iml554-classification-assignment03

September 6, 2024

1 Problem statement:

The given dataset is from Dream Club which connects borrowers with investors. We will use lending data from 2007-2010 and build a classifier model to predict whether or not the borrower has paid back their loan in full.

Here are what the columns represent:

The given dataset is from Dream Club which connects borrowers with investors. We will use lending data from 2007-2010 and build a classifier model to predict whether or not the borrower has paid back their loan in full.

Here are what the columns represen1-

credit.policy: 1 if the customer meets the credit underwriting criteria of LendingClub.com, and 0 othe2- rwise.

purpose: The purpose of the loan (takes values "credit_card", "debt_consolidation", "educational", "major_purchase", "small_business", and "all_3- other").

int.rate: The interest rate of the loan, as a proportion (a rate of 11% would be stored as 0.11). Borrowers judged by LendingClub.com to be more risky are assigned higher inter4- est rates.

installment: The monthly installments owed by the borrower if the loa5- n is funded.

log.annual.inc: The natural log of the self-reported annual income of6- the borrower.

dti: The debt-to-income ratio of the borrower (amount of debt divided by7- annual income).

fico: The FICO credit scor8- e of the borrower.

days.with.cr.line: The number of days the borrower ha9- s had a credit line.

revol.bal: The borrower's revolving balance (amount unpaid at the end of the credi10- t card billing cycle).

revol.util: The borrower's revolving line utilization rate (the amount of the credit line used relative to 11- total credit available).

inq.last.6mths: The borrower's number of inquiries by credi12- tors in the last 6 months.

delinq.2yrs: The number of times the borrower had been 30+ days past due on a 13- payment in the past 2 years.

pub.rec: The borrower's number of derogatory public records (bankruptcy fili14- ngs, tax liens, or judgments).

not.fully.paid: whether or not the borrower paid back their loan in full

2 Q-1: Load the dataset and print the metadata in the notebook. 1M

[406]:
```python
# Importing required packages
import numpy as np
import pandas as pd
import warnings as war
war.filterwarnings("ignore")
```

[407]:
```python
# Defining dataset csv Path
dataSetPath="C:\\Users\\ASUS\\jupyterworkspace\\Assignment & Mini␣
↪Project\\Module_03_Classification\\Assignment\\03_Decision Tree\\loan_data.
↪csv"
# Loading dataSet
dataSetRead=pd.read_csv(dataSetPath)
```

[408]:
```python
# Displaying first 5 records to confirming data loading
print("*****************************************************Displaying below␣
↪first 5 records************************************************************")
dataSetRead.head()
```

```
*****************************************************Displaying below first 5
records************************************************************
```

[408]:
```
   credit.policy             purpose  int.rate  installment  log.annual.inc  \
0              1  debt_consolidation    0.1189       829.10       11.350407
1              1         credit_card    0.1071       228.22       11.082143
2              1  debt_consolidation    0.1357       366.86       10.373491
3              1  debt_consolidation    0.1008       162.34       11.350407
4              1         credit_card    0.1426       102.92       11.299732

     dti  fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
0  19.48   737        5639.958333      28854        52.1               0
1  14.29   707        2760.000000      33623        76.7               0
2  11.63   682        4710.000000       3511        25.6               1
3   8.10   712        2699.958333      33667        73.2               1
4  14.97   667        4066.000000       4740        39.5               0

   delinq.2yrs  pub.rec  not.fully.paid
0            0        0               0
1            0        0               0
2            0        0               0
3            0        0               0
4            1        0               0
```

[409]:
```
# Displaying last 5 records to confirming data loading
print("*****************************************************Displaying below␣
↪last 5 records***********************************************************")
dataSetRead.tail()
```

```
*****************************************************Displaying below last 5
records***********************************************************
```

[409]:
```
      credit.policy             purpose  int.rate  installment  \
9573              0           all_other    0.1461       344.76
9574              0           all_other    0.1253       257.70
9575              0  debt_consolidation    0.1071        97.81
9576              0    home_improvement    0.1600       351.58
9577              0  debt_consolidation    0.1392       853.43

      log.annual.inc    dti  fico  days.with.cr.line  revol.bal  revol.util  \
9573       12.180755  10.39   672       10474.000000     215372        82.1
9574       11.141862   0.21   722        4380.000000        184         1.1
9575       10.596635  13.09   687        3450.041667      10036        82.9
9576       10.819778  19.18   692        1800.000000          0         3.2
9577       11.264464  16.28   732        4740.000000      37879        57.0

      inq.last.6mths  delinq.2yrs  pub.rec  not.fully.paid
9573               2            0        0               1
9574               5            0        0               1
9575               8            0        0               1
9576               5            0        0               1
9577               6            0        0               1
```

[410]:
```
# Displaying dimension of dataSet
print("Dimention of Dataset:- {}".format(dataSetRead.shape[0:2]))
print("Total number of rows in Dataset:- {}".format(dataSetRead.shape[0]))
print("Total number of columns in Dataset:- {}".format(dataSetRead.shape[1]))
```

```
Dimention of Dataset:- (9578, 14)
Total number of rows in Dataset:- 9578
Total number of columns in Dataset:- 14
```

[411]:
```
# Displaying description & statistcal summary of the dataSet
dataSetRead.describe().T
```

[411]:
```
                   count        mean         std        min  \
credit.policy     9578.0    0.804970    0.396245   0.000000
int.rate          9578.0    0.122640    0.026847   0.060000
installment       9578.0  319.089413  207.071301  15.670000
log.annual.inc    9578.0   10.932117    0.614813   7.547502
dti               9578.0   12.606679    6.883970   0.000000
```

```
fico               9578.0    710.846314    37.970537  612.000000
days.with.cr.line  9578.0   4560.767197  2496.930377  178.958333
revol.bal          9578.0  16913.963876 33756.189557    0.000000
revol.util         9578.0     46.799236    29.014417    0.000000
inq.last.6mths     9578.0      1.577469     2.200245    0.000000
delinq.2yrs        9578.0      0.163708     0.546215    0.000000
pub.rec            9578.0      0.062122     0.262126    0.000000
not.fully.paid     9578.0      0.160054     0.366676    0.000000

                           25%          50%           75%           max
credit.policy         1.000000     1.000000      1.000000  1.000000e+00
int.rate              0.103900     0.122100      0.140700  2.164000e-01
installment         163.770000   268.950000    432.762500  9.401400e+02
log.annual.inc       10.558414    10.928884     11.291293  1.452835e+01
dti                   7.212500    12.665000     17.950000  2.996000e+01
fico                682.000000   707.000000    737.000000  8.270000e+02
days.with.cr.line  2820.000000  4139.958333   5730.000000  1.763996e+04
revol.bal          3187.000000  8596.000000  18249.500000  1.207359e+06
revol.util           22.600000    46.300000     70.900000  1.190000e+02
inq.last.6mths        0.000000     1.000000      2.000000  3.300000e+01
delinq.2yrs           0.000000     0.000000      0.000000  1.300000e+01
pub.rec               0.000000     0.000000      0.000000  5.000000e+00
not.fully.paid        0.000000     0.000000      0.000000  1.000000e+00
```

[412]:

```
# Displaying the columns and their respective data types
dataSetRead.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   credit.policy      9578 non-null   int64
 1   purpose            9578 non-null   object
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64
 11  delinq.2yrs        9578 non-null   int64
 12  pub.rec            9578 non-null   int64
 13  not.fully.paid     9578 non-null   int64
dtypes: float64(6), int64(7), object(1)
```

```
memory usage: 1.0+ MB
```

[413]:
```python
# Handling categorical variable purpose
cat_variable = pd.get_dummies(dataSetRead['purpose']).astype(int)
cat_variable
```

[413]:
```
      all_other  credit_card  debt_consolidation  educational  \
0             0            0                   1            0
1             0            1                   0            0
2             0            0                   1            0
3             0            0                   1            0
4             0            1                   0            0
…           …          …                   …          …
9573          1            0                   0            0
9574          1            0                   0            0
9575          0            0                   1            0
9576          0            0                   0            0
9577          0            0                   1            0

      home_improvement  major_purchase  small_business
0                    0               0               0
1                    0               0               0
2                    0               0               0
3                    0               0               0
4                    0               0               0
…                  …             …              …
9573                 0               0               0
9574                 0               0               0
9575                 0               0               0
9576                 1               0               0
9577                 0               0               0

[9578 rows x 7 columns]
```

[414]:
```python
#concatenate the dummies DataFrame with the original one
dataSetRead = pd.concat([dataSetRead,cat_variable],axis=1)
# Drop the original 'purpose' column as it's now redundant
dataSetRead_New = dataSetRead.drop(columns = ['purpose'])
```

[415]:
```python
# Displaying the columns and their respective data types after handeling␣
↪categorical variable purpose
dataSetRead_New.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 20 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
```

```
 0   credit.policy       9578 non-null   int64
 1   int.rate            9578 non-null   float64
 2   installment         9578 non-null   float64
 3   log.annual.inc      9578 non-null   float64
 4   dti                 9578 non-null   float64
 5   fico                9578 non-null   int64
 6   days.with.cr.line   9578 non-null   float64
 7   revol.bal           9578 non-null   int64
 8   revol.util          9578 non-null   float64
 9   inq.last.6mths      9578 non-null   int64
 10  delinq.2yrs         9578 non-null   int64
 11  pub.rec             9578 non-null   int64
 12  not.fully.paid      9578 non-null   int64
 13  all_other           9578 non-null   int32
 14  credit_card         9578 non-null   int32
 15  debt_consolidation  9578 non-null   int32
 16  educational         9578 non-null   int32
 17  home_improvement    9578 non-null   int32
 18  major_purchase      9578 non-null   int32
 19  small_business      9578 non-null   int32
dtypes: float64(6), int32(7), int64(7)
memory usage: 1.2 MB
```

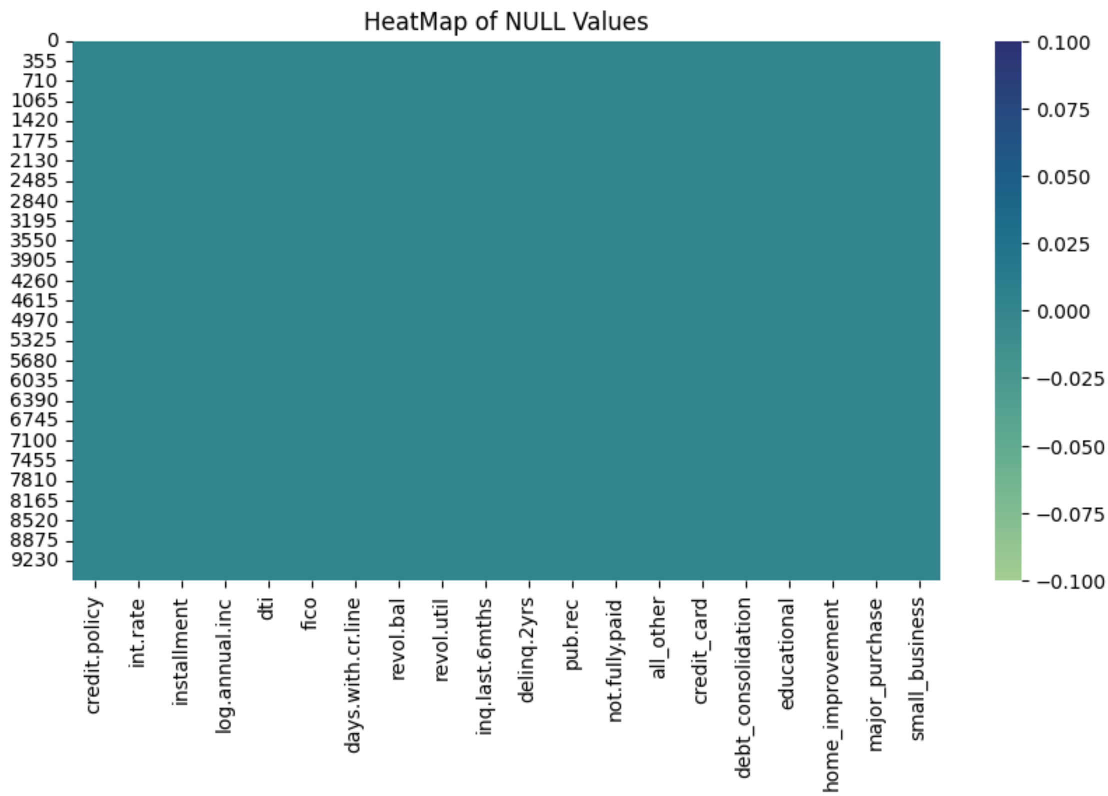
3 Q-2: Print a heatmap to check NULL values. 1M

[416]:
```
# Checking total no. of NULL values for attributes specific
missingValue_Count=dataSetRead_New.isnull().sum()
print(missingValue_Count)
```

```
credit.policy         0
int.rate              0
installment           0
log.annual.inc        0
dti                   0
fico                  0
days.with.cr.line     0
revol.bal             0
revol.util            0
inq.last.6mths        0
delinq.2yrs           0
pub.rec               0
not.fully.paid        0
all_other             0
credit_card           0
debt_consolidation    0
educational           0
home_improvement      0
```

```
major_purchase        0
small_business        0
dtype: int64
```

[417]:
```python
# Import required packages
import matplotlib.pyplot as plt
import seaborn as sbn
plt.figure(figsize=(10,5))
sbn.heatmap(dataSetRead_New.isnull(),cmap="crest")
plt.title('HeatMap of NULL Values')
plt.show()
```



[418]:
```python
# Checking duplicate values in dataSet
duplicatevalue_Count=dataSetRead_New.duplicated().sum()
print("Total duplicates values in dataSet:- {}".format(duplicatevalue_Count))
```

```
Total duplicates values in dataSet:- 0
```

