In [9]:
```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

import pickle
import json

from imblearn.over_sampling import SMOTE

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV, Randomized
from sklearn.metrics import accuracy_score, classification_report, confusion_m
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.cluster import KMeans

from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

import warnings
warnings.filterwarnings("ignore")
```

In [10]:
```
model_details = []
testing_accuracy_list = []
training_accuracy_list = []
```

In [11]:
```
# Function Definition

def outlier_imputation(df):
    q1 = df.quantile(0.25)
    q3 = df.quantile(0.75)

    iqr = q3-q1
    uppertail = q3 + 1.5*iqr
    lowertail = q1 - 1.5*iqr

    median = df.loc[(df <= uppertail) & (df >= lowertail)].median()
    df.loc[(df < lowertail) | (df > uppertail)] = median

    return sns.boxplot(y=df)

def normalization(x_df):
    normal_scaler = MinMaxScaler()
    array = normal_scaler.fit_transform(x_df)

    x_normal_df = pd.DataFrame(array,columns=x_df.columns)
    return x_normal_df

def log_reg_model_training(x_train,y_train):
    log_reg = LogisticRegression()
    log_reg.fit(x_train,y_train)
    model_details.append(model_name)
    return log_reg

def knn_model_building(x_train,y_train):
    knn_clf_model = KNeighborsClassifier()
    knn_clf_model.fit(x_train,y_train)
    model_details.append(model_name)
    return knn_clf_model

def adaboost_clf_training(x_train,y_train):
    adaboost_clf = AdaBoostClassifier(estimator=LogisticRegression(),random_st
    adaboost_clf.fit(x_train,y_train)
    model_details.append(model_name)
    return adaboost_clf

def dt_clf_training(x_train,y_train):
    dt_clf = DecisionTreeClassifier(random_state=18)
    dt_clf.fit(x_train,y_train)
    model_details.append(model_name)
    return dt_clf

def model_evaluation_testing(model,x_test,y_test):
    y_pred_test = model.predict(x_test)
    print("*"*50,"Testing Data","*"*50)

    cnf_matrix = confusion_matrix(y_test,y_pred_test)
    print("The confusion Matrix is :\n",cnf_matrix)
    cmd = ConfusionMatrixDisplay(cnf_matrix)
    cmd.plot()
    plt.show()
    print("="*50)
```

```
    accuracy = accuracy_score(y_test,y_pred_test)
    print("The Accuracy is : ",accuracy)
    print("="*50)

    clf_report = classification_report(y_test,y_pred_test)
    print("The Classification report is :\n",clf_report)

    testing_accuracy_list.append(accuracy)


def model_evaluation_training(model,x_train,y_train):
    y_pred_train = model.predict(x_train)
    print("*"*50,"Training Data","*"*50)

    cnf_matrix = confusion_matrix(y_train,y_pred_train)
    print("The confusion matrix is :\n",cnf_matrix)
    cmd = ConfusionMatrixDisplay(cnf_matrix)
    cmd.plot()
    plt.show()
    print("="*50)

    accuracy = accuracy_score(y_train,y_pred_train)
    print("The accuracy is : ",accuracy)
    print("="*50)

    clf_report = classification_report(y_train,y_pred_train)
    print("The classification Report is :\n",clf_report)

    training_accuracy_list.append(accuracy)

def get_auc_roc_curve(model,x_train,y_train):
    y_pred_prob = model.predict_proba(x_train)
    pred_prob = y_pred_prob[:,1]

    fpr, tpr, thresh = roc_curve(y_train,pred_prob)

    plt.plot(fpr,tpr)
    plt.xlabel("False Positive Rate (FPR)")
    plt.ylabel("True Positive Rate (TPR)")
    plt.title("Receiver Operating Characteristic Curve")
    plt.show()

    auc_value =auc(fpr,tpr)
    print("The AUC is : ",auc_value)

def dt_gscv_best_estimator(x_tarin,y_train):
    dt_clf = DecisionTreeClassifier()
    par_grid = {"criterion":['gini', 'entropy'],
                "max_depth": np.arange(3,8),
                "min_samples_split": np.arange(2,20),
                "min_samples_leaf": np.arange(2,15)}

    gscv_dt_clf = GridSearchCV(dt_clf,par_grid,cv=5,n_jobs=-1)
    gscv_dt_clf.fit(x_tarin,y_train)

    model_details.append("Hyperparameter Tunned GSCV model")
#     best_parameters.append(gscv_adb_clf.best_params_)
```

```
    return gscv_dt_clf.best_estimator_

def dt_rscv_best_estimator(x_train,y_train):
    dt_clf = DecisionTreeClassifier()
    par_grid = {"criterion":['gini', 'entropy'],
                "max_depth": np.arange(3,8),
                "min_samples_split": np.arange(2,20),
                "min_samples_leaf": np.arange(2,15)}

    rscv_dt_clf = RandomizedSearchCV(dt_clf,par_grid,cv=5,n_jobs=-1,random_sta
    rscv_dt_clf.fit(x_train,y_train)

    model_details.append("Hyperparameter Tunned RSCV model")
#     best_parameters.append(rscv_adb_clf.best_params_)

    return rscv_dt_clf.best_estimator_

def rf_gscv_best_estimator(x_tarin,y_train):
    dt_clf = RandomForestClassifier()
    par_grid = {"criterion":['gini', 'entropy'],
                "max_depth": np.arange(3,8),
                "min_samples_split": np.arange(2,20),
                "min_samples_leaf": np.arange(2,15),
              "max_features":['sqrt','log2']}

    gscv_dt_clf = GridSearchCV(dt_clf,par_grid,cv=5,n_jobs=-1)
    gscv_dt_clf.fit(x_tarin,y_train)

    model_details.append("Hyperparameter Tunned GSCV model")
#     best_parameters.append(gscv_adb_clf.best_params_)

    return gscv_dt_clf.best_estimator_

def rf_rscv_best_estimator(x_train,y_train):
    dt_clf = RandomForestClassifier()
    par_grid = {"criterion":['gini', 'entropy'],
                "max_depth": np.arange(3,8),
                "min_samples_split": np.arange(2,20),
                "min_samples_leaf": np.arange(2,15),
              "max_features":['sqrt','log2']}

    rscv_dt_clf = RandomizedSearchCV(dt_clf,par_grid,cv=5,n_jobs=-1,random_sta
    rscv_dt_clf.fit(x_train,y_train)

    model_details.append("Hyperparameter Tunned RSCV model")
#     best_parameters.append(rscv_adb_clf.best_params_)

    return rscv_dt_clf.best_estimator_

def adb_gscv_best_estimator(x_tarin,y_train):
    adb_clf = AdaBoostClassifier()
    par_grid = {"n_estimators":np.arange(50,60),
              "learning_rate":np.arange(0,1,0.001)}

    gscv_adb_clf = GridSearchCV(adb_clf,par_grid,cv=5,n_jobs=-1)
    gscv_adb_clf.fit(x_tarin,y_train)
```

```
    model_details.append("Hyperparameter Tunned GSCV model")
#     best_parameters.append(gscv_adb_clf.best_params_)

    return gscv_adb_clf.best_estimator_

def adb_rscv_best_estimator(x_train,y_train):
    adb_clf = AdaBoostClassifier()
    par_grid = {"n_estimators":np.arange(50,100),
                "learning_rate":np.arange(0,1,0.001)}

    rscv_adb_clf = RandomizedSearchCV(adb_clf,par_grid,cv=5,n_jobs=-1,random_s
    rscv_adb_clf.fit(x_train,y_train)

    model_details.append("Hyperparameter Tunned RSCV model")
#     best_parameters.append(rscv_adb_clf.best_params_)

    return rscv_adb_clf.best_estimator_

def knn_gscv_best_estimator(x_tarin,y_train):
    knn_clf = KNeighborsClassifier()
    par_grid = {"n_neighbors":np.arange(3,30),
                "p":[1,2]}

    gscv_knn_clf = GridSearchCV(knn_clf,par_grid,cv=5)
    gscv_knn_clf.fit(x_tarin,y_train)

    model_details.append("Hyperparameter Tunned GSCV model")
#     best_parameters.append(gscv_knn_clf.best_params_)

    return gscv_knn_clf.best_estimator_

def knn_rscv_best_estimator(x_train,y_train):
    knn_clf = KNeighborsClassifier()
    par_grid = {"n_neighbors":np.arange(3,30),
                "p":[1,2]}

    rscv_knn_clf = RandomizedSearchCV(knn_clf,par_grid,cv=5,random_state=42)
    rscv_knn_clf.fit(x_train,y_train)

    model_details.append("Hyperparameter Tunned RSCV model")
#     best_parameters.append(rscv_knn_clf.best_params_)

    return rscv_knn_clf.best_estimator_
```

Problem Statement

```
"Predicting the Approval of Credit Card for customers using Classification
Model:

Using a dataset of Credit Card of customers from csv file the goal of this
project is to build a Classification Model
```

```
by using supervised machine learning.that can accurately predict the aprroval
or rejection of credit card or rnage of loan allotted based on various
factors such as Age,Gender,Income,Education,Marital Status,Number of
Children,Home Ownership,Credit Score,Loan amount,EMI,Inhand
Sallary,Eligibility,Credit card limit,Range.
The model will be evaluated based on its ability to predict eligibility on a
held-out test dataset,
using metrics such as cnf_matrix and clf_report"
```

Data Gathering

In [12]:
```
df1 = pd.read_csv("Credit Score Classification Dataset - Credit Score Classifi
df1
```

Out[12]:

	Age	Gender	Income	Education	Marital Status	Number of Children	Home Ownership	Credit Score	Loan amount	I
0	25	Female	50000	Bachelor's Degree	Single	0	Rented	High	6000000.0	16666
1	30	Male	100000	Master's Degree	Married	2	Owned	High	10000000.0	33333
2	35	Female	75000	Doctorate	Married	1	Owned	High	6000000.0	25000
3	40	Male	125000	High School Diploma	Single	0	Owned	High	7500000.0	41666
4	45	Female	100000	Bachelor's Degree	Married	3	Owned	High	4000000.0	33333
...	...	...	...	...	...	...	...	...	...	
159	29	Female	27500	High School Diploma	Single	0	Rented	Low	NaN	N
160	34	Male	47500	Associate's Degree	Single	0	Rented	Average	3990000.0	15833
161	39	Female	62500	Bachelor's Degree	Married	2	Owned	High	4000000.0	20833
162	44	Male	87500	Master's Degree	Single	0	Owned	High	3850000.0	29166
163	49	Female	77500	Doctorate	Married	1	Owned	High	1860000.0	25833

164 rows × 14 columns

In [13]:
```
df1["Range"].value_counts()
```

Out[13]:
```
1.5 - 3     98
0           26
3 - 4.5     24
1 - 1.5     16
Name: Range, dtype: int64
```

In [14]:
```
df = df1.drop("Range",axis=1)
df
```

Out[14]:

	Age	Gender	Income	Education	Marital Status	Number of Children	Home Ownership	Credit Score	Loan amount	
0	25	Female	50000	Bachelor's Degree	Single	0	Rented	High	6000000.0	16666
1	30	Male	100000	Master's Degree	Married	2	Owned	High	10000000.0	33333
2	35	Female	75000	Doctorate	Married	1	Owned	High	6000000.0	25000
3	40	Male	125000	High School Diploma	Single	0	Owned	High	7500000.0	41666
4	45	Female	100000	Bachelor's Degree	Married	3	Owned	High	4000000.0	33333
...	...	...	...	...	...	...	...	...	...	
159	29	Female	27500	High School Diploma	Single	0	Rented	Low	NaN	N
160	34	Male	47500	Associate's Degree	Single	0	Rented	Average	3990000.0	15833
161	39	Female	62500	Bachelor's Degree	Married	2	Owned	High	4000000.0	20833
162	44	Male	87500	Master's Degree	Single	0	Owned	High	3850000.0	29166
163	49	Female	77500	Doctorate	Married	1	Owned	High	1860000.0	25833

164 rows × 13 columns

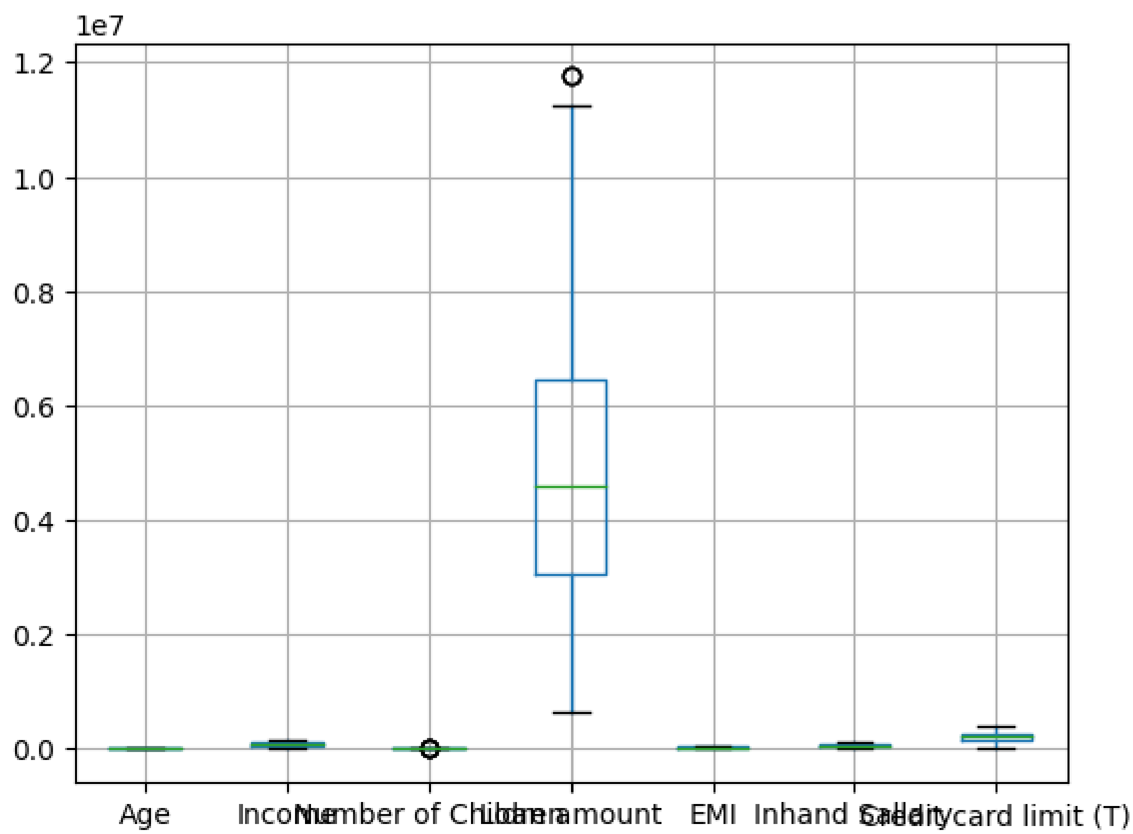
In [15]:
```
# DataFrame for scaling

df22 = df[['Age','Income','Number of Children','Loan amount','EMI','Inhand Sal
```

Exploratory Data Analysis

In [16]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 164 entries, 0 to 163
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Age                    164 non-null    int64  
 1   Gender                 164 non-null    object 
 2   Income                 164 non-null    int64  
 3   Education              164 non-null    object 
 4   Marital Status         151 non-null    object 
 5   Number of Children     164 non-null    int64  
 6   Home Ownership         154 non-null    object 
 7   Credit Score           164 non-null    object 
 8   Loan amount            159 non-null    float64
 9   EMI                    159 non-null    float64
 10  Inhand Sallary         164 non-null    int64  
 11  Eligibility            164 non-null    object 
 12  Credit card limit (T)  164 non-null    int64  
dtypes: float64(2), int64(5), object(6)
memory usage: 16.8+ KB
None
```

In [17]:
```
df.boxplot()
plt.show()
```



Feature Engineering

In [18]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 164 entries, 0 to 163
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Age                    164 non-null    int64  
 1   Gender                 164 non-null    object 
 2   Income                 164 non-null    int64  
 3   Education              164 non-null    object 
 4   Marital Status         151 non-null    object 
 5   Number of Children     164 non-null    int64  
 6   Home Ownership         154 non-null    object 
 7   Credit Score           164 non-null    object 
 8   Loan amount            159 non-null    float64
 9   EMI                    159 non-null    float64
 10  Inhand Sallary         164 non-null    int64  
 11  Eligibility            164 non-null    object 
 12  Credit card limit (T)  164 non-null    int64  
dtypes: float64(2), int64(5), object(6)
memory usage: 16.8+ KB
```

In [19]:
```
df["Gender"].value_counts()
```

Out[19]:
```
Female    86
Male      78
Name: Gender, dtype: int64
```

In [20]:
```
df["Gender"].replace({"Male":0,"Female":1},inplace=True)
Gender_values = {"Male":0,"Female":1}
```

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 164 entries, 0 to 163
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Age                    164 non-null    int64  
 1   Gender                 164 non-null    int64  
 2   Income                 164 non-null    int64  
 3   Education              164 non-null    object 
 4   Marital Status         151 non-null    object 
 5   Number of Children     164 non-null    int64  
 6   Home Ownership         154 non-null    object 
 7   Credit Score           164 non-null    object 
 8   Loan amount            159 non-null    float64
 9   EMI                    159 non-null    float64
 10  Inhand Sallary         164 non-null    int64  
 11  Eligibility            164 non-null    object 
 12  Credit card limit (T)  164 non-null    int64  
dtypes: float64(2), int64(6), object(5)
memory usage: 16.8+ KB
```

In [22]:
```
df["Education"].value_counts().to_dict()
```

Out[22]:
```
{"Bachelor's Degree": 42,
 "Master's Degree": 36,
 'Doctorate': 31,
 'High School Diploma': 30,
 "Associate's Degree": 25}
```

In [23]:
```
df["Education"].replace({"Bachelor's Degree": 2,"Master's Degree": 3,'High Scho
                              'Doctorate': 4,"Associate's Degree": 1},inplace=T

Education_values = {"Bachelor's Degree": 2,"Master's Degree": 3,'High School Di
                              'Doctorate': 4,"Associate's Degree": 1}
```

In [24]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 164 entries, 0 to 163
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Age                    164 non-null    int64  
 1   Gender                 164 non-null    int64  
 2   Income                 164 non-null    int64  
 3   Education              164 non-null    int64  
 4   Marital Status         151 non-null    object 
 5   Number of Children     164 non-null    int64  
 6   Home Ownership         154 non-null    object 
 7   Credit Score           164 non-null    object 
 8   Loan amount            159 non-null    float64
 9   EMI                    159 non-null    float64
 10  Inhand Sallary         164 non-null    int64  
 11  Eligibility            164 non-null    object 
 12  Credit card limit (T)  164 non-null    int64  
dtypes: float64(2), int64(7), object(4)
memory usage: 16.8+ KB
```

In [25]:
```
df["Marital Status"].isna().sum()
```

Out[25]:
```
13
```

In [26]:
```
df["Marital Status"].value_counts()
```

Out[26]:
```
Married    79
Single     72
Name: Marital Status, dtype: int64
```

In [27]:
```
df["Marital Status"].fillna(df["Marital Status"].mode()[0],inplace=True)
```

In [28]:
```
df["Marital Status"].isna().sum()
```

Out[28]:
```
0
```

In [29]:
```
df["Marital Status"].value_counts()
```

Out[29]:
```
Married    92
Single     72
Name: Marital Status, dtype: int64
```

In [30]:
```
df["Marital Status"].replace({"Married":1,"Single":0},inplace=True)
Marital_Status_values = {"Married":1,"Single":0}
```

In [31]:
```
df["Home Ownership"].isna().sum()
```

Out[31]:
```
10
```

In [32]:
```
df["Home Ownership"].value_counts()
```

Out[32]:
```
Owned     105
Rented     49
Name: Home Ownership, dtype: int64
```

In [33]:
```
df["Home Ownership"].fillna(df["Home Ownership"].mode()[0],inplace=True)
```

In [34]:
```
df["Home Ownership"].value_counts()
```

Out[34]:
```
Owned     115
Rented     49
Name: Home Ownership, dtype: int64
```

In [35]:
```
df["Home Ownership"].replace({"Owned":1,"Rented":0},inplace=True)
Home_Owneship_values = {"Owned":1,"Rented":0}
```

In [36]:
```
df["Credit Score"].value_counts()
```

Out[36]:
```
High       113
Average     36
Low         15
Name: Credit Score, dtype: int64
```

In [37]:
```
df["Credit Score"].replace({'High': 2, 'Average': 1, 'Low': 0},inplace=True)
Credit_Score_values = {'High': 2, 'Average': 1, 'Low': 0}
```

In [38]:
```
df["Loan amount"].isna().sum()
```

Out[38]:
```
5
```

In [39]:
```
df["Loan amount"].fillna(df["Loan amount"].mean(),inplace=True)
```

In [40]:
```
df["Loan amount"].isna().sum()
```

Out[40]:
```
0
```

In [41]:
```
df["EMI"].isna().sum()
```
Out[41]:
```
5
```

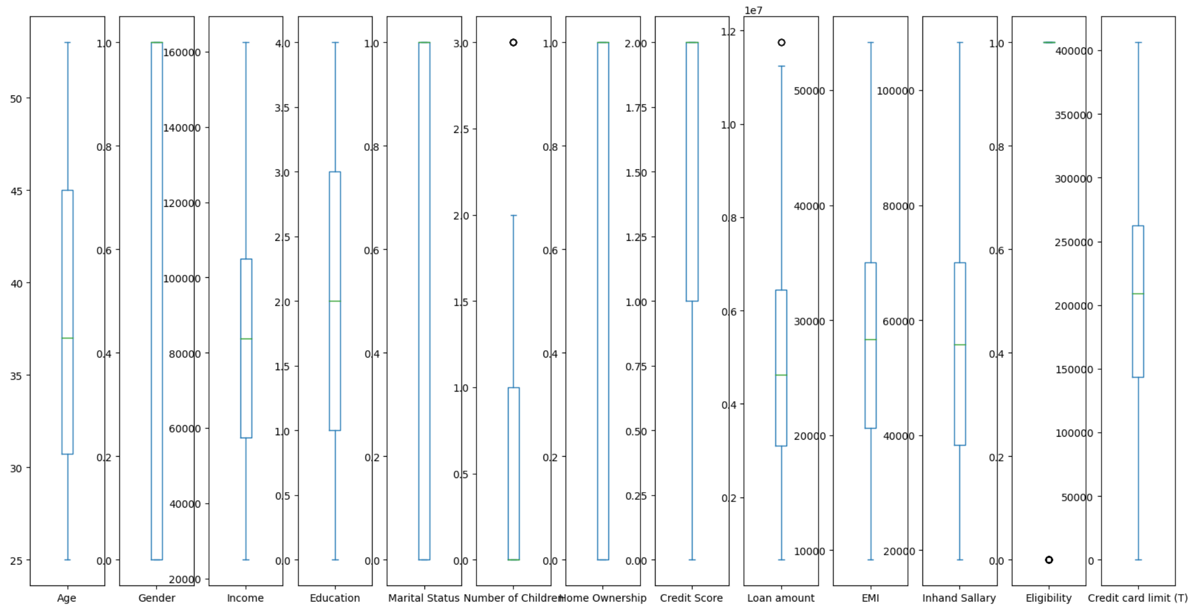
In [42]:
```
df["EMI"].fillna(df["EMI"].mean(),inplace=True)
```

In [43]:
```
df["EMI"].isna().sum()
```
Out[43]:
```
0
```

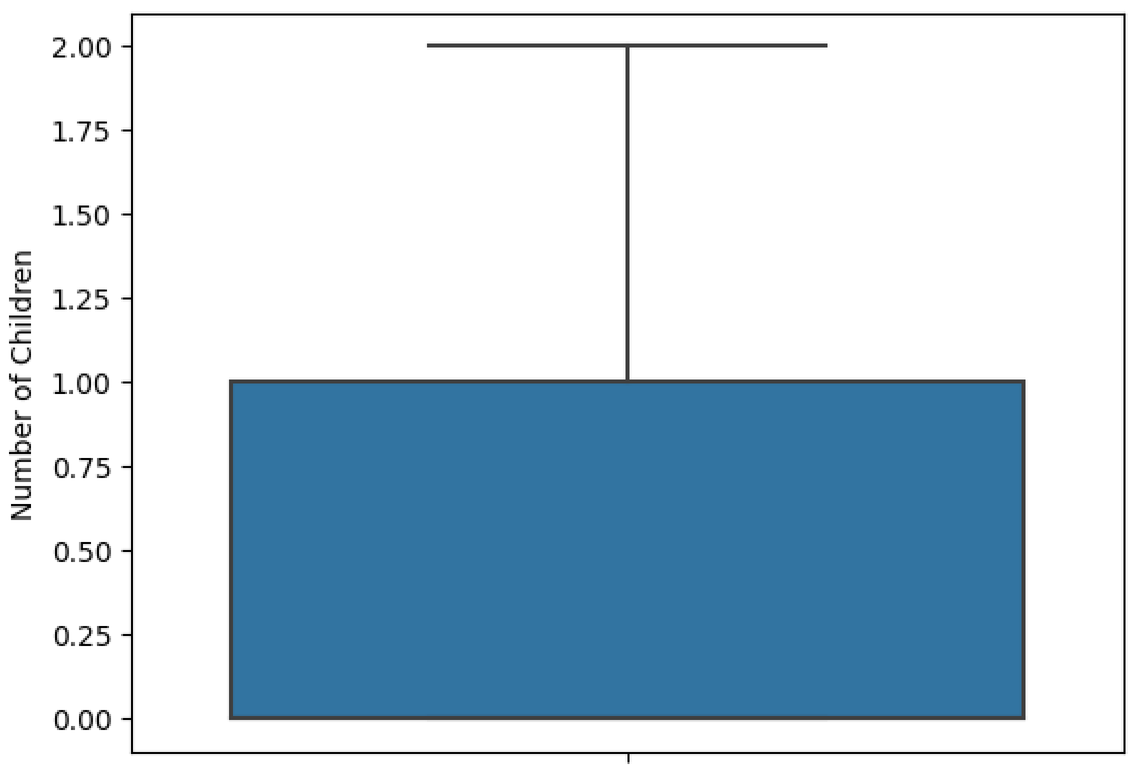
In [44]:
```
df["Eligibility"].replace({"Approved":1,"Rejected":0},inplace=True)
Eligibility_values = {"Approved":1,"Rejected":0}
```

In [45]:
```
df.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 164 entries, 0 to 163
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Age                    164 non-null    int64  
 1   Gender                 164 non-null    int64  
 2   Income                 164 non-null    int64  
 3   Education              164 non-null    int64  
 4   Marital Status         164 non-null    int64  
 5   Number of Children     164 non-null    int64  
 6   Home Ownership         164 non-null    int64  
 7   Credit Score           164 non-null    int64  
 8   Loan amount            164 non-null    float64
 9   EMI                    164 non-null    float64
 10  Inhand Sallary         164 non-null    int64  
 11  Eligibility            164 non-null    int64  
 12  Credit card limit (T)  164 non-null    int64  
dtypes: float64(2), int64(11)
memory usage: 16.8 KB
```

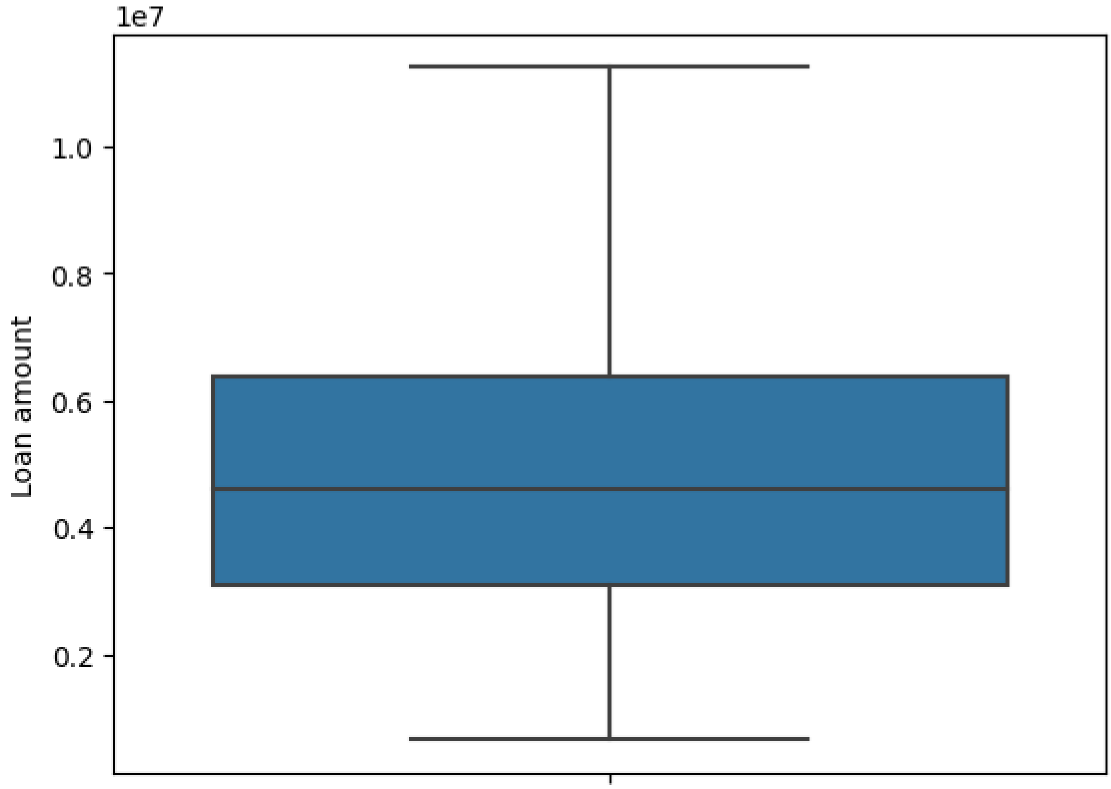
In [46]:
```
df.plot(kind='box',subplots=True,figsize=(20,10))
plt.show()
```



In [47]:
```
outlier_imputation(df["Number of Children"])
plt.show()
```



In [48]:
```
outlier_imputation(df["Loan amount"])
plt.show()
```



Feature Scaling

In [49]:
```
df_normal = normalization(df)
df_normal
```

Out[49]:

	Age	Gender	Income	Education	Marital Status	Number of Children	Home Ownership	Credit Score	Loan amount	
0	0.000000	1.0	0.181818	0.50	0.0	0.0	0.0	1.0	0.504249	0.16
1	0.178571	0.0	0.545455	0.75	1.0	1.0	1.0	1.0	0.881964	0.53
2	0.357143	1.0	0.363636	1.00	1.0	0.5	1.0	1.0	0.504249	0.35
3	0.535714	0.0	0.727273	0.00	0.0	0.0	1.0	1.0	0.645892	0.72
4	0.714286	1.0	0.545455	0.50	1.0	0.0	1.0	1.0	0.315392	0.53
...	...	...	...	...	...	...	...	...	...	
159	0.142857	1.0	0.018182	0.00	0.0	0.0	0.0	0.0	0.393535	0.42
160	0.321429	0.0	0.163636	0.25	0.0	0.0	0.0	0.5	0.314448	0.14
161	0.500000	1.0	0.272727	0.50	1.0	1.0	1.0	1.0	0.315392	0.25
162	0.678571	0.0	0.454545	0.75	0.0	0.0	1.0	1.0	0.301228	0.44
163	0.857143	1.0	0.381818	1.00	1.0	0.5	1.0	1.0	0.113314	0.37

164 rows × 13 columns

Model Building

In [50]:
```
model_name = 'Kmean Clustering'
kmean = KMeans(n_clusters=4,random_state=11)
y_pred = kmean.fit_predict(df_normal)
y_pred
```

Out[50]:
```
array([2, 0, 0, 1, 0, 1, 3, 2, 0, 1, 0, 1, 3, 2, 0, 1, 0, 1, 3, 3, 0, 1,
       0, 1, 3, 3, 0, 1, 0, 2, 0, 0, 1, 0, 1, 3, 2, 0, 1, 0, 1, 3, 2, 0,
       1, 0, 1, 3, 2, 0, 1, 0, 1, 3, 3, 0, 1, 0, 2, 0, 0, 2, 0, 0, 1, 0,
       1, 3, 2, 0, 1, 0, 1, 3, 2, 0, 1, 0, 1, 3, 2, 0, 1, 0, 1, 3, 3, 0,
       1, 0, 2, 0, 0, 2, 0, 0, 1, 0, 1, 3, 2, 0, 1, 0, 1, 3, 2, 0, 1, 0,
       1, 3, 2, 0, 1, 0, 1, 3, 3, 0, 1, 0, 2, 0, 0, 2, 0, 0, 2, 0, 2, 0,
       2, 0, 1, 2, 0, 0, 1, 0, 1, 3, 2, 0, 1, 0, 1, 3, 2, 0, 1, 0, 1, 3,
       2, 0, 1, 0, 1, 3, 3, 0, 1, 0])
```

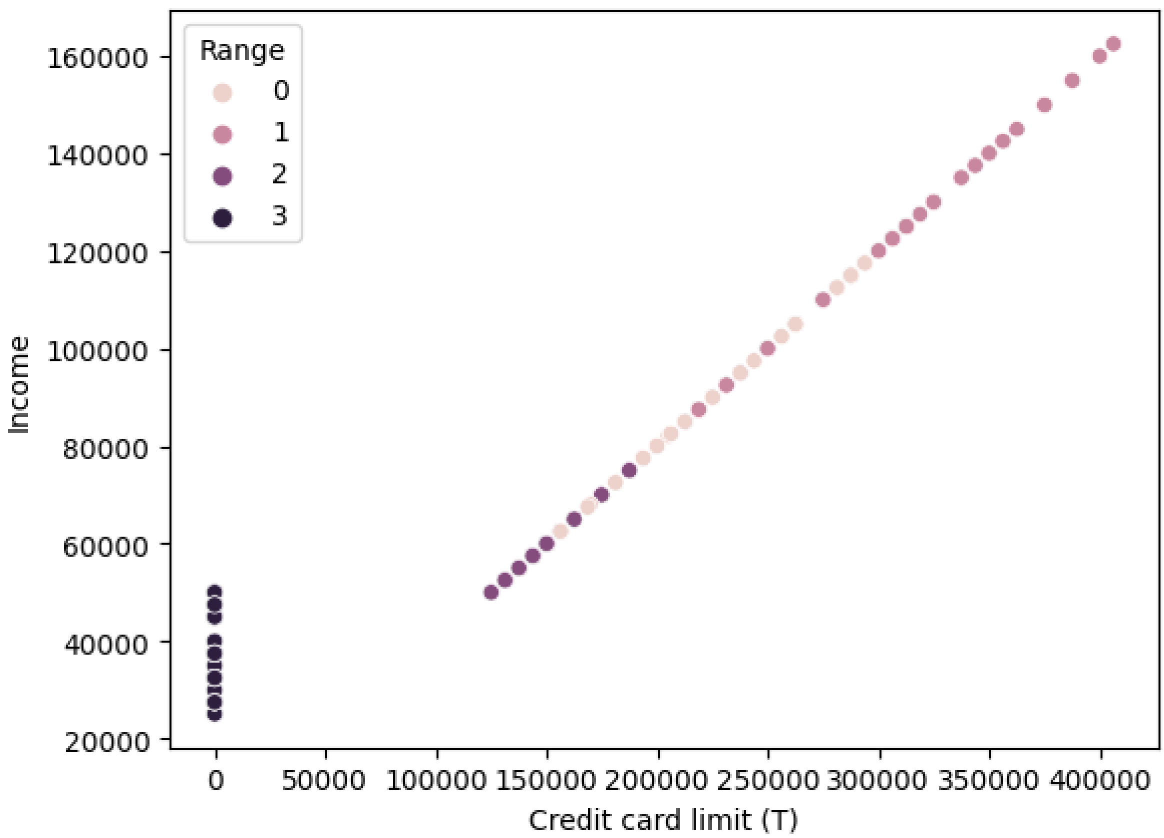
In [51]:
```
df["Range"] = y_pred
df.head(40)
```

Out[51]:

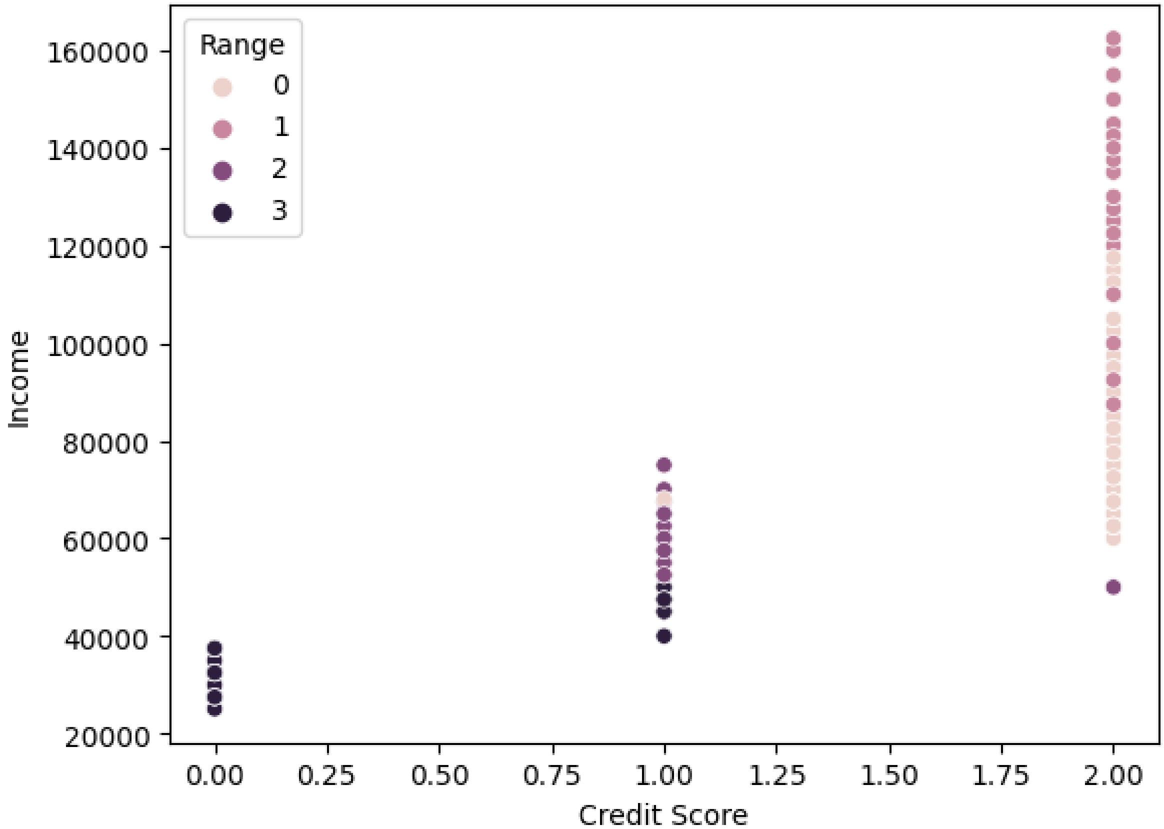
	Age	Gender	Income	Education	Marital Status	Number of Children	Home Ownership	Credit Score	Loan amount	
0	25	1	50000	2	0	0	0	2	6.000000e+06	16666.
1	30	0	100000	3	1	2	1	2	1.000000e+07	33333.
2	35	1	75000	4	1	1	1	2	6.000000e+06	25000.
3	40	0	125000	0	0	0	1	2	7.500000e+06	41666.
4	45	1	100000	2	1	0	1	2	4.000000e+06	33333.
5	50	0	150000	3	1	0	1	2	3.000000e+06	50000.
6	26	1	40000	1	1	0	0	1	4.640000e+06	13333.
7	31	0	60000	2	0	0	0	1	5.760000e+06	20000.
8	36	1	80000	3	1	2	1	2	6.080000e+06	26666.
9	41	0	105000	4	0	0	1	2	5.880000e+06	35000.
10	46	1	90000	0	1	1	1	2	3.240000e+06	30000.
11	51	0	135000	2	1	0	1	2	2.160000e+06	45000.
12	27	1	35000	0	0	0	0	0	1.120000e+06	11666.
13	32	0	55000	1	0	0	0	1	5.060000e+06	18333.
14	37	1	70000	2	1	2	1	2	5.040000e+06	23333.
15	42	0	95000	3	0	0	1	2	4.940000e+06	31666.
16	47	1	85000	4	1	1	1	2	2.720000e+06	28333.
17	52	0	125000	0	1	0	1	2	1.500000e+06	41666.
18	28	1	30000	1	0	0	1	0	8.400000e+05	10000.
19	33	0	50000	0	0	0	0	1	4.400000e+06	16666.
20	38	1	65000	2	1	2	1	2	4.420000e+06	21666.
21	43	0	80000	3	0	0	1	2	3.840000e+06	26666.
22	48	1	70000	4	1	1	1	2	1.960000e+06	23333.
23	53	0	115000	1	1	0	1	2	9.200000e+05	38333.
24	29	1	25000	0	1	0	0	0	4.827535e+06	28485.
25	34	0	45000	1	0	0	0	1	3.780000e+06	15000.
26	39	1	60000	2	1	2	1	2	3.840000e+06	20000.
27	44	0	75000	3	0	0	1	2	3.300000e+06	25000.
28	49	1	65000	4	1	1	1	2	1.560000e+06	21666.
29	25	1	55000	2	0	0	0	1	6.600000e+06	18333.
30	30	0	105000	3	1	2	1	2	1.050000e+07	35000.
31	35	1	80000	4	1	1	1	2	6.400000e+06	26666.
32	40	0	130000	0	0	0	1	2	7.800000e+06	43333.
33	45	1	105000	2	1	0	1	2	4.200000e+06	35000.

	Age	Gender	Income	Education	Marital Status	Number of Children	Home Ownership	Credit Score	Loan amount	
34	50	0	155000	3	1	0	1	2	3.100000e+06	51666.
35	26	1	45000	1	0	0	0	1	5.220000e+06	15000.
36	31	0	65000	2	0	0	0	1	6.240000e+06	21666.
37	36	1	85000	3	1	2	1	2	6.460000e+06	28333.
38	41	0	110000	4	0	0	1	2	6.160000e+06	36666.
39	46	1	95000	0	1	1	1	2	3.420000e+06	31666.

In [52]:
```
sns.scatterplot(data=df,x='Credit card limit (T)',y='Income',hue='Range')
plt.show()
```



In [53]:
```
sns.scatterplot(data=df,x='Credit Score',y='Income',hue='Range')
plt.show()
```



In [54]:
```
df['Range'].value_counts()
```

Out[54]:
```
0    66
1    46
2    26
3    26
Name: Range, dtype: int64
```

In [55]:
```
# train test split

x = df.drop(['Range','Eligibility','Credit card limit (T)'],axis=1)
y = df['Range']

x_train,x_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_s
```

In [56]:
```
x_train, y_train = SMOTE(k_neighbors=1,random_state=15).fit_resample(x_train,y_
y_train.value_counts()
```

Out[56]:
```
0    50
3    50
1    50
2    50
Name: Range, dtype: int64
```

In [57]:
```
x_test, y_test = SMOTE(k_neighbors=1).fit_resample(x_test,y_test)
y_test.value_counts()
```

Out[57]:
```
0    16
3    16
1    16
2    16
Name: Range, dtype: int64
```

Feature Selection

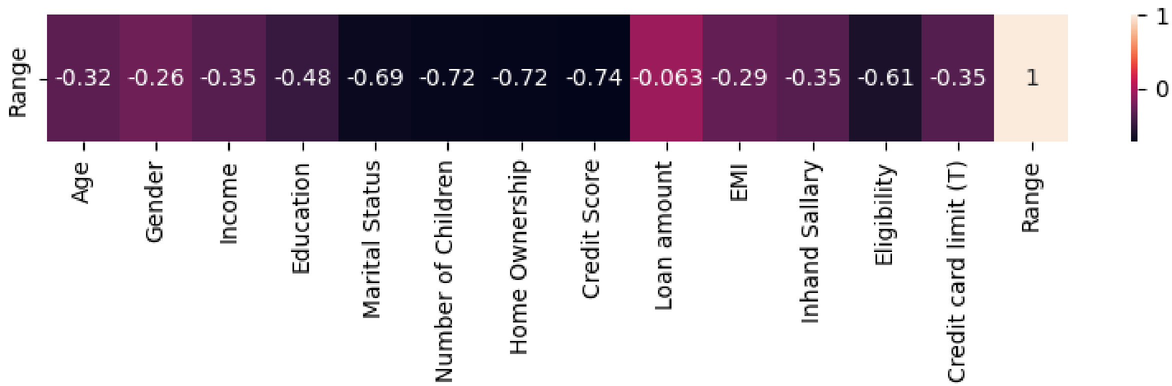
1.kendall coc

In [58]:
```
cor_df = df.copy()
cor_df.corr('kendall').tail(1)
```

Out[58]:

	Age	Gender	Income	Education	Marital Status	Number of Children	Home Ownership	Credit Score
Range	-0.323163	-0.256565	-0.350023	-0.483971	-0.690008	-0.719198	-0.721103	-0.741026

In [59]:
```
plt.figure(figsize=(10,1))
sns.heatmap(cor_df.corr('kendall').tail(1),annot=True)
plt.show()
```



2. ANOVA Test

In [60]:
```
from sklearn.feature_selection import f_classif

f_val,p_val = f_classif(x_train,y_train)
```

In [61]:
```
df4 = pd.DataFrame({"f_val":f_val, "p_val":np.around(p_val,5)}, index=x_train.
df4
```

Out[61]:

	f_val	p_val
Age	142.773777	0.0
Gender	88.984247	0.0
Income	281.682621	0.0
Education	40.632659	0.0
Marital Status	90.020232	0.0
Number of Children	242.598559	0.0
Home Ownership	443.035306	0.0
Credit Score	496.340067	0.0
Loan amount	22.868661	0.0
EMI	143.465497	0.0
Inhand Sallary	257.711203	0.0

Model Building

Logistic regression

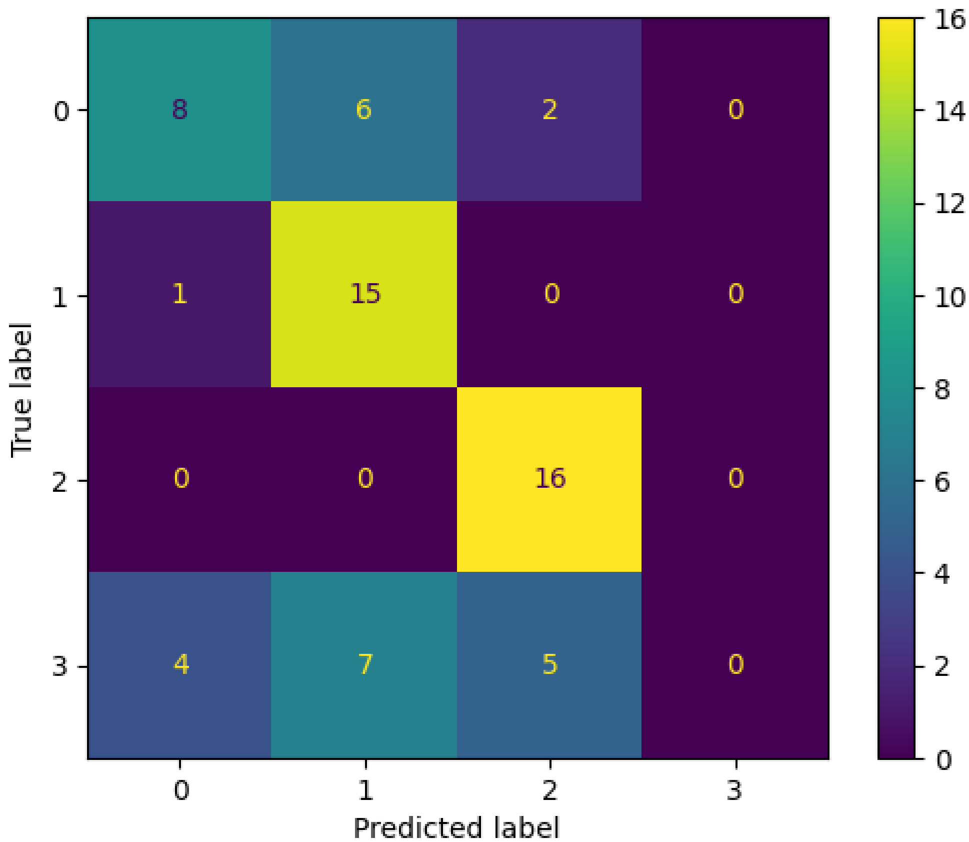
In [57]:
```
log_reg_model = log_reg_model_training(x_train,y_train)
log_reg_model
```

Out[57]:

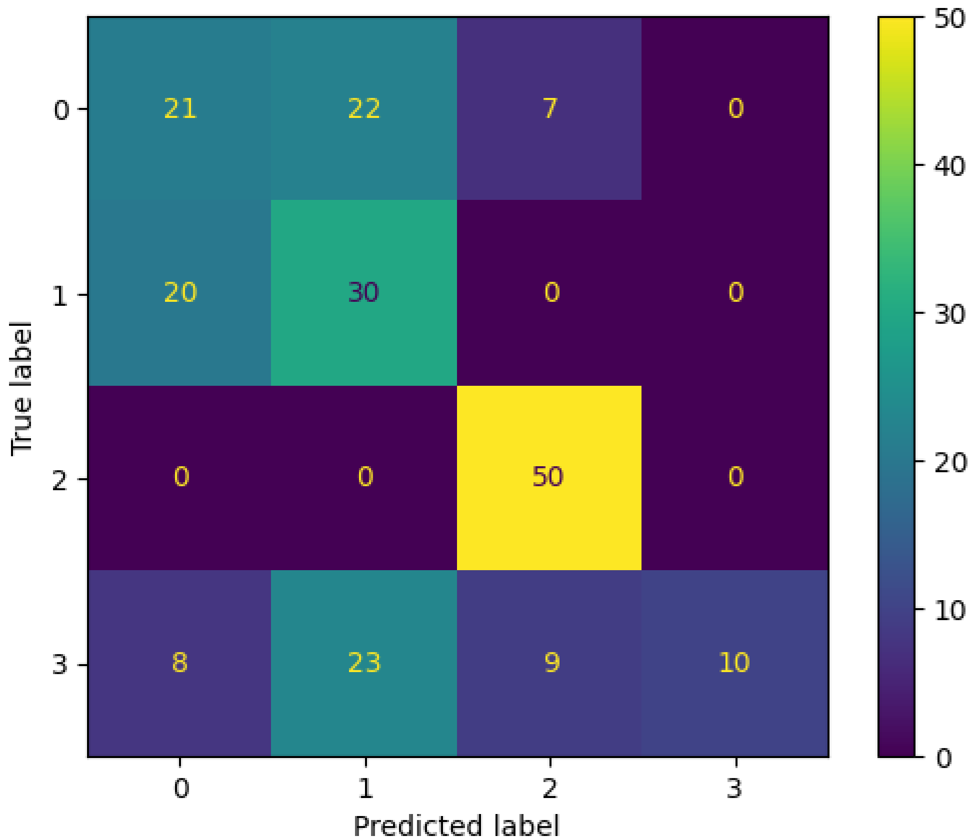
```
▾ LogisticRegression
LogisticRegression()
```

