In [62]:
```
import numpy as np
import pandas as pd
data=pd.read_csv('creditcard.csv')
```

In [63]:
```
data.tail()
```

Out[63]:

	Time	V1	V2	V3	V4	V5	V6	V7	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0

5 rows × 31 columns

In [64]:
```
data.describe()
```

Out[64]:

	Time	V1	V2	V3	V4	V5
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01

8 rows × 31 columns

In [65]:
```
numcols = data.columns
numcols
```

Out[65]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V
10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19',
'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amoun
t',
       'Class'],
      dtype='object')
```

In [ ]:

In [66]:
```
Features = np.array(data[['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V
7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V
20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount'
]])
Features[2, :]
```

Out[66]:
```
array([ 1.00000000e+00, -1.35835406e+00, -1.34016307e+00,  1.77320934e+
00,
        3.79779593e-01, -5.03198133e-01,  1.80049938e+00,  7.91460956e-
01,
        2.47675787e-01, -1.51465432e+00,  2.07642865e-01,  6.24501459e-
01,
        6.60836853e-02,  7.17292731e-01, -1.65945923e-01,  2.34586495e+
00,
       -2.89008319e+00,  1.10996938e+00, -1.21359313e-01, -2.26185710e+
00,
        5.24979725e-01,  2.47998153e-01,  7.71679402e-01,  9.09412262e-
01,
       -6.89280956e-01, -3.27641834e-01, -1.39096572e-01, -5.53527940e-
02,
       -5.97518406e-02,  3.78660000e+02])
```

In [67]:
```
import numpy.random as nr
import sklearn.model_selection as ms
nr.seed(1234)
labels = np.array(data['Class'])
index = range(data.shape[0])
index = ms.train_test_split(index,test_size=0.3)
x_train = np.array(Features[index[0], :])
y_train = np.ravel(labels[index[0]])
x_test = np.array(Features[index[1], :])
y_test = np.ravel(labels[index[1]])
print (x_train.shape)
print (y_train.shape)
```

```
(199364, 30)
(199364,)
```

In [68]:
```
from sklearn.linear_model import LogisticRegression
log_model = LogisticRegression(fit_intercept = False)
log_model.fit(x_train, y_train)
```

Out[68]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=
False,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.00
01,
          verbose=0, warm_start=False)
```

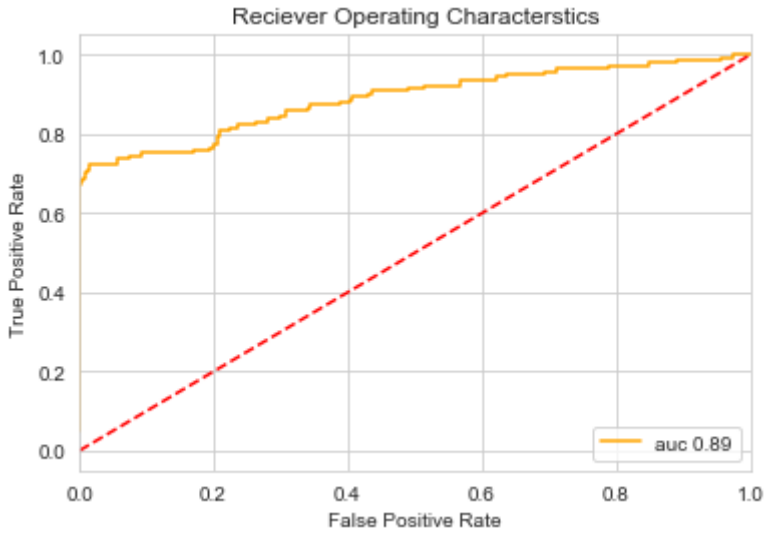
In [69]:
```
print(log_model.intercept_)
print(log_model.coef_)
```

```
0.0
[[-1.06538048e-04  3.44045175e-01 -5.04828064e-01 -1.07200556e+00
   9.58556522e-02 -1.42254101e-01 -7.33140557e-02  2.82781887e-01
  -2.98895654e-01 -4.99939291e-01 -2.66557875e-01 -3.13902844e-01
  -6.41374648e-02 -3.31689375e-01 -7.99147390e-01 -5.36769170e-01
  -4.52869140e-01 -7.00288169e-01 -6.39602162e-02  5.73613285e-02
   1.89268754e-01  2.88184951e-01  4.33047705e-01  1.12167928e-01
  -3.61802493e-02 -3.89343238e-01  8.55358055e-02 -1.06646362e-01
   5.45698054e-02 -1.02707119e-02]]
```

In [72]:
```
import sklearn.metrics as sklm
y_hat_p = log_model.predict_proba(x_test)
def score_model(y_p, threshold):
    return np.array([1 if x > threshold else 0 for x in y_p[:, 1]])
y_hat = score_model(y_hat_p, 0.9)

def print_metrics(y_true, y_predicted):
    metrics = sklm.precision_recall_fscore_support(y_true, y_predicted)
    cfmat = sklm.confusion_matrix(y_true, y_predicted)
    print("                     Predicted Positive        Predicted Nega
tive")
    print("Actually Positive    %6d" %cfmat[0][0] + "                   %
6d" %cfmat[0][1])
    print("Actually Negative    %6d" %cfmat[1][0] + "                   %
6d" %cfmat[1][1])
    print("")
    print("Accuracy: " + str(sklm.accuracy_score(y_true, y_predicted)))
    print("")
    print("             Positive       Negative")
    print("Num Cases:  %6f"%metrics[3][0] + "          %6.2f"%metrics[3]
[1])
    print("precision:  %6.2f"%metrics[0][0] + "          %6.2f"%metrics
[0][1])
    print("Recall:     %6.2f"%metrics[1][0] + "          %6.2f"%metrics
[1][1])
    print("fscore:     %6.2f"%metrics[2][0] + "          %6.2f"%metrics
[2][1])

print_metrics(y_test, y_hat)
```

```
                     Predicted Positive        Predicted Negative
Actually Positive     85285                      17
Actually Negative        53                      88

Accuracy: 0.9991807403766253

             Positive       Negative
Num Cases:  85302.000000          141.00
precision:     1.00              0.84
Recall:        1.00              0.62
fscore:        1.00              0.72
```

In [71]:
```
def plot_roc(y_true, prob):
    fpr, tpr, threshold = sklm.roc_curve(y_true, prob[:, 1])
    auc = sklm.auc(fpr, tpr)
    plt.plot(fpr, tpr, color = "orange", label = 'auc %0.2f' % auc)
    plt.plot([0,1], [0,1], 'r--')
    plt.xlim([0,1])
    plt.ylabel([0,1])
    plt.title("Reciever Operating Characterstics")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend(loc = 'lower right')
    plt.show()
plot_roc(y_test, y_hat_p)
```



In [ ]: