88/notebooks/Untitled1.ipynb?kernel_name=python3#

jupyter Untitled1 Last Checkpoint: 08/26/2021 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [1]:
```
# Importing the libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
pd.set_option('display.max_columns', 500)
```

In [3]:
```
# Reading the dataset
df = pd.read_csv('creditcard.csv')
df.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670

In [4]:
```
df.shape
```

Out[4]:
```
(284807, 31)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [6]:
```
df.describe()
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.040130e-15	-1.698953e-15	-1.893285e-16	-3.147640e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01

In [9]:
```
# Cheking percent of missing values in columns
df_missing_columns = (round(((df.isnull().sum()/len(df.index))*100),2).to_frame('null')).sort_values('null', ascendi
df_missing_columns
```

Out[9]:

	null
Time	0.0
V16	0.0
Amount	0.0
V28	0.0
V27	0.0
V26	0.0
V25	0.0
V24	0.0
V23	0.0
V22	0.0
V21	0.0
V20	0.0
V19	0.0
V18	0.0
V17	0.0
V15	0.0
V1	0.0
V14	0.0
V13	0.0
V12	0.0
V11	0.0
V10	0.0
V9	0.0
V8	0.0
V7	0.0
V6	0.0
V5	0.0
V4	0.0
V3	0.0
V2	0.0
Class	0.0

In [10]:
```
classes = df['Class'].value_counts()
classes
```

Out[10]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [11]:
```
normal_share = round((classes[0]/df['Class'].count()*100),2)
normal_share
```

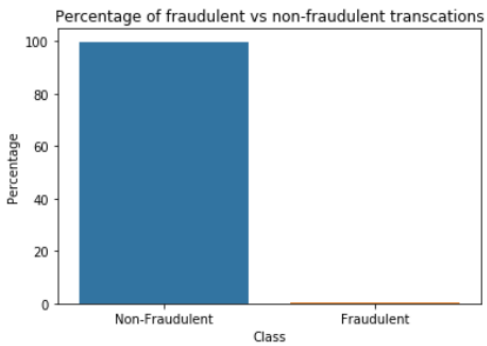
Out[11]: 99.83

In [12]:
```
fraud_share = round((classes[1]/df['Class'].count()*100),2)
fraud_share
```
Out[12]: 0.17

In [13]:
```
# Bar plot for the number of fraudulent vs non-fraudulent transcations
sns.countplot(x='Class', data=df)
plt.title('Number of fraudulent vs non-fraudulent transcations')
plt.show()
```

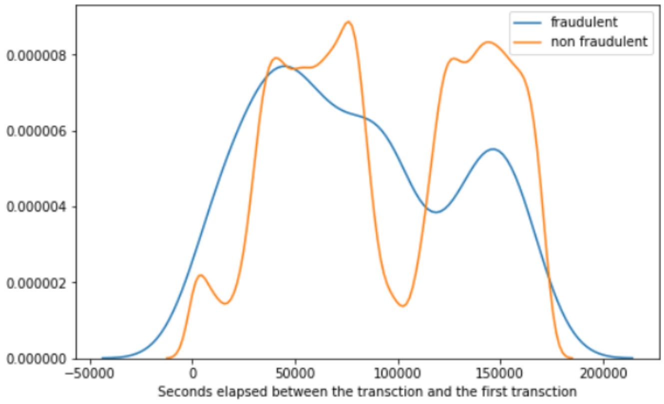
Class

In [14]:
```
# Bar plot for the percentage of fraudulent vs non-fraudulent transcations
fraud_percentage = {'Class':['Non-Fraudulent', 'Fraudulent'], 'Percentage':[normal_share, fraud_share]}
df_fraud_percentage = pd.DataFrame(fraud_percentage)
sns.barplot(x='Class',y='Percentage', data=df_fraud_percentage)
plt.title('Percentage of fraudulent vs non-fraudulent transcations')
plt.show()
```



In [15]:
```
# Creating fraudulent dataframe
data_fraud = df[df['Class'] == 1]
# Creating non fraudulent dataframe
data_non_fraud = df[df['Class'] == 0]
```

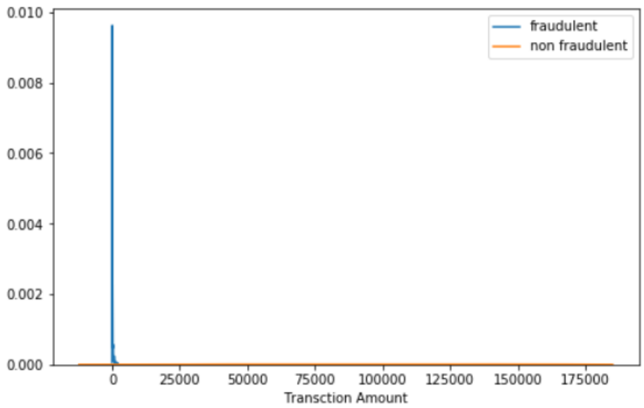
In [16]:
```
# Distribution plot
plt.figure(figsize=(8,5))
ax = sns.distplot(data_fraud['Time'],label='fraudulent',hist=False)
ax = sns.distplot(data_non_fraud['Time'],label='non fraudulent',hist=False)
ax.set(xlabel='Seconds elapsed between the transction and the first transction')
plt.show()
```



In [17]:
```
# Dropping the Time column
df.drop('Time', axis=1, inplace=True)
```

Observe the distribution of classes with amount

In [18]:
```
# Distribution plot
plt.figure(figsize=(8,5))
ax = sns.distplot(data_fraud['Amount'],label='fraudulent',hist=False)
ax = sns.distplot(data_non_fraud['Time'],label='non fraudulent',hist=False)
ax.set(xlabel='Transction Amount')
plt.show()
```



In [7]:
```
# Import library
from sklearn.model_selection import train_test_split
```

In [8]:
```
# Putting feature variables into X
X = df.drop(['Class'], axis=1)
```

In [9]:
```
# Putting target variable to y
y = df['Class']
```

In [10]:
```
# Splitting data into train and test set 80:20
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2, random_state=100)
```

In [11]:
```
# Standardization method
from sklearn.preprocessing import StandardScaler
```

In [12]:
```
# Instantiate the Scaler
scaler = StandardScaler()
```

In [13]:
```
# Fit the data into scaler and transform
X_train['Amount'] = scaler.fit_transform(X_train[['Amount']])
```

In [14]:
```
X_train.head()
```

Out[14]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	
201788	134039.0	2.023734	-0.429219	-0.691061	-0.201461	-0.162486	0.283718	-0.674694	0.192230	1.124319	-0.037763	0.308648	0.875063	-0.009562	0.
179369	124044.0	-0.145286	0.736735	0.543226	0.892662	0.350846	0.089253	0.626708	-0.049137	-0.732566	0.297692	0.519027	0.041275	-0.690783	0.
73138	54997.0	-3.015846	-1.920606	1.229574	0.721577	1.089918	-0.195727	-0.462586	0.919341	-0.612193	-0.966197	1.106534	1.026421	-0.474229	0.
208679	137226.0	1.851980	-1.007445	-1.499762	-0.220770	-0.568376	-1.232633	0.248573	-0.539483	-0.813368	0.785431	-0.784316	0.673626	1.428269	0.
206534	136246.0	2.237844	-0.551513	-1.426515	-0.924369	-0.401734	-1.438232	-0.119942	-0.449263	-0.717258	0.851668	-0.497634	-0.445482	0.324575	0.

Scaling the test set

We don't fit scaler on the test set. We only transform the test set.

In [27]:
```
# Transform the test set
X_test['Amount'] = scaler.transform(X_test[['Amount']])
X_test.head()
```

Out[27]:

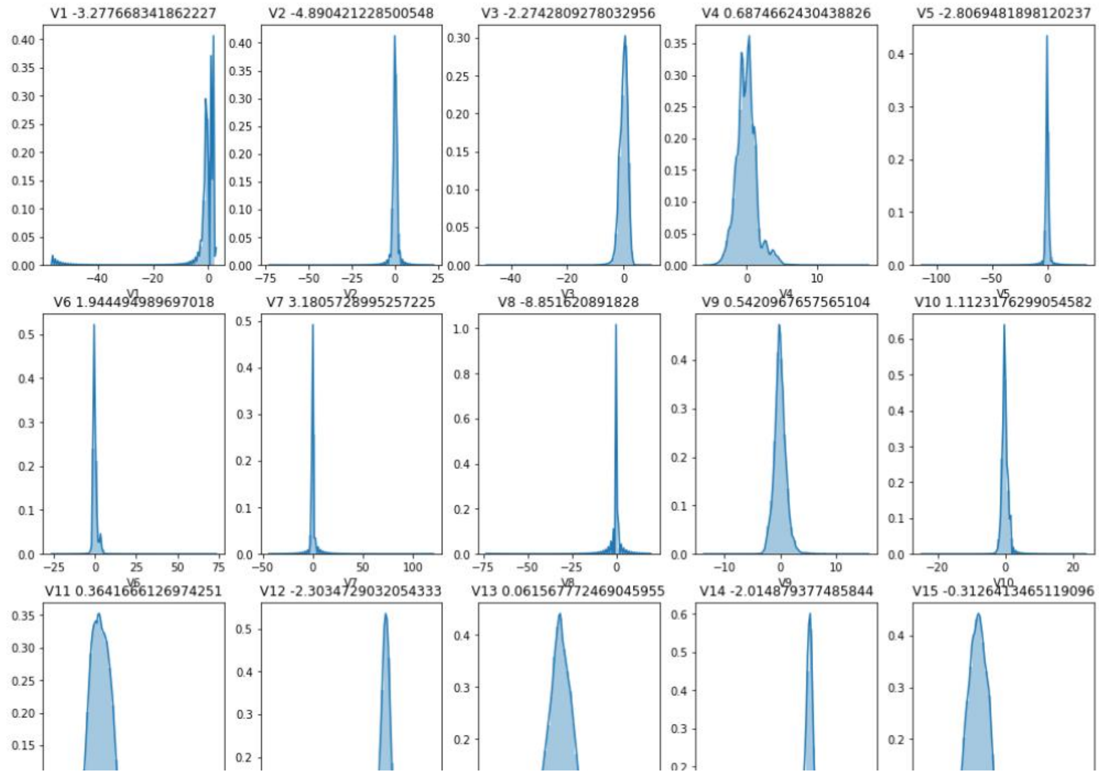
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	
49089	1.229452	-0.235478	-0.627166	0.419877	1.797014	4.069574	-0.896223	1.036103	0.745991	-0.147304	-0.850459	0.397845	-0.259849	-0.277065	-0
154704	2.016893	-0.088751	-2.989257	-0.142575	2.675427	3.332289	-0.652336	0.752811	1.962566	-1.025024	1.126976	-2.418093	1.250341	-0.056209	-0
67247	0.535093	-1.469185	0.868279	0.385462	-1.439135	0.368118	-0.499370	0.303698	1.042073	-0.437209	1.145725	0.907573	-1.095634	-0.055080	-0
251657	2.128486	-0.117215	-1.513910	0.166456	0.359070	-0.540072	0.116023	-0.216140	0.680314	0.079977	-1.705327	-0.127579	-0.207945	0.307878	0
201903	0.558593	1.587908	-2.368767	5.124413	2.171788	-0.500419	1.059829	-0.254233	-1.959060	0.948915	-0.288169	-1.007647	0.470316	-2.771902	0

Checking the Skewness

In [28]:
```
# Listing the columns
cols = X_train.columns
cols
```

Out[28]:
```
Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
       'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',
       'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount'],
      dtype='object')
```