[419]:
```python
# Checking percentagewise distiribution for "not.fully.paid" target variable
dataSetRead_New['not.fully.paid'].value_counts(normalize=True).mul(100).round(2)
```

[419]:
```
not.fully.paid
0    83.99
1    16.01
Name: proportion, dtype: float64
```

[420]:
```python
# Cheking countwise distribution for "not.fully.paid" target variable
dataSetRead_New['not.fully.paid'].value_counts()
```

[420]:
```
not.fully.paid
0    8045
1    1533
Name: count, dtype: int64
```

Analysis:- Above distribution operation shows that dataset is unbalanced hence SMOTE technique need to be implemented for over sampling

3.1 Split data in to features & target

[421]:
```python
# Dropping target variable
X=dataSetRead_New.drop('not.fully.paid',axis=1)
# Taking target variavle
y=dataSetRead_New['not.fully.paid']
```

[422]:
```python
# Importing SMOTE module from imblearn library
from imblearn.over_sampling import SMOTE
# Importing Counter module from collections library
from collections import Counter
counter = Counter(y)
print('Before oversampling', counter)
# Oversampling the dataset using SMOTE
smt = SMOTE(random_state = 2)
X_res, y_res = smt.fit_resample(X, y.ravel())
counter = Counter(y_res)
print('After oversampling', counter)
```

```
Before oversampling Counter({0: 8045, 1: 1533})
After oversampling Counter({0: 8045, 1: 8045})
```

4 Q-3: Perform stratified splitting of train and test data. 1M

4.1 Spliting the Data into Training and Testing Sets

[423]:
```
# Importing train_test_split package
from sklearn.model_selection import train_test_split
# Splitting train & test data
X_train,X_test,y_train,y_test=train_test_split(X_res,y_res,test_size=0.
↪2,random_state=42,stratify=y_res)
```

[424]:
```
# Displaying dimenstion of train & test dataset
print('Shape of X_train = ', X_train.shape)
print('Shape of y_train = ', y_train.shape)
print('Shape of X_test = ', X_test.shape)
print('Shape of y_test = ', y_test.shape)
```

```
Shape of X_train =  (12872, 19)
Shape of y_train =  (12872,)
Shape of X_test =  (3218, 19)
Shape of y_test =  (3218,)
```

[425]:
```
# Converting a NumPy array to Pandas dataframe
y_test=pd.DataFrame(y_test,columns=['not.fully.paid'])
y_train=pd.DataFrame(y_train,columns=['not.fully.paid'])
```

[426]:
```
# Checking distiribution for "credit.policy" target variable in y_test
y_test.value_counts()
```

[426]:
```
not.fully.paid
0                 1609
1                 1609
Name: count, dtype: int64
```

[427]:
```
# Checking distiribution for "credit.policy" target variable in y_train
y_train.value_counts()
```

[427]:
```
not.fully.paid
0                 6436
1                 6436
Name: count, dtype: int64
```

4.2 Data scaling

[428]:
```
# importing required package
# MinMax Scaler is used to perform feature scalling
from sklearn.preprocessing import MinMaxScaler
scalling=MinMaxScaler()
scalling.fit(X_train)
```

[428]:
```
MinMaxScaler()
```

[429]:
```
X_train_scalling=scalling.transform(X_train)
X_test_scalling=scalling.transform(X_test)
```

5 Q-4: Build a classifier model using the Decision Tree algorithm. 2M

[430]:
```
# Importing library of DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
# Creating instance of DecisionTreeClassifier
Classifiermodel_DTC_beforePruning = DecisionTreeClassifier()
```

[431]:
```
# Training the DecisionTreeClassifier model using training data set
Classifiermodel_DTC_beforePruning.fit(X_train_scalling, y_train)
```

[431]:
```
DecisionTreeClassifier()
```

6 Q-5: Print confusion matrix and classification report before and after pruning the Decision tree. (1+1)M

6.1 Printing confusion matrix and classification report before pruning

[432]:
```
# Importing Library of classification_report, accuracy_score
from sklearn.metrics import classification_report, accuracy_score
# Importing required packages
from sklearn.metrics import confusion_matrix
```

[433]:
```
# Prediction on test data for DecisionTreeClassifier model
y_prediction_DTC_beforePruning=Classifiermodel_DTC_beforePruning.
↪predict(X_test_scalling)
```

[434]:
```
# Evaluating the accuracy of DecisionTreeClassifier model
accuracy_DTC_beforePruning=accuracy_score(y_test,y_prediction_DTC_beforePruning)
print("DecisionTreeClassifier model accuracy before pruning: {}".
↪format(round(accuracy_DTC_beforePruning,2)))
```

```
DecisionTreeClassifier model accuracy before pruning: 0.8
```

[435]:
```
# Evaluating classfication report for DecisionTreeClassifier model
report_DCT_beforePruning=classification_report(y_test,y_prediction_DTC_beforePruning)
print(report_DCT_beforePruning)
```

```
              precision    recall  f1-score   support

           0       0.80      0.79      0.80      1609
           1       0.79      0.81      0.80      1609
```

```
    accuracy                           0.80      3218
   macro avg       0.80      0.80      0.80      3218
weighted avg       0.80      0.80      0.80      3218
```

[436]:
```
# Evaluating confusion matrix for DecisionTreeClassifier model
conf_matrix_DCT_beforePruning=confusion_matrix(y_test,y_prediction_DTC_beforePruning)
print(conf_matrix_DCT_beforePruning)
```

```
[[1269  340]
 [ 309 1300]]
```

[461]:
```
falsePositive = conf_matrix_DCT_beforePruning.sum(axis=0) - np.
 ↪diag(conf_matrix_DCT_beforePruning)
falseNegative = conf_matrix_DCT_beforePruning.sum(axis=1) - np.
 ↪diag(conf_matrix_DCT_beforePruning)
truePositive = np.diag(conf_matrix_DCT_beforePruning)
trueNegative = conf_matrix_DCT_beforePruning.sum() - (falsePositive +␣
 ↪falseNegative + truePositive)
print('************* DecisionTreeClassifier model before pruning *************')
for i in range(len(truePositive)):
    print(f"Class {i}:")
    print(f"truePositive: {truePositive[i]}, falsePositive: {falsePositive[i]},␣
 ↪falseNegative: {falseNegative[i]}, trueNegative: {trueNegative[i]}")
    print()
```

```
************* DecisionTreeClassifier model before pruning *************
Class 0:
truePositive: 1269, falsePositive: 309, falseNegative: 340, trueNegative: 1300

Class 1:
truePositive: 1300, falsePositive: 340, falseNegative: 309, trueNegative: 1269
```

6.2 Printing confusion matrix and classification report after pruning

6.3 Pre-Pruning with max_depth

[438]:
```
# Train Decision Tree with pre-pruning (using max_depth)
Classifiermodel_DTC_prePruning = DecisionTreeClassifier(random_state=42,␣
 ↪max_depth=5)   # Adjust max_depth
Classifiermodel_DTC_prePruning.fit(X_train_scalling, y_train)
```

[438]: DecisionTreeClassifier(max_depth=5, random_state=42)

[439]:
```
# Prediction on test data for DecisionTreeClassifier model with pre-pruning
y_prediction_DTC_prePruning=Classifiermodel_DTC_prePruning.
↪predict(X_test_scalling)
```

[440]:
```
# Evaluating the accuracy of DecisionTreeClassifier model with pre-pruning
accuracy_DTC_prePruning=accuracy_score(y_test,y_prediction_DTC_prePruning)
print("DecisionTreeClassifier model accuracy with pre-pruning: {}".
↪format(round(accuracy_DTC_prePruning,2)))
```

```
DecisionTreeClassifier model accuracy with pre-pruning: 0.76
```

[441]:
```
# Evaluating classfication report for DecisionTreeClassifier model with␣
↪pre-pruning
report_DCT_prePruning=classification_report(y_test,y_prediction_DTC_prePruning)
print(report_DCT_prePruning)
```

```
              precision    recall  f1-score   support

           0       0.74      0.79      0.76      1609
           1       0.77      0.73      0.75      1609

    accuracy                           0.76      3218
   macro avg       0.76      0.76      0.76      3218
weighted avg       0.76      0.76      0.76      3218
```

[442]:
```
# Evaluating confusion matrix for DecisionTreeClassifier model with pre-pruning
conf_matrix_DCT_prePruning=confusion_matrix(y_test,y_prediction_DTC_prePruning)
print(conf_matrix_DCT_prePruning)
```

```
[[1267  342]
 [ 439 1170]]
```

[443]:
```
falsePositiveprePruning = conf_matrix_DCT_prePruning.sum(axis=0) - np.
↪diag(conf_matrix_DCT_prePruning)
falseNegativeprePruning = conf_matrix_DCT_prePruning.sum(axis=1) - np.
↪diag(conf_matrix_DCT_prePruning)
truePositiveprePruning = np.diag(conf_matrix_DCT_prePruning)
trueNegativeprePruning = conf_matrix_DCT_prePruning.sum() -␣
↪(falsePositiveprePruning + falseNegativeprePruning + truePositiveprePruning)
print('************* DecisionTreeClassifier model with prePruning*************')
for i in range(len(truePositiveprePruning)):
    print(f"Class {i}:")
    print(f"truePositive: {truePositiveprePruning[i]}, falsePositive:␣
↪{falsePositiveprePruning[i]}, falseNegative: {falseNegativeprePruning[i]},␣
↪trueNegative: {trueNegativeprePruning[i]}")
    print()
```

```
************* DecisionTreeClassifier model with prePruning************
Class 0:
truePositive: 1267, falsePositive: 439, falseNegative: 342, trueNegative: 1170

Class 1:
truePositive: 1170, falsePositive: 342, falseNegative: 439, trueNegative: 1267
```

7 Cost-complexity pruning (Post-pruning)

[444]:
```
Classifiermodel_DTC_postPruning=DecisionTreeClassifier()
path = Classifiermodel_DTC_postPruning.
 ↪cost_complexity_pruning_path(X_train_scalling, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

[445]:
```
# Train a series of decision trees with different alpha values
pruned_models = []
for ccp_alpha in ccp_alphas:
    pruned_model = DecisionTreeClassifier(criterion="gini", ccp_alpha=ccp_alpha)
    pruned_model.fit(X_train_scalling, y_train)
    pruned_models.append(pruned_model)

# Find the model with the best accuracy on test data
best_accuracy = 0
best_pruned_model = None
for pruned_model in pruned_models:
    accuracy = pruned_model.score(X_test_scalling, y_test)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_pruned_model = pruned_model
# Model Accuracy after pruning
accuracy_after_pruning = best_pruned_model.score(X_test_scalling, y_test)
print("DecisionTreeClassifier model accuracy post pruning: {}".
 ↪format(round(accuracy_after_pruning,2)))
```

```
DecisionTreeClassifier model accuracy post pruning: 0.81
```

[466]:
```
# Prediction on test data for DecisionTreeClassifier model post pruning
y_prediction_DTC_postPruning=best_pruned_model.predict(X_test_scalling)
```

[467]:
```
# Displaying classfication report for DecisionTreeClassifier model post pruning
report_DCT_postPruning=classification_report(y_test,y_prediction_DTC_postPruning)
print(report_DCT_postPruning)
```

```
              precision    recall  f1-score   support

           0       0.80      0.84      0.82      1609
```

```
           1       0.83      0.79      0.81      1609

    accuracy                           0.81      3218
   macro avg       0.81      0.81      0.81      3218
weighted avg       0.81      0.81      0.81      3218
```

[468]:
```
# Evaluating confusion matrix for DecisionTreeClassifier model with post pruning
conf_matrix_DCT_postPruning=confusion_matrix(y_test,y_prediction_DTC_postPruning)
print(conf_matrix_DCT_postPruning)
```

```
[[1347  262]
 [ 337 1272]]
```

[469]:
```
falsePositivepostPruning = conf_matrix_DCT_postPruning.sum(axis=0) - np.
 ↪diag(conf_matrix_DCT_postPruning)
falseNegativepostPruning = conf_matrix_DCT_postPruning.sum(axis=1) - np.
 ↪diag(conf_matrix_DCT_postPruning)
truePositivepostPruning = np.diag(conf_matrix_DCT_postPruning)
trueNegativepostPruning = conf_matrix_DCT_postPruning.sum() -␣
 ↪(falsePositivepostPruning + falseNegativepostPruning +␣
 ↪truePositivepostPruning)
print('************* DecisionTreeClassifier model with Post␣
 ↪Pruning*************')
for i in range(len(truePositivepostPruning)):
    print(f"Class {i}:")
    print(f"truePositive: {truePositivepostPruning[i]}, falsePositive:␣
 ↪{falsePositivepostPruning[i]}, falseNegative: {falseNegativepostPruning[i]},␣
 ↪trueNegative: {trueNegativepostPruning[i]}")
    print()
```

```
************* DecisionTreeClassifier model with Post Pruning*************
Class 0:
truePositive: 1347, falsePositive: 337, falseNegative: 262, trueNegative: 1272

