1) First import the useful libraries as this dataset is mostly about numerical part thats why we have to import two libraries that are numpy and pandas,numpy handles the numerical part

In [18]:
```
import pandas as pd
import numpy as np
```

2) Reading and loading the file into a dataframe using read_csv() method:

In [19]:
```
## reading the .data file using pandas
df=pd.read_csv("CreditCardFraud.csv")
```

In [20]:
```
#checking the first five row of data set by using df.head() as like to
 if we want the last element of the list we use df.tail()
df.head()
```

Out[20]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0

5 rows × 31 columns

In [21]:
```
#creating target series
target=df['Class']
target
```

Out[21]:
```
0         0
1         0
2         0
3         0
4         0
         ..
284802    0
284803    0
284804    0
284805    0
284806    0
Name: Class, Length: 284807, dtype: int64
```

In [22]:
```
#dropping the target variable from the data set
df.drop('Class',axis=1,inplace=True)
df.shape
```

Out[22]:
```
(284807, 30)
```

Converting the target variable into numpy array

In [23]:
```
#converting them to numpy arrays
X=np.array(df)
y=np.array(target)
X.shape
y.shape
```

Out[23]:
```
(284807,)
```

In [24]:
```
#distribution of the target variable
```

```
len(y[y==1])
len(y[y==0])
```

Out[24]: 284315

In [26]:
```
#splitting the data set into train and test (75:25)
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,rando
m_state=1)
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

```
(213605, 30) (71202, 30) (213605,) (71202,)
```

SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie together.

In [27]:
```
#applyting SMOTE to oversample the minority class
from imblearn.over_sampling import SMOTE
sm=SMOTE(random_state=2)
X_sm,y_sm=sm.fit_sample(X_train,y_train)
print(X_sm.shape,y_sm.shape)
```

```
(426448, 30) (426448,)
```

In [28]:
```
print(len(y_sm[y_sm==1]),len(y_sm[y_sm==0]))
```

```
213224 213224
```

importing logistic regression models

'''Q)When we should use logistic regression? =>logistic regression is used to find the relationship between dependent and independent variable '''

In [30]:
```
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
from sklearn import metrics
```

In [32]:
```
#Logistic Regression
logreg=LogisticRegression()
logreg.fit(X_sm,y_sm)
y_logreg=logreg.predict(X_test)
y_logreg_prob=logreg.predict_proba(X_test)[:,1]
```

```
C:\Users\Chiranjivi\anaconda3\lib\site-packages\sklearn\linear_model\_l
ogistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=
1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

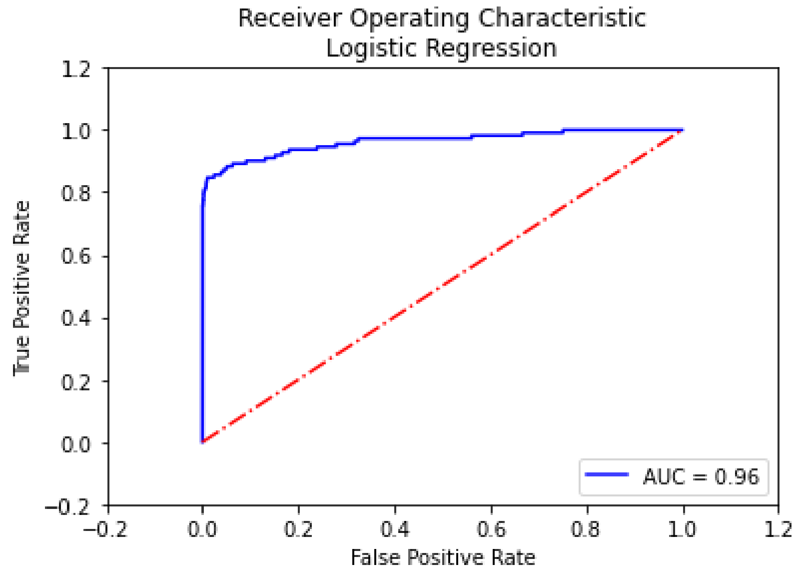
In [33]:
```
#Performance metrics evaluation
print("Confusion Matrix:\n",metrics.confusion_matrix(y_test,y_logreg))
print("Accuracy:\n",metrics.accuracy_score(y_test,y_logreg))
print("Precision:\n",metrics.precision_score(y_test,y_logreg))
print("Recall:\n",metrics.recall_score(y_test,y_logreg))
print("AUC:\n",metrics.roc_auc_score(y_test,y_logreg_prob))
auc=metrics.roc_auc_score(y_test,y_logreg_prob)
```

```
Confusion Matrix:
 [[70179   912]
 [   17    94]]
Accuracy:
 0.9869526136906267
Precision:
 0.09343936381709742
Recall:
 0.8468468468468469
AUC:
 0.9627392299249498
```

The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR).