```
In [29]: # Plotting the distribution of the variables (skewness) of all the columns
         k=0
         plt.figure(figsize=(17,28))
         for col in cols :
             k=k+1
             plt.subplot(6, 5,k)
             sns.distplot(X_train[col])
             plt.title(col+' '+str(X_train[col].skew()))
```

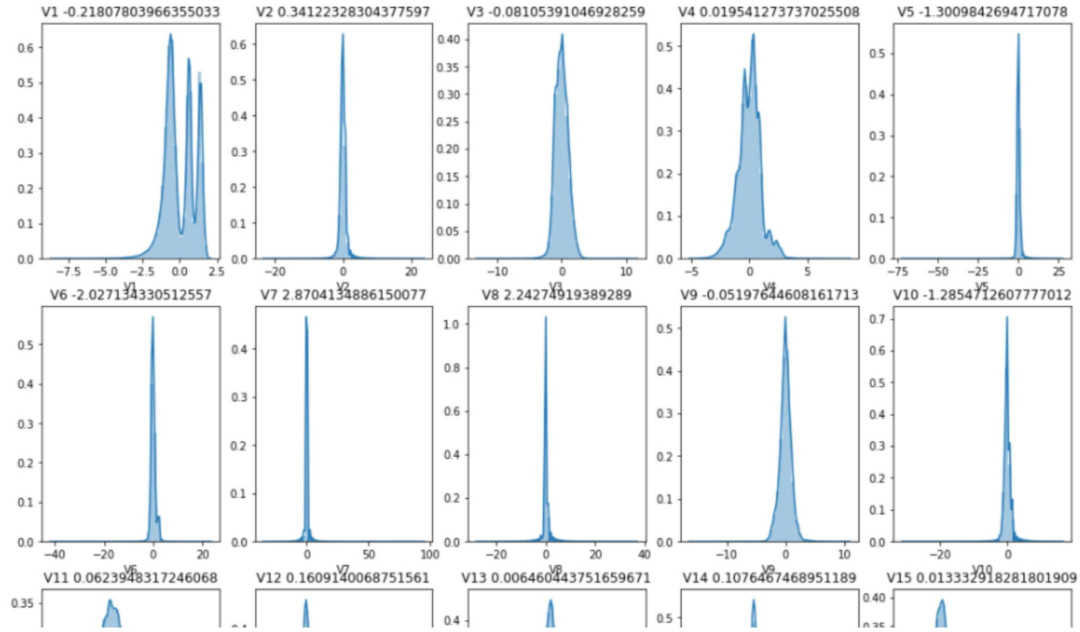


Mitigate skweness with PowerTransformer

In [31]:
```
# Importing PowerTransformer
from sklearn.preprocessing import PowerTransformer
# Instantiate the powertransformer
pt = PowerTransformer(method='yeo-johnson', standardize=True, copy=False)
# Fit and transform the PT on training data
X_train[cols] = pt.fit_transform(X_train)
```

In [32]:
```
# Transform the test set
X_test[cols] = pt.transform(X_test)
```

In [33]:
```
# Plotting the distribution of the variables (skewness) of all the columns
k=0
plt.figure(figsize=(17,28))
for col in cols :
    k=k+1
    plt.subplot(6, 5,k)
    sns.distplot(X_train[col])
    plt.title(col+' '+str(X_train[col].skew()))
```



Logistic regression

In [34]:
```
# Importing scikit logistic regression module
from sklearn.linear_model import LogisticRegression
```

In [35]:
```
# Impoting metrics
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
```

Tuning hyperparameter C

C is the the inverse of regularization strength in Logistic Regression. Higher values of C correspond to less regularization.

In [1]:
```
# Importing libraries for cross validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

In [40]:
```
# Creating KFold object with 5 splits
folds = KFold(n_splits=5, shuffle=True, random_state=4)

# Specify params
params = {"C": [0.01, 0.1, 1, 10, 100, 1000]}

# Specifing score as recall as we are more focused on acheiving the higher sensitivity than the accuracy
model_cv = GridSearchCV(estimator = LogisticRegression(),
                        param_grid = params,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# Fit the model
model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:   43.0s finished
```

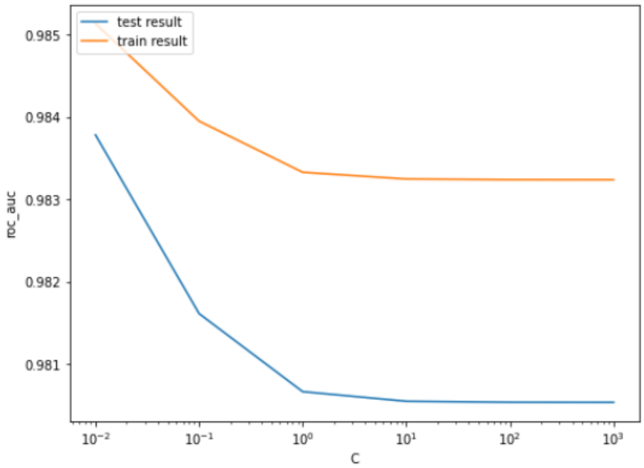
Out[40]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=4, shuffle=True),
             error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.01, 0.1, 1, 10, 100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=1)
```

In [41]:
```
# results of grid search CV
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

Out[41]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	spli
0	0.868854	0.035584	0.023276	0.000451	0.01	{'C': 0.01}	0.986923	0.987267	0.968537	0.982467	
1	1.270447	0.082098	0.024949	0.004554	0.1	{'C': 0.1}	0.986361	0.987820	0.961151	0.980400	
2	1.442430	0.119548	0.023093	0.000241	1	{'C': 1}	0.986199	0.987685	0.958366	0.979495	
3	1.442806	0.093519	0.022923	0.000483	10	{'C': 10}	0.986179	0.987669	0.958011	0.979393	
4	1.434997	0.079551	0.023676	0.001787	100	{'C': 100}	0.986177	0.987666	0.957979	0.979379	
5	1.453797	0.105531	0.022806	0.000277	1000	{'C': 1000}	0.986176	0.987665	0.957976	0.979378	

In [42]:
```
# plot of C versus train and validation scores

plt.figure(figsize=(8, 6))
plt.plot(cv_results['param_C'], cv_results['mean_test_score'])
plt.plot(cv_results['param_C'], cv_results['mean_train_score'])
plt.xlabel('C')
plt.ylabel('roc_auc')
plt.legend(['test result', 'train result'], loc='upper left')
plt.xscale('log')
```



In [43]:
```
# Best score with best C
best_score = model_cv.best_score_
best_C = model_cv.best_params_['C']

print(" The highest test roc_auc is {0} at C = {1}".format(best_score, best_C))
```

```
The highest test roc_auc is 0.9837811907775487 at C = 0.01
```

Logistic regression with optimal C

In [38]:
```
# Instantiate the model with best C
logistic_imb = LogisticRegression(C=0.01)
```

In [39]:
```
# Fit the model on the train set
logistic_imb_model = logistic_imb.fit(X_train, y_train)
```

Prediction on the train set

In [40]:
```
# Predictions on the train set
y_train_pred = logistic_imb_model.predict(X_train)
```

In [41]:
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_train, y_train_pred)
print(confusion)
```

```
[[227427     22]
 [   135    261]]
```

In [42]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [43]:
```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_train, y_train_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

# F1 score
print("F1-Score:-", f1_score(y_train, y_train_pred))
```

```
Accuracy:- 0.9993109350655051
Sensitivity:- 0.6590909090909091
Specificity:- 0.9999032750198946
F1-Score:- 0.7687776141384388
```

In [46]:
```
# classification_report
print(classification_report(y_train, y_train_pred))
```

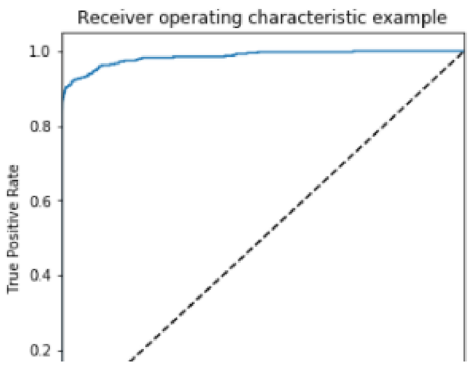
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    227449
           1       0.92      0.66      0.77       396

    accuracy                           1.00    227845
   macro avg       0.96      0.83      0.88    227845
weighted avg       1.00      1.00      1.00    227845
```

ROC on the train set

In [47]:
```
# ROC Curve function

def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return None
```

In [48]:
```
# Predicted probability
y_train_pred_proba = logistic_imb_model.predict_proba(X_train)[:,1]
```

In [49]:
```
# Plot the ROC curve
draw_roc(y_train, y_train_pred_proba)
```



In [50]:
```
# Prediction on the test set
y_test_pred = logistic_imb_model.predict(X_test)
```

In [51]:
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

```
[[56850    16]
 [   42    54]]
```

In [52]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [53]:
```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

# F1 score
print("F1-Score:-", f1_score(y_test, y_test_pred))
```

```
Accuracy:- 0.9989817773252344
Sensitivity:- 0.5625
Specificity:- 0.9997186367952731
F1-Score:- 0.6506024096385543
```

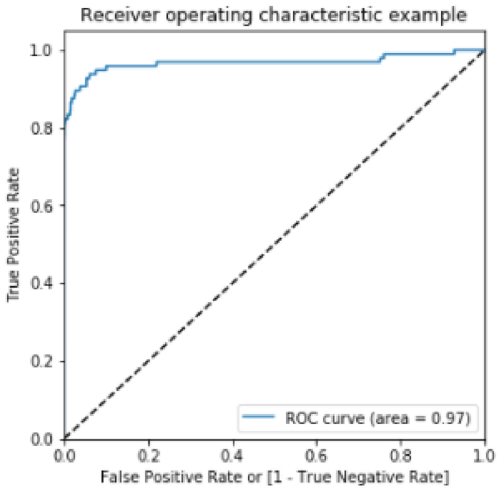
In [54]:
```
# classification_report
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56866
           1       0.77      0.56      0.65        96

    accuracy                           1.00     56962
   macro avg       0.89      0.78      0.83     56962
weighted avg       1.00      1.00      1.00     56962
```

ROC on the test set

In [55]:
```
# Predicted probability
y_test_pred_proba = logistic_imb_model.predict_proba(X_test)[:,1]
```

In [56]:
```
# Plot the ROC curve
draw_roc(y_test, y_test_pred_proba)
```



We can see that we have very good ROC on the test set 0.97, which is almost close to 1.