In [58]:
```
# evaluation for testing data

model_evaluation_testing(log_reg_model,x_test,y_test)

# evaluation for training data

model_evaluation_training(log_reg_model,x_train,y_train)
```

```
************************************************** Testing Data *************
*************************************
The confusion Matrix is :
 [[ 8  6  2  0]
 [ 1 15  0  0]
 [ 0  0 16  0]
 [ 4  7  5  0]]
```



```
==================================================
The Accuracy is :  0.609375
==================================================
The Classification report is :
               precision    recall  f1-score   support

           0       0.62      0.50      0.55        16
           1       0.54      0.94      0.68        16
           2       0.70      1.00      0.82        16
           3       0.00      0.00      0.00        16

    accuracy                           0.61        64
   macro avg       0.46      0.61      0.51        64
weighted avg       0.46      0.61      0.51        64

************************************************** Training Data ************
**************************************
The confusion matrix is :
 [[21 22  7  0]
 [20 30  0  0]
 [ 0  0 50  0]
 [ 8 23  9 10]]
```



```
====================================================
The accuracy is :  0.555
====================================================
The classification Report is :
              precision    recall  f1-score   support

           0       0.43      0.42      0.42        50
           1       0.40      0.60      0.48        50
           2       0.76      1.00      0.86        50
           3       1.00      0.20      0.33        50

    accuracy                           0.56       200
   macro avg       0.65      0.56      0.52       200
weighted avg       0.65      0.56      0.52       200
```

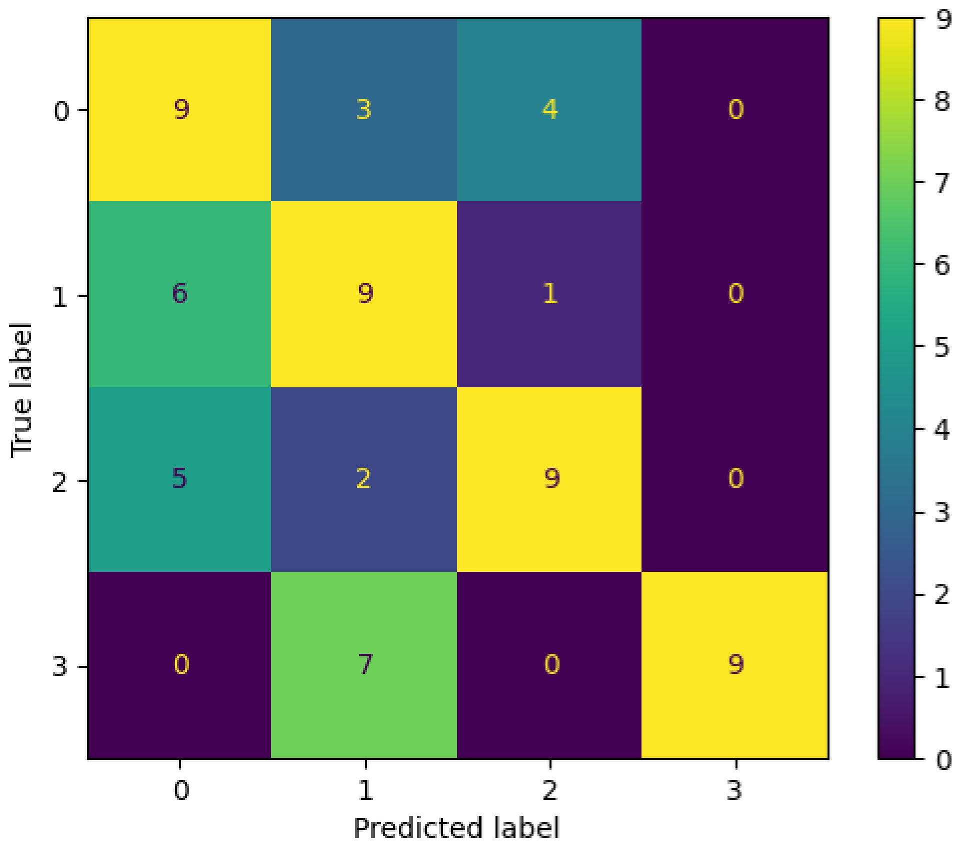
KNN

In [59]:
```
model_name = "KNN Model"
knn_clf_model = knn_model_building(x_train,y_train)
knn_clf_model
```

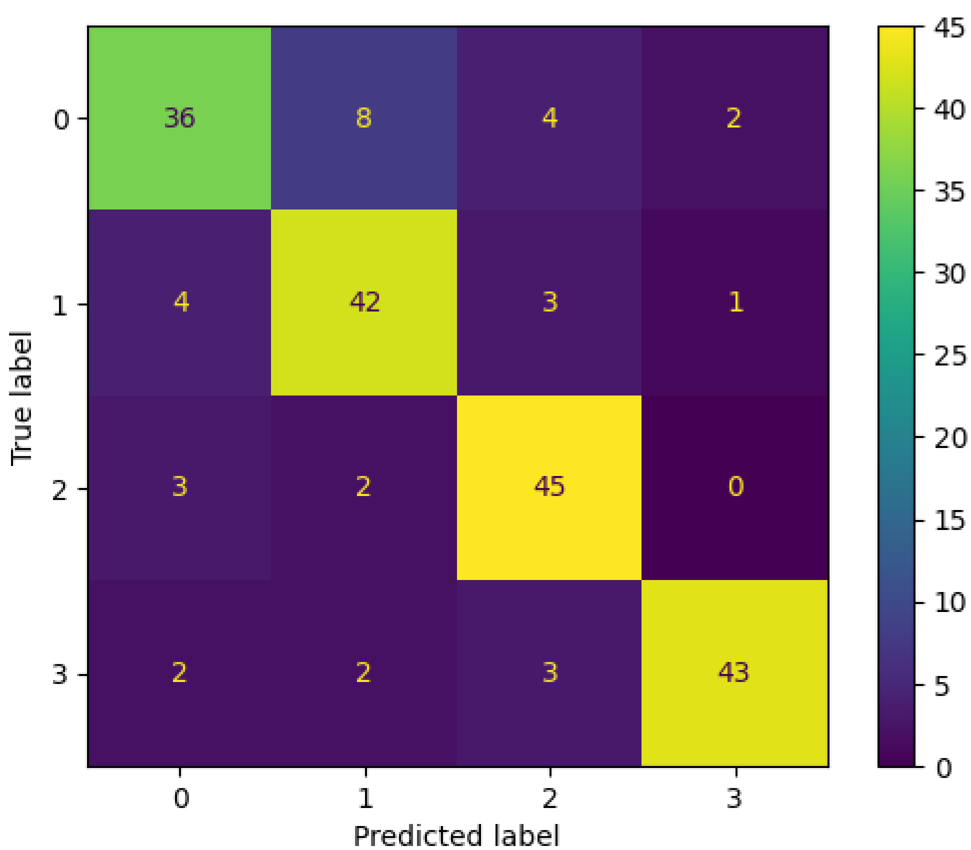
Out[59]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [60]:
```
# evaluation for testing data

model_evaluation_testing(knn_clf_model,x_test,y_test)

# evaluation for training data

model_evaluation_training(knn_clf_model,x_train,y_train)
```

```
************************************************* Testing Data *************
************************************
The confusion Matrix is :
 [[9 3 4 0]
 [6 9 1 0]
 [5 2 9 0]
 [0 7 0 9]]
```



```
==================================================
The Accuracy is :  0.5625
==================================================
The Classification report is :
               precision    recall  f1-score   support

           0       0.45      0.56      0.50        16
           1       0.43      0.56      0.49        16
           2       0.64      0.56      0.60        16
           3       1.00      0.56      0.72        16

    accuracy                           0.56        64
   macro avg       0.63      0.56      0.58        64
weighted avg       0.63      0.56      0.58        64

************************************************** Training Data ************
**************************************
The confusion matrix is :
 [[36  8  4  2]
 [ 4 42  3  1]
 [ 3  2 45  0]
 [ 2  2  3 43]]
```



```
==================================================
The accuracy is :  0.83
==================================================
The classification Report is :
              precision    recall  f1-score   support

           0       0.80      0.72      0.76        50
           1       0.78      0.84      0.81        50
           2       0.82      0.90      0.86        50
           3       0.93      0.86      0.90        50

    accuracy                           0.83       200
   macro avg       0.83      0.83      0.83       200
weighted avg       0.83      0.83      0.83       200
```

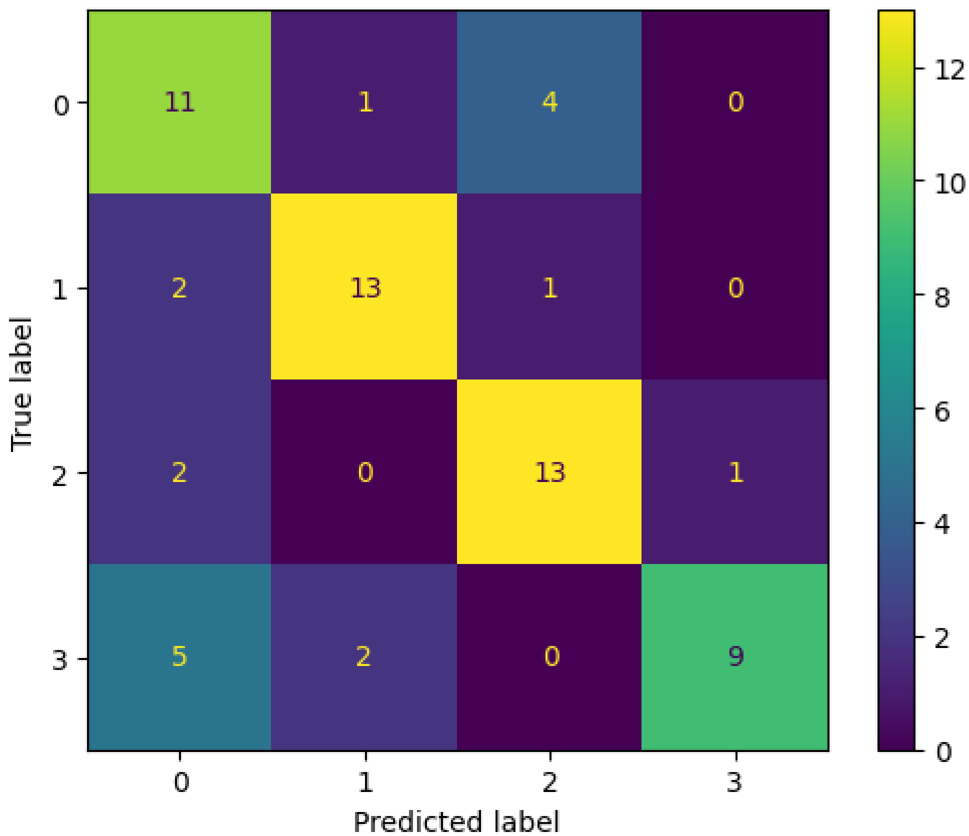
Hyper-Parameter Tunning

In [61]:
```
gscv_knn_clf_model = knn_gscv_best_estimator(x_train,y_train)
gscv_knn_clf_model
```

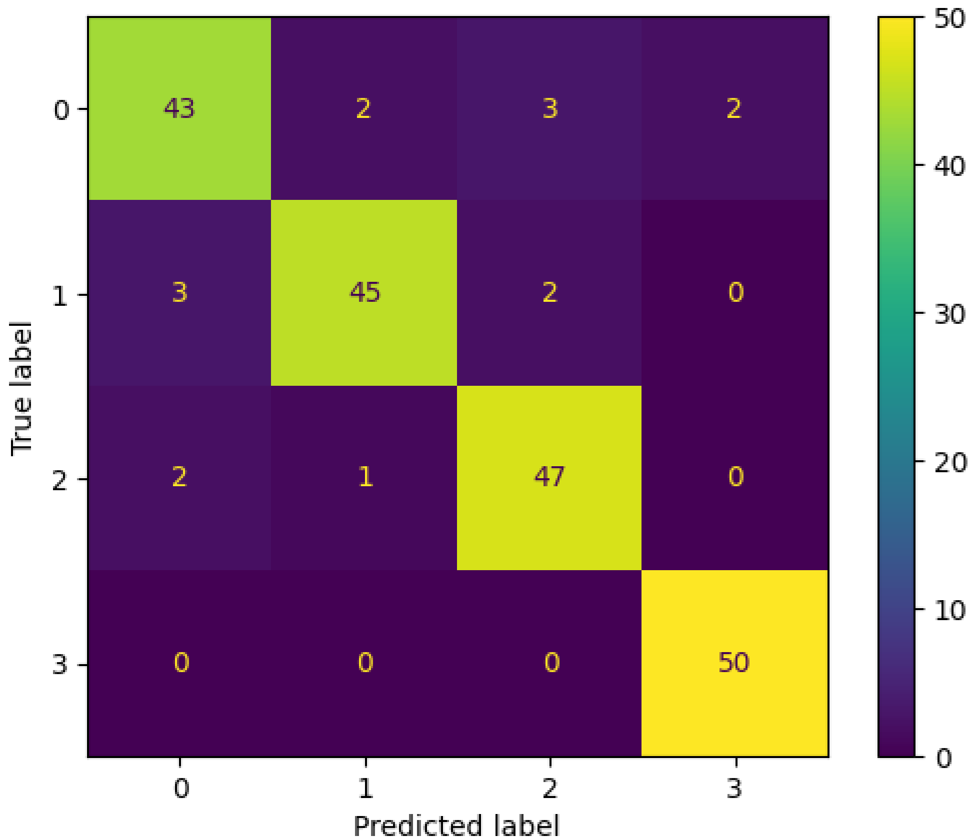
Out[61]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3, p=1)
```

In [62]:
```
# evaluation for testing data

model_evaluation_testing(gscv_knn_clf_model,x_test,y_test)

# evaluation for training data

model_evaluation_training(gscv_knn_clf_model,x_train,y_train)
```

```
********************************************** Testing Data ************
**********************************
The confusion Matrix is :
 [[11  1  4  0]
 [ 2 13  1  0]
 [ 2  0 13  1]
 [ 5  2  0  9]]
```



```
==================================================
The Accuracy is :  0.71875
==================================================
The Classification report is :
               precision    recall  f1-score   support

           0       0.55      0.69      0.61        16
           1       0.81      0.81      0.81        16
           2       0.72      0.81      0.76        16
           3       0.90      0.56      0.69        16

    accuracy                           0.72        64
   macro avg       0.75      0.72      0.72        64
weighted avg       0.75      0.72      0.72        64

************************************************** Training Data ************
**************************************
The confusion matrix is :
 [[43  2  3  2]
 [ 3 45  2  0]
 [ 2  1 47  0]
 [ 0  0  0 50]]
```



```
==================================================
The accuracy is :  0.925
==================================================
The classification Report is :
               precision    recall  f1-score   support

           0       0.90      0.86      0.88        50
           1       0.94      0.90      0.92        50
           2       0.90      0.94      0.92        50
           3       0.96      1.00      0.98        50

    accuracy                           0.93       200
   macro avg       0.92      0.93      0.92       200
weighted avg       0.92      0.93      0.92       200
```

In [68]:

```
rscv_knn_clf_model = knn_rscv_best_estimator(x_train,y_train)
rscv_knn_clf_model
```

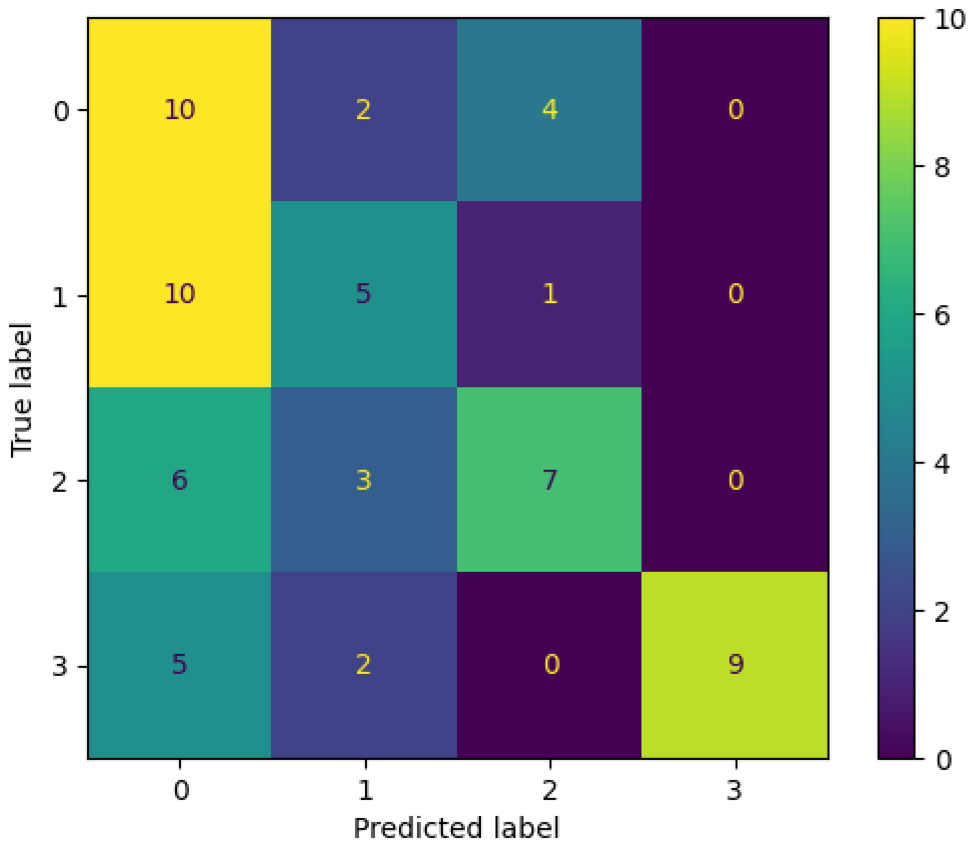
Out[68]:

```
KNeighborsClassifier(n_neighbors=4)
```

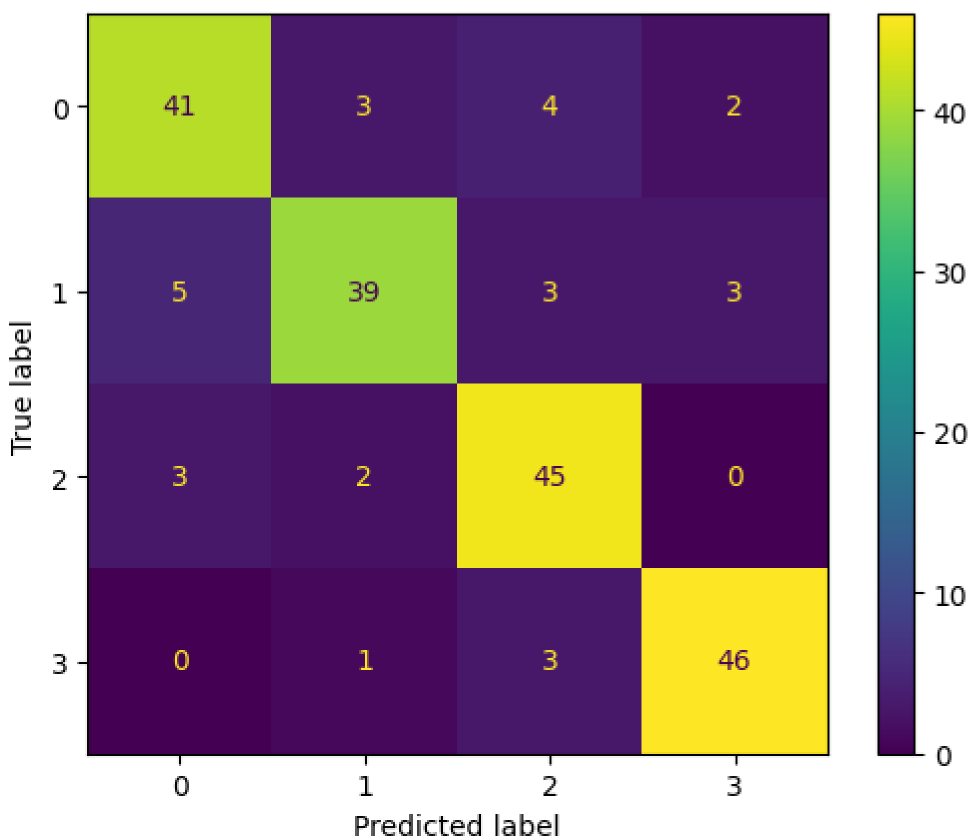
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [69]:
```
# evaluation for testing data

model_evaluation_testing(rscv_knn_clf_model,x_test,y_test)

# evaluation for training data

model_evaluation_training(rscv_knn_clf_model,x_train,y_train)
```

```
************************************************* Testing Data *************************************************
The confusion Matrix is :
 [[10  2  4  0]
 [10  5  1  0]
 [ 6  3  7  0]
 [ 5  2  0  9]]
```



```
==================================================
The Accuracy is :  0.484375
==================================================
The Classification report is :
               precision    recall  f1-score   support

           0       0.32      0.62      0.43        16
           1       0.42      0.31      0.36        16
           2       0.58      0.44      0.50        16
           3       1.00      0.56      0.72        16

    accuracy                           0.48        64
   macro avg       0.58      0.48      0.50        64
weighted avg       0.58      0.48      0.50        64

************************************************** Training Data ************
**************************************
The confusion matrix is :
 [[41  3  4  2]
 [ 5 39  3  3]
 [ 3  2 45  0]
 [ 0  1  3 46]]
```



```
==================================================
The accuracy is :  0.855
==================================================
The classification Report is :
              precision    recall  f1-score   support

           0       0.84      0.82      0.83        50
           1       0.87      0.78      0.82        50
           2       0.82      0.90      0.86        50
           3       0.90      0.92      0.91        50

    accuracy                           0.85       200
   macro avg       0.86      0.85      0.85       200
weighted avg       0.86      0.85      0.85       200
```

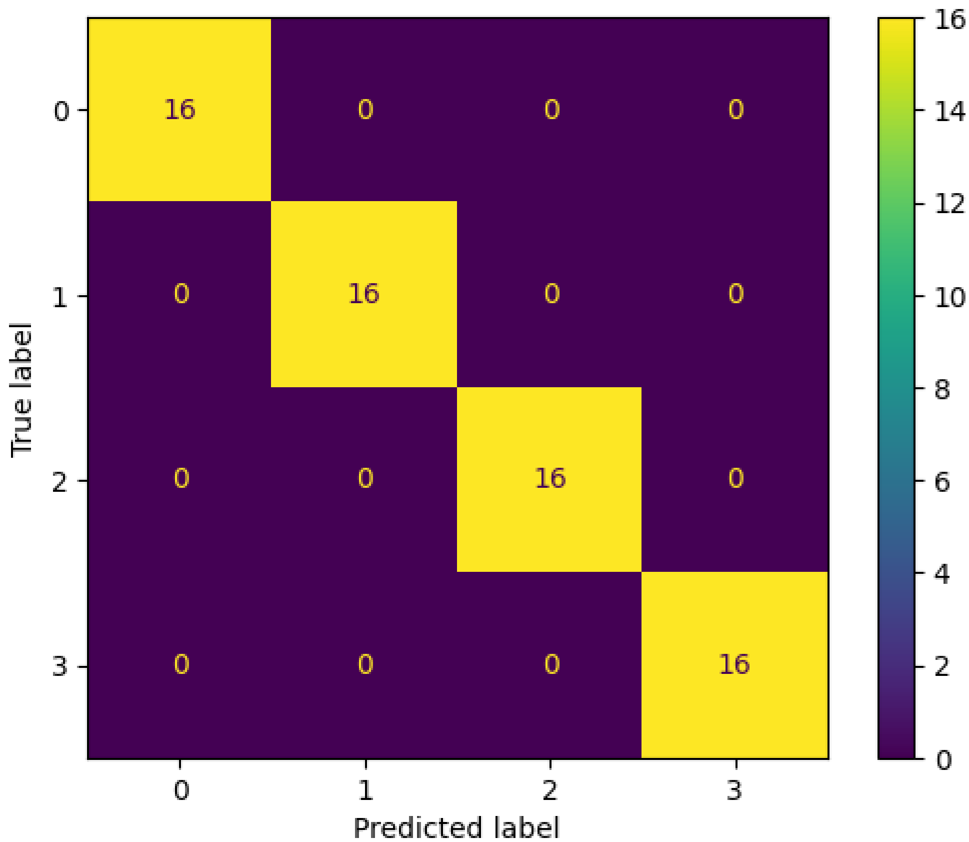
Decision Tree

In [62]:
```
model_name = 'Decision Tree Classifier'
dt_clf = dt_clf_training(x_train,y_train)
dt_clf
```

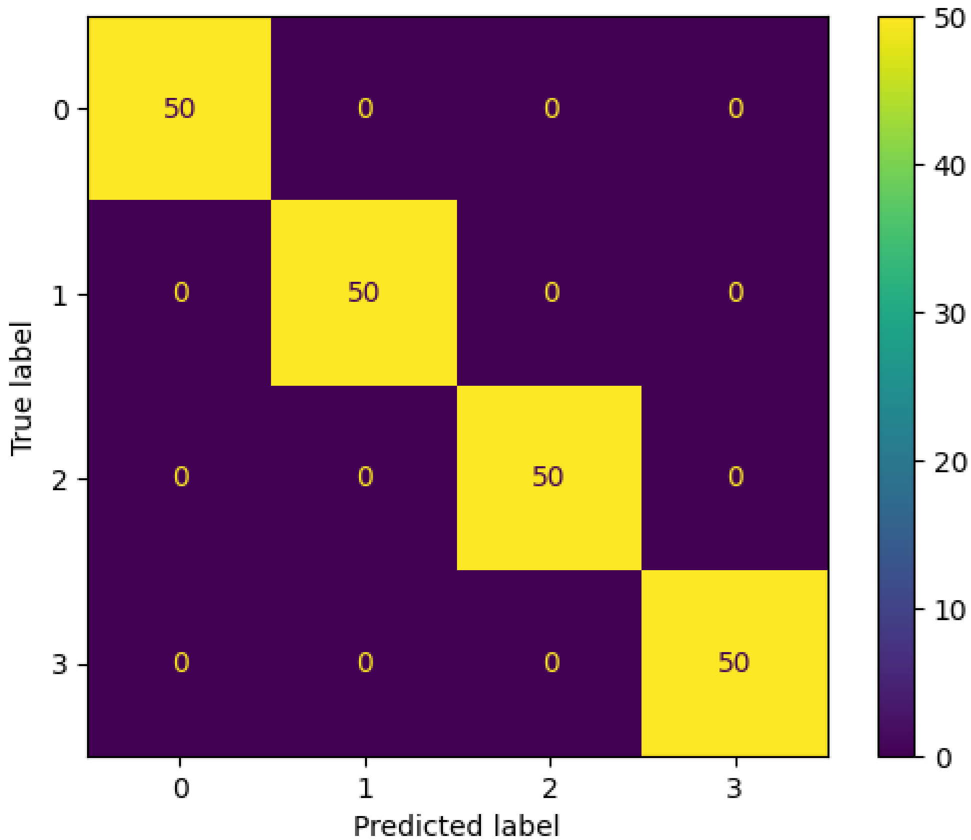
Out[62]:
```
▾         DecisionTreeClassifier
DecisionTreeClassifier(random_state=18)
```

In [63]:
```
# evaluation for testing data

model_evaluation_testing(dt_clf,x_test,y_test)

# evaluation for training data

model_evaluation_training(dt_clf,x_train,y_train)
```

```
************************************************ Testing Data *************
************************************
The confusion Matrix is :
 [[16  0  0  0]
 [ 0 16  0  0]
 [ 0  0 16  0]
 [ 0  0  0 16]]
```



```
==================================================
The Accuracy is :  1.0
==================================================
The Classification report is :
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      1.00      1.00        16
           2       1.00      1.00      1.00        16
           3       1.00      1.00      1.00        16

    accuracy                           1.00        64
   macro avg       1.00      1.00      1.00        64
weighted avg       1.00      1.00      1.00        64

************************************************** Training Data ************
**************************************
The confusion matrix is :
 [[50  0  0  0]
 [ 0 50  0  0]
 [ 0  0 50  0]
 [ 0  0  0 50]]
```



```
==================================================
The accuracy is :  1.0
==================================================
The classification Report is :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      1.00      1.00        50
           2       1.00      1.00      1.00        50
           3       1.00      1.00      1.00        50

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```

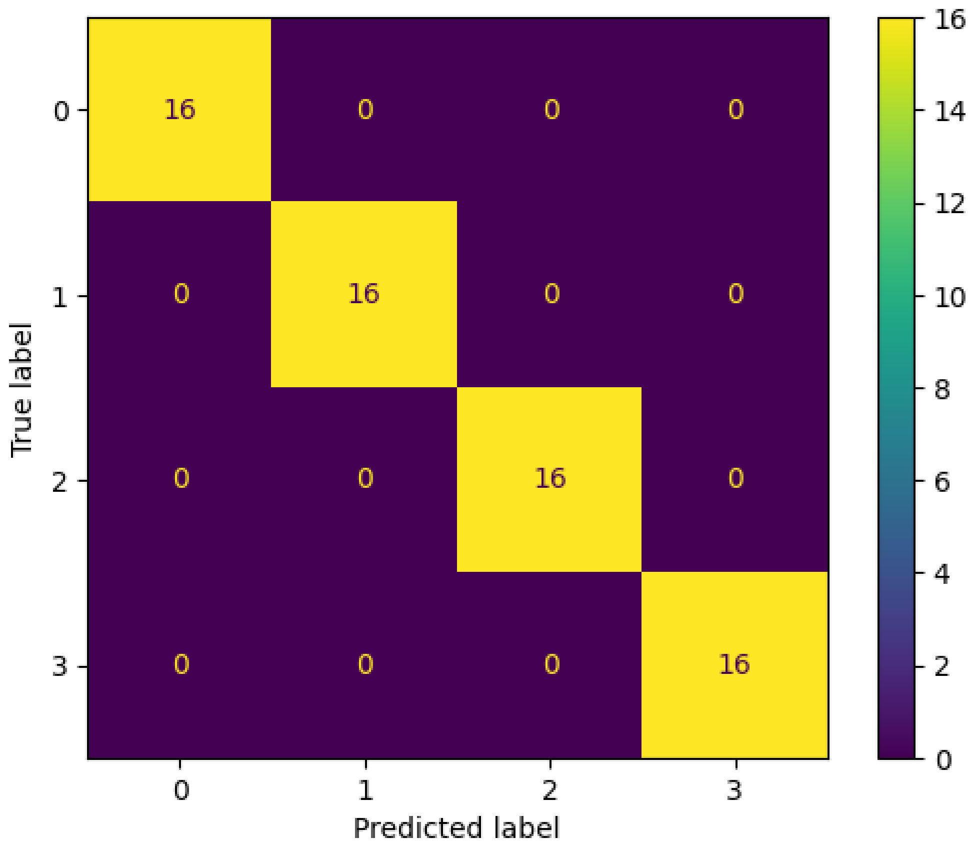
Hyper-Parameter Tunning

In [64]:
```
gscv_dt_clf = dt_gscv_best_estimator(x_train,y_train)
gscv_dt_clf
```

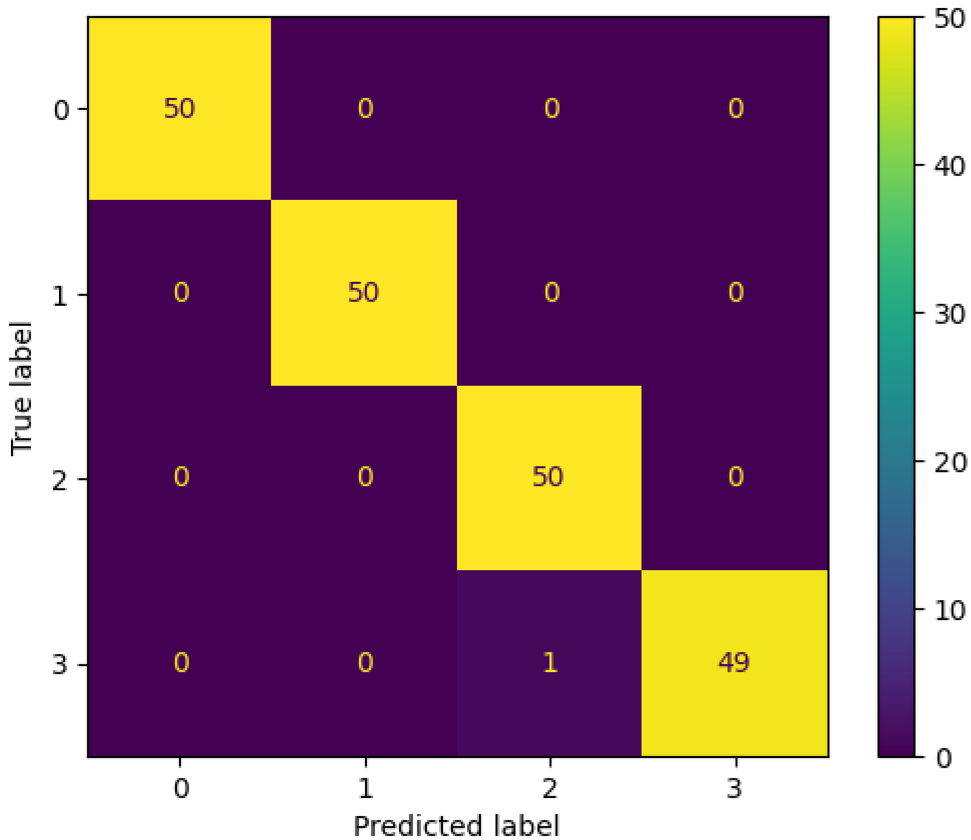
Out[64]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4, min_samples_leaf=2)
```

In [65]:
```
# evaluation for testing data

model_evaluation_testing(gscv_dt_clf,x_test,y_test)

# evaluation for training data

model_evaluation_training(gscv_dt_clf,x_train,y_train)
```

```
************************************************ Testing Data *************************************************
The confusion Matrix is :
 [[16  0  0  0]
 [ 0 16  0  0]
 [ 0  0 16  0]
 [ 0  0  0 16]]
```



```
==================================================
The Accuracy is :  1.0
==================================================
The Classification report is :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      1.00      1.00        16
           2       1.00      1.00      1.00        16
           3       1.00      1.00      1.00        16

    accuracy                           1.00        64
   macro avg       1.00      1.00      1.00        64
weighted avg       1.00      1.00      1.00        64

************************************************** Training Data ************
**************************************
The confusion matrix is :
 [[50  0  0  0]
 [ 0 50  0  0]
 [ 0  0 50  0]
 [ 0  0  1 49]]
```



```
==================================================
The accuracy is :  0.995
==================================================
The classification Report is :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      1.00      1.00        50
           2       0.98      1.00      0.99        50
           3       1.00      0.98      0.99        50

    accuracy                           0.99       200
   macro avg       1.00      0.99      0.99       200
weighted avg       1.00      0.99      0.99       200
```

In [66]:

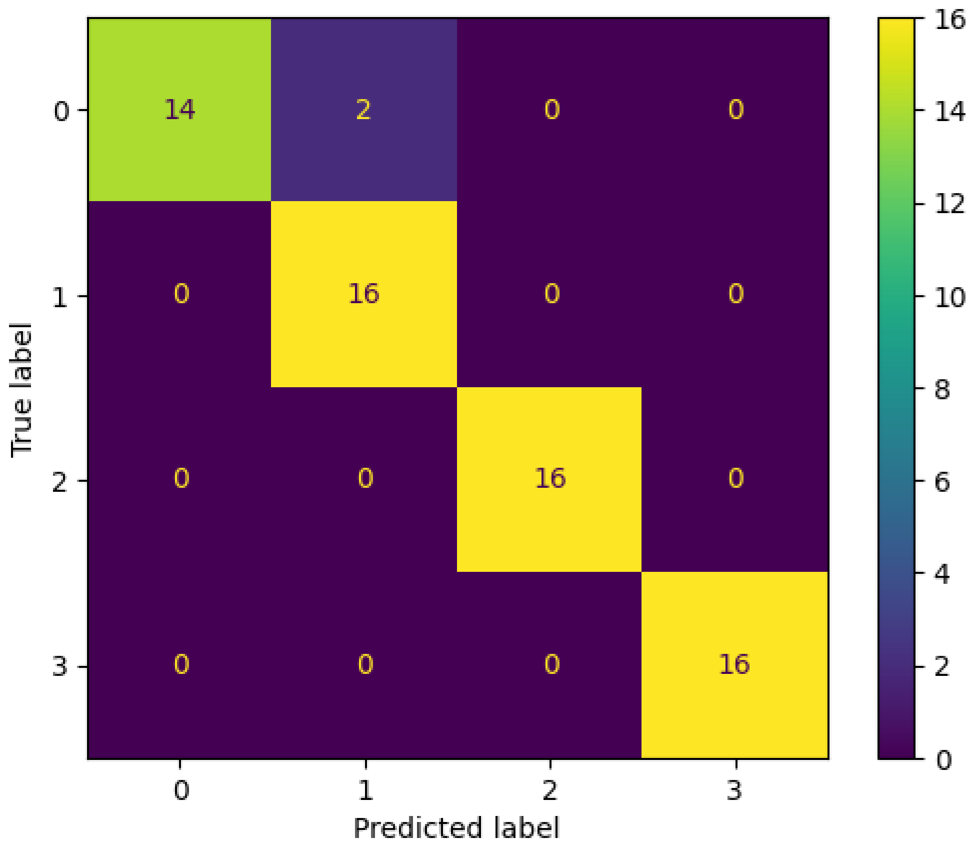
```
rscv_dt_clf = dt_rscv_best_estimator(x_train,y_train)
rscv_dt_clf
```

Out[66]:

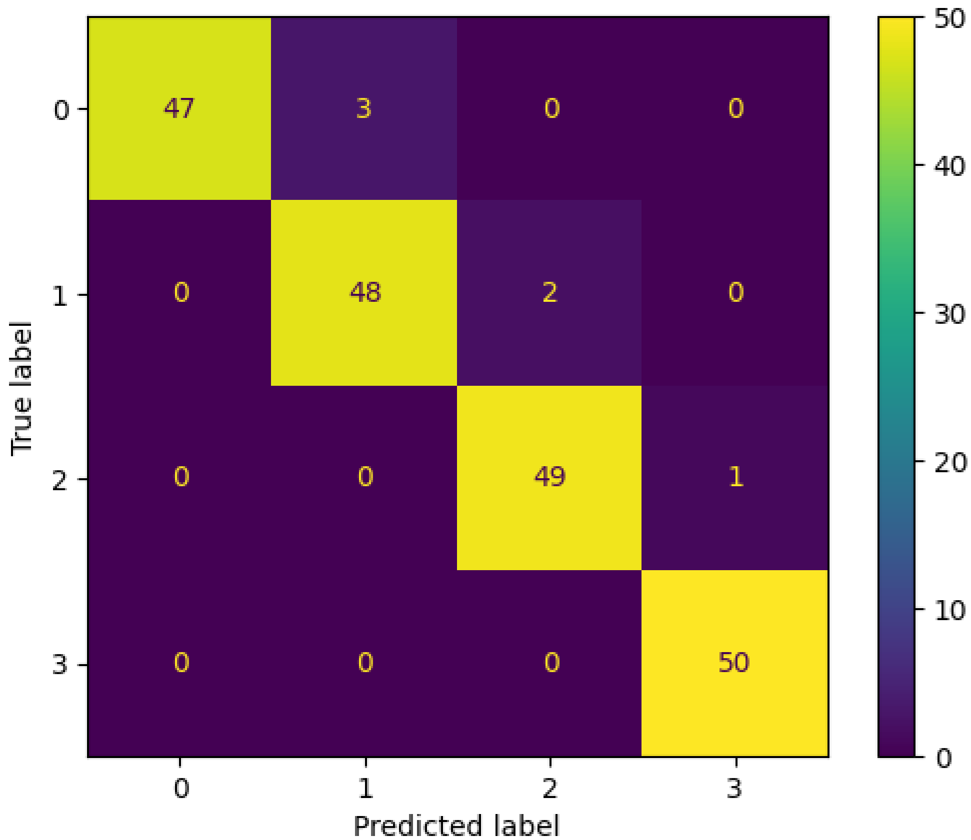
```
DecisionTreeClassifier

DecisionTreeClassifier(max_depth=6, min_samples_leaf=14, min_samples_split=1
1)
```

In [67]:
```
# evaluation for testing data

model_evaluation_testing(rscv_dt_clf,x_test,y_test)

# evaluation for training data

model_evaluation_training(rscv_dt_clf,x_train,y_train)
```

```
************************************************ Testing Data *************
************************************
The confusion Matrix is :
 [[14  2  0  0]
 [ 0 16  0  0]
 [ 0  0 16  0]
 [ 0  0  0 16]]
```



```
==================================================
The Accuracy is :  0.96875
==================================================
The Classification report is :
               precision    recall  f1-score   support

           0       1.00      0.88      0.93        16
           1       0.89      1.00      0.94        16
           2       1.00      1.00      1.00        16
           3       1.00      1.00      1.00        16

    accuracy                           0.97        64
   macro avg       0.97      0.97      0.97        64
weighted avg       0.97      0.97      0.97        64

************************************************** Training Data ************
**************************************
The confusion matrix is :
 [[47  3  0  0]
 [ 0 48  2  0]
 [ 0  0 49  1]
 [ 0  0  0 50]]
```



```
==================================================
The accuracy is :  0.97
==================================================
The classification Report is :
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        50
           1       0.94      0.96      0.95        50
           2       0.96      0.98      0.97        50
           3       0.98      1.00      0.99        50

    accuracy                           0.97       200
   macro avg       0.97      0.97      0.97       200
weighted avg       0.97      0.97      0.97       200
```

```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=14,
min_samples_split=10)
```

Random Forest

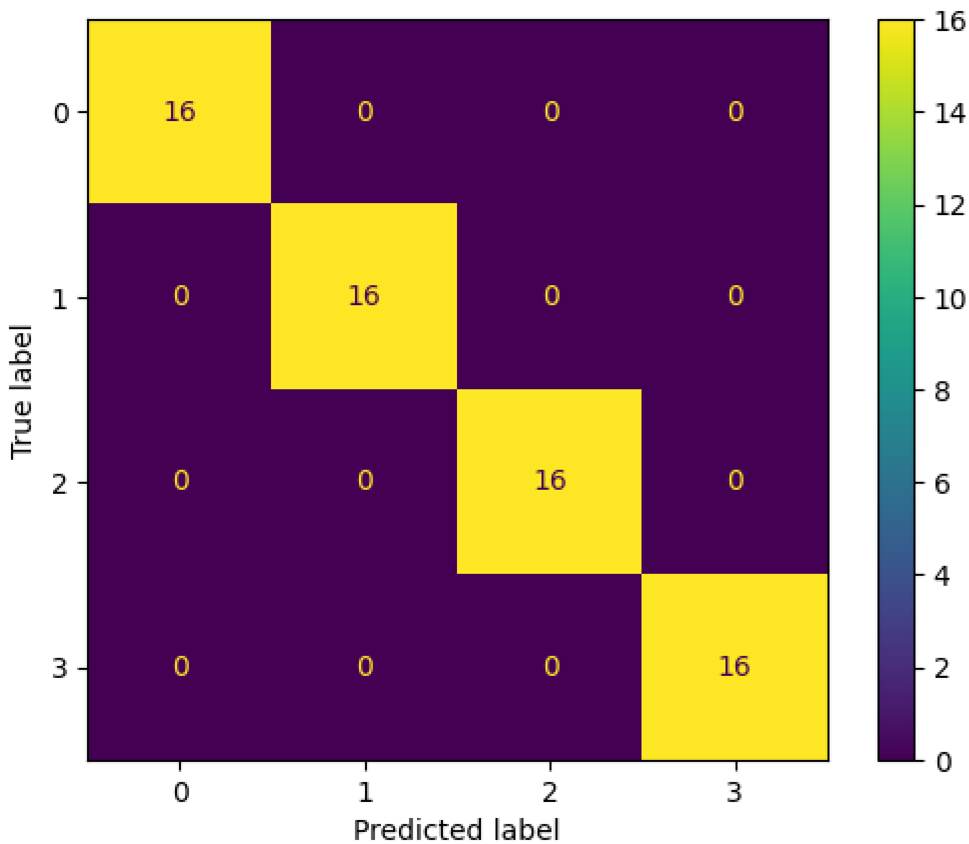
In [79]:
```
rf_clf = RandomForestClassifier(random_state=22)
rf_clf.fit(x_train,y_train)
```

Out[79]:
```
RandomForestClassifier(random_state=22)
```

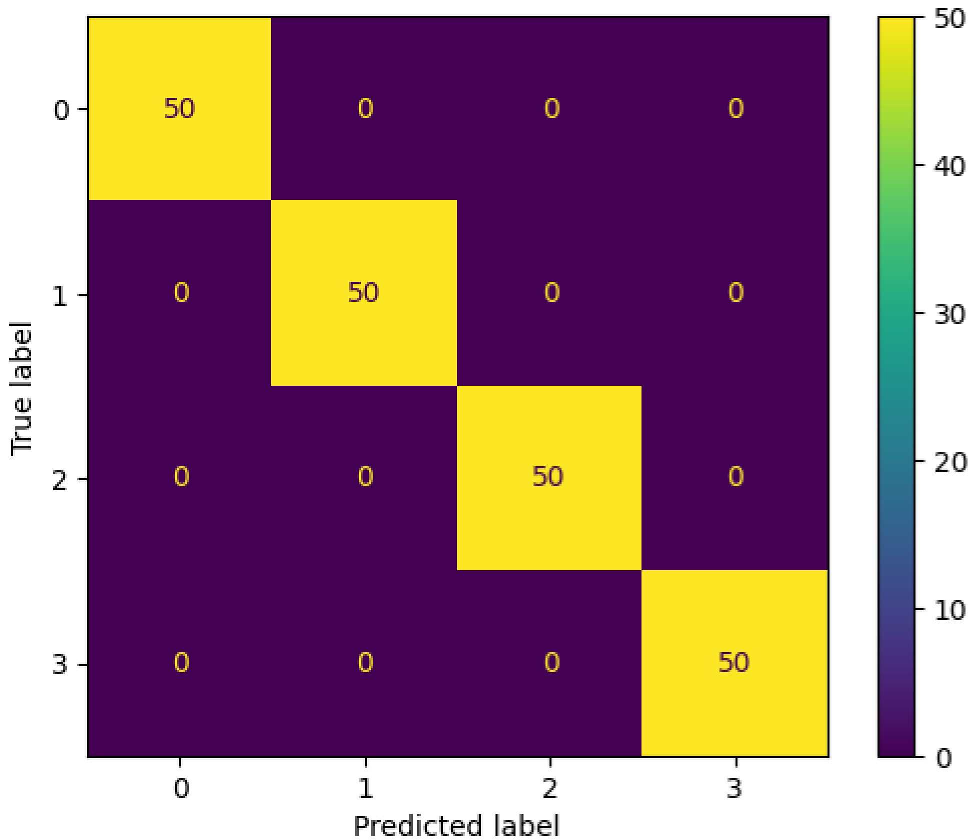
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [80]:
```
# evaluation for testing data

model_evaluation_testing(rf_clf,x_test,y_test)

# evaluation for training data

model_evaluation_training(rf_clf,x_train,y_train)
```

```
************************************************** Testing Data *************
*************************************
The confusion Matrix is :
 [[16  0  0  0]
 [ 0 16  0  0]
 [ 0  0 16  0]
 [ 0  0  0 16]]
```



```
==================================================
The Accuracy is :  1.0
==================================================
The Classification report is :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      1.00      1.00        16
           2       1.00      1.00      1.00        16
           3       1.00      1.00      1.00        16

    accuracy                           1.00        64
   macro avg       1.00      1.00      1.00        64
weighted avg       1.00      1.00      1.00        64

************************************************** Training Data ************
**************************************
The confusion matrix is :
 [[50  0  0  0]
 [ 0 50  0  0]
 [ 0  0 50  0]
 [ 0  0  0 50]]
```



```
==================================================
The accuracy is :  1.0
==================================================
The classification Report is :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      1.00      1.00        50
           2       1.00      1.00      1.00        50
           3       1.00      1.00      1.00        50

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```

Hyper-Parameter Tunning

```
gscv_rf_clf = rf_gscv_best_estimator(x_train,y_train)
gscv_rf_clf
```

```
# evaluation for testing data

model_evaluation_testing(gscv_rf_clf,x_test,y_test)

# evaluation for training data

model_evaluation_training(gscv_rf_clf,x_train,y_train)
```

In [87]:

```
rscv_rf_clf = rf_rscv_best_estimator(x_train,y_train)
rscv_rf_clf
```

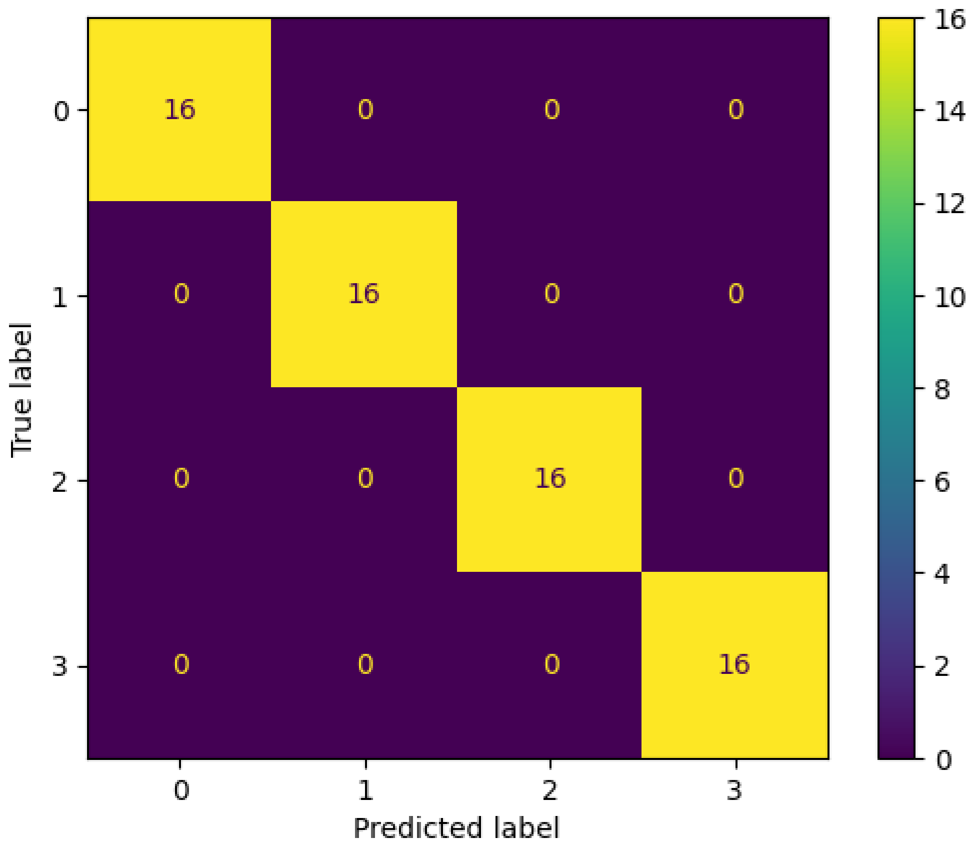
Out[87]:

```
RandomForestClassifier(criterion='entropy', max_depth=4, max_features='log2',
                       min_samples_leaf=10, min_samples_split=11)
```

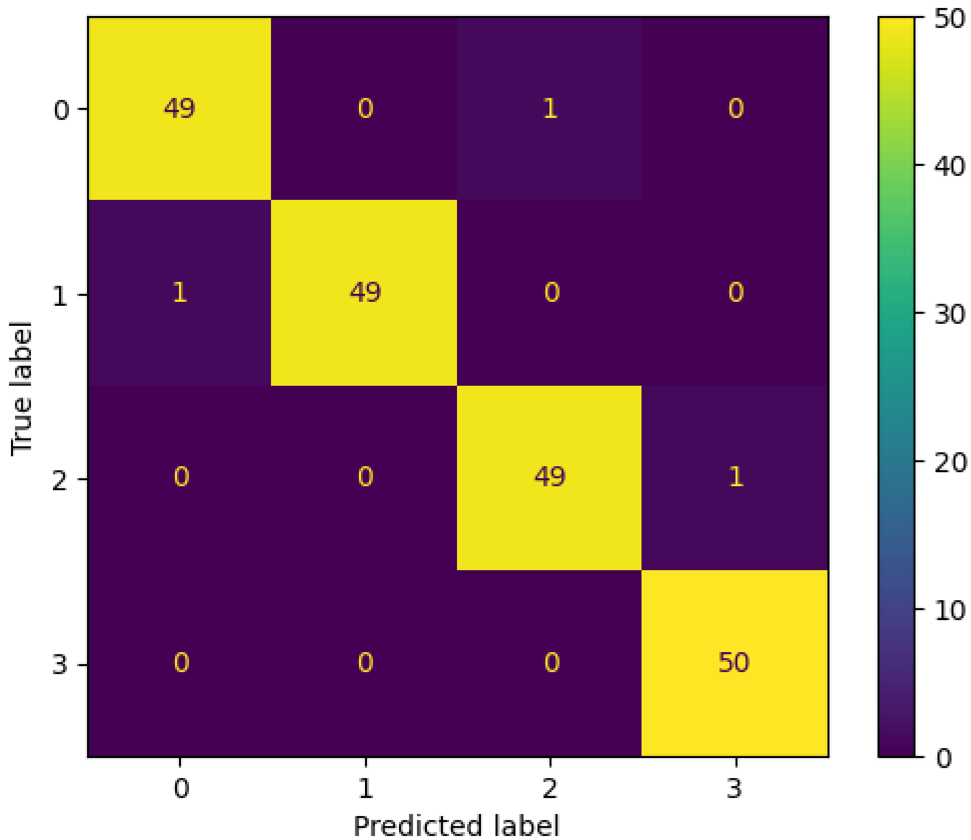
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [88]:
```
# evaluation for testing data

model_evaluation_testing(rscv_rf_clf,x_test,y_test)

# evaluation for training data

model_evaluation_training(rscv_rf_clf,x_train,y_train)
```

```
************************************************** Testing Data *************************************************
The confusion Matrix is :
 [[16  0  0  0]
 [ 0 16  0  0]
 [ 0  0 16  0]
 [ 0  0  0 16]]
```



```
==================================================
The Accuracy is :  1.0
==================================================
The Classification report is :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      1.00      1.00        16
           2       1.00      1.00      1.00        16
           3       1.00      1.00      1.00        16

    accuracy                           1.00        64
   macro avg       1.00      1.00      1.00        64
weighted avg       1.00      1.00      1.00        64

************************************************** Training Data ************
**************************************
The confusion matrix is :
 [[49  0  1  0]
 [ 1 49  0  0]
 [ 0  0 49  1]
 [ 0  0  0 50]]
```



```
==================================================
The accuracy is :  0.985
==================================================
The classification Report is :
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        50
           1       1.00      0.98      0.99        50
           2       0.98      0.98      0.98        50
           3       0.98      1.00      0.99        50

    accuracy                           0.98       200
   macro avg       0.99      0.98      0.98       200
weighted avg       0.99      0.98      0.98       200
```

```
RandomForestClassifier(criterion='entropy', max_depth=4, max_features='log2',
                       min_samples_leaf=10, min_samples_split=11)
```

In [89]:

```
result = rscv_dt_clf.cost_complexity_pruning_path(x_train, y_train)
ccp_alpha_list = result['ccp_alphas']
ccp_alpha_list
```

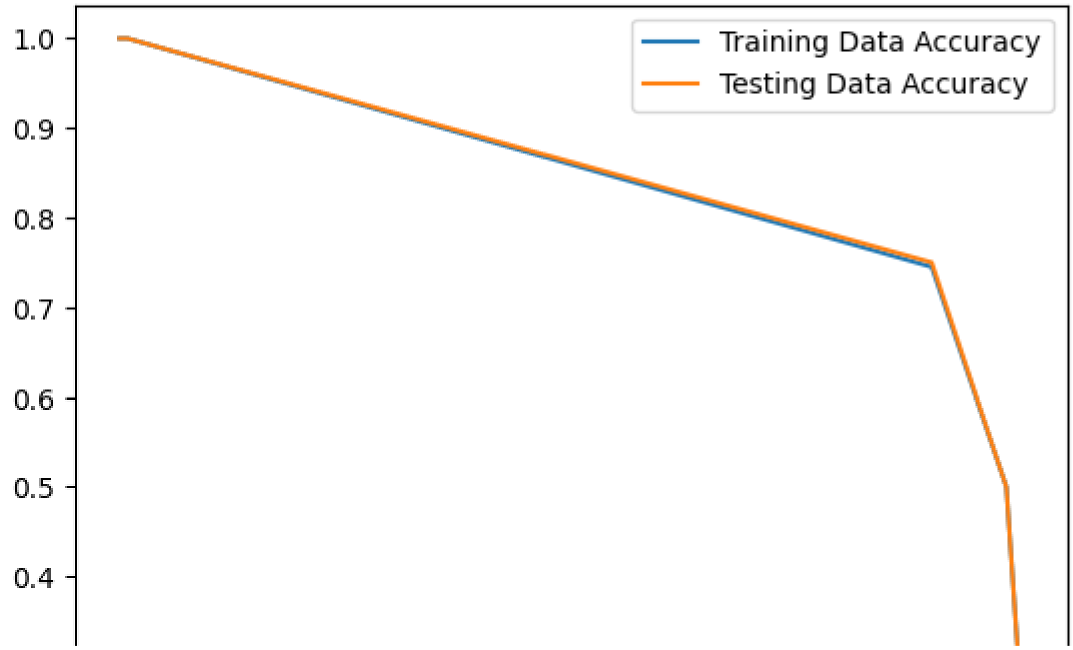
Out[89]:

```
array([0.        , 0.00051429, 0.00051821, 0.00218227, 0.21920552,
       0.23936955, 0.24355178])
```

In [90]:

```
train_accuracy_list = []
test_accuracy_list = []

for i in ccp_alpha_list:
    dt_clf_model = DecisionTreeClassifier(ccp_alpha=i, random_state=11)
    dt_clf_model.fit(x_train, y_train)

    training_accuracy = dt_clf_model.score(x_train, y_train)
    train_accuracy_list.append(training_accuracy)

    testing_accuracy = dt_clf_model.score(x_test, y_test)
    test_accuracy_list.append(testing_accuracy)
```

In [91]:
```
fig, ax = plt.subplots()
ax.plot(ccp_alpha_list, train_accuracy_list, label = "Training Data Accuracy")
ax.plot(ccp_alpha_list, test_accuracy_list, label = "Testing Data Accuracy")
ax.legend()
plt.show()
```



In [176]:
```
max_test = test_accuracy_list.index(max(test_accuracy_list))
max_test
best_ccp = ccp_alpha_list[max_test]
best_ccp
```

Out[176]:
```
0.0
```

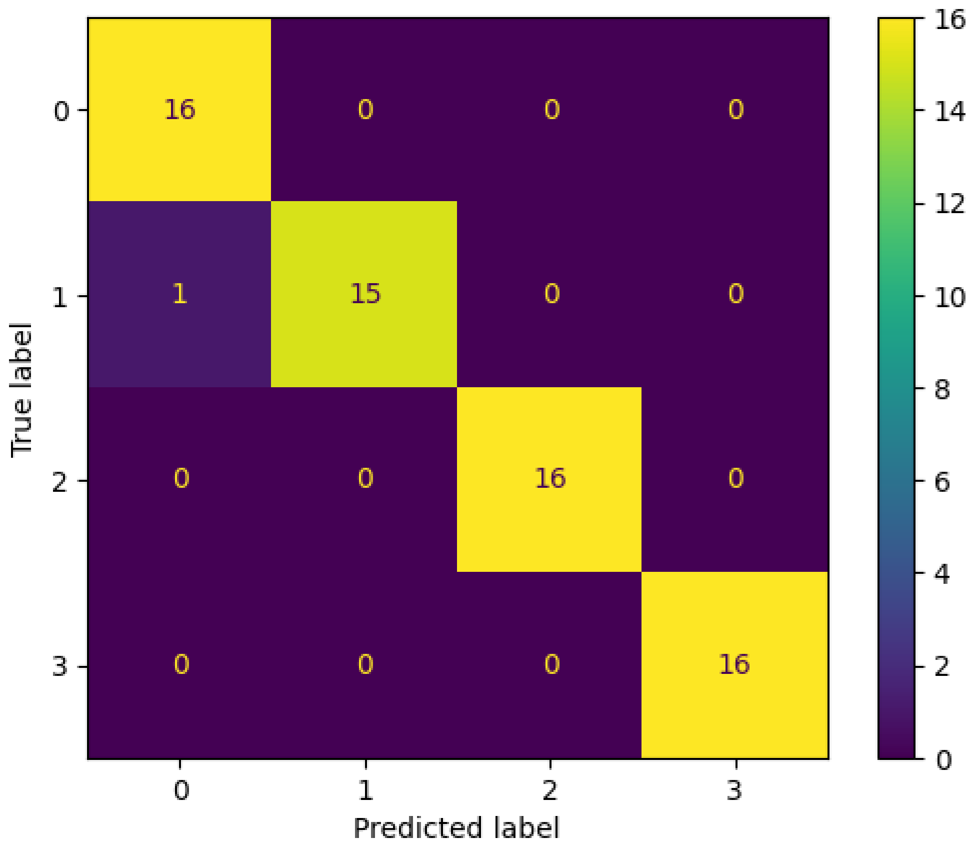
In [92]:
```
dt_clf_model_1 = DecisionTreeClassifier(criterion='entropy', max_depth=7, max_
                       min_samples_leaf=4, min_samples_split=4,ccp_alpha= 0.21

dt_clf_model_1.fit(x_train, y_train)
```

Out[92]:
```
DecisionTreeClassifier(ccp_alpha=0.21920552, criterion='entropy', max_depth=
7,
                       max_features='log2', min_samples_leaf=4,
                       min_samples_split=4, random_state=11)
```

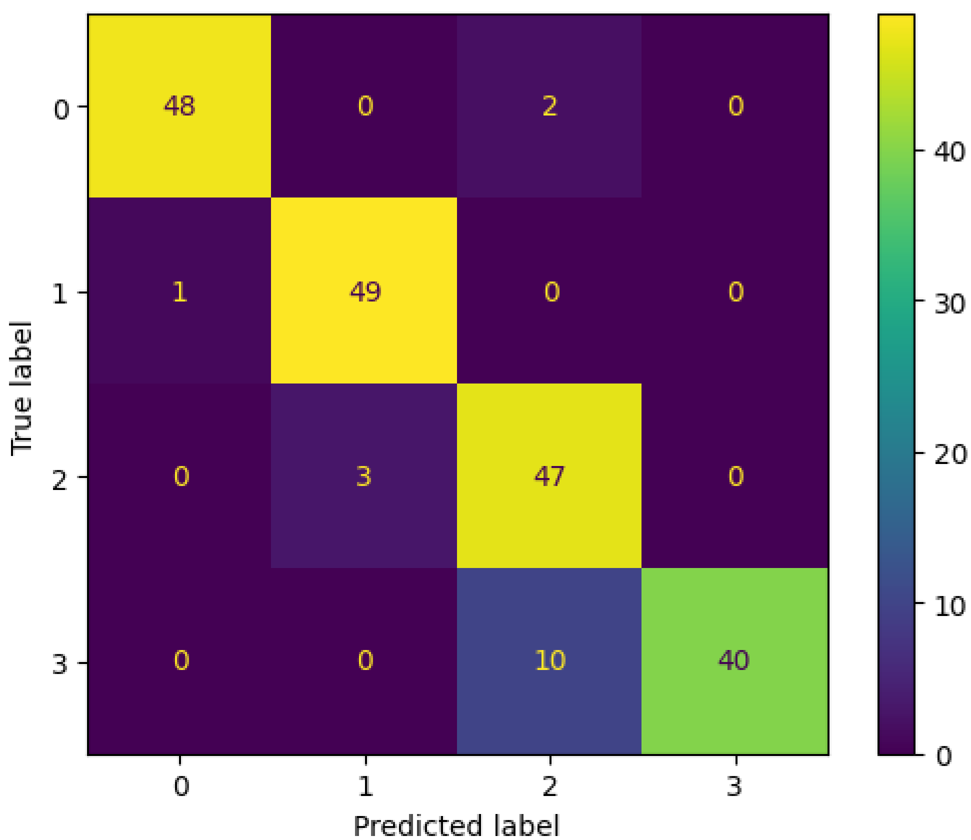
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [93]:
```
# evaluation for testing data

model_evaluation_testing(dt_clf_model_1,x_test,y_test)

# evaluation for training data

model_evaluation_training(dt_clf_model_1,x_train,y_train)
```

```
************************************************** Testing Data *************
*************************************
The confusion Matrix is :
 [[16  0  0  0]
 [ 1 15  0  0]
 [ 0  0 16  0]
 [ 0  0  0 16]]
```



```
==================================================
The Accuracy is :  0.984375
==================================================
The Classification report is :
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        16
           1       1.00      0.94      0.97        16
           2       1.00      1.00      1.00        16
           3       1.00      1.00      1.00        16

    accuracy                           0.98        64
   macro avg       0.99      0.98      0.98        64
weighted avg       0.99      0.98      0.98        64

************************************************** Training Data ************
**************************************
The confusion matrix is :
 [[48  0  2  0]
 [ 1 49  0  0]
 [ 0  3 47  0]
 [ 0  0 10 40]]
```



```
==================================================
The accuracy is :  0.92
==================================================
The classification Report is :
              precision    recall  f1-score   support

           0       0.98      0.96      0.97        50
           1       0.94      0.98      0.96        50
           2       0.80      0.94      0.86        50
           3       1.00      0.80      0.89        50

    accuracy                           0.92       200
   macro avg       0.93      0.92      0.92       200
weighted avg       0.93      0.92      0.92       200
```

Adaboost

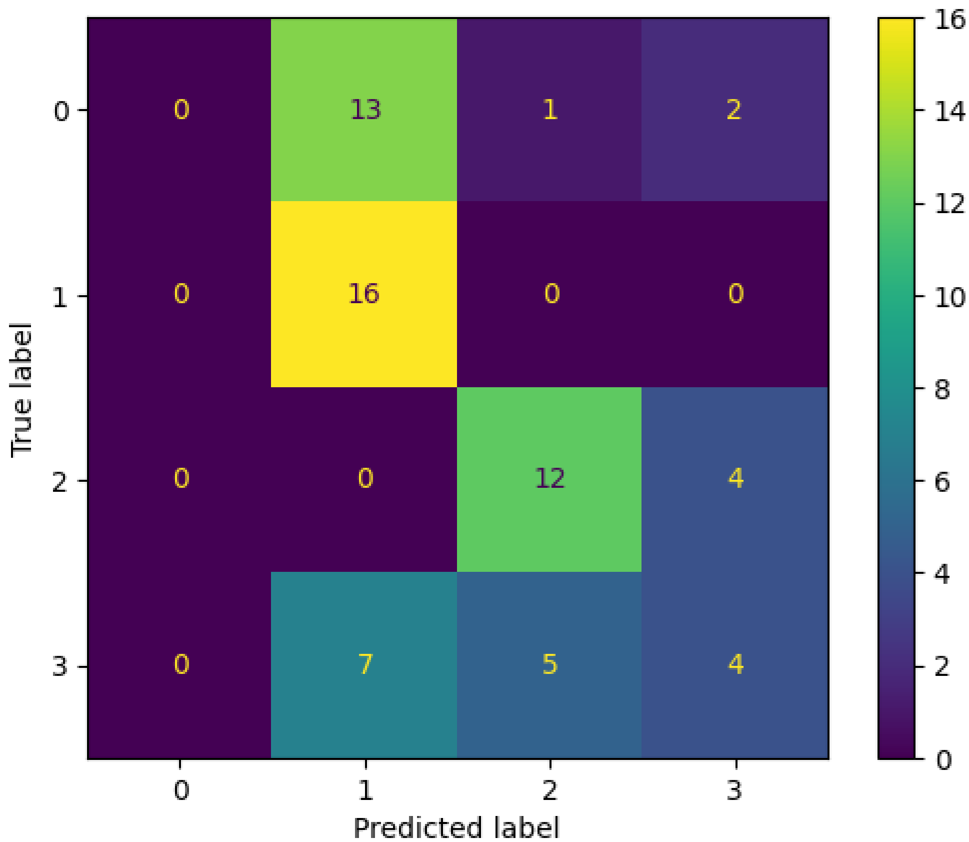
In [94]:
```
model_name = 'Adaptive Boosting Classifier'
adb_clf = adaboost_clf_training(x_train,y_train)
adb_clf
```

Out[94]:
```
AdaBoostClassifier(estimator=LogisticRegression(), random_state=39)
```

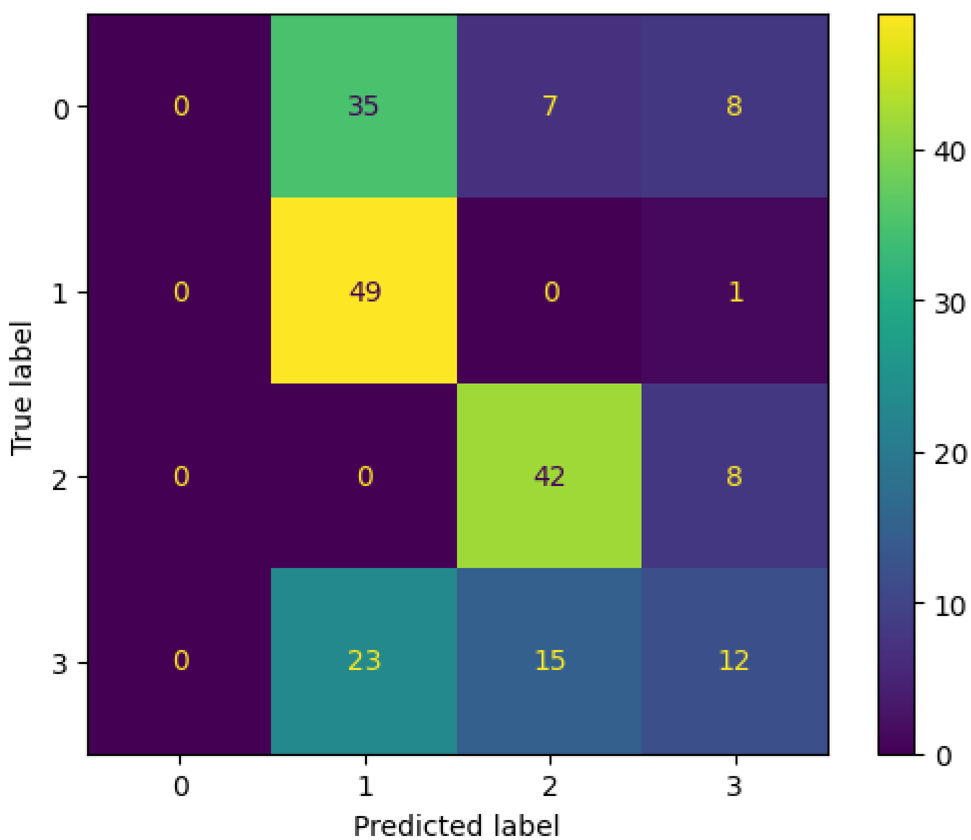
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [95]:
```
# evaluation for testing data

model_evaluation_testing(adb_clf,x_test,y_test)

# evaluation for training data

model_evaluation_training(adb_clf,x_train,y_train)
```

```
************************************************ Testing Data *************
************************************
The confusion Matrix is :
 [[ 0 13  1  2]
 [ 0 16  0  0]
 [ 0  0 12  4]
 [ 0  7  5  4]]
```



```
==================================================
The Accuracy is :  0.5
==================================================
The Classification report is :
               precision    recall  f1-score   support

           0       0.00      0.00      0.00        16
           1       0.44      1.00      0.62        16
           2       0.67      0.75      0.71        16
           3       0.40      0.25      0.31        16

    accuracy                           0.50        64
   macro avg       0.38      0.50      0.41        64
weighted avg       0.38      0.50      0.41        64

************************************************** Training Data ************
**************************************
The confusion matrix is :
 [[ 0 35  7  8]
 [ 0 49  0  1]
 [ 0  0 42  8]
 [ 0 23 15 12]]
```



```
==================================================
The accuracy is :  0.515
==================================================
The classification Report is :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        50
           1       0.46      0.98      0.62        50
           2       0.66      0.84      0.74        50
           3       0.41      0.24      0.30        50

    accuracy                           0.52       200
   macro avg       0.38      0.51      0.42       200
weighted avg       0.38      0.52      0.42       200
```

Hyper-Parameter Tunning

```
gscv_adb_clf = adb_gscv_best_estimator(x_train,y_train)
gscv_adb_clf
```

```
# evaluation for testing data

model_evaluation_testing(gscv_adb_clf,x_test,y_test)

# evaluation for training data

model_evaluation_training(gscv_adb_clf,x_train,y_train)
```

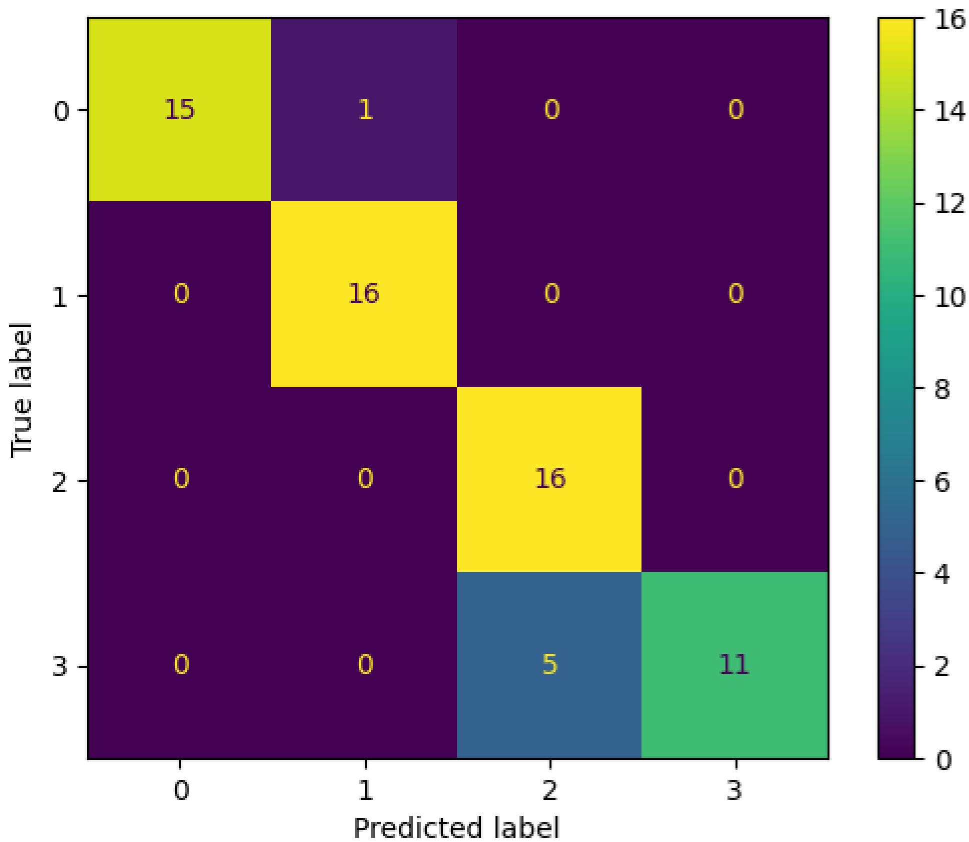
In [102]:
```
rscv_adb_clf = adb_rscv_best_estimator(x_train,y_train)
rscv_adb_clf
```

Out[102]:
```
AdaBoostClassifier(learning_rate=0.864, n_estimators=85)
```

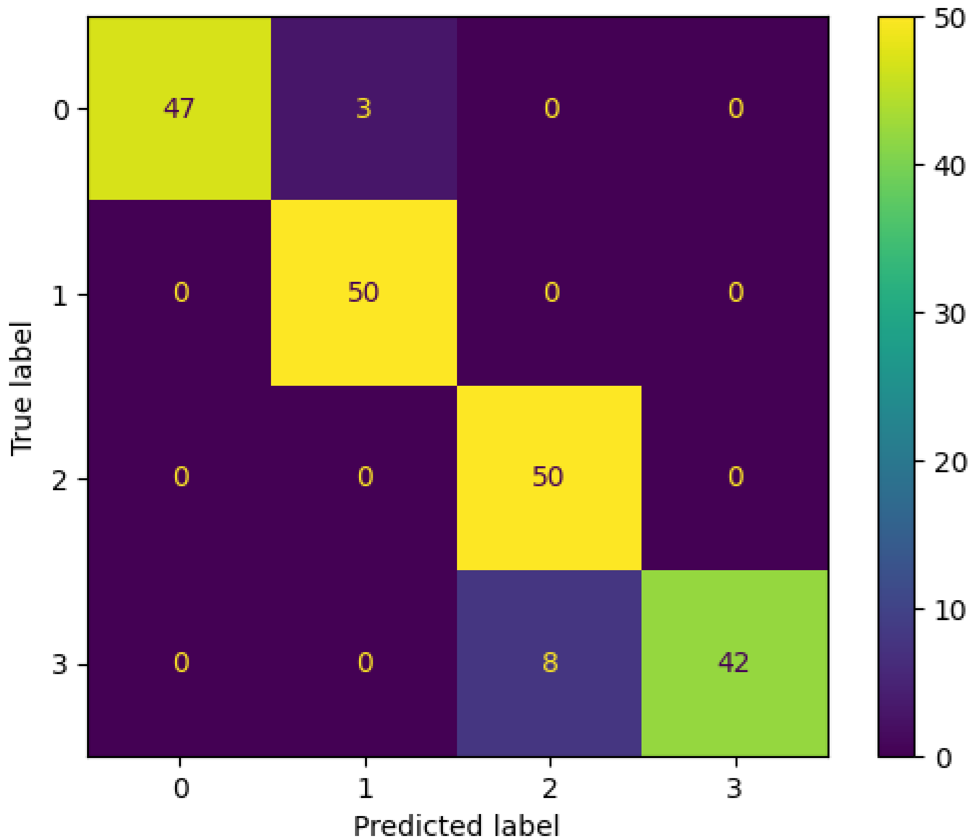
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [103]:
```
# evaluation for testing data

model_evaluation_testing(rscv_adb_clf,x_test,y_test)

# evaluation for training data

model_evaluation_training(rscv_adb_clf,x_train,y_train)
```

```
************************************************* Testing Data *************************************************
The confusion Matrix is :
 [[15  1  0  0]
 [ 0 16  0  0]
 [ 0  0 16  0]
 [ 0  0  5 11]]
```

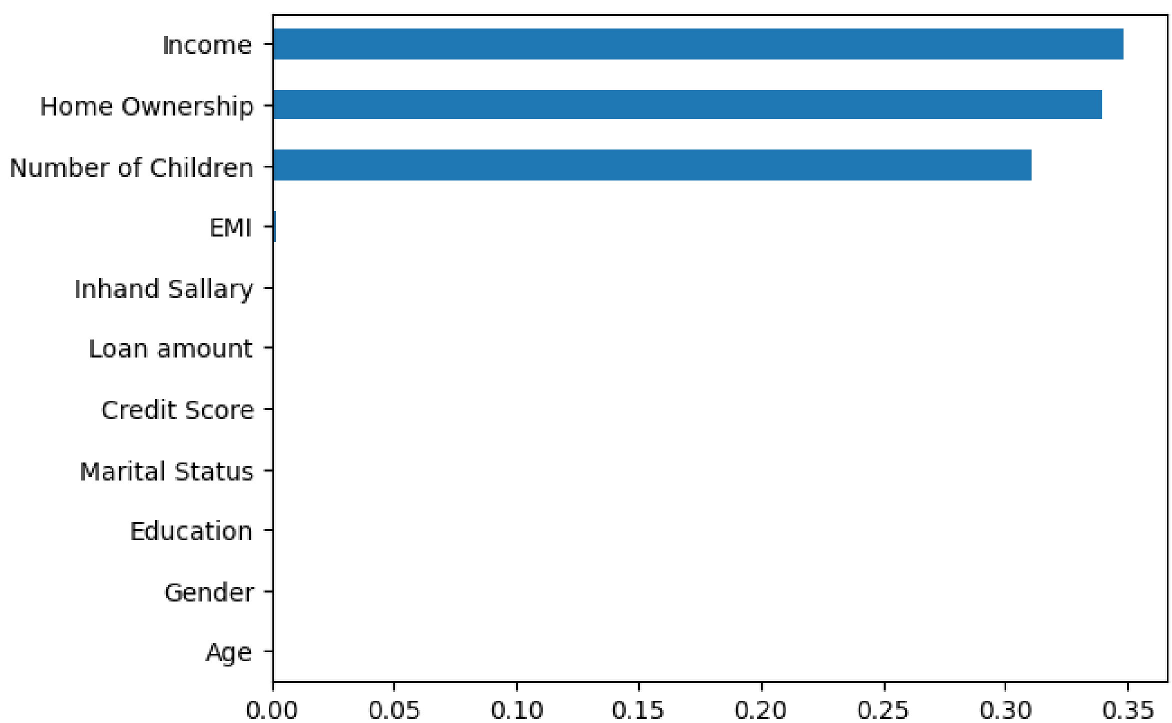


```
==================================================
The Accuracy is :  0.90625
==================================================
The Classification report is :
               precision    recall  f1-score   support

           0       1.00      0.94      0.97        16
           1       0.94      1.00      0.97        16
           2       0.76      1.00      0.86        16
           3       1.00      0.69      0.81        16

    accuracy                           0.91        64
   macro avg       0.93      0.91      0.90        64
weighted avg       0.93      0.91      0.90        64

************************************************** Training Data ************
**************************************
The confusion matrix is :
 [[47  3  0  0]
 [ 0 50  0  0]
 [ 0  0 50  0]
 [ 0  0  8 42]]
```



```
==================================================
The accuracy is :  0.945
==================================================
The classification Report is :
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        50
           1       0.94      1.00      0.97        50
           2       0.86      1.00      0.93        50
           3       1.00      0.84      0.91        50

    accuracy                           0.94       200
   macro avg       0.95      0.94      0.94       200
weighted avg       0.95      0.94      0.94       200
```

In [105]:
```
s2 = pd.Series(rscv_dt_clf.feature_importances_,index=x_train.columns)
s2.sort_values().plot(kind='barh')
plt.show()
```




```
comparison_df = pd.DataFrame({"Model Name":model_details,"Testing
Accuracy":testing_accuracy_list,
                              "Training Accuracy":training_accuracy_list})
comparison_df
```

```
Selected Model : Decision Tree after RandomisedSearchCV = rscv_dt_clf

Testing Accuracy = 96.87 %
Training Accuracy = 97.5 %
```

In [68]:
```
with open("credit_card_limit_prediction.pkl", "wb") as f:
    pickle.dump(rscv_dt_clf, f)
```

In [69]:
```
json_data = {"Gender":Gender_values,
             "Education":Education_values,
             "Marital Status":Marital_Status_values,
             "Home Ownership":Home_Owneship_values,
             "Credit Score":Credit_Score_values,
             "columns":list(x.columns)}
json_data
```

Out[69]:
```
{'Gender': {'Male': 0, 'Female': 1},
 'Education': {"Bachelor's Degree": 2,
  "Master's Degree": 3,
  'High School Diploma': 0,
  'Doctorate': 4,
  "Associate's Degree": 1},
 'Marital Status': {'Married': 1, 'Single': 0},
 'Home Ownership': {'Owned': 1, 'Rented': 0},
 'Credit Score': {'High': 2, 'Average': 1, 'Low': 0},
 'columns': ['Age',
  'Gender',
  'Income',
  'Education',
  'Marital Status',
  'Number of Children',
  'Home Ownership',
  'Credit Score',
  'Loan amount',
  'EMI',
  'Inhand Sallary']}
```

In [70]:
```
with open("project_data.json",'w') as f:
    json.dump(json_data,f)
```

In [71]:
```
normal_scaler = MinMaxScaler()
normal_scaler.fit_transform(df22)

with open("normal_scaler.pkl",'wb') as f:
    pickle.dump(normal_scaler,f)
```

Single User Input

In [72]:
```
Age=25.00
Gender='Female'
Income=50000.00
Education="Bachelor's Degree"
Marital_Status= 'Single'
Number_of_Children=0.00
Home_Ownership='Rented'
Credit_Score='High'
Loan_amount=6000000.00
EMI=16666.67
Inhand_Sallary=33333.00
```

In [73]:
```
test_array = np.zeros(x.shape[1],dtype=int)

test_array[0] = Age
test_array[1] = json_data["Gender"][Gender]
test_array[2] = Income
test_array[3] = json_data["Education"][Education]
test_array[4] = json_data["Marital Status"][Marital_Status]
test_array[5] = Number_of_Children
test_array[6] = json_data["Home Ownership"][Home_Ownership]
test_array[7] = json_data["Credit Score"][Credit_Score]
test_array[8] = Loan_amount
test_array[9] = EMI
test_array[10] = Inhand_Sallary


test_array
```

Out[73]:
```
array([     25,       1,   50000,       2,       0,       0,       0,
          2, 6000000,   16666,   33333])
```

In [74]:
```
test_df = pd.DataFrame([test_array],columns=x_train.columns)
test_df_2 = test_df[['Age','Income','Number of Children','Loan amount','EMI','I
scaled_df = normal_scaler.transform(test_df_2)
test_df[['Age','Income','Number of Children','Loan amount','EMI','Inhand Salla
```

In [75]:
```
limit = round(rscv_dt_clf.predict(test_df)[0],2)

if limit == 0:
    print("Credit Card is Approved, Predicted Limit is 3 to 4.5 Lakhs")
elif limit == 1:
    print("Credit Card is Approved, Predicted Limit is 1.5 to 3 Lakhs")
elif limit == 3:
    print("Credit Card is Approved, Predicted Limit is 1 to 1.5 Lakhs")
else:
    print("Sorry, Your request for Credit Card is Declined")
```

```
Credit Card is Approved, Predicted Limit is 1 to 1.5 Lakhs
```

In [ ]: