In [1]:
```
# importing all relevent  liabrary of python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
# extracting data from csv to data frame
df=pd.read_csv(r"C:\Users\Admin\Downloads\new_data_flight.csv")
df
```

Out[3]:

	MONTH	DAY_OF_WEEK	DEP_DEL15	DEP_TIME_BLK	DISTANCE_GROUP
0	1	7	0	0800-0859	2
1	1	7	0	0700-0759	7
2	1	7	0	0600-0659	7
3	1	7	0	0600-0659	9
4	1	7	0	0001-0559	7
...	...	...	...	...	...
29994	1	2	0	0600-0659	2
29995	1	2	1	1300-1359	3
29996	1	2	0	0700-0759	3
29997	1	2	1	0700-0759	6
29998	1	2	0	0600-0659	3

29999 rows × 26 columns

In [3]:
```
print(len(df))
```

```
29999
```

In [4]:
```
#check wheather null values is available or not
df.isnull().sum()
```

Out[4]:
```
MONTH                           0
DAY_OF_WEEK                     0
DEP_DEL15                       0
DEP_TIME_BLK                    0
DISTANCE_GROUP                  0
SEGMENT_NUMBER                  0
CONCURRENT_FLIGHTS              0
NUMBER_OF_SEATS                 0
CARRIER_NAME                    0
AIRPORT_FLIGHTS_MONTH           0
AIRLINE_FLIGHTS_MONTH           0
AIRLINE_AIRPORT_FLIGHTS_MONTH   0
AVG_MONTHLY_PASS_AIRPORT        0
AVG_MONTHLY_PASS_AIRLINE        0
FLT_ATTENDANTS_PER_PASS         0
GROUND_SERV_PER_PASS            0
PLANE_AGE                       0
DEPARTING_AIRPORT               0
LATITUDE                        0
LONGITUDE                       0
PREVIOUS_AIRPORT                0
PRCP                            0
SNOW                            0
SNWD                            0
TMAX                            0
AWND                            0
dtype: int64
```

In [5]:
```
# taking information of all columns in data frame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29999 entries, 0 to 29998
Data columns (total 26 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   MONTH                           29999 non-null  int64  
 1   DAY_OF_WEEK                     29999 non-null  int64  
 2   DEP_DEL15                       29999 non-null  int64  
 3   DEP_TIME_BLK                    29999 non-null  object 
 4   DISTANCE_GROUP                  29999 non-null  int64  
 5   SEGMENT_NUMBER                  29999 non-null  int64  
 6   CONCURRENT_FLIGHTS              29999 non-null  int64  
 7   NUMBER_OF_SEATS                 29999 non-null  int64  
 8   CARRIER_NAME                    29999 non-null  object 
 9   AIRPORT_FLIGHTS_MONTH           29999 non-null  int64  
 10  AIRLINE_FLIGHTS_MONTH           29999 non-null  int64  
 11  AIRLINE_AIRPORT_FLIGHTS_MONTH   29999 non-null  int64  
 12  AVG_MONTHLY_PASS_AIRPORT        29999 non-null  int64  
 13  AVG_MONTHLY_PASS_AIRLINE        29999 non-null  int64  
 14  FLT_ATTENDANTS_PER_PASS         29999 non-null  float64
 15  GROUND_SERV_PER_PASS            29999 non-null  float64
 16  PLANE_AGE                       29999 non-null  int64  
 17  DEPARTING_AIRPORT               29999 non-null  object 
 18  LATITUDE                        29999 non-null  float64
 19  LONGITUDE                       29999 non-null  float64
 20  PREVIOUS_AIRPORT                29999 non-null  object 
 21  PRCP                            29999 non-null  float64
 22  SNOW                            29999 non-null  float64
 23  SNWD                            29999 non-null  float64
 24  TMAX                            29999 non-null  int64  
 25  AWND                            29999 non-null  float64
dtypes: float64(8), int64(14), object(4)
memory usage: 6.0+ MB
```

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

Out[6]:

	MONTH	DAY_OF_WEEK	DEP_DEL15	DEP_TIME_BLK	DISTANCE_GROUP
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
29994	False	False	False	False	False
29995	False	False	False	False	False
29996	False	False	False	False	False
29997	False	False	False	False	False
29998	False	False	False	False	False

29999 rows × 26 columns

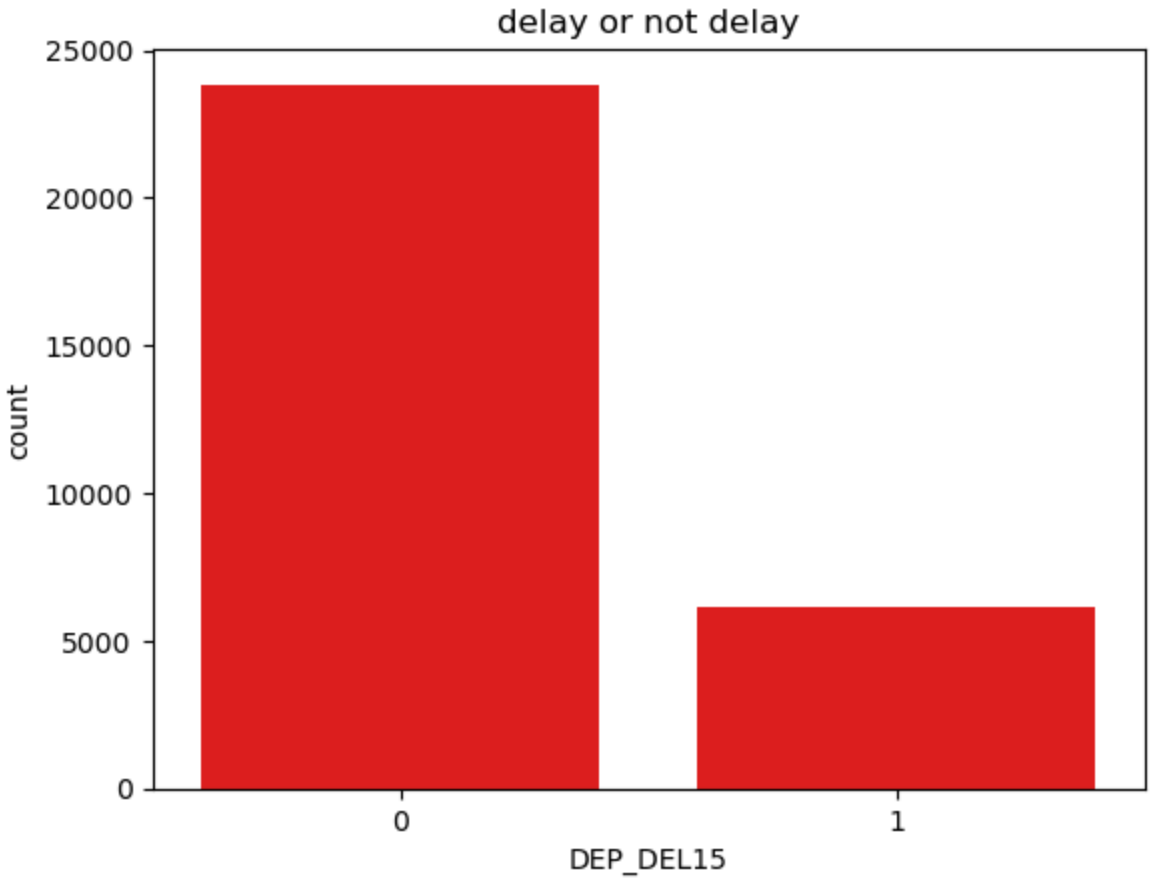
In [8]:
```
# deleting irrelevent columns
```

In [7]:
```
df = df.drop(columns=['DEP_TIME_BLK',"NUMBER_OF_SEATS","SEGMENT_NUMBER","AVG
                      "AVG_MONTHLY_PASS_AIRLINE","PLANE_AGE","PREVIOUS_AIRPOR
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29999 entries, 0 to 29998
Data columns (total 19 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   MONTH                           29999 non-null  int64
 1   DAY_OF_WEEK                     29999 non-null  int64
 2   DEP_DEL15                       29999 non-null  int64
 3   DISTANCE_GROUP                  29999 non-null  int64
 4   CONCURRENT_FLIGHTS              29999 non-null  int64
 5   CARRIER_NAME                    29999 non-null  object
 6   AIRPORT_FLIGHTS_MONTH           29999 non-null  int64
 7   AIRLINE_FLIGHTS_MONTH           29999 non-null  int64
 8   AIRLINE_AIRPORT_FLIGHTS_MONTH   29999 non-null  int64
 9   FLT_ATTENDANTS_PER_PASS         29999 non-null  float64
 10  GROUND_SERV_PER_PASS            29999 non-null  float64
 11  DEPARTING_AIRPORT               29999 non-null  object
 12  LATITUDE                        29999 non-null  float64
 13  LONGITUDE                       29999 non-null  float64
 14  PRCP                            29999 non-null  float64
 15  SNOW                            29999 non-null  float64
 16  SNWD                            29999 non-null  float64
 17  TMAX                            29999 non-null  int64
 18  AWND                            29999 non-null  float64
dtypes: float64(8), int64(9), object(2)
memory usage: 4.3+ MB
```

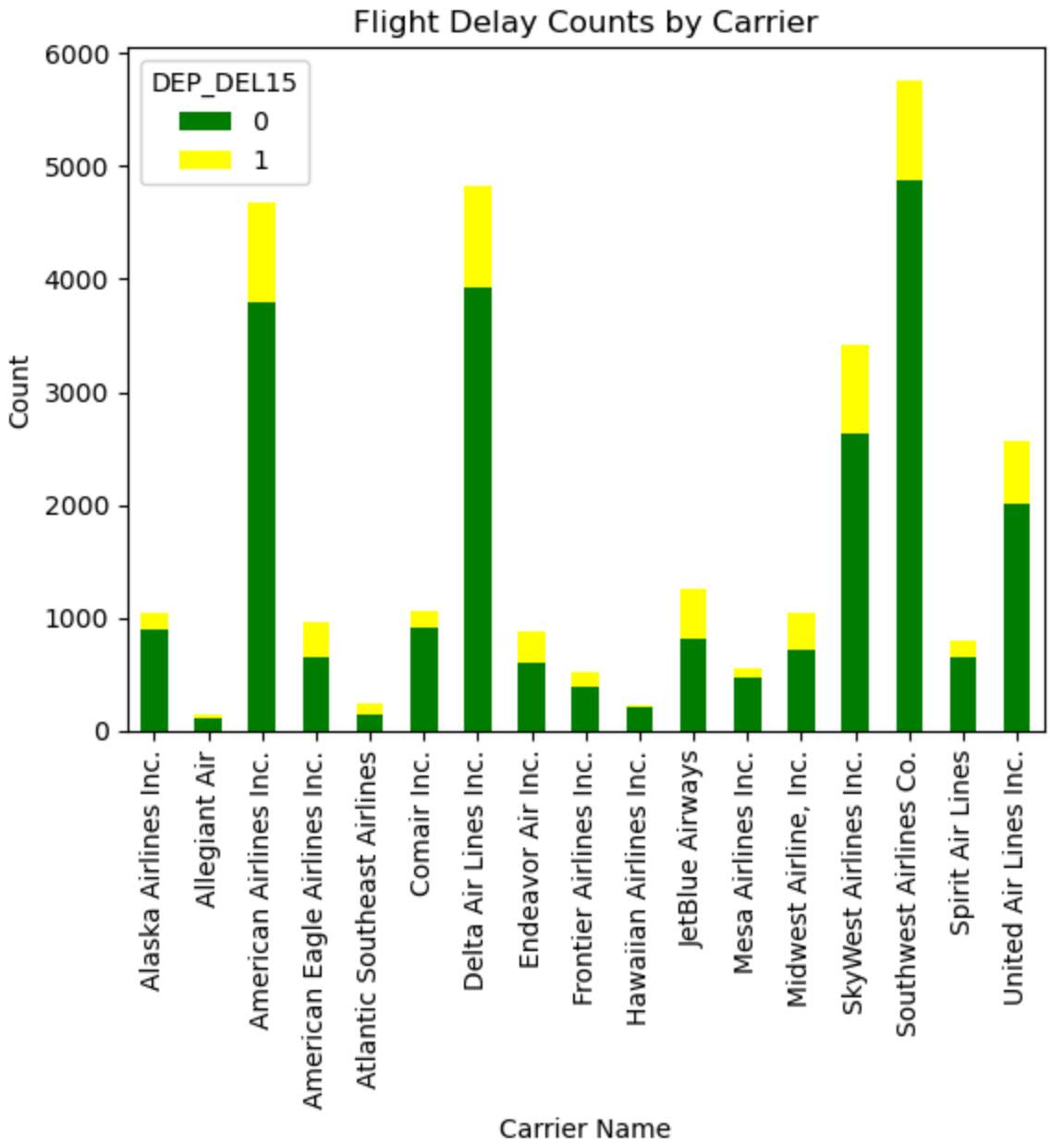
In [9]:
```
#counts 0 and 1 are in balance
sns.countplot(x='DEP_DEL15', data=df,color="red")
plt.title("delay or not delay")
plt.show()
print(df['DEP_DEL15'].value_counts())
```




```
DEP_DEL15
0    23842
1     6157
Name: count, dtype: int64
```

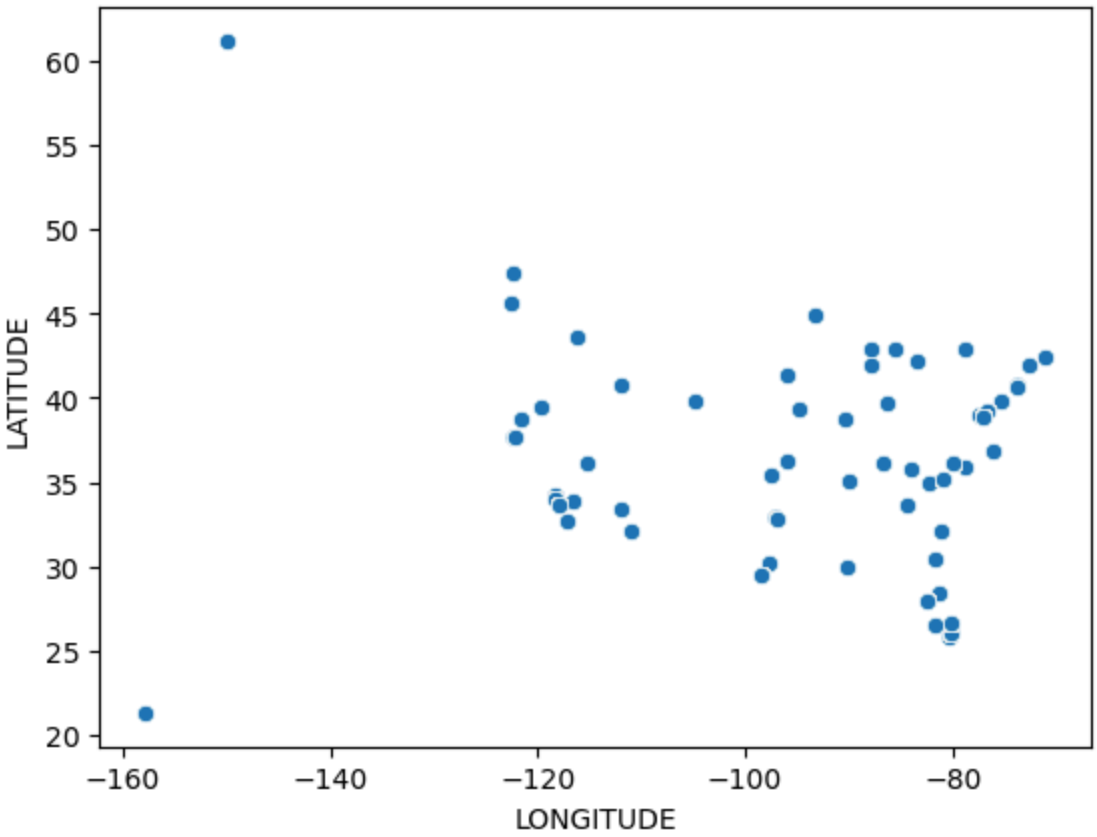
In [10]:

```python
#  here we Count the occurrences of DEP_DEL15 (flight delay) for each carrie
carrier_delay_counts = df.groupby('CARRIER_NAME')['DEP_DEL15'].value_counts(

# Create the bar plot
carrier_delay_counts.plot(kind='bar', stacked=True, color=['green', 'yellow'

# Add labels and title
plt.xlabel('Carrier Name')
plt.ylabel('Count')
plt.title('Flight Delay Counts by Carrier')

# Show the plot
plt.xticks(rotation=90)
plt.show()
```



In [11]:
```
sns.scatterplot(x='LONGITUDE', y='LATITUDE', data=df)
```

Out[11]:
```
<Axes: xlabel='LONGITUDE', ylabel='LATITUDE'>
```



In [12]:
```
# converting object column into numeric using labelencoder
from sklearn.preprocessing import LabelEncoder
lr=LabelEncoder()
lr
```

Out[12]:
```
LabelEncoder()
```

In [13]:
```
df_category =list(df.select_dtypes("object").columns)
df_category
for i in df_category:
    df[i]=lr.fit_transform(df[i])
df
```

Out[13]:

	MONTH	DAY_OF_WEEK	DEP_DEL15	DISTANCE_GROUP	CONCURRENT_I
0	1	7	0	2	
1	1	7	0	7	
2	1	7	0	7	
3	1	7	0	9	
4	1	7	0	7	
...	...	...	...	...	
29994	1	2	0	2	
29995	1	2	1	3	
29996	1	2	0	3	
29997	1	2	1	6	
29998	1	2	0	3	

29999 rows × 19 columns

In [14]:

```
# scaling data for better prdiction using standard scaler
from sklearn.preprocessing import StandardScaler
#target_column="DEP_DEL15"
scaler= StandardScaler()
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns
numerical_columns = numerical_columns.drop("DEP_DEL15")
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
df
```

Out[14]:

	MONTH	DAY_OF_WEEK	DEP_DEL15	DISTANCE_GROUP	CONCURRENT_I
0	0.0	2.941529	0	-0.759451	-(
1	0.0	2.941529	0	1.323336	-(
2	0.0	2.941529	0	1.323336	-(
3	0.0	2.941529	0	2.156451	-(
4	0.0	2.941529	0	1.323336	-(
...	...	...	...	...	
29994	0.0	-2.337421	0	-0.759451	-(
29995	0.0	-2.337421	1	-0.342894	(
29996	0.0	-2.337421	0	-0.342894	1
29997	0.0	-2.337421	1	0.906779	1
29998	0.0	-2.337421	0	-0.342894	-(

29999 rows × 19 columns

In [15]:
```
# defining x columns
x = df.drop(df.columns[2], axis=1)
x
```

Out[15]:

	MONTH	DAY_OF_WEEK	DISTANCE_GROUP	CONCURRENT_FLIGHTS	CAI
0	0.0	2.941529	-0.759451	-0.245402	
1	0.0	2.941529	1.323336	-0.049428	
2	0.0	2.941529	1.323336	-0.147415	
3	0.0	2.941529	2.156451	-0.147415	
4	0.0	2.941529	1.323336	-0.980303	
...	...	...	...	...	
29994	0.0	-2.337421	-0.759451	-0.098421	
29995	0.0	-2.337421	-0.342894	0.832454	
29996	0.0	-2.337421	-0.342894	1.518362	
29997	0.0	-2.337421	0.906779	1.518362	
29998	0.0	-2.337421	-0.342894	-0.098421	

29999 rows × 18 columns

In [16]:
```
# defining y column which is our target column
y=df["DEP_DEL15"]
y
```

Out[16]:
```
0        0
1        0
2        0
3        0
4        0
        ..
29994    0
29995    1
29996    0
29997    1
29998    0
Name: DEP_DEL15, Length: 29999, dtype: int64
```

In [19]:
```
print(len(x))
```

```
29999
```

In [20]:
```
print(len(y))
```

```
29999
```

In [17]:
```
# our data was unbalance here we are balancing our data using randomoversamp
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42)
x_resampled, y_resampled = ros.fit_resample(x,y)
```

In [18]:
```
# display the data is balanced
y_resampled.value_counts()
```

Out[18]:
```
DEP_DEL15
0    23842
1    23842
Name: count, dtype: int64
```

In [19]:
```
# split data into train and test
from sklearn.model_selection import train_test_split
```

In [20]:
```
x_train,x_test,y_train,y_test=train_test_split(x_resampled,y_resampled,test_
```

In [21]:
```
#import algorithm for training
from sklearn.linear_model import LogisticRegression
```

In [22]:
```
lr=LogisticRegression()
```

In [32]:
```
lr.fit(x_train,y_train)
```

Out[32]:

▾ LogisticRegression

LogisticRegression()

In [24]:
```
y_pred=lr.predict(x_test)
y_pred
```

Out[24]:
```
array([0, 0, 1, ..., 1, 1, 1], dtype=int64)
```

In [33]:
```
from sklearn.metrics import classification_report
```

In [34]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.63      0.66      0.65      4769
           1       0.65      0.62      0.63      4768

    accuracy                           0.64      9537
   macro avg       0.64      0.64      0.64      9537
weighted avg       0.64      0.64      0.64      9537
```

In [35]:
```
accuracy = lr.score(x_test, y_test)
print(f"Test accuracy: {accuracy}")
```

```
Test accuracy: 0.6392995700954178
```

In [ ]:
```
# importing GridSearchCV for make accuracy better
```

In [36]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'lbfgs'],
}

grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid_search.fit(x_train, y_train)

print(f"Best parameters: {grid_search.best_params_}")
```

```
Best parameters: {'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'}
```

In [33]:
```
grid_search.score(x_test,y_test)
```

Out[33]:
```
0.6392995700954178
```

In [92]:
```
lrh=LogisticRegression(C=1,penalty="l2",solver="lbfgs")
lrh.fit(x_train,y_train)
y_pred=lr.predict(x_test)
y_pred
```

Out[92]:
```
array([0, 0, 1, ..., 1, 1, 1], dtype=int64)
```

In [93]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.63      0.66      0.65      4769
           1       0.65      0.62      0.63      4768

    accuracy                           0.64      9537
   macro avg       0.64      0.64      0.64      9537
weighted avg       0.64      0.64      0.64      9537
```

In [95]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [39]:
```
DC=DecisionTreeClassifier()
```

In [40]:
```
DC.fit(x_train,y_train)
```

Out[40]:
```
DecisionTreeClassifier()
```

In [41]:
```
y_pred=DC.predict(x_test)
y_pred
```

Out[41]:
```
array([0, 1, 1, ..., 0, 1, 1], dtype=int64)
```

In [42]:
```
print(classification_report(y_test,y_pred))
```

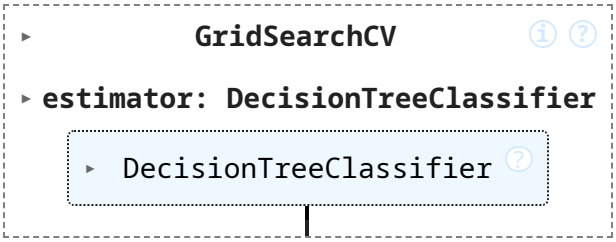
```
              precision    recall  f1-score   support

           0       0.85      0.76      0.80      4769
           1       0.78      0.87      0.82      4768

    accuracy                           0.81      9537
   macro avg       0.82      0.81      0.81      9537
weighted avg       0.82      0.81      0.81      9537
```

In [41]:
```
# now hypertuning for more accurracy
```

In [43]:
```
from sklearn.model_selection import GridSearchCV
```

In [44]:
```
Parameter={"max_depth":[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20],"m
GS=GridSearchCV(DC,Parameter)
GS.fit(x_train,y_train)
GS
```

Out[44]:



In [45]:
```
GS.best_params_
```

Out[45]:
```
{'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 5}
```

In [96]:
```
DCh=DecisionTreeClassifier(max_depth=20,min_samples_leaf=2,min_samples_split
DCh.fit(x_train,y_train)
y_pred=DCh.predict(x_test)
y_pred
```

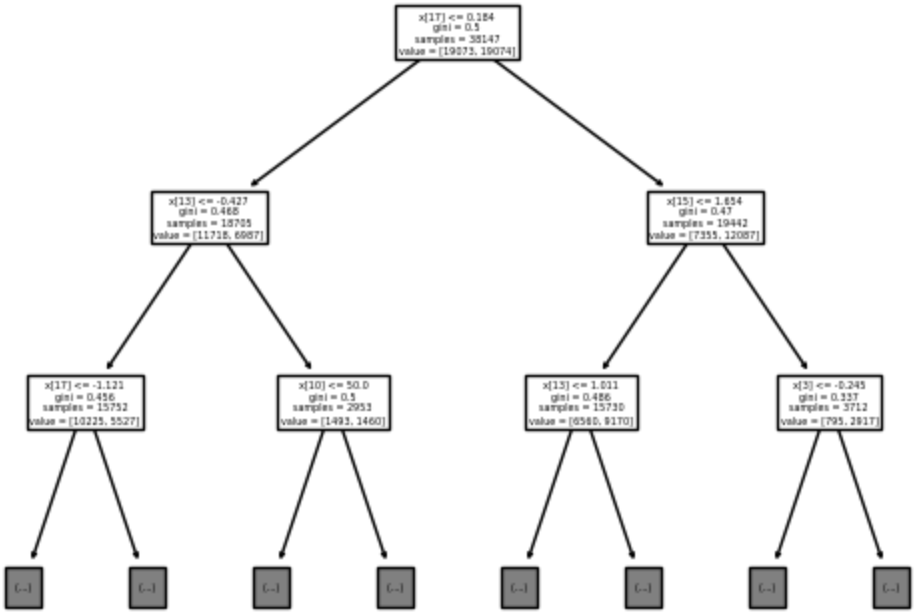
Out[96]:
```
array([0, 1, 1, ..., 0, 1, 1], dtype=int64)
```

In [46]:
```
# we got best accuraccy using GridSearchCV
```

In [97]:
```
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.79	0.74	0.76	4769
1	0.75	0.80	0.78	4768
accuracy			0.77	9537
macro avg	0.77	0.77	0.77	9537
weighted avg	0.77	0.77	0.77	9537

In [48]:
```
from sklearn.tree import plot_tree
plot_tree(DC,max_depth=2)
plt.show()
```



In [64]:
```
from sklearn.ensemble import RandomForestClassifier
RF=RandomForestClassifier(n_estimators=100)
RF.fit(x_train,y_train)
y_pred=RF.predict(x_test)
y_pred
```

Out[64]:
```
array([1, 1, 1, ..., 0, 1, 1], dtype=int64)
```

In [65]:
```
print(classification_report(y_test,y_pred))
```

```
                precision    recall  f1-score   support

           0       0.87      0.76      0.81      4769
           1       0.78      0.89      0.83      4768

    accuracy                           0.82      9537
   macro avg       0.83      0.82      0.82      9537
weighted avg       0.83      0.82      0.82      9537
```

In [74]:
```
from sklearn.svm import SVC
SV = SVC(C=1, gamma='scale', kernel='rbf')  # Using default gamma ('scale')
SV.fit(x_train, y_train)
y_pred = SV.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
                precision    recall  f1-score   support

           0       0.63      0.67      0.65      4769
           1       0.65      0.61      0.63      4768

    accuracy                           0.64      9537
   macro avg       0.64      0.64      0.64      9537
weighted avg       0.64      0.64      0.64      9537
```

In [77]:
```
from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
```

In [78]:
```
GB=GaussianNB()
```

In [79]:
```
GB.fit(x_train,y_train)
```

Out[79]:

▾ GaussianNB

```
GaussianNB()
```

In [80]:
```
y_pred=GB.predict(x_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
                precision    recall  f1-score   support

           0       0.57      0.77      0.66      4769
           1       0.65      0.43      0.52      4768

    accuracy                           0.60      9537
   macro avg       0.61      0.60      0.59      9537
weighted avg       0.61      0.60      0.59      9537
```

In [39]:
```
GB.score(x_test,y_test)
```

Out[39]:
```
0.5989304812834224
```

In [82]:
```
BN=BernoulliNB()
```

In [83]:
```
BN.fit(x_train,y_train)
```

Out[83]:
```
BernoulliNB()
```

In [84]:
```
y_pred=BN.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.64      0.67      0.65      4769
           1       0.65      0.63      0.64      4768

    accuracy                           0.65      9537
   macro avg       0.65      0.65      0.65      9537
weighted avg       0.65      0.65      0.65      9537
```

In [98]:
```
print("logisticregression  score")
print(lr.score(x_test,y_test))
#-------------------------------
print("logisticregression score with hypertune")
print(lrh.score(x_test,y_test))
#-------------------------------
print("DecisionTree score ")
print(DC.score(x_test,y_test))
#-------------------------------
print("DecisionTree score with hypertune")
print(DCh.score(x_test,y_test))
#-------------------------------
print("Randomforest score")
print(RF.score(x_test,y_test))
#-------------------------------
print("svm score")
print(SV.score(x_test,y_test))
#-------------------------------
print("naive_bayes with GaussianNB")
print(GB.score(x_test,y_test))
#-------------------------------
print("naive_bayes with BernoulliNB")
print(BN.score(x_test,y_test))
```

```
logisticregression  score
0.6392995700954178
logisticregression score with hypertune
0.6392995700954178
DecisionTree score
0.7693194925028836
DecisionTree score with hypertune
0.7689000733983433
Randomforest score
0.8230051378840306
svm score
0.6407675369613086
naive_bayes with GaussianNB
0.5989304812834224
naive_bayes with BernoulliNB
0.6476879521862221
```

Best Model:

The Random Forest model, with a score of 0.8230, appears to be the best-performing model from the list, as it has the highest accuracy compared to the others.