In [37]:
```python
# Importing XGBoost
from xgboost import XGBClassifier
```

Tuning the hyperparameters

In [65]:
```python
# hyperparameter tuning with XGBoost

# creating a KFold object
folds = 3

# specify range of hyperparameters
param_grid = {'learning_rate': [0.2, 0.6],
             'subsample': [0.3, 0.6, 0.9]}


# specify model
xgb_model = XGBClassifier(max_depth=2, n_estimators=200)

# set up GridSearchCV()
model_cv = GridSearchCV(estimator = xgb_model,
                        param_grid = param_grid,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)


# fit the model
model_cv.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  18 out of  18 | elapsed: 12.6min finished
```

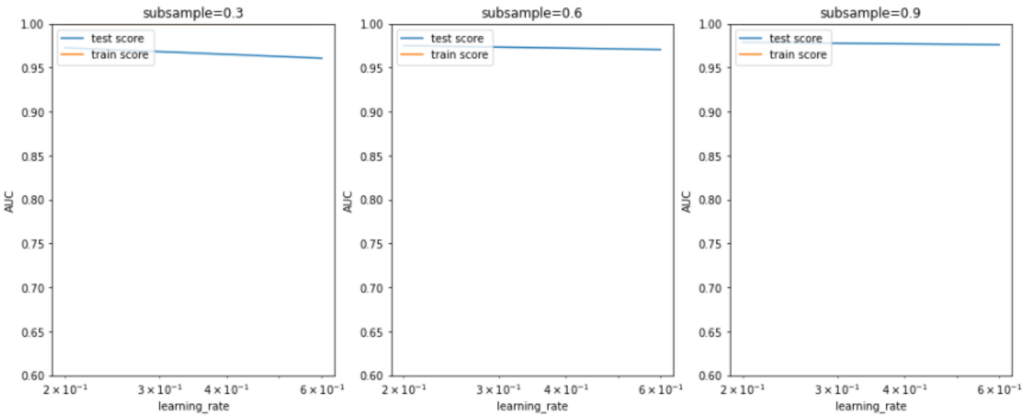
Out[65]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=2, min_child_weight=1,
                                     missing=None, n_estimators=200, n_jobs=1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.2, 0.6],
                         'subsample': [0.3, 0.6, 0.9]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=1)
```

In [66]:
```python
# cv results
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

Out[66]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param_subsample	params	split0_test_score	split1_test_score	s
0	33.505086	0.727619	0.384090	0.004365	0.2	0.3	{'learning_rate': 0.2, 'subsample': 0.3}	0.973826	0.963073	
1	44.019166	0.072776	0.384185	0.006928	0.2	0.6	{'learning_rate': 0.2, 'subsample': 0.6}	0.980747	0.967647	
2	45.915397	0.132965	0.382851	0.006526	0.2	0.9	{'learning_rate': 0.2, 'subsample': 0.9}	0.980109	0.973928	
3	32.986417	0.376595	0.399465	0.001861	0.6	0.3	{'learning_rate': 0.6, 'subsample': 0.3}	0.957307	0.958330	
4	42.858867	0.385860	0.394540	0.003410	0.6	0.6	{'learning_rate': 0.6, 'subsample': 0.6}	0.971740	0.963928	
5	45.059620	0.152377	0.397230	0.002187	0.6	0.9	{'learning_rate': 0.6, 'subsample': 0.9}	0.978537	0.971728	

In [67]:
```
# # plotting
plt.figure(figsize=(16,6))

param_grid = {'learning_rate': [0.2, 0.6],
             'subsample': [0.3, 0.6, 0.9]}


for n, subsample in enumerate(param_grid['subsample']):


    # subplot 1/n
    plt.subplot(1,len(param_grid['subsample']), n+1)
    df = cv_results[cv_results['param_subsample']==subsample]

    plt.plot(df["param_learning_rate"], df["mean_test_score"])
    plt.plot(df["param_learning_rate"], df["mean_train_score"])
    plt.xlabel('learning_rate')
    plt.ylabel('AUC')
    plt.title("subsample={0}".format(subsample))
    plt.ylim([0.60, 1])
    plt.legend(['test score', 'train score'], loc='upper left')
    plt.xscale('log')
```



In [68]:
```
model_cv.best_params_
```
Out[68]:
```
{'learning_rate': 0.2, 'subsample': 0.9}
```
In [38]:
```
# chosen hyperparameters
# 'objective':'binary:logistic' outputs probability rather than label, which we need for calculating auc
params = {'learning_rate': 0.2,
          'max_depth': 2,
          'n_estimators':200,
          'subsample':0.9,
         'objective':'binary:logistic'}

# fit model on training data
xgb_imb_model = XGBClassifier(params = params)
xgb_imb_model.fit(X_train, y_train)
```
Out[38]:
```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='binary:logistic',
              params={'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 200,
                      'objective': 'binary:logistic', 'subsample': 0.9},
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              subsample=1, tree_method=None, validate_parameters=False,
              verbosity=None)
```

Prediction on the train set

In [39]:
```
# Predictions on the train set
y_train_pred = xgb_imb_model.predict(X_train)
```
In [40]:
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_train, y_train_pred)
print(confusion)
```
```
[[227449      0]
 [     0    396]]
```

In [41]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [42]:
```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_train, y_train_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

# F1 score
print("F1-Score:-", f1_score(y_train, y_train_pred))
```

```
Accuracy:- 1.0
Sensitivity:- 1.0
Specificity:- 1.0
F1-Score:- 1.0
```

In [43]:
```
# classification_report
print(classification_report(y_train, y_train_pred))
```

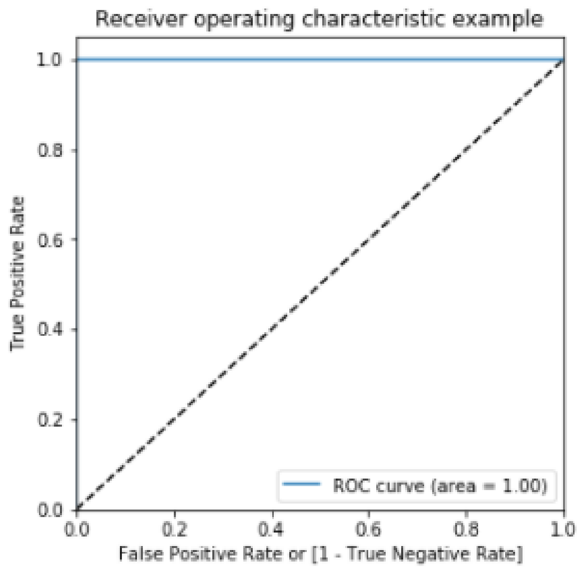
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    227449
           1       1.00      1.00      1.00       396

    accuracy                           1.00    227845
   macro avg       1.00      1.00      1.00    227845
weighted avg       1.00      1.00      1.00    227845
```

In [58]:
```
# Predicted probability
y_train_pred_proba_imb_xgb = xgb_imb_model.predict_proba(X_train)[:,1]
```

In [59]:
```
# roc_auc
auc = metrics.roc_auc_score(y_train, y_train_pred_proba_imb_xgb)
auc
```

Out[59]:
```
1.0
```

In [60]:
```python
# Plot the ROC curve
draw_roc(y_train, y_train_pred_proba_imb_xgb)
```



Prediction on the test set

In [49]:
```python
# Predictions on the test set
y_test_pred = xgb_imb_model.predict(X_test)
```

In [50]:
```python
# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

```
[[56859     7]
 [   24    72]]
```

In [51]:
```python
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [52]:
```python
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

# F1 score
print("F1-Score:-", f1_score(y_test, y_test_pred))
```

```
Accuracy:- 0.9994557775359011
Sensitivity:- 0.75
```

In [53]:
```
# classification_report
print(classification_report(y_test, y_test_pred))
```

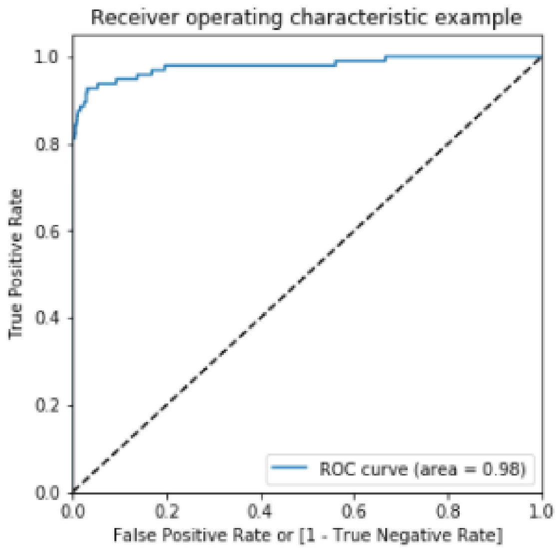
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56866
           1       0.91      0.75      0.82        96

    accuracy                           1.00     56962
   macro avg       0.96      0.87      0.91     56962
weighted avg       1.00      1.00      1.00     56962
```

In [54]:
```
# Predicted probability
y_test_pred_proba = xgb_imb_model.predict_proba(X_test)[:,1]
```

In [55]:
```
# roc_auc
auc = metrics.roc_auc_score(y_test, y_test_pred_proba)
auc
```

Out[55]: `0.9785370798602564`

In [56]:
```
# Plot the ROC curve
draw_roc(y_test, y_test_pred_proba)
```



In [75]:
```
# Importing decision tree classifier
from sklearn.tree import DecisionTreeClassifier
```

In [83]:
```
# Create the parameter grid
param_grid = {
    'max_depth': range(5, 15, 5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
}


# Instantiate the grid search model
dtree = DecisionTreeClassifier()

grid_search = GridSearchCV(estimator = dtree,
                           param_grid = param_grid,
                           scoring= 'roc_auc',
                           cv = 3,
                           verbose = 1)

# Fit the grid search to the data
grid_search.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  24 out of  24 | elapsed:  2.2min finished
```

Out[83]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': range(5, 15, 5),
                         'min_samples_leaf': range(50, 150, 50),
                         'min_samples_split': range(50, 150, 50)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=1)
```

In [84]:

```
# cv results
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

Out[84]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min_samples_leaf	param_min_samples_split	params	split
0	3.762192	0.022569	0.024710	0.000671	5	50	50	{'max_depth': 5, 'min_samples_leaf': 50, 'min_...	
1	3.764455	0.016738	0.024145	0.000872	5	50	100	{'max_depth': 5, 'min_samples_leaf': 50, 'min_...	
2	3.760637	0.012987	0.024381	0.000568	5	100	50	{'max_depth': 5, 'min_samples_leaf': 100, 'min...	
3	3.750272	0.029414	0.024302	0.000159	5	100	100	{'max_depth': 5, 'min_samples_leaf': 100, 'min...	
4	7.425092	0.014732	0.030241	0.003743	10	50	50	{'max_depth': 10, 'min_samples_leaf': 50, 'min...	
5	7.398933	0.015277	0.025900	0.000441	10	50	100	{'max_depth': 10, 'min_samples_leaf': 50, 'min...	
6	7.358769	0.028188	0.026375	0.000218	10	100	50	{'max_depth': 10, 'min_samples_leaf': 100, 'mi...	
7	7.382580	0.027872	0.026896	0.000646	10	100	100	{'max_depth': 10, 'min_samples_leaf': 100, 'mi...	

In [85]:

```
# Printing the optimal sensitivity score and hyperparameters
print("Best roc_auc:-", grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
Best roc_auc:- 0.9382050164508641
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=100, min_samples_split=100,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [76]:

```
# Model with optimal hyperparameters
dt_imb_model = DecisionTreeClassifier(criterion = "gini",
                                      random_state = 100,
                                      max_depth=5,
                                      min_samples_leaf=100,
                                      min_samples_split=100)

dt_imb_model.fit(X_train, y_train)
```

Out[76]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=100, min_samples_split=100,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=100, splitter='best')
```

Prediction on the train set

In [77]:
```
# Predictions on the train set
y_train_pred = dt_imb_model.predict(X_train)
```

In [78]:
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_train, y_train)
print(confusion)
```

```
[[227449      0]
 [     0    396]]
```

In [79]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [80]:
```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_train, y_train_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

# F1 score
print("F1-Score:-", f1_score(y_train, y_train_pred))
```

```
Accuracy:- 0.9991704887094297
Sensitivity:- 1.0
Specificity:- 1.0
F1-Score:- 0.7490039840637449
```

In [81]:
```
# classification_report
print(classification_report(y_train, y_train_pred))
```

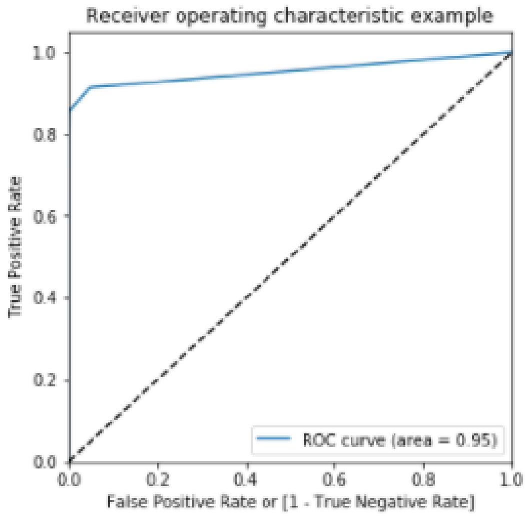
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    227449
           1       0.79      0.71      0.75       396

    accuracy                           1.00    227845
   macro avg       0.89      0.86      0.87    227845
weighted avg       1.00      1.00      1.00    227845
```

In [82]:
```
# Predicted probability
y_train_pred_proba = dt_imb_model.predict_proba(X_train)[:,1]
```

In [83]:
```
# roc_auc
auc = metrics.roc_auc_score(y_train, y_train_pred_proba)
auc
```

Out[83]:
```
0.9534547393930157
```

In [84]:
```
# Plot the ROC curve
draw_roc(y_train, y_train_pred_proba)
```



Prediction on the test set

In [85]:
```
# Predictions on the test set
y_test_pred = dt_imb_model.predict(X_test)
```

In [86]:
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

```
[[56836    30]
 [   40    56]]
```

In [87]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [88]:
```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

# F1 score
print("F1-Score:-", f1_score(y_train, y_train_pred))
```

```
Accuracy:- 0.9987711105649381
Sensitivity:- 0.5833333333333334
Specificity:- 0.9994724439911371
F1-Score:- 0.7490039840637449
```

In [92]:
```
# classification_report
print(classification_report(y_test, y_test_pred))
```

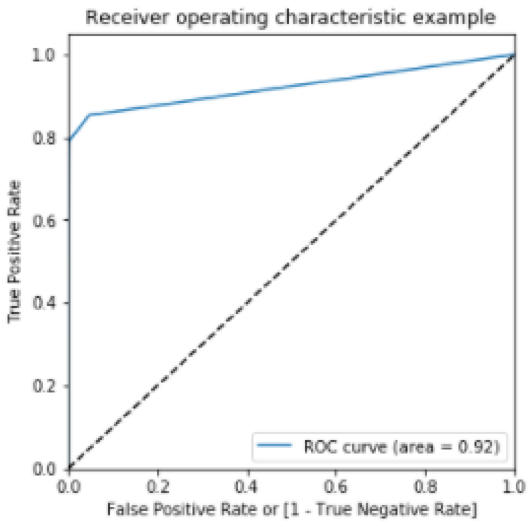
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56866
           1       0.65      0.58      0.62        96

    accuracy                           1.00     56962
   macro avg       0.83      0.79      0.81     56962
weighted avg       1.00      1.00      1.00     56962
```

In [90]:
```
# Predicted probability
y_test_pred_proba = dt_imb_model.predict_proba(X_test)[:,1]
```

In [91]:
```
# roc_auc
auc = metrics.roc_auc_score(y_test, y_test_pred_proba)
auc
```

Out[91]:
```
0.92174979703748
```

In [93]:
```
# Plot the ROC curve
draw_roc(y_test, y_test_pred_proba)
```



Random forest

In [94]:
```
# Importing random forest classifier
from sklearn.ensemble import RandomForestClassifier
```

In [100]:
```
param_grid = {
    'max_depth': range(5,10,5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
    'n_estimators': [100,200,300],
    'max_features': [10, 20]
}
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf,
                           param_grid = param_grid,
                           cv = 2,
                           n_jobs = -1,
                           verbose = 1,
                           return_train_score=True)

# Fit the model
grid_search.fit(X_train, y_train)
```

```
Fitting 2 folds for each of 24 candidates, totalling 48 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 out of  48 | elapsed: 101.0min finished
```

Out[100]:
```
GridSearchCV(cv=2, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': range(5, 10, 5), 'max_features': [10, 20],
                         'min_samples_leaf': range(50, 150, 50),
                         'min_samples_split': range(50, 150, 50),
                         'n_estimators': [100, 200, 300]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=1)
```

In [101]:
```
# printing the optimal accuracy score and hyperparameters
print('We can get accuracy of',grid_search.best_score_,'using',grid_search.best_params_)
```

```
We can get accuracy of 0.9992933790590904 using {'max_depth': 5, 'max_features': 10, 'min_samples_leaf': 50, 'min_samples_spli
t': 50, 'n_estimators': 100}
```

In [95]:
```
# model with the best hyperparameters

rfc_imb_model = RandomForestClassifier(bootstrap=True,
                             max_depth=5,
                             min_samples_leaf=50,
                             min_samples_split=50,
                             max_features=10,
                             n_estimators=100)
```

In [96]:
```
# Fit the model
rfc_imb_model.fit(X_train, y_train)
```

Out[96]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=5, max_features=10,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=50, min_samples_split=50,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Prediction on the train set

In [97]:
```
# Predictions on the train set
y_train_pred = rfc_imb_model.predict(X_train)
```

In [98]:
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_train, y_train)
print(confusion)
```

```
[[227449      0]
 [     0    396]]
```

In [99]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [100]:
```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_train, y_train_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

# F1 score
print("F1-Score:-", f1_score(y_train, y_train_pred))
```

```
Accuracy:- 0.9993460466545239
Sensitivity:- 1.0
Specificity:- 1.0
F1-Score:- 0.7983761840324763
```

In [101]:
```
# classification_report
print(classification_report(y_train, y_train_pred))
```

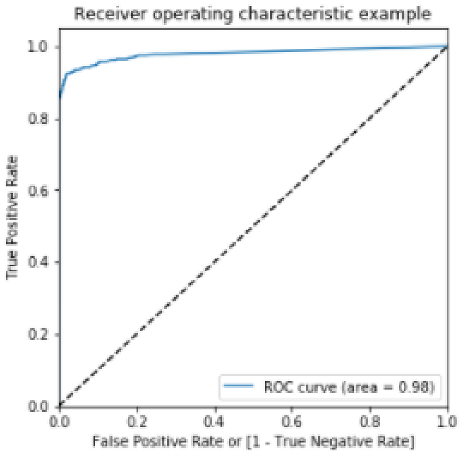
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    227449
           1       0.86      0.74      0.80       396

    accuracy                           1.00    227845
   macro avg       0.93      0.87      0.90    227845
weighted avg       1.00      1.00      1.00    227845
```

In [102]:
```
# Predicted probability
y_train_pred_proba = rfc_imb_model.predict_proba(X_train)[:,1]
```

In [103]:
```
# roc_auc
auc = metrics.roc_auc_score(y_train, y_train_pred_proba)
auc
```

Out[103]:
```
0.9791822295960585
```

In [104]:
```
# Plot the ROC curve
draw_roc(y_train, y_train_pred_proba)
```



Prediction on the test set

In [105]:
```
# Predictions on the test set
y_test_pred = rfc_imb_model.predict(X_test)
```

In [106]:
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

```
[[56841    25]
 [   36    60]]
```

In [107]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [108]:
```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

# F1 score
print("F1-Score:-", f1_score(y_train, y_train_pred))
```

```
Accuracy:- 0.9989291106351603
Sensitivity:- 0.625
Specificity:- 0.9995603699926142
F1-Score:- 0.7983761840324763
```

In [109]:
```
# classification_report
print(classification_report(y_test, y_test_pred))
```

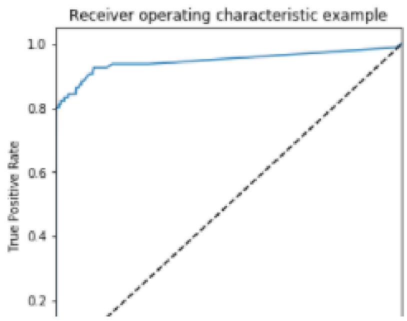
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56866
           1       0.71      0.62      0.66        96

    accuracy                           1.00     56962
   macro avg       0.85      0.81      0.83     56962
weighted avg       1.00      1.00      1.00     56962
```

In [110]:
```
# Predicted probability
y_test_pred_proba = rfc_imb_model.predict_proba(X_test)[:,1]
```

In [111]:
```
# roc_auc
auc = metrics.roc_auc_score(y_test, y_test_pred_proba)
auc
```

Out[111]: 0.9474696179029063

In [112]:
```
# Plot the ROC curve
draw_roc(y_test, y_test_pred_proba)
```



jupyter credit_card_fraud_detection (autosaved)

Logout

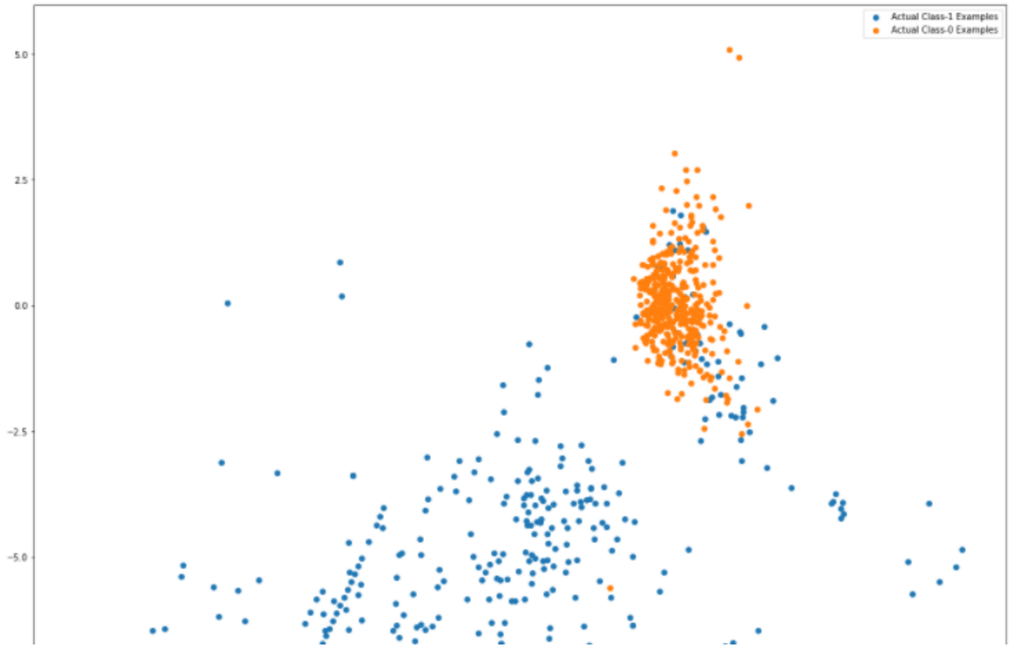
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

```
print('2nd Top var =', var_imp.index(np.sort(xgb_imb_model.feature_importances_)[-2])+1)
print('3rd Top var =', var_imp.index(np.sort(xgb_imb_model.feature_importances_)[-3])+1)
# Variable on Index-16 and Index-13 seems to be the top 2 variables
top_var_index = var_imp.index(np.sort(xgb_imb_model.feature_importances_)[-1])
second_top_var_index = var_imp.index(np.sort(xgb_imb_model.feature_importances_)[-2])

X_train_1 = X_train.to_numpy()[np.where(y_train==1.0)]
X_train_0 = X_train.to_numpy()[np.where(y_train==0.0)]

np.random.shuffle(X_train_0)

import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = [20, 20]

plt.scatter(X_train_1[:, top_var_index], X_train_1[:, second_top_var_index], label='Actual Class-1 Examples')
plt.scatter(X_train_0[:X_train_1.shape[0], top_var_index], X_train_0[:X_train_1.shape[0], second_top_var_index],
            label='Actual Class-0 Examples')
plt.legend()
```

```
Top var = 17
2nd Top var = 14
3rd Top var = 10
```

Out[57]: <matplotlib.legend.Legend at 0x11887c88>



File Edit View Insert Cell Kernel Widgets Help Not Trusted

Run Code

In [116]:
```
# Importing undersampler library
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter
```

In [117]:
```
# instantiating the random undersampler
rus = RandomUnderSampler()
# resampling X, y
X_train_rus, y_train_rus = rus.fit_resample(X_train, y_train)
```

In [118]:
```
# Befor sampling class distribution
print('Before sampling class distribution:-',Counter(y_train))
# new class distribution
print('New class distribution:-',Counter(y_train_rus))
```

```
Before sampling class distribution:- Counter({0: 227449, 1: 396})
New class distribution:- Counter({0: 396, 1: 396})
```

Model building on balanced data with Undersampling

Logistic Regression

In [50]:
```
# Creating KFold object with 5 splits
folds = KFold(n_splits=5, shuffle=True, random_state=4)

# Specify params
params = {"C": [0.01, 0.1, 1, 10, 100, 1000]}

# Specifing score as roc-auc
model_cv = GridSearchCV(estimator = LogisticRegression(),
                        param_grid = params,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# Fit the model
model_cv.fit(X_train_rus, y_train_rus)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    0.7s finished
```

Out[50]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=4, shuffle=True),
             error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.01, 0.1, 1, 10, 100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=1)
```

jupyter credit_card_fraud_detection (autosaved)

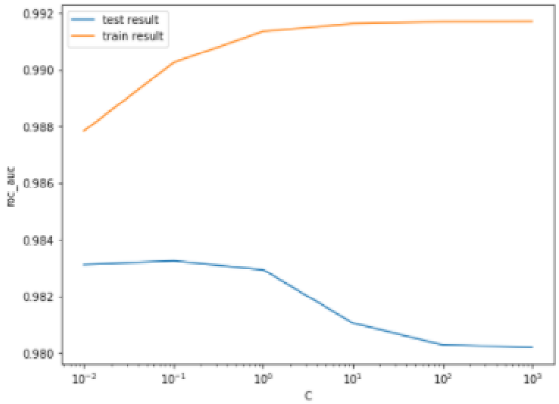
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

In [51]:
```
# results of grid search CV
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

Out[51]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	spli
0	0.016201	0.009065	0.0040	1.095540e-03	0.01	{'C': 0.01}	0.983943	0.995410	0.972276	0.976110	
1	0.016801	0.002136	0.0042	7.483665e-04	0.1	{'C': 0.1}	0.981240	0.995568	0.976122	0.974186	
2	0.026201	0.004118	0.0040	1.095453e-03	1	{'C': 1}	0.981081	0.994302	0.978365	0.971621	
3	0.020201	0.002786	0.0030	9.536743e-08	10	{'C': 10}	0.975199	0.994777	0.978846	0.966330	
4	0.020801	0.002561	0.0030	6.324097e-04	100	{'C': 100}	0.972496	0.994619	0.978846	0.965368	
5	0.021601	0.001497	0.0026	4.898624e-04	1000	{'C': 1000}	0.972178	0.994619	0.978686	0.965368	

In [52]:
```
# plot of C versus train and validation scores

plt.figure(figsize=(8, 6))
plt.plot(cv_results['param_C'], cv_results['mean_test_score'])
plt.plot(cv_results['param_C'], cv_results['mean_train_score'])
plt.xlabel('C')
plt.ylabel('roc_auc')
plt.legend(['test result', 'train result'], loc='upper left')
plt.xscale('log')
```



In [53]:
```
# Best score with best C
best_score = model_cv.best_score_
best_C = model_cv.best_params_['C']

print(" The highest test roc_auc is {0} at C = {1}".format(best_score, best_C))
```

```
 The highest test roc_auc is 0.9832637280039689 at C = 0.1
```

jupyter credit_card_fraud_detection (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [119]:
```
# Instantiate the model with best C
logistic_bal_rus = LogisticRegression(C=0.1)
```

In [120]:
```
# Fit the model on the train set
logistic_bal_rus_model = logistic_bal_rus.fit(X_train_rus, y_train_rus)
```

Prediction on the train set

In [121]:
```
# Predictions on the train set
y_train_pred = logistic_bal_rus_model.predict(X_train_rus)
```

In [122]:
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_train_rus, y_train_pred)
print(confusion)
```

```
[[391   5]
 [ 32 364]]
```

In [123]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [124]:
```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_train_rus, y_train_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

# F1 score
print("F1-Score:-", f1_score(y_train_rus, y_train_pred))
```

```
Accuracy:- 0.9532828282828283
Sensitivity:- 0.9191919191919192
Specificity:- 0.9873737373737373
F1-Score:- 0.9516339869281046
```

In [125]:
```
# classification_report
print(classification_report(y_train_rus, y_train_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95       396
           1       0.99      0.92      0.95       396

    accuracy                           0.95       792
   macro avg       0.96      0.95      0.95       792
weighted avg       0.96      0.95      0.95       792
```

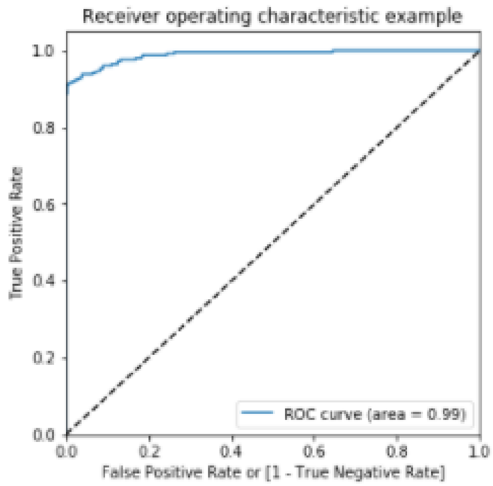
jupyter credit_card_fraud_detection (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [127]:
```python
# roc_auc
auc = metrics.roc_auc_score(y_train_rus, y_train_pred_proba)
auc
```