In [34]:

```
#plotting the ROC curve
fpr,tpr,thresholds=metrics.roc_curve(y_test,y_logreg_prob)
plt.plot(fpr,tpr,'b', label='AUC = %0.2f'% auc)
plt.plot([0,1],[0,1],'r-.')
plt.xlim([-0.2,1.2])
plt.ylim([-0.2,1.2])
plt.title('Receiver Operating Characteristic\nLogistic Regression')
plt.legend(loc='lower right')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

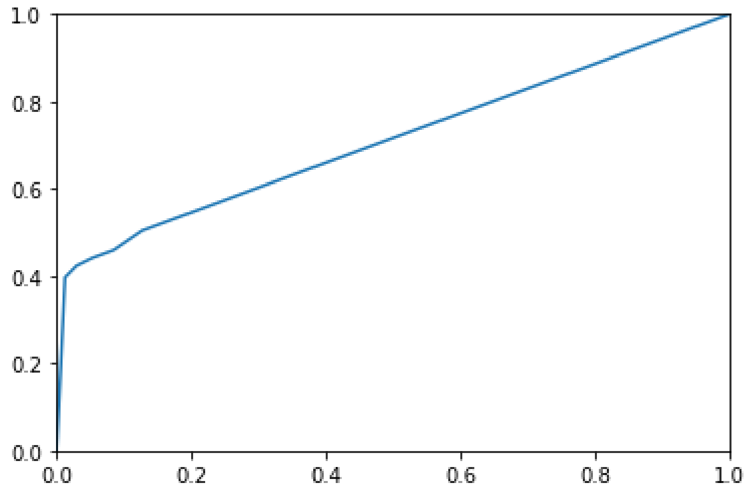


In [35]:

```
#trying with anothear alogrirthim
#K Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X_sm,y_sm)
y_knn=knn.predict(X_test)
y_knn_prob=knn.predict_proba(X_test)[:,1]

#metrics evaluation
print(metrics.confusion_matrix(y_test,y_knn))
print(metrics.accuracy_score(y_test,y_knn))
print(metrics.precision_score(y_test,y_knn))
print(metrics.recall_score(y_test,y_knn))
print(metrics.roc_auc_score(y_test,y_knn_prob))

#plotting the ROC curve
fpr,tpr,thresholds=metrics.roc_curve(y_test,y_knn_prob)
plt.plot(fpr,tpr)
plt.xlim([0.0,1.0])
```

```
plt.ylim([0.0,1.0])
plt.show()
```

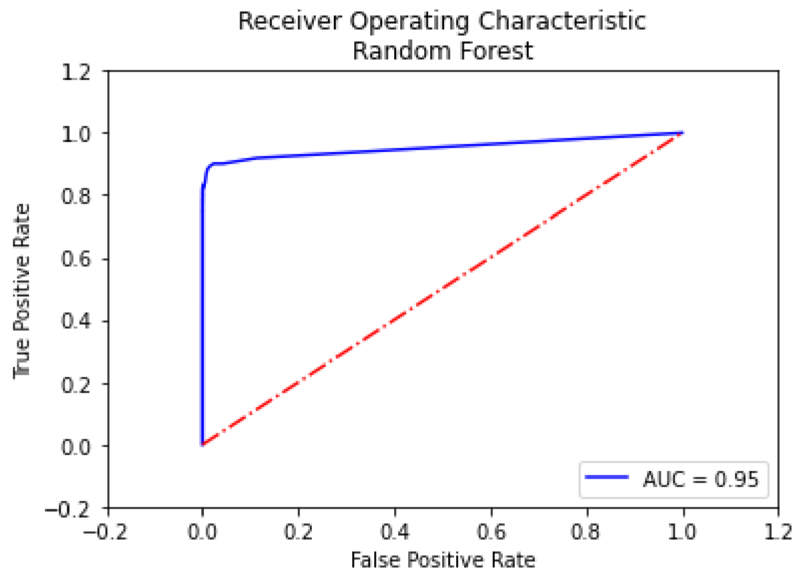
```
[[67323  3768]
 [   62    49]]
0.9462093761411197
0.012837306785433586
0.44144144144144143
0.711052171300304
```



In [36]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier(random_state=3)
rf.fit(X_sm,y_sm)
y_rf=rf.predict(X_test)
y_rf_prob=rf.predict_proba(X_test)[:,1]

#Performance metrics evaluation
print("Confusion Matrix:\n",metrics.confusion_matrix(y_test,y_rf))
print("Accuracy:\n",metrics.accuracy_score(y_test,y_rf))
print("Precision:\n",metrics.precision_score(y_test,y_rf))
print("Recall:\n",metrics.recall_score(y_test,y_rf))
print("AUC:\n",metrics.roc_auc_score(y_test,y_rf_prob))
```

```
auc=metrics.roc_auc_score(y_test,y_rf_prob)

#plotting the ROC curve
fpr,tpr,thresholds=metrics.roc_curve(y_test,y_rf_prob)
plt.plot(fpr,tpr,'b', label='AUC = %0.2f'% auc)
plt.plot([0,1],[0,1],'r-.')
plt.xlim([-0.2,1.2])
plt.ylim([-0.2,1.2])
plt.title('Receiver Operating Characteristic\nRandom Forest')
plt.legend(loc='lower right')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
Confusion Matrix:
 [[71077    14]
 [   24    87]]
Accuracy:
 0.9994663071262043
Precision:
 0.8613861386138614
Recall:
 0.7837837837837838
AUC:
 0.9527897311161015
```



In [ ]:

Conclusion

Fraud detection is a complex issue that requires a substantial amount of planning before throwing machine learning algorithms at it which makes sure that the customer's money is safe and not easily tampered with.As compared to three alogorithims which we had used random forest alog is giving us best accuracy score . Future work will include a comprehensive parameter tuning of the Random Forest algorithm here is a link you can refer for https://towardsdatascience.com/hyperparameter-tuning-for-machine-learning-models-1b80d783b946 . Having a data set with non-anonymized features would make this particularly interesting as outputting the feature importance would enable one to see what specific factors are most important for detecting fraudulent transactions.

In [ ]: