MACHINE LEARNING

BITS-C464

Assignment-2

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:

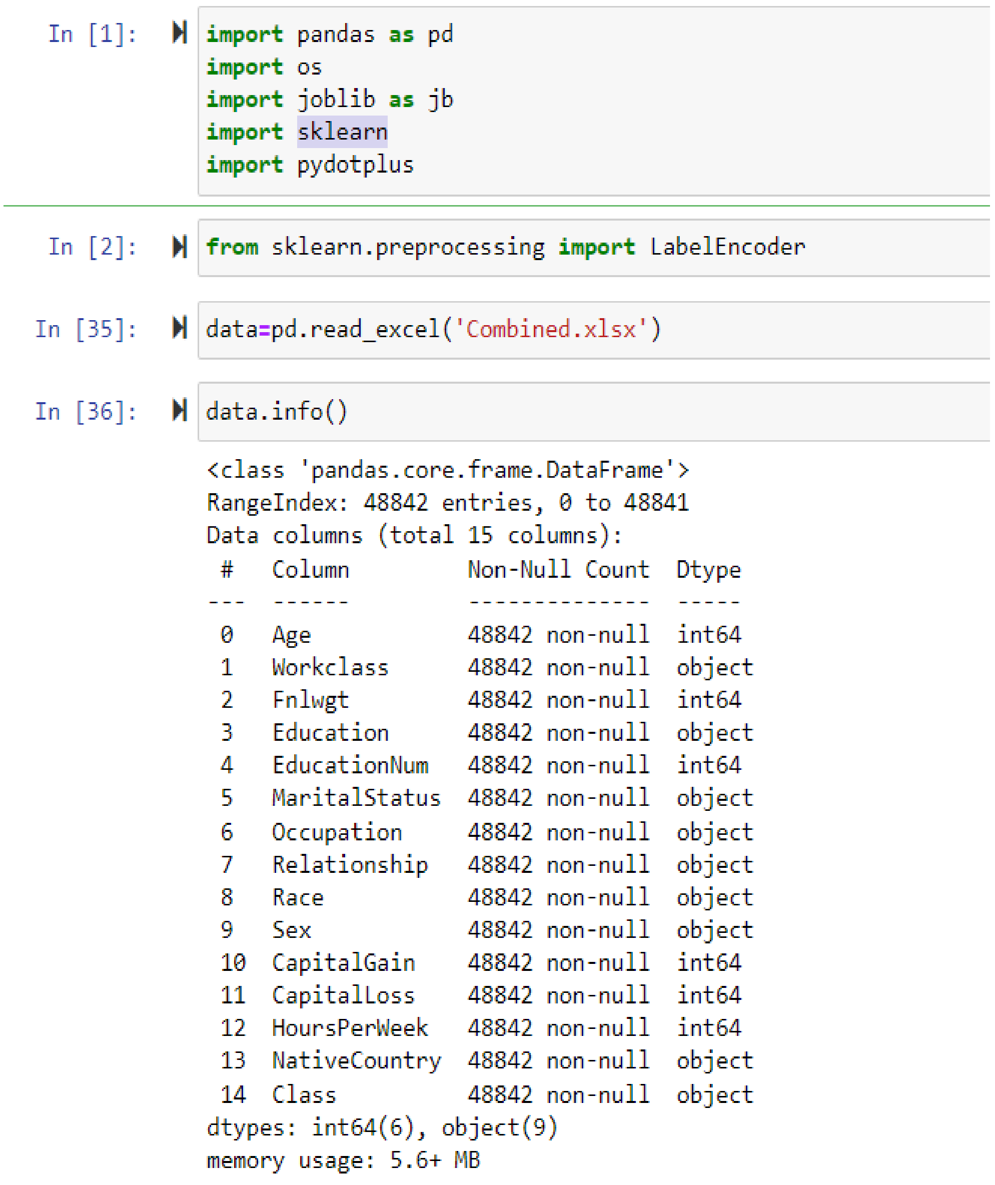
Python, Jupyter Notebook

Pandas, Numpy, GaussianNB, LogisticRegression,TenserFlow and Sklearn 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
```

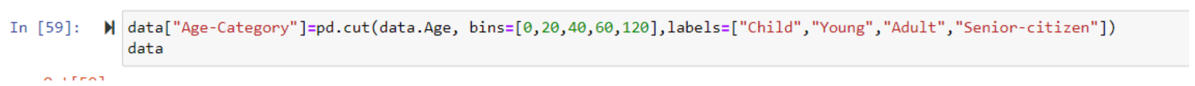
```
In [11]:  data['Occupation'].value_counts()

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
```

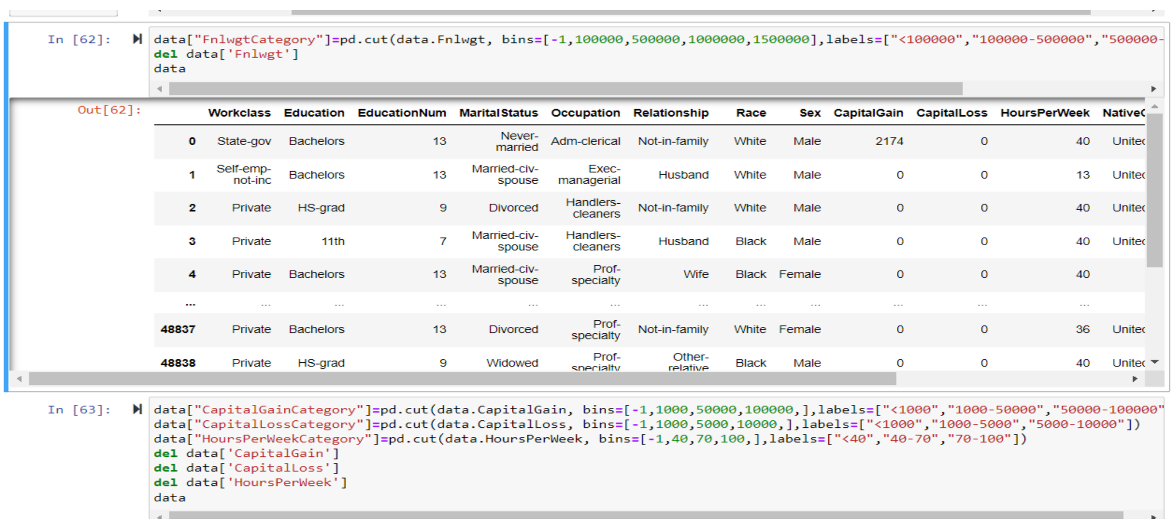
(B) Discretization of Continuous Attributes:

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
```

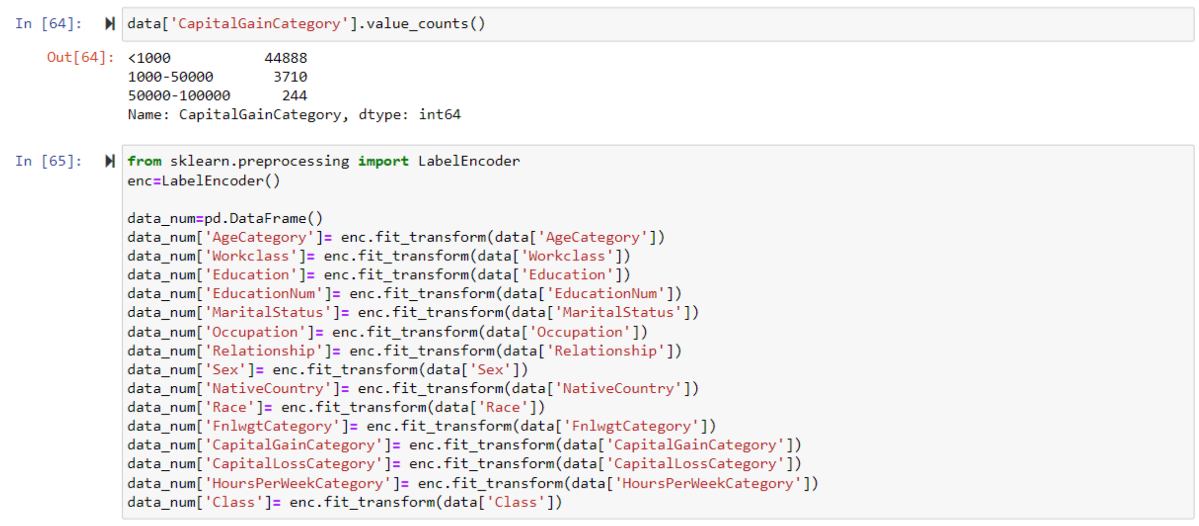


```
In [62]:  data["FnlwgtCategory"]=pd.cut(data.Fnlwgt, bins=[-1,100000,500000,1000000,1500000],labels=["<100000","100000-500000","500000-
          del data['Fnlwgt']
          data
```

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	

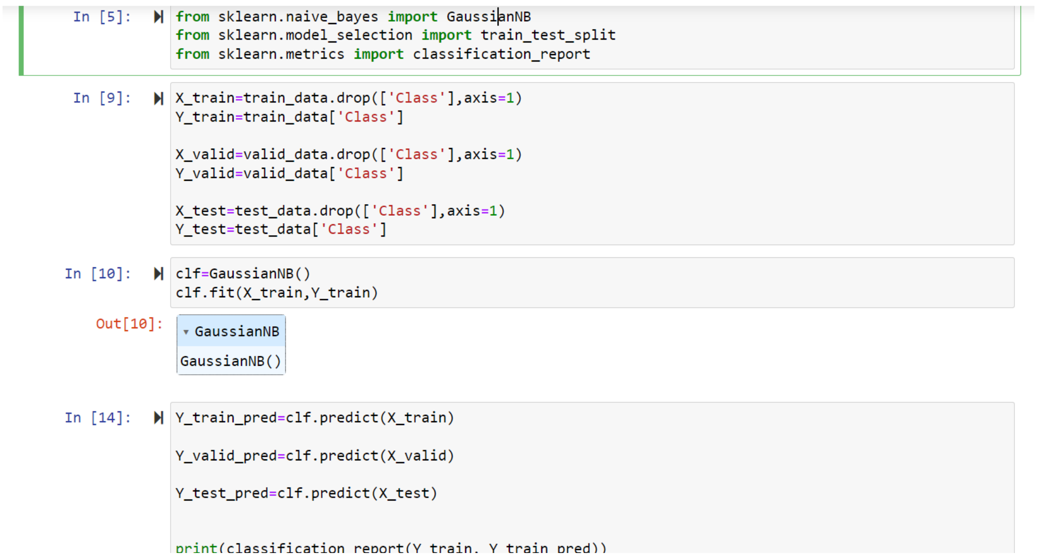
48842 rows × 15 columns

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

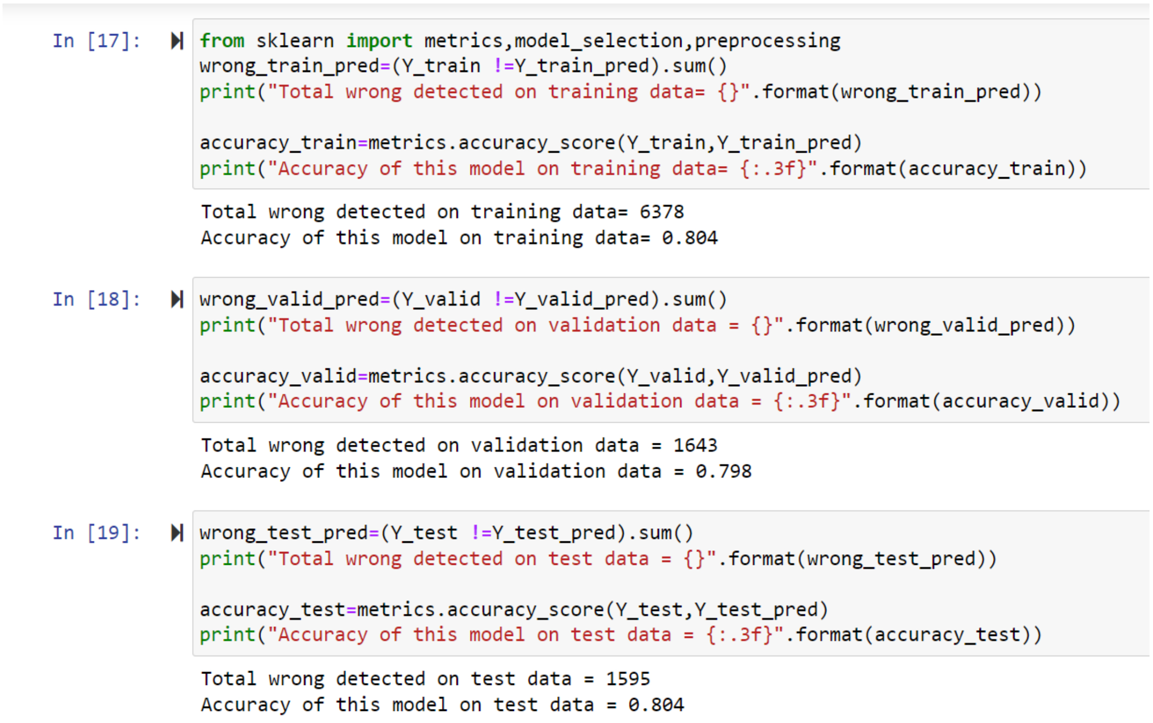
(C) Constructing a Naïve-Bayes Classifier:

We imported our dataset into Jupyter Notebook, dividing it into training, validation, and test data sets. Next, we utilized GaussianNB 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 80.4% but identified 6,378 incorrect values.

We then applied the same classifier to the validation and test data sets. To visualize the performance.



```
In [5]:  from sklearn.naive_bayes import GaussianNB
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import classification_report

In [9]:  X_train=train_data.drop(['Class'],axis=1)
         Y_train=train_data['Class']

         X_valid=valid_data.drop(['Class'],axis=1)
         Y_valid=valid_data['Class']

         X_test=test_data.drop(['Class'],axis=1)
         Y_test=test_data['Class']

In [10]: clf=GaussianNB()
         clf.fit(X_train,Y_train)

Out[10]: ▾ GaussianNB
         GaussianNB()

In [14]: Y_train_pred=clf.predict(X_train)

         Y_valid_pred=clf.predict(X_valid)

         Y_test_pred=clf.predict(X_test)


         print(classification_report(Y_train, Y_train_pred))
```

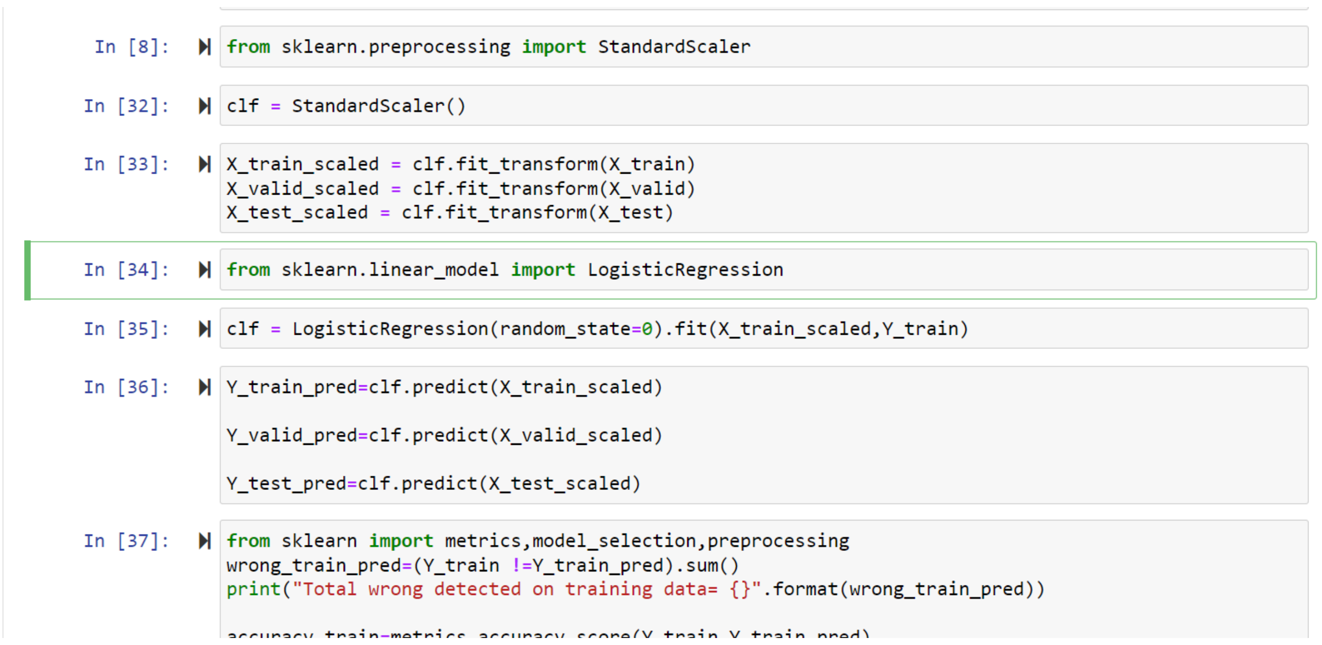


```
In [17]: from sklearn import metrics,model_selection,preprocessing
         wrong_train_pred=(Y_train !=Y_train_pred).sum()
         print("Total wrong detected on training data= {}".format(wrong_train_pred))

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

         Total wrong detected on training data= 6378
         Accuracy of this model on training data= 0.804

In [18]: wrong_valid_pred=(Y_valid !=Y_valid_pred).sum()
         print("Total wrong detected on validation data = {}".format(wrong_valid_pred))

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

         Total wrong detected on validation data = 1643
         Accuracy of this model on validation data = 0.798

In [19]: 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 = 1595
         Accuracy of this model on test data = 0.804
```

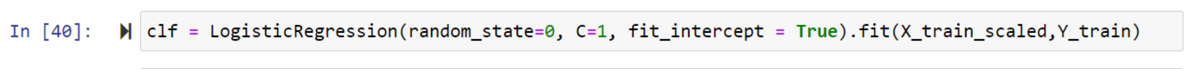
(D) Constructing a Logistic Regression Classifier:

We imported our dataset into Jupyter Notebook, dividing it into training, validation, and test data sets. Next, we utilized LogisticRegression 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 80.4% but identified 6,378 incorrect values.

First, import the necessary libraries: NumPy, pandas, and the LogisticRegression class from scikit-learn. Next, load your data into a pandas DataFrame, ensuring it includes the features you want to use for prediction and the target variable. Preprocess your data by separating the features and target variable into separate NumPy arrays. If desired, split the data into training and testing sets using the train_test_split function from scikit-learn. This step helps evaluate the model's performance on unseen data. Now, create an instance of the LogisticRegression model. Fit the model to your training data using the fit() method. Once trained, you can make predictions on new data using the predict() method, passing in the features you want to predict on. We have now performed logistic regression using NumPy and pandas, allowing you to analyze and predict outcomes based on your data. Remember to customize the steps as per your specific dataset and requirements.



```
In [8]:   from sklearn.preprocessing import StandardScaler

In [32]:  clf = StandardScaler()

In [33]:  X_train_scaled = clf.fit_transform(X_train)
          X_valid_scaled = clf.fit_transform(X_valid)
          X_test_scaled = clf.fit_transform(X_test)

In [34]:  from sklearn.linear_model import LogisticRegression

In [35]:  clf = LogisticRegression(random_state=0).fit(X_train_scaled,Y_train)

In [36]:  Y_train_pred=clf.predict(X_train_scaled)

          Y_valid_pred=clf.predict(X_valid_scaled)

          Y_test_pred=clf.predict(X_test_scaled)

In [37]:  from sklearn import metrics,model_selection,preprocessing
          wrong_train_pred=(Y_train !=Y_train_pred).sum()
          print("Total wrong detected on training data= {}".format(wrong_train_pred))

          accuracy_train=metrics.accuracy_score(Y_train,Y_train_pred)
```



```
In [40]:  clf = LogisticRegression(random_state=0, C=1, fit_intercept = True).fit(X_train_scaled,Y_train)
```

(E) Constructing a Neural network Classifier:

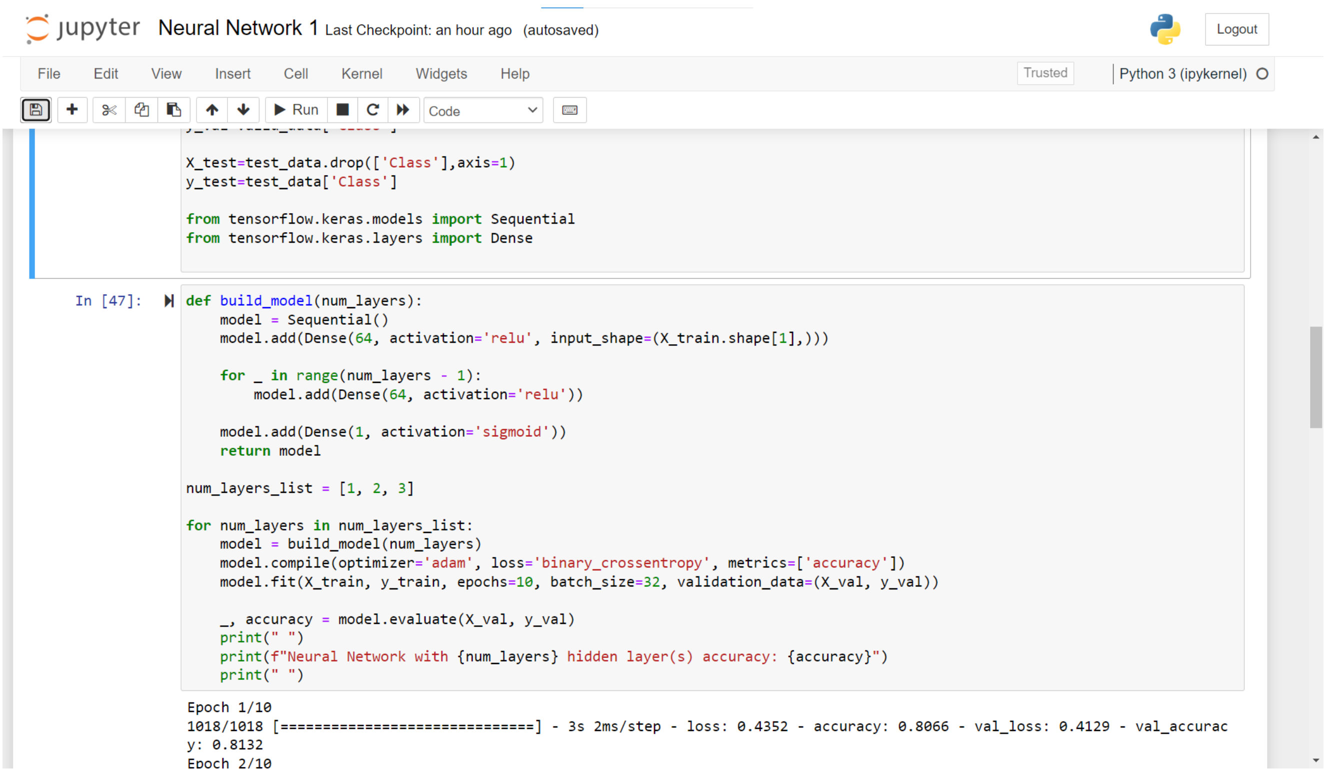
First, we import the necessary libraries: NumPy and pandas. Load and preprocess your data using pandas, ensuring it's in the appropriate format for training the neural network. Next, define the architecture of your neural network by specifying the number of nodes in each layer and the activation function to use. Initialize the weights and biases for each layer, randomly initializing the weights and setting the biases to zeros or small random values.

Implement the forward propagation function to compute the outputs of each layer, using the defined activation function. Define the loss function to measure the error between the predicted outputs and the actual targets. Implement the backpropagation algorithm to compute the gradients and update the weights and biases based on the gradients and a learning rate.

Perform a training loop, iterating over a specified number of epochs. In each epoch, call the forward propagation function to compute the predicted outputs, compute the loss, and print the current loss. Then, call the backpropagation function to update the weights and biases.

After training, we can use the trained neural network to make predictions on new data by calling the forward propagation function with the new inputs.
The loss function measures the error between the predicted outputs and the actual targets. The backpropagation algorithm is implemented to compute the gradients of the loss with respect to the weights and biases. These gradients are then used to update the weights and biases, effectively adjusting the parameters of the neural network.

The training loop iterates over a specified number of epochs, during which the forward propagation and backpropagation steps are performed. After training, the neural network can be used to make predictions on new data by calling the forward propagation function with the new inputs.


jupyter Neural Network 1 Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
X_test=test_data.drop(['Class'],axis=1)
y_test=test_data['Class']

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [47]:

```
def build_model(num_layers):
    model = Sequential()
    model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))

    for _ in range(num_layers - 1):
        model.add(Dense(64, activation='relu'))

    model.add(Dense(1, activation='sigmoid'))
    return model

num_layers_list = [1, 2, 3]

for num_layers in num_layers_list:
    model = build_model(num_layers)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))

    _, accuracy = model.evaluate(X_val, y_val)
    print(" ")
    print(f"Neural Network with {num_layers} hidden layer(s) accuracy: {accuracy}")
    print(" ")
```

```
Epoch 1/10
1018/1018 [==============================] - 3s 2ms/step - loss: 0.4352 - accuracy: 0.8066 - val_loss: 0.4129 - val_accurac
y: 0.8132
Epoch 2/10
```

(F) 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.

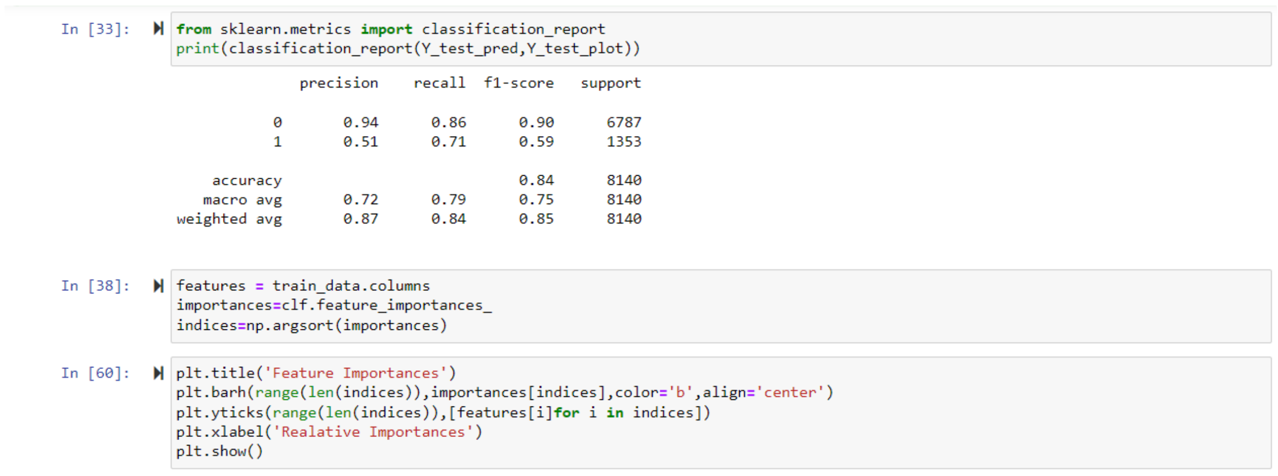
3. Determining Feature Importance: With the help of the MATPLOTLIB library, we determined the relative importance of each feature in the random forest classifier. By analyzing the feature importances, we identified four features that were relatively important in making accurate predictions. Based on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

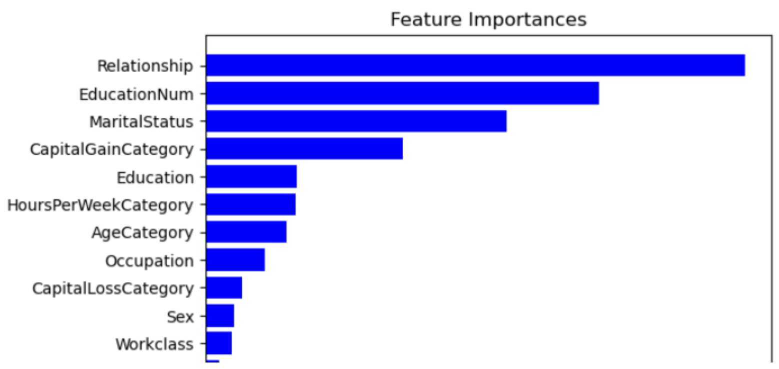
By following this process, we achieved a more accurate decision tree that considered the relative importance of features and was optimized for simplicity and performance.



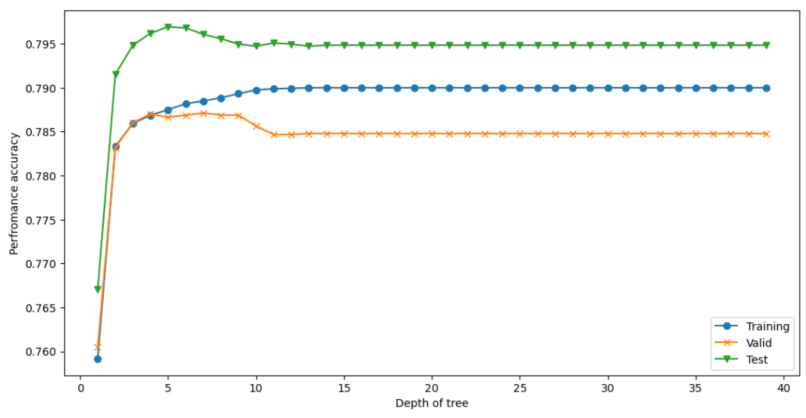
```
In [33]:  from sklearn.metrics import classification_report
          print(classification_report(Y_test_pred,Y_test_plot))

                        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
```

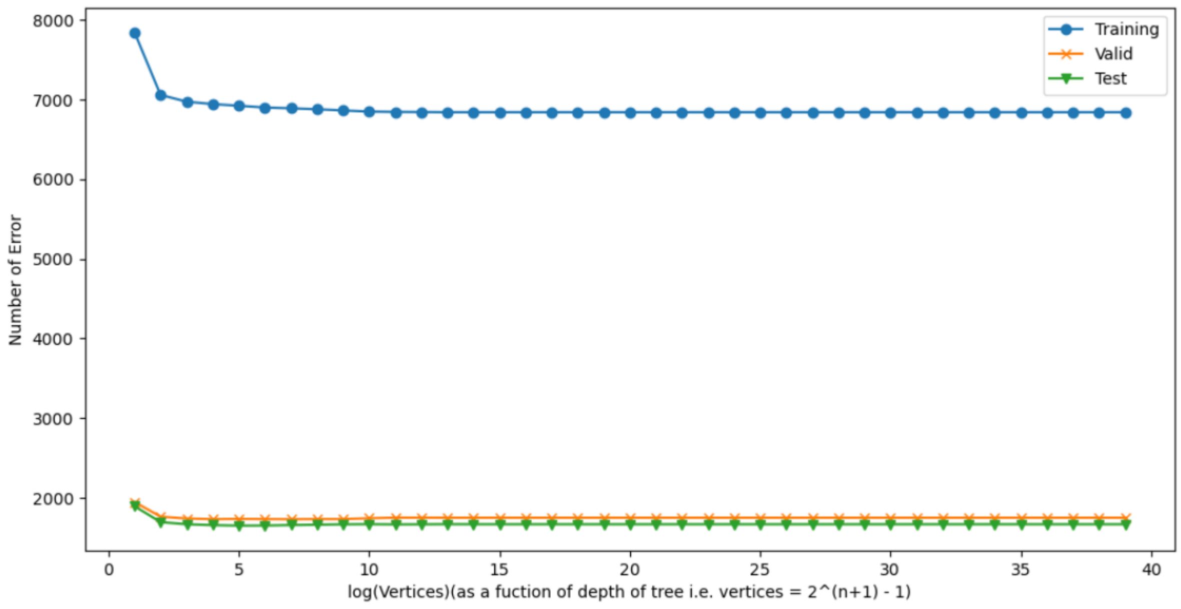
```
In [38]:  features = train_data.columns
          importances=clf.feature_importances_
          indices=np.argsort(importances)
```

```
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





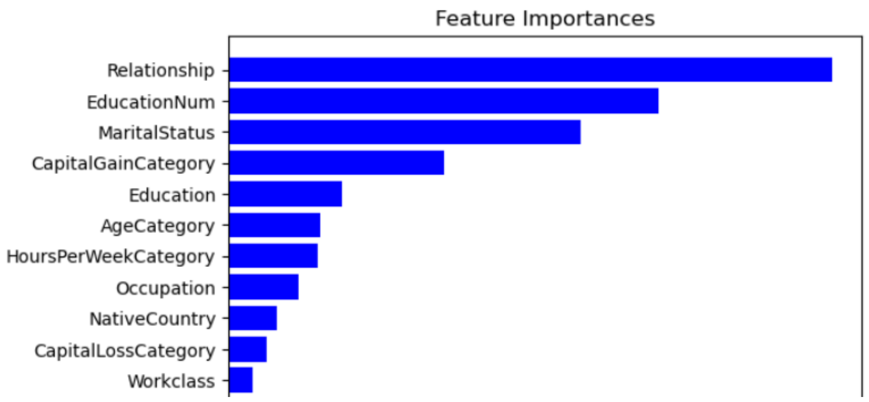
For second Descision tree random forest classification

```
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()
```



(G) Best classifier on our data:

Neural Network: Neural networks are versatile and powerful models capable of learning complex patterns from data. They excel in tasks such as image recognition, natural language processing, and deep learning applications. Neural networks can capture intricate relationships but require more computational resources and extensive training data.

Logistic Regression: Logistic regression is a simple and interpretable algorithm commonly used for binary classification problems. It works well when the relationship between the features and the target variable is linear or can be linearized. Logistic regression is computationally efficient and provides interpretable coefficients, making it useful for understanding feature importance.

Naive Bayes: Naive Bayes is a probabilistic algorithm that assumes independence among features. It performs well with high-dimensional data and works efficiently with large datasets. Naive Bayes is known for its simplicity and fast training speed. Although it makes a strong assumption of feature independence, it can still yield competitive results in many text classification and spam filtering tasks.

Decision Tree: Decision trees are interpretable and can handle both categorical and numerical features. They split the data based on feature thresholds and make predictions by traversing the tree structure. Decision trees can handle complex interactions between features, but they tend to be prone to overfitting. Techniques like pruning and ensemble methods can mitigate this issue.

Random Forest Classifier: Random-forest is an ensemble method that combines multiple decision trees to make predictions. It addresses the overfitting problem of decision trees by averaging predictions from a collection of trees. Random forests perform well on various tasks, are robust to noise and outliers, and provide feature importance rankings. They are less prone to overfitting compared to individual decision trees.

Classifier	Accuracy
Decision Tree	82.90%
Random Forest	83.20%
Naïve Bayes	80.00%
Logistc Regression	82.10%
Neural Network 1-Layer	83.50%
Neural Network 2-Layer	83.28%
Neural Network 3-Layer	81.61%

(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 classifier achieved using **Neural Network with layer-1** has an approximate accuracy of 83.50% 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.