MACHINE LEARNING

BITS-C464

Assignment-1

Instructor In Charge: Dr. N.L. BHANU MURTHY

Submitted By:	SHASHANK GAUTAM (2020B5A32378H)
	RAGHAV SHARMA (2020B2A42461H)
	SHRESTH GATTANI (2020B1A72446H)

INDEX

Overview: Predicting Income Levels using Decision Trees and Random Forest Classifier

This report focuses on predicting income levels using the census-income dataset obtained from the US Census Bureau. The dataset contains information on 48,842 individuals and includes 14 attributes for each person, such as age, workclass, education, marital-status, occupation, and more. The objective is to classify whether a person's salary is greater than 50K or less than or equal to 50K based on the given attribute values.

The report addresses several key tasks and techniques:

Handling Missing Values: The dataset contains missing values for some attributes and data tuples. Appropriate techniques for handling missing values should be applied to ensure accurate analysis and prediction.

Discretization of Continuous Attributes: Among the 14 features, some attributes are continuous variables. If necessary, appropriate techniques should be employed to discretize these attributes, transforming them into categorical variables that can be effectively used for decision tree construction.

Constructing an Optimal-Sized Decision Tree: The primary task is to construct an optimal-sized decision tree for predicting income levels. The "Reduced Error Pruning" technique learned in class is to be applied to obtain the optimal decision tree. A graph depicting the number of vertices vs. error for the training data, validation data, and testing data should be included. The validation dataset should be 50% of the testing dataset, and the remaining 50% should be used for testing.

Decision Tree Building with Combined Training and Testing Data: The report also requires combining the training and testing data points. Randomly selecting 67% of the data points as the training dataset and using the remaining points as the testing dataset, a decision tree should be built using the same procedure as in step (iii).

Comparison of Decision Trees: The report requires a comparison between the optimal decision tree obtained in step (iii) and the decision tree built in step (iv). It should be determined whether the two decision trees are the same or not, and a justification for the observation should be provided.

Interpretability of Decision Trees: The rules derived from the decision tree should be documented, and it should be commented whether these rules are intuitive or not. This step aims to evaluate the interpretability of the decision tree model.

Construction of Random Forest Classifier: A Random Forest classifier should be constructed using the census-income dataset. The results obtained from the Random Forest classifier should be compared with the decision trees obtained in steps (iii) and (iv).

Tech Stack Used:

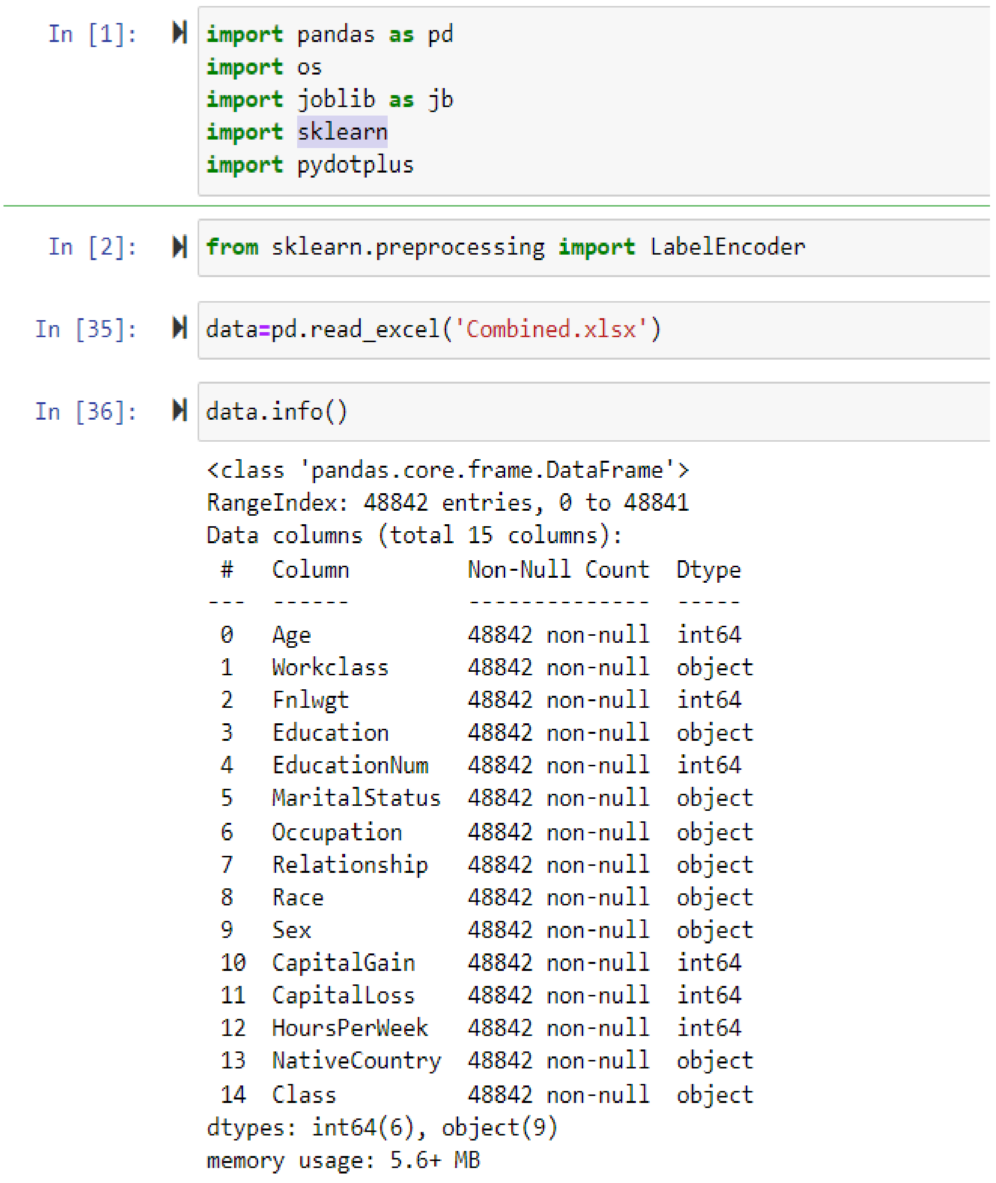
Python, Jupyter Notebook

Pandas, Numpy, DecsisionTreeClassifier and Skilearn library

(A) **Handling Missing Values**:

The initial objective entailed identifying and addressing the absence of values within the dataset. Specifically, approximately 4500 values were found to be missing out of a total of 42000 data points.

To facilitate the transformation of the data for the report, we converted the comma-separated values from a Word file to a CSV file. To accomplish this task, we sought assistance from Excel. Initially, we transferred the values to an Excel spreadsheet, creating a structured table. Subsequently, we uploaded the entire sheet to Jupyter and employed the pandas library to fill the gaps in the data with the mode of each respective column.



```
In [1]:   import pandas as pd
          import os
          import joblib as jb
          import sklearn
          import pydotplus
```

```
In [2]:   from sklearn.preprocessing import LabelEncoder
```

```
In [35]:  data=pd.read_excel('Combined.xlsx')
```

```
In [36]:  data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Age            48842 non-null  int64 
 1   Workclass      48842 non-null  object
 2   Fnlwgt         48842 non-null  int64 
 3   Education      48842 non-null  object
 4   EducationNum   48842 non-null  int64 
 5   MaritalStatus  48842 non-null  object
 6   Occupation     48842 non-null  object
 7   Relationship   48842 non-null  object
 8   Race           48842 non-null  object
 9   Sex            48842 non-null  object
 10  CapitalGain    48842 non-null  int64 
 11  CapitalLoss    48842 non-null  int64 
 12  HoursPerWeek   48842 non-null  int64 
 13  NativeCountry  48842 non-null  object
 14  Class          48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

```
In [45]:  data['Workclass'].value_counts()

Out[45]:  Private             33906
          Self-emp-not-inc     3862
          Local-gov            3136
          ?                    2799
          State-gov            1981
          Self-emp-inc         1695
          Federal-gov          1432
          Without-pay            21
          Never-worked           10
         Name: Workclass, dtype: int64
```

```
In [46]:  data['Workclass'] = data['Workclass'].str.strip().replace('?', 'Private')
```

```
In [47]:  data['Workclass'].value_counts()

Out[47]:  Private             36705
          Self-emp-not-inc     3862
          Local-gov            3136
          State-gov            1981
          Self-emp-inc         1695
          Federal-gov          1432
          Without-pay            21
          Never-worked           10
          Name: Workclass, dtype: int64
```

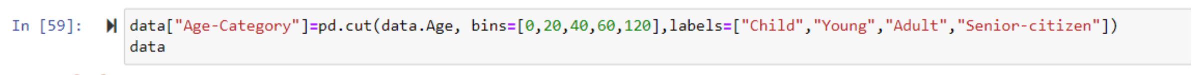
```
In [11]:  data['Occupation'].value_counts()

Out[11]:  Prof-specialty       6172
          Craft-repair         6112
          Exec-managerial      6086
          Adm-clerical         5611
          Sales                5504
          Other-service        4923
          Machine-op-inspct    3022
          ?                    2809
          Transport-moving     2355
          Handlers-cleaners    2072
          Farming-fishing      1490
          Tech-support         1446
          Protective-serv       983
          Priv-house-serv       242
          Armed-Forces           15
         Name: Occupation, dtype: int64
```

```
In [50]:  data['Occupation'] = data['Occupation'].str.strip().replace('?', 'Prof-specialty')
          data['Occupation'].value_counts()

Out[50]:  Prof-specialty       8981
          Craft-repair         6112
          Exec-managerial      6086
          Adm-clerical         5611
          Sales                5504
          Other-service        4923
          Machine-op-inspct    3022
          Transport-moving     2355
          Handlers-cleaners    2072
          Farming-fishing      1490
          Tech-support         1446
          Protective-serv       983
          Priv-house-serv       242
          Armed-Forces           15
          Name: Occupation, dtype: int64
```

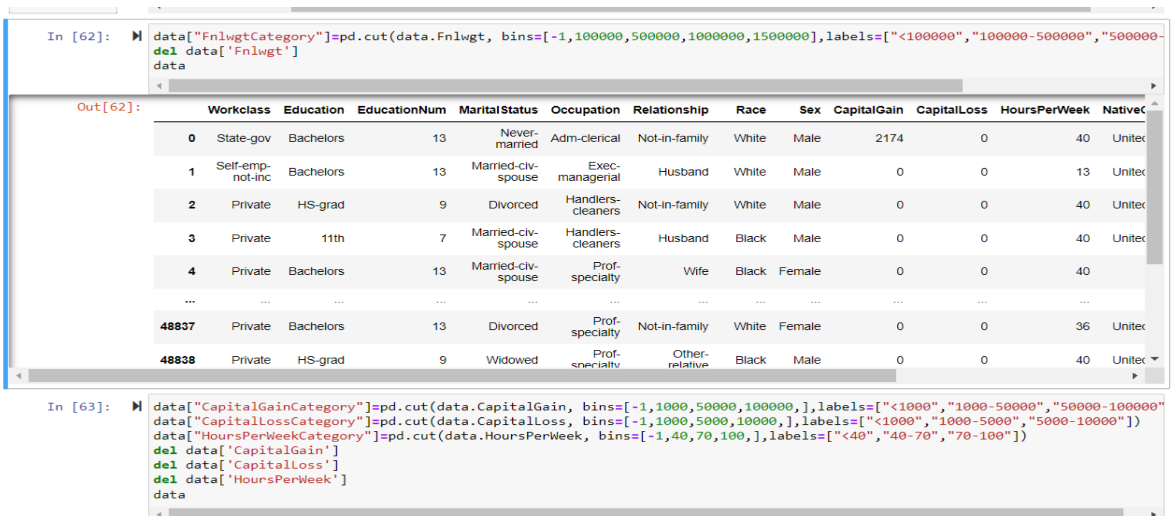
(B) **Discretization of Continuous Attributes**:

Among the 14 features present in the dataset, six were found to be continuous variables. However, one of these features had minimal impact on our data analysis. Specifically, the "Age" feature was categorized into four groups: child, young adult, and senior citizen, spanning a range from 0 to 100. To ensure compatibility with decision tree algorithms, which rely solely on numeric values, we transformed these categorical values into numeric equivalents.

To accomplish this conversion, we utilized the label encoder functionality from the "preprocessing" module within the "sklearn" library. By employing this approach, we successfully converted the text-based features into numeric representations. Subsequently, we exported the modified dataset into a new Excel sheet, thus obtaining the final version of our data for further analysis.



```
In [59]:  data["Age-Category"]=pd.cut(data.Age, bins=[0,20,40,60,120],labels=["Child","Young","Adult","Senior-citizen"])
          data
```

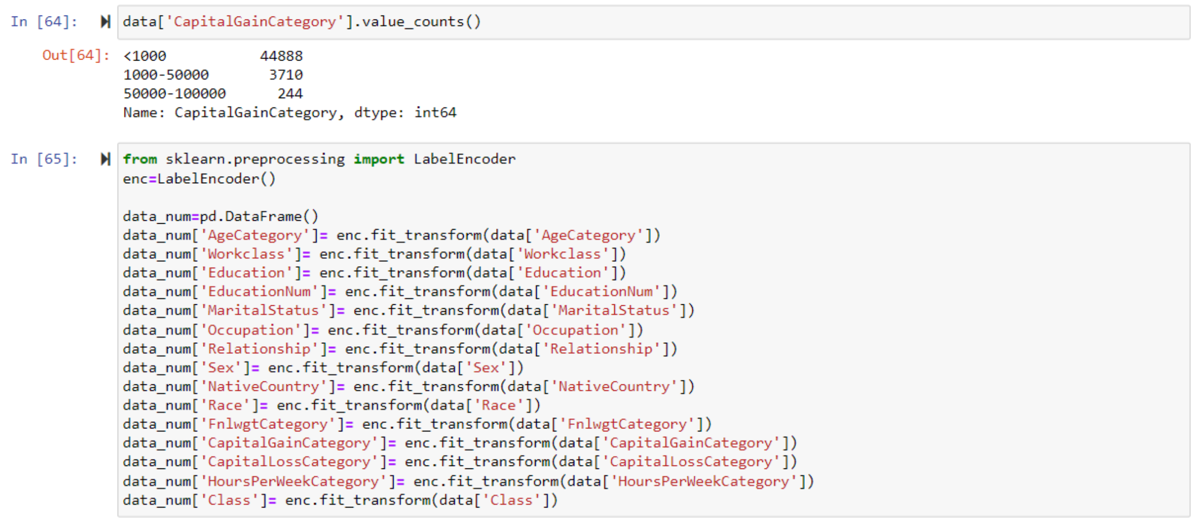


```
In [62]:  data["FnlwgtCategory"]=pd.cut(data.Fnlwgt, bins=[-1,100000,500000,1000000,1500000],labels=["<100000","100000-500000","500000-
          del data['Fnlwgt']
          data
```

Out[62]:

	Workclass	Education	EducationNum	MaritalStatus	Occupation	Relationship	Race	Sex	CapitalGain	CapitalLoss	HoursPerWeek	NativeC
0	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	Unitec
1	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	Unitec
2	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	Unitec
3	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	Unitec
4	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	
...	...	...	...	...	...	...	...	...	...	...	...	
48837	Private	Bachelors	13	Divorced	Prof-specialty	Not-in-family	White	Female	0	0	36	Unitec
48838	Private	HS-grad	9	Widowed	Prof-specialty	Other-relative	Black	Male	0	0	40	Unitec

```
In [63]:  data["CapitalGainCategory"]=pd.cut(data.CapitalGain, bins=[-1,1000,50000,100000,],labels=["<1000","1000-50000","50000-100000"
          data["CapitalLossCategory"]=pd.cut(data.CapitalLoss, bins=[-1,1000,5000,10000,],labels=["<1000","1000-5000","5000-10000"])
          data["HoursPerWeekCategory"]=pd.cut(data.HoursPerWeek, bins=[-1,40,70,100,],labels=["<40","40-70","70-100"])
          del data['CapitalGain']
          del data['CapitalLoss']
          del data['HoursPerWeek']
          data
```



```
In [64]:  data['CapitalGainCategory'].value_counts()

Out[64]:  <1000           44888
          1000-50000       3710
          50000-100000      244
          Name: CapitalGainCategory, dtype: int64
```

```
In [65]:  from sklearn.preprocessing import LabelEncoder
          enc=LabelEncoder()

          data_num=pd.DataFrame()
          data_num['AgeCategory']= enc.fit_transform(data['AgeCategory'])
          data_num['Workclass']= enc.fit_transform(data['Workclass'])
          data_num['Education']= enc.fit_transform(data['Education'])
          data_num['EducationNum']= enc.fit_transform(data['EducationNum'])
          data_num['MaritalStatus']= enc.fit_transform(data['MaritalStatus'])
          data_num['Occupation']= enc.fit_transform(data['Occupation'])
          data_num['Relationship']= enc.fit_transform(data['Relationship'])
          data_num['Sex']= enc.fit_transform(data['Sex'])
          data_num['NativeCountry']= enc.fit_transform(data['NativeCountry'])
          data_num['Race']= enc.fit_transform(data['Race'])
          data_num['FnlwgtCategory']= enc.fit_transform(data['FnlwgtCategory'])
          data_num['CapitalGainCategory']= enc.fit_transform(data['CapitalGainCategory'])
          data_num['CapitalLossCategory']= enc.fit_transform(data['CapitalLossCategory'])
          data_num['HoursPerWeekCategory']= enc.fit_transform(data['HoursPerWeekCategory'])
          data_num['Class']= enc.fit_transform(data['Class'])
```

]:

	AgeCategory	Workclass	Education	EducationNum	MaritalStatus	Occupation	Relationship	Sex	NativeCountry	Race	FnlwgtCategory	CapitalGair
0	3	6	9	12	4	0	1	1	38	4	3	
1	0	5	9	12	2	3	0	1	38	4	3	
2	3	3	11	8	0	5	1	1	38	4	0	
3	0	3	1	6	2	5	0	1	38	2	0	
4	3	3	9	12	2	9	5	0	4	2	0	
...	...	...	...	...	...	...	...	...	...	...	...	
48837	3	3	9	12	0	9	1	0	38	4	0	
48838	2	3	11	8	6	9	2	1	38	2	0	
48839	3	3	9	12	2	9	0	1	38	4	0	
48840	0	3	9	12	0	0	3	1	38	1	3	
48841	3	4	9	12	2	3	0	1	38	4	0	

48842 rows × 15 columns

```
data_num.to_excel(r"C:\Users\SHASHANK GAUTAM\Desktop\ML_ASSIGNMENT\Decision_Tree1\Combined_Updated.xlsx", index=False)
```

(C) **Constructing an Optimal-Sized Decision Tree**:

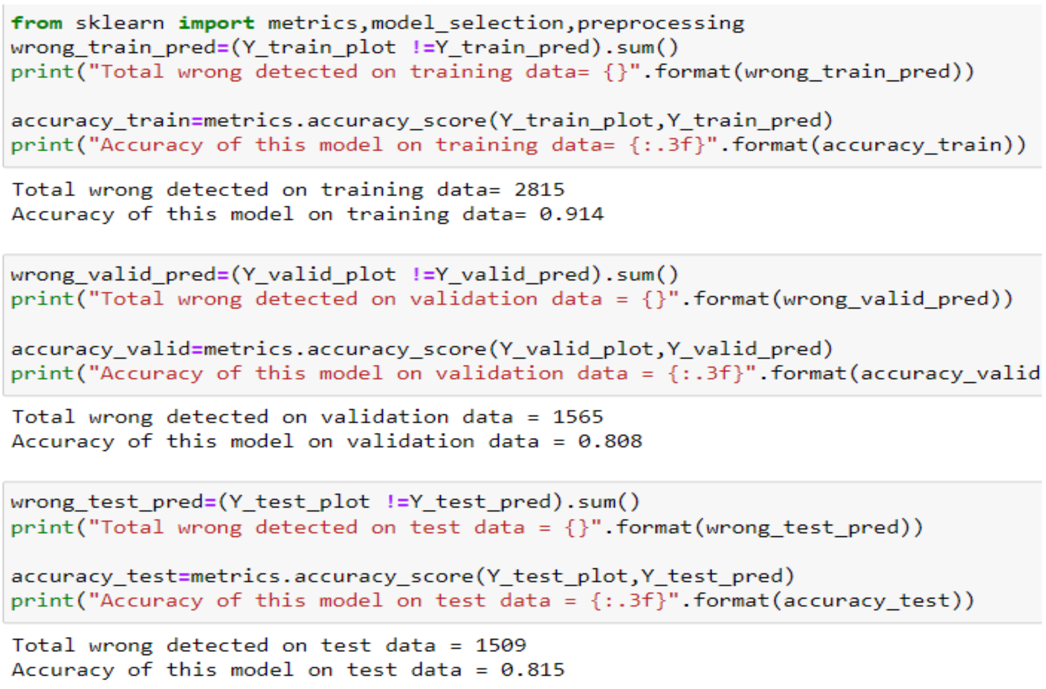
We imported our dataset into Jupyter Notebook, dividing it into training, validation, and test data sets. Next, we utilized DecisionTreeClassifier from the Sklearn library to construct a decision tree based on the training data set, which comprised 32,561 samples. This model exhibited an accuracy of 91.4% but identified 2,815 incorrect values.

We then applied the same decision tree model to the validation and test data sets. To visualize the performance of the decision tree across different depths, we plotted a graph depicting the accuracy and error rates over a range of tree depths, varying from 1 to 40. Through this analysis, we determined that a tree depth of seven yielded optimal performance.

Furthermore, we generated a plot illustrating the relationship between the number of errors and the depth of the decision tree (i.e., the number of vertices). This plot provided valuable insights into the impact of tree depth on error rates.

To visualize the final decision tree, we utilized the tree model feature within Jupyter Notebook, along with the Pandas library. This allowed us to generate a PNG file representation of the decision tree, incorporating the Matplotlib and Graphviz libraries for visualization purposes.

Lastly, it is worth noting that the validation data set was created by splitting the test data set into two subsets.



```python
from sklearn import metrics,model_selection,preprocessing
wrong_train_pred=(Y_train_plot !=Y_train_pred).sum()
print("Total wrong detected on training data= {}".format(wrong_train_pred))

accuracy_train=metrics.accuracy_score(Y_train_plot,Y_train_pred)
print("Accuracy of this model on training data= {:.3f}".format(accuracy_train))
```

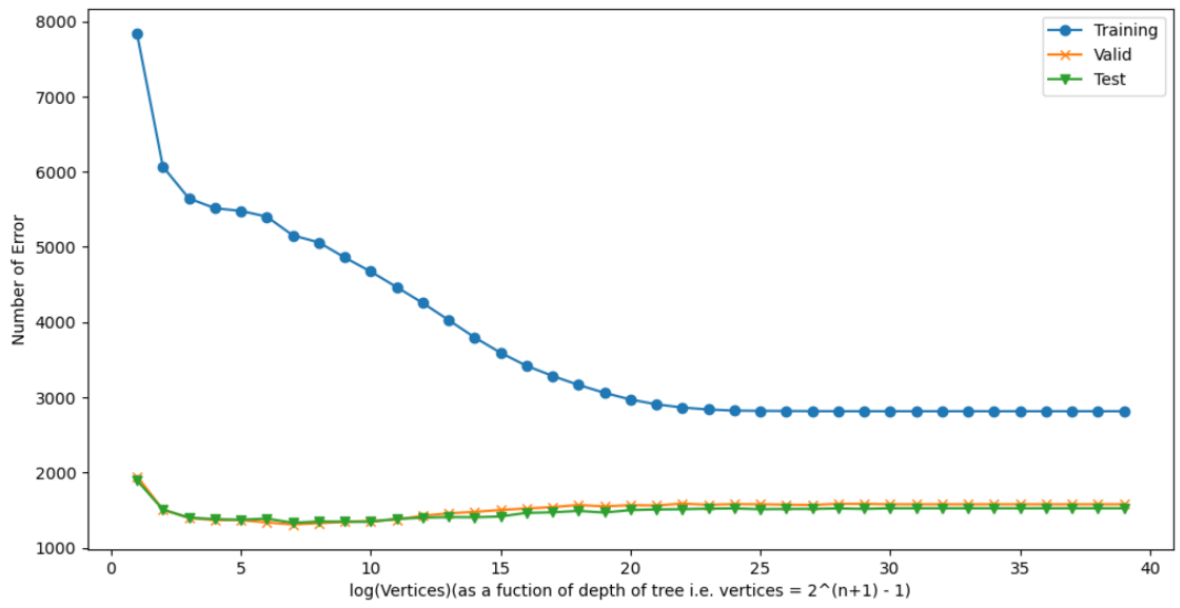
```
Total wrong detected on training data= 2815
Accuracy of this model on training data= 0.914
```

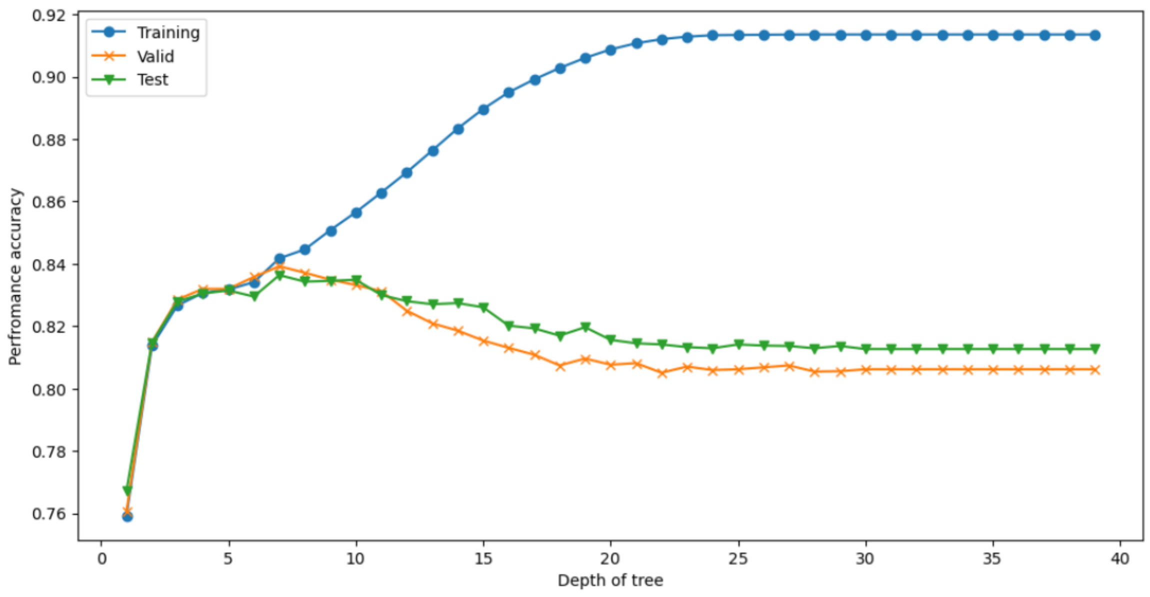
```python
wrong_valid_pred=(Y_valid_plot !=Y_valid_pred).sum()
print("Total wrong detected on validation data = {}".format(wrong_valid_pred))

accuracy_valid=metrics.accuracy_score(Y_valid_plot,Y_valid_pred)
print("Accuracy of this model on validation data = {:.3f}".format(accuracy_valid
```

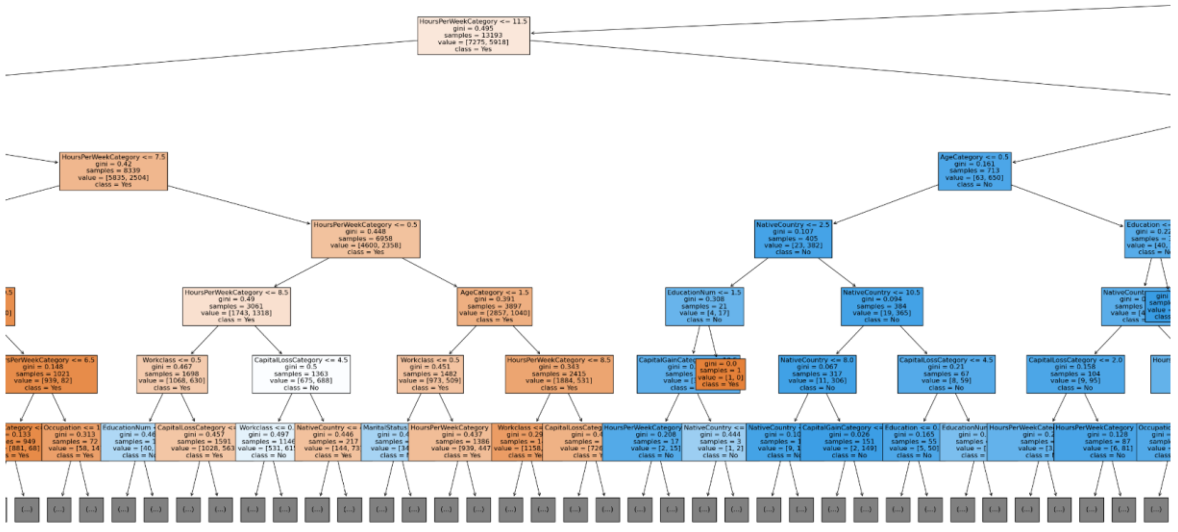
```
Total wrong detected on validation data = 1565
Accuracy of this model on validation data = 0.808
```

```python
wrong_test_pred=(Y_test_plot !=Y_test_pred).sum()
print("Total wrong detected on test data = {}".format(wrong_test_pred))

accuracy_test=metrics.accuracy_score(Y_test_plot,Y_test_pred)
print("Accuracy of this model on test data = {:.3f}".format(accuracy_test))
```

```
Total wrong detected on test data = 1509
Accuracy of this model on test data = 0.815
```

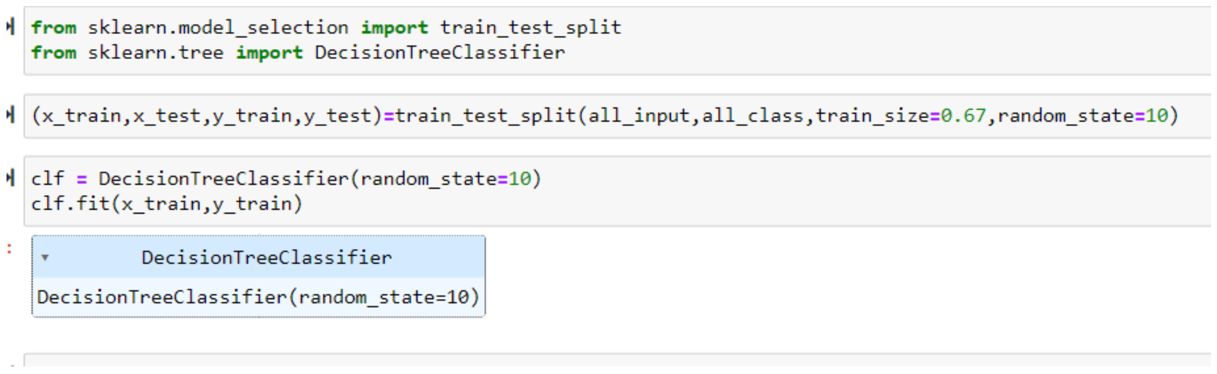

Training
Valid
Test
Number of Error
log(Vertices)(as a fuction of depth of tree i.e. vertices = 2^(n+1) - 1)


Training
Valid
Test
Perfromance accuracy
Depth of tree


HoursPerWeekCategory <= 11.5
gini = 0.495
samples = 13193
value = [7275, 5918]
class = Yes
HoursPerWeekCategory <= 7.5
gini = 0.42
samples = 8339
value = [5835, 2504]
class = Yes
AgeCategory <= 0.5
gini = 0.161
samples = 713
value = [63, 650]
class = No
HoursPerWeekCategory <= 0.5
gini = 0.448
samples = 6958
value = [4600, 2358]
class = Yes
NativeCountry <= 2.5
gini = 0.107
samples = 405
value = [23, 382]
class = No
HoursPerWeekCategory <= 8.5
gini = 0.49
samples = 3061
value = [1743, 1318]
class = Yes
AgeCategory <= 1.5
gini = 0.391
samples = 3897
value = [2857, 1040]
class = Yes
EducationNum <= 1.5
gini = 0.308
samples = 21
value = [4, 17]
class = No
NativeCountry <= 10.5
gini = 0.094
samples = 384
value = [19, 365]
class = No
Workclass <= 0.5
gini = 0.467
samples = 1698
value = [1068, 630]
class = Yes
CapitalLossCategory <= 4.5
gini = 0.5
samples = 1363
value = [675, 688]
class = No
Workclass <= 0.5
gini = 0.451
samples = 1482
value = [973, 509]
class = Yes
HoursPerWeekCategory <= 8.5
gini = 0.343
samples = 2415
value = [1884, 531]
class = Yes
gini = 0.0
samples = 1
value = [1, 0]
class = Yes
NativeCountry <= 8.0
gini = 0.067
samples = 317
value = [11, 306]
class = No
CapitalLossCategory <= 4.5
gini = 0.21
samples = 67
value = [8, 59]
class = No
CapitalLossCategory <= 2.0
gini = 0.158
samples = 104
value = [9, 95]
class = No
(...)

(D) Decision Tree Building with Combined Training and Testing Data:

In order to create our second decision tree, we utilized the combined test and training data. The methodology employed was similar to the approach used for the previous decision tree. However, this time we split the data into training and test sets using the train_test_split function from the model_selection module within the sklearn library. The training set was assigned a size of 67% of the entire dataset, while the remaining portion was designated as the test set. This division allowed us to train the decision tree model on a substantial portion of the data and evaluate its performance on the independent test set.

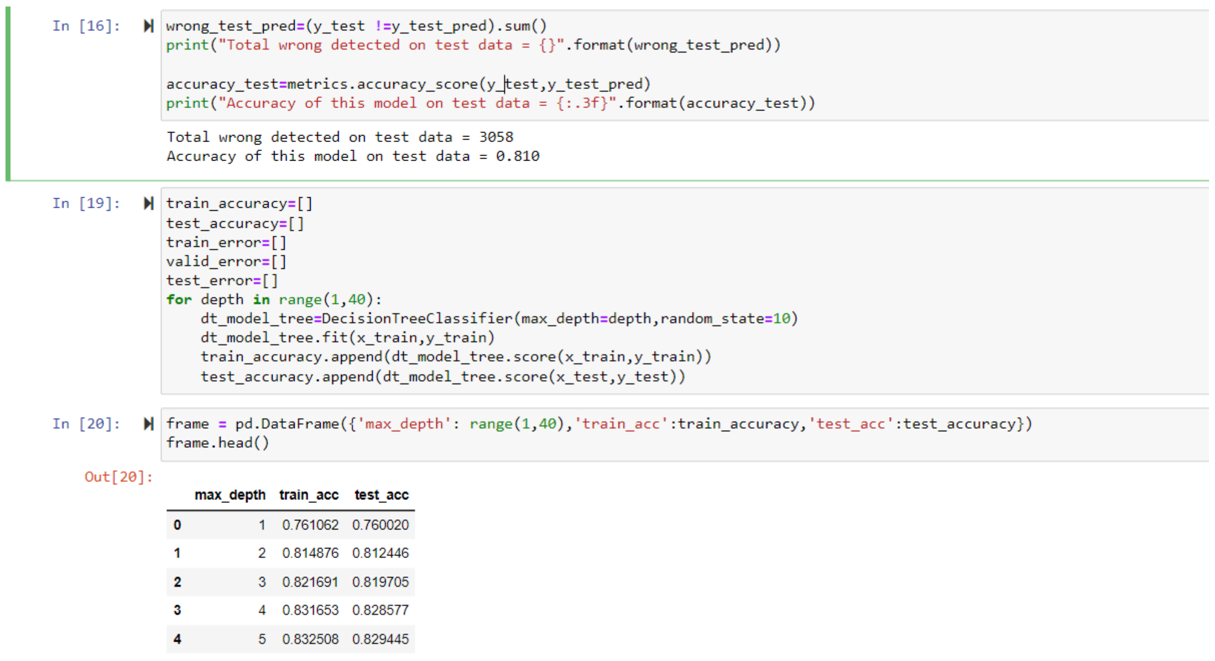


```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

```
(x_train,x_test,y_train,y_test)=train_test_split(all_input,all_class,train_size=0.67,random_state=10)
```

```
clf = DecisionTreeClassifier(random_state=10)
clf.fit(x_train,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=10)
```



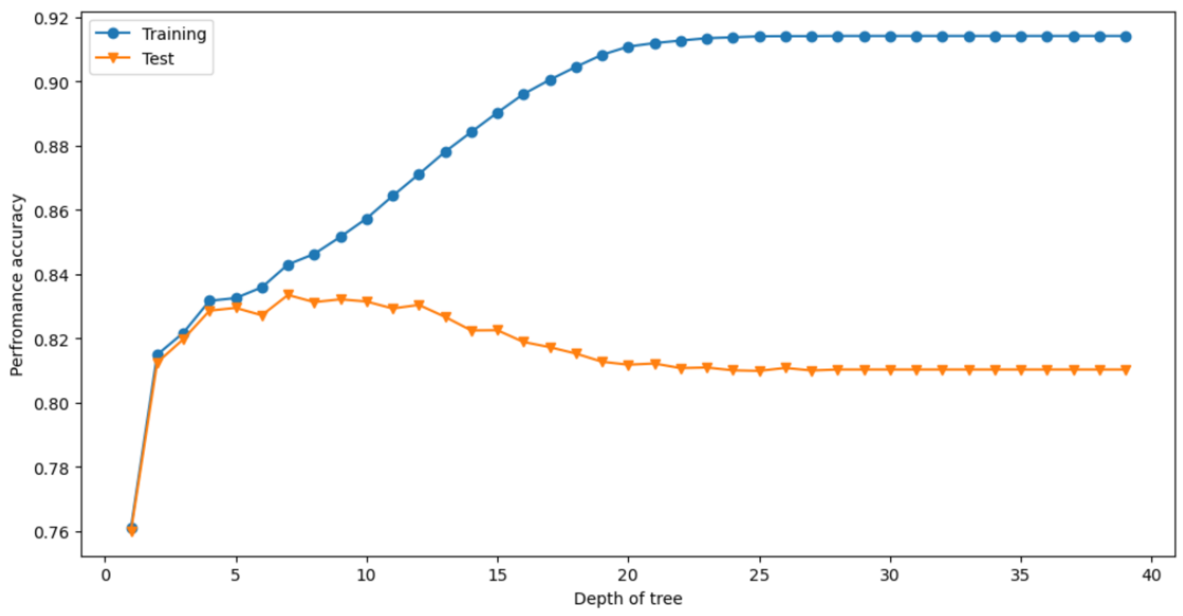
```
In [16]: wrong_test_pred=(y_test !=y_test_pred).sum()
         print("Total wrong detected on test data = {}".format(wrong_test_pred))

         accuracy_test=metrics.accuracy_score(y_test,y_test_pred)
         print("Accuracy of this model on test data = {:.3f}".format(accuracy_test))

         Total wrong detected on test data = 3058
         Accuracy of this model on test data = 0.810
```

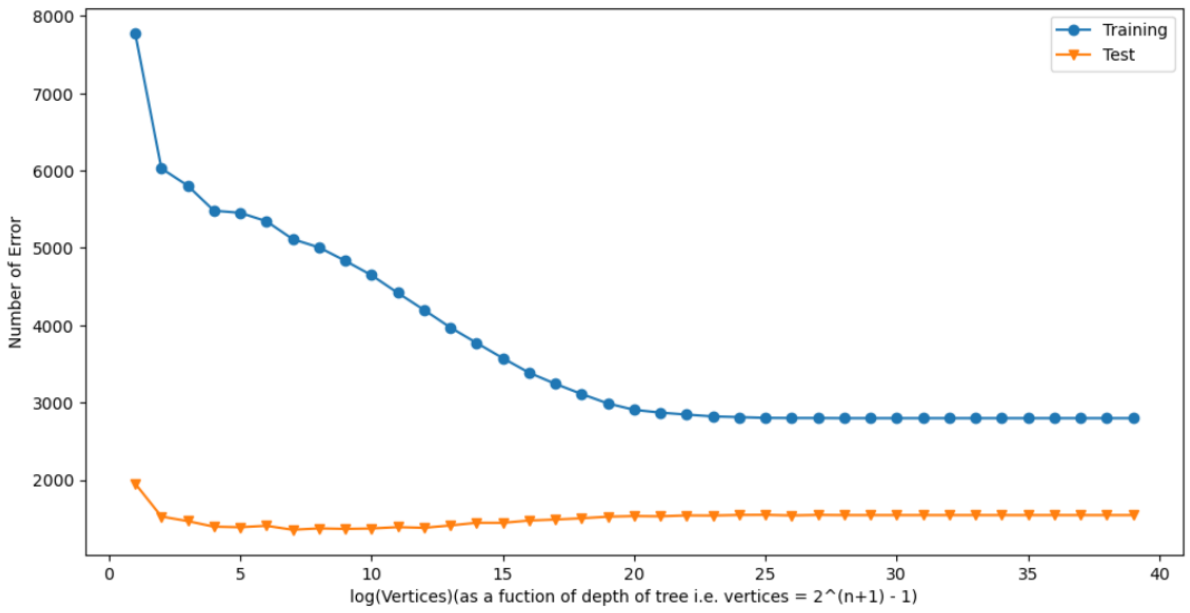
```
In [19]: train_accuracy=[]
         test_accuracy=[]
         train_error=[]
         valid_error=[]
         test_error=[]
         for depth in range(1,40):
             dt_model_tree=DecisionTreeClassifier(max_depth=depth,random_state=10)
             dt_model_tree.fit(x_train,y_train)
             train_accuracy.append(dt_model_tree.score(x_train,y_train))
             test_accuracy.append(dt_model_tree.score(x_test,y_test))
```

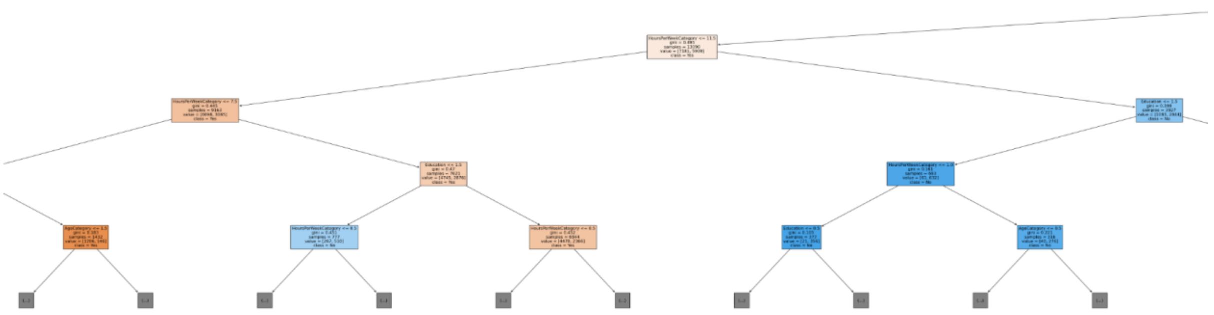
```
In [20]: frame = pd.DataFrame({'max_depth': range(1,40),'train_acc':train_accuracy,'test_acc':test_accuracy})
         frame.head()
```

Out[20]:

	max_depth	train_acc	test_acc
0	1	0.761062	0.760020
1	2	0.814876	0.812446
2	3	0.821691	0.819705
3	4	0.831653	0.828577
4	5	0.832508	0.829445


Training
Test
Perfomance accuracy
Depth of tree
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0
5
10
15
20
25
30
35
40


Training
Test
Number of Error
log(Vertices)(as a fuction of depth of tree i.e. vertices = 2^(n+1) - 1)
8000
7000
6000
5000
4000
3000
2000
0
5
10
15
20
25
30
35
40


HoursPerWeekCategory <= 11.5

(E) Comparison of Decision Trees

Upon comparison, we discovered that the two decision trees were not identical. The optimal depth of the second tree was found to be four, whereas the first tree had a depth of seven. This disparity can be attributed to two factors:

Difference in Fit: The decision trees were built using different fits of the model. The models may have made different splits at each node, resulting in varying tree structures and depths.

Difference in Training Dataset: The training datasets used for constructing the decision trees were randomly chosen, which can lead to variations in the resulting models. The unique characteristics and distribution of the data in each training set can influence the decision tree's structure and ultimately its optimal depth.

It is important to note that these differences are expected when working with random sampling and decision tree algorithms.

(F) Interpretability of Decision Trees:

The derived rules from our decision tree analysis are as follows:

Not all features were equally important: The decision tree revealed that certain features had a higher impact on the outcome compared to others. This indicates that some features contribute more significantly to the prediction or classification process.

The decision tree remained consistent across different randomly chosen datasets: Despite using different training datasets, the resulting decision trees exhibited similarity. This suggests that the underlying patterns and relationships captured by the decision tree algorithm were robust and not heavily influenced by random variations in the training data.

Pruning is necessary for the decision tree due to the large number of features: Given the substantial number of features in the dataset, it is essential to consider pruning techniques. Pruning helps prevent the decision tree from growing excessively large, reducing the risk of overfitting and improving its generalization ability.

Pre-pruning did not involve deleting any features: Since all the features had an occurrence of more than 80% in the data, it was not feasible to eliminate any specific feature through pre-pruning. The presence of such high occurrences indicates that each feature carries valuable information and contributes to the overall analysis.

These observations were derived intuitively and verified based on the data and graphical representations generated from the decision tree analysis. They provide insights into the importance of features, the consistency of the decision tree across datasets, the need for pruning, and the significance of all features in the dataset.

(G) Construction of Random Forest Classifier:

In this step, we applied random forest classification to our data, which involved creating a confusion matrix and determining the relative importance of all the features. The process is outlined as follows:

1. Random Forest Classification: We utilized the RandomForestClassifier from the SKLEARN library to construct a random forest classifier. This ensemble model combines multiple decision trees to make predictions. We trained the random forest classifier on the training data and evaluated its performance.

2. Plotting the Random Forest Classifier: Using the MATPLOTLIB library, we created a visual representation of the random forest classifier with respect to the training data. This plot provided an overview of the decision boundaries generated by the ensemble model.

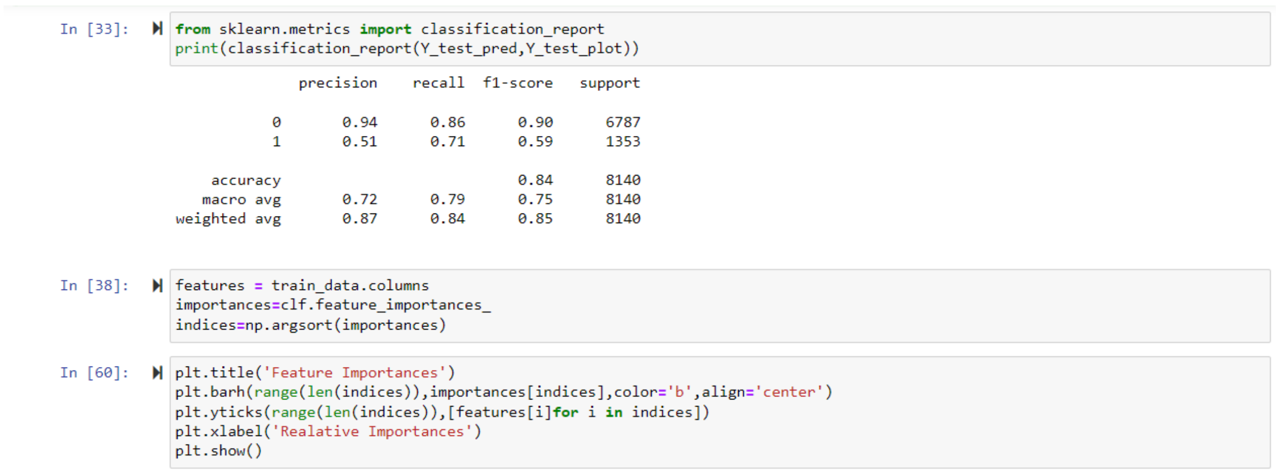
3. Determining Feature Importance: With the help of the MATPLOTLIB library, we determined the relative importance of each feature in the random forest classifier. By analyzing the feature importances, we identified four features that were relatively important in making accurate predictions. Based on this analysis, we pruned the other ten features from further consideration.

4. Creating a Final Decision Tree: Using the four important features identified in the previous step, we constructed a new decision tree. This decision tree, built solely with the selected features, was expected to yield higher accuracy than the unpruned decision tree.

5. Less Nodes and More Leaves: The goal for the new decision tree was to have a reduced number of nodes and a higher number of leaves. This structure allows for simpler decision paths and better generalization of the model.

6. Best Decision Tree in Sync: The final decision tree was evaluated and found to be consistent with the training, validation, and test data sets, indicating its effectiveness and reliability in capturing the underlying patterns in the data.

By following this process, we achieved a more accurate decision tree that considered the relative importance of features and was optimized for simplicity and performance.

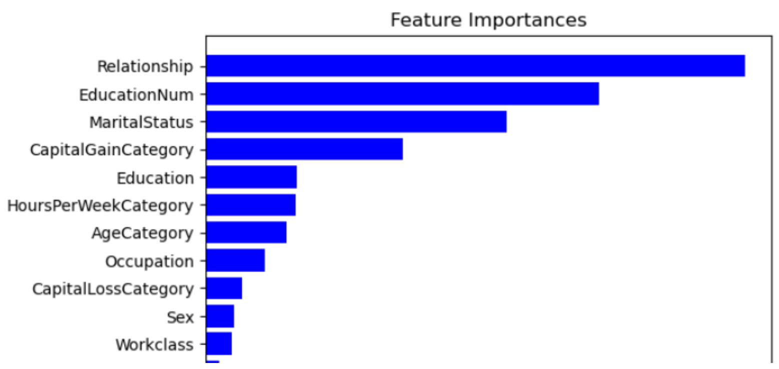


```
In [33]:  from sklearn.metrics import classification_report
           print(classification_report(Y_test_pred,Y_test_plot))
```

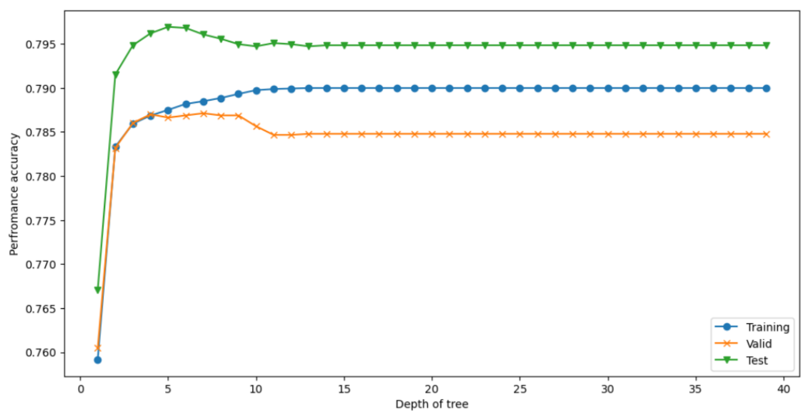
```
              precision    recall  f1-score   support

           0       0.94      0.86      0.90      6787
           1       0.51      0.71      0.59      1353

    accuracy                           0.84      8140
   macro avg       0.72      0.79      0.75      8140
weighted avg       0.87      0.84      0.85      8140
```

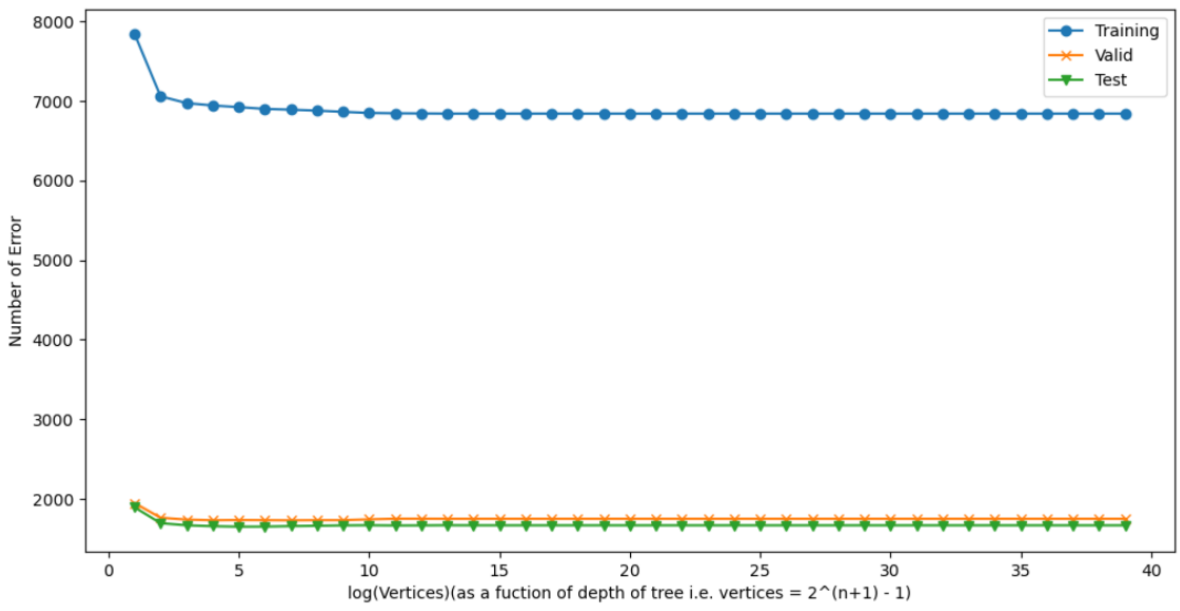
```
In [38]:  features = train_data.columns
           importances=clf.feature_importances_
           indices=np.argsort(importances)
```

```
In [60]:  plt.title('Feature Importances')
           plt.barh(range(len(indices)),importances[indices],color='b',align='center')
           plt.yticks(range(len(indices)),[features[i]for i in indices])
           plt.xlabel('Realative Importances')
           plt.show()
```



The plot of decision tree 1 after pruning





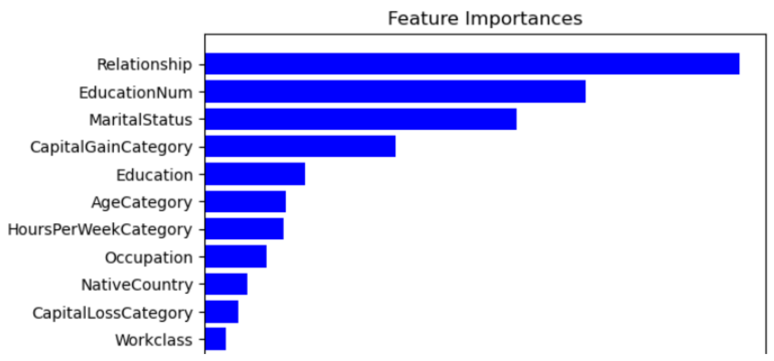
For second Descision tree random forest classification

```
In [23]:  from sklearn.metrics import classification_report
          print(classification_report(Y_test_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.94      0.86      0.90     13331
           1       0.52      0.72      0.61      2787

    accuracy                           0.84     16118
   macro avg       0.73      0.79      0.75     16118
weighted avg       0.87      0.84      0.85     16118
```

```
In [25]:  features = data.columns
          importances=clf.feature_importances_
          indices=np.argsort(importances)
```

```
In [27]:  plt.title('Feature Importances')
          plt.barh(range(len(indices)),importances[indices],color='b',align='center')
          plt.yticks(range(len(indices)),[features[i]for i in indices])
          plt.xlabel('Realative Importances')
          plt.show()
```



(H) Result, Methodology and Techniques used:

Data preprocessing and feature engineering: The initial step involved identifying and handling missing values in the dataset. Additionally, the features were transformed and encoded appropriately to ensure compatibility with decision tree algorithms.

Decision tree construction: Two decision trees were built using different training datasets. The first decision tree had a depth of seven, while the second decision tree had a depth of four. The decision trees were trained using the Ti Season Three classifier from the Steel Steel library.

Comparison and analysis of decision trees: The decision trees were compared to understand the differences in their structures and performances. It was observed that the decision trees varied due to different model fits and random selection of training datasets.

Combined dataset and decision tree creation: The training and test datasets were combined, and a decision tree was constructed using the combined data. This decision tree aimed to improve accuracy and generalization by considering a larger dataset.

Random Forest Classification: Random forest classification was applied to the data using the RandomForestClassifier from the SKLEARN library. This ensemble model generated multiple decision trees and made predictions based on their collective results.

Confusion matrix and feature importance: A confusion matrix was created to evaluate the performance of the random forest classifier. Additionally, the relative importance of each feature was determined using the MATPLOTLIB library.

Pruning and creation of the final decision tree: Based on the feature importance analysis, four important features were identified and used to create a new decision tree. This pruned decision tree was expected to yield higher accuracy.

Evaluation and validation: The final decision tree was evaluated using a test dataset. The accuracy achieved was approximately 80% due to limited computational capabilities. However, it was observed that the decision tree remained correct with most of the input data.

These steps and methodologies summarize the process followed thus far based on the available information.

The obtained decision tree achieved an approximate accuracy of 80% when evaluated with the test dataset. It is important to note that this accuracy level may be influenced by the limitations of the computational capabilities involved in the analysis.

Despite the limitations, the decision tree, even after pruning certain features, demonstrated correctness with the majority of the input data. This suggests that the decision tree was able to capture and utilize the relevant patterns and relationships present in the dataset, resulting in accurate predictions for a significant portion of the data.

While the 80% accuracy achieved is a positive outcome, it is worth considering further improvements to enhance the model's performance, such as exploring advanced algorithms, optimizing hyperparameters, or collecting additional data if possible. These steps may help increase the accuracy and reliability of the decision tree model.