Class 1:
truePositive: 1272, falsePositive: 262, falseNegative: 337, trueNegative: 1347
```

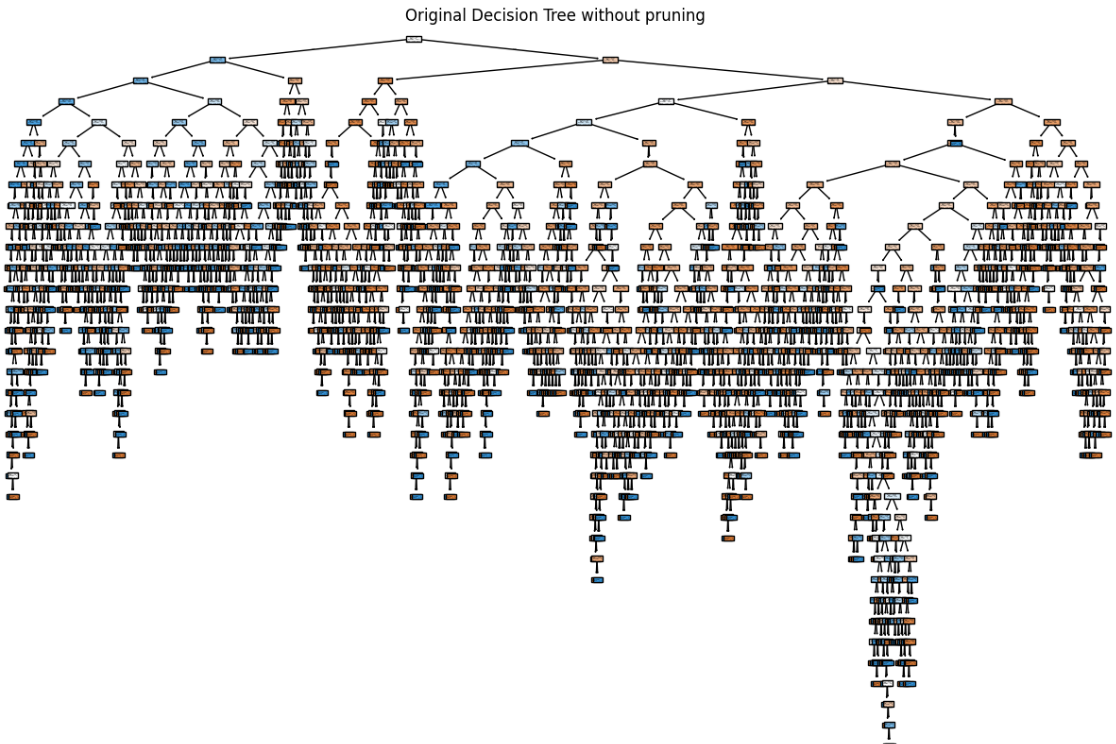
7.0.1 Key Considerations:

Pre-pruning is easier to apply and more straightforward but may result in underfitting if the tree is pruned too early.

Post-pruning offers a more refined approach by growing the tree fully and then trimming unnecessary branches, but it requires more effort, such as determining the optimal value of ccp_alpha through cross-validation.

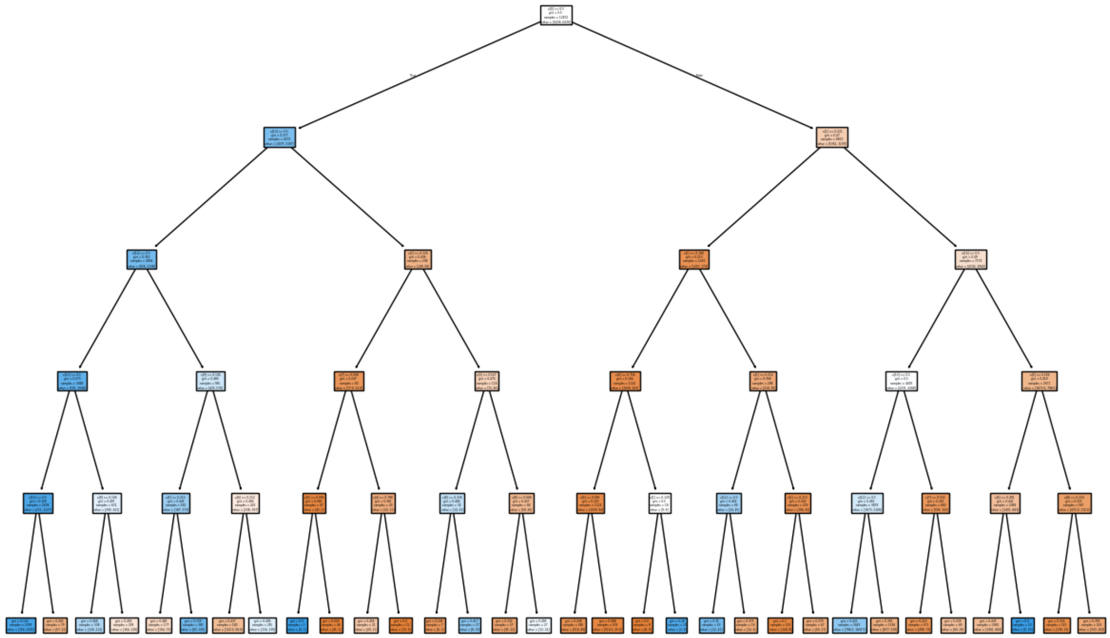
8 Q-6 Plot the final decision tree model. 1M

[470]:
```
from sklearn.tree import plot_tree
# Plot original tree
plt.figure(figsize=(15, 10))
plot_tree(Classifiermodel_DTC_beforePruning, filled=True)
plt.title("Original Decision Tree without pruning ")
plt.show()
```

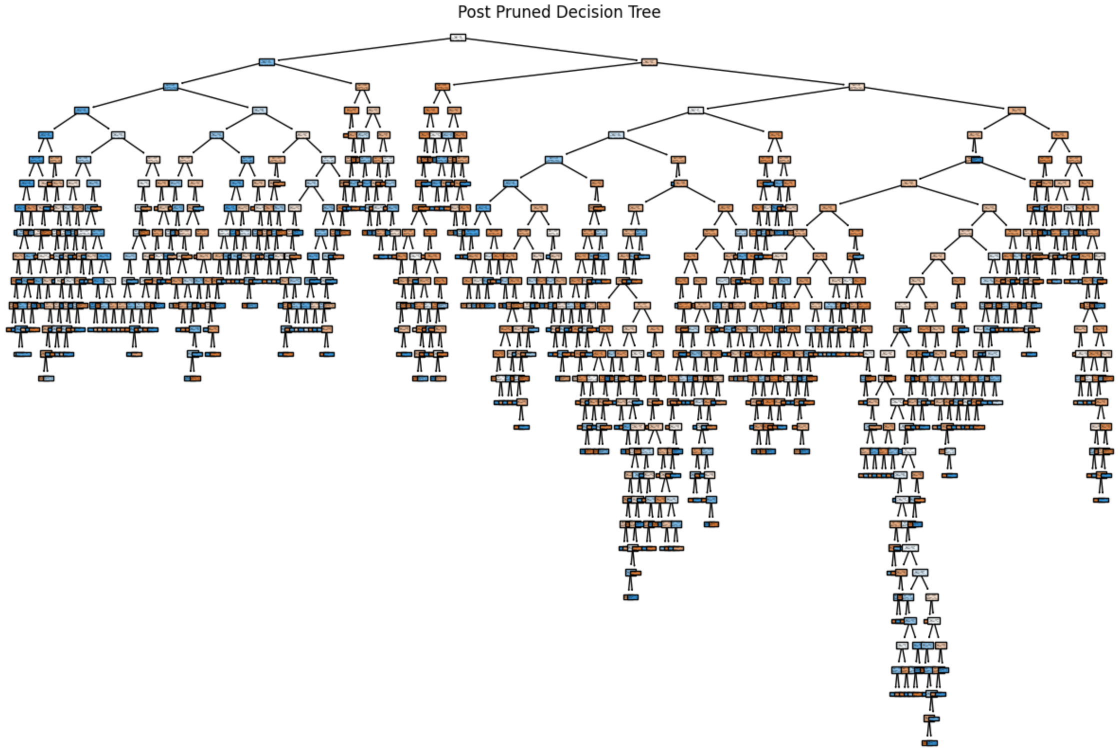


[471]:
```
# Plot pre pruned tree
plt.figure(figsize=(15, 10))
plot_tree(Classifiermodel_DTC_prePruning, filled=True)
plt.title("Pre Pruned Decision Tree")
plt.show()
```

Pre Pruned Decision Tree



[472]:
```
# Plot post pruned tree
plt.figure(figsize=(15, 10))
plot_tree( best_pruned_model, filled=True)
plt.title("Post Pruned Decision Tree")
plt.show()
```



9 Q-7: Find out the stratified cross-validation accuracy 1M

[473]:
```
# Importing required package
from sklearn.model_selection import cross_val_score, StratifiedKFold
```

[474]:
```
# Initialize the Decision Tree Classifier
Classifier_DST = DecisionTreeClassifier(random_state=42)

# Perform Stratified K-Fold Cross-Validation
# Here, we use 10 folds (cv=10)
stratified_kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
cross_val_scores = cross_val_score(Classifier_DST, X_res, y_res,
    cv=stratified_kfold, scoring='accuracy')

# Print cross-validation accuracy scores and mean accuracy
print("Cross-Validation Accuracy Scores for each fold:")
print(cross_val_scores)

print("\nMean Stratified Cross-Validation Accuracy:")
print(cross_val_scores.mean())
```

```
Cross-Validation Accuracy Scores for each fold:
```

```
[0.81417029 0.80795525 0.80111871 0.79801119 0.81230578 0.82660037
 0.79552517 0.79303915 0.81852082 0.7998757 ]

Mean Stratified Cross-Validation Accuracy:
0.8067122436295836
```