In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

DATA COLLECTION AND PREPARATION :-

IMPORTING THE REQUIRED LIBRARIES :-

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import tensorflow as tf
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score,classification_report,confusic
import pickle
import warnings
warnings.filterwarnings('ignore')
```

COLLECT AND READ THE DATASET :-

In [ ]:
```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/flightdata.csv')
```

In [ ]:
```
pd.set_option('display.max_rows',100)
pd.set_option('display.max_columns',1000)
pd.set_option('display.width',1000)
```

In [ ]:
```
df.head()
```

Out[ ]:

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	TAIL_NU
0	2016	1	1	1	5	DL	N836I
1	2016	1	1	1	5	DL	N964I
2	2016	1	1	1	5	DL	N813I
3	2016	1	1	1	5	DL	N587N
4	2016	1	1	1	5	DL	N836I

DESCRIPTIVE STATISTICAL :-

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11231 entries, 0 to 11230
Data columns (total 26 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   YEAR                 11231 non-null  int64  
 1   QUARTER              11231 non-null  int64  
 2   MONTH                11231 non-null  int64  
 3   DAY_OF_MONTH         11231 non-null  int64  
 4   DAY_OF_WEEK          11231 non-null  int64  
 5   UNIQUE_CARRIER       11231 non-null  object 
 6   TAIL_NUM             11231 non-null  object 
 7   FL_NUM               11231 non-null  int64  
 8   ORIGIN_AIRPORT_ID    11231 non-null  int64  
 9   ORIGIN               11231 non-null  object 
 10  DEST_AIRPORT_ID      11231 non-null  int64  
 11  DEST                 11231 non-null  object 
 12  CRS_DEP_TIME         11231 non-null  int64  
 13  DEP_TIME             11124 non-null  float64
 14  DEP_DELAY            11124 non-null  float64
 15  DEP_DEL15            11124 non-null  float64
 16  CRS_ARR_TIME         11231 non-null  int64  
 17  ARR_TIME             11116 non-null  float64
 18  ARR_DELAY            11043 non-null  float64
 19  ARR_DEL15            11043 non-null  float64
 20  CANCELLED            11231 non-null  float64
 21  DIVERTED             11231 non-null  float64
 22  CRS_ELAPSED_TIME     11231 non-null  float64
 23  ACTUAL_ELAPSED_TIME  11043 non-null  float64
 24  DISTANCE             11231 non-null  float64
 25  Unnamed: 25          0 non-null      float64
dtypes: float64(12), int64(10), object(4)
memory usage: 2.2+ MB
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
YEAR                      0
QUARTER                   0
MONTH                     0
DAY_OF_MONTH              0
DAY_OF_WEEK               0
UNIQUE_CARRIER            0
TAIL_NUM                  0
FL_NUM                    0
ORIGIN_AIRPORT_ID         0
ORIGIN                    0
DEST_AIRPORT_ID           0
DEST                      0
CRS_DEP_TIME              0
DEP_TIME                107
DEP_DELAY               107
DEP_DEL15               107
CRS_ARR_TIME              0
ARR_TIME                115
ARR_DELAY               188
ARR_DEL15               188
CANCELLED                 0
DIVERTED                  0
CRS_ELAPSED_TIME          0
ACTUAL_ELAPSED_TIME     188
DISTANCE                  0
Unnamed: 25           11231
dtype: int64
```

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

Out[ ]:

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_NUM
count	11231.0	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000
mean	2016.0	2.544475	6.628973	15.790758	3.960199	1334.325617
std	0.0	1.090701	3.354678	8.782056	1.995257	811.875227
min	2016.0	1.000000	1.000000	1.000000	1.000000	7.000000
25%	2016.0	2.000000	4.000000	8.000000	2.000000	624.000000
50%	2016.0	3.000000	7.000000	16.000000	4.000000	1267.000000
75%	2016.0	3.000000	9.000000	23.000000	6.000000	2032.000000
max	2016.0	4.000000	12.000000	31.000000	7.000000	2853.000000

So , At last the columns that can be useful for prediction are...

In [ ]:
```
df = df[['FL_NUM','MONTH','DAY_OF_MONTH','DAY_OF_WEEK','ORIGIN','DEST','C
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
FL_NUM            0
MONTH             0
DAY_OF_MONTH      0
DAY_OF_WEEK       0
ORIGIN            0
DEST              0
CRS_ARR_TIME      0
DEP_DEL15       107
ARR_DEL15       188
DEP_DELAY       107
dtype: int64
```

HANDLING MISSING VALUES :-

In [ ]:
```
df['DEP_DEL15'].mode()
```

Out[ ]:
```
0    0.0
Name: DEP_DEL15, dtype: float64
```

In [ ]:
```
df['ARR_DEL15'].mode()
```

Out[ ]:
```
0    0.0
Name: ARR_DEL15, dtype: float64
```

In [ ]:
```
df['DEP_DEL15'].fillna(0.0,inplace=True)
df['ARR_DEL15'].fillna(0.0,inplace=True)
df['DEP_DELAY'].fillna(df['DEP_DELAY'].median(),inplace=True)
```

```
<ipython-input-64-148c2a153b28>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['DEP_DEL15'].fillna(0.0,inplace=True)
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
FL_NUM          0
MONTH           0
DAY_OF_MONTH    0
DAY_OF_WEEK     0
ORIGIN          0
DEST            0
CRS_ARR_TIME    0
DEP_DEL15       0
ARR_DEL15       0
DEP_DELAY       0
dtype: int64
```

In [ ]:
```
df.to_csv('df_reduced.csv')
df.head()
```

Out[ ]:

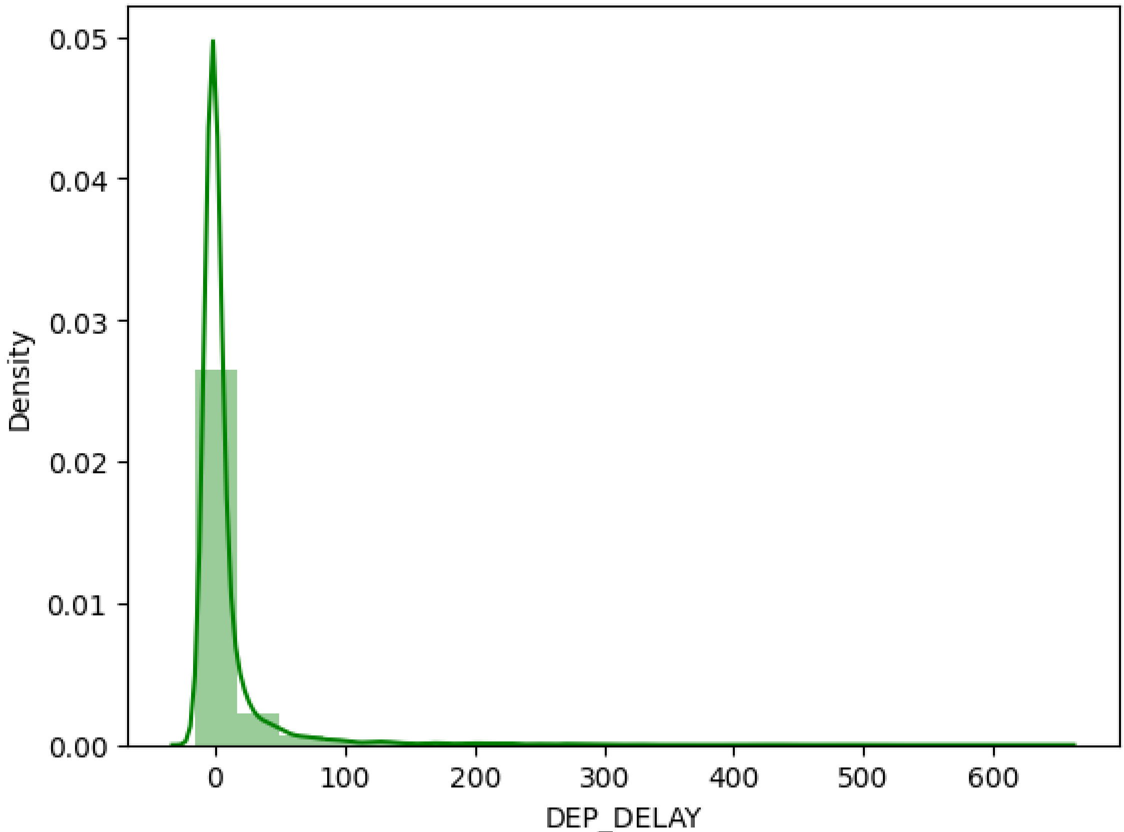
	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DE
0	1399	1	1	5	ATL	SEA	2143	
1	1476	1	1	5	DTW	MSP	1435	
2	1597	1	1	5	ATL	SEA	1215	
3	1768	1	1	5	SEA	MSP	1335	
4	1823	1	1	5	SEA	DTW	607	

EDA : EXPLORATORY DATA ANALYSIS :-

UNIVARIATE ANALYSIS :-

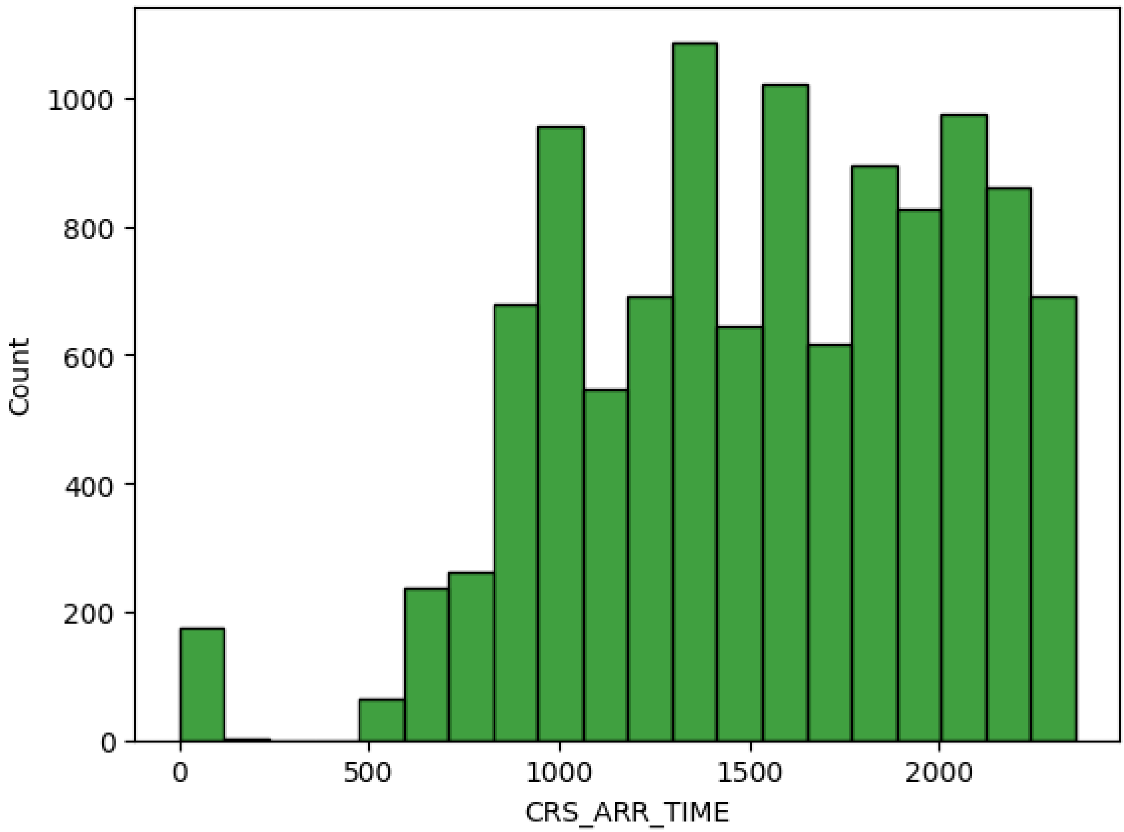
In [ ]:
```
sns.distplot(df['DEP_DELAY'],color='green',bins=20)
```

Out[ ]: <Axes: xlabel='DEP_DELAY', ylabel='Density'>



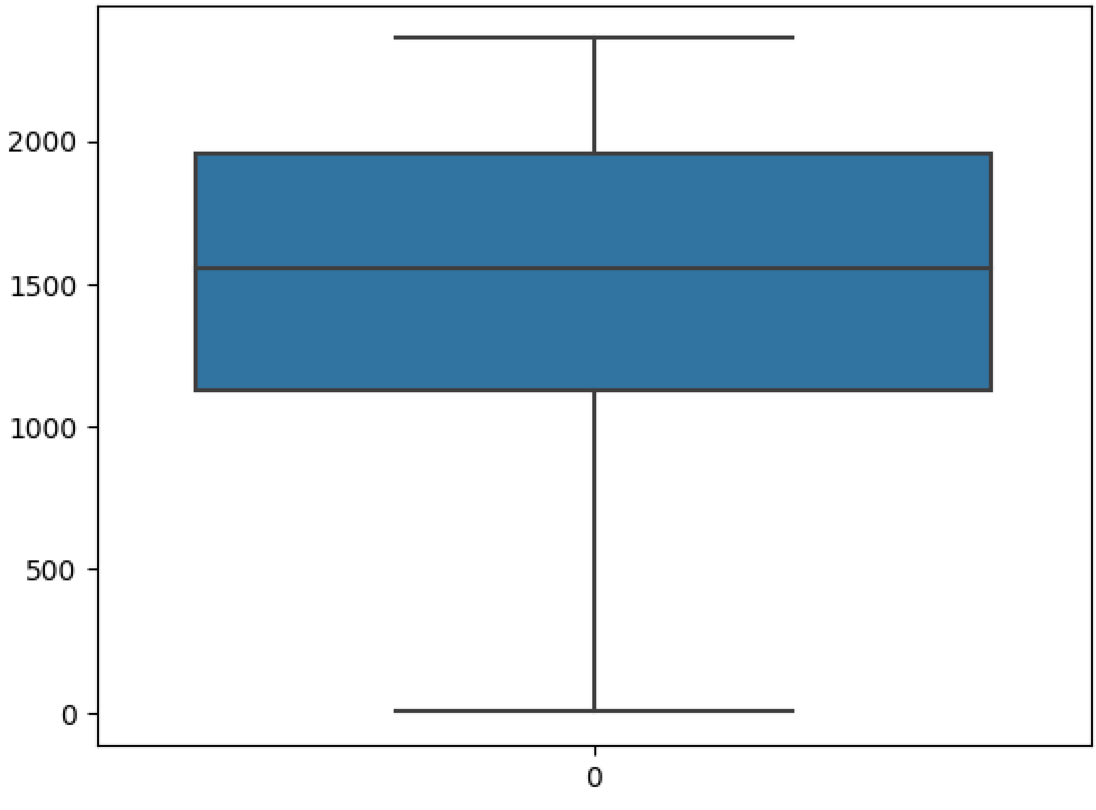
In [ ]:
```
sns.histplot(df['CRS_ARR_TIME'],color='green',bins=20)
```

Out[ ]: <Axes: xlabel='CRS_ARR_TIME', ylabel='Count'>



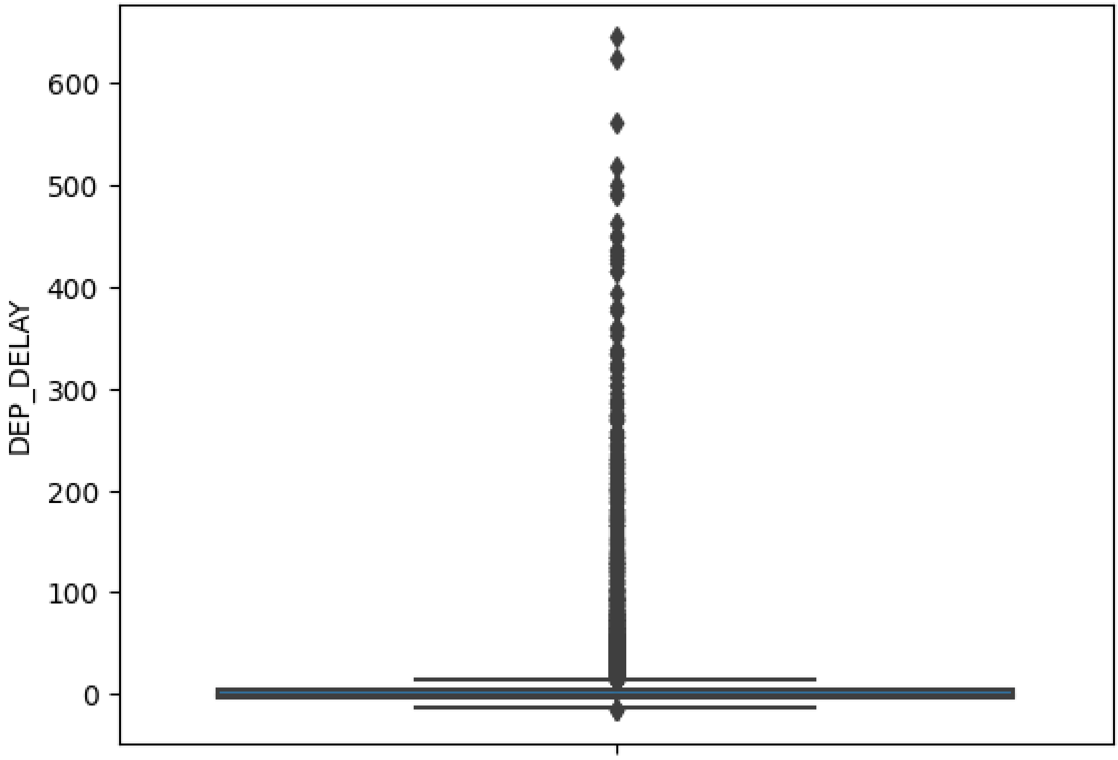
In [ ]:
```
sns.boxplot(df['CRS_ARR_TIME'])
```

Out[ ]: <Axes: >

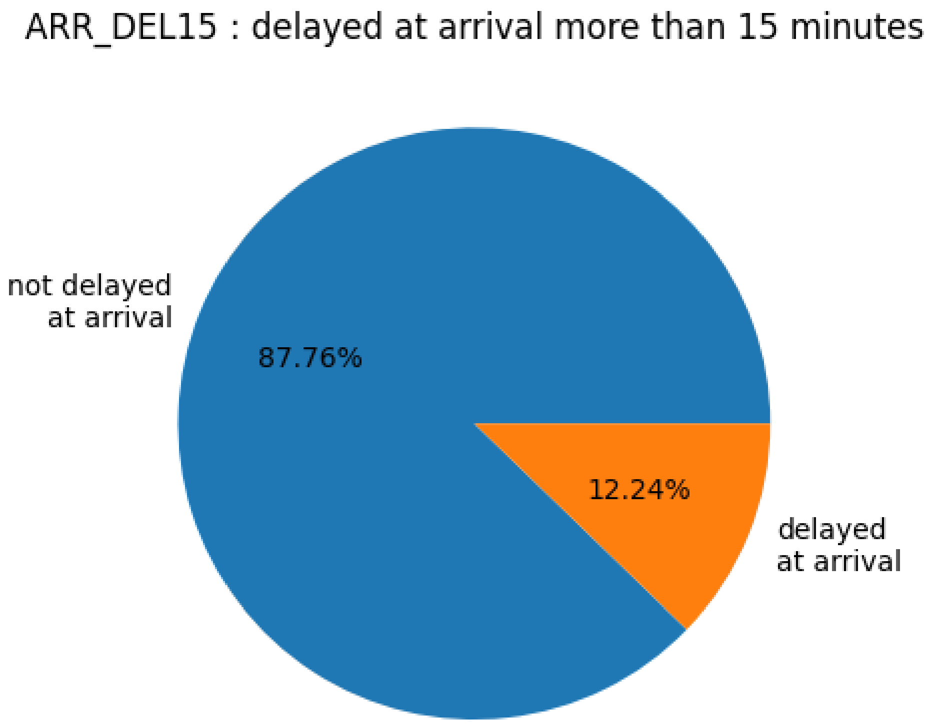


In [ ]:
```
sns.boxplot(df,y='DEP_DELAY')
```

Out[ ]: <Axes: ylabel='DEP_DELAY'>



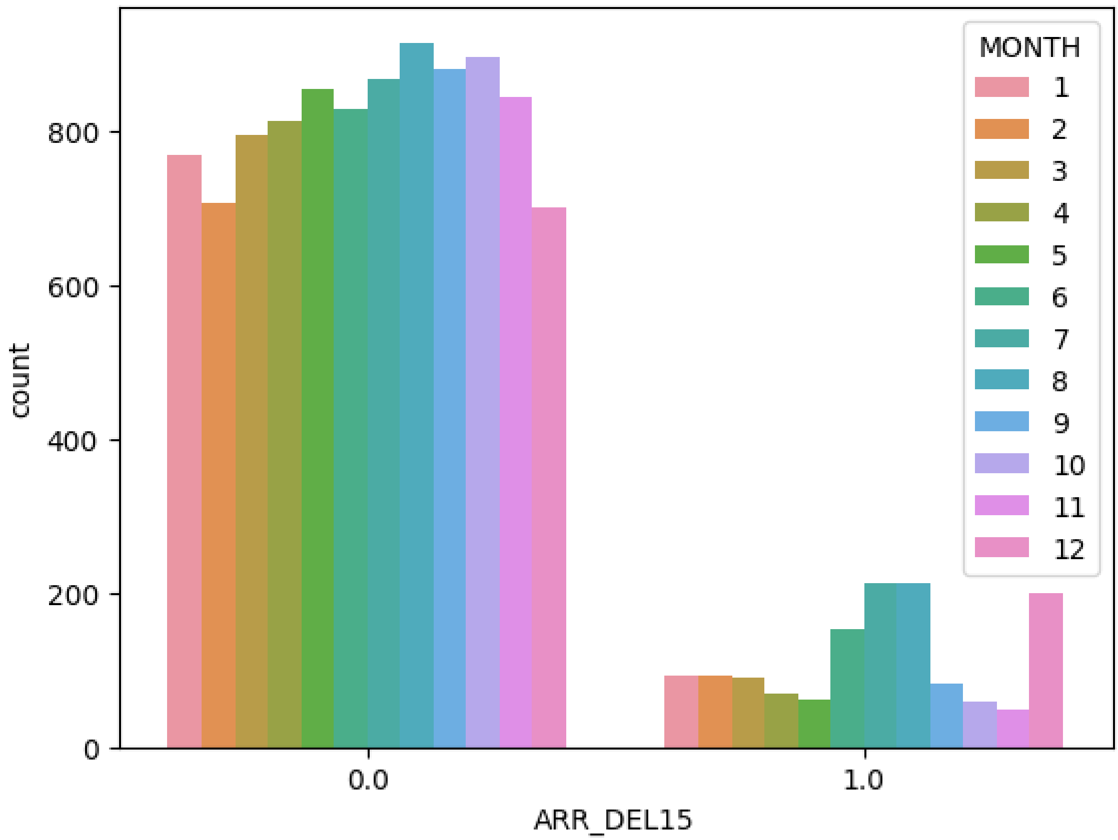
In [ ]:
```
plt.title('ARR_DEL15 : delayed at arrival more than 15 minutes')
plt.pie(df.ARR_DEL15.value_counts(),labels = ['delayed\nat arrival' if x
plt.show()
```



BIVARIATE ANALYSIS :-

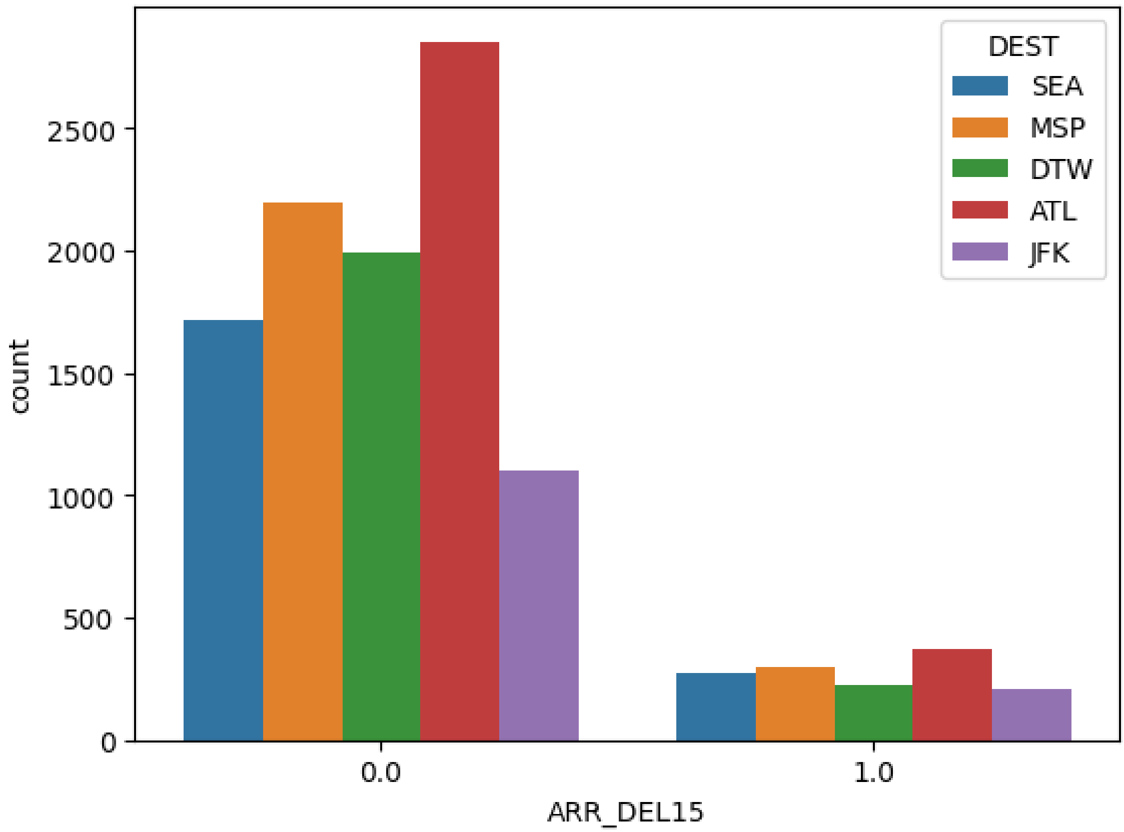
In [ ]:
```
sns.countplot(data = df,x='ARR_DEL15',hue='MONTH')
```

Out[ ]: <Axes: xlabel='ARR_DEL15', ylabel='count'>



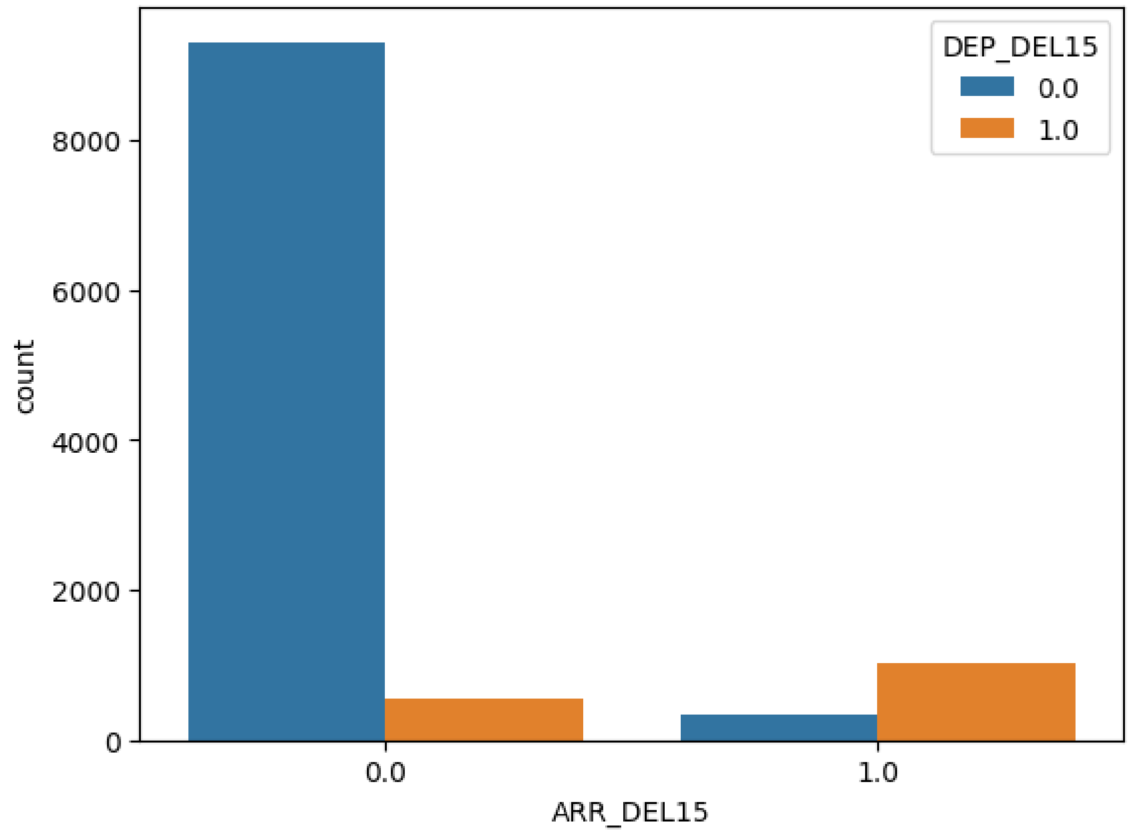

```
In [ ]: sns.countplot(data = df,x='ARR_DEL15',hue='DEST')
```

```
Out[ ]: <Axes: xlabel='ARR_DEL15', ylabel='count'>
```



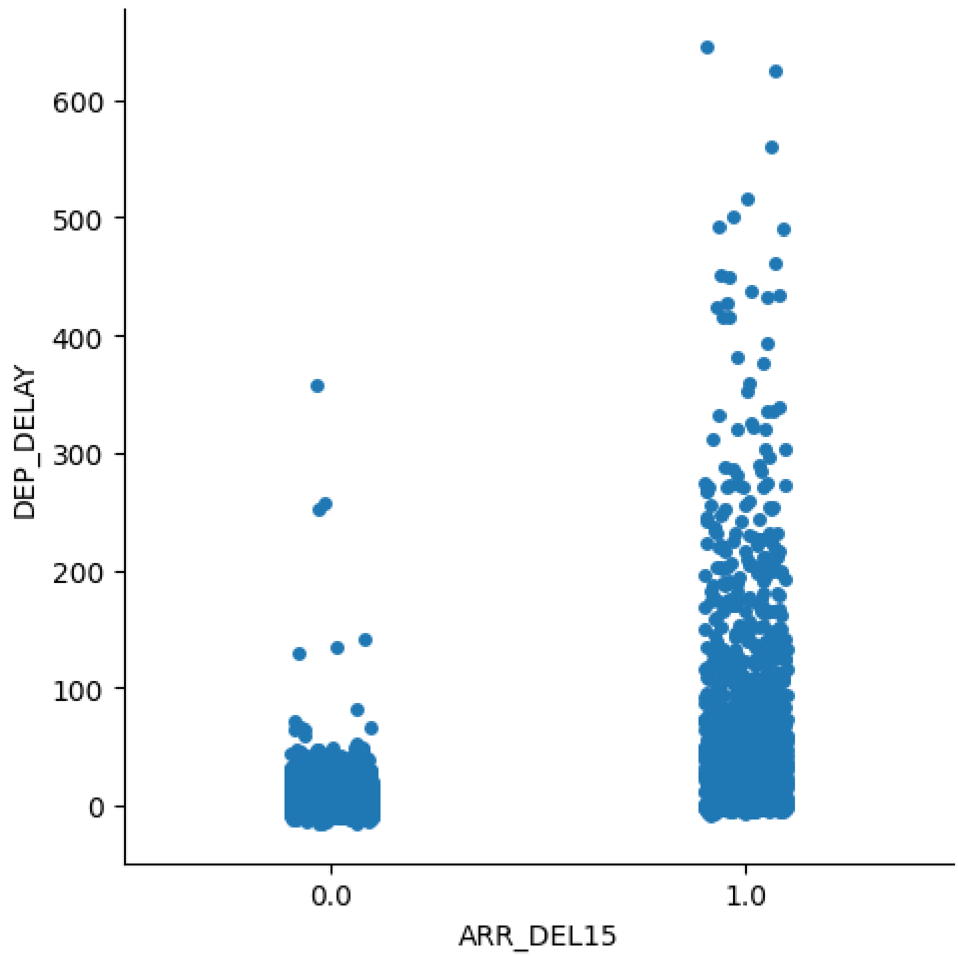

```
In [ ]: sns.countplot(data = df,x='ARR_DEL15',hue='DEP_DEL15')
```

```
Out[ ]: <Axes: xlabel='ARR_DEL15', ylabel='count'>
```



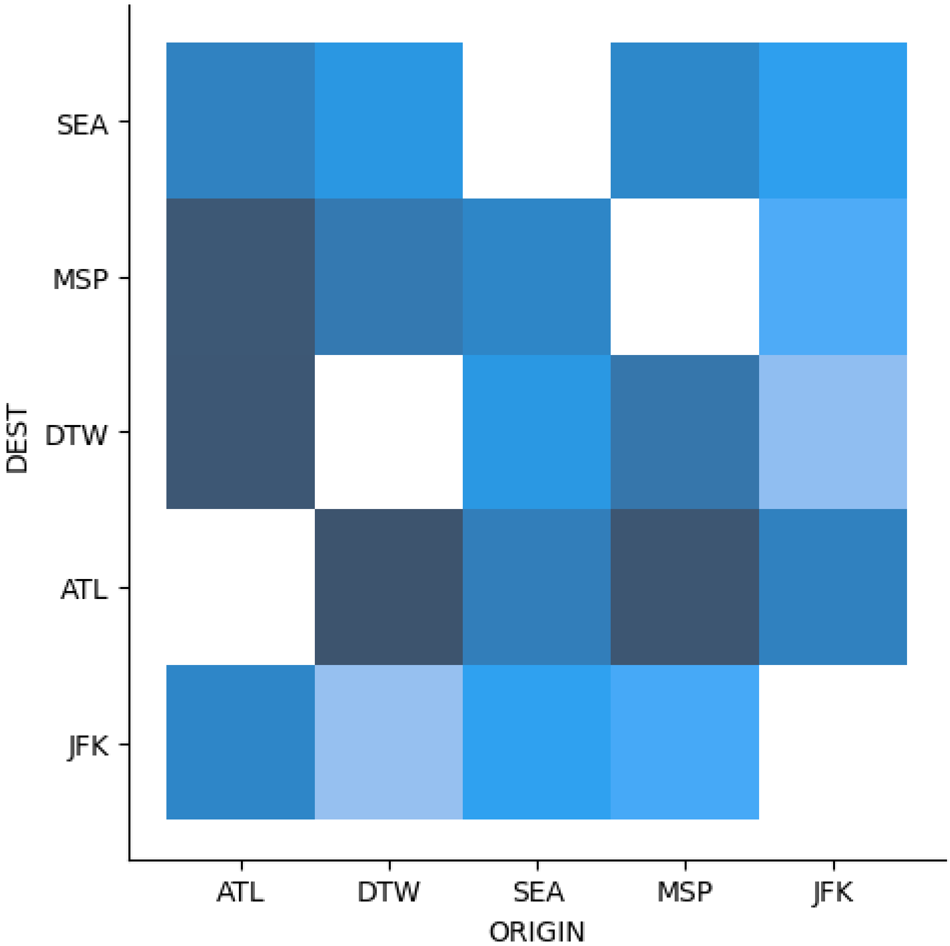
In [ ]:
```
sns.catplot(x='ARR_DEL15',y='DEP_DELAY',data=df)
```

Out[ ]: <seaborn.axisgrid.FacetGrid at 0x7f81cdc61a00>



In [ ]:
```
sns.displot(df,x='ORIGIN',y='DEST')
```

Out[ ]: <seaborn.axisgrid.FacetGrid at 0x7f81c986bc40>

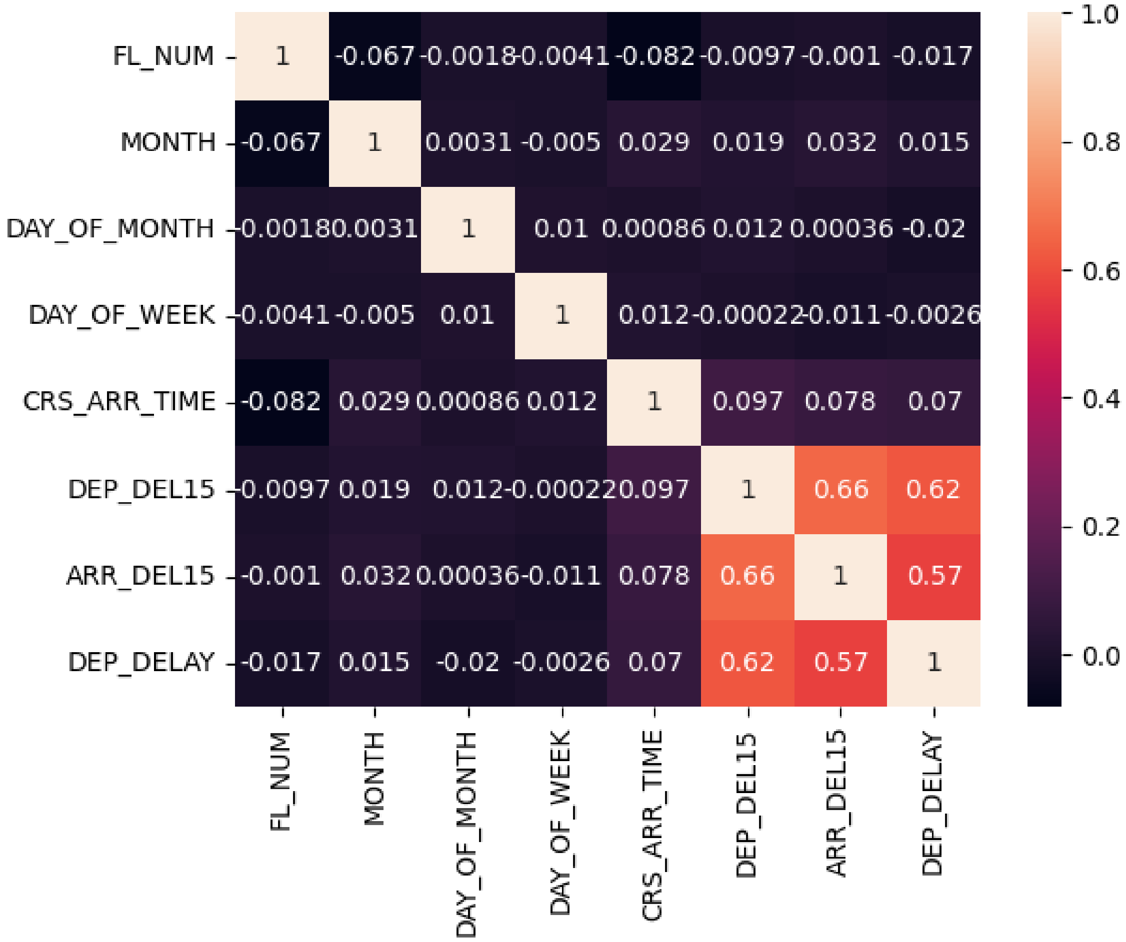


MULTIVARIATE ANALYSIS :-

In [ ]:
```
sns.heatmap(df.corr(),annot=True)
```

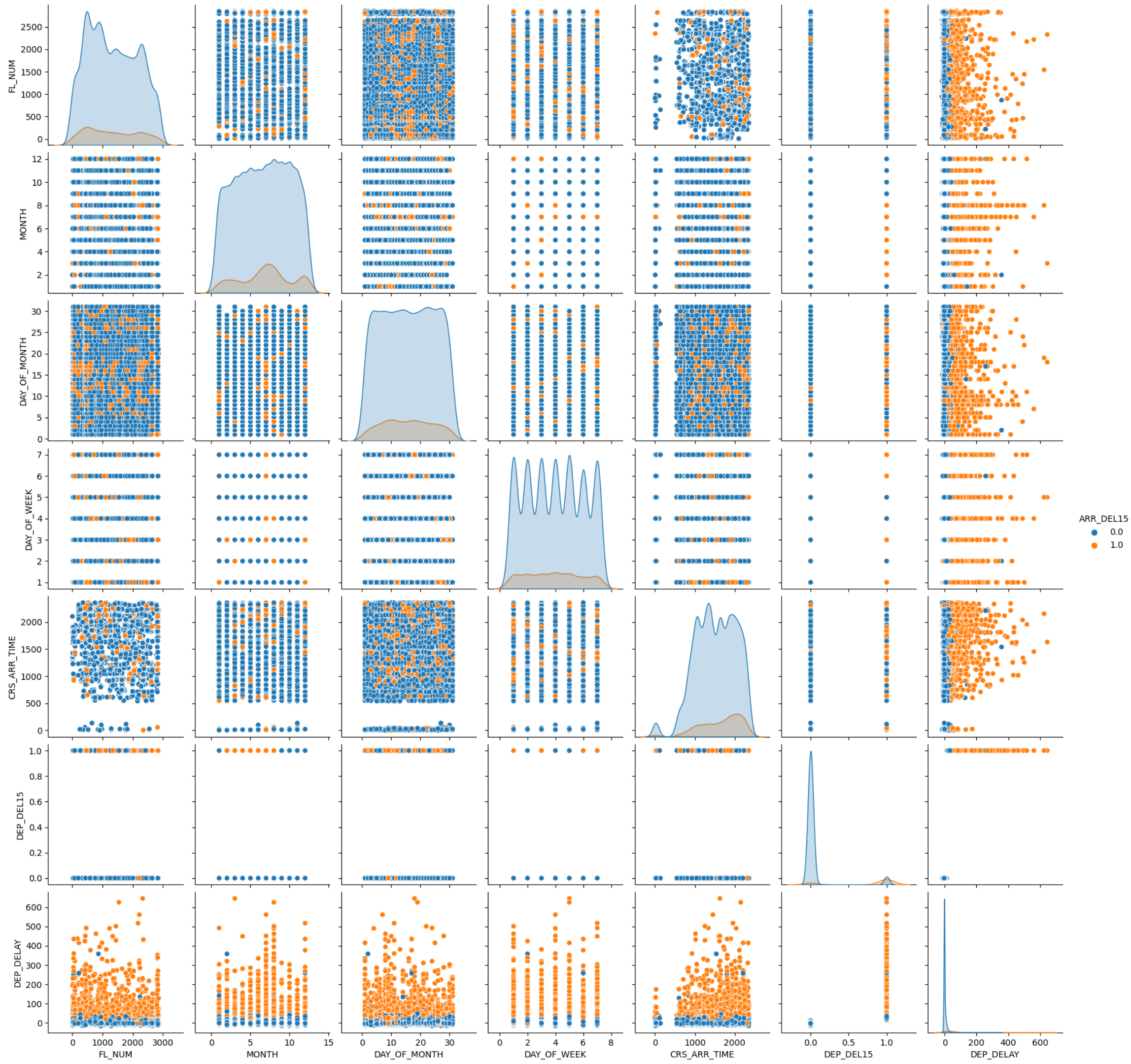
```
<ipython-input-77-8df7bcac526d>:1: FutureWarning: The default value of n
umeric_only in DataFrame.corr is deprecated. In a future version, it wil
l default to False. Select only valid columns or specify the value of nu
meric_only to silence this warning.
  sns.heatmap(df.corr(),annot=True)
```

Out[ ]: <Axes: >



In [ ]:
```
sns.pairplot(df,hue='ARR_DEL15')
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x7f81c8050fd0>




```
In [ ]: x = df[['FL_NUM','MONTH','DAY_OF_MONTH','DAY_OF_WEEK','ORIGIN','DEST','CR
        y = df['ARR_DEL15']
```

HANDLING CATEGORICAL VALUES & SCALING THE DATA :-

```
In [ ]: ct1 = ColumnTransformer([('oe',OrdinalEncoder(),['FL_NUM']),('ohe',OneHot
        ct2 = ColumnTransformer([('sc',StandardScaler(),['oe__FL_NUM','remainder_

        x = pd.DataFrame(ct1.fit_transform(x),columns=ct1.get_feature_names_out()
        x = pd.DataFrame(ct2.fit_transform(x),columns=ct2.get_feature_names_out()

        x.head()
```

Out[ ]:

	sc__oe__FL_NUM	sc__remainder__CRS_ARR_TIME	sc__remainder__DEP_DELAY	remainde
0	0.109290	1.205371	-0.174060	
1	0.239959	-0.203612	-0.256033	
2	0.395756	-0.641431	-0.174060	
3	0.581707	-0.402620	-0.201385	
4	0.642016	-1.851405	-0.338007	

In [ ]:
```
pickle.dump(ct1,open('col_trans1.pkl','wb'))
pickle.dump(ct2,open('col_trans2.pkl','wb'))
```

In [ ]:
```
x.head()
```

Out[ ]:

	sc__oe__FL_NUM	sc__remainder__CRS_ARR_TIME	sc__remainder__DEP_DELAY	remainde
0	0.109290	1.205371	-0.174060	
1	0.239959	-0.203612	-0.256033	
2	0.395756	-0.641431	-0.174060	
3	0.581707	-0.402620	-0.201385	
4	0.642016	-1.851405	-0.338007	

In [ ]:
```
y.head()
```

Out[ ]:
```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: ARR_DEL15, dtype: float64
```

SPLITTING THE DATASET INTO TRAINING AND TESTING :-

In [ ]:
```
x_train , x_test , y_train , y_test = train_test_split(x,y,test_size=0.2)
```

MODEL BUILDING :-

RANDOM FOREST MODEL :-

In [ ]:
```
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
y_pred = rfc.predict(x_test)

acc = accuracy_score(y_test,y_pred)
acc
```

Out[ ]:
```
0.9425901201602136
```

DECISION TREE MODEL :-

In [ ]:
```
dtc = DecisionTreeClassifier()
dtc.fit(x_train,y_train)
y_pred = dtc.predict(x_test)

acc = accuracy_score(y_test,y_pred)
acc
```

Out[ ]:
```
0.9020916777926123
```

ANN MODEL :-

In [ ]:
```
ann = Sequential()
ann.add(Dense(8,activation='relu'))
ann.add(Dense(32,activation='relu'))
ann.add(Dense(32,activation='relu'))
ann.add(Dense(1,activation='sigmoid'))

ann.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accurac

ann.fit(x_train,y_train,batch_size=4,validation_split=0.2,epochs=15)
```

```
Epoch 1/15
1797/1797 [==============================] - 5s 2ms/step - loss: 0.2333
- accuracy: 0.9217 - val_loss: 0.2344 - val_accuracy: 0.9238
Epoch 2/15
1797/1797 [==============================] - 4s 2ms/step - loss: 0.1771
- accuracy: 0.9421 - val_loss: 0.2213 - val_accuracy: 0.9238
Epoch 3/15
1797/1797 [==============================] - 6s 3ms/step - loss: 0.1716
- accuracy: 0.9431 - val_loss: 0.2309 - val_accuracy: 0.9226
Epoch 4/15
1797/1797 [==============================] - 4s 2ms/step - loss: 0.1679
- accuracy: 0.9438 - val_loss: 0.2247 - val_accuracy: 0.9277
Epoch 5/15
1797/1797 [==============================] - 4s 2ms/step - loss: 0.1645
- accuracy: 0.9450 - val_loss: 0.2272 - val_accuracy: 0.9243
Epoch 6/15
1797/1797 [==============================] - 6s 3ms/step - loss: 0.1625
- accuracy: 0.9467 - val_loss: 0.2304 - val_accuracy: 0.9260
Epoch 7/15
1797/1797 [==============================] - 4s 2ms/step - loss: 0.1596
- accuracy: 0.9467 - val_loss: 0.2323 - val_accuracy: 0.9226
Epoch 8/15
1797/1797 [==============================] - 4s 2ms/step - loss: 0.1557
- accuracy: 0.9489 - val_loss: 0.2344 - val_accuracy: 0.9221
Epoch 9/15
1797/1797 [==============================] - 8s 4ms/step - loss: 0.1535
- accuracy: 0.9475 - val_loss: 0.2386 - val_accuracy: 0.9249
Epoch 10/15
1797/1797 [==============================] - 4s 2ms/step - loss: 0.1498
- accuracy: 0.9521 - val_loss: 0.2380 - val_accuracy: 0.9243
Epoch 11/15
1797/1797 [==============================] - 4s 2ms/step - loss: 0.1470
- accuracy: 0.9516 - val_loss: 0.2423 - val_accuracy: 0.9226
Epoch 12/15
1797/1797 [==============================] - 5s 3ms/step - loss: 0.1451
- accuracy: 0.9524 - val_loss: 0.2501 - val_accuracy: 0.9226
Epoch 13/15
1797/1797 [==============================] - 4s 2ms/step - loss: 0.1424
- accuracy: 0.9537 - val_loss: 0.2503 - val_accuracy: 0.9215
Epoch 14/15
1797/1797 [==============================] - 4s 2ms/step - loss: 0.1399
- accuracy: 0.9527 - val_loss: 0.2648 - val_accuracy: 0.9182
Epoch 15/15
1797/1797 [==============================] - 5s 3ms/step - loss: 0.1361
- accuracy: 0.9556 - val_loss: 0.2646 - val_accuracy: 0.9204
```

Out[ ]:
```
<keras.callbacks.History at 0x7f81c02ac0a0>
```

In [ ]:
```
y_pred = ann.predict(x_train)
```

```
y_pred = [0 if x<0.5 else 1 for x in y_pred]
acc = accuracy_score(y_train,y_pred)
print('train data prediction accuracy : ',acc)

y_pred = ann.predict(x_test)

y_pred = [0 if x<0.5 else 1 for x in y_pred]
acc = accuracy_score(y_test,y_pred)
print('test data prediction accuracy : ',acc)
```

```
281/281 [==============================] - 1s 2ms/step
train data prediction accuracy :  0.9491317898486198
71/71 [==============================] - 0s 1ms/step
test data prediction accuracy :  0.9345794392523364
```

HYPER PARAMETER TUNING :-

In [ ]:

```
from scipy.stats import randint
params = {
    'n_estimators':[int(x) for x in np.linspace(50,500,50)],
    'criterion':['gini','entropy'],
    'max_features':['sqrt','log2'],
    'max_depth':[None,5,10,15,20,25,30],
    'min_samples_split':[int(x) for x in np.linspace(2,20)],
    'min_samples_leaf':[int(x) for x in np.linspace(1,20)],
}
rscv = RandomizedSearchCV(estimator=RandomForestClassifier(),param_distri
rscv.fit(x_train,y_train)

y_pred = rscv.predict(x_test)
acc = accuracy_score(y_pred,y_test)
print('accuracy score : ',acc)
print(rscv.best_params_)
```

```
accuracy score :  0.9457053849577214
{'n_estimators': 114, 'min_samples_split': 7, 'min_samples_leaf': 3, 'ma
x_features': 'sqrt', 'max_depth': 20, 'criterion': 'entropy'}
```

In [ ]:

```
rfc2 = RandomForestClassifier(n_estimators= 114, min_samples_split= 7, mi
rfc2.fit(x_train,y_train)
y_pred = rfc2.predict(x_test)

acc = accuracy_score(y_test,y_pred)
print('accuracy score : ',acc)
```

```
accuracy score :  0.945260347129506
```

In [ ]:

```
params = {
    'max_depth':list(range(3,14,2)),
    'criterion':['gini','entropy'],
    'min_samples_split':list(range(2,11,2)),
    'min_samples_leaf':list(range(1,6))
}
gscv = GridSearchCV(estimator=DecisionTreeClassifier(),param_grid=params,
gscv.fit(x_train,y_train)

y_pred = gscv.predict(x_test)
acc = accuracy_score(y_pred,y_test)
print('accuracy score : ',acc)
print(gscv.best_params_)
```

```
accuracy score :  0.9434801958166444
{'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 1, 'min_sam
ples_split': 2}
```

In [ ]:
```
dtc2 = DecisionTreeClassifier(criterion= 'entropy', max_depth= 5, min_sam
dtc2.fit(x_train,y_train)
y_pred = dtc2.predict(x_test)

acc = accuracy_score(y_test,y_pred)
print('accuracy score : ',acc)
```

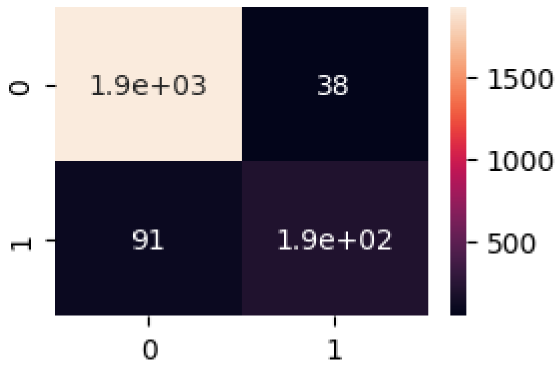
```
accuracy score :  0.943035157988429
```

TESTING THE MODEL WITH MULTIPLE EVALUATION METRICS (AFTER HYPER PARAMETER TUNING) :-

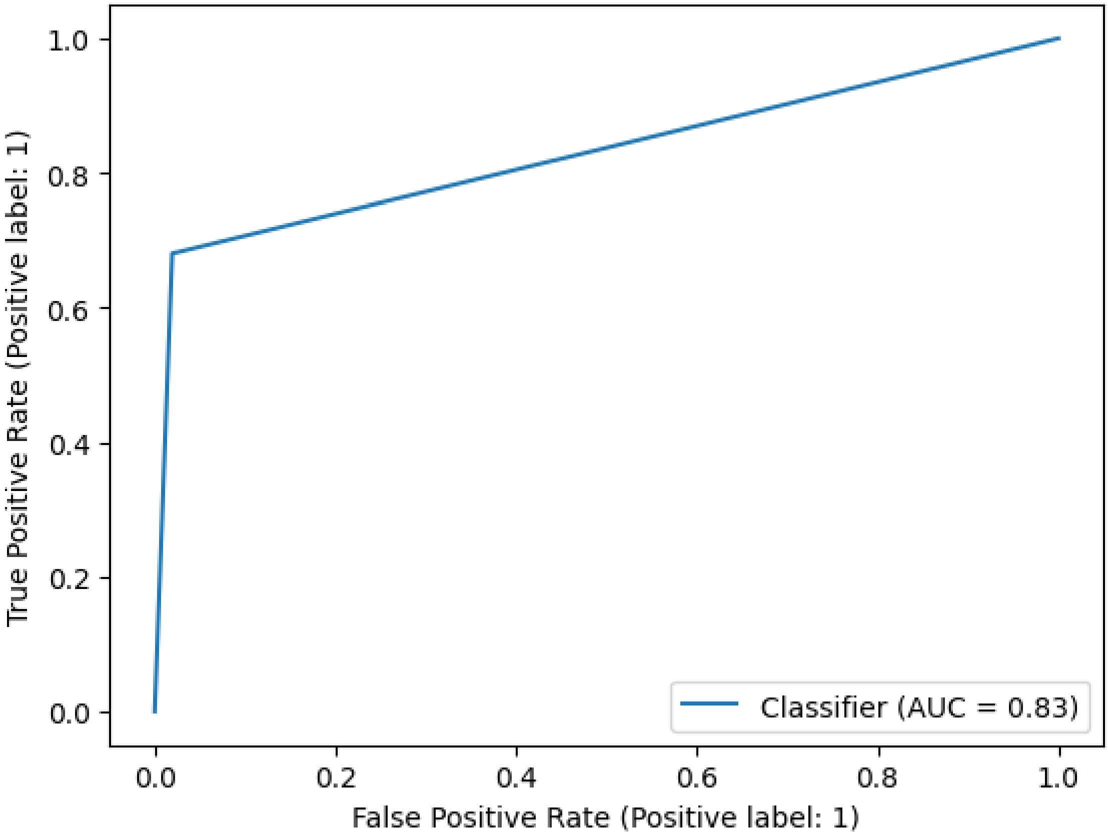
In [ ]:
```
def cl_res(name,model):
    y_pred = model.predict(x_test)
    if(name=='artificial_neural_network'):
        y_pred = [0 if x<0.5 else 1 for x in y_pred]
    print(name,' :-\n-------------------------')
    print('accuracy score of ',name,' : ',accuracy_score(y_test,y_pred))
    print(classification_report(y_test,y_pred,target_names=['no delay','d
    print('confusion matrix : \n',confusion_matrix(y_test,y_pred))
    print('\n')
    # plt.subplot(121)
    plt.figure(figsize=(3,2))
    sns.heatmap(confusion_matrix(y_test,y_pred),annot=True)
    # plt.subplot(122)
    plt.figure(figsize=(1,1))
    RocCurveDisplay.from_predictions(y_test,y_pred)
    plt.show()
    print('\n\n')
```

In [ ]:
```
cl_res('random_forest_classifier(before tuning)',rfc)
```

```
random_forest_classifier(before tuning)  :-
-------------------------
accuracy score of  random_forest_classifier(before tuning)  :  0.9425901
201602136
              precision    recall  f1-score   support

    no delay       0.95      0.98      0.97      1962
       delay       0.84      0.68      0.75       285

    accuracy                           0.94      2247
   macro avg       0.90      0.83      0.86      2247
weighted avg       0.94      0.94      0.94      2247

confusion matrix :
 [[1924   38]
 [  91  194]]
```

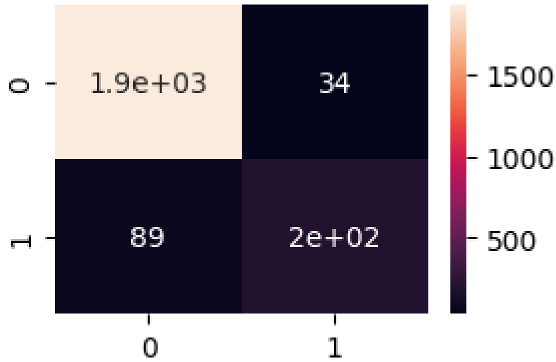



```
<Figure size 100x100 with 0 Axes>
```

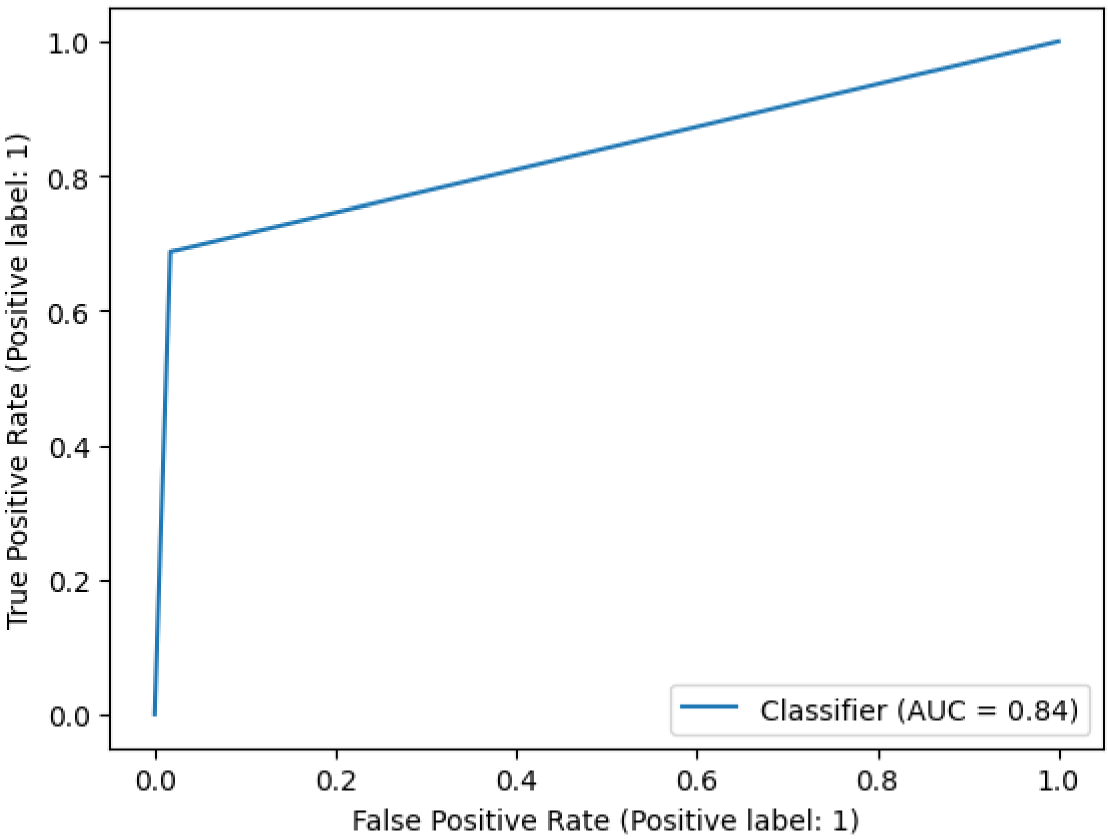


In [ ]:
```
cl_res('random_forest_classifier(after tuning)',rfc2)
```

```
random_forest_classifier(after tuning)  :-
-------------------------
accuracy score of  random_forest_classifier(after tuning)  :  0.94526034
7129506
              precision    recall  f1-score   support

    no delay       0.96      0.98      0.97      1962
       delay       0.85      0.69      0.76       285

    accuracy                           0.95      2247
   macro avg       0.90      0.84      0.87      2247
weighted avg       0.94      0.95      0.94      2247

confusion matrix :
 [[1928   34]
 [  89  196]]
```

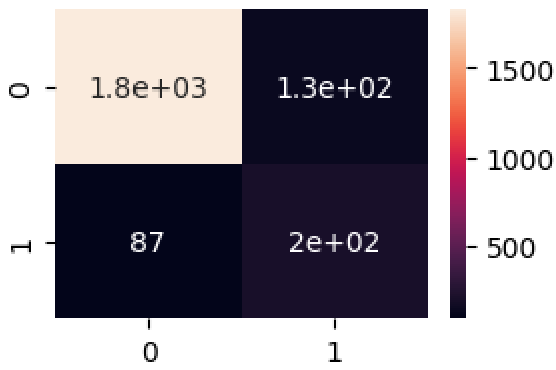


<Figure size 100x100 with 0 Axes>

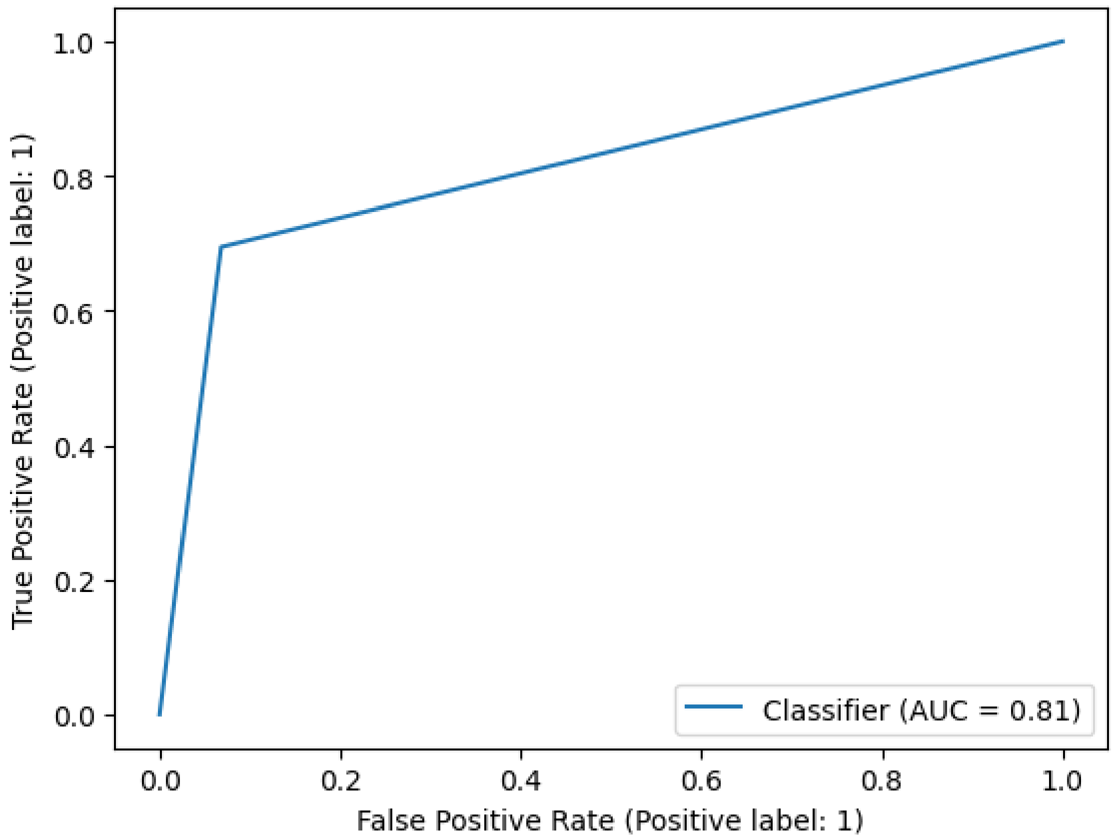


```
In [ ]: cl_res('decision_tree_classifier(before tuning)',dtc)
```

```
decision_tree_classifier(before tuning)  :-
-------------------------
accuracy score of  decision_tree_classifier(before tuning)  :  0.9020916
777926123
              precision    recall  f1-score   support

    no delay       0.95      0.93      0.94      1962
       delay       0.60      0.69      0.64       285

    accuracy                           0.90      2247
   macro avg       0.78      0.81      0.79      2247
weighted avg       0.91      0.90      0.91      2247

confusion matrix :
 [[1829  133]
 [  87  198]]
```

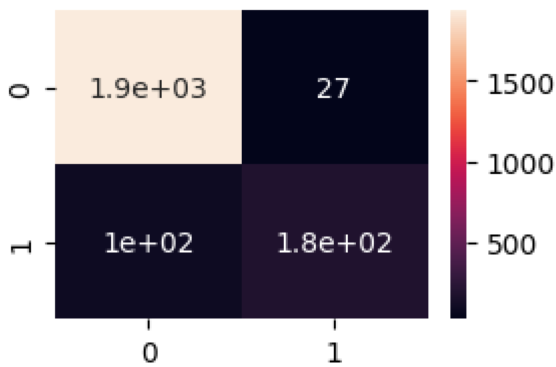



```
<Figure size 100x100 with 0 Axes>
```

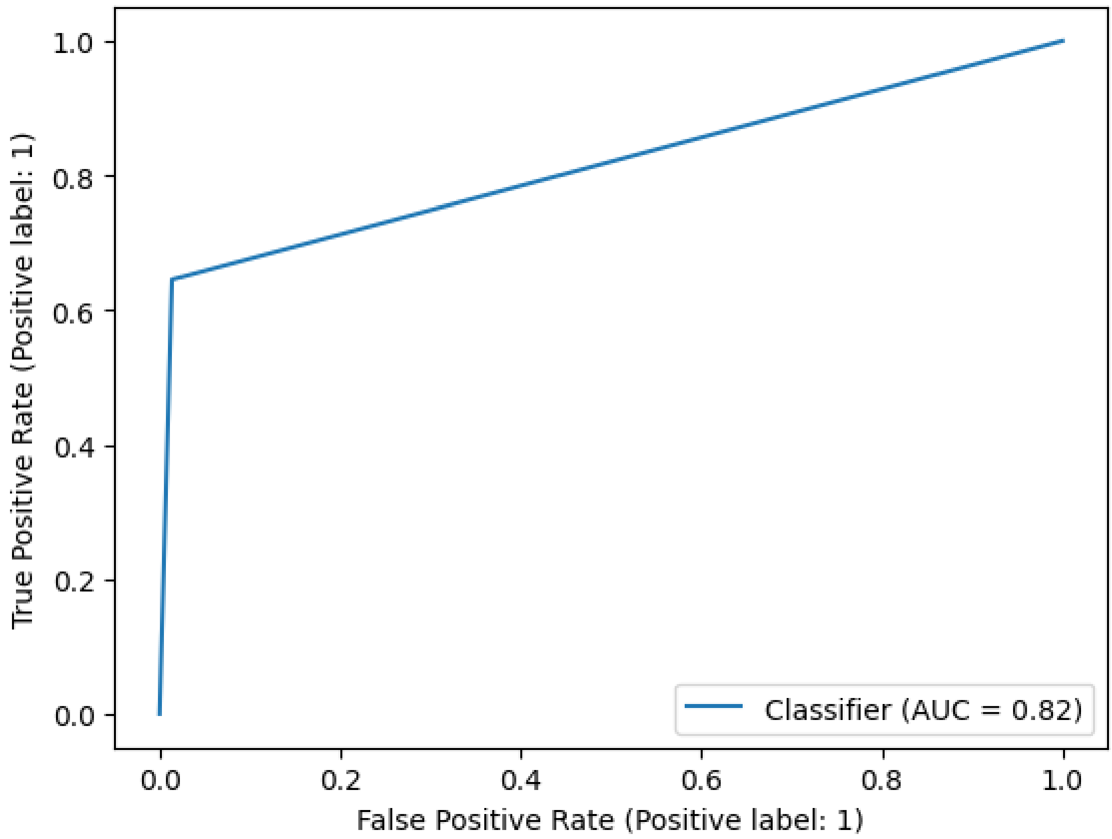


In [ ]:
```
cl_res('decision_tree_classifier(after tuning)',dtc2)
```

```
decision_tree_classifier(after tuning)  :-
-------------------------
accuracy score of  decision_tree_classifier(after tuning)  :  0.94303515
7988429
              precision    recall  f1-score   support

    no delay       0.95      0.99      0.97      1962
       delay       0.87      0.65      0.74       285

    accuracy                           0.94      2247
   macro avg       0.91      0.82      0.85      2247
weighted avg       0.94      0.94      0.94      2247

confusion matrix :
 [[1935   27]
 [ 101  184]]
```

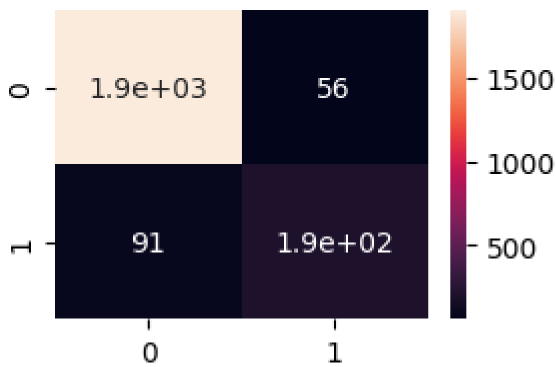



```
<Figure size 100x100 with 0 Axes>
```

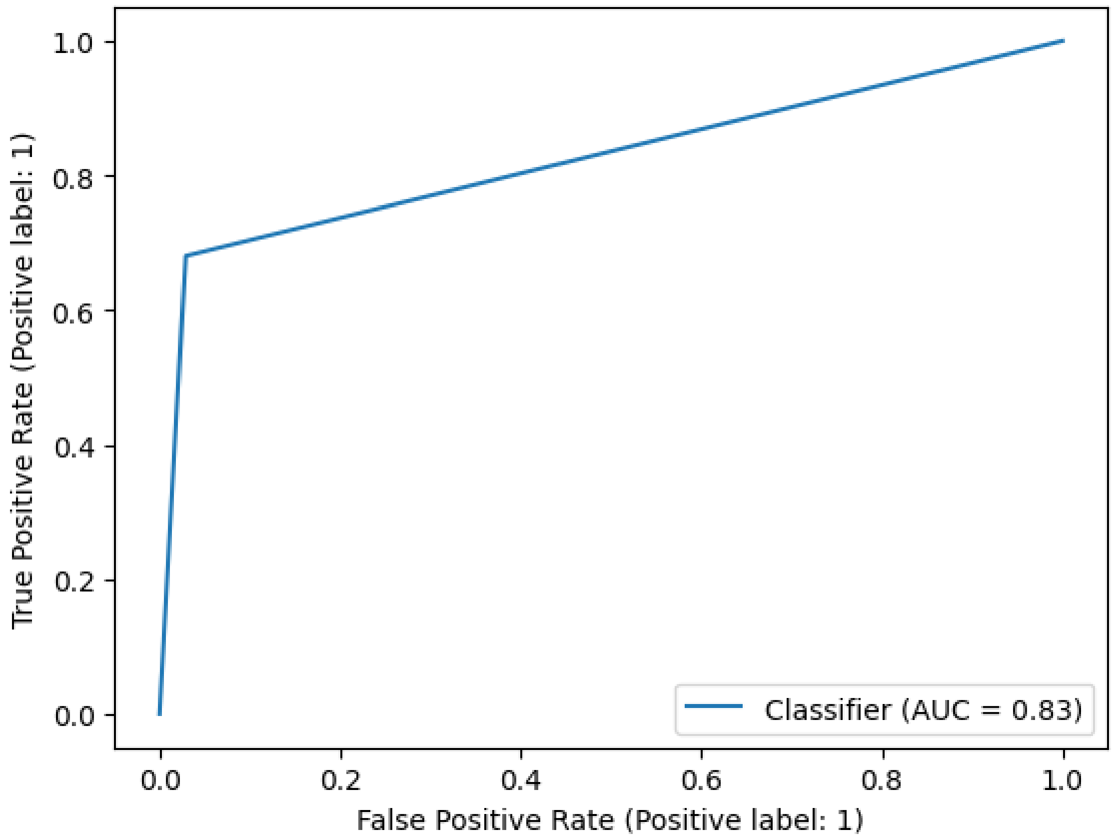


In [ ]:
```
cl_res('artificial_neural_network',ann)
```

```
71/71 [==============================] - 0s 1ms/step
artificial_neural_network  :-
-------------------------
accuracy score of  artificial_neural_network  :  0.9345794392523364
              precision    recall  f1-score   support

    no delay       0.95      0.97      0.96      1962
       delay       0.78      0.68      0.73       285

    accuracy                           0.93      2247
   macro avg       0.87      0.83      0.84      2247
weighted avg       0.93      0.93      0.93      2247

confusion matrix :
 [[1906   56]
 [  91  194]]
```



<Figure size 100x100 with 0 Axes>



- Here Random Forest Classifier (after tuning) has the highest accuracy score and good at other evaluation metrics, so we are going to save that model.

SAVING THE MODEL :-

In [ ]:
```
pickle.dump(rfc2,open('random_forest_classifier.pkl','wb'))
```