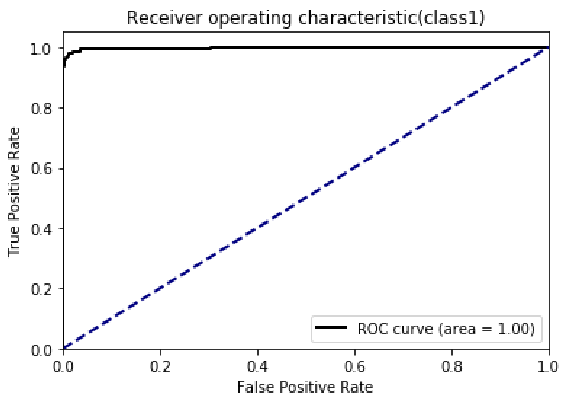
```
# Create model
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
clf_qda = QuadraticDiscriminantAnalysis()
clf_qda.fit(X_train, y_train.argmax(axis=1))

# Predicting the test set result
y_pred_qda = clf_qda.predict(X_test)

# Making the Confusion Matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_qda)

print(f"Accuracy: {asc(y_pred_qda, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

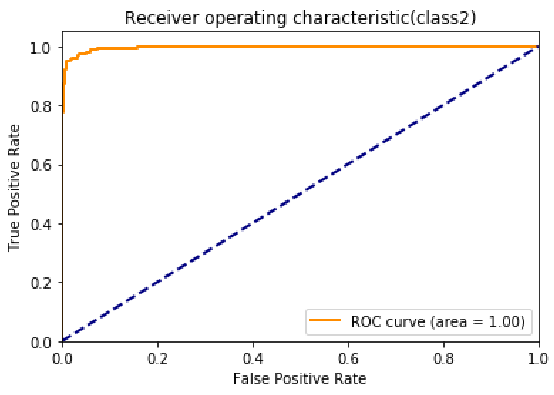
The confusion matrix and accuracy obtained using this model are.

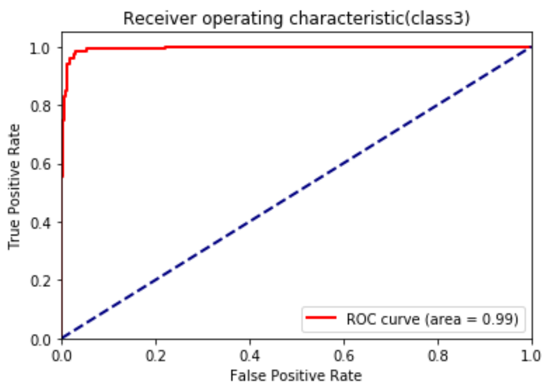
```
Accuracy: 0.97

Confusion Matrix:
[[140   0   1   1]
 [  0 151   6   0]
 [  2   2 137   5]
 [  0   0   1 154]]
```

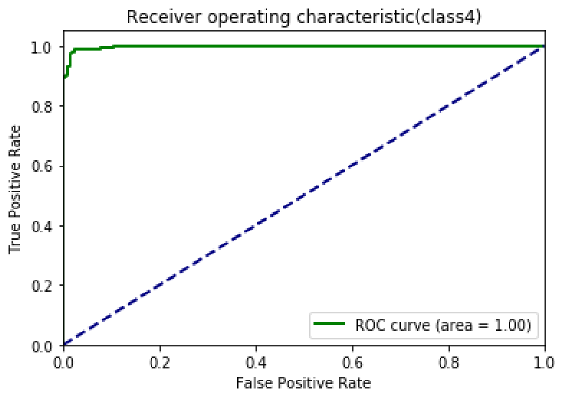
ROC and AUC for QDA Model:

```
classifier = OneVsRestClassifier(clf_lda)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```



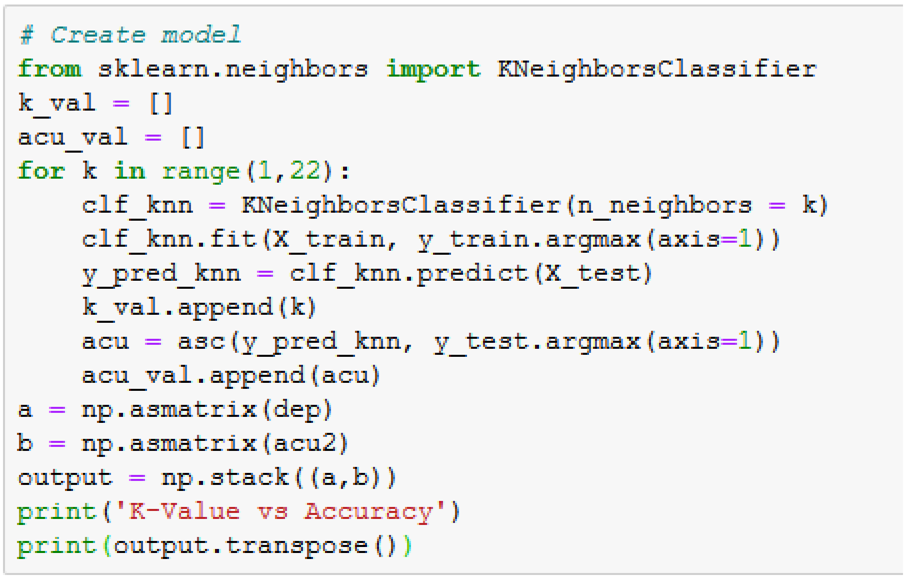






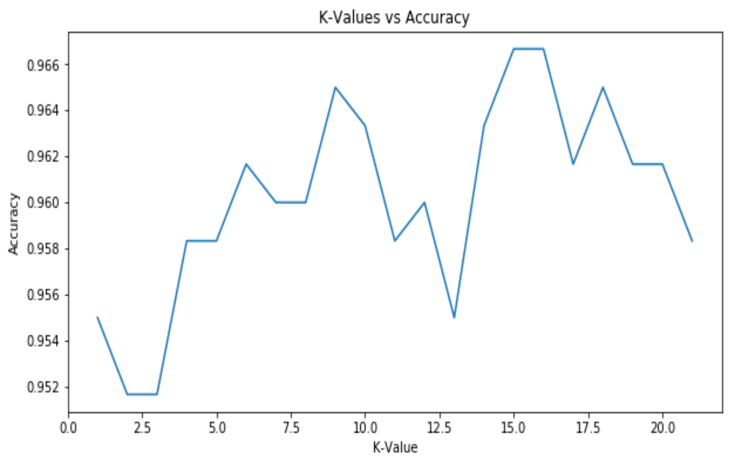
Creating a Model with K-Nearest Neighbours:

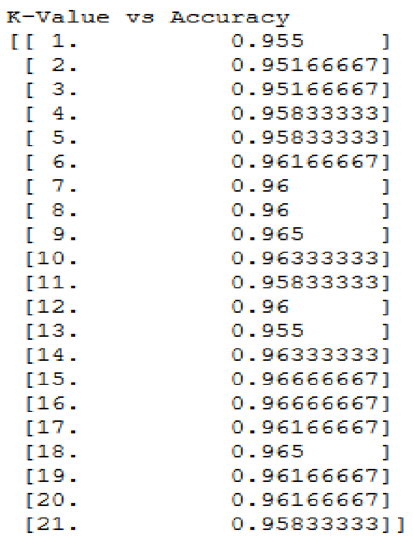
Here we are implementing KNN with the help of sklearn library. Inorder to find the optimum K value, we have done hyper-parametric tuning for the value of K in the range of 1 to 22.



```
# Create model
from sklearn.neighbors import KNeighborsClassifier
k_val = []
acu_val = []
for k in range(1,22):
    clf_knn = KNeighborsClassifier(n_neighbors = k)
    clf_knn.fit(X_train, y_train.argmax(axis=1))
    y_pred_knn = clf_knn.predict(X_test)
    k_val.append(k)
    acu = asc(y_pred_knn, y_test.argmax(axis=1))
    acu_val.append(acu)
a = np.asmatrix(dep)
b = np.asmatrix(acu2)
output = np.stack((a,b))
print('K-Value vs Accuracy')
print(output.transpose())
```

The variation of K value with accuracy is plotted to find the optimum value of K for our model.





```
K-Value vs Accuracy
[[ 1.          0.955     ]
 [ 2.          0.95166667]
 [ 3.          0.95166667]
 [ 4.          0.95833333]
 [ 5.          0.95833333]
 [ 6.          0.96166667]
 [ 7.          0.96      ]
 [ 8.          0.96      ]
 [ 9.          0.965     ]
 [10.          0.96333333]
 [11.          0.95833333]
 [12.          0.96      ]
 [13.          0.955     ]
 [14.          0.96333333]
 [15.          0.96666667]
 [16.          0.96666667]
 [17.          0.96166667]
 [18.          0.965     ]
 [19.          0.96166667]
 [20.          0.96166667]
 [21.          0.95833333]]
```

Now the best model will be the one with maximum accuracy for a given K value is found out.

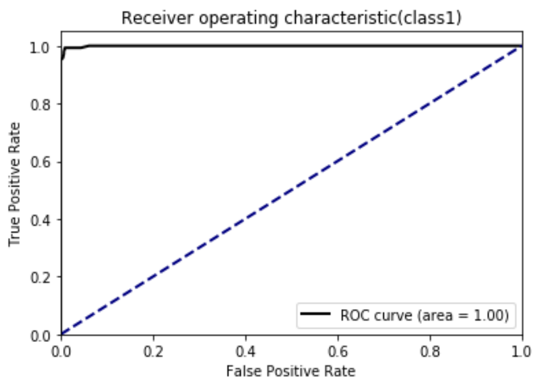
```
i = acu_val.index(max(acu_val))
clf_knn = KNeighborsClassifier(n_neighbors = k_val[i])
clf_knn.fit(X_train, y_train.argmax(axis=1))
y_pred_knn = clf_knn.predict(X_test)

# Making the Confusion Matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred_knn)
print(f"Accuracy: {asc(y_pred_knn, y_test.argmax(axis=1))}")
print(("\nConfusion Matrix:"))
print(cm)
```

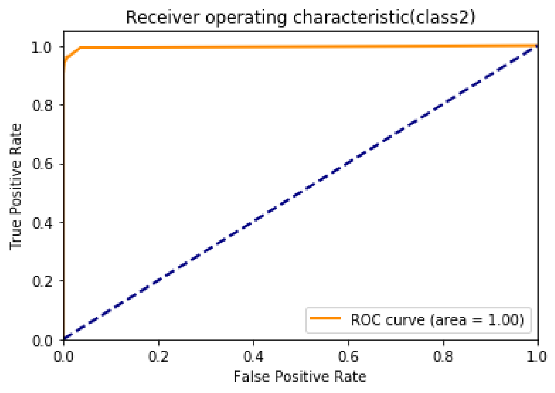
Accuracy and confusion matrix of the obtained model is as follows

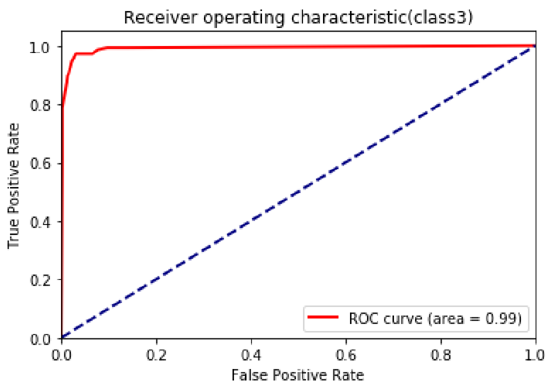
```
Accuracy: 0.9666666666666667

Confusion Matrix:
[[141   0   1   0]
 [  1 144  12   0]
 [  2   0 142   2]
 [  1   0   1 153]]
```

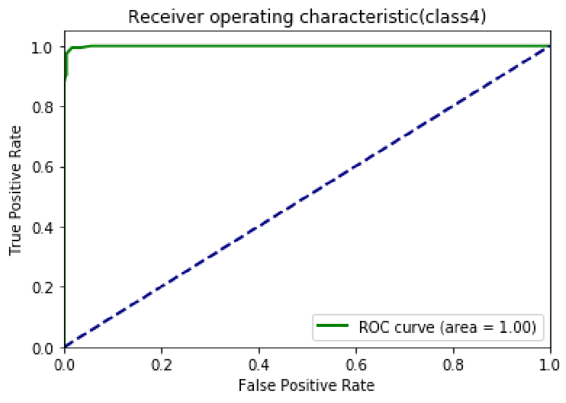
ROC and AUC for KNN Model:

```
classifier = OneVsRestClassifier(clf_knn)
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

#Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(4):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

#plotting ROC curve and AUC
plt.figure()
lw = 2
for i in range(4):
    color = ['k','darkorange','r','g']
    plt.plot(fpr[i], tpr[i], color=color[i],
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver operating characteristic(class{i+1})')
    plt.legend(loc="lower right")
    plt.show()
```


Receiver operating characteristic(class1)
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver operating characteristic(class2)
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0

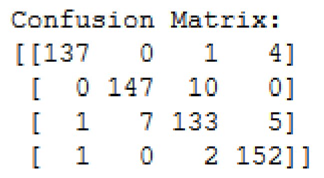

Receiver operating characteristic(class3)
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0

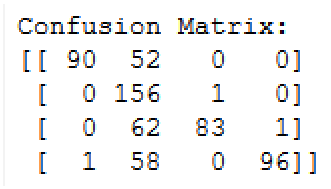

Receiver operating characteristic(class4)
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0

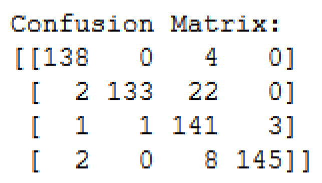
Tabulating the Results:

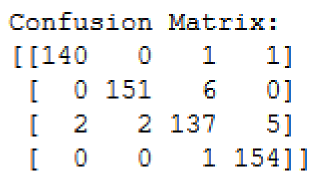
The results from all the four models created after imputation using median method is shown below.

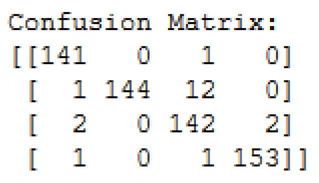
Model	Accuracy Score	Confusion Matrix

Decision Tree	0.948	 `Confusion Matrix:`
		`[[137   0   1   4]`
		` [  0 147  10   0]`
		` [  1   7 133   5]`
		` [  1   0   2 152]]`

SVM	0.708	 `Confusion Matrix:`
		`[[ 90  52   0   0]`
		` [  0 156   1   0]`
		` [  0  62  83   1]`
		` [  1  58   0  96]]`

LDA	0.928	 `Confusion Matrix:`
		`[[138   0   4   0]`
		` [  2 133  22   0]`
		` [  1   1 141   3]`
		` [  2   0   8 145]]`

QDA	0.970	 `Confusion Matrix:`
		`[[140   0   1   1]`
		` [  0 151   6   0]`
		` [  2   2 137   5]`
		` [  0   0   1 154]]`

KNN	0.966	 `Confusion Matrix:`
		`[[141   0   1   0]`
		` [  1 144  12   0]`
		` [  2   0 142   2]`
		` [  1   0   1 153]]`

From the above table we can see that QDA model produced the maximum accuracy.

Performing K-fold cross validation on model with best accuracy:

```
a = 0
for i in range(2,500):
    scores_res = model_selection.cross_val_score(clf_qda, X, y.argmax(axis=1), cv=i)
    if a < scores_res.mean():
        a = scores_res.mean()
        j = i
print(f'Max Accuracy = {a:.3}\nNo. of Folds = {j}')
    #print(scores_res.mean())
```

The max accuracy and the number of folds at which we obtain the max accuracy is as follows:

```
Max Accuracy = 0.972
No. of Folds = 340
```

Tabulating the best results:

Imputation Method	Mean	Median	Decision Tree	KNN
Best Model	QDA	QDA	KNN	KNN
Accuracy	0.968	0.970	0.973	0.976
K-Fold Accuracy	0.969	0.972	0.975	0.977

Conclusion:

After going through various imputation and classification techniques we observed that by using KNN regression to impute the missing data and by further using KNN classification to create our model we get the maximum accuracy as compared to other methods as shown in the above table.

Lessons learned:

- ➢ Knock off data from a dataset
- ➢ Various imputation techniques to fill missing data
- ➢ One hot encoding
- ➢ Creating ROC and AUC for multiclass classification
- ➢ Comparing various classification models
- ➢ Imputation using mean,KNN is sensitive to outliers whereas median,decision tree are not much affected by it.