Out[127]: 0.9892230384654627

In [128]:
```python
# Plot the ROC curve
draw_roc(y_train_rus, y_train_pred_proba)
```



Prediction on the test set

In [129]:
```python
# Prediction on the test set
y_test_pred = logistic_bal_rus_model.predict(X_test)
```

In [130]:
```python
# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

```
[[55658  1208]
 [   13    83]]
```

In [131]:
```python
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [132]:
```python
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))
```

```
Accuracy:- 0.9785646571398476
Sensitivity:- 0.8645833333333334
Specificity:- 0.978757078043119
```

File Edit View Insert Cell Kernel Widgets Help

Run Code

In [133]:
```
# classification_report
print(classification_report(y_test, y_test_pred))
```

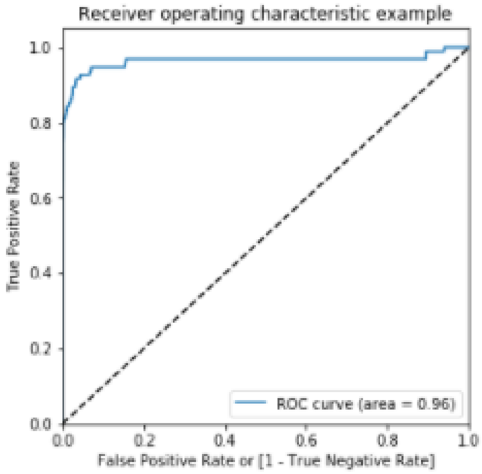
```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56866
           1       0.06      0.86      0.12        96

    accuracy                           0.98     56962
   macro avg       0.53      0.92      0.55     56962
weighted avg       1.00      0.98      0.99     56962
```

In [134]:
```
# Predicted probability
y_test_pred_proba = logistic_bal_rus_model.predict_proba(X_test)[:,1]
```

In [135]:
```
# roc_auc
auc = metrics.roc_auc_score(y_test, y_test_pred_proba)
auc
```

Out[135]: 0.9639748854031114

In [136]:
```
# Plot the ROC curve
draw_roc(y_test, y_test_pred_proba)
```



Model summary

- Train set
  - Accuracy = 0.95
  - Sensitivity = 0.92
  - Specificity = 0.98
  - ROC = 0.99
- Test set
  - Accuracy = 0.97
  - Sensitivity = 0.86
  - Specificity = 0.97
  - ROC = 0.96

XGBoost

In [73]:
```
# hyperparameter tuning with XGBoost

# creating a KFold object
folds = 3

# specify range of hyperparameters
param_grid = {'learning_rate': [0.2, 0.6],
             'subsample': [0.3, 0.6, 0.9]}


# specify model
xgb_model = XGBClassifier(max_depth=2, n_estimators=200)

# set up GridSearchCV()
model_cv = GridSearchCV(estimator = xgb_model,
                        param_grid = param_grid,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# fit the model
model_cv.fit(X_train_rus, y_train_rus)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  18 out of  18 | elapsed:    3.9s finished
```

Out[73]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=2, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estimato...
                                     objective='binary:logistic',
                                     random_state=None, reg_alpha=None,
                                     reg_lambda=None, scale_pos_weight=None,
                                     subsample=None, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.2, 0.6],
                         'subsample': [0.3, 0.6, 0.9]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=1)
```

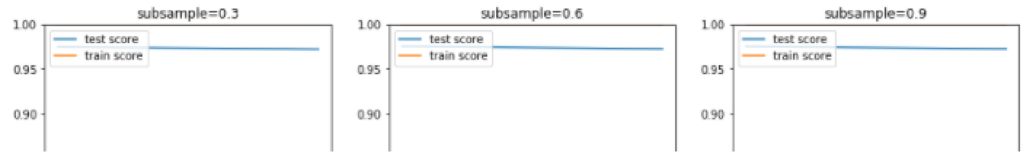
e Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

In [74]:
```
# cv results
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

Out[74]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param_subsample	params	split0_test_score	split1_test_score	spl
0	0.210345	0.069442	0.016668	0.014384	0.2	0.3	{'learning_rate': 0.2, 'subsample': 0.3}	0.967172	0.973714	
1	0.168343	0.003300	0.008334	0.000471	0.2	0.6	{'learning_rate': 0.2, 'subsample': 0.6}	0.969295	0.974518	
2	0.247348	0.025370	0.008334	0.000471	0.2	0.9	{'learning_rate': 0.2, 'subsample': 0.9}	0.969238	0.974690	
3	0.247347	0.116300	0.011667	0.005437	0.6	0.3	{'learning_rate': 0.6, 'subsample': 0.3}	0.967172	0.969754	
4	0.188344	0.026248	0.007001	0.000817	0.6	0.6	{'learning_rate': 0.6, 'subsample': 0.6}	0.964073	0.976297	
5	0.171343	0.023115	0.008334	0.000471	0.6	0.9	{'learning_rate': 0.6, 'subsample': 0.9}	0.970500	0.968951	

In [75]:
```
# # plotting
plt.figure(figsize=(16,6))

param_grid = {'learning_rate': [0.2, 0.6],
             'subsample': [0.3, 0.6, 0.9]}


for n, subsample in enumerate(param_grid['subsample']):


    # subplot 1/n
    plt.subplot(1,len(param_grid['subsample']), n+1)
    df = cv_results[cv_results['param_subsample']==subsample]

    plt.plot(df["param_learning_rate"], df["mean_test_score"])
    plt.plot(df["param_learning_rate"], df["mean_train_score"])
    plt.xlabel('learning_rate')
    plt.ylabel('AUC')
    plt.title("subsample={0}".format(subsample))
    plt.ylim([0.60, 1])
    plt.legend(['test score', 'train score'], loc='upper left')
    plt.xscale('log')
```



In [145]:
```
# Creating KFold object with 5 splits
folds = KFold(n_splits=5, shuffle=True, random_state=4)

# Specify params
params = {"C": [0.01, 0.1, 1, 10, 100, 1000]}

# Specifing score as roc-auc
model_cv = GridSearchCV(estimator = LogisticRegression(),
                        param_grid = params,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# Fit the model
model_cv.fit(X_train_ros, y_train_ros)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:  1.4min finished
```

Out[145]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=4, shuffle=True),
             error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.01, 0.1, 1, 10, 100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=1)
```

In [146]:
```
# results of grid search CV
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

Out[146]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	spli
0	2.392937	0.133817	0.052003	0.003847	0.01	{'C': 0.01}	0.988802	0.988039	0.988728	0.988207	
1	2.366276	0.096595	0.048522	0.003303	0.1	{'C': 0.1}	0.988821	0.988048	0.988751	0.988206	
2	2.725587	0.393503	0.056963	0.008990	1	{'C': 1}	0.988819	0.988049	0.988751	0.988202	
3	2.949569	0.306817	0.061003	0.008391	10	{'C': 10}	0.988820	0.988049	0.988751	0.988202	
4	2.584676	0.096526	0.056722	0.007632	100	{'C': 100}	0.988820	0.988050	0.988751	0.988201	
5	2.384325	0.060643	0.050203	0.003371	1000	{'C': 1000}	0.988820	0.988050	0.988751	0.988201	